Developer Guide

Amazon DocumentDB

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DocumentDB: Developer Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon DocumentDB 1
Overview 1
Clusters 3
Instances 4
Regions and AZs 7
Regions 7
Availability zones 7
Pricing 9
Free trial 10
Monitoring 10
Interfaces 10
AWS Management Console 10
AWS CLI 10
MongoDB drivers 11
What's next? 11
How it works 11
Amazon DocumentDB endpoints 13
TLS support 17
Amazon DocumentDB storage 17
Amazon DocumentDB replication 18
Amazon DocumentDB reliability 18
Read preference options 19
TTL deletes 24
Billable resources 24
What is a document database? 27
Use cases 28
Understanding documents 29
Working with documents 35
Get started guide 47
Prerequisites 47
Step 1: Create a cluster 49
Step 2: Connect to your cluster 52
Step 3: Insert and query data 53
Step 4: Explore 55

Quick start using AWS CloudFormation ... 56
Prerequisites ... 56
Required IAM Permissions ... 57
Amazon EC2 Key Pair ... 60
Launching an Amazon DocumentDB AWS CloudFormation stack ... 60
Accessing the Amazon DocumentDB cluster ... 65
Termination protection and deletion protection ... 65
MongoDB compatibility ... 67
MongoDB 5.0 compatibility ... 67
What's new in Amazon DocumentDB 5.0 ... 67
Get started with Amazon DocumentDB 5.0 ... 68
Upgrade or migrate to Amazon DocumentDB 5.0 ... 69
Functional differences ... 69
MongoDB 4.0 compatibility ... 70
Amazon DocumentDB 4.0 features ... 71
Get started with Amazon DocumentDB 4.0 ... 72
Upgrade or migrate to Amazon DocumentDB 4.0 ... 72
Functional differences ... 73
Transactions ... 74
Requirements ... 74
Best practices ... 75
Limitations ... 75
Monitoring and diagnostics ... 76
Transaction isolation level ... 77
Use cases ... 77
Multi-Statement Transactions ... 77
Multi-collection transactions ... 79
Transaction API examples for callback API ... 81
Transaction API examples for core API ... 81
Supported commands ... 115
Unsupported capabilities ... 116
Sessions ... 116
Causal consistency ... 117
Retryable writes ... 118
Transaction errors ... 118
Best practices ... 119

Basic operational guidelines ... 119
Instance sizing ... 120
Working with indexes ... 122
Building Indexes ... 122
Index selectivity ... 122
Impact of indexes on writing data ... 123
Identifying missing indexes ... 123
Identifying unused indexes ... 123
Security best practices ... 123
Cost optimization ... 124
Using metrics to identify performance issues ... 125
Viewing performance metrics ... 125
Setting a CloudWatch alarm ... 126
Evaluating performance metrics ... 126
Evaluating Amazon DocumentDB instance usage with CloudWatch metrics ... 127
Tuning queries ... 129
TTL and time series workloads ... 129
Migrations ... 130
Working with cluster parameter groups ... 130
Aggregation pipeline queries ... 130
`batchInsert` and `batchUpdate` ... 131
Functional differences with MongoDB ... 132
Functional benefits of Amazon DocumentDB ... 132
Implicit transactions ... 132
Updated functional differences ... 133
Array indexing ... 134
Multi-key indexes ... 135
Null characters in strings ... 136
Role-based access control ... 136
`$regex` indexing ... 136
Projection for nested documents ... 137
Functional differences with MongoDB ... 137
`$vectorSearch` operator ... 138
`OpCountersCommand` ... 138
Admin databases and collections ... 138
`cursormaxTimeMS` ... 138

explain() ... 138
Index builds ... 139
Lookup with empty key in path ... 139
MongoDB APIs, operations, and data types ... 139
`mongodump` and `mongorestore` utilities ... 139
Result ordering ... 140
Retryable writes ... 140
Sparse index ... 141
Using `$elemMatch` within an `$all` expression ... 141
`$ne`, `$nin`, `$nor`, `$not`, `$exists`, and `$elemMatch` indexing ... 142
Dollar($) and dot(.) in field names ... 142
`$lookup` ... 142
`$natural` and reverse sorting ... 146
Supported MongoDB APIs, operations, and data types ... 147
Database commands ... 147
Administrative Commands ... 148
Aggregation ... 149
Authentication ... 150
Diagnostic commands ... 150
Query and write operations ... 151
Role management commands ... 152
Sessions commands ... 153
User management ... 153
Sharding commands ... 154
Query and projection operators ... 156
Array Operators ... 156
Bitwise operators ... 156
Comment operator ... 157
Comparison operators ... 157
Element operators ... 157
Evaluation query operators ... 158
Logical operators ... 158
Projection operators ... 158
Update operators ... 159
Array operators ... 159
Bitwise operators ... 160

Field operators ... 160
Update modifiers ... 160
Geospatial ... 161
Geometry specifiers ... 161
Query selectors ... 161
Cursor methods ... 162
Aggregation pipeline operators ... 164
Accumulator expressions ... 165
Arithmetic operators ... 166
Array operators ... 167
Boolean operators ... 168
Comparison operators ... 168
Conditional expression operators ... 169
Data type operator ... 169
Data size operator ... 169
Date operators ... 169
Literal operator ... 171
Merge operator ... 171
Natural operator ... 171
Set operators ... 171
Stage operators ... 172
String operators ... 174
System variables ... 175
Text search operator ... 175
Type conversion operators ... 175
Variable operators ... 176
Miscellaneous operators ... 176
Data types ... 177
Indexes and index properties ... 178
Indexes ... 178
Index properties ... 179
Generative AI ... 180
SageMaker Canvas ... 180
How to build no-code ML models with SageMaker AI Canvas ... 180
Configuring the SageMaker AI domain and user profile ... 181
Configuring IAM access permissions for Amazon DocumentDB and SageMaker AI Canvas . 181

Creating database users and roles for SageMaker AI Canvas ... 181
Available regions ... 182
Vector Search ... 183
Inserting vectors ... 183
Creating a vector index ... 184
Getting an index definition ... 188
Querying vectors ... 189
Features and limitations ... 193
Best practices ... 194
Migrating to Amazon DocumentDB ... 196
Migration tools ... 196
AWS Database Migration Service ... 196
Command line utilities ... 197
Discovery ... 197
Planning: Amazon DocumentDB cluster requirements ... 201
Migration approaches ... 204
Offline ... 204
Online ... 205
Hybrid ... 207
Migration sources ... 209
Migration connectivity ... 209
Testing ... 212
Migration plan testing considerations ... 213
Performance testing ... 216
Failover testing ... 216
Additional resources ... 217
Migration playbook ... 217
Migration process ... 217
Additional resources ... 222
Upgrading the Amazon DocumentDB engine version ... 223
MVU prerequisites and limitations ... 223
Best practices for in-place major version upgrades ... 225
Test in-place major version upgrades using cloned clusters ... 225
Before an in-place major version upgrade ... 226
During an in-place major version upgrade ... 227
After an in-place major version upgrade ... 229

Performing an in-place major version upgrade ... 230
Differences between Amazon DocumentDB 3.6/4.0 to 5.0 upgraded clusters and new Amazon DocumentDB 5.0 clusters ... 233
Troubleshooting an in-place major version upgrade ... 233
Upgrading using AWS DMS ... 235
Step 1: Enable change streams ... 235
Step 2: Modify the change streams retention duration ... 236
Step 3: Migrate your indexes ... 236
Step 4: Create an AWS DMS replication instance ... 237
Step 5: Create an AWS DMS source endpoint ... 240
Step 6: Create an AWS DMS target endpoint ... 242
Step 7: Create and run a migration task ... 244
Step 8: Changing the application endpoint to the target Amazon DocumentDB cluster ... 246
Security ... 247
Password management with AWS Secrets Manager ... 248
Limitations for Secrets Manager integration with Amazon DocumentDB ... 248
Overview of managing primary user passwords with AWS Secrets Manager ... 249
Enforcing Amazon DocumentDB management of the primary user password in AWS Secrets Manager ... 249
Managing the primary user password for a cluster with Secrets Manager ... 250
Data protection ... 250
Client-side field level encryption ... 251
Encrypting data at rest ... 259
Encrypting data in transit ... 264
Key management ... 275
Identity and Access Management ... 276
Audience ... 276
Authenticating with identities ... 277
Managing access using policies ... 280
How Amazon DocumentDB works with IAM ... 283
Identity-based policy examples ... 290
Troubleshooting ... 293
Managing access permissions to your Amazon DocumentDB resources ... 295
Using identity-based policies (IAM policies) ... 300
AWS managed policies for Amazon DocumentDB ... 304
Amazon DocumentDB API permissions reference ... 322

Authentication using IAM identity ... 331
Getting started with IAM users and roles ... 331
Configuring compute types ... 335
Monitoring authentication requests ... 335
Using IAM authentication ... 336
Drivers supporting IAM ... 336
IAM identity authentication FAQ ... 336
Managing Amazon DocumentDB users ... 337
Primary and `serviceadmin` user ... 337
Creating additional users ... 338
Automatically rotating passwords ... 340
Role-Based Access Control ... 341
RBAC concepts ... 342
Getting started with RBAC built-in roles ... 344
Getting started with RBAC user-defined roles ... 347
Connecting to Amazon DocumentDB as a User ... 351
Common commands ... 353
Functional differences ... 358
Limits ... 358
Database access using Role-Based Access Control ... 359
Logging and monitoring ... 368
Updating certificates ... 369
Updating your application and Amazon DocumentDB cluster ... 369
Automatic server certificate rotation ... 373
Troubleshooting ... 374
Frequently Asked Questions ... 375
Updating certificates — GovCloud ... 381
Updating your application and Amazon DocumentDB cluster ... 369
Troubleshooting ... 374
Frequently Asked Questions ... 375
Compliance validation ... 394
Resilience ... 395
Infrastructure security ... 396
VPC endpoints (AWS PrivateLink) ... 396
Considerations for VPC endpoints ... 397
Region availability ... 398

Creating an interface VPC endpoint for Amazon DocumentDB API ... 399
Creating a VPC endpoint policy for Amazon DocumentDB API ... 399
Security best practices ... 400
Auditing events ... 401
Supported events ... 402
Enable auditing ... 407
Disable auditing ... 414
Access your events ... 416
Filter DML events ... 417
Backing up and restoring ... 423
Back up and restore: concepts ... 424
Understanding backup storage usage ... 426
Dumping, restoring, importing, and exporting data ... 427
`mongodump` ... 428
`mongorestore` ... 429
`mongoexport` ... 429
`mongoimport` ... 430
Tutorial ... 430
Cluster snapshot considerations ... 433
Backup storage ... 434
Backup window ... 434
Backup retention period ... 436
Copy cluster snapshot encryption ... 436
Comparing automatic and manual snapshots ... 437
Creating a manual cluster snapshot ... 438
Copying a cluster snapshot ... 442
Copying shared snapshots ... 442
Copying snapshots across AWS Regions ... 443
Limitations ... 443
Handling encryption ... 443
Parameter group considerations ... 444
Copying a cluster snapshot ... 444
Sharing a cluster snapshot ... 451
Sharing an encrypted snapshot ... 452
Sharing a snapshot ... 455
Restoring from a cluster snapshot ... 458

Restoring to a point in time ... 466
Deleting a cluster snapshot ... 472
Managing Amazon DocumentDB ... 474
Operational tasks overview ... 474
Adding a replica to an Amazon DocumentDB cluster ... 475
Describing clusters and instances ... 476
Creating a cluster snapshot ... 478
Restoring from a snapshot ... 479
Removing an instance from a cluster ... 480
Deleting a cluster ... 480
Global clusters ... 481
What is a global cluster? ... 481
How are global clusters useful? ... 481
What are the current limitations of global clusters? ... 482
Quick start guide ... 483
Managing global clusters ... 499
Connecting global clusters ... 507
Monitoring global clusters ... 507
Disaster recovery ... 508
Managing clusters ... 524
Understanding clusters ... 524
Cluster settings ... 527
Cluster storage configurations ... 530
Determining a cluster's status ... 533
Cluster lifecycle ... 534
Scaling clusters ... 576
Cloning a volume for a cluster ... 579
Understanding cluster fault tolerance ... 592
Managing instances ... 594
Determining an instance's status ... 594
Instance lifecycle ... 594
Managing instance classes ... 618
NVMe-backed instances ... 629
Managing subnet groups ... 632
Creating a subnet group ... 633
Describing a subnet group ... 638

Modifying a subnet group ... 641
Deleting a subnet group ... 644
High availability and replication ... 645
Read scaling ... 646
High availability ... 646
Adding replicas ... 647
Failover ... 648
Replication lag ... 652
Managing indexes ... 654
Amazon DocumentDB index creation ... 654
Maintaining Amazon DocumentDB indexes ... 660
Managing document compression ... 661
Managing document compression ... 661
Monitoring document compression ... 663
Managing events ... 664
Viewing event categories ... 665
Viewing Amazon DocumentDB events ... 667
Choosing regions and availability zones ... 670
Region availability ... 671
Managing cluster parameter groups ... 673
Describing cluster parameter groups ... 673
Creating cluster parameter groups ... 680
Modifying cluster parameter groups ... 683
Modifying clusters to use customized cluster parameter groups ... 687
Copying cluster parameter groups ... 689
Resetting cluster parameter groups ... 691
Deleting cluster parameter groups ... 694
Cluster parameters reference ... 697
Understanding endpoints ... 714
Finding a cluster's endpoints ... 715
Finding an instance's endpoint ... 717
Connecting to endpoints ... 721
Understanding Amazon DocumentDB ARNs ... 722
Constructing an ARN ... 722
Finding an ARN ... 726
Tagging resources ... 728

Overview of resource tags ... 728
Tag constraints ... 729
Adding or updating tags ... 730
Listing tags ... 731
Removing tags ... 733
Maintaining Amazon DocumentDB ... 734
Engine patch notifications ... 735
View pending maintenance ... 736
Engine updates ... 738
User-initiated updates ... 741
Manage maintenance windows ... 743
Operating system updates ... 745
Understanding service-linked roles ... 748
Service-linked role permissions ... 749
Creating a service-linked role ... 750
Modifying a service-linked role ... 751
Deleting a service-linked role ... 751
Supported regions for Amazon DocumentDB service-linked roles ... 752
Using change streams ... 752
Supported operations ... 753
Billing ... 753
Limitations ... 754
Enabling change streams ... 755
Using change streams with Python ... 756
Full document lookup ... 759
Resuming a change stream ... 760
Resuming with `startAtOperationTime` ... 761
Resuming with `postBatchResumeToken` ... 763
Transactions ... 765
Modifying log retention duration ... 765
Change streams on secondary instances ... 768
Using AWS Lambda with change streams ... 769
Limitations ... 770
Using elastic clusters ... 771
Elastic cluster use cases ... 771
User profiles ... 772

Content management and historical records ... 772
Advantages of elastic clusters ... 772
AWS service integration ... 772
Region and version availability ... 773
Region availability for elastic clusters ... 773
Version availability ... 774
Limitations ... 774
Elastic cluster management ... 774
Query and write operations ... 774
Collection and index management ... 775
Administration and diagnostics ... 775
Opt-in features ... 775
How it works ... 776
Amazon DocumentDB elastic cluster sharding ... 776
Elastic cluster migration ... 780
Elastic cluster scaling ... 780
Elastic cluster reliability ... 780
Elastic cluster storage and availability ... 780
Functional differences between Amazon DocumentDB 4.0 and elastic clusters ... 781
Get started ... 782
Prerequisites ... 783
Step 1: Create an elastic cluster ... 784
Step 2: Connect to your elastic cluster ... 790
Step 3: Shard your collection, insert and query data ... 791
Step 4: Explore ... 795
Best practices ... 795
Choosing shard keys ... 795
Connection management ... 796
Unsharded collections ... 796
Scaling elastic clusters ... 796
Monitoring elastic clusters ... 797
Managing elastic clusters ... 797
Modifying elastic cluster configurations ... 798
Monitoring an elastic cluster ... 801
Deleting an elastic cluster ... 805
Managing elastic cluster snapshots ... 807

Stopping and starting an elastic cluster ... 821
Maintaining elastic clusters ... 826
Data encryption at rest ... 834
How Amazon DocumentDB elastic clusters use grants in AWS KMS ... 835
Create a customer managed key ... 836
Monitoring your encryption keys for Amazon DocumentDB elastic clusters ... 837
Learn more ... 843
Service-linked roles ... 843
Service-linked role permissions for elastic clusters ... 844
Monitoring Amazon DocumentDB ... 848
Monitoring a cluster's status ... 849
Cluster status values ... 850
Monitoring a cluster's status ... 851
Monitoring an instance's status ... 853
Instance status values ... 854
Monitoring instance status using the AWS Management Console or AWS CLI ... 856
Instance health status values ... 858
Monitoring instance health status using the AWS Management Console ... 858
Viewing Amazon DocumentDB recommendations ... 860
Event subscriptions ... 863
Subscribing to events ... 864
Manage subscriptions ... 867
Categories and messages ... 871
Monitoring Amazon DocumentDB with CloudWatch ... 874
Amazon DocumentDB metrics ... 874
Viewing CloudWatch data ... 890
Amazon DocumentDB dimensions ... 896
Monitoring Opcounter metrics ... 896
Monitoring database connections ... 896
Logging Amazon DocumentDB API calls with CloudTrail ... 897
Amazon DocumentDB Information in CloudTrail ... 897
Profiling operations ... 898
Supported operations ... 899
Limitations ... 899
Enabling the profiler ... 900
Disabling the profiler ... 904

Disabling profiler logs export ... 905
Accessing your profiler logs ... 908
Common queries ... 908
Monitoring with Performance Insights ... 909
Performance Insights concepts ... 910
Enabling and disabling Performance Insights ... 913
Configuring access policies for Performance Insights ... 916
Analyzing metrics with the Performance Insights dashboard ... 921
Retrieving metrics with the Performance Insights API ... 939
Amazon CloudWatch metrics for Performance Insights ... 953
Performance Insights for counter metrics ... 956
OpenSearch integration ... 958
Amazon OpenSearch Service as a destination ... 958
Step 1: Create an Amazon OpenSearch Service domain or OpenSearch serverless collection ... 959
Step 2: Enable change streams on the Amazon DocumentDB cluster ... 959
Step 3: Set up the pipeline role with permissions to write to the Amazon S3 bucket and destination domain or collection ... 959
Step 4: Add the permissions required on the pipeline role to create X-ENI ... 960
Step 5: Create the pipeline ... 961
Limitations ... 961
Developing with Amazon DocumentDB ... 963
Connect to DocumentDB ... 963
Connecting programmatically ... 963
Connecting from outside an Amazon VPC ... 991
Connecting as a replica set ... 993
Connect using Studio 3T ... 998
Connect using DataGrip ... 1009
Connect using Amazon EC2 ... 1017
Connect using the JDBC driver ... 1054
Connect using the ODBC driver ... 1071
Program with DocumentDB ... 1087
DocumentDB Java programming guide ... 1087
Using JSON schema validation ... 1118
Quotas and limits ... 1130
Supported instance types ... 1130

Supported regions 1132
Regional quotas 1133
Aggregation limits 1136
Cluster limits 1136
Instance limits 1138
Naming constraints 1140
TTL constraints 1141
Elastic cluster limits 1142
Elastic cluster shard limits 1143
Elastic cluster CPU, memory, connection, and cursor limits per shard 1143
Querying 1145
Querying documents 1145
Retrieving all documents 1146
Matching field values 1146
Embedded documents 1146
Field values in embedded documents 1147
Matching an array 1147
Matching values in an array 1147
Using operators 1148
Query plan 1148
Query Plan 1148
Query plan cache 1150
Explain results 1150
Scan and filter stage 1151
Index intersection 1152
Index union 1153
Multiple index intersection/union 1154
Compound index 1154
Sort stage 1155
Group stage 1155
Geospatial data 1155
Overview 1
Indexing and storing geospatial data 1156
Querying geospatial data 1158
Limitations 1161
Partial index 1161

Create a partial index ... 1162
Supported operators ... 1162
Query using a partial index ... 1162
Partial index functionalities ... 1163
Partial index limitations ... 1167
Text search ... 1169
Supported functionalities ... 1169
Using Amazon DocumentDB text index ... 1169
Differences with MongoDB ... 1175
Best practices and guidelines ... 1175
Limitations ... 1175
Troubleshooting ... 1177
Connection issues ... 1177
Cannot connect to an Amazon DocumentDB endpoint ... 1177
Testing a connection to an Amazon DocumentDB instance ... 1182
Connecting to an invalid endpoint ... 1183
Driver configuration impacting number of connections ... 1184
Index creation ... 1184
Index build fails ... 1184
Background index build latency issues and fails ... 1184
Performance and resource utilization ... 1185
View insert, update, and delete statistics ... 1186
Analyze cache performance ... 1187
Find and Terminate Long Running or Blocked Queries ... 1189
See a Query Plan and Optimize a Query ... 1190
How can I see a query plan in elastic clusters? ... 1192
List All Running Operations on an Instance ... 1195
Know When a Query Is Making Progress ... 1197
Determine Why a System Suddenly Runs Slowly ... 1200
Determine the Cause of High CPU Utilization ... 1202
Find the open cursors on an instance ... 1203
View the current Amazon DocumentDB engine version ... 1203
Analyze index usage and identify unused indexes ... 1203
Identify missed indexes ... 1205
Summary of useful queries ... 1207
Resource management API reference ... 1209

Actions ... 1209
Amazon DocumentDB (with MongoDB compatibility) ... 1212
Amazon DocumentDB Elastic Clusters ... 1401
Data Types ... 1473
Amazon DocumentDB (with MongoDB compatibility) ... 1475
Amazon DocumentDB Elastic Clusters ... 1553
Common Errors ... 1571
Common Parameters ... 1573
Release notes ... 1576
June 18, 2025 ... 1578
New feature ... 1578
May 8, 2025 ... 1579
New feature ... 1579
April 2, 2025 ... 1579
New feature ... 1579
Bug fixes and other changes ... 1579
March 24, 2025 ... 1580
New feature ... 1580
February 6, 2025 ... 1580
New feature ... 1580
January 28, 2025 ... 1580
New feature ... 1580
January 15, 2025 ... 1580
New features ... 1580
Bug fixes and other changes ... 1581
December 18, 2024 ... 1581
New features ... 1581
November 12, 2024 ... 1581
New features ... 1581
November 6, 2024 ... 1581
New features ... 1581
November 1, 2024 ... 1582
New feature ... 1582
October 22, 2024 ... 1582
New feature ... 1582
September 18, 2024 ... 1582

New feature ... 1582
September 17, 2024 ... 1583
New features ... 1583
Bug fixes and other changes ... 1583
August 22, 2024 ... 1583
New feature ... 1583
August 20, 2024 ... 1584
New feature ... 1584
August 8, 2024 ... 1584
New feature ... 1584
July 23, 2024 ... 1584
New features ... 1584
Bug fixes and other changes ... 1585
July 22, 2024 ... 1586
New features ... 1586
Bug fixes and other changes ... 1586
July 9, 2024 ... 1587
New feature ... 1587
July 8, 2024 ... 1587
New feature ... 1587
June 25, 2024 ... 1587
New feature ... 1587
May 29, 2024 ... 1587
New features ... 1587
April 3, 2024 ... 1588
New features ... 1588
Bug fixes and other changes ... 1588
February 22, 2024 ... 1588
New features ... 1588
January 30, 2024 ... 1589
New features ... 1589
January 10, 2024 ... 1589
New features ... 1589
Bug fixes and other changes ... 1591
December 20, 2023 ... 1591
Other changes ... 1591

December 13, 2023 1591
New features 1591
November 29, 2023 1591
New features 1591
November 21, 2023 1591
New features 1592
November 17, 2023 1592
New features 1592
Bug fixes and other changes 1592
November 6, 2023 1592
New features 1592
Bug fixes and other changes 1593
September 25, 2023 1593
New features 1593
September 20, 2023 1593
New features 1593
September 15, 2023 1593
New features 1593
September 11, 2023 1593
New features 1593
August 3, 2023 1594
New features 1594
July 13, 2023 1594
New features 1594
Bug fixes and other changes 1594
June 7, 2023 1595
Bug fixes and other changes 1595
May 10, 2023 1595
Bug fixes and other changes 1595
April 4, 2023 1595
Bug fixes and other changes 1595
March 22, 2023 1596
New features 1596
March 1, 2023 1596
New features 1596
February 27, 2023 1597

Bug fixes and other changes 1597
February 2, 2023 1597
Bug fixes and other changes 1597
November 30, 2022 1597
New features 1597
August 9, 2022 1598
New features 1598
Bug fixes and other changes 1598
July 25, 2022 1598
New features 1598
June 27, 2022 1599
New features 1599
April 29, 2022 1599
New features 1599
April 7, 2022 1599
New features 1599
March 16, 2022 1599
New features 1599
February 8, 2022 1600
New features 1600
January 24, 2022 1600
New features 1600
January 21, 2022 1600
New features 1600
October 25, 2021 1601
New features 1601
Bug fixes and other changes 1601
June 24, 2021 1602
New features 1602
May 4, 2021 1602
New features 1602
Bug fixes and other changes 1603
January 15, 2021 1603
New features 1603
November 9, 2020 1603
New features 1603

Bug fixes and other changes 1605
October 30, 2020 1606
New features 1606
Bug fixes and other changes 1606
September 22, 2020 1606
New features 1606
Bug fixes and other changes 1607
July 10, 2020 1607
New features 1607
Bug fixes and other changes 1607
June 30, 2020 1607
New features 1607
Bug fixes and other changes 1607
Document history 1609

What is Amazon DocumentDB (with MongoDB compatibility)

Amazon DocumentDB (with MongoDB compatibility) is a fast, reliable, and fully managed database service. Amazon DocumentDB makes it easy to set up, operate, and scale MongoDB-compatible databases in the cloud. With Amazon DocumentDB, you can run the same application code and use the same drivers and tools that you use with MongoDB.

Before using Amazon DocumentDB, you should review the concepts and features described in How it works. After that, complete the steps in Get started guide.

Topics

- Overview of Amazon DocumentDB
- Clusters
- Instances
- Regions and availability zones
- Amazon DocumentDB Pricing
- Monitoring
- Interfaces
- What's next?
- Amazon DocumentDB: how it works
- What is a document database?

Overview of Amazon DocumentDB

The following are some high-level features of Amazon DocumentDB:

- Amazon DocumentDB supports two types of clusters: instance-based clusters and elastic clusters. Elastic clusters support workloads with millions of reads/writes per second and petabytes of storage capacity. For more information about elastic clusters, see Using Amazon DocumentDB elastic clusters. The content below refers to Amazon DocumentDB instance-based clusters .

- Amazon DocumentDB automatically grows the size of your storage volume as your database storage needs grow. Your storage volume grows in increments of 10 GB, up to a maximum of 128 TiB. You don't need to provision any excess storage for your cluster to handle future growth.
- With Amazon DocumentDB, you can increase read throughput to support high-volume application requests by creating up to 15 replica instances. Amazon DocumentDB replicas share the same underlying storage, lowering costs and avoiding the need to perform writes at the replica nodes. This capability frees up more processing power to serve read requests and reduces the replica lag time—often down to single digit milliseconds. You can add replicas in minutes regardless of the storage volume size. Amazon DocumentDB also provides a reader endpoint, so the application can connect without having to track replicas as they are added and removed.
- Amazon DocumentDB lets you scale the compute and memory resources for each of your instances up or down. Compute scaling operations typically complete in a few minutes.
- Amazon DocumentDB runs in Amazon Virtual Private Cloud (Amazon VPC), so you can isolate your database in your own virtual network. You can also configure firewall settings to control network access to your cluster.
- Amazon DocumentDB continuously monitors the health of your cluster. On an instance failure, Amazon DocumentDB automatically restarts the instance and associated processes. Amazon DocumentDB doesn't require a crash recovery replay of database redo logs, which greatly reduces restart times. Amazon DocumentDB also isolates the database cache from the database process, enabling the cache to survive an instance restart.
- On instance failure, Amazon DocumentDB automates failover to one of up to 15 Amazon DocumentDB replicas that you create in other Availability Zones. If no replicas have been provisioned and a failure occurs, Amazon DocumentDB tries to create a new Amazon DocumentDB instance automatically.
- The backup capability in Amazon DocumentDB enables point-in-time recovery for your cluster. This feature allows you to restore your cluster to any second during your retention period, up to the last 5 minutes. You can configure your automatic backup retention period up to 35 days. Automated backups are stored in Amazon Simple Storage Service (Amazon S3), which is designed for 99.999999999% durability. Amazon DocumentDB backups are automatic, incremental, and continuous, and they have no impact on your cluster performance.
- With Amazon DocumentDB, you can encrypt your databases using keys that you create and control through AWS Key Management Service (AWS KMS). On a database cluster running with Amazon DocumentDB encryption, data stored at rest in the underlying storage is encrypted. The automated backups, snapshots, and replicas in the same cluster are also encrypted.

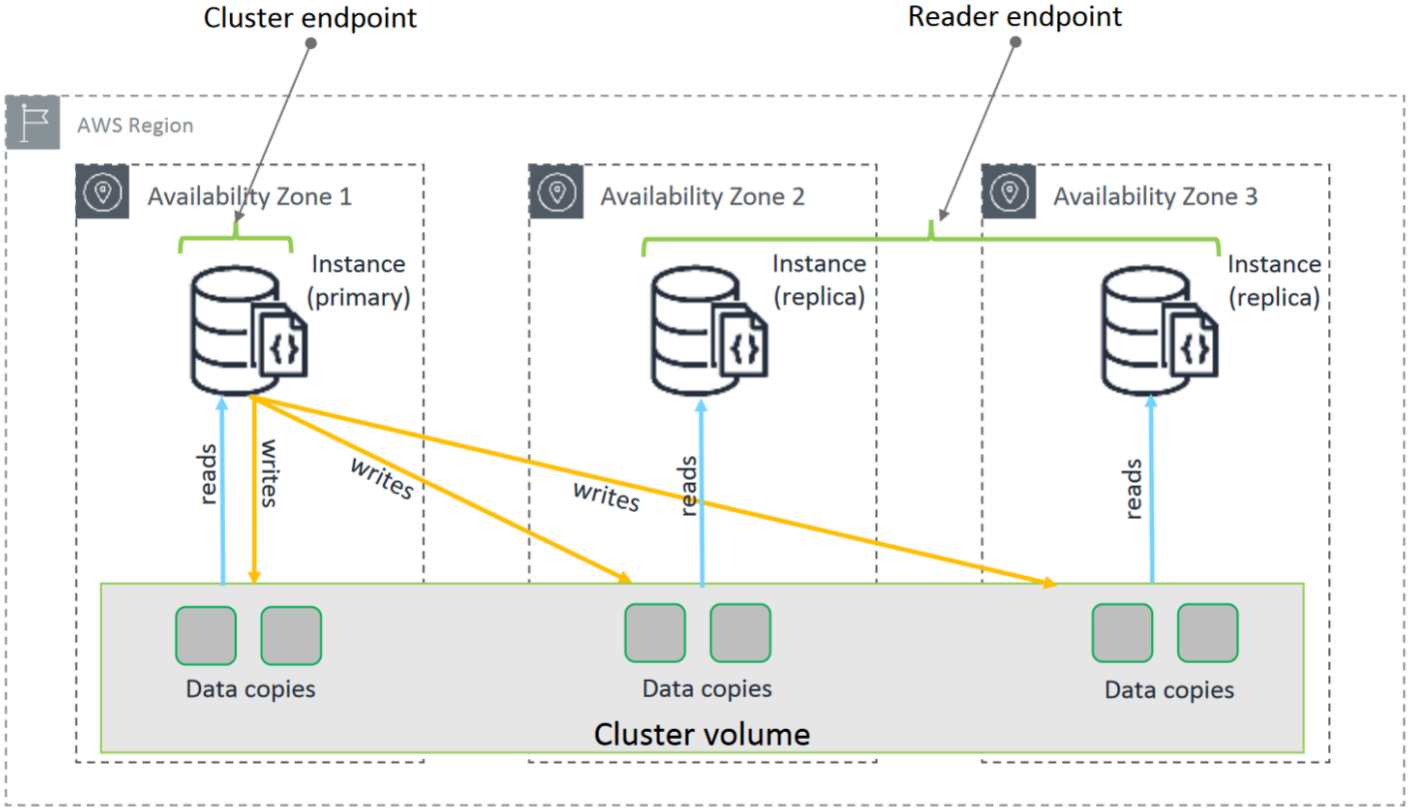
- Amazon DocumentDB is authorized under Federal Risk and Authorization Management Program (FedRAMP). It has FedRAMP High authorization for AWS GovCloud (US) regions and FedRAMP Moderate authorization for AWS US East/West Regions. For details about AWS and compliance efforts, see AWS Services in Scope by Compliance Program.

If you are new to AWS services, use the following resources to learn more:

- AWS offers services for computing, databases, storage, analytics, and other functionality. For an overview of all AWS services, see Cloud Computing with Amazon Web Services.
- AWS provides a number of database services. For guidance on which service is best for your environment, see Databases on AWS.

Clusters

A *cluster* consists of 0 to 16 instances and a cluster storage volume that manages the data for those instances. All writes are done through the primary instance. All instances (primary and replicas) support reads. The cluster's data is stored in the cluster volume with copies in three different Availability Zones.



Amazon DocumentDB 5.0 instance-based clusters support two storage configurations for a database cluster: Amazon DocumentDB standard and Amazon DocumentDB I/O-optimized. For more information see Amazon DocumentDB cluster storage configurations.

Instances

An Amazon DocumentDB instance is an isolated database environment in the cloud. An instance can contain multiple user-created databases. You can create and modify an instance using the AWS Management Console or the AWS CLI.

The computation and memory capacity of an instance are determined by its *instance class*. You can select the instance that best meets your needs. If your needs change over time, you can choose a different instance class. For instance class specifications, see Instance class specifications.

Amazon DocumentDB instances run only in the Amazon VPC environment. Amazon VPC gives you control of your virtual networking environment: You can choose your own IP address range, create subnets, and configure routing and access control lists (ACLs).

Before you can create Amazon DocumentDB instances, you must create a cluster to contain the instances.

Not all instance classes are supported in every region. The following table shows which instance classes are supported in each region.

> **Note**
>
> For a complete list of instance types supported by Amazon DocumentDB in each instance class, see Instance class specifications.

Supported instance classes by Region

Region	R6GD	R6G	R5	R4	T4G	T3
US East (Ohio)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
US East (N. Virginia)	Supporte	Supporte	Supported	Supporte	Supporte	Supported

Region	R6GD	R6G	R5	R4	T4G	T3
US West (Oregon)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
Africa (Cape Town)		Supporte	Supported		Supporte	Supported
South America (São Paulo)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Hong Kong)		Supporte	Supported		Supporte	Supported
Asia Pacific (Hyderabad)			Supported			Supported
Asia Pacific (Mumbai)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Seoul)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Sydney)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Singapore)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Tokyo)	Supporte	Supporte	Supported		Supporte	Supported
Canada (Central)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Frankfurt)	Supporte	Supporte	Supported		Supporte	Supported

Region	R6GD	R6G	R5	R4	T4G	T3
Europe (Ireland)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
Europe (London)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Milan)		Supporte	Supported		Supporte	Supported
Europe (Paris)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Spain)		Supporte	Supported		Supporte	Supported
Europe (Stockholm)		Supporte	Supported		Supporte	Supported
Middle East (UAE)		Supporte	Supported		Supporte	Supported
China (Beijing)	Supporte	Supporte	Supported		Supporte	Supported
China (Ningxia)		Supporte	Supported		Supporte	Supported
Israel (Tel Aviv)		Supporte	Supported		Supporte	Supported
AWS GovCloud (US-West)	Supporte	Supporte	Supported		Supporte	Supported
AWS GovCloud (US-East)	Supporte	Supporte	Supported		Supporte	Supported

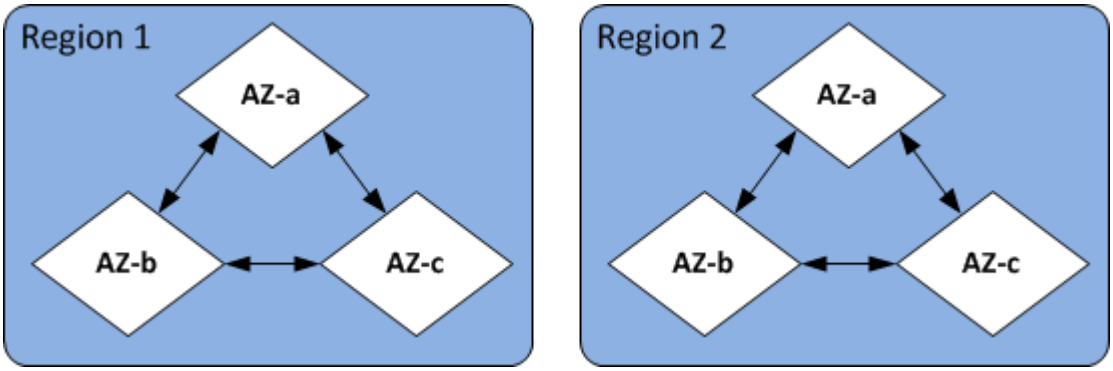
Regions and availability zones

Regions and Availability Zones define the physical locations of your cluster and instances.

Regions

AWS Cloud computing resources are housed in highly available data center facilities in different areas of the world (for example, North America, Europe, or Asia). Each data center location is called a *Region*.

Each AWS Region is designed to be completely isolated from the other AWS Regions. Within each are multiple Availability Zones. By launching your nodes in different Availability Zones, you can achieve the greatest possible fault tolerance. The following diagram shows a high-level view of how AWS Regions and Availability Zones work.



Availability zones

Each AWS Region contains multiple distinct locations called *Availability Zones*. Each Availability Zone is engineered to be isolated from failures in other Availability Zones, and to provide inexpensive, low-latency network connectivity to other Availability Zones in the same Region. By launching instances for a given cluster in multiple Availability Zones, you can protect your applications from the unlikely event of an Availability Zone failing.

The Amazon DocumentDB architecture separates storage and compute. For the storage layer, Amazon DocumentDB replicates six copies of your data across three AWS Availability Zones. As an example, if you are launching an Amazon DocumentDB cluster in a Region that only supports two Availability Zones, your data storage will be replicated six ways across three Availability Zones but your compute instances will only be available in two Availability Zones.

The following table lists the number of Availability Zones that you can use in a given AWS Region to provision compute instances for your cluster.

Region Name	Region	Availability Zones (compute)
US East (Ohio)	`us-east-2`	3
US East (N. Virginia)	`us-east-1`	6
US West (Oregon)	`us-west-2`	4
Africa (Cape Town)	`af-south-1`	3
South America (São Paulo)	`sa-east-1`	3
Asia Pacific (Hong Kong)	`ap-east-1`	3
Asia Pacific (Hyderabad)	`ap-south-2`	3
Asia Pacific (Mumbai)	`ap-south-1`	3
Asia Pacific (Seoul)	`ap-northeast-2`	4
Asia Pacific (Singapore)	`ap-southeast-1`	3
Asia Pacific (Sydney)	`ap-southeast-2`	3
Asia Pacific (Tokyo)	`ap-northeast-1`	3
Canada (Central)	`ca-central-1`	3
China (Beijing) Region	`cn-north-1`	3
China (Ningxia)	`cn-northwest-1`	3
Europe (Frankfurt)	`eu-central-1`	3
Europe (Ireland)	`eu-west-1`	3
Europe (London)	`eu-west-2`	3

Region Name	Region	Availability Zones (compute)
Europe (Milan)	`eu-south-1`	3
Europe (Paris)	`eu-west-3`	3
Europe (Spain)	`eu-south-2`	3
Europe (Stockholm)	`eu-north-1`	3
Middle East (UAE)	`me-central-1`	3
Israel (Tel Aviv)	`il-central-1`	3
AWS GovCloud (US-West)	`us-gov-west-1`	3
AWS GovCloud (US-East)	`us-gov-east-1`	3

Amazon DocumentDB Pricing

Amazon DocumentDB clusters are billed based on the following components:

- **Instance hours (per hour)**—Based on the instance class of the instance (for example, `db.r5.xlarge`). Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. Amazon DocumentDB usage is billed in one second increments, with a minimum of 10 minutes. For more information, see Managing instance classes.
- **I/O requests (per 1 million requests per month)** — Total number of storage I/O requests that you make in a billing cycle.
- **Backup storage (per GiB per month)** — Backup storage is the storage that is associated with automated database backups and any active database snapshots that you have taken. Increasing your backup retention period or taking additional database snapshots increases the backup storage consumed by your database. Backup storage is metered in GB-months and per second does not apply. For more information, see Backing up and restoring in Amazon DocumentDB.
- **Data transfer (per GB)** — Data transfer in and out of your instance from or to the internet or other AWS Regions.

For detailed information, see Amazon DocumentDB pricing.

Free trial

You can try Amazon DocumentDB for free using the 1-month free trial. For more information, see Free trial in Amazon DocumentDB pricing or see the Amazon DocumentDB free trial FAQ.

Monitoring

There are several ways that you can track the performance and health of an instance. You can use the free Amazon CloudWatch service to monitor the performance and health of an instance. You can find performance charts on the Amazon DocumentDB console. You can subscribe to Amazon DocumentDB events to be notified when changes occur with an instance, snapshot, parameter group, or security group.

For more information, see the following:

- Monitoring Amazon DocumentDB with CloudWatch
- Logging Amazon DocumentDB API calls with AWS CloudTrail

Interfaces

There are multiple ways for you to interact with Amazon DocumentDB, including the AWS Management Console and the AWS CLI.

AWS Management Console

The AWS Management Console is a simple web-based user interface. You can manage your clusters and instances from the console with no programming required. To access the Amazon DocumentDB console, sign in to the AWS Management Console and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to manage your Amazon DocumentDB clusters and instances. With minimal configuration, you can start using all of the functionality provided by the Amazon DocumentDB console from your favorite terminal program.

- To install the AWS CLI, see Installing the AWS Command Line Interface.

- To begin using the AWS CLI for Amazon DocumentDB, see AWS Command Line Interface Reference for Amazon DocumentDB.

MongoDB drivers

For developing and writing applications against an Amazon DocumentDB cluster, you can also use the MongoDB drivers with Amazon DocumentDB. For more information, see the MongoDB shell tab in Connecting with TLS enabled or Connecting with TLS disabled.

What's next?

The preceding sections introduced you to the basic infrastructure components that Amazon DocumentDB offers. What should you do next? Depending upon your circumstances, see one of the following topics to get started:

- Get started with Amazon DocumentDB by creating a cluster and instance using AWS CloudFormation Amazon DocumentDB quick start using AWS CloudFormation.
- Get started with Amazon DocumentDB by creating a cluster and instance using the instructions in our Get started guide.
- Get started with Amazon DocumentDB by creating an elastic cluster using the instructions in Get started with Amazon DocumentDB elastic clusters.
- Migrate your MongoDB implementation to Amazon DocumentDB using the guidance at Migrating to Amazon DocumentDB

Amazon DocumentDB: how it works

Amazon DocumentDB (with MongoDB compatibility) is a fully managed, MongoDB-compatible database service. With Amazon DocumentDB, you can run the same application code and use the same drivers and tools that you use with MongoDB. Amazon DocumentDB is compatible with MongoDB 3.6, 4.0, and 5.0.

Topics

- Amazon DocumentDB endpoints
- TLS support
- Amazon DocumentDB storage

- Amazon DocumentDB replication
- Amazon DocumentDB reliability
- Read preference options
- TTL deletes
- Billable resources

When you use Amazon DocumentDB, you begin by creating a *cluster*. A cluster consists of zero or more database instances and a cluster volume that manages the data for those instances. An Amazon DocumentDB *cluster volume* is a virtual database storage volume that spans multiple Availability Zones. Each Availability Zone has a copy of the cluster data.

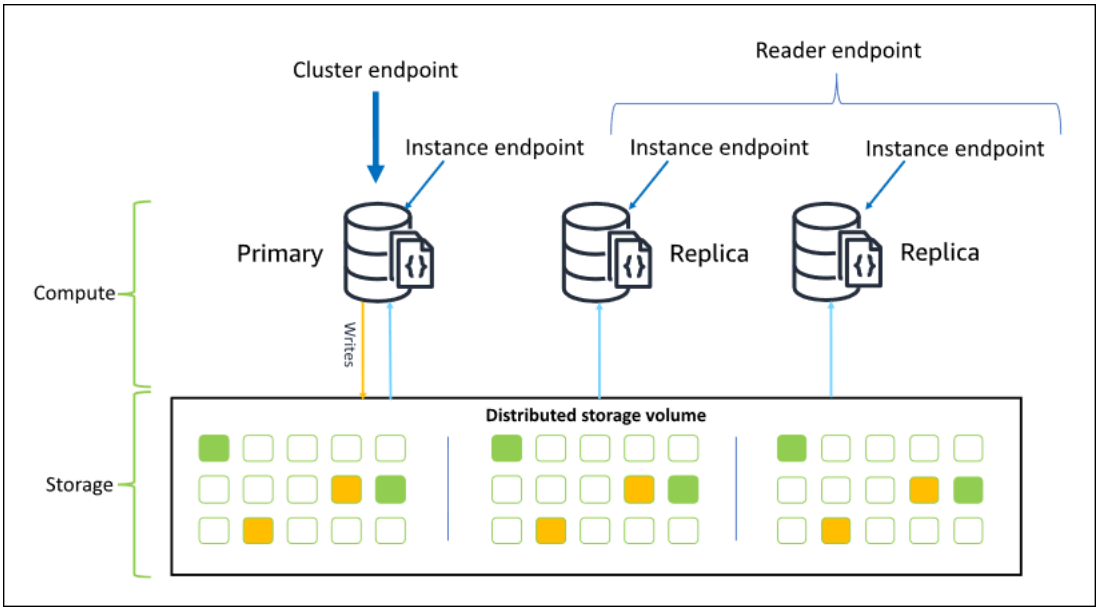
An Amazon DocumentDB cluster consists of two components:

- **Cluster volume**—Uses a cloud-native storage service to replicate data six ways across three Availability Zones, providing highly durable and available storage. An Amazon DocumentDB cluster has exactly one cluster volume, which can store up to 128 TiB of data.
- **Instances**—Provide the processing power for the database, writing data to, and reading data from, the cluster storage volume. An Amazon DocumentDB cluster can have 0–16 instances.

Instances serve one of two roles:

- **Primary instance**—Supports read and write operations, and performs all the data modifications to the cluster volume. Each Amazon DocumentDB cluster has one primary instance.
- **Replica instance**—Supports only read operations. An Amazon DocumentDB cluster can have up to 15 replicas in addition to the primary instance. Having multiple replicas enables you to distribute read workloads. In addition, by placing replicas in separate Availability Zones, you also increase your cluster availability.

The following diagram illustrates the relationship between the cluster volume, the primary instance, and replicas in an Amazon DocumentDB cluster:



Cluster instances do not need to be of the same instance class, and they can be provisioned and terminated as desired. This architecture lets you scale your cluster's compute capacity independently of its storage.

When your application writes data to the primary instance, the primary executes a durable write to the cluster volume. It then replicates the state of that write (not the data) to each active replica. Amazon DocumentDB replicas do not participate in processing writes, and thus Amazon DocumentDB replicas are advantageous for read scaling. Reads from Amazon DocumentDB replicas are eventually consistent with minimal replica lag—usually less than 100 milliseconds after the primary instance writes the data. Reads from the replicas are guaranteed to be read in the order in which they were written to the primary. Replica lag varies depending on the rate of data change, and periods of high write activity might increase the replica lag. For more information, see the `ReplicationLag` metrics at Amazon DocumentDB metrics.

Amazon DocumentDB endpoints

Amazon DocumentDB provides multiple connection options to serve a wide range of use cases. To connect to an instance in an Amazon DocumentDB cluster, you specify the instance's endpoint. An *endpoint* is a host address and a port number, separated by a colon.

We recommend that you connect to your cluster using the cluster endpoint and in replica set mode (see Connecting to Amazon DocumentDB as a replica set) unless you have a specific use case for connecting to the reader endpoint or an instance endpoint. To route requests to your replicas, choose a driver read preference setting that maximizes read scaling while meeting your

application's read consistency requirements. The `secondaryPreferred` read preference enables replica reads and frees up the primary instance to do more work.

The following endpoints are available from an Amazon DocumentDB cluster.

Cluster Endpoint

The *cluster endpoint* connects to your cluster's current primary instance. The cluster endpoint can be used for read and write operations. An Amazon DocumentDB cluster has exactly one cluster endpoint.

The cluster endpoint provides failover support for read and write connections to the cluster. If your cluster's current primary instance fails, and your cluster has at least one active read replica, the cluster endpoint automatically redirects connection requests to a new primary instance. When connecting to your Amazon DocumentDB cluster, we recommend that you connect to your cluster using the cluster endpoint and in replica set mode (see Connecting to Amazon DocumentDB as a replica set).

The following is an example Amazon DocumentDB cluster endpoint:

```
sample-cluster.cluster-123456789012.us-east-1.docdb.amazonaws.com:27017
```

The following is an example connection string using this cluster endpoint:

```
mongodb://username:password@sample-cluster.cluster-123456789012.us-
east-1.docdb.amazonaws.com:27017
```

For information about finding a cluster's endpoints, see Finding a cluster's endpoints.

Reader endpoint

The *reader endpoint* load balances read-only connections across all available replicas in your cluster. A cluster reader endpoint will perform as the cluster endpoint if you are connecting through `replicaSet` mode, meaning in the connection string, the replica set parameter is `&replicaSet=rs0`. In this case, you will be able to perform write operations on the primary. However, if you connect to the cluster specifying `directConnection=true`, then attempting to perform a write operation over a connection to the reader endpoint results in an error. An Amazon DocumentDB cluster has exactly one reader endpoint.

If the cluster contains only one (primary) instance, the reader endpoint connects to the primary instance. When you add a replica instance to your Amazon DocumentDB cluster, the reader endpoint opens read-only connections to the new replica after it is active.

The following is an example reader endpoint for an Amazon DocumentDB cluster:

```
sample-cluster.cluster-ro-123456789012.us-east-1.docdb.amazonaws.com:27017
```

The following is an example connection string using a reader endpoint:

```
mongodb://username:password@sample-cluster.cluster-ro-123456789012.us-
east-1.docdb.amazonaws.com:27017
```

The reader endpoint load balances read-only connections, not read requests. If some reader endpoint connections are more heavily used than others, your read requests might not be equally balanced among instances in the cluster. It is recommended to distribute requests by connecting to the cluster endpoint as a replica set and utilizing the secondaryPreferred read preference option.

For information about finding a cluster's endpoints, see Finding a cluster's endpoints.

Instance endpoint

An *instance endpoint* connects to a specific instance within your cluster. The instance endpoint for the current primary instance can be used for read and write operations. However, attempting to perform write operations to an instance endpoint for a read replica results in an error. An Amazon DocumentDB cluster has one instance endpoint per active instance.

An instance endpoint provides direct control over connections to a specific instance for scenarios in which the cluster endpoint or reader endpoint might not be appropriate. An example use case is provisioning for a periodic read-only analytics workload. You can provision a larger-than-normal replica instance, connect directly to the new larger instance with its instance endpoint, run the analytics queries, and then terminate the instance. Using the instance endpoint keeps the analytics traffic from impacting other cluster instances.

The following is an example instance endpoint for a single instance in an Amazon DocumentDB cluster:

```
sample-instance.123456789012.us-east-1.docdb.amazonaws.com:27017
```

The following is an example connection string using this instance endpoint:

```
mongodb://username:password@sample-instance.123456789012.us-
east-1.docdb.amazonaws.com:27017
```

> **Note**
>
> An instance's role as primary or replica can change due to a failover event. Your applications should never assume that a particular instance endpoint is the primary instance. We do not recommend connecting to instance endpoints for production applications. Instead, we recommend that you connect to your cluster using the cluster endpoint and in replica set mode (see Connecting to Amazon DocumentDB as a replica set). For more advanced control of instance failover priority, see Understanding Amazon DocumentDB cluster fault tolerance.

For information about finding a cluster's endpoints, see Finding an instance's endpoint.

Replica set mode

You can connect to your Amazon DocumentDB cluster endpoint in replica set mode by specifying the replica set name `rs0`. Connecting in replica set mode provides the ability to specify the Read Concern, Write Concern, and Read Preference options. For more information, see Read consistency.

The following is an example connection string connecting in replica set mode:

```
mongodb://username:password@sample-cluster.cluster-123456789012.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0
```

When you connect in replica set mode, your Amazon DocumentDB cluster appears to your drivers and clients as a replica set. Instances added and removed from your Amazon DocumentDB cluster are reflected automatically in the replica set configuration.

Each Amazon DocumentDB cluster consists of a single replica set with the default name `rs0`. The replica set name cannot be modified.

Connecting to the cluster endpoint in replica set mode is the recommended method for general use.

> **Note**
>
> All instances in an Amazon DocumentDB cluster listen on the same TCP port for connections.

TLS support

For more details on connecting to Amazon DocumentDB using Transport Layer Security (TLS), see Encrypting data in transit.

Amazon DocumentDB storage

Amazon DocumentDB data is stored in a *cluster volume*, which is a single, virtual volume that uses solid state drives (SSDs). A cluster volume consists of six copies of your data, which are replicated automatically across multiple Availability Zones in a single AWS Region. This replication helps ensure that your data is highly durable, with less possibility of data loss. It also helps ensure that your cluster is more available during a failover because copies of your data already exist in other Availability Zones. These copies can continue to serve data requests to the instances in your Amazon DocumentDB cluster.

How data storage is billed

Amazon DocumentDB automatically increases the size of a cluster volume as the amount of data increases. An Amazon DocumentDB cluster volume can grow to a maximum size of 128 TiB; however, you are only charged for the space that you use in an Amazon DocumentDB cluster volume. Starting with Amazon DocumentDB 4.0, when data is removed, such as by dropping a collection or index, the overall allocated space decreases by a comparable amount. Thus, you can reduce storage charges by deleting collections, indexes, and databases that you no longer need. In Amazon DocumentDB version 3.6, the cluster volume can reuse space that's freed up when you remove data, but the volume itself never decreases in size. As a result in version 3.6, you may not witness any change in storage when you drop a collection or index, even though the freed up space is reused.

> **Note**
>
> With Amazon DocumentDB 3.6, storage costs are based on the storage "high water mark" (the maximum amount that was allocated for the Amazon DocumentDB cluster at any point in time). You can manage costs by avoiding ETL practices that create large

volumes of temporary information, or that load large volumes of new data prior to removing unneeded older data. If removing data from an Amazon DocumentDB cluster results in a substantial amount of allocated but unused space, resetting the high water mark requires doing a logical data dump and restore to a new cluster, using a tool such as `mongodump` or `mongorestore`. Creating and restoring a snapshot does not reduce the allocated storage because the physical layout of the underlying storage remains the same in the restored snapshot.

Note

Using utilities like `mongodump` and `mongorestore` incur I/O charges based on the sizes of the data that is being read and written to the storage volume.

For information about Amazon DocumentDB data storage and I/O pricing, see Amazon DocumentDB (with MongoDB compatibility) pricing and Pricing FAQs.

Amazon DocumentDB replication

In an Amazon DocumentDB cluster, each replica instance exposes an independent endpoint. These replica endpoints provide read-only access to the data in the cluster volume. They enable you to scale the read workload for your data over multiple replicated instances. They also help improve the performance of data reads and increase the availability of the data in your Amazon DocumentDB cluster. Amazon DocumentDB replicas are also failover targets and are quickly promoted if the primary instance for your Amazon DocumentDB cluster fails.

Amazon DocumentDB reliability

Amazon DocumentDB is designed to be reliable, durable, and fault tolerant. (To improve availability, you should configure your Amazon DocumentDB cluster so that it has multiple replica instances in different Availability Zones.) Amazon DocumentDB includes several automatic features that make it a reliable database solution.

Storage auto-repair

Amazon DocumentDB maintains multiple copies of your data in three Availability Zones, greatly reducing the chance of losing data due to a storage failure. Amazon DocumentDB automatically

detects failures in the cluster volume. When a segment of a cluster volume fails, Amazon DocumentDB immediately repairs the segment. It uses the data from the other volumes that make up the cluster volume to help ensure that the data in the repaired segment is current. As a result, Amazon DocumentDB avoids data loss and reduces the need to perform a point-in-time restore to recover from an instance failure.

Survivable cache warming

Amazon DocumentDB manages its page cache in a separate process from the database so that the page cache can survive independently of the database. In the unlikely event of a database failure, the page cache remains in memory. This ensures that the buffer pool is warmed with the most current state when the database restarts.

Crash recovery

Amazon DocumentDB is designed to recover from a crash almost instantaneously, and to continue serving your application data. Amazon DocumentDB performs crash recovery asynchronously on parallel threads so that your database is open and available almost immediately after a crash.

Resource governance

Amazon DocumentDB safeguards resources that are needed to run critical processes in the service, such as health checks. To do this, and when an instance is experiencing high memory pressure, Amazon DocumentDB will throttle requests. As a result, some operations may be queued to wait for the memory pressure to subside. If memory pressure continues, queued operations may timeout. You can monitor whether or not the service throttling operations due to low memory with the following CloudWatch metrics: `LowMemThrottleQueueDepth`, `LowMemThrottleMaxQueueDepth`, `LowMemNumOperationsThrottled`, `LowMemNumOperationsTimedOut`. For more information, see Monitoring Amazon DocumentDB with CloudWatch. If you see sustained memory pressure on your instance as a result of the LowMem CloudWatch metrics, we advise that you scale-up your instance to provide additional memory for your workload.

Read preference options

Amazon DocumentDB uses a cloud-native shared storage service that replicates data six times across three Availability Zones to provide high levels of durability. Amazon DocumentDB does not rely on replicating data to multiple instances to achieve durability. Your cluster’s data is durable whether it contains a single instance or 15 instances.

Topics

- Write durability
- Read isolation
- Read consistency
- High availability
- Scaling reads

Write durability

Amazon DocumentDB uses a unique, distributed, fault-tolerant, self-healing storage system. This system replicates six copies (V=6) of your data across three AWS Availability Zones to provide high availability and durability. When writing data, Amazon DocumentDB ensures that all writes are durably recorded on a majority of nodes before acknowledging the write to the client. If you are running a three-node MongoDB replica set, using a write concern of `{w:3, j:true}` would yield the best possible configuration when comparing with Amazon DocumentDB.

Writes to an Amazon DocumentDB cluster must be processed by the cluster's writer instance. Attempting to write to a reader results in an error. An acknowledged write from an Amazon DocumentDB primary instance is durable, and can't be rolled back. Amazon DocumentDB is highly durable by default and doesn't support a non-durable write option. You can't modify the durability level (that is, write concern). Amazon DocumentDB ignores w=anything and is effectively w: 3 and j: true. You cannot reduce it.

Because storage and compute are separated in the Amazon DocumentDB architecture, a cluster with a single instance is highly durable. Durability is handled at the storage layer. As a result, an Amazon DocumentDB cluster with a single instance and one with three instances achieve the same level of durability. You can configure your cluster to your specific use case while still providing high durability for your data.

Writes to an Amazon DocumentDB cluster are atomic within a single document.

Amazon DocumentDB does not support the `wtimeout` option and will not return an error if a value is specified. Writes to the primary Amazon DocumentDB instance are guaranteed not to block indefinitely.

Read isolation

Reads from an Amazon DocumentDB instance only return data that is durable before the query begins. Reads never return data modified after the query begins execution nor are dirty reads possible under any circumstances.

Read consistency

Data read from an Amazon DocumentDB cluster is durable and will not be rolled back. You can modify the read consistency for Amazon DocumentDB reads by specifying the read preference for the request or connection. Amazon DocumentDB does not support a non-durable read option.

Reads from an Amazon DocumentDB cluster's primary instance are strongly consistent under normal operating conditions and have read-after-write consistency. If a failover event occurs between the write and subsequent read, the system can briefly return a read that is not strongly consistent. All reads from a read replica are eventually consistent and return the data in the same order, and often with less than 100 ms replica lag.

Amazon DocumentDB read preferences

Amazon DocumentDB supports setting a read preference option only when reading data from the cluster endpoint in replica set mode. Setting a read preference option affects how your MongoDB client or driver routes read requests to instances in your Amazon DocumentDB cluster. You can set read preference options for a specific query, or as a general option in your MongoDB driver. (Consult your client or driver's documentation for instructions on how to set a read preference option.)

If your client or driver is not connecting to an Amazon DocumentDB cluster endpoint in replica set mode, the result of specifying a read preference is undefined.

Amazon DocumentDB does not support setting *tag sets* as a read preference.

Supported Read Preference Options

- **`primary`**—Specifying a `primary` read preference helps ensure that all reads are routed to the cluster's primary instance. If the primary instance is unavailable, the read operation fails. A `primary` read preference yields read-after-write consistency and is appropriate for use cases that prioritize read-after-write consistency over high availability and read scaling.

  The following example specifies a `primary` read preference:

```
db.example.find().readPref('primary')
```

- **primaryPreferred**—Specifying a primaryPreferred read preference routes reads to the primary instance under normal operation. If there is a primary failover, the client routes requests to a replica. A primaryPreferred read preference yields read-after-write consistency during normal operation, and eventually consistent reads during a failover event. A primaryPreferred read preference is appropriate for use cases that prioritize read-after-write consistency over read scaling, but still require high availability.

  The following example specifies a primaryPreferred read preference:

```
db.example.find().readPref('primaryPreferred')
```

- **secondary**—Specifying a secondary read preference ensures that reads are only routed to a replica, never the primary instance. If there are no replica instances in a cluster, the read request fails. A secondary read preference yields eventually consistent reads and is appropriate for use cases that prioritize primary instance write throughput over high availability and read-after-write consistency.

  The following example specifies a secondary read preference:

```
db.example.find().readPref('secondary')
```

- **secondaryPreferred**—Specifying a secondaryPreferred read preference ensures that reads are routed to a read replica when one or more replicas are active. If there are no active replica instances in a cluster, the read request is routed to the primary instance. A secondaryPreferred read preference yields eventually consistent reads when the read is serviced by a read replica. It yields read-after-write consistency when the read is serviced by the primary instance (barring failover events). A secondaryPreferred read preference is appropriate for use cases that prioritize read scaling and high availability over read-after-write consistency.

  The following example specifies a secondaryPreferred read preference:

```
db.example.find().readPref('secondaryPreferred')
```

- **`nearest`**—Specifying a `nearest` read preference routes reads based solely on the measured latency between the client and all instances in the Amazon DocumentDB cluster. A `nearest` read preference yields eventually consistent reads when the read is serviced by a read replica. It yields read-after-write consistency when the read is serviced by the primary instance (barring failover events). A `nearest` read preference is appropriate for use cases that prioritize achieving the lowest possible read latency and high availability over read-after-write consistency and read scaling.

  The following example specifies a `nearest` read preference:

  ```
  db.example.find().readPref('nearest')
  ```

High availability

Amazon DocumentDB supports highly available cluster configurations by using replicas as failover targets for the primary instance. If the primary instance fails, an Amazon DocumentDB replica is promoted as the new primary, with a brief interruption during which read and write requests made to the primary instance fail with an exception.

If your Amazon DocumentDB cluster doesn't include any replicas, the primary instance is re-created during a failure. However, promoting an Amazon DocumentDB replica is much faster than re-creating the primary instance. So we recommend that you create one or more Amazon DocumentDB replicas as failover targets.

Replicas that are intended for use as failover targets should be of the same instance class as the primary instance. They should be provisioned in different Availability Zones from the primary. You can control which replicas are preferred as failover targets. For best practices on configuring Amazon DocumentDB for high availability, see Understanding Amazon DocumentDB cluster fault tolerance.

Scaling reads

Amazon DocumentDB replicas are ideal for read scaling. They are fully dedicated to read operations on your cluster volume, that is, replicas do not process writes. Data replication happens

within the cluster volume and not between instances. So each replica's resources are dedicated to processing your queries, not replicating and writing data.

If your application needs more read capacity, you can add a replica to your cluster quickly (usually in less than ten minutes). If your read capacity requirements diminish, you can remove unneeded replicas. With Amazon DocumentDB replicas, you pay only for the read capacity that you need.

Amazon DocumentDB supports client-side read scaling through the use of Read Preference options. For more information, see Amazon DocumentDB read preferences.

TTL deletes

Deletes from a TTL index area achieved via a background process are best effort and are not guaranteed within a specific timeframe. Factors like instance size, instance resource utilization, document size, and overall throughput can affect the timing of a TTL delete.

When the TTL monitor deletes your documents, each deletion incurs IO costs, which will increase your bill. If throughput and TTL delete rates increase, you should expect an increase in your bill due to increased IO usage.

When you create a TTL index on an existing collection, you must delete all expired documents before creating the index. The current TTL implementation is optimized for deleting a small fraction of documents in the collection, which is typical if TTL was enabled on the collection from the start, and may result in higher IOPS than necessary if a large number of documents need to be deleted at one go.

If you do not want to create a TTL index to delete documents, you can instead segment documents into collections based on time, and simply drop those collections when the documents are no longer needed. For example: you can create one collection per week and drop it without incurring IO costs. This can be significantly more cost effective than using a TTL index.

Billable resources

Identifying billable Amazon DocumentDB resources

As a fully managed database service, Amazon DocumentDB charges for instances, storage, I/Os, backups, and data transfer. For more information, see Amazon DocumentDB (with MongoDB compatibility) pricing.

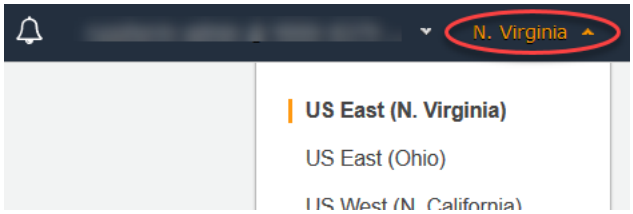
To discover billable resources in your account and potentially delete the resources, you can use the AWS Management Console or AWS CLI.

Using the AWS Management Console

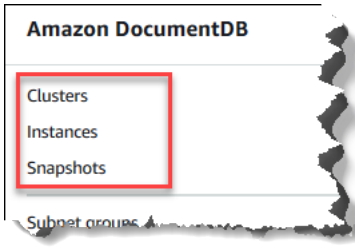
Using the AWS Management Console, you can discover the Amazon DocumentDB clusters, instances, and snapshots that you have provisioned for a given AWS Region.

To discover clusters, instances, and snapshots

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. To discover billable resources in a Region other than your default Region, in the upper-right corner of the screen, choose the AWS Region that you want to search.



3. In the navigation pane, choose the type of billable resource that you're interested in: **Clusters**, **Instances**, or **Snapshots**.



4. All your provisioned clusters, instances, or snapshots for the Region are listed in the right pane. You will be charged for clusters, instances, and snapshots.

Using the AWS CLI

Using the AWS CLI, you can discover the Amazon DocumentDB clusters, instances, and snapshots that you have provisioned for a given AWS Region.

To discover clusters and instances

The following code lists all your clusters and instances for the specified Region. If you want to search for clusters and instances in your default Region, you can omit the `--region` parameter.

Example

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
    --region us-east-1 \
    --query 'DBClusters[?Engine==`docdb`]' | \
       grep -e "DBClusterIdentifier" -e "DBInstanceIdentifier"
```

For Windows:

```
aws docdb describe-db-clusters ^
    --region us-east-1 ^
    --query 'DBClusters[?Engine==`docdb`]' | ^
       grep -e "DBClusterIdentifier" -e "DBInstanceIdentifier"
```

Output from this operation looks something like the following.

```
"DBClusterIdentifier": "docdb-2019-01-09-23-55-38",
        "DBInstanceIdentifier": "docdb-2019-01-09-23-55-38",
        "DBInstanceIdentifier": "docdb-2019-01-09-23-55-382",
"DBClusterIdentifier": "sample-cluster",
"DBClusterIdentifier": "sample-cluster2",
```

To discover snapshots

The following code lists all your snapshots for the specified Region. If you want to search for snapshots in your default Region, you can omit the `--region` parameter.

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-snapshots \
  --region us-east-1 \
  --query 'DBClusterSnapshots[?Engine==`docdb`].
[DBClusterSnapshotIdentifier,SnapshotType]'
```

For Windows:

```
aws docdb describe-db-cluster-snapshots ^
  --region us-east-1 ^
```

```
  --query 'DBClusterSnapshots[?Engine==`docdb`].
[DBClusterSnapshotIdentifier,SnapshotType]'
```

Output from this operation looks something like the following.

```
[
    [
        "rds:docdb-2019-01-09-23-55-38-2019-02-13-00-06",
        "automated"
    ],
    [
        "test-snap",
        "manual"
    ]
]
```

You only need to delete `manual` snapshots. `Automated` snapshots are deleted when you delete the cluster.

Deleting unwanted billable resources

To delete a cluster, you must first delete all the instances in the cluster.

- To delete instances, see Deleting an Amazon DocumentDB instance.

> **Important**
>
> Even if you delete the instances in a cluster, you are still billed for the storage and backup usage associated with that cluster. To stop all charges, you must also delete your cluster and manual snapshots.

- To delete clusters, see Deleting an Amazon DocumentDB cluster.
- To delete manual snapshots, see Deleting a cluster snapshot.

What is a document database?

Some developers don't think of their data model in terms of normalized rows and columns. Typically, in the application tier, data is represented as a JSON document because it is more intuitive for developers to think of their data model as a document.

The popularity of document databases has grown because they let you persist data in a database by using the same document model format that you use in your application code. Document databases provide powerful and intuitive APIs for flexible and agile development.

Topics

- Document database use Cases
- Understanding documents
- Working with documents

Document database use Cases

Your use case drives whether you need a document database or some other type of database for managing your data. Document databases are useful for workloads that require a flexible schema for fast, iterative development. The following are some examples of use cases for which document databases can provide significant advantages:

Topics

- User profiles
- Real-time big data
- Content management

User profiles

Because document databases have a flexible schema, they can store documents that have different attributes and data values. Document databases are a practical solution to online profiles in which different users provide different types of information. Using a document database, you can store each user's profile efficiently by storing only the attributes that are specific to each user.

Suppose that a user elects to add or remove information from their profile. In this case, their document could be easily replaced with an updated version that contains any recently added attributes and data or omits any newly omitted attributes and data. Document databases easily manage this level of individuality and fluidity.

Real-time big data

Historically, the ability to extract information from operational data was hampered by the fact that operational databases and analytical databases were maintained in different environments—

operational and business/reporting respectively. Being able to extract operational information in real time is critical in a highly competitive business environment. By using document databases, a business can store and manage operational data from any source and concurrently feed the data to the BI engine of choice for analysis. There is no requirement to have two environments.

Content management

To effectively manage content, you must be able to collect and aggregate content from a variety of sources, and then deliver it to the customer. Due to their flexible schema, document databases are perfect for collecting and storing any type of data. You can use them to create and incorporate new types of content, including user-generated content, such as images, comments, and videos.

Understanding documents

Document databases are used for storing semistructured data as a document—rather than normalizing data across multiple tables, each with a unique and fixed structure, as in a relational database. Documents stored in a document database use nested key-value pairs to provide the document's structure or schema. However, different types of documents can be stored in the same document database, thus meeting the requirement for processing similar data that is in different formats. For example, because each document is self-describing, the JSON-encoded documents for an online store that are described in the topic Example documents in a document database can be stored in the same document database.

Topics

- SQL vs. non-relational terminology
- Simple documents
- Embedded documents
- Example documents in a document database
- Understanding normalization in a document database

SQL vs. non-relational terminology

The following table compares terminology used by document databases (MongoDB) with terminology used by SQL databases.

SQL	MongoDB
Table	Collection
Row	Document
Column	Field
Primary key	ObjectId
Index	Index
View	View
Nested table or object	Embedded document
Array	Array

Simple documents

All documents in a document database are self-describing. This documentation uses JSON-like formatted documents, although you can use other means of encoding.

A simple document has one or more fields that are all at the same level within the document. In the following example, the fields SSN, `LName`, `FName`, `DOB`, `Street`, `City`, `State-Province`, `PostalCode`, and `Country` are all siblings within the document.

```
{
   "SSN": "123-45-6789",
   "LName": "Rivera",
   "FName": "Martha",
   "DOB": "1992-11-16",
   "Street": "125 Main St.",
   "City": "Anytown",
   "State-Province": "WA",
   "PostalCode": "98117",
   "Country": "USA"
}
```

When information is organized in a simple document, each field is managed individually. To retrieve a person's address, you must retrieve `Street`, `City`, `State-Province`, `PostalCode`, and `Country` as individual data items.

Embedded documents

A complex document organizes its data by creating embedded documents within the document. Embedded documents help manage data in groupings and as individual data items, whichever is more efficient in a given case. Using the preceding example, you could embed an `Address` document in the main document. Doing this results in the following document structure:

```
{
   "SSN": "123-45-6789",
   "LName": "Rivera",
   "FName": "Martha",
   "DOB": "1992-11-16",
   "Address":
   {
       "Street": "125 Main St.",
       "City": "Anytown",
       "State-Province": "WA",
       "PostalCode": "98117",
       "Country": "USA"
   }
}
```

You can now access the data in the document as individual fields (`"SSN":`), as an embedded document (`"Address":`), or as a member of an embedded document (`"Address": {"Street":}`).

Example documents in a document database

As stated earlier, because each document in a document database is self-describing, the structure of documents within a document database can be different from one another. The following two documents, one for a book and another for a periodical, are different structurally. Yet both of them can be in the same document database.

The following is a sample book document:

```
{
```

```
    "_id" : "9876543210123",
    "Type": "book",
    "ISBN": "987-6-543-21012-3",
    "Author":
    {
        "LName":"Roe",
        "MI": "T",
        "FName": "Richard"
    },
    "Title": "Understanding Document Databases"
}
```

The following is a sample periodical document with two articles:

```
{
    "_id" : "0123456789012",
    "Publication": "Programming Today",
    "Issue":
    {
        "Volume": "14",
        "Number": "09"
    },
    "Articles" : [
        {
            "Title": "Is a Document Database Your Best Solution?",
            "Author":
            {
                "LName": "Major",
                "FName": "Mary"
            }
        },
        {
            "Title": "Databases for Online Solutions",
            "Author":
            {
                "LName": "Stiles",
                "FName": "John"
            }
        }
    ],
    "Type": "periodical"
}
```

Compare the structure of these two documents. With a relational database, you need either separate "periodical" and "books" tables, or a single table with unused fields, such as "Publication," "Issue," "Articles," and "MI," as `null` values. Because document databases are semistructured, with each document defining its own structure, these two documents can coexist in the same document database with no `null` fields. Document databases are good at dealing with sparse data.

Developing against a document database enables quick, iterative development. This is because you can change the data structure of a document dynamically, without having to change the schema for the entire collection. Document databases are well suited for agile development and dynamically changing environments.

Understanding normalization in a document database

Document databases are not normalized; data found in one document can be repeated in another document. Further, some data discrepancies can exist between documents. For example, consider the scenario in which you make a purchase at an online store and all the details of your purchases are stored in a single document. The document might look something like the following JSON document:

```
{
    "DateTime": "2018-08-15T12:13:10Z",
    "LName" : "Santos",
    "FName" : "Paul",
    "Cart" : [
        {
            "ItemId" : "9876543210123",
            "Description" : "Understanding Document Databases",
            "Price" : "29.95"
        },
        {
            "ItemId" : "0123456789012",
            "Description" : "Programming Today",
            "Issue": {
                "Volume": "14",
                "Number": "09"
            },
            "Price" : "8.95"
        },
        {
            "ItemId": "234567890-K",
            "Description": "Gel Pen (black)",
```

```
            "Price": "2.49"
        }
    ],
    "PaymentMethod" :
    {
        "Issuer" : "MasterCard",
        "Number" : "1234-5678-9012-3456"
    },
    "ShopperId" : "1234567890"
}
```

All this information is stored as a document in a transaction collection. Later, you realize that you forgot to purchase one item. So you again log on to the same store and make another purchase, which is also stored as another document in the transaction collection.

```
{
    "DateTime": "2018-08-15T14:49:00Z",
    "LName" : "Santos",
    "FName" : "Paul",
    "Cart" : [
        {
            "ItemId" : "2109876543210",
            "Description" : "Document Databases for Fun and Profit",
            "Price" : "45.95"
        }
    ],
    "PaymentMethod" :
    {
        "Issuer" : "Visa",
        "Number" : "0987-6543-2109-8765"
    },
    "ShopperId" : "1234567890"
}
```

Notice the redundancy between these two documents—your name and shopper ID (and, if you used the same credit card, your credit card information). But that's okay because storage is inexpensive, and each document completely records a single transaction that can be retrieved quickly with a simple key-value query that requires no joins.

There is also an apparent discrepancy between the two documents—your credit card information. This is only an apparent discrepancy because it is likely that you used a different credit card for each purchase. Each document is accurate for the transaction that it documents.

Working with documents

As a document database, Amazon DocumentDB makes it easy to store, query, and index JSON data. In Amazon DocumentDB, a collection is analogous to a table in a relational database, except there is no single schema enforced upon all documents. Collections let you group similar documents together while keeping them all in the same database, without requiring that they be identical in structure.

Using the example documents from earlier sections, it is likely that you'd have collections for `reading_material` and `office_supplies`. It is the responsibility of your software to enforce which collection a document belongs in.

The following examples use the MongoDB API to show how to add, query, update, and delete documents.

Topics

- Adding documents
- Querying documents
- Updating documents
- Deleting documents

Adding documents

In Amazon DocumentDB, a database is created when first you add a document to a collection. In this example, you are creating a collection named `example` in the `test` database, which is the default database when you connect to a cluster. Because the collection is implicitly created when the first document is inserted, there is no error checking of the collection name. Therefore, a typo in the collection name, such as `eexample` instead of `example`, will create and add the document to `eexample` collection rather than the intended collection. Error checking must be handled by your application.

The following examples use the MongoDB API to add documents.

Topics

- Adding a single document
- Adding multiple documents

Adding a single document

To add a single document to a collection, use the `insertOne( {} )` operation with the document that you want added to the collection.

```
db.example.insertOne(
    {
        "Item": "Ruler",
        "Colors": ["Red","Green","Blue","Clear","Yellow"],
        "Inventory": {
            "OnHand": 47,
            "MinOnHand": 40
        },
        "UnitPrice": 0.89
    }
)
```

Output from this operation looks something like the following (JSON format).

```
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5bedafbcf65ff161707de24f")
}
```

Adding multiple documents

To add multiple documents to a collection, use the `insertMany( [{},...,{}] )` operation with a list of the documents that you want added to the collection. Although the documents in this particular list have different schemas, they can all be added to the same collection.

```
db.example.insertMany(
    [
        {
            "Item": "Pen",
            "Colors": ["Red","Green","Blue","Black"],
            "Inventory": {
                "OnHand": 244,
                "MinOnHand": 72
            }
        },
        {
```

```
            "Item": "Poster Paint",
            "Colors": ["Red","Green","Blue","Black","White"],
            "Inventory": {
                "OnHand": 47,
                "MinOnHand": 50
            }
        },
        {
            "Item": "Spray Paint",
            "Colors": ["Black","Red","Green","Blue"],
            "Inventory": {
                "OnHand": 47,
                "MinOnHand": 50,
                "OrderQnty": 36
            }
        }
    ]
)
```

Output from this operation looks something like the following (JSON format).

```
{
    "acknowledged" : true,
    "insertedIds" : [
            ObjectId("5bedb07941ca8d9198f5934c"),
            ObjectId("5bedb07941ca8d9198f5934d"),
            ObjectId("5bedb07941ca8d9198f5934e")
    ]
}
```

Querying documents

At times, you might need to look up your online store's inventory so that customers can see and purchase what you're selling. Querying a collection is relatively easy, whether you want all documents in the collection or only those documents that satisfy a particular criterion.

To query for documents, use the `find()` operation. The `find()` command has a single document parameter that defines the criteria to use in choosing the documents to return. The output from `find()` is a document formatted as a single line of text with no line breaks. To format the output document for easier reading, use `find().pretty()`. All the examples in this topic use `.pretty()` to format the output.

Use the four documents you inserted into the `example` collection in the preceding two exercises — `insertOne()` and `insertMany()`.

Topics

- Retrieving all documents in a collection
- Retrieving documents that match a field value
- Retrieving documents that match an embedded document
- Retrieving documents that match a field value in an embedded document
- Retrieving documents that match an array
- Retrieving documents that match a value in an array
- Retrieving documents using operators

Retrieving all documents in a collection

To retrieve all the documents in your collection, use the `find()` operation with an empty query document.

The following query returns all documents in the `example` collection.

```
db.example.find( {} ).pretty()
```

Retrieving documents that match a field value

To retrieve all documents that match a field and value, use the `find()` operation with a query document that identifies the fields and values to match.

Using the preceding documents, this query returns all documents where the "Item" field equals "Pen".

```
db.example.find( { "Item": "Pen" } ).pretty()
```

Retrieving documents that match an embedded document

To find all the documents that match an embedded document, use the `find()` operation with a query document that specifies the embedded document name and all the fields and values for that embedded document.

When matching an embedded document, the document's embedded document must have the same name as in the query. In addition, the fields and values in the embedded document must match the query.

The following query returns only the "Poster Paint" document. This is because the "Pen" has different values for "`OnHand`" and "`MinOnHand`", and "Spray Paint" has one more field (`OrderQnty`) than the query document.

```
db.example.find({"Inventory": {
    "OnHand": 47,
    "MinOnHand": 50 } } ).pretty()
```

Retrieving documents that match a field value in an embedded document

To find all the documents that match an embedded document, use the `find()` operation with a query document that specifies the embedded document name and all the fields and values for that embedded document.

Given the preceding documents, the following query uses "dot notation" to specify the embedded document and fields of interest. Any document that matches these are returned, regardless of what other fields might be present in the embedded document. The query returns "Poster Paint" and "Spray Paint" because they both match the specified fields and values.

```
db.example.find({"Inventory.OnHand": 47, "Inventory.MinOnHand": 50 }).pretty()
```

Retrieving documents that match an array

To find all documents that match an array, use the `find()` operation with the array name that you are interested in and all the values in that array. The query returns all documents that have an array with that name in which the array values are identical to and in the same order as in the query.

The following query returns only the "Pen" because the "Poster Paint" has an additional color (White), and "Spray Paint" has the colors in a different order.

```
db.example.find( { "Colors": ["Red","Green","Blue","Black"] } ).pretty()
```

Retrieving documents that match a value in an array

To find all the documents that have a particular array value, use the `find()` operation with the array name and the value that you're interested in.

```
db.example.find( { "Colors": "Red" } ).pretty()
```

The preceding operation returns all three documents because each of them has an array named `Colors` and the value "Red" somewhere in the array. If you specify the value "`White`," the query would only return "Poster Paint."

Retrieving documents using operators

The following query returns all documents where the "`Inventory.OnHand`" value is less than 50.

```
db.example.find(
        { "Inventory.OnHand": { $lt: 50 } } )
```

For a listing of supported query operators, see Query and projection operators.

Updating documents

Typically, your documents are not static and are updated as part of your application workflows. The following examples show some of the ways that you can update documents.

To update an existing document, use the `update()` operation. The `update()` operation has two document parameters. The first document identifies which document or documents to update. The second document specifies the updates to make.

When you update an existing field — whether that field is a simple field, an array, or an embedded document — you specify the field name and its values. At the end of the operation, it is as though the field in the old document has been replaced by the new field and values.

Topics

- Updating the values of an existing field
- Adding a new field
- Replacing an embedded document
- Inserting new fields into an embedded document
- Removing a field from a document

- Removing a field from multiple documents

Updating the values of an existing field

Use the following four documents that you added earlier for the following updating operations.

```
{
    "Item": "Ruler",
    "Colors": ["Red","Green","Blue","Clear","Yellow"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 40
    },
    "UnitPrice": 0.89
},
{
    "Item": "Pen",
    "Colors": ["Red","Green","Blue","Black"],
    "Inventory": {
        "OnHand": 244,
        "MinOnHand": 72
    }
},
{
    "Item": "Poster Paint",
    "Colors": ["Red","Green","Blue","Black","White"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50
    }
},
{
    "Item": "Spray Paint",
    "Colors": ["Black","Red","Green","Blue"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50,
        "OrderQnty": 36
    }
}
```

To update a simple field

To update a simple field, use `update()` with `$set` to specify the field name and new value. The following example changes the `Item` from "Pen" to "Gel Pen".

```
db.example.update(
    { "Item" : "Pen" },
    { $set: { "Item": "Gel Pen" } }
)
```

Results from this operation look something like the following.

```
{
    "Item": "Gel Pen",
    "Colors": ["Red","Green","Blue","Black"],
    "Inventory": {
        "OnHand": 244,
        "MinOnHand": 72
    }
}
```

To update an array

The following example replaces the existing array of colors with a new array that includes `Orange` and drops `White` from the list of colors. The new list of colors is in the order specified in the `update()` operation.

```
db.example.update(
    { "Item" : "Poster Paint" },
    { $set: { "Colors": ["Red","Green","Blue","Orange","Black"] } }
)
```

Results from this operation look something like the following.

```
{
    "Item": "Poster Paint",
    "Colors": ["Red","Green","Blue","Orange","Black"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50
    }
}
```

Adding a new field

To modify a document by adding one or more new fields, use the `update()` operation with a query document that identifies the document to insert into and the new fields and values to insert using the `$set` operator.

The following example adds the field `UnitPrice` with the value `3.99` to the Spray Paints document. Note that the value `3.99` is numeric and not a string.

```
db.example.update(
    { "Item": "Spray Paint" },
    { $set: { "UnitPrice": 3.99 } }
)
```

Results from this operation look something like the following (JSON format).

```
{
    "Item": "Spray Paint",
    "Colors": ["Black","Red","Green","Blue"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50,
        "OrderQnty": 36
    },
    "UnitPrice": 3.99
}
```

Replacing an embedded document

To modify a document by replacing an embedded document, use the `update()` operation with documents that identify the embedded document and its new fields and values using the `$set` operator.

Given the following document.

```
db.example.insert({
    "DocName": "Document 1",
    "Date": {
        "Year": 1987,
        "Month": 4,
        "Day": 18
```

```
    }
})
```

To replace an embedded document

The following example replaces the current Date document with a new one that has only the fields `Month` and `Day`; `Year` has been eliminated.

```
db.example.update(
    { "DocName" : "Document 1" },
    { $set: { "Date": { "Month": 4, "Day": 18 } } }
)
```

Results from this operation look something like the following.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18
    }
}
```

Inserting new fields into an embedded document

To add fields to an embedded document

To modify a document by adding one or more new fields to an embedded document, use the `update()` operation with documents that identify the embedded document and "dot notation" to specify the embedded document and the new fields and values to insert using the `$set` operator.

Given the following document, the following code uses "dot notation" to insert the `Year` and `DoW` fields to the embedded `Date` document, and `Words` into the parent document.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18
    }
}
```

```
db.example.update(
    { "DocName" : "Document 1" },
    { $set: { "Date.Year": 1987,
              "Date.DoW": "Saturday",
              "Words": 2482 } }
)
```

Results from this operation look something like the following.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
}
```

Removing a field from a document

To modify a document by removing a field from the document, use the `update()` operation with a query document that identifies the document to remove the field from, and the `$unset` operator to specify the field to remove.

The following example removes the `Words` field from the preceding document.

```
db.example.update(
    { "DocName" : "Document 1" },
    { $unset: { Words:1 } }
)
```

Results from this operation look something like the following.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
```

```
        "DoW": "Saturday"
    }
}
```

Removing a field from multiple documents

To modify a document by removing a field from multiple documents, use the `update()` operation with the `$unset` operator and the `multi` option set to `true`.

The following example removes the `Inventory` field from all documents in the example collection. If a document does not have the `Inventory` field, no action is taken on that document. If `multi: true` is omitted, the action is performed only on the first document that meets the criterion.

```
db.example.update(
    {},
    { $unset: { Inventory:1 } },
    { multi: true }
)
```

Deleting documents

To remove a document from your database, use the `remove()` operation, specifying which document to remove. The following code removes "Gel Pen" from your `example` collection.

```
db.example.remove( { "Item": "Gel Pen" } )
```

To remove all documents from your database, use the `remove()` operation with an empty query.

```
db.example.remove( { } )
```

Get started with Amazon DocumentDB

There are many ways to connect and get started with Amazon DocumentDB. This guide is the quickest, simplest, and easiest way for users to get started using our powerful document database. This guide uses AWS CloudShell to connect and query your Amazon DocumentDB cluster directly from the AWS Management Console. New customers who are eligible for the AWS Free Tier can use Amazon DocumentDB and CloudShell for free. If your AWS CloudShell environment or Amazon DocumentDB cluster makes use of resources beyond the free tier, you are charged the normal AWS rates for those resources. This guide will get you started with Amazon DocumentDB in less than five minutes.

> **Note**
>
> The instructions in this guide are specifically for creating and connecting to Amazon DocumentDB instance-based clusters where Amazon DocumentDB and AWS CloudShell are available.
>
> - If you want to create and connect to Amazon DocumentDB elastic clusters, see Get started with Amazon DocumentDB elastic clusters.
> - If you are located in AWS China Regions, see Connect Amazon EC2 automatically.

Topics

- Prerequisites
- Step 1: Create a cluster
- Step 2: Connect to your cluster
- Step 3: Insert and query data
- Step 4: Explore

Prerequisites

Before you create your first Amazon DocumentDB cluster, you must do the following:

Create an Amazon Web Services (AWS) account

Before you can begin using Amazon DocumentDB, you must have an Amazon Web Services (AWS) account. The AWS account is free. You pay only for the services and resources that you use.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

Set up the needed AWS Identity and Access Management (IAM) permissions.

Access to manage Amazon DocumentDB resources such as clusters, instances, and cluster parameter groups requires credentials that AWS can use to authenticate your requests. For more information, see Identity and Access Management for Amazon DocumentDB.

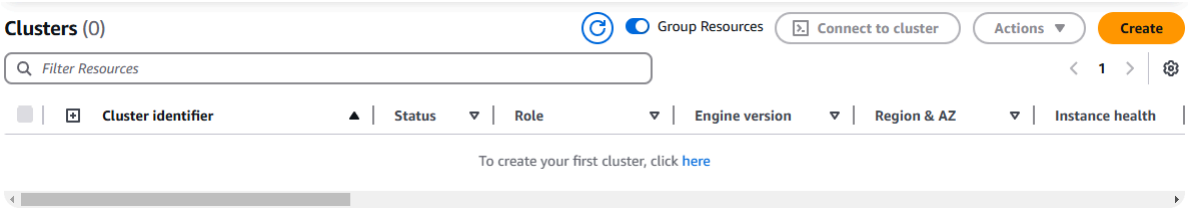
1. In the search bar of the AWS Management Console, type in IAM and select **IAM** in the drop down menu that appears.
2. Once you're in the IAM console, select **Users** from the navigation pane.
3. Select your username.
4. Click **Add permissions**.
5. Select **Attach policies directly**.
6. Type `AmazonDocDBFullAccess` in the search bar and select it once it appears in the search results.
7. Click **Next**.
8. Click **Add permissions**.

> **Note**
>
> Your AWS account includes a default VPC in each Region. If you choose to use an Amazon VPC, complete the steps in the Create an Amazon VPC topic in the *Amazon VPC User Guide*.

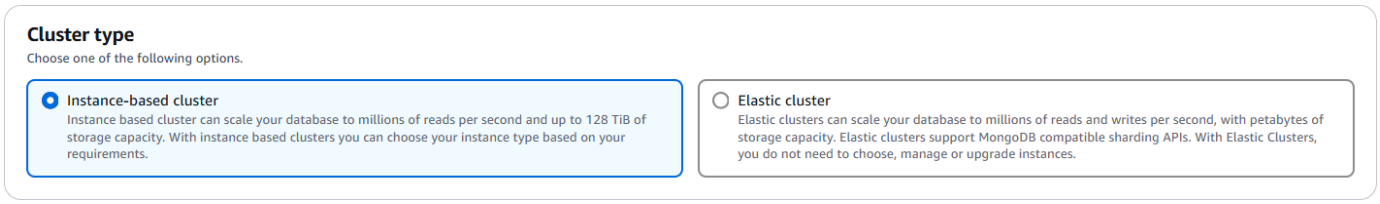
Step 1: Create a cluster

In this step you will create an Amazon DocumentDB cluster.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. On the Amazon DocumentDB management console, under **Clusters**, choose **Create**.

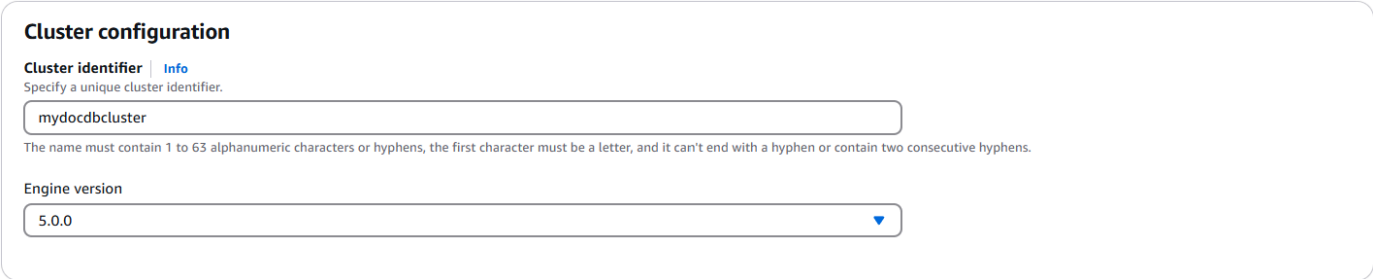


3. On the Create Amazon DocumentDB cluster page, in the **Cluster type** section, choose **Instance-based cluster** (this is the default option).

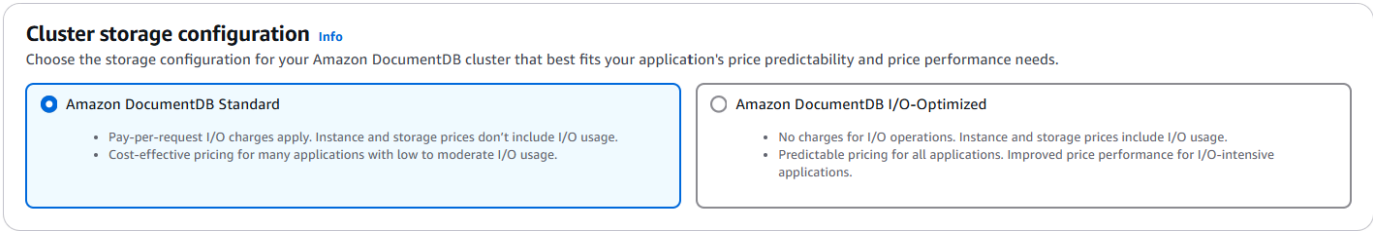


> **Note**
>
> The other option in this category is **Elastic cluster**. To learn more about Amazon DocumentDB elastic clusters, see Using Amazon DocumentDB elastic clusters

4. In the **Cluster configuration** section:
   a. For **Cluster identifier**, enter a unique name, such as **`mydocdbcluster`**. Note that the console will change all cluster names into lower-case regardless of how they are entered.
   b. For **Engine version**, choose **5.0.0**.



5. In the **Cluster storage configuration** section, choose **Amazon DocumentDB Standard** (this is the default option).



> **Note**
>
> The other option in this category is **Amazon DocumentDB I/O-Optimized**. To learn more about either option, see Amazon DocumentDB cluster storage configurations

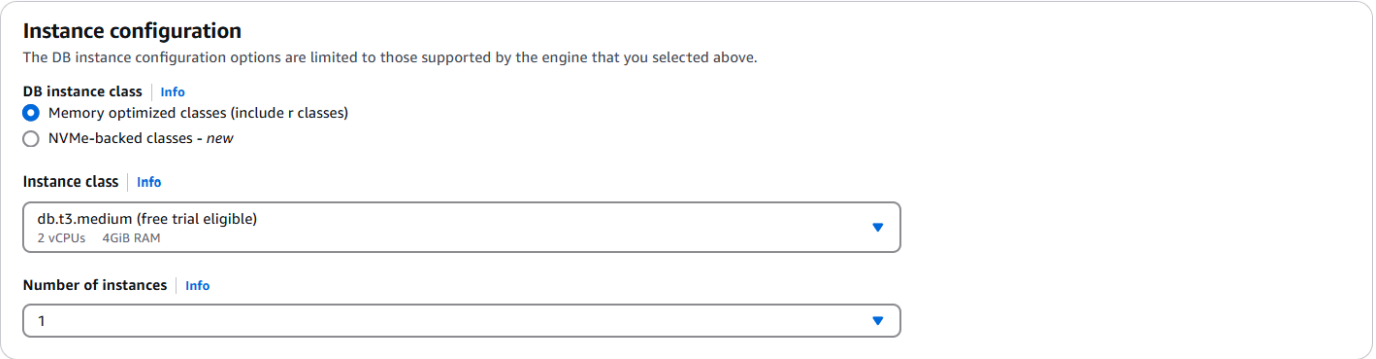
6. In the **Instance configuration** section:

   a. For **DB instance class**, choose **Memory optimized classes (include r classes)** (this is default).

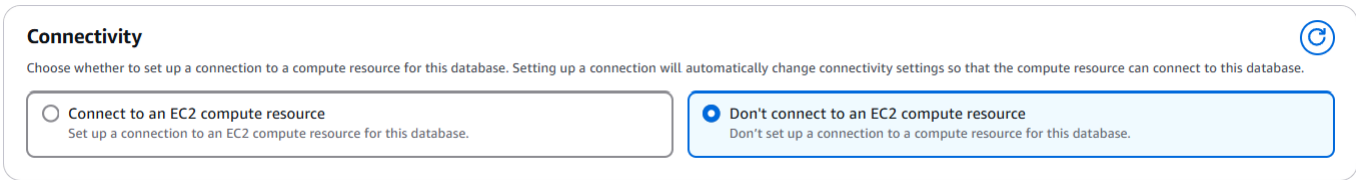
      The other instance option is **NVMe-backed classes**. To learn more, see NVMe-backed instances.

   b. For **Instance class**, choose **db.t3.medium**. This is eligible for the AWS free trial.

   c. For **Number of instances**, choose **1** instance. Choosing one instance helps minimize costs. If this were a production system, we would recommend that you provision three instances for high availability.

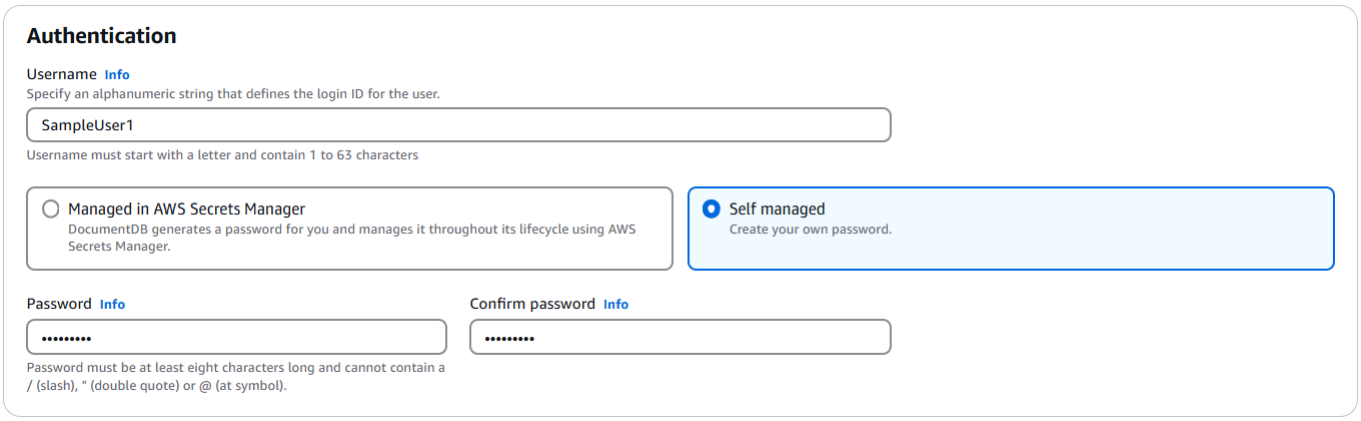


7. In the **Connectivity** section, leave the default setting of **Don't connect to an EC2 compute resource**.



8. In the **Authentication** section, enter a username for the primary user, and then choose **Self managed**. Enter a password, then confirm it.

   If you instead chose **Managed in AWS Secrets Manager**, see [Password management with Amazon DocumentDB and AWS Secrets Manager] for more information.



9. Leave all other options as default and choose **Create cluster**.

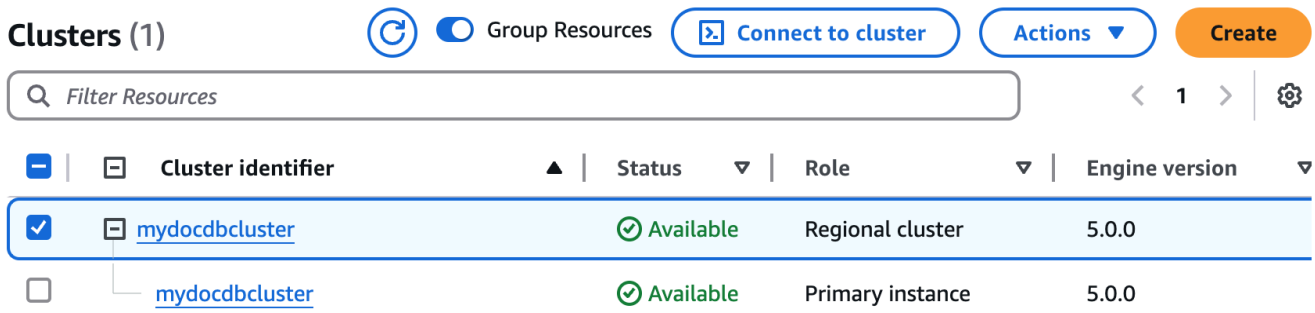
Amazon DocumentDB is now provisioning your cluster, which can take up to a few minutes to finish.

> **Note**
>
> For information about cluster status values, see Cluster status values in the Monitoring Amazon DocumentDB chapter.

Step 2: Connect to your cluster

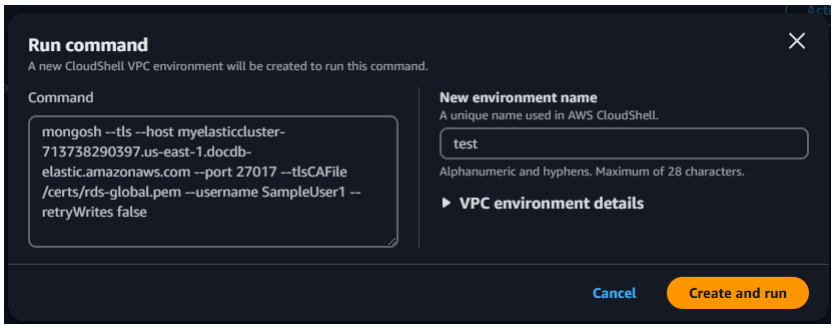
Connect to your Amazon DocumentDB cluster using AWS CloudShell.

1. On the Amazon DocumentDB management console, under **Clusters**, locate the cluster you created. Choose your cluster by clicking the check box next to it.



2. Click **Connect to cluster** (which is next to the **Actions** dropdown menu). This button is enabled only after you have clicked the checkbox next to your cluster, and the status of both the regional cluster and primary instance(s) show as **Available**. The CloudShell **Run command** screen appears.

3. In the **New environment name** field, enter a unique name, such as "test" and click **Create and run**. VPC environment details are automatically configured for your Amazon DocumentDB database.



4. When prompted, enter the password you created in Step 1: Create an Amazon DocumentDB cluster (sub-step 7).

After you enter your password and your prompt becomes `rs0 [direct: primary] <env-name>>`, you are successfully connected to your Amazon DocumentDB cluster.

> **Note**
>
> For information about troubleshooting, see Troubleshooting Amazon DocumentDB.

Step 3: Insert and query data

Now that you are connected to your cluster, you can run a few queries to get familiar with using a document database.

1. To insert a single document, enter the following:

   ```
   db.collection.insertOne({"hello":"DocumentDB"})
   ```

   You get the following output:

   ```
   {
     acknowledged: true,
     insertedId: ObjectId('673657216bdf6258466b128c')
   }
   ```

2. You can read the document that you wrote with the `findOne()` command (because it only returns a single document). Input the following:

   ```
   db.collection.findOne()
   ```

You get the following output:

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. To perform a few more queries, consider a gaming profiles use case. First, insert a few entries into a collection titled `profiles`. Input the following:

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

You get the following output:

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. Use the `find()` command to return all the documents in the profiles collection. Input the following:

```
db.profiles.find()
```

You will get an output that will match the data you typed in Step 3.

5. Use a query for a single document using a filter. Input the following:

```
db.profiles.find({name: "Katie"})
```

You get the following output:

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. Now let's try to find a profile and modify it using the `findAndModify` command. We'll give the user Matt an extra 10 points with the following code:

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
```

```
    })
```

You get the following output (note that his score hasn't increased yet):

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. You can verify that his score has changed with the following query:

   ```
   db.profiles.find({name: "Matt"})
   ```

   You get the following output:

   ```
   { "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
   ```

Step 4: Explore

Congratulations! You have successfully completed the Get started guide for Amazon DocumentDB instance-based clusters.

What's next? Learn how to fully leverage this database with some of its popular features:

- Managing Amazon DocumentDB
- Scaling
- Backing up and restoring

> **Note**
>
> The cluster you created from this get started exercise will continue to accrue costs unless you delete it. For directions, see Deleting an Amazon DocumentDB Cluster.

Amazon DocumentDB quick start using AWS CloudFormation

This section contains steps and other information to help you get started quickly with Amazon DocumentDB (with MongoDB compatibility) using AWS CloudFormation. For general information about Amazon DocumentDB, see What is Amazon DocumentDB (with MongoDB compatibility).

These instructions use an AWS CloudFormation template to create a cluster and instances in your default Amazon VPC. For instructions on creating these resources yourself, see Get started with Amazon DocumentDB.

> **Important**
>
> The AWS CloudFormation stack that is created by this template creates multiple resources, including resources in Amazon DocumentDB (for example, a cluster and instances) and Amazon Elastic Compute Cloud (for example, a subnet group).
> Some of these resources are not free-tier resources. For pricing information, see Amazon DocumentDB Pricing and Amazon EC2 Pricing. You can delete the stack when you are finished with it to stop any charges.

This AWS CloudFormation stack is intended for tutorial purposes only. If you use this template for a production environment, we recommend that you use stricter IAM policies and security. For information about securing resources, see Amazon VPC Security and Amazon EC2 Network and Security.

Topics

- Prerequisites
- Launching an Amazon DocumentDB AWS CloudFormation stack
- Accessing the Amazon DocumentDB cluster
- Termination protection and deletion protection

Prerequisites

Before you create an Amazon DocumentDB cluster, you must have the following:

- A default Amazon VPC
- The required IAM permissions

Required IAM Permissions

The following permissions allow you to create resources for the AWS CloudFormation stack:

AWS Managed Policies

- `AWSCloudFormationReadOnlyAccess`
- `AmazonDocDBFullAccess`

Additional IAM Permissions

The following policy outlines the additional permissions that are required to create and delete this AWS CloudFormation stack.

In the following examples, replace each *`user input placeholder`* with your resource's information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DocDBPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBCluster",
                "rds:DeleteDBCluster",
                "rds:ModifyDBCluster",
                "rds:DescribeDBClusters",
                "rds:CreateDBInstance",
                "rds:DeleteDBInstance",
                "rds:ModifyDBInstance",
                "rds:DescribeDBInstances",
                "rds:CreateDBSubnetGroup",
                "rds:DeleteDBSecurityGroup",
                "rds:DescribeDBSubnetGroups"
            ],
            "Resource": [
                "arn:aws:rds:{AWS_REGION}:{AWS_ACCOUNT_ID}:cluster:*",
```

```
                "arn:aws:rds:{AWS_REGION}:{AWS_ACCOUNT_ID}:db:*",
                "arn:aws:rds:{AWS_REGION}:{AWS_ACCOUNT_ID}:subgrp:*"
            ]
        },
        {
            "Sid": "EC2NetworkingPermissions",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateSecurityGroup",
                "ec2:DeleteSecurityGroup",
                "ec2:DescribeSecurityGroups",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets"
            ],
            "Resource": [
                "arn:aws:ec2:{AWS_REGION}:{AWS_ACCOUNT_ID}:security-group/*",
                "arn:aws:ec2:{AWS_REGION}:{AWS_ACCOUNT_ID}:vpc/*",
                "arn:aws:ec2:{AWS_REGION}:{AWS_ACCOUNT_ID}:subnet/*"
            ]
        },
        {
            "Sid": "EC2DescribePermissions",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAvailabilityZones"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchLogsPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:{AWS_REGION}:{AWS_ACCOUNT_ID}:log-group:/aws/docdb/*",
                "arn:aws:logs:{AWS_REGION}:{AWS_ACCOUNT_ID}:log-group:/aws/docdb/*:log-
stream:*"
```

```
            ]
        },
        {
            "Sid": "KMSPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:CreateKey",
                "kms:DescribeKey",
                "kms:EnableKey",
                "kms:ListKeys",
                "kms:PutKeyPolicy"
            ],
            "Resource": "arn:aws:kms:{AWS_REGION}:{AWS_ACCOUNT_ID}:key/*"
        },
        {
            "Sid": "KMSEncryption",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:{AWS_REGION}:{AWS_ACCOUNT_ID}:key/*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "rds.{AWS_REGION}.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "IAMServiceLinkedRole",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{AWS_ACCOUNT_ID}:role/aws-service-role/
rds.amazonaws.com/AWSServiceRoleForRDS",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
                }
            }
        }
    ]
```

```
}
```

Amazon EC2 Key Pair

You must have a key pair (and the PEM file) available in the Region where you will create the AWS CloudFormation stack. If you need to create a key pair, see Creating a Key Pair Using Amazon EC2 in the *Amazon EC2 User Guide*.

Launching an Amazon DocumentDB AWS CloudFormation stack

This section describes how to launch and configure an Amazon DocumentDB AWS CloudFormation stack.

1. Sign in to the AWS Management Console at https://console.aws.amazon.com/.
2. The following table lists the Amazon DocumentDB stack templates for each AWS Region. Choose **Launch Stack** for the AWS Region you want to launch your stack in.

Region	View Template	View in Designer	Launch
US East (Ohio)	View Template	View in Designer	Launch Stack
US East (N. Virginia)	View Template	View in Designer	Launch Stack
US West (Oregon)	View Template	View in Designer	Launch Stack
Asia Pacific (Mumbai)	View Template	View in Designer	Launch Stack
Asia Pacific (Seoul)	View Template	View in Designer	Launch Stack
Asia Pacific (Singapore)	View Template	View in Designer	Launch Stack
Asia Pacific (Sydney)	View Template	View in Designer	Launch Stack

Region	View Template	View in Designer	Launch
Asia Pacific (Tokyo)	View Template	View in Designer	Launch Stack
Canada (Central)	View Template	View in Designer	Launch Stack
Europe (Frankfurt)	View Template	View in Designer	Launch Stack
Europe (Ireland)	View Template	View in Designer	Launch Stack
Europe (London)	View Template	View in Designer	Launch Stack
Europe (Paris)	View Template	View in Designer	Launch Stack

3. **Create stack** — Describes the Amazon DocumentDB template that you selected. Every stack is based on a template — a JSON or YAML file — that contains configuration about the AWS resources you want to include in the stack. Because you chose to launch a stack from the provided templates above, your template has already been configured to create an Amazon DocumentDB stack for the AWS Region you chose.

   When you launch an AWS CloudFormation stack, deletion protection for your Amazon DocumentDB cluster is disabled by default. If you want to enable deletion protection for your cluster, complete the following steps. Otherwise, choose **Next** to continue to the next step.

   To enable deletion protection for your Amazon DocumentDB cluster:

   1. Choose **View in Designer** from the bottom right corner of the **Create stack** page.
   2. Modify the template using the integrated JSON and YAML editor in the resulting AWS CloudFormation Designer page of the console. Scroll to the `Resources` section and modify it to include `DeletionProtection`, as follows. For more information about using AWS CloudFormation Designer, see What Is AWS CloudFormation Designer?.

      JSON:

```
"Resources": {
    "DBCluster": {
```

```
        "Type": "AWS::DocDB::DBCluster",
        "DeletionPolicy": "Delete",
        "Properties": {
            "DBClusterIdentifier": {
                "Ref": "DBClusterName"
            },
            "MasterUsername": {
                "Ref": "MasterUser"
            },
            "MasterUserPassword": {
                "Ref": "MasterPassword"
            },
            "DeletionProtection": "true"
        }
    },
```

YAML:

```
Resources:
  DBCluster:
    Type: 'AWS::DocDB::DBCluster'
    DeletionPolicy: Delete
    Properties:
      DBClusterIdentifier: !Ref DBClusterName
      MasterUsername: !Ref MasterUser
      MasterUserPassword: !Ref MasterPassword
      DeletionProtection: 'true'
```

3. Choose **Create Stack** ( ) from the top left corner of the page to save your changes and create a stack with these changes enabled.
4. After you save your changes, you will be redirected to the **Create stack** page.
5. Choose **Next** to continue.

4. **Specify stack details** — Enter the stack name and parameters for your template. Parameters are defined in your template and allow you to input custom values when you create or update a stack.
   - Under **Stack name**, enter a name for your stack or accept the provided name. The stack name can include letters (A—Z and a—z), numbers (0—9), and dashes (—).

- Under **Parameters**, enter the following details:
  - **DBClusterName** — Enter a name for your Amazon DocumentDB cluster or accept the provided name.

    Cluster naming constraints:
    - Length is [1—63] letters, numbers, or hyphens.
    - First character must be a letter.
    - Cannot end with a hyphen or contain two consecutive hyphens.
    - Must be unique for all clusters across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.
  - **DBInstanceClass** — From the drop-down list, select the instance class for your Amazon DocumentDB cluster.
  - **DBInstanceName** — Enter a name for your Amazon DocumentDB instance or accept the provided name.

    Instance naming constraints:
    - Length is [1—63] letters, numbers, or hyphens.
    - First character must be a letter.
    - Cannot end with a hyphen or contain two consecutive hyphens.
    - Must be unique for all instances across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.
  - **MasterPassword** — The database admin account password.
  - **MasterUser** — The database admin account username. The MasterUser must begin with a letter and can contain only alphanumeric characters.

  Choose **Next** to save your changes and continue.

5. **Configure stack options** — Configure your stack's tags, permissions, and additional options.
   - **Tags** — Specify tags (key-value) pairs to apply to your resources in your stack. You can add up to 50 unique tags for each stack.
   - **Permissions** — Optional. Choose an IAM role to explicitly define how AWS CloudFormation can create, modify, or delete resources in the stack. If you don't choose a role, AWS CloudFormation uses permissions based on your user credentials. Before you specify a service role, ensure that you have permission to pass it (`iam:PassRole`). The `iam:PassRole` permission specifies which roles you can use.

> **Note**
>
> When you specify a service role, AWS CloudFormation always uses that role for all operations that are performed on that stack. Other users that have permissions to perform operations on this stack will be able to use this role, even if they don't have permission to pass it. If the role includes permissions that the user shouldn't have, you can unintentionally escalate a user's permissions. Ensure that the role grants least privilege.

- **Advanced options** — You can set the following advanced options:
  - **Stack policy** — Optional. Defines the resources that you want to protect from unintentional updates during a stack update. By default, all resources can be updated during a stack update.

    You can enter the stack policy directly as JSON, or upload a JSON file containing the stack policy. For more information, see Prevent Updates to Stack Resources.
  - **Rollback configuration** — Optional. Specify CloudWatch Logs alarms for AWS CloudFormation to monitor when creating and updating the stack. If the operation breaches an alarm threshold, AWS CloudFormation rolls it back.
  - **Notification options** — Optional. Specify topics for Simple Notification System (SNS).
  - **Stack creation options** — Optional. You can specify the following options:
    - **Rollback on failure** — Whether or not the stack should be rolled back if the stack creation fails.
    - **Timeout** —The number of minutes before a stack creation times out.
    - **Termination protection** — Prevents the stack from being accidentally deleted.

    > **Note**
    >
    > AWS CloudFormation termination protection is different from the Amazon DocumentDB concept of deletion protection. For more information, see Termination protection and deletion protection.

Choose **Next** to continue.

6. **Review <stack-name>** — Review your stack template, details, and configuration options. You can also open a **quick-create link** at the bottom of the page to create stacks with the same basic configurations as this one.

   - Choose **Create** to create the stack.
   - Alternatively, you can choose **Create change set**. A change set is a preview of how this stack will be configured before creating the stack. This allows you to examine various configurations before executing the change set.

Accessing the Amazon DocumentDB cluster

Once the AWS CloudFormation stack has been completed, you can use an Amazon EC2 instance to connect to your Amazon DocumentDB cluster. For information about connecting to an Amazon EC2 instance using SSH, see Connect to Your Linux Instance in the *Amazon EC2 User Guide*.

After you are connected, see the following sections, which contain information about using Amazon DocumentDB.

- Step 3: Insert and query data
- Managing Amazon DocumentDB resources
- Monitoring Amazon DocumentDB

Termination protection and deletion protection

It is an Amazon DocumentDB best practice to enable deletion protection and termination protection. CloudFormation termination protection is a distinctly different feature from the Amazon DocumentDB deletion protection feature.

- **Termination protection** — You can prevent a stack from being accidentally deleted by enabling termination protection for your CloudFormation stack. If a user attempts to delete a stack with termination protection enabled on it, the deletion fails and the stack remains unchanged. Termination protection is disabled by default when you create a stack using CloudFormation. You can enable termination protection on a stack when you create it. For more information, see Setting AWS CloudFormation Stack Options.
- **Deletion protection** — Amazon DocumentDB also provides the ability to enable deletion protection for a cluster. If a user attempts to delete an Amazon DocumentDB cluster with

deletion protection enabled on it, the deletion fails and the cluster remains unchanged. Deletion protection, when enabled, safeguards against accidental deletes from the Amazon DocumentDB AWS Management Console, AWS CLI, and CloudFormation. For more information on enabling and disabling deletion protection for an Amazon DocumentDB cluster, see Deletion protection.

Amazon DocumentDB compatibility with MongoDB

Amazon DocumentDB supports MongoDB compatibility including MongoDB 4.0 and MongoDB 5.0. MongoDB compatibility means that a vast majority of the applications, drivers, and tools you already use today with your MongoDB databases can be used with Amazon DocumentDB with little or no change. This section describes everything you need to know about Amazon DocumentDB compatibility with MongoDB including new capabilities and features, getting started, migrations paths, and functional differences.

Topics

- MongoDB 5.0 compatibility
- MongoDB 4.0 compatibility

MongoDB 5.0 compatibility

Topics

- What's new in Amazon DocumentDB 5.0
- Get started with Amazon DocumentDB 5.0
- Upgrade or migrate to Amazon DocumentDB 5.0
- Functional differences

What's new in Amazon DocumentDB 5.0

Amazon DocumentDB 5.0 introduces new features and capabilities that include storage limits and client-side field level encryption. The summary below introduces some of major features that were introduced in Amazon DocumentDB 5.0. To see a full list of the new capabilities, see the Release notes.

- Increased storage limit to 128 TiB for all instance-based Amazon DocumentDB clusters and shard-based elastic clusters.
- Introduced Amazon DocumentDB 5.0 Engine Version 3.0.775)
  - Support for MongoDB 5.0 API drivers

- Support for Client-side Field Level Encryption (FLE). You can now encrypt fields at the client-side before writing the data to Amazon DocumentDB cluster. For more information, see Client-side field level encryption.
- New aggregation operators: `$dateAdd`, `$dateSubtract`
- Supports for indexes with `$elemMatch` operator. As a result, queries having `$elemMatch` will result in index scans.

Amazon DocumentDB does not support every MongoDB 5.0 feature. When we built Amazon DocumentDB 5.0, we worked backwards from the feature and capabilities that our customers asked us to build the most. We will continue to add additional MongoDB 5.0 capabilities based on what customers ask us to build. For the latest list of supported APIs, please see Supported MongoDB APIs, operations, and data types in Amazon DocumentDB.

Get started with Amazon DocumentDB 5.0

To get started with Amazon DocumentDB 5.0, please see the Get Started Guide. You can create a new Amazon DocumentDB 5.0 cluster using the AWS Management Console or the AWS SDK, AWS CLI, or AWS CloudFormation. When connecting to Amazon DocumentDB, it is required that you use a MongoDB driver or utility that is compatible with MongoDB 5.0 or higher.

> **Note**
>
> When using the AWS SDK, AWS CLI, or AWS CloudFormation, the engine version will default to 5.0.0. You must explicitly specify the parameter `engineVersion = 4.0.0` to create a new Amazon DocumentDB 4.0 cluster or `engineVersion = 3.6.0` to create a new Amazon DocumentDB 3.6 cluster. For a given Amazon DocumentDB cluster, you can determine the cluster version using the AWS CLI to call `describe-db-clusters` or use the Amazon DocumentDB management console to view the engine version number for a particular cluster.

Amazon DocumentDB 5.0 supports Amazon EC2 Graviton2 processors such as `r6g` and `t4.medium` instance types for your clusters and is available in all supported regions. For more information on pricing, see Amazon DocumentDB (with MongoDB compatibility) Pricing.

Upgrade or migrate to Amazon DocumentDB 5.0

You can migrate from MongoDB 3.6 or MongoDB 4.0 to Amazon DocumentDB 5.0 using the AWS DMS or utilities like `mongodump`, `mongorestore`, `mongoimport`, and `mongoexport`. For instructions on how to migrate, see Upgrading your Amazon DocumentDB cluster using AWS Database Migration Service.

Functional differences

Functional differences between Amazon DocumentDB 4.0 and 5.0

With the release of Amazon DocumentDB 5.0, there are functional differences between Amazon DocumentDB 4.0 and Amazon DocumentDB 5.0:

- The backup built-in role now supports `serverStatus`. Action - Developers and applications with backup role can collect statistics about the state of the Amazon DocumentDB cluster.
- The `SecondaryDelaySecs` field replaces `slaveDelay` in `replSetGetConfig` output.
- The **hello** command replaces `isMaster` - **hello** returns a document that describes the role of a Amazon DocumentDB cluster.
- Amazon DocumentDB 5.0 now supports index scans with the `$elemMatch` operator in the first nesting level. Index scans are supported when the query only filter has one level of the `$elemMatch` filter but are not supported if a nested `$elemMatch` query is included.

  For example, in Amazon DocumentDB 5.0, if you include the `$elemMatch` operator in the nested level, it will not return a value as it does in Amazon DocumentDB 4.0:

```
db.foo.insert(
[
    {a: {b: 5}},
    {a: {b: [5]}},
    {a: {b: [3, 7]}},
    {a: [{b: 5}]},
    {a: [{b: 3}, {b: 7}]},
    {a: [{b: [5]}]},
    {a: [{b: [3, 7]}]},
    {a: [[{b: 5}]]},
    {a: [[{b: 3}, {b: 7}]]},
    {a: [[{b: [5]}]]},
    {a: [[{b: [3, 7]}]]}
```

```
]);

// DocumentDB 5.0
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }

// DocumentDB 4.0
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
{ "a" : [ [ { "b" : [ 5 ] } ] ] }
```

- The "$" projection in Amazon DocumentDB 4.0 returns all documents with all fields. With Amazon DocumentDB 5.0, the **find** command with a "$" projection returns documents that match the query parameter containing only the field that matched the "$" projection.
- In Amazon DocumentDB 5.0, the **find** commands with `$regex` and `$options` query parameters return an error: "Cannot set options in both `$regex` and `$options`".
- With Amazon DocumentDB 5.0, `$indexOfCP` now returns "-1" when:
  - the substring is not found in the string expression, or
  - start is a number greater than end, or
  - start is a number greater than the byte length of the string.
- In Amazon DocumentDB 4.0, `$indexOfCP` returns "0" when the start position is a number greater than end or the byte length of the string.
- With Amazon DocumentDB 5.0, projection operations in `_id fields`, for example `{"_id.nestedField" : 1}`, return documents that only include the projected field. Whereas in Amazon DocumentDB 4.0, nested field projection commands do not filter out any document.

MongoDB 4.0 compatibility

Topics

- Amazon DocumentDB 4.0 features
- Get started with Amazon DocumentDB 4.0
- Upgrade or migrate to Amazon DocumentDB 4.0
- Functional differences

Amazon DocumentDB 4.0 features

Amazon DocumentDB 4.0 introduced many new features and capabilities that included ACID transactions and improvements to change streams. The summary below shows some of the major features that were introduced in Amazon DocumentDB 4.0. To see a full list of the capabilities, see the Release notes.

- **ACID Transactions**: Amazon DocumentDB now supports the ability to perform transactions across multiple documents, statements, collections, and databases. Transactions simplify application development by enabling you to perform atomic, consistent, isolated, and durable (ACID) operations across one or more documents within an Amazon DocumentDB cluster. For more information, see Transactions in Amazon DocumentDB.
- **Change streams**: You now have the ability to open a change stream at the cluster level (`client.watch()` or `mongo.watch()`) and the database (`db.watch()`), you can specify a `startAtOperationTime` to open a change stream cursor, and lastly you can now extend your change stream retention period to 7 days (previously 24 hours). For more information, see Using change streams with Amazon DocumentDB.
- **AWS Database Migration Service** (AWS DMS): You can now use AWS DMS to migrate your MongoDB 4.0 workloads to Amazon DocumentDB. AWS DMS now supports a MongoDB 4.0 source, Amazon DocumentDB 4.0 target, and an Amazon DocumentDB 3.6 source for performing upgrades between Amazon DocumentDB 3.6 and 4.0. For more information, see AWS DMS Documentation.
- **Performance and indexing**: You can now utilize an index with `$lookup`, find queries with a projection that contain one field or one field and the `_id` field can be served direct from the index and without needing to read from the collection (covered query), the ability to `hint()` with `findAndModify`, performance optimizations for `$addToSet`, and improvements to reduce overall index sizes. For more information, see Release notes.
- **Operators**: Amazon DocumentDB 4.0 now supports a number of new aggregation operators: `$ifNull`, `$replaceRoot`, `$setIsSubset`, `$setIntersection`, `$setUnion`, `$setEquals`. You can see all the MongoDB APIs, Operations, and Data Types that we support at Supported MongoDB APIs, operations, and data types in Amazon DocumentDB.
- **Role based access control** (RBAC): With both `ListCollection` and `ListDatabase` commands you can now optionally use the `authorizedCollections` and `authorizedDatabases` parameters to allow users to list the collections and databases that they have permission to access without requiring the `listCollections` and `listDatabase` roles, respectively. You also have the ability to kill your own cursors without requiring the `KillCursor` role.

Amazon DocumentDB does not support every MongoDB 4.0 feature. When we built Amazon DocumentDB 4.0, we worked backwards from the feature and capabilities that our customers asked us to build the most. We will continue to add additional MongoDB 4.0 capabilities based on what customers ask us to build. For example, Amazon DocumentDB 4.0 does not currently support the type conversion operators or the string operators that were introduced in MongoDB 4.0. For the latest list of supported APIs, please see Supported MongoDB APIs, operations, and data types in Amazon DocumentDB.

Get started with Amazon DocumentDB 4.0

To get started with Amazon DocumentDB 4.0, please see the Get Started Guide. You can create a new Amazon DocumentDB 4.0 cluster using the AWS Management Console or the AWS SDK, AWS CLI, or AWS CloudFormation. When connecting to Amazon DocumentDB, it is required that you use a MongoDB driver or utility that is compatible with MongoDB 4.0 or higher.

> **Note**
>
> When using the AWS SDK, AWS CLI, or AWS CloudFormation, the engine version will default to 5.0.0. You must explicitly specify the parameter `engineVersion = 4.0.0` to create a new Amazon DocumentDB 4.0 cluster or `engineVersion = 3.6.0` to create a new Amazon DocumentDB 3.6 cluster. For a given Amazon DocumentDB cluster, you can determine the cluster version using the AWS CLI to call `describe-db-clusters` or use the Amazon DocumentDB management console to view the engine version number for a particular cluster.

Amazon DocumentDB 4.0 supports `r5`, `r6g`, `t3.medium`, and `t4g.medium` instance types for your clusters and is available in all supported regions. There is no additional cost for using Amazon DocumentDB 4.0. For more information on pricing, see Amazon DocumentDB (with MongoDB compatibility) Pricing.

Upgrade or migrate to Amazon DocumentDB 4.0

You can migrate from MongoDB 3.6 or MongoDB 4.0 to Amazon DocumentDB 4.0 utilizing the AWS DMS or utilities like `mongodump`, `mongorestore`, `mongoimport`, and `mongoexport`. Similarly, you can use the same tools to upgrade from Amazon DocumentDB 3.6 to Amazon DocumentDB 4.0. For instructions on how to migrate, see Upgrading your Amazon DocumentDB cluster using AWS Database Migration Service.

Functional differences

Functional differences between Amazon DocumentDB 3.6 and 4.0

With the release of Amazon DocumentDB 4.0, there are functional differences between Amazon DocumentDB 3.6 and Amazon DocumentDB 4.0:

- **Projection for nested documents**: Amazon DocumentDB 3.6 considers the first field in a nested document when applying a projection. However, Amazon DocumentDB 4.0 will parse subdocuments and apply the projection to each sub document as well. For example: if the projection is `"a.b.c": 1`, then the behavior in both versions is identical. However, if the projection is `{a:{b:{c:1}}}` then Amazon DocumentDB 3.6 will only apply the projection to 'a' and not 'b' or 'c'.
- **Behavior for `minKey`, `maxKey`**: In Amazon DocumentDB 4.0, the behavior for `{x:{$gt:MaxKey}}` returns nothing, and for `{x:{$lt:MaxKey}}` returns everything.
- **Document comparison differences**: Comparing numerical values of different types (double, int, long) in subdocuments (e.g., b in `{"_id" :1, "a" :{"b":1}}`) now provides a consistent output across numerical data types and for each level of a document.

Functional differences between Amazon DocumentDB 4.0 and MongoDB 4.0

Below are functional differences between Amazon DocumentDB 4.0 and MongoDB 4.0.

- **Lookup with empty key in path**: When a collection contains a document with empty key inside the array (e.g. `{"x" : [ { "" : 10 }, { "b" : 20 } ]}`), and when the key used in the query ends in an empty string (e.g. `x.`), then Amazon DocumentDB will return that document since it traverses all the documents in the array whereas MongoDB will not return that document.
- **`$setOnInsert` along with $ in the path**: The field operator `$setOnInsert` will not work in combination with $ in the path in Amazon DocumentDB, which is also consistent with MongoDB 4.0.

Transactions in Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) now supports MongoDB 4.0 compatibility including transactions. You can perform transactions across multiple documents, statements, collections, and databases. Transactions simplify application development by enabling you to perform atomic, consistent, isolated, and durable (ACID) operations across one or more documents within an Amazon DocumentDB cluster. Common use cases for transactions include financial processing, fulfilling and managing orders, and building multi-player games.

There is no additional cost for transactions. You only pay for the read and write IOs that you consume as part of the transactions.

Topics

- Requirements
- Best practices
- Limitations
- Monitoring and diagnostics
- Transaction isolation level
- Use cases
- Supported commands
- Unsupported capabilities
- Sessions
- Transaction errors

Requirements

To use the transactions feature, you need to meet the following requirements:

- You must be using the Amazon DocumentDB 4.0 engine.
- You must use a driver compatible with MongoDB 4.0 or greater.

Best practices

Here are some best practices so that you can get the most using transactions with Amazon DocumentDB.

- Always commit or abort the transaction after it is complete. Leaving a transaction in an incomplete state ties up database resources and can cause write conflicts.
- It is recommended to keep transactions to the smallest number of commands needed. If you have transactions with multiple statements that can be divided up into multiple smaller transactions, it is advisable to do so to reduce the likelihood of a timeout. Always aim to create short transactions, not long-running reads.

Limitations

- Amazon DocumentDB does not support cursors within a transaction.
- Amazon DocumentDB cannot create new collections in a transaction and cannot query/update against non-existing collections.
- Document-level write locks are subject to a 1 minute timeout, which is not configurable by the user.
- Retryable writes, retryable commit, and retryable abort commands are not supported in Amazon DocumentDB. If you are using legacy mongo shell (not mongosh), do not include the `retryWrites=false` command in any code string. By default, retryable writes are disabled. Including `retryWrites=false` might cause a failure in normal read commands.
- Each Amazon DocumentDB instance has an upper bound limit on the number of concurrent transaction open on the instance at one time. For the limits, please see Instance limits.
- For a given transaction, the transaction log size must be less than 32MB.
- Amazon DocumentDB does support `count()` within a transactions, but not all drivers support this capability. An alternative is to use the `countDocuments()` API, which translates the count query into an aggregation query on the client side.
- Transactions have a one minute execution limit and sessions have a 30-minute timeout. If a transaction times out, it will be aborted, and any subsequent commands issued within the session for the existing transaction will yield the following error:

```
WriteCommandError({
"ok" : 0,
```

```
"operationTime" : Timestamp(1603491424, 627726),
"code" : 251,
"errmsg" : "Given transaction number 0 does not match any in-progress transactions."
}
```

Monitoring and diagnostics

With the support for transactions in Amazon DocumentDB 4.0, additional CloudWatch metrics were added to help you monitor your transactions.

New CloudWatch Metrics

- `DatabaseTransactions`: The number of open transactions taken at a one-minute period.
- `DatabaseTransactionsAborted`: The number of aborted transactions taken at a one-minute period.
- `DatabaseTransactionsMax`: The maximum number of open transactions in a one-minute period.
- `TransactionsAborted`: The number of transactions aborted on an instance in a one-minute period.
- `TransactionsCommitted`: The number of transactions committed on an instance in a one-minute period.
- `TransactionsOpen`: The number of transactions open on an instance taken at a one-minute period.
- `TransactionsOpenMax`: The maximum number of transactions open on an instance in a one-minute period.
- `TransactionsStarted`: The number of transactions started on an instance in a one-minute period.

> **Note**
>
> For more CloudWatch metrics for Amazon DocumentDB, go to Monitoring Amazon DocumentDB with CloudWatch.

Additionally, new fields were added to both `currentOp` `lsid`, `transactionThreadId`, and a new state for "`idle transaction`" and `serverStatus` transactions: `currentActive`, `currentInactive`, `currentOpen`, `totalAborted`, `totalCommitted`, and `totalStarted`.

Transaction isolation level

When starting a transaction, you have the ability to specify both the `readConcern` and `writeConcern` as shown in the example below:

```
mySession.startTransaction({readConcern: {level: 'snapshot'}, writeConcern:
{w: 'majority'}});
```

For `readConcern`, Amazon DocumentDB supports snapshot isolation by default. If a `readConcern` of local, available, or majority are specified, Amazon DocumentDB will upgrade the `readConcern` level to snapshot. Amazon DocumentDB does not support the linearizable `readConcern` and specifying such a read concern will result in an error.

For `writeConcern`, Amazon DocumentDB supports majority by default and a write quorum is achieved when four copies of the data are persisted across three AZs. If a lower `writeConcern` is specified, Amazon DocumentDB will upgrade the `writeConcern` to majority. Further, all Amazon DocumentDB writes are journaled and journaling cannot be disabled.

Use cases

In this section, we will walk through two use cases for transactions: multi-statement and multi-collection.

Multi-Statement Transactions

Amazon DocumentDB transactions are multi-statement, which means you can write a transaction that spans multiple statements with an explicit commit or rollback. You can group `insert`, `update`, `delete`, and `findAndModify` actions as a single atomic operation.

A common use case for multi-statement transactions is a debit-credit transaction. For example: you owe a friend money for clothes. Thus, you need to debit (withdraw) $500 from your account and credit $500 (deposit) to your friend's account. To perform that operation, you perform both the debit and credit operations within a single transaction to ensure atomicity. Doing so prevents

scenarios where $500 is debited from your account, but not credited to your friend's account. Here's what this use case would look like:

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success Scenario***
// Setup bank account for Alice and Bob. Each have $1000 in their account

var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;

session.commitTransaction();

accountColl.find();

// *** Transfer $500 from Alice to Bob inside a transaction: Failure Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;
```

```
var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;

session.abortTransaction();
```

Multi-collection transactions

Our transactions are also multi-collection, which means they can be used to perform multiple operations within a single transaction and across multiple collections. This provides a consistent view of data and maintains your data's integrity. When you commit the commands as a single <>, the transactions are all-or-nothing executions—in that, they will either all succeed or all fail.

Here is an example of multi-collection transactions, using the same scenario and data from the example for multi-statement transactions.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var amountToTransfer = 500;
var collectionName = "account";

var session = db.getMongo().startSession({causalConsistency: false});
var accountCollInBankA = session.getDatabase("bankA")[collectionName];
var accountCollInBankB = session.getDatabase("bankB")[collectionName];

accountCollInBankA.drop();
```

```
accountCollInBankB.drop();

accountCollInBankA.insert({name: "Alice", balance: 1000});
accountCollInBankB.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountCollInBankA.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountCollInBankB.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;

session.commitTransaction();

accountCollInBankA.find(); // Alice holds $500 in bankA
accountCollInBankB.find(); // Bob holds $1500 in bankB

// *** Transfer $500 from Alice to Bob inside a transaction: Failure Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var accountCollInBankA = session.getDatabase("bankA")[collectionName];
var accountCollInBankB = session.getDatabase("bankB")[collectionName];

accountCollInBankA.drop();
accountCollInBankB.drop();

accountCollInBankA.insert({name: "Alice", balance: 1000});
accountCollInBankB.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;
```

```
var newAliceBalance = aliceBalance - amountToTransfer;
accountCollInBankA.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountCollInBankB.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;

session.abortTransaction();

accountCollInBankA.find(); // Alice holds $1000 in bankA
accountCollInBankB.find(); // Bob holds $1000 in bankB
```

Transaction API examples for callback API

The callback API is only available for 4.2+ drivers.

Javascript

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Javascript.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success ***
// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
```

```
assert(aliceBalance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert.eq(newAliceBalance, findAliceBalance);

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
assert.eq(newBobBalance, findBobBalance);

session.commitTransaction();

accountColl.find();
```

Node.js

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Node.js.

```
// Node.js callback API:

const bankDB = await mongoclient.db("bank");
var accountColl = await bankDB.createCollection("account");
var amountToTransfer = 500;

const session = mongoclient.startSession({causalConsistency: false});
await accountColl.drop();

await accountColl.insertOne({name: "Alice", balance: 1000}, { session });
await accountColl.insertOne({name: "Bob", balance: 1000}, { session });

const transactionOptions = {
    readConcern: { level: 'snapshot' },
    writeConcern: { w: 'majority' }
    };

// deduct $500 from Alice's account
var aliceBalance = await accountColl.findOne({name: "Alice"}, {session});
assert(aliceBalance.balance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
```

```
session.startTransaction(transactionOptions);
await accountColl.updateOne({name: "Alice"}, {$set: {balance: newAliceBalance}},
 {session });
await session.commitTransaction();
aliceBalance = await accountColl.findOne({name: "Alice"}, {session});
assert(newAliceBalance == aliceBalance.balance);

// add $500 to Bob's account
var bobBalance = await accountColl.findOne({name: "Bob"}, {session});
var newBobBalance = bobBalance.balance + amountToTransfer;
session.startTransaction(transactionOptions);
await accountColl.updateOne({name: "Bob"}, {$set: {balance: newBobBalance}},
 {session });
await session.commitTransaction();
bobBalance = await accountColl.findOne({name: "Bob"}, {session});
assert(newBobBalance == bobBalance.balance);
```

C#

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with C#.

```
// C# Callback API

var dbName = "bank";
var collName = "account";
var amountToTransfer = 500;

using (var session = client.StartSession(new ClientSessionOptions{CausalConsistency
 = false}))
{
    var bankDB = client.GetDatabase(dbName);
    var accountColl = bankDB.GetCollection<BsonDocument>(collName);
    bankDB.DropCollection(collName);
    accountColl.InsertOne(session, new BsonDocument { {"name", "Alice"}, {"balance",
 1000 } });
    accountColl.InsertOne(session, new BsonDocument { {"name", "Bob"}, {"balance",
 1000 } });

    // start transaction
    var transactionOptions = new TransactionOptions(
            readConcern: ReadConcern.Snapshot,
            writeConcern: WriteConcern.WMajority);
```

```
    var result = session.WithTransaction(
        (sess, cancellationtoken) =>
        {
            // deduct $500 from Alice's account
            var aliceBalance = accountColl.Find(sess,
 Builders<BsonDocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
            Debug.Assert(aliceBalance >= amountToTransfer);
            var newAliceBalance = aliceBalance.AsInt32 - amountToTransfer;
            accountColl.UpdateOne(sess, Builders<BsonDocument>.Filter.Eq("name",
 "Alice"),
                                  Builders<BsonDocument>.Update.Set("balance",
 newAliceBalance));
            aliceBalance = accountColl.Find(sess,
 Builders<BsonDocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
            Debug.Assert(aliceBalance == newAliceBalance);

            // add $500 from Bob's account
            var bobBalance = accountColl.Find(sess,
 Builders<BsonDocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
            var newBobBalance = bobBalance.AsInt32 + amountToTransfer;
            accountColl.UpdateOne(sess, Builders<BsonDocument>.Filter.Eq("name",
 "Bob"),
                                  Builders<BsonDocument>.Update.Set("balance",
 newBobBalance));
            bobBalance = accountColl.Find(sess,
 Builders<BsonDocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
            Debug.Assert(bobBalance == newBobBalance);

            return "Transaction committed";
        }, transactionOptions);
   // check values outside of transaction
    var aliceNewBalance = accountColl.Find(Builders<BsonDocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
    var bobNewBalance = accountColl.Find(Builders<BsonDocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
    Debug.Assert(aliceNewBalance ==  500);
    Debug.Assert(bobNewBalance ==  1500);
}
```

Ruby

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Ruby.

```
// Ruby Callback API

dbName = "bank"
collName = "account"
amountToTransfer = 500

session = client.start_session(:causal_consistency=> false)
bankDB = Mongo::Database.new(client, dbName)
accountColl = bankDB[collName]
accountColl.drop()

accountColl.insert_one({"name"=>"Alice", "balance"=>1000})
accountColl.insert_one({"name"=>"Bob", "balance"=>1000})

    # start transaction
    session.with_transaction(read_concern: {level: :snapshot}, write_concern:
 {w: :majority}) do
        # deduct $500 from Alice's account
        aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
        assert aliceBalance >= amountToTransfer
        newAliceBalance = aliceBalance - amountToTransfer
        accountColl.update_one({"name"=>"Alice"}, { "$set" =>
 {"balance"=>newAliceBalance} }, :session=> session)
        aliceBalance = accountColl.find({"name"=>>"Alice"}, :session=>
 session).first['balance']
        assert_equal(newAliceBalance, aliceBalance)

        # add $500 from Bob's account
        bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
        newBobBalance = bobBalance + amountToTransfer
        accountColl.update_one({"name"=>"Bob"}, { "$set" =>
 {"balance"=>newBobBalance} }, :session=> session)
        bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
        assert_equal(newBobBalance, bobBalance)
    end
```

```
   # check results outside of transaction
    aliceBalance = accountColl.find({"name"=>"Alice"}).first['balance']
    bobBalance = accountColl.find({"name"=>"Bob"}).first['balance']
    assert_equal(aliceBalance, 500)
    assert_equal(bobBalance, 1500)

session.end_session
```

Go

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Go.

```
// Go - Callback API
type Account struct {
    Name string
    Balance  int
}

ctx := context.TODO()

dbName := "bank"
collName := "account"
amountToTransfer := 500

session, err := client.StartSession(options.Session().SetCausalConsistency(false))
assert.NilError(t, err)
defer session.EndSession(ctx)

bankDB := client.Database(dbName)
accountColl := bankDB.Collection(collName)
accountColl.Drop(ctx)

_, err = accountColl.InsertOne(ctx, bson.M{"name" : "Alice", "balance":1000})
_, err = accountColl.InsertOne(ctx, bson.M{"name" : "Bob", "balance":1000})

transactionOptions := options.Transaction().SetReadConcern(readconcern.Snapshot()).

 SetWriteConcern(writeconcern.New(writeconcern.WMajority()))
_, err = session.WithTransaction(ctx, func(sessionCtx mongo.SessionContext)
 (interface{}, error) {
    var result Account
    // deduct $500 from Alice's account
```

```
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    newAliceBalance := aliceBalance - amountToTransfer
    _, err = accountColl.UpdateOne(sessionCtx, bson.M{"name": "Alice"},
 bson.M{"$set": bson.M{"balance": newAliceBalance}})
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance = result.Balance
    assert.Equal(t, aliceBalance, newAliceBalance)

    // add $500 to Bob's account
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    newBobBalance := bobBalance + amountToTransfer
    _, err = accountColl.UpdateOne(sessionCtx, bson.M{"name": "Bob"}, bson.M{"$set":
 bson.M{"balance": newBobBalance}})
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance = result.Balance
    assert.Equal(t, bobBalance, newBobBalance)

    if err != nil {
        return nil, err
    }
    return "transaction committed", err
}, transactionOptions)

// check results outside of transaction
var result Account
err = accountColl.FindOne(ctx, bson.M{"name": "Alice"}).Decode(&result)
aliceNewBalance := result.Balance
err = accountColl.FindOne(ctx, bson.M{"name": "Bob"}).Decode(&result)
bobNewBalance := result.Balance
assert.Equal(t, aliceNewBalance, 500)
assert.Equal(t, bobNewBalance, 1500)
```

Java

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Java.

```
// Java (sync) - Callback API
MongoDatabase bankDB = mongoClient.getDatabase("bank");
MongoCollection accountColl = bankDB.getCollection("account");
accountColl.drop();
```

```
int amountToTransfer = 500;

// add sample data
accountColl.insertOne(new Document("name", "Alice").append("balance", 1000));
accountColl.insertOne(new Document("name", "Bob").append("balance", 1000));

TransactionOptions txnOptions = TransactionOptions.builder()
        .readConcern(ReadConcern.SNAPSHOT)
        .writeConcern(WriteConcern.MAJORITY)
        .build();
ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
try ( ClientSession clientSession = mongoClient.startSession(sessionOptions) ) {
    clientSession.withTransaction(new TransactionBody<Void>() {
        @Override
        public Void execute() {
            // deduct $500 from Alice's account
            List<Document> documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
            int aliceBalance = (int) documentList.get(0).get("balance");
            int newAliceBalance = aliceBalance - amountToTransfer;

            accountColl.updateOne(clientSession, new Document("name", "Alice"), new
 Document("$set", new Document("balance", newAliceBalance)));

            // check Alice's new balance
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
            int updatedBalance = (int) documentList.get(0).get("balance");
            Assert.assertEquals(updatedBalance, newAliceBalance);

            // add $500 to Bob's account
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
            int bobBalance = (int) documentList.get(0).get("balance");
            int newBobBalance = bobBalance + amountToTransfer;

            accountColl.updateOne(clientSession, new Document("name", "Bob"), new
 Document("$set", new Document("balance", newBobBalance)));

            // check Bob's new balance
```

```
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
            updatedBalance = (int) documentList.get(0).get("balance");
            Assert.assertEquals(updatedBalance, newBobBalance);

            return null;
        }
    }, txnOptions);
}
```

C

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with C.

```
// Sample Code for C with Callback

#include <bson.h>
#include <mongoc.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

typedef struct {
    int64_t balance;
    bson_t *account;
    bson_t *opts;
    mongoc_collection_t *collection;
} ctx_t;

bool callback_session (mongoc_client_session_t *session, void *ctx, bson_t **reply,
 bson_error_t *error)
{
    bool r = true;
    ctx_t *data = (ctx_t *) ctx;
    bson_t local_reply;
    bson_t *selector = data->account;
    bson_t *update = BCON_NEW ("$set", "{", "balance", BCON_INT64 (data->balance),
 "}");

    mongoc_collection_update_one (data->collection, selector, update, data->opts,
 &local_reply, error);
```

```
    *reply = bson_copy (&local_reply);
    bson_destroy (&local_reply);
    bson_destroy (update);
    return r;
}

void test_callback_money_transfer(mongoc_client_t* client, mongoc_collection_t*
 collection, int amount_to_transfer){

    bson_t reply;
    bool r = true;
    const bson_t *doc;
    bson_iter_t iter;
    ctx_t alice_ctx;
    ctx_t bob_ctx;
    bson_error_t error;

    // find query
    bson_t *alice_query = bson_new ();
    BSON_APPEND_UTF8(alice_query, "name", "Alice");

    bson_t *bob_query = bson_new ();
    BSON_APPEND_UTF8(bob_query, "name", "Bob");

    // create session
    // set causal consistency to false
    mongoc_session_opt_t *session_opts = mongoc_session_opts_new ();
    mongoc_session_opts_set_causal_consistency (session_opts, false);
    // start the session
    mongoc_client_session_t *client_session = mongoc_client_start_session (client,
 session_opts, &error);

    // add session to options
    bson_t *opts = bson_new();
    mongoc_client_session_append (client_session, opts, &error);

    // deduct 500 from Alice
    // find account balance of Alice
    mongoc_cursor_t *cursor = mongoc_collection_find_with_opts (collection,
 alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
```

```
    int64_t alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance >= amount_to_transfer);
    int64_t new_alice_balance = alice_balance - amount_to_transfer;

    // set variables which will be used by callback function
    alice_ctx.collection = collection;
    alice_ctx.opts = opts;
    alice_ctx.balance = new_alice_balance;
    alice_ctx.account = alice_query;

    // callback
    r = mongoc_client_session_with_transaction (client_session, &callback_session,
NULL, &alice_ctx, &reply, &error);
    assert(r);

    // find account balance of Alice after transaction
    cursor = mongoc_collection_find_with_opts (collection, alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance == new_alice_balance);
    assert(alice_balance == 500);

        // add 500 to bob's balance
    // find account balance of Bob
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    int64_t bob_balance = (bson_iter_value (&iter))->value.v_int64;
    int64_t new_bob_balance = bob_balance + amount_to_transfer;

    bob_ctx.collection = collection;
    bob_ctx.opts = opts;
    bob_ctx.balance = new_bob_balance;
    bob_ctx.account = bob_query;

    // set read & write concern
    mongoc_read_concern_t *read_concern = mongoc_read_concern_new ();
    mongoc_write_concern_t *write_concern = mongoc_write_concern_new ();
    mongoc_transaction_opt_t *txn_opts = mongoc_transaction_opts_new ();

    mongoc_write_concern_set_w(write_concern, MONGOC_WRITE_CONCERN_W_MAJORITY);
```

```
    mongoc_read_concern_set_level(read_concern, MONGOC_READ_CONCERN_LEVEL_SNAPSHOT);
    mongoc_transaction_opts_set_write_concern (txn_opts, write_concern);
    mongoc_transaction_opts_set_read_concern (txn_opts, read_concern);

    // callback
    r = mongoc_client_session_with_transaction (client_session, &callback_session,
 txn_opts, &bob_ctx, &reply, &error);
    assert(r);

 // find account balance of Bob after transaction
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    bob_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(bob_balance == new_bob_balance);
    assert(bob_balance == 1500);

    // cleanup
    bson_destroy(alice_query);
    bson_destroy(bob_query);
    mongoc_client_session_destroy(client_session);
    bson_destroy(opts);
    mongoc_transaction_opts_destroy(txn_opts);
    mongoc_read_concern_destroy(read_concern);
    mongoc_write_concern_destroy(write_concern);
    mongoc_cursor_destroy(cursor);
    bson_destroy(doc);
}
int main(int argc, char* argv[]) {
    mongoc_init ();
    mongoc_client_t* client = mongoc_client_new (<connection uri>);
    bson_error_t error;

    // connect to bank db
    mongoc_database_t *database = mongoc_client_get_database (client, "bank");
    // access account collection
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "bank",
 "account");
    // set amount to transfer
    int64_t amount_to_transfer = 500;
    // delete the collection if already existing
    mongoc_collection_drop(collection, &error);
```

```
    // open Alice account
    bson_t *alice_account = bson_new ();
    BSON_APPEND_UTF8(alice_account, "name", "Alice");
    BSON_APPEND_INT64(alice_account, "balance", 1000);

    // open Bob account
    bson_t *bob_account = bson_new ();
    BSON_APPEND_UTF8(bob_account, "name", "Bob");
    BSON_APPEND_INT64(bob_account, "balance", 1000);

    bool r = true;

    r = mongoc_collection_insert_one(collection, alice_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}
    r = mongoc_collection_insert_one(collection, bob_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}

    test_callback_money_transfer(client, collection, amount_to_transfer);

}
```

Python

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Python.

```
// Sample Python code with callback api

import pymongo

def callback(session, balance, query):
    collection.update_one(query, {'$set': {"balance": balance}}, session=session)

client = pymongo.MongoClient(<connection uri>)
rc_snapshot = pymongo.read_concern.ReadConcern('snapshot')
wc_majority = pymongo.write_concern.WriteConcern('majority')

# To start, drop and create an account collection and insert balances for both Alice
 and Bob
collection = client.get_database("bank").get_collection("account")
collection.drop()
collection.insert_one({"_id": 1, "name": "Alice", "balance": 1000})
collection.insert_one({"_id": 2, "name": "Bob", "balance": 1000})
```

```
amount_to_transfer = 500

# deduct 500 from Alice's account
alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert alice_balance >= amount_to_transfer
new_alice_balance = alice_balance - amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.with_transaction(lambda s: callback(s, new_alice_balance, {"name":
 "Alice"}), read_concern=rc_snapshot, write_concern=wc_majority)

updated_alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert updated_alice_balance == new_alice_balance

# add 500 to Bob's account
bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert bob_balance >= amount_to_transfer
new_bob_balance = bob_balance + amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.with_transaction(lambda s: callback(s, new_bob_balance, {"name":
 "Bob"}), read_concern=rc_snapshot, write_concern=wc_majority)

updated_bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert updated_bob_balance == new_bob_balance
```

Transaction API examples for core API

Javascript

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Javascript.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success ***
// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
```

```
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert(aliceBalance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert.eq(newAliceBalance, findAliceBalance);

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
assert.eq(newBobBalance, findBobBalance);

session.commitTransaction();

accountColl.find();
```

C#

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with C#.

```
// C# Core API

public void TransferMoneyWithRetry(IMongoCollection<bSondocument> accountColl,
 IClientSessionHandle session)
{
    var amountToTransfer = 500;

    // start transaction
   var transactionOptions = new TransactionOptions(
                readConcern: ReadConcern.Snapshot,
                writeConcern: WriteConcern.WMajority);
```

```
    session.StartTransaction(transactionOptions);
   try
    {
        // deduct $500 from Alice's account
        var aliceBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
        Debug.Assert(aliceBalance >= amountToTransfer);
        var newAliceBalance = aliceBalance.AsInt32 - amountToTransfer;
        accountColl.UpdateOne(session, Builders<bSondocument>.Filter.Eq("name",
 "Alice"),
                                Builders<bSondocument>.Update.Set("balance",
 newAliceBalance));
        aliceBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
        Debug.Assert(aliceBalance == newAliceBalance);

        // add $500 from Bob's account
        var bobBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
        var newBobBalance = bobBalance.AsInt32 + amountToTransfer;
        accountColl.UpdateOne(session, Builders<bSondocument>.Filter.Eq("name",
 "Bob"),
                                Builders<bSondocument>.Update.Set("balance",
 newBobBalance));
        bobBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
        Debug.Assert(bobBalance == newBobBalance);

    }
    catch (Exception e)
    {
        session.AbortTransaction();
        throw;
    }

    session.CommitTransaction();
 }

}
public void DoTransactionWithRetry(MongoClient client)
```

```
{
    var dbName = "bank";
    var collName = "account";
    using (var session = client.StartSession(new
 ClientSessionOptions{CausalConsistency = false}))
    {
        try
        {
            var bankDB = client.GetDatabase(dbName);
            var accountColl = bankDB.GetCollection<bSondocument>(collName);
            bankDB.DropCollection(collName);
            accountColl.InsertOne(session, new BsonDocument { {"name", "Alice"},
 {"balance", 1000 } });
            accountColl.InsertOne(session, new BsonDocument { {"name", "Bob"},
 {"balance", 1000 } });

            while(true) {
                try
                {
                        TransferMoneyWithRetry(accountColl, session);
                        break;
                }
                catch (MongoException e)
                {
                    if(e.HasErrorLabel("TransientTransactionError"))
                    {
                        continue;
                    }
                    else
                    {
                        throw;
                    }
                }
            }

            // check values outside of transaction
            var aliceNewBalance =
 accountColl.Find(Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
            var bobNewBalance =
 accountColl.Find(Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
            Debug.Assert(aliceNewBalance ==  500);
            Debug.Assert(bobNewBalance ==  1500);
```

```
        }
        catch (Exception e)
        {
            Console.WriteLine("Error running transaction: " + e.Message);
        }
    }
}
```

Ruby

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Ruby.

```
# Ruby Core API

def transfer_money_w_retry(session, accountColl)
    amountToTransfer = 500

    session.start_transaction(read_concern: {level: :snapshot}, write_concern:
 {w: :majority})
    # deduct $500 from Alice's account
    aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
    assert aliceBalance >= amountToTransfer
    newAliceBalance = aliceBalance - amountToTransfer
    accountColl.update_one({"name"=>"Alice"}, { "$set" =>
 {"balance"=>newAliceBalance} }, :session=> session)
    aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
    assert_equal(newAliceBalance, aliceBalance)

    # add $500 to Bob's account
    bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
    newBobBalance = bobBalance + amountToTransfer
    accountColl.update_one({"name"=>"Bob"}, { "$set" =>
 {"balance"=>newBobBalance} }, :session=> session)
    bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
    assert_equal(newBobBalance, bobBalance)

    session.commit_transaction
```

```
end

def do_txn_w_retry(client)
     dbName = "bank"
    collName = "account"

    session = client.start_session(:causal_consistency=> false)
    bankDB = Mongo::Database.new(client, dbName)
    accountColl = bankDB[collName]
    accountColl.drop()

    accountColl.insert_one({"name"=>"Alice", "balance"=>1000})
    accountColl.insert_one({"name"=>"Bob", "balance"=>1000})

    begin
        transferMoneyWithRetry(session, accountColl)
        puts "transaction committed"
    rescue Mongo::Error => e
        if e.label?('TransientTransactionError')
            retry
        else
            puts "transaction failed"
            raise
        end
    end

    # check results outside of transaction
    aliceBalance = accountColl.find({"name"=>"Alice"}).first['balance']
    bobBalance = accountColl.find({"name"=>"Bob"}).first['balance']
    assert_equal(aliceBalance, 500)
    assert_equal(bobBalance, 1500)

end
```

Go

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Go.

```
// Go - Core API
type Account struct {
    Name string
    Balance  int
```

```
}

func transferMoneyWithRetry(sessionContext mongo.SessionContext, accountColl
 *mongo.Collection, t *testing.T) error {
    amountToTransfer := 500

    transactionOptions :=
 options.Transaction().SetReadConcern(readconcern.Snapshot()).

 SetWriteConcern(writeconcern.New(writeconcern.WMajority()))
    if err := sessionContext.StartTransaction(transactionOptions); err != nil {
        panic(err)
    }

    var result Account
    // deduct $500 from Alice's account
    err := accountColl.FindOne(sessionContext, bson.M{"name":
 "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    newAliceBalance := aliceBalance - amountToTransfer
    _, err = accountColl.UpdateOne(sessionContext, bson.M{"name": "Alice"},
 bson.M{"$set": bson.M{"balance": newAliceBalance}})
    if err != nil {
        sessionContext.AbortTransaction(sessionContext)
    }
    err = accountColl.FindOne(sessionContext, bson.M{"name":
 "Alice"}).Decode(&result)
    aliceBalance = result.Balance
    assert.Equal(t, aliceBalance, newAliceBalance)

    // add $500 to Bob's account
    err = accountColl.FindOne(sessionContext, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    newBobBalance := bobBalance + amountToTransfer
    _, err = accountColl.UpdateOne(sessionContext, bson.M{"name": "Bob"},
 bson.M{"$set": bson.M{"balance": newBobBalance}})
    if err != nil {
        sessionContext.AbortTransaction(sessionContext)
    }
    err = accountColl.FindOne(sessionContext, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance = result.Balance
    assert.Equal(t, bobBalance, newBobBalance)

    err = sessionContext.CommitTransaction(sessionContext)
```

```
    return err
}

func doTransactionWithRetry(t *testing.T) {
    ctx := context.TODO()

    dbName := "bank"
    collName := "account"
    bankDB := client.Database(dbName)
    accountColl := bankDB.Collection(collName)

    client.UseSessionWithOptions(ctx, options.Session().SetCausalConsistency(false),
 func(sessionContext mongo.SessionContext) error {
        accountColl.Drop(ctx)
        accountColl.InsertOne(sessionContext, bson.M{"name" : "Alice",
 "balance":1000})
        accountColl.InsertOne(sessionContext, bson.M{"name" : "Bob",
 "balance":1000})
        for {
            err := transferMoneyWithRetry(sessionContext, accountColl, t)
            if err == nil {
                println("transaction committed")
                return nil
            }
            if mongoErr := err.(mongo.CommandError);
 mongoErr.HasErrorLabel("TransientTransactionError") {
                continue
            }
            println("transaction failed")
            return err
        }
    })

    // check results outside of transaction
    var result Account
    accountColl.FindOne(ctx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    assert.Equal(t, aliceBalance, 500)
    accountColl.FindOne(ctx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    assert.Equal(t, bobBalance, 1500)
}
```

Java

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Java.

```
// Java (sync) - Core API

public void transferMoneyWithRetry() {
   // connect to server
    MongoClientURI mongoURI = new MongoClientURI(uri);
    MongoClient mongoClient = new MongoClient(mongoURI);

    MongoDatabase bankDB = mongoClient.getDatabase("bank");
    MongoCollection accountColl = bankDB.getCollection("account");
    accountColl.drop();

   // insert some sample data
    accountColl.insertOne(new Document("name", "Alice").append("balance", 1000));
    accountColl.insertOne(new Document("name", "Bob").append("balance", 1000));

    while (true) {
        try {
            doTransferMoneyWithRetry(accountColl, mongoClient);
            break;
        } catch (MongoException e) {
            if (e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL)) {
                continue;
            } else {
                throw e;
            }
        }
    }
}

public void doTransferMoneyWithRetry(MongoCollection accountColl, MongoClient
 mongoClient) {
    int amountToTransfer = 500;

   TransactionOptions txnOptions = TransactionOptions.builder()
      .readConcern(ReadConcern.SNAPSHOT)
      .writeConcern(WriteConcern.MAJORITY)
      .build();
    ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
```

```
    try ( ClientSession clientSession = mongoClient.startSession(sessionOptions) ) {
        clientSession.startTransaction(txnOptions);

        // deduct $500 from Alice's account
        List<Document> documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
        int aliceBalance = (int) documentList.get(0).get("balance");
        Assert.assertTrue(aliceBalance >= amountToTransfer);
        int newAliceBalance = aliceBalance - amountToTransfer;
        accountColl.updateOne(clientSession, new Document("name", "Alice"), new
 Document("$set", new Document("balance", newAliceBalance)));

        // check Alice's new balance
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
        int updatedBalance = (int) documentList.get(0).get("balance");
        Assert.assertEquals(updatedBalance, newAliceBalance);

        // add $500 to Bob's account
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
        int bobBalance = (int) documentList.get(0).get("balance");
        int newBobBalance = bobBalance + amountToTransfer;
        accountColl.updateOne(clientSession, new Document("name", "Bob"), new
 Document("$set", new Document("balance", newBobBalance)));

        // check Bob's new balance
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
        updatedBalance = (int) documentList.get(0).get("balance");
        Assert.assertEquals(updatedBalance, newBobBalance);

        // commit transaction
        clientSession.commitTransaction();
    }
}
// Java (async) -- Core API
public void transferMoneyWithRetry() {
    // connect to the server
    MongoClient mongoClient = MongoClients.create(uri);
```

```
    MongoDatabase bankDB = mongoClient.getDatabase("bank");
    MongoCollection accountColl = bankDB.getCollection("account");
    SubscriberLatchWrapper<Void> dropCallback = new SubscriberLatchWrapper<>();
    mongoClient.getDatabase("bank").drop().subscribe(dropCallback);
    dropCallback.await();

    // insert some sample data
    SubscriberLatchWrapper<InsertOneResult> insertionCallback = new
 SubscriberLatchWrapper<>();
    accountColl.insertOne(new Document("name", "Alice").append("balance",
 1000)).subscribe(insertionCallback);
    insertionCallback.await();

    insertionCallback = new SubscriberLatchWrapper<>();
    accountColl.insertOne(new Document("name", "Bob").append("balance",
 1000)).subscribe(insertionCallback);;
    insertionCallback.await();

    while (true) {
        try {
            doTransferMoneyWithRetry(accountColl, mongoClient);
            break;
        } catch (MongoException e) {
            if (e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL)) {
                continue;
            } else {
                throw e;
            }
        }
    }
}

public void doTransferMoneyWithRetry(MongoCollection accountColl, MongoClient
 mongoClient) {
    int amountToTransfer = 500;

    // start the transaction
    TransactionOptions txnOptions = TransactionOptions.builder()
            .readConcern(ReadConcern.SNAPSHOT)
            .writeConcern(WriteConcern.MAJORITY)
            .build();
    ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
```

```
   SubscriberLatchWrapper<ClientSession> sessionCallback = new
SubscriberLatchWrapper<>();
   mongoClient.startSession(sessionOptions).subscribe(sessionCallback);
   ClientSession session = sessionCallback.get().get(0);
   session.startTransaction(txnOptions);

   // deduct $500 from Alice's account
   SubscriberLatchWrapper<Document> findCallback = new SubscriberLatchWrapper<>();
   accountColl.find(session, new Document("name",
"Alice")).first().subscribe(findCallback);
   Document documentFound = findCallback.get().get(0);
   int aliceBalance = (int) documentFound.get("balance");
   int newAliceBalance = aliceBalance - amountToTransfer;

   SubscriberLatchWrapper<UpdateResult> updateCallback = new
SubscriberLatchWrapper<>();
   accountColl.updateOne(session, new Document("name",
"Alice"), new Document("$set", new Document("balance",
newAliceBalance))).subscribe(updateCallback);
   updateCallback.await();

   // check Alice's new balance
   findCallback = new SubscriberLatchWrapper<>();
   accountColl.find(session, new Document("name",
"Alice")).first().subscribe(findCallback);
   documentFound = findCallback.get().get(0);
   int updatedBalance = (int) documentFound.get("balance");
   Assert.assertEquals(updatedBalance, newAliceBalance);

   // add $500 to Bob's account
   findCallback = new SubscriberLatchWrapper<>();
   accountColl.find(session, new Document("name",
"Bob")).first().subscribe(findCallback);
   documentFound = findCallback.get().get(0);
   int bobBalance = (int) documentFound.get("balance");
   int newBobBalance = bobBalance + amountToTransfer;

   updateCallback = new SubscriberLatchWrapper<>();
   accountColl.updateOne(session, new Document("name", "Bob"), new Document("$set",
new Document("balance", newBobBalance))).subscribe(updateCallback);
   updateCallback.await();

   // check Bob's new balance
```

```
    findCallback = new SubscriberLatchWrapper<>();
    accountColl.find(session, new Document("name",
 "Bob")).first().subscribe(findCallback);
    documentFound = findCallback.get().get(0);
    updatedBalance = (int) documentFound.get("balance");
    Assert.assertEquals(updatedBalance, newBobBalance);

    // commit the transaction
    SubscriberLatchWrapper<Void> transactionCallback = new
 SubscriberLatchWrapper<>();
    session.commitTransaction().subscribe(transactionCallback);
    transactionCallback.await();
}

public class SubscriberLatchWrapper<T> implements Subscriber<T> {

    /**
     * A Subscriber that stores the publishers results and provides a latch so can
 block on completion.
     *
     * @param <T> The publishers result type
     */
    private final List<T> received;
    private final List<RuntimeException> errors;
    private final CountDownLatch latch;
    private volatile Subscription subscription;
    private volatile boolean completed;

    /**
     * Construct an instance
     */
    public SubscriberLatchWrapper() {
        this.received = new ArrayList<>();
        this.errors = new ArrayList<>();
        this.latch = new CountDownLatch(1);
    }

    @Override
    public void onSubscribe(final Subscription s) {
        subscription = s;
        subscription.request(Integer.MAX_VALUE);
    }

    @Override
```

```
    public void onNext(final T t) {
        received.add(t);
    }

    @Override
    public void onError(final Throwable t) {
        if (t instanceof RuntimeException) {
            errors.add((RuntimeException) t);
        } else {
            errors.add(new RuntimeException("Unexpected exception", t));
        }
        onComplete();
    }

    @Override
    public void onComplete() {
        completed = true;
        subscription.cancel();
        latch.countDown();
    }

    /**
     * Get received elements
     *
     * @return the list of received elements
     */
    public List<T> getReceived() {
        return received;
    }

    /**
     * Get received elements.
     *
     * @return the list of receive elements
     */
    public List<T> get() {
        return await().getReceived();
    }

    /**
     * Await completion or error
     *
     * @return this
     */
```

```
    public SubscriberLatchWrapper<T> await() {
        subscription.request(Integer.MAX_VALUE);
        try {
            if (!latch.await(300, TimeUnit.SECONDS)) {
                throw new MongoTimeoutException("Publisher onComplete timed out for
 300 seconds");
            }
        } catch (InterruptedException e) {
            throw new MongoInterruptedException("Interrupted waiting for
 observeration", e);
        }
        if (!errors.isEmpty()) {
            throw errors.get(0);
        }
        return this;
    }

    public boolean getCompleted() {
        return this.completed;
    }

    public void close() {
        subscription.cancel();
        received.clear();
    }
}
```

C

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with C.

```
// Sample C code with core session

bool core_session(mongoc_client_session_t *client_session, mongoc_collection_t*
 collection, bson_t *selector, int64_t balance){
    bool r = true;
    bson_error_t error;
    bson_t *opts = bson_new();
    bson_t *update = BCON_NEW ("$set", "{", "balance", BCON_INT64 (balance), "}");

    // set read & write concern
```

```
    mongoc_read_concern_t *read_concern = mongoc_read_concern_new ();
    mongoc_write_concern_t *write_concern = mongoc_write_concern_new ();
    mongoc_transaction_opt_t *txn_opts = mongoc_transaction_opts_new ();

    mongoc_write_concern_set_w(write_concern, MONGOC_WRITE_CONCERN_W_MAJORITY);
    mongoc_read_concern_set_level(read_concern, MONGOC_READ_CONCERN_LEVEL_SNAPSHOT);
    mongoc_transaction_opts_set_write_concern (txn_opts, write_concern);
    mongoc_transaction_opts_set_read_concern (txn_opts, read_concern);

    mongoc_client_session_start_transaction (client_session, txn_opts, &error);
    mongoc_client_session_append (client_session, opts, &error);

    r = mongoc_collection_update_one (collection, selector, update, opts, NULL,
 &error);

    mongoc_client_session_commit_transaction (client_session, NULL, &error);
    bson_destroy (opts);
    mongoc_transaction_opts_destroy(txn_opts);
    mongoc_read_concern_destroy(read_concern);
    mongoc_write_concern_destroy(write_concern);
    bson_destroy (update);
    return r;
}

void test_core_money_transfer(mongoc_client_t* client, mongoc_collection_t*
 collection, int amount_to_transfer){

    bson_t reply;
    bool r = true;
    const bson_t *doc;
    bson_iter_t iter;
    bson_error_t error;

    // find query
    bson_t *alice_query = bson_new ();
    BSON_APPEND_UTF8(alice_query, "name", "Alice");

    bson_t *bob_query = bson_new ();
    BSON_APPEND_UTF8(bob_query, "name", "Bob");

    // create session
    // set causal consistency to false
    mongoc_session_opt_t *session_opts = mongoc_session_opts_new ();
    mongoc_session_opts_set_causal_consistency (session_opts, false);
```

```
   // start the session
   mongoc_client_session_t *client_session = mongoc_client_start_session (client,
session_opts, &error);

   // add session to options
   bson_t *opts = bson_new();
   mongoc_client_session_append (client_session, opts, &error);

   // deduct 500 from Alice
   // find account balance of Alice
   mongoc_cursor_t *cursor = mongoc_collection_find_with_opts (collection,
alice_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   int64_t alice_balance = (bson_iter_value (&iter))->value.v_int64;
   assert(alice_balance >= amount_to_transfer);
   int64_t new_alice_balance = alice_balance - amount_to_transfer;

   // core
   r = core_session (client_session, collection, alice_query, new_alice_balance);
   assert(r);

   // find account balance of Alice after transaction
   cursor = mongoc_collection_find_with_opts (collection, alice_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   alice_balance = (bson_iter_value (&iter))->value.v_int64;
   assert(alice_balance == new_alice_balance);
   assert(alice_balance == 500);

   // add 500 to Bob's balance
   // find account balance of Bob
   cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   int64_t bob_balance = (bson_iter_value (&iter))->value.v_int64;
   int64_t new_bob_balance = bob_balance + amount_to_transfer;

   //core
   r = core_session (client_session, collection, bob_query, new_bob_balance);
   assert(r);
```

```
    // find account balance of Bob after transaction
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    bob_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(bob_balance == new_bob_balance);
    assert(bob_balance == 1500);

    // cleanup
    bson_destroy(alice_query);
    bson_destroy(bob_query);
    mongoc_client_session_destroy(client_session);
    bson_destroy(opts);
    mongoc_cursor_destroy(cursor);
    bson_destroy(doc);
}

int main(int argc, char* argv[]) {
    mongoc_init ();
    mongoc_client_t* client = mongoc_client_new (<connection uri>);
    bson_error_t error;

    // connect to bank db
    mongoc_database_t *database = mongoc_client_get_database (client, "bank");
    // access account collection
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "bank",
 "account");
    // set amount to transfer
    int64_t amount_to_transfer = 500;
    // delete the collection if already existing
    mongoc_collection_drop(collection, &error);

    // open Alice account
    bson_t *alice_account = bson_new ();
    BSON_APPEND_UTF8(alice_account, "name", "Alice");
    BSON_APPEND_INT64(alice_account, "balance", 1000);

    // open Bob account
    bson_t *bob_account = bson_new ();
    BSON_APPEND_UTF8(bob_account, "name", "Bob");
    BSON_APPEND_INT64(bob_account, "balance", 1000);
```

```
    bool r = true;

    r = mongoc_collection_insert_one(collection, alice_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}
    r = mongoc_collection_insert_one(collection, bob_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}

    test_core_money_transfer(client, collection, amount_to_transfer);

}
```

Scala

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Scala.

```
// Scala Core API
def transferMoneyWithRetry(sessionObservable: SingleObservable[ClientSession] ,
 database: MongoDatabase ): Unit = {
    val accountColl = database.getCollection("account")
    var amountToTransfer = 500

    var transactionObservable: Observable[ClientSession] =
 sessionObservable.map(clientSession => {
    clientSession.startTransaction()

    // deduct $500 from Alice's account
    var aliceBalance = accountColl.find(clientSession, Document("name" ->
 "Alice")).await().head.getInteger("balance")
    assert(aliceBalance >= amountToTransfer)
    var newAliceBalance = aliceBalance - amountToTransfer
    accountColl.updateOne(clientSession, Document("name" -> "Alice"),
 Document("$set" -> Document("balance" -> newAliceBalance))).await()
    aliceBalance = accountColl.find(clientSession, Document("name" ->
 "Alice")).await().head.getInteger("balance")
    assert(aliceBalance == newAliceBalance)

    // add $500 to Bob's account
    var bobBalance = accountColl.find(clientSession, Document("name" ->
 "Bob")).await().head.getInteger("balance")
    var newBobBalance = bobBalance + amountToTransfer
    accountColl.updateOne(clientSession, Document("name" -> "Bob"), Document("$set"
 -> Document("balance" -> newBobBalance))).await()
```

```
    bobBalance = accountColl.find(clientSession, Document("name" ->
 "Bob")).await().head.getInteger("balance")
    assert(bobBalance == newBobBalance)

    clientSession
    })

    transactionObservable.flatMap(clientSession =>
 clientSession.commitTransaction()).await()
}

def doTransactionWithRetry(): Unit = {
    val client: MongoClient = MongoClientWrapper.getMongoClient()
    val database: MongoDatabase = client.getDatabase("bank")
    val accountColl = database.getCollection("account")
    accountColl.drop().await()

    val sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build()
    var  sessionObservable: SingleObservable[ClientSession] =
 client.startSession(sessionOptions)
    accountColl.insertOne(Document("name" -> "Alice", "balance" -> 1000)).await()
    accountColl.insertOne(Document("name" -> "Bob", "balance" -> 1000)).await()

    var retry = true
    while (retry) {
        try {
        transferMoneyWithRetry(sessionObservable, database)
        println("transaction committed")
        retry = false
        }
        catch {
        case e: MongoException if
 e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL) => {
            println("retrying transaction")
        }
        case other: Throwable => {
            println("transaction failed")
            retry = false
            throw other

        }
        }
    }
```

```
    // check results outside of transaction
    assert(accountColl.find(Document("name" ->
 "Alice")).results().head.getInteger("balance") == 500)
    assert(accountColl.find(Document("name" ->
 "Bob")).results().head.getInteger("balance") == 1500)

    accountColl.drop().await()

}
```

Python

The following code demonstrates how to utilize the Amazon DocumentDB transaction API with Python.

```
// Sample Python code with Core api

import pymongo

client = pymongo.MongoClient(<connection_string>)
rc_snapshot = pymongo.read_concern.ReadConcern('snapshot')
wc_majority = pymongo.write_concern.WriteConcern('majority')

# To start, drop and create an account collection and insert balances for both Alice
 and Bob
collection = client.get_database("bank").get_collection("account")
collection.drop()
collection.insert_one({"_id": 1, "name": "Alice", "balance": 1000})
collection.insert_one({"_id": 2, "name": "Bob", "balance": 1000})

amount_to_transfer = 500

# deduct 500 from Alice's account
alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert alice_balance >= amount_to_transfer
new_alice_balance = alice_balance - amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.start_transaction(read_concern=rc_snapshot, write_concern=wc_majority)
    collection.update_one({"name": "Alice"}, {'$set': {"balance":
 new_alice_balance}}, session=session)
    session.commit_transaction()
```

```
updated_alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert updated_alice_balance == new_alice_balance

# add 500 to Bob's account
bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert bob_balance >= amount_to_transfer
new_bob_balance = bob_balance + amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.start_transaction(read_concern=rc_snapshot, write_concern=wc_majority)
    collection.update_one({"name": "Bob"}, {'$set': {"balance": new_bob_balance}},
 session=session)
    session.commit_transaction()

updated_bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert updated_bob_balance == new_bob_balance
```

Supported commands

Command	Supported
abortTransaction	Yes
commitTransaction	Yes
endSessions	Yes
killSession	Yes
killAllSession	Yes
killAllSessionsByPattern	No
refreshSessions	No
startSession	Yes

Unsupported capabilities

Methods	Stages or Commands

`db.collection.aggregate()`	`$collStats`
	`$currentOp`
	`$indexStats`
	`$listSessions`
	`$out`

`db.collection.count()`
`db.collection.countDocuments()`
	`$where`
	`$near`
	`$nearSphere`

`db.collection.insert()`	`insert` is not supported if it is not run against an existing collection. This method is supported if it targets a pre-existing collectio n.

Sessions

MongoDB sessions are a framework that is used to support retryable writes, causal consistency, transactions, and manage operations across databases. When a session is created, a logical session identifier (lsid) is generated by the client and is used to tag all operations within that session when sending commands to the server.

Amazon DocumentDB supports the use of sessions to enable transactions, but does not support causal consistency or retryable writes.

When utilizing transactions within Amazon DocumentDB, a transaction will be initiated from within a session using the `session.startTransaction()` API and a session supports a single transaction at a time. Similarly, transactions are completed using either the commit (`session.commitTransaction()`) or abort (`session.abortTransaction()`) APIs.

Causal consistency

Causal consistency guarantees that within a single client session the client will observe read-after-write consistency, monatomic reads/writes, and writes will follow reads and these guarantees apply across all instances in a cluster, not just the primary. Amazon DocumentDB does not support causal consistency and the following statement will result in an error.

```
var mySession = db.getMongo().startSession();
var mySessionObject = mySession.getDatabase('test').getCollection('account');

mySessionObject.updateOne({"_id": 2}, {"$inc": {"balance": 400}});
//Result:{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

mySessionObject.find()
//Error: error: {
//         "ok" : 0,
//         "code" : 303,
//         "errmsg" : "Feature not supported: 'causal consistency'",
//         "operationTime" : Timestamp(1603461817, 493214)
//}

mySession.endSession()
```

You can disable causal consistency within a session. Please note, doing so will enable you to utilize the session framework, but will not provide causal consistency guarantees for reads. When using Amazon DocumentDB, reads from the primary will be read-after-write consistent and reads from the replica instances will be eventually consistent. Transactions are the primary use case for utilizing sessions.

```
var mySession = db.getMongo().startSession({causalConsistency: false});
var mySessionObject = mySession.getDatabase('test').getCollection('account');

mySessionObject.updateOne({"_id": 2}, {"$inc": {"balance": 400}});
//Result:{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

mySessionObject.find()
//{ "_id" : 1, "name" : "Bob", "balance" : 100 }
//{ "_id" : 2, "name" : "Alice", "balance" : 1700 }
```

Retryable writes

Retryable writes is a capability in which the client will attempt to retry write operations, one time, when network errors occur or if the client is unable to find the primary. In Amazon DocumentDB, retryable writes are not supported and must be disabled. You can disable it with the command (`retryWrites=false`) in the connection string.

> **Note**
>
> If you are using legacy mongo shell (not mongosh), do not include the `retryWrites=false` command in any code string. By default, retryable writes are disabled. Including `retryWrites=false` might cause a failure in normal read commands.

Transaction errors

When using transactions, there are scenarios that can yield an error that states that a transaction number does not match any in progress transaction.

The error can be generated in at least two different scenarios:

- After the one-minute transaction timeout.
- After an instance restart (due to patching, crash recovery, etc.), it is possible to receive this error even in cases where the transaction successfully committed. During an instance restart, the database can't tell the difference between a transaction that successfully completed versus a transaction that aborted. In other words, the transaction completion state is ambiguous.

The best way to handle this error is to make transactional updates idempotent -- for example, by using the `$set` mutator instead of an increment/decrement operation. See below:

```
{ "ok" : 0,
"operationTime" : Timestamp(1603938167, 1),
"code" : 251,
"errmsg" : "Given transaction number 1 does not match any in-progress transactions."
}
```

Best practices for Amazon DocumentDB

Learn best practices for working with Amazon DocumentDB (with MongoDB compatibility). This section is continually updated as new best practices are identified.

Topics

- Basic operational guidelines
- Instance sizing
- Working with indexes
- Security best practices
- Cost optimization
- Using metrics to identify performance issues
- TTL and time series workloads
- Migrations
- Working with cluster parameter groups
- Aggregation pipeline queries
- batchInsert and batchUpdate

Basic operational guidelines

The following are basic operational guidelines that everyone should follow when working with Amazon DocumentDB. The Amazon DocumentDB Service Level Agreement requires that you follow these guidelines.

- Deploy a cluster consisting of two or more Amazon DocumentDB instances in two AWS Availability Zones. For production workloads, we recommend deploying a cluster consisting of three or more Amazon DocumentDB instances in three Availability Zones.
- Use the service within the stated service limits. For more information, see Amazon DocumentDB Quotas and limits.
- Monitor your memory, CPU, connections, and storage usage. To help you maintain system performance and availability, set up Amazon CloudWatch to notify you when usage patterns change or when you approach the capacity of your deployment.

- Scale up your instances when you are approaching capacity limits. Your instances should be provisioned with enough compute resources (i.e., RAM, CPU) to accommodate unforeseen increases in demand from your applications.
- Set your backup retention period to align with your recovery point objective.
- Test failover for your cluster to understand how long the process takes for your use case. For more information, see Amazon DocumentDB Failover.
- Connect to your Amazon DocumentDB cluster with the cluster endpoint (see Amazon DocumentDB endpoints) and in replica set mode (see Connecting to Amazon DocumentDB as a replica set) to minimize the impact of a failover on your application.
- Choose a driver read preference setting that maximizes read scaling while meeting your application's read consistency requirements. The `secondaryPreferred` read preference enables replica reads and frees up the primary instance to do more work. For more information, see Read preference options.
- Design your application to be resilient in the event of network and database errors. Use your driver's error mechanism to distinguish between transient errors and persistent errors. Retry transient errors using an exponential backoff mechanism when appropriate. Ensure that your application considers data consistency when implementing retry logic.
- Enable cluster deletion protection for all production clusters, or any cluster that has valuable data. Before deleting an Amazon DocumentDB cluster, take a final snapshot. If you are deploying resources with AWS CloudFormation, enable termination protection. For more information, see Termination protection and deletion protection.
- When creating an Amazon DocumentDB cluster, the `--engine-version` is an optional parameter that defaults to the latest major engine version. The current major engine version is 5.0.0. When new major engine versions are released, the default engine version for `--engine-version` will be updated to reflect the last major engine version. As a result, for production workloads, and especially those that are dependent on scripting, automation, or AWS CloudFormation templates, we recommend that you explicitly specify the `--engine-version` to the intended major version.

Instance sizing

One of the most critical aspects of choosing an instance size in Amazon DocumentDB is the amount of RAM for your cache. Amazon DocumentDB reserves one-third of the RAM for its own services, meaning that only two-thirds of the instance RAM is available for the cache. Thus, it is an Amazon DocumentDB best practice to choose an instance type with enough RAM to fit your

working set (i.e., data and indexes) in memory. Having properly sized instances will help optimize for overall performance and potentially minimize I/O cost.

To determine whether your application's working set fits in memory, monitor the `BufferCacheHitRatio` using Amazon CloudWatch for each instance in a cluster that is under load.

The `BufferCacheHitRatio` CloudWatch metric measures the percentage of data and indexes served from an instance's memory cache (versus the storage volume). Generally speaking, the value of `BufferCacheHitRatio` should be as high as possible, as reading data from working set memory is faster and more cost-effective than reading from the storage volume. While it is desirable to keep `BufferCacheHitRatio` as close to 100% as possible, the best achievable value will depend on your application's access patterns and performance requirements. To maintain the highest possible `BufferCacheHitRatio`, it is recommended that the instances in your cluster are provisioned with enough RAM to be able to fit your indexes and working data set in memory.

If your indexes do not fit into memory, you will see a lower `BufferCacheHitRatio`. Continually reading from disk incurs additional I/O costs and is not better than reading from memory. If your `BufferCacheHitRatio` ratio is lower than expected, scale up the instance size for your cluster to provide more RAM to fit working set data in memory. If scaling up the instance class results in a dramatic increase in `BufferCacheHitRatio`, then your application's working set did not fit in memory. Continue to scale up until `BufferCacheHitRatio` no longer increases dramatically after a scaling operation. For information about monitoring an instance's metrics, see Amazon DocumentDB metrics.

Depending on your workload and latency requirements, it may be acceptable for your application to have higher `BufferCacheHitRatio` values during steady state usage, but have the `BufferCacheHitRatio` dip periodically as analytic queries that need to scan an entire collection are run on an instance. These periodic dips in `BufferCacheHitRatio` may manifest as higher latency for subsequent queries that need to repopulate the working set data from the storage volume back into the buffer cache. **We recommend that you test your workloads in a pre-production environment with a representative production workload first in order to understand the performance characteristics and `BufferCacheHitRatio` before deploying the workload to production.**

The `BufferCacheHitRatio` is an instance-specific metric, so different instances within the same cluster may have different `BufferCacheHitRatio` values depending on how reads are distributed among the primary and replica instances. If your operational workload cannot handle periodic

increases in latency from repopulating the working set cache after running analytic queries, you should try to isolate the regular workload's buffer cache from that of the analytic queries. You can achieve complete `BufferCacheHitRatio` isolation by directing operational queries to the primary instance and analytic queries only to the replica instances. You can also achieve partial isolation by directing analytic queries to a specific replica instance with the understanding that some percentage of regular queries will also run on that replica and could potentially be affected.

Appropriate `BufferCacheHitRatio` values depend on your use case and application requirements. There is no one best or minimum value for this metric; only you can decide if the tradeoff from a temporarily lower `BufferCacheHitRatio` is acceptable from a cost and performance perspective.

Working with indexes

Building Indexes

When importing data into Amazon DocumentDB, you should create your indexes before importing large datasets. You can use the Amazon DocumentDB Index Tool to extract indexes from a running MongoDB instance or `mongodump` directory, and create those indexes in an Amazon DocumentDB cluster. For more guidance on migrations, see Migrating to Amazon DocumentDB.

Index selectivity

We recommend that you limit the creation of indexes to fields where the number of duplicate values is less than 1% of the total number of documents in the collection. As an example, if your collection contains 100,000 documents, only create indexes on fields where the same value occurs 1000 times or fewer.

Choosing an index with a high number of unique values (i.e., a high cardinality) ensures that filter operations return a small number of documents, thereby yielding good performance during index scans. An example of a high-cardinality index is a unique index, which guarantees that equality predicates return at most a single document. Examples of low-cardinality include an index over a Boolean field and an index over day of the week. Due to their poor performance, low cardinality indexes are unlikely to be chosen by the database's query optimizer. At the same time, low cardinality indexes continue to consume resources such as disk space and I/Os. As a rule of thumb, you should target indexes on fields where the typical value frequency is 1% of the total collection size or less.

Additionally, it is recommended to only create indexes on fields that are commonly utilized as a filter and regularly look for unused indexes. For more information, see How do I analyze index usage and identify unused indexes?.

Impact of indexes on writing data

While indexes can improve query performance by avoiding the need to scan every document in a collection, this improvement comes with a tradeoff. For each index on a collection, every time a document is inserted, updated, or deleted, the database must update the collection and write the fields to each of the indexes for the collection. For example, if a collection has nine indexes, the database must perform ten writes before acknowledging the operation to the client. Thus, each additional index incurs additional write latency, I/O's, and increase in the overall utilized storage.

Cluster instances need to be appropriately sized to keep all working set memory. This avoids the need to continuously read index pages from the storage volume, which negatively impacts performance and generates higher I/O costs. For more information, see Instance sizing.

For best performance, minimize the number of indexes in your collections, adding only those indexes necessary to improve performance for common queries. While workloads vary, a good guideline is to keep the number of indexes per collection to five or fewer.

Identifying missing indexes

Identifying missing indexes is a best practice that we recommend performing on a regular basis. For more information, please see How do I identify missing indexes?.

Identifying unused indexes

Identifying and removing unused indexes is a best practice that we recommend performing on a regular basis. For more information, please see How do I analyze index usage and identify unused indexes?.

Security best practices

For security best practices, you must use AWS Identity and Access Management (IAM) accounts to control access to Amazon DocumentDB API operations, especially operations that create, modify, or delete Amazon DocumentDB resources. Such resources include clusters, security groups, and parameter groups. You must also use IAM to control actions that perform common administrative

actions such as backing up restoring clusters. When creating IAM roles, employ the principle of least privilege.

- Enforce least privilege with role-based access control.
- Assign an individual IAM account to each person who manages Amazon DocumentDB resources. Do not use the AWS account root user to manage Amazon DocumentDB resources. Create an IAM user for everyone, including yourself.
- Grant each IAM user the minimum set of permissions that are required to perform their duties.
- Use IAM groups to effectively manage permissions for multiple users. For more information about IAM, see the IAM User Guide. For information about IAM best practices, see IAM Best Practices.
- Regularly rotate your IAM credentials.
- Configure AWS Secrets Manager to automatically rotate the secrets for Amazon DocumentDB. For more information, see Rotating Your AWS Secrets Manager Secrets and Rotating Secrets for Amazon DocumentDB in the *AWS Secrets Manager User Guide*.
- Grant each Amazon DocumentDB user the minimum set of permissions that are required to perform their duties. For more information, see Database access using Role-Based Access Control.
- Use Transport Layer Security (TLS) to encrypt your data in transit and AWS KMS to encrypt your data at rest.

Cost optimization

The following best practices can help you manage and minimize your costs when using Amazon DocumentDB. For pricing information, see Amazon DocumentDB (with MongoDB compatibility) pricing and Amazon DocumentDB (with MongoDB compatibility) FAQs.

- Create billing alerts at thresholds of 50 percent and 75 percent of your expected bill for the month. For more information about creating billing alerts, see Creating a Billing Alarm.
- Amazon DocumentDB's architecture separates storage and compute, so even a single-instance cluster is highly durable. The cluster storage volume replicates data six ways across three Availability Zones, providing extremely high durability regardless of the number of instances in the cluster. A typical production cluster has three or more instances to provide high availability. However, you can optimize costs by using a single instance development cluster when high availability is not required.

- For development and test scenarios, stop a cluster when it is no longer needed and start the cluster when development resumes. For more information, see Stopping and starting an Amazon DocumentDB cluster.
- Both TTL and change streams incur I/O's when data is written, read, and deleted. If you have enabled these features but are not utilizing them in your application, disabling the features can help reduce costs.

Using metrics to identify performance issues

Topics

- Viewing performance metrics
- Setting a CloudWatch alarm
- Evaluating performance metrics
- Evaluating Amazon DocumentDB instance usage with CloudWatch metrics
- Tuning queries

To identify performance issues caused by insufficient resources and other common bottlenecks, you can monitor the metrics available for your Amazon DocumentDB cluster.

Viewing performance metrics

Monitor performance metrics on a regular basis to see the average, maximum, and minimum values for a variety of time ranges. This helps you identify when performance is degraded. You can also set Amazon CloudWatch alarms for particular metric thresholds so that you are alerted if they are reached.

To troubleshoot performance issues, it’s important to understand the baseline performance of the system. After you set up a new cluster and get it running with a typical workload, capture the average, maximum, and minimum values of all the performance metrics at different intervals (for example, 1 hour, 24 hours, 1 week, 2 weeks). This gives you an idea of what is normal. It helps to get comparisons for both peak and off-peak hours of operation. You can then use this information to identify when performance is dropping below standard levels.

You can view performance metrics using the AWS Management Console or AWS CLI. For more information, see Viewing CloudWatch data.

Setting a CloudWatch alarm

To set a CloudWatch alarm, see Using Amazon CloudWatch Alarms in the *Amazon CloudWatch User Guide*.

Evaluating performance metrics

An instance has several different categories of metrics. How you determine acceptable values depends on the metric.

CPU

- **CPU Utilization** — The percentage of the computer processing capacity used.

Memory

- **Freeable Memory** — How much RAM is available on the instance.
- **Swap Usage** — How much swap space is used by the instance, in megabytes.

Input/output operations

- **Read IOPS**, **Write IOPS** — The average number of disk read or write operations per second.
- **Read Latency**, **Write Latency** — The average time for a read or write operation in milliseconds.
- **Read Throughput**, **Write Throughput** — The average number of megabytes read from or written to disk per second.
- **Disk Queue Depth** — The number of I/O operations that are waiting to be written to or read from disk.

Network traffic

- **Network Receive Throughput**, **Network Transmit Throughput** — The rate of network traffic to and from the instance in megabytes per second.

Database connections

- **DB Connections** — The number of client sessions that are connected to the instance.

Generally speaking, acceptable values for performance metrics depend on what your baseline looks like and what your application is doing. Investigate consistent or trending variances from your baseline.

The following are recommendations and advice about specific types of metrics:

- **High CPU consumption** — High values for CPU consumption might be appropriate, provided that they are in keeping with your goals for your application (like throughput or concurrency) and are expected. If your CPU consumption is consistently over 80 percent, consider scaling up your instances.
- **High RAM consumption** — If your `FreeableMemory` metric frequently dips below 10% of the total instance memory, consider scaling up your instances. For more information on what happens when your DocumentDB instance is experiencing high memory pressure, see Amazon DocumentDB Resource Governance.
- **Swap usage** — This metric should remain at or near zero. If your swap usage is significant, consider scaling up your instances.
- **Network traffic** — For network traffic, talk with your system administrator to understand what the expected throughput is for your domain network and internet connection. Investigate network traffic if throughput is consistently lower than expected.
- **Database connections** — Consider constraining database connections if you see high numbers of user connections together with decreases in instance performance and response time. The best number of user connections for your instance varies based on your instance class and the complexity of the operations being performed. For issues with any performance metrics, one of the first things you can do to improve performance is tune the most used and most expensive queries to see if that lowers the pressure on system resources.

If your queries are tuned and an issue persists, consider upgrading your Amazon DocumentDB instance class to one with more of the resource (CPU, RAM, disk space, network bandwidth, I/O capacity) that is related to the issue you're experiencing.

Evaluating Amazon DocumentDB instance usage with CloudWatch metrics

You can use CloudWatch metrics to watch your instance throughput and discover if your instance class provides sufficient resources for your applications. For information about your instance class

limits, see Instance limits and locate the specifications for your instance class to find your network performance.

If your instance usage is near the instance class limit, then performance may begin to slow. The CloudWatch metrics can confirm this situation so you can plan to manually scale-up to a larger instance class.

Combine the following CloudWatch metrics values to find out if you are nearing the instance class limit:

- **NetworkThroughput**—The amount of network throughput received and transmitted by the clients for each instance in the Amazon DocumentDB cluster. This throughput value doesn't include network traffic between instances in the cluster and the cluster storage volume.
- **StorageNetworkThroughput**—The amount of network throughput received and sent to the Amazon DocumentDB cluster storage volume by each instance in the Amazon DocumentDB cluster.

Add the **NetworkThroughput** to the **StorageNetworkThroughput** to find the network throughput received from and sent to the Amazon DocumentDB cluster storage volume by each instance in your Amazon DocumentDB cluster. The instance class limit for your instance should be greater than the sum of these two combined metrics.

You can use the following metrics to review additional details of the network traffic from your client applications when sending and receiving:

- **NetworkReceiveThroughput**—The amount of network throughput received from clients by each instance in the Amazon DocumentDB cluster. This throughput doesn't include network traffic between instances in the cluster and the cluster storage volume.
- **NetworkTransmitThroughput**—The amount of network throughput sent to clients by each instance in the Amazon DocumentDB cluster. This throughput doesn't include network traffic between instances in the cluster and the cluster storage volume.
- **StorageNetworkReceiveThroughput**—The amount of network throughput received from the Amazon DocumentDB cluster storage volume by each instance in the cluster.
- **StorageNetworkTransmitThroughput**—The amount of network throughput sent to the Amazon DocumentDB cluster storage volume by each instance in the cluster.

Add all of these metrics together to evaluate how your network usage compares to the instance class limit. The instance class limit should be greater than the sum of these combined metrics.

The network limits and CPU utilization by an instance are mutual. When the network throughput increases, then the CPU utilization also increases. Monitoring the CPU and network usage provides information about how and why the resources are being exhausted.

To help minimize network usage, you can consider:

- Using a larger instance class.
- Dividing the write requests in batches to reduce overall transactions.
- Directing the read-only workload to a read-only instance.
- Deleting any unused indexes.

Tuning queries

One of the best ways to improve cluster performance is to tune your most commonly used and most resource-intensive queries to make them less expensive to run.

You can use the profiler (see Profiling Amazon DocumentDB operations) to log the execution time and details of operations that were performed on your cluster. Profiler is useful for monitoring the slowest operations on your cluster to help you improve individual query performance and overall cluster performance.

You can also use the `explain` command to learn how to analyze a query plan for a particular query. Use this information to modify a query or underlying collection to improve your query performance (for example, adding an index).

TTL and time series workloads

Document deletion resulting from TTL index expiry is a best effort process. Documents are not guaranteed to be deleted within any specific period. Factors like instance size, instance resource utilization, document size, overall throughput, the number of indexes, and whether indexes and the working set fit in memory can all affect the timing of when expired documents are deleted by the TTL process.

When the TTL monitor deletes your documents, each deletion incurs I/O costs, which increases your bill. If throughput and TTL delete rates increase, you should expect a higher bill due to

increased I/O usage. However, if you do not create a TTL index to delete documents, but instead segment documents into collections based on time and simply drop those collections when they are no longer needed, you will not incur any IO costs. This can be significantly more cost effective than using a TTL index.

For time-series workloads, you can consider creating rolling collections instead of a TTL index as rolling collections can be a better way to delete data and less I/O intensive. If you have large collections (especially collections over 1TB) or TTL deletion I/O costs are a concern, we recommend that you partition documents into collections based on time, and drop collections when the documents are no longer needed. You can create one collection per day or one per week, depending on your data ingest rate. While requirements will vary depending on your application, a good rule of thumb is to have more smaller collections rather than a few large collections. Dropping these collections does not incur I/O costs, and can be faster and more cost effective than using a TTL index.

Migrations

As a best practice, we recommend that when migrating data to Amazon DocumentDB, you first create your indexes in Amazon DocumentDB before migrating the data. Creating the indexes first can reduce the overall time and increase the speed of the migration. To do this, you can use the Amazon DocumentDB Index Tool. For more information on migrations, see the Amazon DocumentDB migration guide.

We also recommend that before you migrate your production database, it is a best practice to fully test your application on Amazon DocumentDB, taking into consideration functionality, performance, operations, and cost.

Working with cluster parameter groups

We recommend that you try out cluster parameter group changes on a test cluster before applying the changes to your production clusters. For information about backing up your cluster, see Backing up and restoring in Amazon DocumentDB.

Aggregation pipeline queries

When creating an aggregation pipeline query with multiple stages and evaluating only a subset of the data in the query, use the `$match` stage as the first stage or in the beginning of the pipeline.

Using `$match` first will reduce the number of documents subsequent stages within the aggregation pipeline query will need to process, thus improving the performance of your query.

`batchInsert` and `batchUpdate`

When performing a high rate of concurrent `batchInsert` and/or `batchUpdate` operations, and the amount of `FreeableMemory` (CloudWatch Metric) goes to zero on your primary instance, you can either reduce the concurrency of the batch insert or update workload or, if concurrency of the workload cannot be reduced, increase the instance size to increase the amount of `FreeableMemory`.

Functional differences: Amazon DocumentDB and MongoDB

The following are the functional differences between Amazon DocumentDB (with MongoDB compatibility) and MongoDB.

Topics

- Functional benefits of Amazon DocumentDB
- Updated functional differences
- Functional differences with MongoDB

Functional benefits of Amazon DocumentDB

Implicit transactions

In Amazon DocumentDB, all CRUD statements (`findAndModify`, `update`, `insert`, `delete`) guarantee atomicity and consistency, even for operations that modify multiple documents. With the launch of Amazon DocumentDB 4.0, explicit transactions that provide ACID properties for multi-statement and multi-collection operations are now supported. For more on using transactions in Amazon DocumentDB, please see Transactions in Amazon DocumentDB.

The following are examples of operations in Amazon DocumentDB that modify multiple documents that satisfy both atomic and consistent behaviors.

```
db.miles.update(
    { "credit_card": { $eq: true } },
    { $mul: { "flight_miles.$[]": NumberInt(2) } },
    { multi: true }
)
```

```
db.miles.updateMany(
    { "credit_card": { $eq: true } },
    { $mul: { "flight_miles.$[]": NumberInt(2) } }
)
```

```
db.runCommand({
  update: "miles",
  updates: [
    {
      q: { "credit_card": { $eq: true } },
      u: { $mul: { "flight_miles.$[]": NumberInt(2) } },
      multi: true
    }
  ]
})
```

```
db.products.deleteMany({
  "cost": { $gt: 30.00 }
})
```

```
db.runCommand({
  delete: "products",
  deletes: [{ q: { "cost": { $gt: 30.00 } }, limit: 0 }]
})
```

The individual operations that compose bulk operations such as `updateMany` and `deleteMany` are atomic but the entirety of the bulk operation is not atomic. For example, the entirety of the `insertMany` operation is atomic if the individual insert operations execute successfully without error. If an error is encountered with an `insertMany` operation, each individual insert statement within the `insertMany` operation will execute as an atomic operation. If you require ACID properties for `insertMany`, `updateMany`, and `deleteMany` operations, it is recommended to use a transaction.

Updated functional differences

Amazon DocumentDB continues to improve compatibility with MongoDB by working backwards from the capabilities our customers ask us to build. This section contains the functional differences that we have removed in Amazon DocumentDB to make migrations and building applications easier for our customers.

Topics

- Array indexing
- Multi-key indexes
- Null characters in strings
- Role-based access control
- $regex indexing
- Projection for nested documents

Array indexing

As of April 23, 2020, Amazon DocumentDB now supports the ability to index arrays that are greater than 2,048 bytes. The limit for an individual item in an array still remains as 2,048 bytes, which is consistent with MongoDB.

If you are creating a new index, no action is needed to take advantage of the improved functionality. If you have an existing index, you can take advantage of the improved functionality by dropping the index and then recreating it. The current index version with the improved capabilities is `"v" : 3`.

> **Note**
>
> For production clusters, the dropping of the index may have an impact on your application performance. We recommend that you first test and proceed with caution when making changes to a production system. In addition, the time it will take to recreate the index will be a function of the overall data size of the collection.

You can query for the version of your indexes using the following command.

```
db.collection.getIndexes()
```

Output from this operation looks something like the following. In this output, the version of the index is `"v" : 3`, which is the most current index version.

```
[
    {
        "v" : 3,
```

```
        "key" : {
        "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

Multi-key indexes

As of April 23, 2020, Amazon DocumentDB now supports the ability to create a compound index with multiple keys in the same array.

If you are creating a new index, no action is needed to take advantage of the improved functionality. If you have an existing index, you can take advantage of the improved functionality by dropping the index and then recreating it. The current index version with the improved capabilities is `"v" : 3`.

> **Note**
>
> For production clusters, the dropping of the index may have an impact on your application performance. We recommend that you first test and proceed with caution when making changes to a production system. In addition, the time it will take to recreate the index will be a function of the overall data size of the collection.

You can query for the version of your indexes using the following command.

```
db.collection.getIndexes()
```

Output from this operation looks something like the following. In this output, the version of the index is `"v" : 3`, which is the most current index version.

```
[
    {
        "v" : 3,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
```

```
        "ns" : "test.test"
    }
]
```

Null characters in strings

As of June 22, 2020, Amazon DocumentDB now supports null characters (`'\0'`) in strings.

Role-based access control

As of March 26, 2020, Amazon DocumentDB supports role-based access control (RBAC) for built-in roles. To learn more, see Role-Based Access Control.

`$regex` indexing

As of June 22, 2020, Amazon DocumentDB now supports the ability for `$regex` operators to utilize an index.

To utilize an index with the `$regex` operator, you must use the `hint()` command. When using `hint()`, you must specify the name of the field you are applying the `$regex` on. For example, if you have an index on field `product` with the index name as `p_1`, `db.foo.find({product: /^x.*/}).hint({product:1})` will utilize the `p_1` index, but `db.foo.find({product: /^x.*/}).hint("p_1")` will not utilize the index. You can verify if an index is chosen by utilizing the `explain()` command or using the profiler for logging slow queries. For example, `db.foo.find({product: /^x.*/}).hint("p_1").explain()`.

> **Note**
>
> The `hint()` method can only be used with one index at a time.

The use of an index for a `$regex` query is optimized for regex queries that utilize a prefix and do not specify the `i`, `m`, or `o` regex options.

When using an index with `$regex`, it is recommended that you create an index on highly selective fields where the number of duplicate values is less than 1% of the total number of documents in the collection. As an example, if your collection contains 100,000 documents, only create indexes on fields where the same value occurs 1000 times or fewer.

Projection for nested documents

There is a functional difference with `$project` operator between Amazon DocumentDB and MongoDB in version 3.6 that has been resolved in Amazon DocumentDB 4.0 but will remain unsupported in Amazon DocumentDB 3.6.

Amazon DocumentDB 3.6 only considers the first field in a nested document when applying a projection whereas MongoDB 3.6 will parse subdocuments and apply the projection to each sub document as well.

For example: if the projection is `"a.b.c": 1`, then the behavior works as expect in both Amazon DocumentDB and MongoDB. However, if the projection is `{a:{b:{c:1}}}` then Amazon DocumentDB 3.6 will only apply the projection to a and not b or c. In Amazon DocumentDB 4.0, the projection `{a:{b:{c:1}}}` will be applied to a, b, and c.

Functional differences with MongoDB

Topics

- $vectorSearch operator
- OpCountersCommand
- Admin databases and collections
- cursormaxTimeMS
- explain()
- Index builds
- Lookup with empty key in path
- MongoDB APIs, operations, and data types
- mongodump and mongorestore utilities
- Result ordering
- Retryable writes
- Sparse index
- Using $elemMatch within an $all expression
- $ne, $nin, $nor, $not, $exists, and $elemMatch indexing
- Dollar($) and dot(.) in field names
- $lookup

- $natural and reverse sorting

$vectorSearch operator

Amazon DocumentDB does not support `$vectorSearch` as an independent operator. Instead we support, `vectorSearch` inside the `$search` operator. For more information, see Vector search for Amazon DocumentDB.

OpCountersCommand

Amazon DocumentDB's `OpCountersCommand` behavior deviates from MongoDB's `opcounters.command` as follows:

- MongoDB's `opcounters.command` counts all commands except insert, update, and delete while Amazon DocumentDB's `OpCountersCommand` also excludes the `find` command.
- Amazon DocumentDB counts some internal commands toward the `OpCountersCommand`.

Admin databases and collections

Amazon DocumentDB does not support the admin or local database nor MongoDB `system.*` or `startup_log` collections respectively.

cursormaxTimeMS

In Amazon DocumentDB, `cursor.maxTimeMS` resets the counter for each `getMore` request. Thus, if a 3000MS `maxTimeMS` is specified, the query takes 2800MS, and each subsequent `getMore` request takes 300MS, then the cursor will not timeout. The cursor will only timeout when a single operations, either the query or an individual `getMore` request, takes more than the specified `maxTimeMS`. Further, the sweeper that checks cursor execution time runs at a five (5) minute granularity.

explain()

Amazon DocumentDB emulates the MongoDB 3.6, 4.0, and 5.0 APIs on a purpose-built database engine that utilizes a distributed, fault-tolerant, self-healing storage system. As a result, query plans and the output of `explain()` may differ between Amazon DocumentDB and MongoDB. Customers who want control over their query plan can use the `$hint` operator to enforce selection of a preferred index.

Index builds

Amazon DocumentDB allows only one index build to occur on a collection at any given time. Either in the foreground or the background. If operations such as `createIndex()` or `dropIndex()` occur on the same collection when an index build is currently in progress, the newly attempted operation will fail.

By default, index builds in Amazon DocumentDB and MongoDB version 4.0 occur in the background. MongoDB version 4.2, and later ignores the background index build option if specified to createIndexes or its shell helpers `createIndex()` and `createIndexes()`.

A Time to Live (TTL) index starts expiring documents after the index build is completed.

Lookup with empty key in path

When you look up with a key that includes empty string as part of the path (e.g. `x.`, `x..`b), and the object has an empty string key path (e.g. `{"x" : [ { "" : 10 }, { "b" : 20 } ]}`) inside an array, Amazon DocumentDB will return different results than if you were to run the same look up in MongoDB.

In MongoDB, the empty key path look up within array works as expected when the empty string key is not at the end of path look up. However, when the empty string key is at the end of path look up, it does not look into the array.

However in Amazon DocumentDB, only the first element within the array is read, because `getArrayIndexFromKeyString` converts empty string to `0`, so string key look up is treated as array index look up.

MongoDB APIs, operations, and data types

Amazon DocumentDB is compatible with the MongoDB 3.6, 4.0, and 5.0 APIs. For an up-to-date list of supported functionality, see Supported MongoDB APIs, operations, and data types in Amazon DocumentDB.

`mongodump` and `mongorestore` utilities

Amazon DocumentDB does not support an admin database and thus does not dump or restore the admin database when using the `mongodump` or `mongorestore` utilities. When you create a new database in Amazon DocumentDB using `mongorestore`, you need to re-create the user roles in addition to the restore operation.

> **Note**
>
> We recommend MongoDB Database Tools up to and including version 100.6.1 for Amazon DocumentDB. You can access the MongoDB Database Tools downloads here.

Result ordering

Amazon DocumentDB does not guarantee implicit result sort ordering of result sets. To ensure the ordering of a result set, explicitly specify a sort order using `sort()`.

The following example sorts the items in the inventory collection in descending order based on the stock field.

```
db.inventory.find().sort({ stock: -1 })
```

When using the `$sort` aggregation stage, the sort order is not preserved unless the `$sort` stage is the last stage in the aggregation pipeline. When using the `$sort` aggregation stage in combination with the `$group` aggregation stage, the `$sort` aggregation stage is only applied to the `$first` and `$last` accumulators. In Amazon DocumentDB 4.0, support was added for `$push` to respect sort order from the previous `$sort` stage.

Retryable writes

Starting with MongoDB 4.2 compatible drivers, retryable writes are enabled by default. However, Amazon DocumentDB does not currently support retryable writes. The functional difference will manifest itself in an error message similar to the following.

```
{"ok":0,"errmsg":"Unrecognized field: 'txnNumber'","code":9,"name":"MongoError"}
```

Retryable writes can be disabled via the connection string (for example, `MongoClient("mongodb://my.mongodb.cluster/db?retryWrites=false")`) or the MongoClient constructor's keyword argument (for example, `MongoClient("mongodb://my.mongodb.cluster/db", retryWrites=False)`).

The following is a Python example that disables retryable writes in the connection string.

```
client =
 pymongo.MongoClient('mongodb://
<username>:<password>@docdb-2019-03-17-16-49-12.cluster-ccuszbx3pn5e.us-
east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0',w='majority',j=True,retryWrites=False)
```

Sparse index

To use a sparse index that you have created in a query, you must use the `$exists` clause on the fields that cover the index. If you omit `$exists`, Amazon DocumentDB will not use the sparse index.

The following is an example.

```
db.inventory.count({ "stock": { $exists: true }})
```

For sparse, multi-key indexes, Amazon DocumentDB does not support a unique key constraint if the look up of a document results in a set of values and only a subset of the indexed fields is missing. For example, `createIndex({"a.b" : 1 }, { unique : true, sparse :true })` is not supported, given the input of `"a" : [ { "b" : 2 }, { "c" : 1 } ]`, as `"a.c"` is stored in the index.

Using `$elemMatch` within an `$all` expression

Amazon DocumentDB does not currently support the use of the `$elemMatch` operator within an `$all` expression. As a workaround, you can use the `$and` operator with `$elemMatch` as follows.

Original operation:

```
db.col.find({
  qty: {
    $all: [
      { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } },
      { "$elemMatch": { num: 40, size: "XL" } }
    ]
  }
})
```

Updated operation:

```
db.col.find({
  $and: [
    { qty: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } } },
    { qty: { "$elemMatch": { qty: 40, size: "XL" } } }
  ]
})
```

$ne, $nin, $nor, $not, $exists, and $elemMatch indexing

Amazon DocumentDB does not currently support the ability to use indexes with the `$ne`, `$nin`, `$nor`, `$not`, `$exists`, and `$distinct` operators. As a result, using these operators will result in collection scans. Performing a filter or match before using one of these operators will reduce the amount of data that needs to be scanned, and thus can improve performance.

Amazon DocumentDB added support for index scans with the `$elemMatch` operator in Amazon DocumentDB 5.0 and elastic clusters. Index scans are supported when the query only filter has one level of the `$elemMatch` filter but are not supported if a nested `$elemMatch` query is included.

`$elemMatch` query shape that supports index scans in Amazon DocumentDB 5.0:

```
db.foo.find( { "a": {$elemMatch: { "b": "xyz", "c": "abc"} } })
```

`$elemMatch` query shape that does not support index scans in Amazon DocumentDB 5.0:

```
db.foo.find( { "a": {$elemMatch: { "b": {$elemMatch: { "d": "xyz", "e": "abc"} }} } })
```

Dollar($) and dot(.) in field names

Amazon DocumentDB does not support querying Dollar($) prefixed fields in $in, $nin and $all in nested objects. For example, the following query is not valid in Amazon DocumentDB:

```
coll.find({"field": {"$all": [{ "$a": 1 }]}})
```

$lookup

Amazon DocumentDB supports the ability to do equality matches (for example, left outer join) and also supports uncorrelated subqueries, but does not support correlated subqueries.

Utilizing an index with $lookup

You can now utilize an index with the `$lookup` stage operator. Based on your use case, there are multiple indexing algorithms that you can use to optimize for performance. This section will explain the different indexing algorithms for `$lookup` and help you choose the best one for your workload.

By default, Amazon DocumentDB will utilize the hash algorithm when `allowDiskUse:false` is used and sort merge when `allowDiskUse:true` is used.

> **Note**
>
> The `allowDiskUse` option is currently not supported for the `find` command. The option is only supported as part of aggregation. We recommend using the aggregation framework with `allowDiskUse:true` to handle large queries that might exceed memory limits.

For some use cases, it may be desirable to force the query optimizer to use a different algorithm. Below are the different indexing algorithms that the `$lookup` aggregation operator can use:

- **Nested loop**: A nested loop plan is typically beneficial for a workload if the foreign collection is <1 GB and the field in the foreign collection has an index. If the nested loop algorithm is being used, the explain plan will show the stage as `NESTED_LOOP_LOOKUP`.
- **Sort merge**: A sort merge plan is typically beneficial for a workload if the foreign collection does not have an index on the field used in lookup and the working dataset doesn't fit in memory. If the sort merge algorithm is being used, the explain plan will show the stage as `SORT_LOOKUP`.
- **Hash**: A hash plan is typically beneficial for a workload if the foreign collection is < 1 GB and the working dataset fits in memory. If the hash algorithm is being used, the explain plan will show the stage as `HASH_LOOKUP`.

You can identify the indexing algorithm that is being used for the `$lookup` operator by using `explain` on the query. Below is an example:

```
db.localCollection.explain().aggregate(
   [
      {
         $lookup:
```

```
            {
               from: "foreignCollection",
               localField: "a",
               foreignField: "b",
               as: "joined"
            }
      }
   ]
)

output
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.localCollection",
        "winningPlan" : {
            "stage" : "SUBSCAN",
            "inputStage" : {
                "stage" : "SORT_AGGREGATE",
                "inputStage" : {
                    "stage" : "SORT",
                    "inputStage" : {
                        "stage" : "NESTED_LOOP_LOOKUP",
                        "inputStages" : [
                            {
                                "stage" : "COLLSCAN"
                            },
                            {
                                "stage" : "FETCH",
                                "inputStage" : {
                                    "stage" : "COLLSCAN"
                                }
                            }
                        ]
                    }
                }
            }
        }
    },
    "serverInfo" : {
        "host" : "devbox-test",
        "port" : 27317,
        "version" : "3.6.0"
    },
```

```
    "ok" : 1
}
```

As an alternative to using the `explain()` method, you can use the profiler to review the algorithm that is being utilized with your use of the `$lookup` operator. For more information on the profiler, please see Profiling Amazon DocumentDB operations.

Using a `planHint`

If you wish to force the query optimizer to use a different indexing algorithm with `$lookup`, you can use a `planHint`. To do that, use the comment in the aggregation stage options to force a different plan. Below is an example of the syntax for the comment:

```
comment : {
    comment :  "<string>",
    lookupStage : { planHint : "SORT" | "HASH" | "NESTED_LOOP" }
}
```

Below is an example of using the `planHint` to force the query optimizer to use the HASH indexing algorithm:

```
db.foo.aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"HASH\" }}"
```

To test which algorithm is best for your workload, you can use the `executionStats` parameter of the `explain` method to measure the execution time of the `$lookup` stage while modifying the indexing algorithm (i.e., `HASH`/`SORT`/`NESTED_LOOP`).

The following example shows how to use `executionStats` to measure the execution time of the `$lookup` stage using the `SORT` algorithm.

```
db.foo.explain("executionStats").aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"SORT\" }}"
```

`$natural` and reverse sorting

Amazon DocumentDB supports `$natural` for forward collection scans only. Reverse collection scans (`{$natural: -1}`) will lead to a `MongoServerError`.

Supported MongoDB APIs, operations, and data types in Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) is a fast, scalable, highly-available, and fully managed document database service that supports MongoDB workloads. Amazon DocumentDB is compatible with the MongoDB 3.6, 4.0, and 5.0 APIs. This section lists the supported functionality. For support using MongoDB APIs and drivers, please consult the MongoDB Community Forums. For support using the Amazon DocumentDB service, please contact the appropriate AWS support team. For functional differences between Amazon DocumentDB and MongoDB, please see Functional differences: Amazon DocumentDB and MongoDB.

MongoDB commands and operators that are internal-only or not applicable to a fully-managed service are not supported and are not included in the list of supported functionality.

We have added over 50+ additional capabilities since launch, and will continue to work backwards from our customers to deliver the capabilities that they need. For information on the most recent launches, see Amazon DocumentDB Announcements.

If there is a feature that isn't supported that you'd like us to build, let us know by sending an email with your accountID, the requested features, and use case to the Amazon DocumentDB service team.

Topics

- Database commands
- Query and projection operators
- Update operators
- Geospatial
- Cursor methods
- Aggregation pipeline operators
- Data types
- Indexes and index properties

Database commands

Topics

- Administrative Commands
- Aggregation
- Authentication
- Diagnostic commands
- Query and write operations
- Role management commands
- Sessions commands
- User management
- Sharding commands

Administrative Commands

Command	3.6	4.0	5.0	Elastic cluster
Capped Collections	No	No	No	No
cloneCollectionAsCapped	No	No	No	No
collMod	Partial	Partial	Partial	Partial
collMod: expireAfterSeconds	Yes	Yes	Yes	Yes
convertToCapped	No	No	No	No
copydb	No	No	No	No
create	Yes	Yes	Yes	Yes
createView	No	No	No	No
createIndexes	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
currentOp	Yes	Yes	Yes	Yes
drop	Yes	Yes	Yes	Yes
dropDatabase	Yes	Yes	Yes	Yes
dropIndexes	Yes	Yes	Yes	Yes
filemd5	No	No	No	No
getAuditConfig	No	Yes	Yes	No
killCursors	Yes	Yes	Yes	Yes
killOp	Yes	Yes	Yes	Yes
listCollections*	Yes	Yes	Yes	Yes
listDatabases	Yes	Yes	Yes	Yes
listIndexes	Yes	Yes	Yes	Yes
reIndex	No	No	Yes	No
renameCollection	Yes	Yes	Yes	No
setAuditConfig	No	Yes	Yes	No

* The `type` key in the filter option is not supported.

Aggregation

Command	3.6	4.0	5.0	Elastic cluster
aggregate	Yes	Yes	Yes	Yes
count	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
distinct	Yes	Yes	Yes	Yes
mapReduce	No	No	No	No

Authentication

Command	3.6	4.0	5.0	Elastic cluster
authenticate	Yes	Yes	Yes	Yes
logout	Yes	Yes	Yes	Yes

Diagnostic commands

Command	3.6	4.0	5.0	Elastic cluster
buildInfo	Yes	Yes	Yes	Yes
collStats	Yes	Yes	Yes	Yes
connPoolStats	No	No	No	No
connectio nStatus	Yes	Yes	Yes	Yes
dataSize	Yes	Yes	Yes	Yes
dbHash	No	No	No	No
dbStats	Yes	Yes	Yes	Yes
explain	Yes	Yes	Yes	Yes
explain: executionStats	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
features	No	No	No	No
hostInfo	Yes	Yes	Yes	Yes
listCommands	Yes	Yes	Yes	Yes
profiler	Yes	Yes	Yes	No
serverStatus	Yes	Yes	Yes	Yes
top	Yes	Yes	Yes	Yes

Query and write operations

Command	3.6	4.0	5.0	Elastic cluster
Change streams	Yes	Yes	Yes	No
delete	Yes	Yes	Yes	Yes
find	Yes	Yes	Yes	Yes
findAndModify	Yes	Yes	Yes	Yes
getLastError	No	No	No	No
getMore	Yes	Yes	Yes	Yes
getPrevError	No	No	No	No
GridFS	Yes	Yes	Yes	No
insert	Yes	Yes	Yes	Yes
parallelC ollectionScan	No	No	No	No
resetError	No	No	No	No

Command	3.6	4.0	5.0	Elastic cluster
update	Yes	Yes	Yes	Yes
ReplaceOne	Yes	Yes	Yes	Yes

Role management commands

C	3	4	5	E c
c e	Ye	Ye	Ye	No
c l t	Ye	Ye	Ye	No a
c	Ye	Ye	Ye	No
g s	Ye	Ye	Ye	No
r e e	Ye	Ye	Ye	No l
r v c	Ye	Ye	Ye	No
r	Ye	Ye	Ye	No
u e	Ye	Ye	Ye	No

Sessions commands

Command	3.6	4.0	5.0	Elastic cluster
abortTransaction	No	Yes	Yes	No
commitTransaction	No	Yes	Yes	No
endSessions	No	No	No	No
killAllSessions	No	Yes	Yes	No
killAllSessionsByPattern	No	No	No	No
killSessions	No	Yes	Yes	No
refreshSessions	No	No	No	No
startSession	No	Yes	Yes	No

User management

Command	3.6	4.0	5.0	Elastic cluster
createUser	Yes	Yes	Yes	Yes
dropAllUsersFromDatabase	Yes	Yes	Yes	Yes
dropUser	Yes	Yes	Yes	Yes
grantRolesToUser	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
revokeRolesFromUser	Yes	Yes	Yes	Yes
updateUser	Yes	Yes	Yes	Yes
usersInfo	Yes	Yes	Yes	Yes

Sharding commands

Command	Elastic cluster
abortReshardCollection	No
addShard	No
addShardToZone	No
balancerCollectionStatus	No
balancerStart	No
balancerStatus	No
balancerStop	No
checkShardingIndex	No
clearJumboFlag	No
cleanupOrphaned	No
cleanupReshardCollection	No
commitReshardCollection	No
enableSharding	Yes
flushRouterConfig	No

Command	Elastic cluster
getShardMap	No
getShardVersion	No
isdbgrid	No
listShards	No
medianKey	No
moveChunk	No
movePrimary	No
mergeChunks	No
refineCollectionShardKey	No
removeShard	No
removeShardFromZone	No
reshardCollection	No
setAllowMigrations	No
setShardVersion	No
shardCollection	Yes
shardingState	No
split	No
splitVector	No
unsetSharding	No
updateZoneKeyRange	No

Query and projection operators

Topics

- Array Operators
- Bitwise operators
- Comment operator
- Comparison operators
- Element operators
- Evaluation query operators
- Logical operators
- Projection operators

Array Operators

Command	3.6	4.0	5.0	Elastic cluster
$all	Yes	Yes	Yes	Yes
$elemMatch	Yes	Yes	Yes	Yes
$size	Yes	Yes	Yes	Yes

Bitwise operators

Command	3.6	4.0	5.0	Elastic cluster
$bitsAllSet	Yes	Yes	Yes	Yes
$bitsAnySet	Yes	Yes	Yes	Yes
$bitsAllClear	Yes	Yes	Yes	Yes
$bitsAnyClear	Yes	Yes	Yes	Yes

Comment operator

Command	3.6	4.0	5.0	Elastic cluster
$comment	Yes	Yes	Yes	Yes

Comparison operators

Command	3.6	4.0	5.0	Elastic cluster
$eq	Yes	Yes	Yes	Yes
$gt	Yes	Yes	Yes	Yes
$gte	Yes	Yes	Yes	Yes
$in	Yes	Yes	Yes	Yes
$lt	Yes	Yes	Yes	Yes
$lte	Yes	Yes	Yes	Yes
$ne	Yes	Yes	Yes	Yes
$nin	Yes	Yes	Yes	Yes

Element operators

Command	3.6	4.0	5.0	Elastic cluster
$exists	Yes	Yes	Yes	Yes
$type	Yes	Yes	Yes	Yes

Evaluation query operators

Command	3.6	4.0	5.0	Elastic cluster
$expr	No	Yes	Yes	No
$jsonSchema	No	Yes	Yes	No
$mod	Yes	Yes	Yes	Yes
$regex	Yes	Yes	Yes	Yes
$text	No	No	Yes	No
$where	No	No	No	No

Logical operators

Command	3.6	4.0	5.0	Elastic cluster
$and	Yes	Yes	Yes	Yes
$nor	Yes	Yes	Yes	Yes
$not	Yes	Yes	Yes	Yes
$or	Yes	Yes	Yes	Yes

Projection operators

Command	3.6	4.0	5.0	Elastic cluster
$	Yes	Yes	Yes	Yes
$elemMatch	Yes	Yes	Yes	Yes
$meta	No	No	Yes	No

Command	3.6	4.0	5.0	Elastic cluster
$slice	Yes	Yes	Yes	Yes

Update operators

Topics

- Array operators
- Bitwise operators
- Field operators
- Update modifiers

Array operators

Command	3.6	4.0	5.0	Elastic cluster
$	Yes	Yes	Yes	Yes
$[]	Yes	Yes	Yes	Yes
$[<identifier>]	Yes	Yes	Yes	Yes
$addToSet	Yes	Yes	Yes	Yes
$pop	Yes	Yes	Yes	Yes
$pullAll	Yes	Yes	Yes	Yes
$pull	Yes	Yes	Yes	Yes
$push	Yes	Yes	Yes	Yes

Bitwise operators

Command	3.6	4.0	5.0	Elastic cluster
$bit	Yes	Yes	Yes	Yes

Field operators

Operator	3.6	4.0	5.0	Elastic cluster
$currentDate	Yes	Yes	Yes	Yes
$inc	Yes	Yes	Yes	Yes
$max	Yes	Yes	Yes	Yes
$min	Yes	Yes	Yes	Yes
$mul	Yes	Yes	Yes	Yes
$rename	Yes	Yes	Yes	Yes
$set	Yes	Yes	Yes	Yes
$setOnInsert	Yes	Yes	Yes	Yes
$unset	Yes	Yes	Yes	Yes

Update modifiers

Operator	3.6	4.0	5.0	Elastic cluster
$each	Yes	Yes	Yes	Yes
$position	Yes	Yes	Yes	Yes
$slice	Yes	Yes	Yes	Yes

Operator	3.6	4.0	5.0	Elastic cluster
$sort	Yes	Yes	Yes	Yes

Geospatial

Geometry specifiers

Query Selectors	3.6	4.0	5.0	Elastic cluster
$box	No	No	No	No
$center	No	No	No	No
$centerSphere	No	No	No	No
$geometry	Yes	Yes	Yes	Yes
$maxDistance	Yes	Yes	Yes	Yes
$minDistance	Yes	Yes	Yes	Yes
$nearSphere	Yes	Yes	Yes	Yes
$polygon	No	No	No	No
$uniqueDocs	No	No	No	No

Query selectors

Command	3.6	4.0	5.0	Elastic cluster
$geoIntersects	Yes	Yes	Yes	Yes
$geoWithin	Yes	Yes	Yes	Yes
$near	No	No	No	No

Command	3.6	4.0	5.0	Elastic cluster
$nearSphere	Yes	Yes	Yes	Yes
$polygon	No	No	No	No
$uniqueDocs	No	No	No	No

Cursor methods

Command	3.6	4.0	5.0	Elastic cluster
cursor.batchSize()	Yes	Yes	Yes	Yes
cursor.close()	Yes	Yes	Yes	Yes
cursor.collation()	No	No	No	No
cursor.comment()	Yes	Yes	Yes	Yes
cursor.count()	Yes	Yes	Yes	Yes
cursor.explain()	Yes	Yes	Yes	No
cursor.forEach()	Yes	Yes	Yes	Yes
cursor.hasNext()	Yes	Yes	Yes	Yes
cursor.hint()	Yes	Yes	Yes	Yes*
cursor.isClosed()	Yes	Yes	Yes	Yes
cursor.isExhausted()	Yes	Yes	Yes	No
cursor.itcount()	Yes	Yes	Yes	No

Command	3.6	4.0	5.0	Elastic cluster
cursor.limit()	Yes	Yes	Yes	No
cursor.map()	Yes	Yes	Yes	No
cursor.max()	No	No	No	No
cursor.maxScan()	Yes	Yes	Yes	No
cursor.maxTimeMS()	Yes	Yes	Yes	No
cursor.min()	No	No	No	No
cursor.next()	Yes	Yes	Yes	Yes
cursor.noCursorTimeout()	No	No	No	No
cursor.objsLeftInBatch()	Yes	Yes	Yes	No
cursor.pretty()	Yes	Yes	Yes	No
cursor.readConcern()	Yes	Yes	Yes	No
cursor.readPref()	Yes	Yes	Yes	No
cursor.returnKey()	No	No	No	No
cursor.showRecordId()	No	No	No	No
cursor.size()	Yes	Yes	Yes	No
cursor.skip()	Yes	Yes	Yes	No
cursor.sort()	Yes	Yes	Yes	No

Command	3.6	4.0	5.0	Elastic cluster
cursor.tailable()	No	No	No	No
cursor.toArray()	Yes	Yes	Yes	No

* Index `hint` is supported with index expressions. For example, `db.foo.find().hint({x:1})`.

Aggregation pipeline operators

Topics

- Accumulator expressions
- Arithmetic operators
- Array operators
- Boolean operators
- Comparison operators
- Conditional expression operators
- Data type operator
- Data size operator
- Date operators
- Literal operator
- Merge operator
- Natural operator
- Set operators
- Stage operators
- String operators
- System variables
- Text search operator
- Type conversion operators
- Variable operators
- Miscellaneous operators

Accumulator expressions

Expression	3.6	4.0	5.0	Elastic cluster
$accumulator	-	-	No	No
$addToSet	Yes	Yes	Yes	Yes
$avg	Yes	Yes	Yes	Yes
$count	-	-	No	No
$covariancePop	No	No	No	No
$covarian ceSamp	No	No	No	No
$denseRank	No	No	No	No
$derivative	No	No	No	No
$document Number	No	No	No	No
$expMovingAvg	No	No	No	No
$first	Yes	Yes	Yes	Yes
$integral	No	No	No	No
$last	Yes	Yes	Yes	Yes
$max	Yes	Yes	Yes	Yes
$min	Yes	Yes	Yes	Yes
$push	Yes	Yes	Yes	Yes
$rank	No	No	No	No
$shift	No	No	No	No

Expression	3.6	4.0	5.0	Elastic cluster
$stdDevPop	No	No	No	No
$stdDevSamp	No	No	No	No
$sum	Yes	Yes	Yes	Yes

Arithmetic operators

Command	3.6	4.0	5.0	Elastic cluster
$abs	Yes	Yes	Yes	Yes
$add	Yes	Yes	Yes	Yes
$ceil	No	Yes	Yes	Yes
$divide	Yes	Yes	Yes	Yes
$exp	No	Yes	Yes	Yes
$floor	No	Yes	Yes	Yes
$ln	No	Yes	Yes	Yes
$log	No	Yes	Yes	Yes
$log10	No	Yes	Yes	Yes
$mod	Yes	Yes	Yes	Yes
$multiply	Yes	Yes	Yes	Yes
$pow	No	No	No	No
$round	-	-	No	No
$sqrt	No	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
$subtract	Yes	Yes	Yes	Yes
$trunc	No	No	No	No

Array operators

Command	3.6	4.0	5.0	Elastic cluster
$arrayElemAt	Yes	Yes	Yes	Yes
$arrayToObject	Yes	Yes	Yes	Yes
$concatArrays	Yes	Yes	Yes	Yes
$filter	Yes	Yes	Yes	Yes
$first	-	-	No	No
$in	Yes	Yes	Yes	Yes
$indexOfArray	Yes	Yes	Yes	Yes
$isArray	Yes	Yes	Yes	Yes
$last	-	-	No	No
$objectToArray	Yes	Yes	Yes	Yes
$range	Yes	Yes	Yes	Yes
$reverseArray	Yes	Yes	Yes	Yes
$reduce	Yes	Yes	Yes	Yes
$size	Yes	Yes	Yes	Yes
$slice	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
$zip	Yes	Yes	Yes	Yes

Boolean operators

Command	3.6	4.0	5.0	Elastic cluster
$and	Yes	Yes	Yes	Yes
$not	Yes	Yes	Yes	Yes
$or	Yes	Yes	Yes	Yes

Comparison operators

Command	3.6	4.0	5.0	Elastic cluster
$cmp	Yes	Yes	Yes	Yes
$eq	Yes	Yes	Yes	Yes
$gt	Yes	Yes	Yes	Yes
$gte	Yes	Yes	Yes	Yes
$lt	Yes	Yes	Yes	Yes
$lte	Yes	Yes	Yes	Yes
$ne	Yes	Yes	Yes	Yes

Conditional expression operators

Command	3.6	4.0	5.0	Elastic cluster
$cond	Yes	Yes	Yes	Yes
$ifNull	Yes	Yes	Yes	Yes
$switch	No	Yes	Yes	No

Data type operator

Command	3.6	4.0	5.0	Elastic cluster
$type	Yes	Yes	Yes	Yes

Data size operator

Command	3.6	4.0	5.0	Elastic cluster
$binarySize	-	-	No	No
$bsonSize	-	-	No	No

Date operators

Command	3.6	4.0	5.0	Elastic cluster
$dateAdd	No	No	Yes	Yes
$dateDiff	-	-	No	No
$dateFromParts	No	No	No	No

Command	3.6	4.0	5.0	Elastic cluster
$dateFromString	Yes	Yes	Yes	Yes
$dateSubtract	No	No	Yes	Yes
$dateToParts	No	No	No	No
$dateToString	Yes	Yes	Yes	Yes
$dateTrunc	-	-	No	No
$dayOfMonth	Yes	Yes	Yes	Yes
$dayOfWeek	Yes	Yes	Yes	Yes
$dayOfYear	Yes	Yes	Yes	Yes
$hour	Yes	Yes	Yes	Yes
$isoDayOfWeek	Yes	Yes	Yes	Yes
$isoWeek	Yes	Yes	Yes	Yes
$isoWeekYear	Yes	Yes	Yes	Yes
$millisecond	Yes	Yes	Yes	Yes
$minute	Yes	Yes	Yes	Yes
$month	Yes	Yes	Yes	Yes
$second	Yes	Yes	Yes	Yes
$week	Yes	Yes	Yes	Yes
$year	Yes	Yes	Yes	Yes

Literal operator

Command	3.6	4.0	5.0	Elastic cluster
$literal	Yes	Yes	Yes	Yes

Merge operator

Command	3.6	4.0	5.0	Elastic cluster
$mergeObjects	Yes	Yes	Yes	Yes

Natural operator

Command	3.6	4.0	5.0	Elastic cluster
$natural	Yes	Yes	Yes	Yes

Set operators

Command	3.6	4.0	5.0	Elastic cluster
$allElementsTrue	No	Yes	Yes	Yes
$anyElementTrue	No	Yes	Yes	Yes
$setDifference	No	Yes	Yes	Yes
$setEquals	Yes	Yes	Yes	Yes
$setIntersection	Yes	Yes	Yes	Yes
$setIsSubset	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
$setUnion	Yes	Yes	Yes	Yes
$setWindowFields	No	No	No	No

Stage operators

Command	3.6	4.0	5.0	Elastic cluster
$addFields	Yes	Yes	Yes	Yes
$bucket	No	No	No	No
$bucketAuto	No	No	No	No
$collStats	No	No	No	No
$count	Yes	Yes	Yes	Yes
$currentOp	Yes	Yes	Yes	Yes
$facet	No	No	No	No
$geoNear	Yes	Yes	Yes	Yes
$graphLookup	No	No	No	No
$group	Yes	Yes	Yes	Yes
$indexStats	Yes	Yes	Yes	Yes
$limit	Yes	Yes	Yes	Yes
$listLocalSessions	No	No	No	No
$listSessions	No	No	No	No

Command	3.6	4.0	5.0	Elastic cluster
$lookup	Yes	Yes	Yes	Yes
$match	Yes	Yes	Yes	Yes
$merge	-	-	No	No
$out	Yes	Yes	Yes	No
$planCacheStats	-	-	No	No
$project	Yes	Yes	Yes	Yes
$redact	Yes	Yes	Yes	Yes
$replaceRoot	Yes	Yes	Yes	Yes
$sample	Yes	Yes	Yes	Yes
$set	-	-	No	No
$setWindowFields	-	-	No	No
$skip	Yes	Yes	Yes	Yes
$sort	Yes	Yes	Yes	Yes
$sortByCount	No	No	No	No
$unionWith	-	-	No	No
$unset	-	-	No	No
$unwind	Yes	Yes	Yes	Yes

String operators

Command	3.6	4.0	5.0	Elastic cluster
$concat	Yes	Yes	Yes	Yes
$indexOfBytes	Yes	Yes	Yes	Yes
$indexOfCP	Yes	Yes	Yes	Yes
$ltrim	No	No	No	No
$regexFind	-	-	Yes	No
$regexFindAll	-	-	No	No
$regexMatch	-	-	Yes	No
$replaceAll	-	-	No	No
$replaceOne	-	-	No	No
$rtrim	No	No	No	No
$split	Yes	Yes	Yes	Yes
$strcasecmp	Yes	Yes	Yes	Yes
$strLenBytes	Yes	Yes	Yes	Yes
$strLenCP	Yes	Yes	Yes	Yes
$substr	Yes	Yes	Yes	Yes
$substrBytes	Yes	Yes	Yes	Yes
$substrCP	Yes	Yes	Yes	Yes
$toLower	Yes	Yes	Yes	Yes
$toUpper	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
$trim	No	No	No	No

System variables

Command	3.6	4.0	5.0	Elastic cluster
$$CURRENT	No	No	No	No
$$DESCEND	Yes	Yes	Yes	Yes
$$KEEP	Yes	Yes	Yes	Yes
$$PRUNE	Yes	Yes	Yes	Yes
$$REMOVE	No	No	No	No
$$ROOT	Yes	Yes	Yes	Yes

Text search operator

Command	3.6	4.0	5.0	Elastic cluster
$meta	No	No	Yes	No
$search	No	No	Yes	No

Type conversion operators

Command	3.6	4.0	5.0	Elastic cluster
$convert	No	Yes	Yes	Yes
$isNumber	-	-	No	No

Command	3.6	4.0	5.0	Elastic cluster
$toBool	No	Yes	Yes	Yes
$toDate	No	Yes	Yes	Yes
$toDecimal	No	Yes	Yes	Yes
$toDouble	No	Yes	Yes	Yes
$toInt	No	Yes	Yes	Yes
$toLong	No	Yes	Yes	Yes
$toObjectId	No	Yes	Yes	Yes
$toString	No	Yes	Yes	Yes

Variable operators

Command	3.6	4.0	5.0	Elastic cluster
$let	Yes	Yes	Yes	Yes
$map	Yes	Yes	Yes	Yes

Miscellaneous operators

Command	3.6	4.0	5.0	Elastic cluster
$getField	-	-	No	No
$rand	-	-	No	No
$sampleRate	-	-	No	No

Data types

Command	3.6	4.0	5.0	Elastic cluster
32-bit Integer (int)	Yes	Yes	Yes	Yes
64-bit Integer (long)	Yes	Yes	Yes	Yes
Array	Yes	Yes	Yes	Yes
Binary Data	Yes	Yes	Yes	Yes
Boolean	Yes	Yes	Yes	Yes
Date	Yes	Yes	Yes	Yes
DBPointer	No	No	No	No
DBRefs	No	No	No	No
Decimal128	Yes	Yes	Yes	Yes
Double	Yes	Yes	Yes	Yes
JavaScript	No	No	No	No
JavaScript (with scope)	No	No	No	No
MaxKey	Yes	Yes	Yes	Yes
MinKey	Yes	Yes	Yes	Yes
Null	Yes	Yes	Yes	Yes
Object	Yes	Yes	Yes	Yes
ObjectId	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
Regular Expression	Yes	Yes	Yes	Yes
String	Yes	Yes	Yes	Yes
Symbol	No	No	No	No
Timestamp	Yes	Yes	Yes	Yes
Undefined	No	No	No	No

Indexes and index properties

Topics

- Indexes
- Index properties

Indexes

Command	3.6	4.0	5.0	Elastic cluster
2dsphere	Yes	Yes	Yes	Yes
2d Index	No	No	No	No
Compound Index	Yes	Yes	Yes	Yes
Hashed Index	No	No	No	No
Multikey Index	Yes	Yes	Yes	Yes
Single Field Index	Yes	Yes	Yes	Yes

Command	3.6	4.0	5.0	Elastic cluster
Text Index	No	No	Yes	No
Wildcard	No	No	No	No

Index properties

Command	3.6	4.0	5.0	Elastic cluster
Background	Yes	Yes	Yes	Yes
Case Insensitive	No	No	No	No
Hidden	No	No	No	No
Partial	No	No	Yes	No
Sparse	Yes	Yes	Yes	Yes
Text	No	No	Yes	No
TTL	Yes	Yes	Yes	Yes
Unique	Yes	Yes	Yes	Yes
Vector	No	No	Yes	No

Amazon DocumentDB generative artificial intelligence

Amazon DocumentDB offers capabilities to enable machine learning (ML) and generative artificial intelligence (AI) models to work with data stored in Amazon DocumentDB in real time. Customers no longer have to spend time managing separate infrastructure, writing code to connect with another service, and duplicating data from their primary database.

For more information on artificial intelligence and how AWS can support your AI needs, see this "What-is" article.

Topics

- No-code machine learning with Amazon SageMaker AI Canvas
- Vector search for Amazon DocumentDB

No-code machine learning with Amazon SageMaker AI Canvas

Amazon SageMaker AI Canvas enables you to build your own AI/ML models without having to write a single line of code. You can build ML models for common use cases such as regression and forecasting and can access and evaluate foundation models (FMs) from Amazon Bedrock. You can also access public FMs from Amazon SageMaker AI JumpStart for content generation, text extraction, and text summarization to support generative AI solutions.

How to build no-code ML models with SageMaker AI Canvas

Amazon DocumentDB now integrates with Amazon SageMaker AI Canvas to enable no-code machine learning (ML) with data stored in Amazon DocumentDB. You can now build ML models for regression and forecasting needs and use foundation models for content summarization and generation using data stored in Amazon DocumentDB without writing a single line of code.

SageMaker AI Canvas provides a visual interface that allows Amazon DocumentDB customers to generate predictions without requiring any AI/ML expertise or write a single line of code. Customers can now launch the SageMaker AI Canvas workspace from the AWS Management Console, import and join Amazon DocumentDB data for data preparation and model training. Data in Amazon DocumentDB can now be used in SageMaker AI Canvas to build and augment models to predict customer churn, detect fraud, predict maintenance failures, forecast business metrics, and generate content. Customers can now publish and share ML-driven insights across

teams using SageMaker AI Canvas's native integration with QuickSight. Data ingestion pipelines in SageMaker AI Canvas run on Amazon DocumentDB secondary instances by default, ensuring that the performance of application and SageMaker AI Canvas ingestion workloads are not hindered.

Amazon DocumentDB customers can get started with SageMaker AI Canvas by navigating to the new Amazon DocumentDB No-Code ML Console page and connecting to new or available SageMaker AI Canvas workspaces.

Configuring the SageMaker AI domain and user profile

You can connect to Amazon DocumentDB clusters from SageMaker AI domains that are running in VPC Only mode. By launching a SageMaker AI domain in your VPC, you can control the data flow from your SageMaker AI Studio and Canvas environments. This allows you to restrict internet access, monitor and inspect traffic using standard AWS networking and security capabilities, and connect to other AWS resources through VPC endpoints. Please refer to Amazon SageMaker AI Canvas Getting started and Configure Amazon SageMaker AI Canvas in a VPC without internet access located in the *Amazon SageMaker AI Developer Guide* to create your SageMaker AI domain to connect to your Amazon DocumentDB cluster.

Configuring IAM access permissions for Amazon DocumentDB and SageMaker AI Canvas

An Amazon DocumentDB user that has `AmazonDocDBConsoleFullAccess` attached to their associated role and identity can access the AWS Management Console. Add the following actions to the aforementioned role or identity to provide access to no-code machine learning with Amazon SageMaker AI Canvas.

```
"sagemaker:CreatePresignedDomainUrl",
"sagemaker:DescribeDomain",
"sagemaker:ListDomains",
"sagemaker:ListUserProfiles"
```

Creating database users and roles for SageMaker AI Canvas

You can restrict access to the actions that users can perform on databases using role-based access control (RBAC) in Amazon DocumentDB. RBAC works by granting one or more roles to a user. These roles determine the operations that a user can perform on database resources.

As a Canvas user, you connect to a Amazon DocumentDB database with username and password credentials. You can create a database user/role for a Canvas user that has read access to the specific databases using Amazon DocumentDB RBAC functionality.

For example, use the `createUser` operation:

```
db.createUser({
user: "canvas_user",
pwd: "<insert-password>",
roles: [{role: "read", db: "sample-database-1"}]
})
```

This creates a `canvas_user` which has read permissions to the `sample-database-1` database. Your Canvas analysts can use this credential to access data in your Amazon DocumentDB cluster. Refer to Database access using Role-Based Access Control to learn more.

Available regions

The no-code integration is available in regions where both Amazon DocumentDB and Amazon SageMaker AI Canvas are supported. The regions include:

- us-east-1 (N. Virginia)
- us-east-2 (Ohio)
- us-west-2 (Oregon)
- ap-northeast-1 (Tokyo)
- ap-northeast-2 (Seoul)
- ap-south-1 (Mumbai)
- ap-southeast-1 (Singapore)
- ap-southeast-2 (Sydney)
- eu-central-1 (Frankfurt)
- eu-west-1 (Ireland)

Please refer to Amazon SageMaker AI Canvas in the *Amazon SageMaker AI Developer Guide* for the latest region availability.

Vector search for Amazon DocumentDB

Vector search is a method used in machine learning to find similar data points to a given data point by comparing their vector representations using distance or similarity metrics. The closer the two vectors are in the vector space, the more similar the underlying items are considered to be. This technique helps capture the semantic meaning of the data. This approach is useful in various applications, such as recommendation systems, natural language processing, and image recognition.

Vector search for Amazon DocumentDB combines the flexibility and rich querying capability of a JSON-based document database with the power of vector search. If you want to use your existing Amazon DocumentDB data or a flexible document data structure to build machine learning and generative AI use cases, such as semantic search experience, product recommendation, personalization, chatbots, fraud detection, and anomaly detection, then vector search for Amazon DocumentDB is an ideal choice for you. Vector search is available on Amazon DocumentDB 5.0 instance-based clusters.

Topics

- Inserting vectors
- Creating a vector index
- Getting an index definition
- Querying vectors
- Features and limitations
- Best practices

Inserting vectors

To insert vectors into your Amazon DocumentDB database, you can use existing insert methods:

Example

In the following example, a collection of five documents within a test database is created. Each document includes two fields: the product name and its corresponding vector embedding.

```
db.collection.insertMany([
  {"product_name": "Product A", "vectorEmbedding": [0.2, 0.5, 0.8]},
  {"product_name": "Product B", "vectorEmbedding": [0.7, 0.3, 0.9]},
  {"product_name": "Product C", "vectorEmbedding": [0.1, 0.2, 0.5]},
```

```
  {"product_name": "Product D", "vectorEmbedding": [0.9, 0.6, 0.4]},
  {"product_name": "Product E", "vectorEmbedding": [0.4, 0.7, 0.2]}
]);
```

Creating a vector index

Amazon DocumentDB supports both Hierarchical Navigable Small World (HNSW) indexing and Inverted File with Flat Compression (IVFFlat) indexing methods. An IVFFlat index segregates vectors into lists and subsequently searches a selected subset of those lists that are nearest to the query vector. On the other hand, an HNSW index organizes the vector data into a multi-layered graph. Although HNSW has slower build times compared to IVFFlat, it delivers better query performance and recall. Unlike IVFFlat, HNSW has no training step involved, allowing the index to be generated without any initial data load. For the majority of use cases, we recommend using the HNSW index type for vector search.

If you do not create a vector index, Amazon DocumentDB performs an exact nearest neighbor search, ensuring perfect recall. However, in production scenarios, speed is crucial. We recommend using vector indexes, which may trade some recall for improved speed. It's important to note that adding a vector index can lead to different query results.

Templates

You can use the following `createIndex` or `runCommand` templates to build a vector index on a vector field:

Using createIndex

In certain drivers, such as mongosh and Java, using the `vectorOptions` parameters in `createIndex` may result in an error. In such cases, we recommend using `runCommand`:

```
db.collection.createIndex(
  { "<vectorField>": "vector" },
  { "name": "<indexName>",
    "vectorOptions": {
      "type": " <hnsw> | <ivfflat> ",
      "dimensions": <number_of_dimensions>,
      "similarity": " <euclidean> | <cosine> | <dotProduct> ",
      "lists": <number_of_lists> [applicable for IVFFlat],
      "m": <max number of connections> [applicable for HNSW],
      "efConstruction": <size of the dynamic list for index build> [applicable for
 HNSW]
```

```
    }
  }
);
```

Using runCommand

In certain drivers, such as mongosh and Java, using the `vectorOptions` parameters in `createIndex` may result in an error. In such cases, we recommend using `runCommand`:

```
db.runCommand(
  { "createIndexes": "<collection>",
  "indexes": [{
      key: { "<vectorField>": "vector" },
      vectorOptions: {
          type: " <hnsw> | <ivfflat> ",
          dimensions: <number of dimensions>,
          similarity: " <euclidean> | <cosine> | <dotProduct> ",
          lists: <number_of_lists> [applicable for IVFFlat],
          m: <max number of connections> [applicable for HNSW],
          efConstruction: <size of the dynamic list for index build> [applicable for
 HNSW]
          },
      name: "myIndex"
      }]
  }
);
```

Parameter	Requirement	Data type	Description	Value(s)
name	optional	string	Specifies the name of the index.	Alphanumeric
type	optional		Specifies the type of index.	Supported: hnsw or ivfflat Default: HNSW (engine patch 3.0.4574 onwards)

Parameter	Requirement	Data type	Description	Value(s)
`dimensions`	required	integer	Specifies the number of dimensions in the vector data.	Maximum of 2,000 dimension s.
`similarity`	required	string	Specifies the distance metric used for the similarity calculation.	• **`euclidean`** • **`cosine`** • **`dotProduct`**
`lists`	required for IVFFlat	integer	Specifies the number of clusters that the IVFFlat index uses to group the vector data. The recommend ed setting is the # of documents /1000 for up to 1M documents and `sqrt(# of documents)` for over 1M documents.	Minimum: 1 Maximum: Refer to the lists per instance type table in Features and limitations below.
`m`	optional	integer	Specifies the max number of connections for an HNSW index	Default: 16 Range [2, 100]

Parameter	Requirement	Data type	Description	Value(s)
`efConstruction`	optional	integer	Specifies the size of the dynamic candidate list for constructing the graph for HNSW index. `efConstruction` must be greater than or equal to (2 * m)	Default: 64 Range [4, 1000]

It is important that you set the value of sub-parameters such as `lists` for IVFFlat and `m` and `efConstruction` for HNSW appropriately as it will affect the accuracy/recall, build time, and performance of your search. A higher list value increases the speed of the query as it reduces the number of vectors in each list, resulting in smaller regions. However, a smaller region size may lead to more recall errors, resulting in lower accuracy. For HNSW, increasing the value of `m` and `efConstruction` increases the accuracy, but also increases index build time and size. See the following examples:

Examples

HNSW

```
db.collection.createIndex(
  { "vectorEmbedding": "vector" },
  { "name": "myIndex",
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 3,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }
);
```

IVFFlat

```
db.collection.createIndex(
  { "vectorEmbedding": "vector" },
  { "name": "myIndex",
    "vectorOptions": {
      "type": "ivfflat",
      "dimensions": 3,
      "similarity": "euclidean",
      "lists":1
    }
  }
)
```

Getting an index definition

You can view the details of your indexes, including vector indexes, using the `getIndexes` command:

Example

```
db.collection.getIndexes()
```

Example output

```
[
 {
  "v" : 4,
  "key" : {
   "_id" : 1
  },
  "name" : "_id_",
  "ns" : "test.collection"
 },
 {
  "v" : 4,
  "key" : {
   "vectorEmbedding" : "vector"
  },
  "name" : "myIndex",
  "vectorOptions" : {
```

```
    "type" : "ivfflat",
    "dimensions" : 3,
    "similarity" : "euclidean",
    "lists" : 1
   },
   "ns" : "test.collection"
  }
]
```

Querying vectors

Vector query template

Use the following template to query a vector:

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": <query vector>,
        "path": "<vectorField>",
        "similarity": "<distance metric>",
        "k": <number of results>,
        "probes":<number of probes> [applicable for IVFFlat],
        "efSearch":<size of the dynamic list during search> [applicable for HNSW]
      }
    }
  }
]);
```

Parameter	Requirement	Type	Description	Value(s)
`vectorSearch`	required	operator	Used inside $search command to query the vectors.	
`vector`	required	array	Indicates the query vector that will be used	

Parameter	Requirement	Type	Description	Value(s)
			to find similar vectors.	
`path`	required	string	Defines the name of the vector field.	
k	required	integer	Specifies the number of results that the search returns.	
`similarity`	required	string	Specifies the distance metric used for the similarity calculation.	• **`euclidean`** • **`cosine`** • **`dotProduct`**

Parameter	Requirement	Type	Description	Value(s)
`probes`	optional	integer	The number of clusters you want vector search to inspect. A higher value provides better recall at the cost of speed. It can be set to the number of lists for exact nearest neighbor search (at which point the planner won't use the index). The recommended setting to start fine-tuning is `sqrt(# of lists)`.	Default: 1
`efSearch`	optional	integer	Specifies the size of the dynamic candidate list that HNSW index uses during search. A higher value of `efSearch` provides better recall at cost of speed.	Default: 40 Range [1, 1000]

It is important to fine tune the value of `efSearch` (HNSW) or `probes` (IVFFlat) to achieve your desired performance and accuracy. See the following example operations:

HNSW

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": [0.2, 0.5, 0.8],
        "path": "vectorEmbedding",
        "similarity": "euclidean",
        "k": 2,
        "efSearch": 40
      }
    }
  }
]);
```

IVFFlat

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": [0.2, 0.5, 0.8],
        "path": "vectorEmbedding",
        "similarity": "euclidean",
        "k": 2,
        "probes": 1
      }
    }
  }
]);
```

Example output

Output from this operation looks something like the following:

```
{ "_id" : ObjectId("653d835ff96bee02cad7323c"), "product_name" : "Product A",
 "vectorEmbedding" : [ 0.2, 0.5, 0.8 ] }
```

```
{ "_id" : ObjectId("653d835ff96bee02cad7323e"), "product_name" : "Product C",
 "vectorEmbedding" : [ 0.1, 0.2, 0.5 ] }
```

Features and limitations

Version compatibility

- Vector search for Amazon DocumentDB is only available on Amazon DocumentDB 5.0 instance-based clusters.

Vectors

- Amazon DocumentDB can index vectors of up to 2,000 dimensions. However, up to 16,000 dimensions can be stored without an index.

Indexes

- For IVFFlat index creation, the recommended setting for lists parameter is the number of documents/1000 for up to 1M documents and `sqrt(# of documents)` for over 1M documents. Due to a working memory limit, Amazon DocumentDB supports a certain maximum value of the lists parameter depending on the number of dimensions. For your reference, the following table provides the maximum values of lists parameter for vectors of 500, 1000, and 2,000 dimensions:

Instance type	Lists with 500 dimensions	Lists with 1000 dimensions	Lists with 2000 dimensions
t3.med	372	257	150
r5.l	915	741	511
r5.xl	1,393	1,196	901
r5.2xl	5,460	5,230	4,788
r5.4xl	7,842	7,599	7,138
r5.8xl	11,220	10,974	10,498

Instance type	Lists with 500 dimensions	Lists with 1000 dimensions	Lists with 2000 dimensions
r5.12xl	13,774	13,526	13,044
r5.16xl	15,943	15,694	15,208
r5.24xl	19,585	19,335	18,845

- No other index options such as `compound`, `sparse` or `partial` are supported with vector indexes.
- Parallel index build is not supported for HNSW index. It is only supported for IVFFlat index.

Vector query

- For vector search query, it is important to fine tune the parameters such as `probes` or `efSearch` for optimum results. The higher the value of `probes` or `efSearch` parameter, the higher the recall and lower the speed. The recommended setting to start fine tuning the probes parameter is `sqrt(# of lists)`.

Best practices

Learn best practices for working with vector search in Amazon DocumentDB. This section is continually updated as new best practices are identified.

- Inverted File with Flat Compression (IVFFlat) index creation involves clustering and organizing the data points based on similarities. Hence, in order for an index to be more effective, we recommend that you at least load some data before creating the index.
- For vector search queries, it is important to fine tune the parameters such as `probes` or `efSearch` for optimum results. The higher the value of the `probes` or `efSearch` parameter, the higher is the recall and lower is the speed. The recommended setting to start fine tuning the `probes` parameter is `sqrt(lists)`.

Resources

- Vector search what's new blog post
- Semantic search code sample

- Amazon DocumentDB vector search code samples

Migrating to Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) is a fully managed database service that is compatible with the MongoDB API. You can migrate your data to Amazon DocumentDB from MongoDB databases running on premises or on Amazon Elastic Compute Cloud (Amazon EC2) using the process detailed in this section.

Topics

- Migration tools
- Discovery
- Planning: Amazon DocumentDB cluster requirements
- Migration approaches
- Migration sources
- Migration connectivity
- Testing
- Performance testing
- Failover testing
- Additional resources
- Migration playbook: MongoDB to Amazon DocumentDB

Migration tools

To migrate to Amazon DocumentDB, the two primary tools that most customers use are the AWS Database Migration Service (AWS DMS) and command line utilities like `mongodump` and `mongorestore`. As a best practice, and for either of these options, we recommend that you first create indexes in Amazon DocumentDB before beginning your migration as it can reduce the overall time and increase the speed of the migration. To do this, you can use the Amazon DocumentDB Index Tool.

AWS Database Migration Service

AWS Database Migration Service (AWS DMS) is a cloud service that makes it easy to migrate relational databases and non-relational databases to Amazon DocumentDB. You can use AWS DMS

to migrate your data to Amazon DocumentDB from databases hosted on-premises or on EC2. With AWS DMS, you can perform one-time migrations, or you can replicate ongoing changes to keep sources and targets in sync.

For more information on using AWS DMS to migrate to Amazon DocumentDB, please see:

- Using MongoDB as a Source for AWS DMS
- Using Amazon DocumentDB as a Target for AWS Database Migration Service
- Walkthrough: Migrating from MongoDB to Amazon DocumentDB

Command line utilities

Common utilities for migrating data to and from Amazon DocumentDB include `mongodump`, `mongorestore`, `mongoexport`, and `mongoimport`. Typically, `mongodump` and `mongorestore` are the most efficient utilities as they dump and restore data from your databases in a binary format. This is generally the best option and yields a smaller data size compared to logical exports. `mongoexport` and `mongoimport` are useful if you want to export and import data in a logical format like JSON or CSV as the data is human readable but is generally slower than the `mongodump`/`mongorestore` and yields a larger data size.

The Migration approaches section below will discuss when it is best to use AWS DMS and command line utilities based on your use case and requirements.

Discovery

For each of your MongoDB deployments, you should identify and record two sets of data: *Architecture Details* and *Operational Characteristics*. This information will help you choose the appropriate migration approach and cluster sizing.

Architecture Details

- **Name**

  Choose a unique name for tracking this deployment.

- **Version**

Record the version of MongoDB that your deployment is running. To find the version, connect to a replica set member with the mongo shell and run the `db.version()` operation.

- **Type**

  Record whether your deployment is a standalone mongo instance, a replica set, or a sharded cluster.

- **Members**

  Record the hostnames, addresses, and ports of each cluster, replica set, or standalone member.

  For a clustered deployment, you can find shard members by connecting to a mongo host with the mongo shell and running the `sh.status()` operation.

  For a replica set, you can obtain the members by connecting to a replica set member with the mongo shell and running the `rs.status()` operation.

- **Oplog sizes**

  For replica sets or sharded clusters, record the size of the oplog for each replica set member. To find a member's oplog size, connect to the replica set member with the mongo shell and run the `ps.printReplicationInfo()` operation.

- **Replica set member priorities**

  For replica sets or sharded clusters, record the priority for each replica set member. To find the replica set member priorities, connect to a replica set member with the mongo shell and run the `rs.conf()` operation. The priority is shown as the value of the `priority` key.

- **TLS/SSL usage**

  Record whether Transport Layer Security (TLS)/Secure Sockets Layer (SSL) is used on each node for encryption in transit.

Operational Characteristics

- **Database statistics**

  For each collection, record the following information:

  - Name
  - Data size
  - Collection count

  To find the database statistics, connect to your database with the mongo shell and run the command `db.runCommand({dbstats: 1})`.

- **Collection statistics**

  For each collection, record the following information:

  - Namespace
  - Data size
  - Index count
  - Whether the collection is capped

- **Index statistics**

  For each collection, record the following index information:

  - Namespace
  - ID
  - Size
  - Keys

- TTL
- Sparse
- Background

To find the index information, connect to your database with the mongo shell and run the command `db.collection.getIndexes()`.

- **Opcounters**

  This information helps you understand your current MongoDB workload patterns (read-heavy, write-heavy, or balanced). It also provides guidance on your initial Amazon DocumentDB instance selection.

  The following are the key pieces of information to collect over the monitoring period (in counts/sec):

  - Queries
  - Inserts
  - Updates
  - Deletes

  You can obtain this information by graphing the output of the `db.serverStatus()` command over time. You can also use the *mongostat* tool to obtain instantaneous values for these statistics. However, with this option you run the risk of planning your migration on usage periods other than your peak load.

- **Network statistics**

  This information helps you understand your current MongoDB workload patterns (read-heavy, write-heavy, or balanced). It also provides guidance on your initial Amazon DocumentDB instance selection.

The following are the key pieces of information to collect over the monitoring period (in counts/sec):

- Connections
- Network bytes in
- Network bytes out

You can get this information by graphing the output of the `db.serverStatus()` command over time. You can also use the *mongostat* tool to obtain instantaneous values for these statistics. However, with this option you run the risk of planning your migration on usage periods other than your peak load.

Planning: Amazon DocumentDB cluster requirements

Successful migration requires that you carefully consider both your Amazon DocumentDB cluster's configuration and how applications will access your cluster. Consider each of the following dimensions when determining your cluster requirements:

- **Availability**

  Amazon DocumentDB provides high availability through the deployment of replica instances, which can be promoted to a primary instance in a process known as *failover*. By deploying replica instances to different Availability Zones, you can achieve higher levels of availability.

  The following table provides guidelines for Amazon DocumentDB deployment configurations to meet specific availability goals.

Availability Goal	Total Instances	Replicas	Availability Zones
99%	1	0	1

Availability Goal	Total Instances	Replicas	Availability Zones
99.9%	2	1	2
99.99%	3	2	3

Overall system reliability must consider all components, not just the database. For best practices and recommendations for meeting overall system reliability needs, see the AWS Well-Architected Reliability Pillar Whitepaper.

- **Performance**

  Amazon DocumentDB instances allow you to read from and write to your cluster's storage volume. Cluster instances come in a number of types, with varying amounts of memory and vCPU, which affect your cluster's read and write performance. Using the information you gathered in the discovery phase, choose an instance type that can support your workload performance requirements. For a list of supported instance types, see Managing instance classes.

  When choosing an instance type for your Amazon DocumentDB cluster, consider the following aspects of your workload's performance requirements:

  - **vCPUs**—Architectures that require higher connection counts might benefit from instances with more vCPUS.

  - **Memory**—When possible, keeping your working dataset in memory provides maximum performance. A starting guideline is to reserve a third of your instance's memory for the Amazon DocumentDB engine, leaving two-thirds for your working dataset.

  - **Connections**—The minimum optimal connection count is eight connections per Amazon DocumentDB instance vCPU. Although the Amazon DocumentDB instance connection limit is much higher, performance benefits of additional connections decline above eight connections per vCPU.

- **Network**—Workloads with a large number of clients or connections should consider the aggregate network performance required for inserted and retrieved data. Bulk operations can make more efficient use of network resources.

- **Insert Performance**—Single document inserts are generally the slowest way to insert data into Amazon DocumentDB. Bulk insert operations can be dramatically faster than single inserts.

- **Read Performance**—Reads from working memory are always faster than reads returned from the storage volume. Therefore, optimizing your instance memory size to retain your working set in memory is ideal.

In addition to serving reads from your primary instance, Amazon DocumentDB clusters are automatically configured as replica sets. You can then route read-only queries to read replicas by setting read preference in your MongoDB driver. You can scale read traffic by adding replicas, reducing the overall load on the primary instance.

It is possible to deploy Amazon DocumentDB replicas of different instance types in the same cluster. An example use case might be to stand up a replica with a larger instance type to serve temporary analytics traffic. If you deploy a mixed set of instance types, be sure to configure the failover priority for each instance. This helps ensure that a failover event always promotes a replica of sufficient size to handle your write load.

- **Recovery**

  Amazon DocumentDB continuously backs up your data as it is written. It provides point-in-time recovery (PITR) capabilities within a configurable period of 1–35 days, known as the *backup retention period*. The default backup retention period is one day. Amazon DocumentDB also automatically creates daily snapshots of your storage volume, which are also retained for the configured backup retention period.

If you want to retain snapshots beyond the backup retention period, you can also initiate manual snapshots at any time using the AWS Management Console and AWS Command Line Interface (AWS CLI). For more information, see Backing up and restoring in Amazon DocumentDB.

Consider the following as you plan your migration:

- Choose a backup retention period of 1–35 days that meets your recovery point objective (RPO).
- Decide if you require manual snapshots, and if so, at what interval.

Migration approaches

There are three primary approaches for migrating your data to Amazon DocumentDB.

> **Note**
>
> Although you can create indexes at any time in Amazon DocumentDB, it is faster overall to create your indexes before importing large datasets. As a best practice, we recommend that for each of the approaches below, you first create your indexes in Amazon DocumentDB before performing the migration. To do this, you can use the Amazon DocumentDB Index Tool.

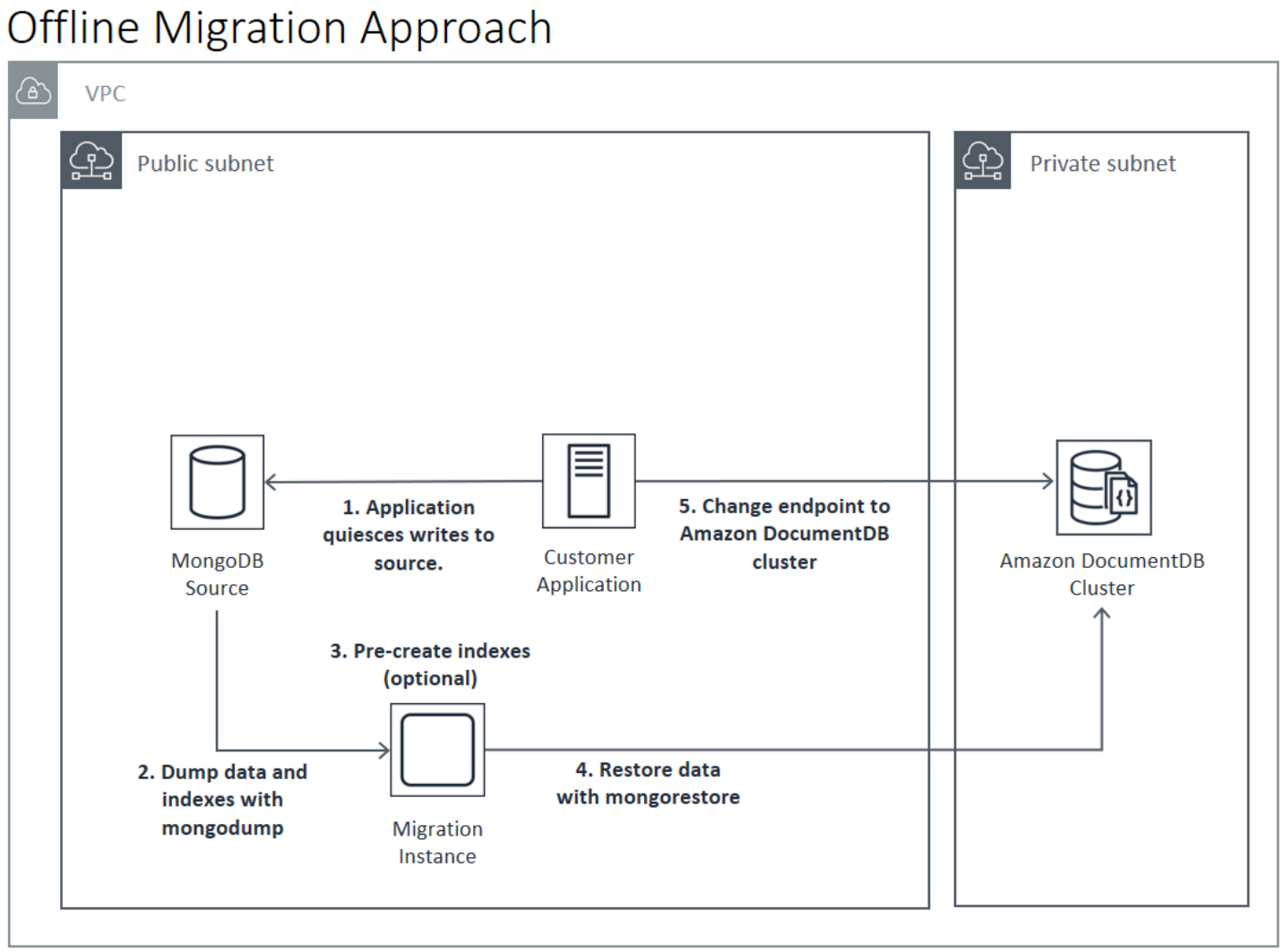
Offline

The *offline* approach uses the `mongodump` and `mongorestore` tools to migrate your data from your source MongoDB deployment to your Amazon DocumentDB cluster. The offline method is the simplest migration approach, but it also incurs the most downtime for your cluster.

The basic process for offline migration is as follows:

1. Quiesce writes to your MongoDB source.
2. Dump collection data and indexes from the source MongoDB deployment.

3. If you are migrating to an Elastic Cluster, create your sharded collections using the `sh.shardCollection()` command. If you are migrating to an instance-based cluster, skip to the next step.
4. Restore indexes to the Amazon DocumentDB cluster.
5. Restore collection data to the Amazon DocumentDB cluster.
6. Change your application endpoint to write to the Amazon DocumentDB cluster.



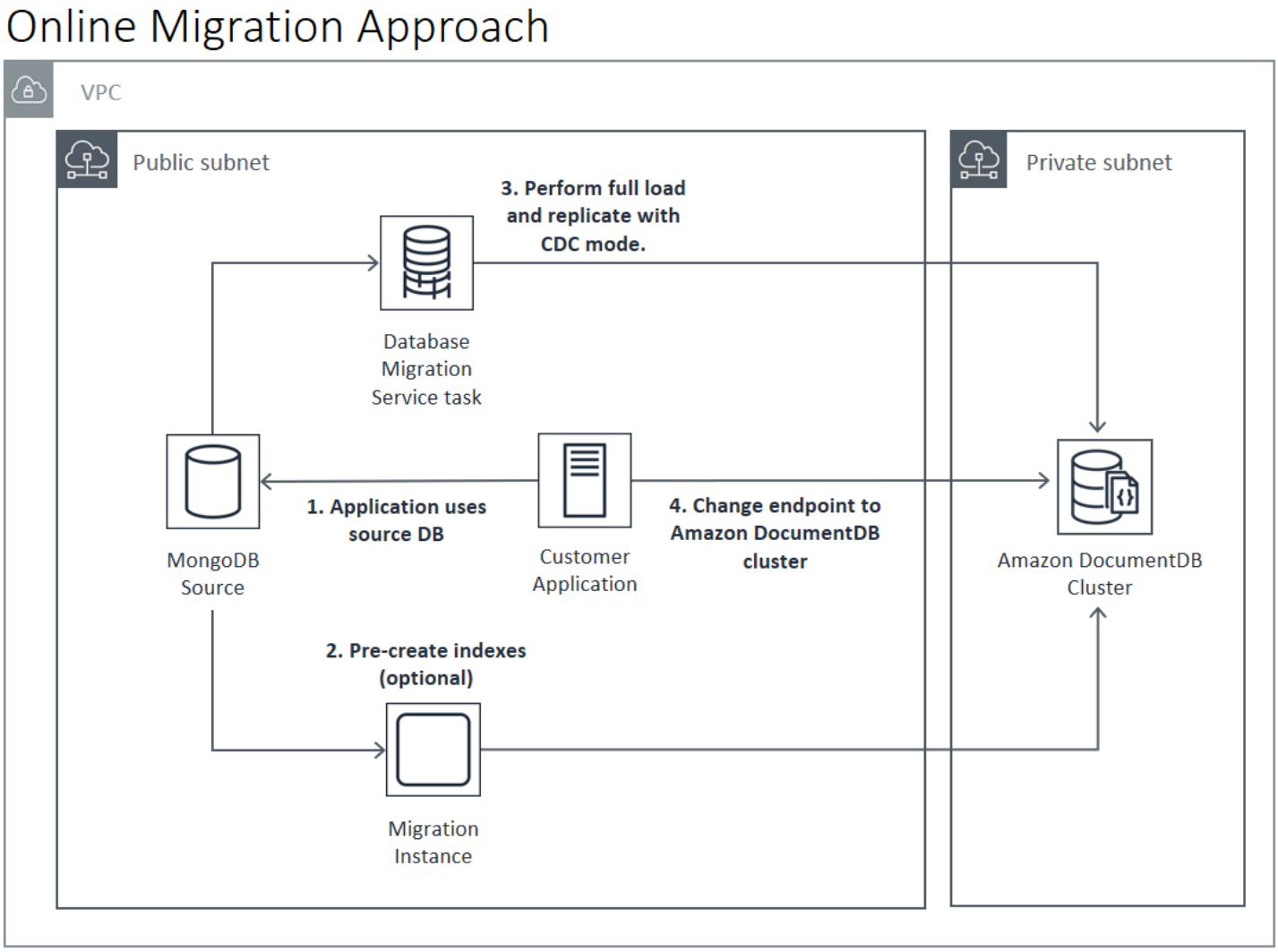
Online

The *online* approach uses AWS Database Migration Service (AWS DMS). It performs a full load of data from your source MongoDB deployment to your Amazon DocumentDB cluster. It then

switches to change data capture (CDC) mode to replicate changes. The online approach minimizes downtime for your cluster, but it is the slowest of the three methods.

The basic process for online migration is as follows:

1. Your application uses the source DB normally.
2. If you are migrating to an Elastic Cluster, create your sharded collections using the `sh.shardCollection()` command. If you are migrating to an instance-based cluster, skip to the next step.
3. Pre-create indexes in the Amazon DocumentDB cluster.
4. Create an AWS DMS task to perform a full load, and then enable CDC from the source MongoDB deployment to the Amazon DocumentDB cluster.
5. After the AWS DMS task has completed a full load and is replicating changes to the Amazon DocumentDB, switch the application's endpoint to the Amazon DocumentDB cluster.



For more information about using AWS DMS to migrate, see Using Amazon DocumentDB as a Target for AWS Database Migration Service and the related Tutorial in the *AWS Database Migration Service User Guide*.

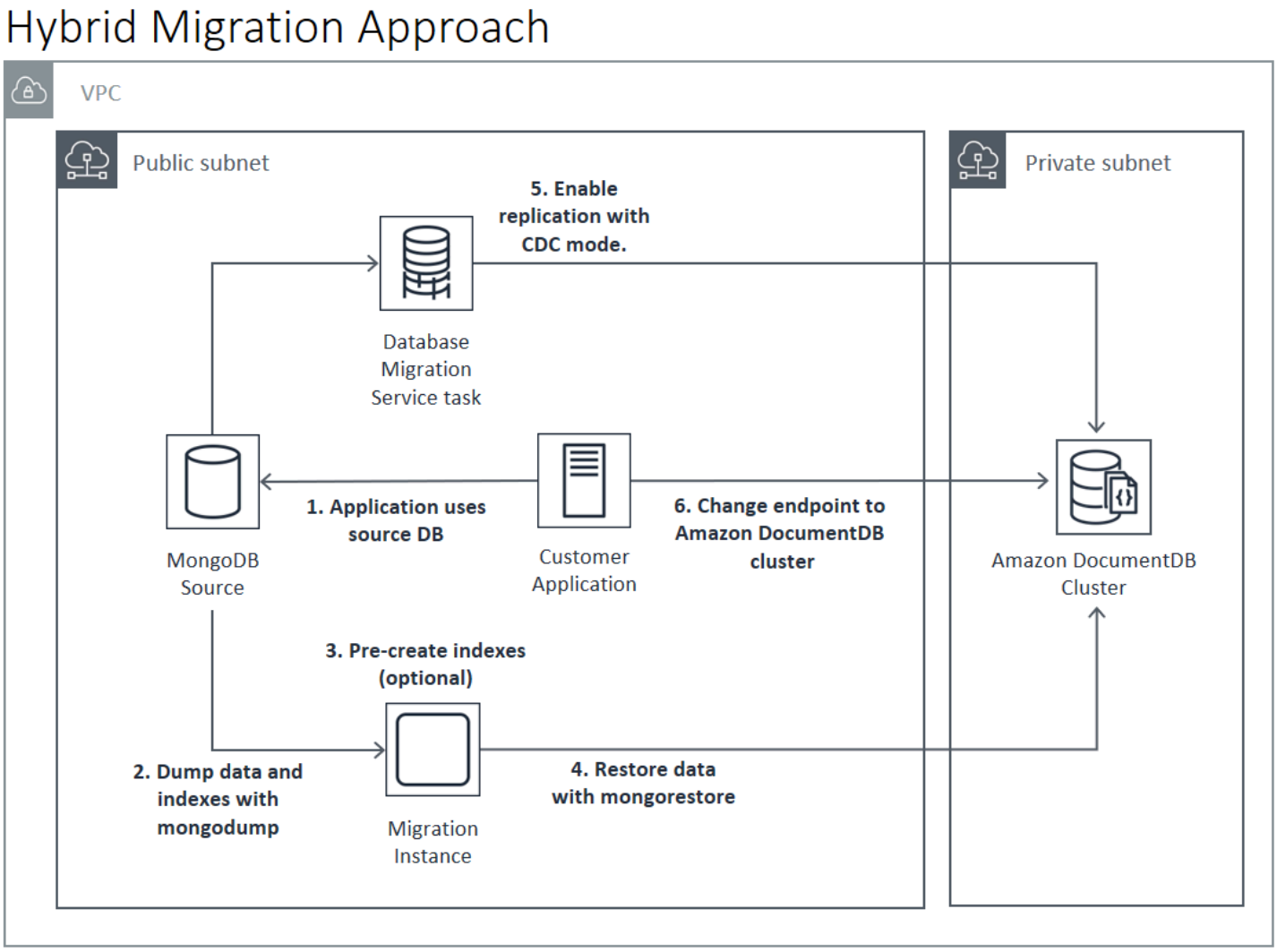
Hybrid

The *hybrid* approach uses the `mongodump` and `mongorestore` tools to migrate your data from your source MongoDB deployment to your Amazon DocumentDB cluster. It then uses AWS DMS in CDC mode to replicate changes. The hybrid approach balances migration speed and downtime, but it is the most complex of the three approaches.

The basic process for hybrid migration is as follows:

1. Your application uses the source MongoDB deployment normally.

2. Dump collection data and indexes from the source MongoDB deployment.
3. Restore indexes to the Amazon DocumentDB cluster.
4. If you are migrating to an Elastic Cluster, create your sharded collections using the `sh.shardCollection()` command. If you are migrating to an instance-based cluster, skip to the next step.
5. Restore collection data to the Amazon DocumentDB cluster.
6. Create an AWS DMS task to enable CDC from the source MongoDB deployment to the Amazon DocumentDB cluster.
7. When the AWS DMS task is replicating changes within an acceptable window, change your application endpoint to write to the Amazon DocumentDB cluster.



Regardless of the migration approach that you choose, it's most efficient to pre-create indexes in your Amazon DocumentDB cluster before migrating your data. This is because Amazon DocumentDB indexes are inserted data in parallel, but creating an index on existing data is a single-threaded operation.

Because AWS DMS does not migrate indexes (only your data), there is no extra step required to avoid creating indexes a second time.

Migration sources

If your MongoDB source is a standalone mongo process and you want to use the online or hybrid migration approaches, first convert your standalone mongo to a replica set so that the oplog is created to use as a CDC source.

If you are migrating from a MongoDB replica set or sharded cluster, consider creating a chained or hidden secondary for each replica set or shard to use as your migration source. Performing data dumps can force working set data out of memory and impact performance on production instances. You can reduce this risk by migrating from a node not serving production data.

Migration Source Versions

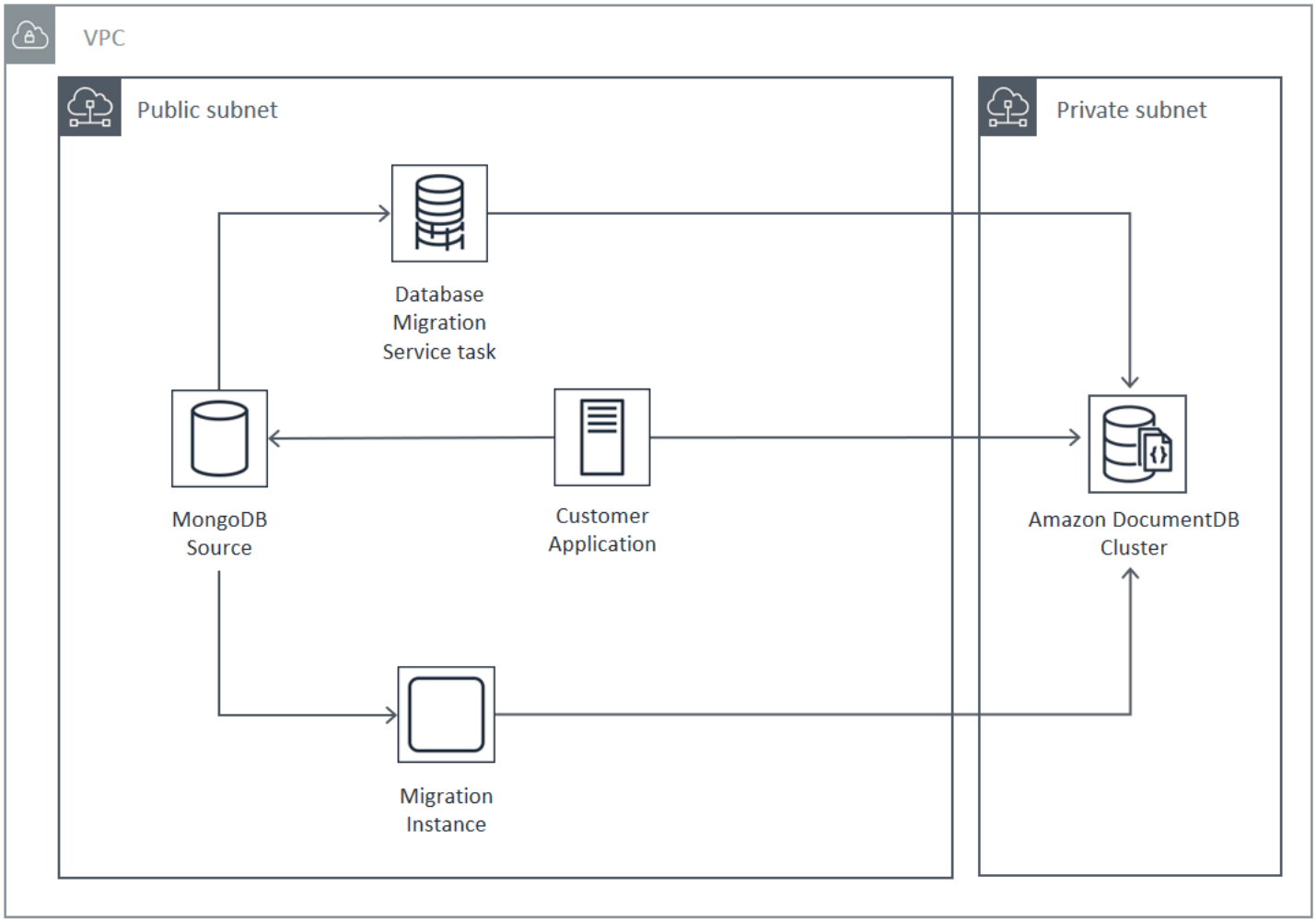
If your source MongoDB database version is different from the compatibility version of your destination Amazon DocumentDB cluster, you might need to take other preparation steps to ensure a successful migration. The two most common requirements encountered are the need to upgrade the source MongoDB installation to a supported version for migration (MongoDB version 3.0 or greater), and upgrading your application drivers to support the target Amazon DocumentDB version.

Ensure that if your migration has either of these requirements, you include those steps in your migration plan to upgrade and test any driver changes.

Migration connectivity

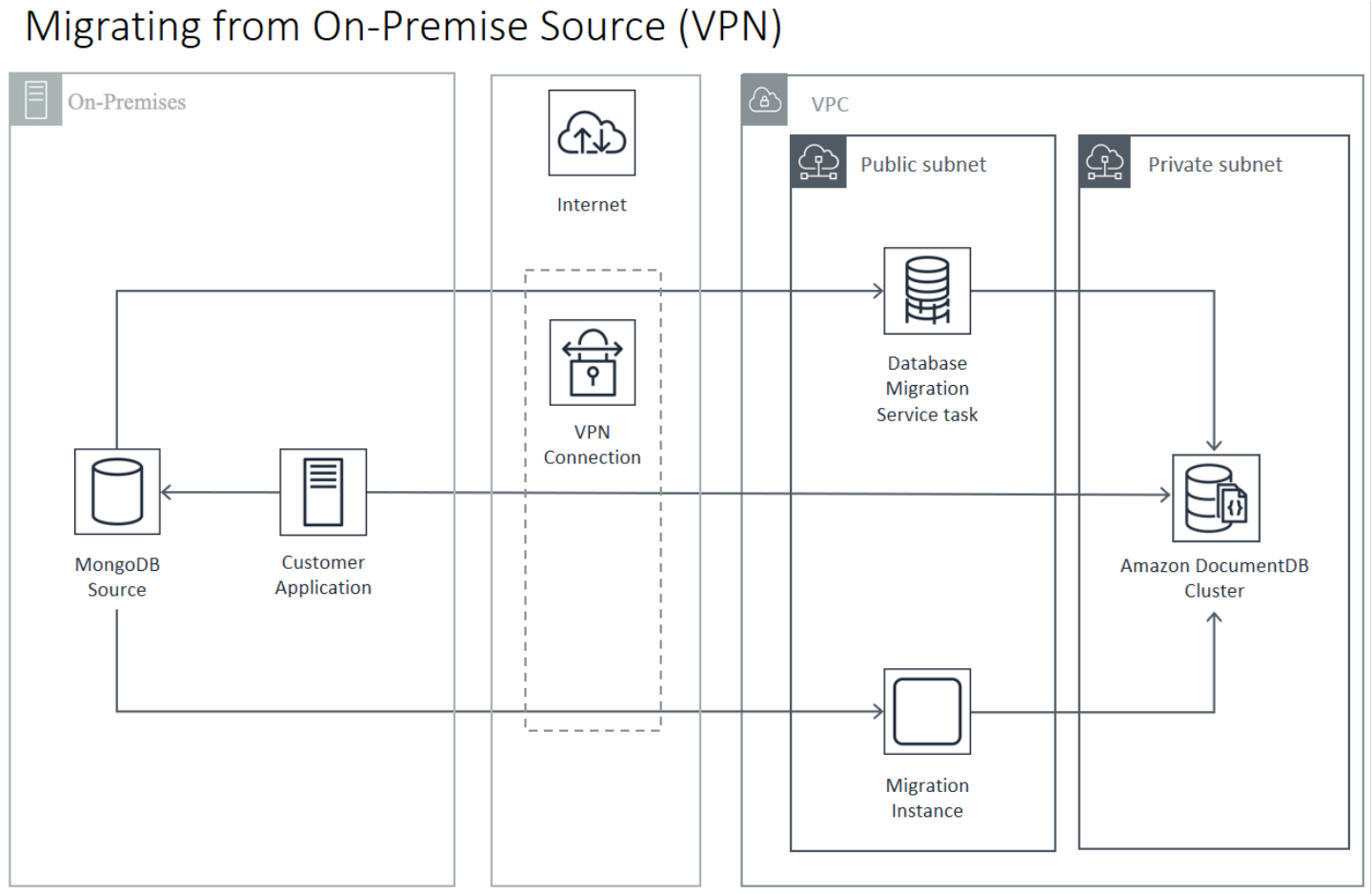
You can migrate to Amazon DocumentDB from a source MongoDB deployment running in your data center or from a MongoDB deployment running on an Amazon EC2 instance. Migrating from MongoDB running on EC2 is straightforward, and only requires that you correctly configure your security groups and subnets.

Migrating from EC2 Source



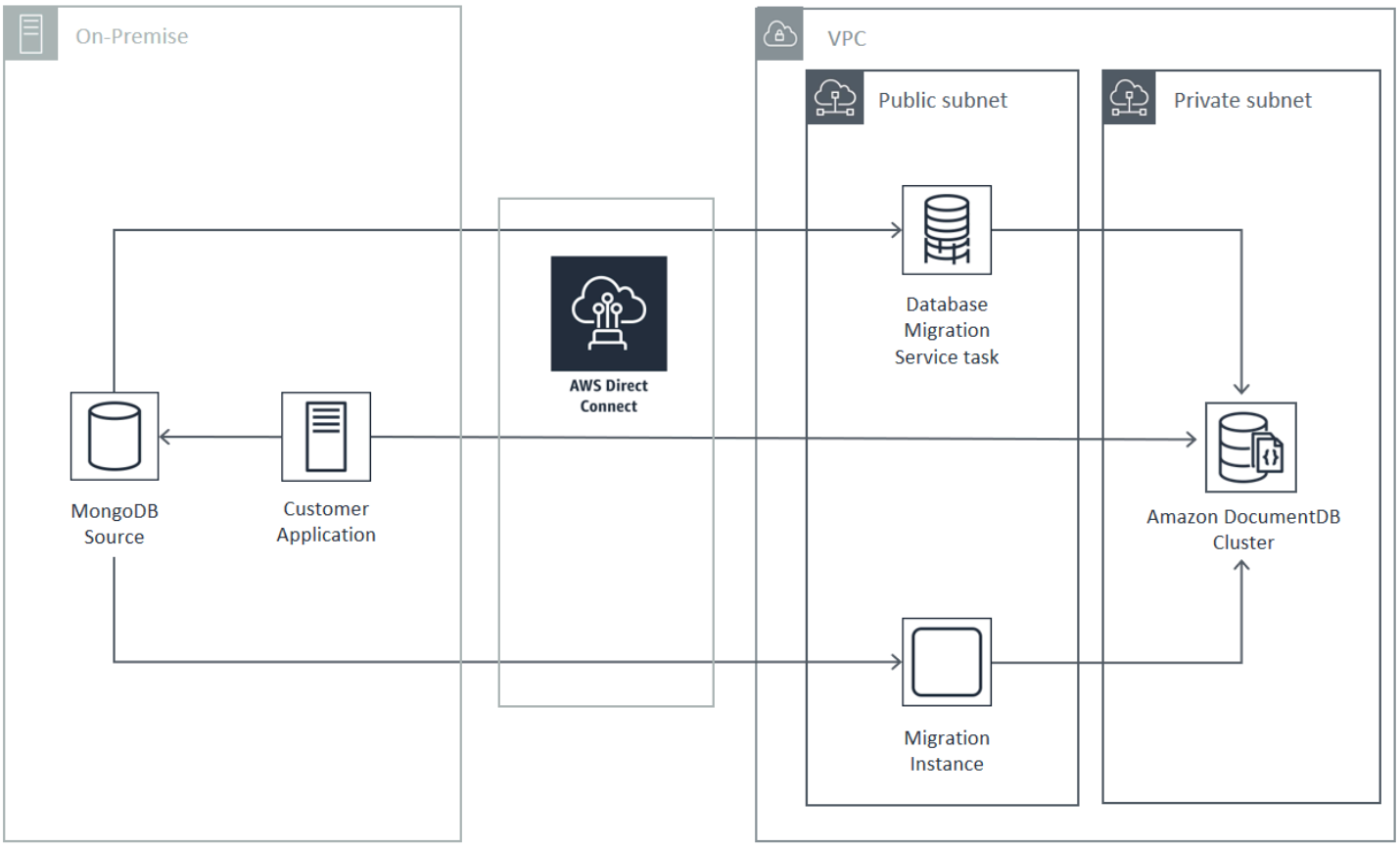
Migrating from an on-premises database requires connectivity between your MongoDB deployment and your virtual private cloud (VPC). You can accomplish this through a virtual private network (VPN) connection, or by using the AWS Direct Connect service. Although you can migrate over the internet to your VPC, this connection method is the least desirable from a security standpoint.

The following diagram illustrates a migration to Amazon DocumentDB from an on-premises source via a VPN connection.



The following represents a migration to Amazon DocumentDB from an on-premises source using AWS Direct Connect.

Migrating from On-Premise Source (Direct Connect)



Online and hybrid migration approaches require the use of an AWS DMS instance, which must run on Amazon EC2 in an Amazon VPC. All approaches require a migration server to run `mongodump` and `mongorestore`. It is generally easier to run the migration server on an Amazon EC2 instance in the VPC where your Amazon DocumentDB cluster is launched because it dramatically simplifies connectivity to your Amazon DocumentDB cluster.

Testing

The following are goals of pre-migration testing:

- Verify that your chosen approach achieves your desired migration outcome.
- Verify that your instance type and read preference choices meet your application performance requirements.
- Verify your application's behavior during failover.

Migration plan testing considerations

Consider the following when testing your Amazon DocumentDB migration plan.

Topics

- Restoring indexes
- Dumping data
- Restoring data
- Oplog sizing
- AWS Database Migration Service configuration
- Migrating from a sharded cluster

Restoring indexes

By default, `mongorestore` creates indexes for dumped collections, but it creates them after the data is restored. It is faster overall to create indexes in Amazon DocumentDB before data is restored to the cluster. This is because the indexing operations are parallelized during the data load.

If you choose to pre-create your indexes, you can skip the index creation step when restoring data with `mongorestore` by supplying the `--noIndexRestore` option.

Dumping data

The `mongodump` tool is the preferred method of dumping data from your source MongoDB deployment. Depending on the resources available on your migration instance, you might be able to speed up your `mongodump` by increasing the number of parallel connections dumped from the default 4 using the `--numParallelCollections` option.

Restoring data

The `mongorestore` tool is the preferred method for restoring dumped data to your Amazon DocumentDB instance. You can improve restore performance by increasing the number of workers for each collection during restore with the `--numInsertionWorkersPerCollection` option. One worker per vCPU on your Amazon DocumentDB cluster primary instance is a good place to start.

Amazon DocumentDB does not currently support the `mongorestore` tool's `--oplogReplay` option.

By default, `mongorestore` skips insert errors and continues the restore process. This can occur if you are restoring unsupported data to your Amazon DocumentDB instance. For example, it can happen if you have a document that contains keys or values with null strings. If you prefer to have the `mongorestore` operation fail entirely if any restore error is encountered, use the `--stopOnError` option.

Oplog sizing

The MongoDB operations log (`oplog`) is a capped collection that contains all data modifications to your database. You can view the size of the oplog and the time range it contains by running the `db.printReplicationInfo()` operation on a replica set or shard member.

If you are using the online or hybrid approaches, ensure that the oplog on each replica set or shard is large enough to contain all changes made during the entire duration of the data migration process (whether via `mongodump` or an AWS DMS task full load), plus a reasonable buffer. For more information, see *Check the Size of the Oplog* in the MongoDB documentation. Determine the minimum required oplog size by recording the elapsed time taken by the first test run of your `mongodump` or `mongorestore` process or AWS DMS full load task.

AWS Database Migration Service configuration

The *AWS Database Migration Service User Guide* covers the components and steps required to migrate your MongoDB source data to your Amazon DocumentDB cluster. The following is the basic process for using AWS DMS to perform an online or hybrid migration:

To perform a migration using AWS DMS

1. Create a MongoDB source endpoint. For more information, see Using MongoDB as a Source for AWS DMS.
2. Create an Amazon DocumentDB target endpoint. For more information, see Working with AWS DMS Endpoints.

   If you are configuring your target endpoint as an elastic cluster, note that your existing Amazon DocumentDB SSL certificate will not work with elastic clusters and you will need to attach a new SSL certificate to your endpoint using the following steps:

a. Visit https://www.amazontrust.com/repository/SFSRootCAG2.pem and save the contents as a "SFSRootCAG2.pem" file. This is the certificate file you will need to import in subsequent steps.

b. When creating the elastic cluster endpoint, under **Endpoint Configuration**, choose **Add new CA certificate**.

- For **Certificate identifier**, enter `SFSRootCAG2.pem`.
- For **Import certificate file**, choose **Choose file** and navigate to the `SFSRootCAG2.pem` file that you previously downloaded. Select and open the file. Choose **Import certificate**, then choose `SFSRootCAG2.pem` from the **Choose a certificate** drop down.

3. Create at least one AWS DMS replication instance. For more information, see Working with an AWS DMS Replication Instance.
4. Create at least one AWS DMS replication task. For more information, see Working with AWS DMS Tasks.

   For an online migration, your migration task uses the migration type **Migrate existing data and replicate ongoing changes**.

   For a hybrid migration, your migration task uses the migration type **Replicate data changes only**. You can choose the CDC start time to align with your dump time from your `mongodump` operation. The MongoDB oplog is idempotent. To avoid missing changes, it's a good idea to leave a few minutes worth of overlap between your `mongodump` finish time and your CDC start time.

Migrating from a sharded cluster

The process for migrating data from a MongoDB sharded cluster to your Amazon DocumentDB instance is essentially that of several replica set migrations in parallel. A key consideration when testing a sharded cluster migration is that some shards might be more heavily used than others. This situation leads to varying elapsed times for data migration. Ensure that you evaluate each shard's `oplog` requirements when planning and testing.

The following are some configuration issues to consider when migrating a sharded cluster:

- Before running `mongodump` or starting an AWS DMS migration task, you must disable the sharded cluster balancer and wait for any in-process migrations to complete. For more information, see *Disable the Balancer* in the MongoDB documentation.
- If you are using AWS DMS to replicate data, run the `cleanupOrphaned` command on each shard before running the migration tasks. If you don't run this command, the tasks might fail because of duplicate document IDs. Note that this command might affect performance. For more information, see *cleanupOrphaned* in the MongoDB documentation.
- If you are using the `mongodump` tool to dump data, you should run one `mongodump` process per shard. The most time-efficient approach might require multiple migration servers to maximize your dump performance.
- If you are using AWS Database Migration Service to replicate data, you must create a source endpoint for each shard. Also run at least one migration task for each shard that you are migrating. The most time-efficient approach might require multiple replication instances to maximize your migration performance.

Performance testing

After you successfully migrate your data to your test Amazon DocumentDB cluster, execute your test workload against the cluster. Verify through Amazon CloudWatch metrics that your performance meets or exceeds your MongoDB source deployment's current throughput.

Verify the following key Amazon DocumentDB metrics:

- Network throughput
- Write throughput
- Read throughput
- Replica lag

For more information, see Monitoring Amazon DocumentDB.

Failover testing

Verify that your application's behavior during an Amazon DocumentDB failover event meets your availability requirements. To initiate a manual failover of an Amazon DocumentDB cluster on the console, on the **Clusters** page, choose the **Failover** action on the **Actions** menu.

You can also initiate a failover by executing the `failover-db-cluster` operation from the AWS CLI. For more information, see `failover-db-cluster` in the Amazon DocumentDB section of the AWS CLI reference.

Additional resources

See the following topics in the *AWS Database Migration Service User Guide*:

- Using Amazon DocumentDB as a Target for AWS Database Migration Service
- Walkthrough: Migrating from MongoDB to Amazon DocumentDB

Migration playbook: MongoDB to Amazon DocumentDB

This migration playbook provides you with resources and steps to help you migrate from a MongoDB database to Amazon DocumentDB.

Migration process

Listed below are the high-level steps typically involved in migrating your data from a MongoDB database to Amazon DocumentDB.

Topics

- Step 1: Compatibility and functional differences
- Step 2: Proof of concept
- Step 3: Migrate the data
- Step 4: Data validation
- Step 5: Application cutover

Step 1: Compatibility and functional differences

Amazon DocumentDB interacts with the Apache 2.0 open-source MongoDB 3.6, 4.0, and 5.0 APIs. As a result, you can use the same MongoDB drivers, applications, and tools with Amazon DocumentDB with little or no changes.

The first step is to check for compatibility between the operators and indexes your application uses on your MongoDB database and their availability in Amazon DocumentDB, as well as to understand the functional differences between them.

Operators compatibility

Use the Amazon DocumentDB compatibility tool* to easily discover if your application uses any unsupported operators in its queries. This tool can scan your MongoDB database server log files or your application source code to provide a report of unsupported operators. If you find the usage of unsupported operators, you need to modify your application to work around unsupported operators.

To check compatibility between the MongoDB operators used in your setup and the supported Amazon DocumentDB operators, execute the following:

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
cd amazon-documentdb-tools/compat-tool/
python3 compat.py --version <Amazon DocumentDB version> --directory <mongodb logfiles/
source code>
```

For more information, see Supported MongoDB APIs, operations, and data types in Amazon DocumentDB.

* Not officially supported by AWS.

Indexes compatibility

You can use the Amazon DocumentDB index tool* to find out if you are using any index types unsupported in Amazon DocumentDB. This tool needs a connection to your source database to read index definitions.

For this, you first need to dump index definitions into a directory using the `--dump-indexes` option. Then run the tool with the `--show-issues` option, providing the directory to locate incompatible indexes.

Export indexes:

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
sudo pip install -r amazon-documentdb-tools/index-tool/requirements.txt
mkdir <directory to dump index definitions>
python3 migrationtools/documentdb_index_tool.py --dump-indexes --dir <directory> --uri
 <source-mongodb-uri>
```

Check for incompatible indexes:

```
python3 migrationtools/documentdb_index_tool.py --show-issues --dir <dumped-index-
definitions-directory>
```

If you find the usage of any unsupported index types, you must modify your application or data model to work around or continue without the incompatible indexes.

For more information on supported index types and properties in Amazon DocumentDB, see Indexes and index properties and How to index on Amazon DocumentDB.

* Not officially supported by AWS.

Functional differences

Review Functional differences with MongoDB to get familiar with the differences.

Step 2: Proof of concept

Perform a proof of concept by running your application or your regular test suite on Amazon DocumentDB to test for functionality and performance. You may need to populate your Amazon DocumentDB cluster with data to perform the tests. For example, you can use the `mongodump` and `mongorestore` tools to copy data from your source MongoDB.

Functional testing

Create an Amazon DocumentDB cluster (see Creating an Amazon DocumentDB cluster) and run your application or your functional test suite to validate if all the application workflows continue working seamlessly on Amazon DocumentDB.

Performance testing

Run performance tests on your application or performance test suite running on Amazon DocumentDB with a workload similar to your production workload to see if the setup meets your latency requirements. Fine-tune your workload for performance or scale your Amazon DocumentDB cluster as applicable. For more information, see Troubleshooting performance and resource utilization and Scaling Amazon DocumentDB clusters.

It is important to size your Amazon DocumentDB cluster with the right instance types for optimal performance. For more information, see best practices for Instance sizing.

Failover testing

You may want to observe how your application responds to an Amazon DocumentDB primary node reboot, a primary node failover, or a primary node deletion in a multi-node cluster, as well as when replica nodes are rebooted or removed. This will help you confirm that your application is resilient to these events. For more information, see Testing failover.

To understand the exceptions that an application should tolerate and how to handle them efficiently, see Building resilient applications with Amazon DocumentDB.

> **Note**
>
> There is no substitute for testing your workload on Amazon DocumentDB

Step 3: Migrate the data

After a successful proof of concept, migrate your data to Amazon DocumentDB. Most of our customers use online or offline migration approaches to migrate their data.

Online migration

Using the online migration method, you can migrate data from your source database, ranging from a few gigabytes to multiple terabytes, to Amazon DocumentDB with near-zero downtime. For more information, see AWS Database Migration Service (AWS DMS).

If you are migrating from a MongoDB database you can use AWS DMS to do a full load and replicate ongoing changes.

For a step-by-step process, see Migrating to Amazon DocumentDB with the online method.

Additional information can be found in the Using Amazon DocumentDB as a target for AWS Database Migration Service section in the *AWS Database Migration Service User Guide*.

Points to note with AWS DMS:

- Segmentation: When migrating multi-terabyte databases using AWS DMS, it may be slow with the default settings, as DMS's full load is single-threaded per collection by default, resulting in longer migration times. To speed up the full load for large database migrations, you can use the segmentation feature in AWS DMS.

  For more details on how to use segmentation with AWS DMS, see Using Auto Segmentation With AWS DMS.

- DMS Instance type: To speed up the data migration, you need to choose the right DMS instance.

Offline migration

Offline migration is the most straightforward approach to move databases to Amazon DocumentDB. This approach is mainly used for POCs and for workloads that can take write downtime during the migration.

For a step-by-step process, see Migrate from MongoDB to Amazon DocumentDB using the offline method.

Step 4: Data validation

Once the data is migrated successfully, validate the data for correctness to gain confidence. On the AWS DMS migration task console, you can find migrated data metrics. For more information, see verify migrated data.

You can also use the Amazon DocumentDB DataDiffer Tool* to validate the data consistency between the source and target collections.

* Not officially supported by AWS.

Step 5: Application cutover

This involves changing your application's database connection string to use your Amazon DocumentDB cluster.

For more information on connecting to Amazon DocumentDB, see Connecting to Amazon DocumentDB as a replica set.

Online Migration

After the full data load is done, AWS DMS continues to replicate ongoing changes from your source to Amazon DocumentDB. After the changes are caught up and your data validation checks are complete, you can perform a cutover to Amazon DocumentDB.

Offline Migration

Once done with the full data load and data validation checks, you can perform the cutover to Amazon DocumentDB.

Additional resources

Here are some additional resources that could help your migration:

- Video: Best practices for migrating to Amazon DocumentDB
- Video: Getting Started with Amazon DocumentDB Observability and Monitoring
- Additional utilities: Amazon DocumentDB Tools*
- Migration developer guide: Migrating to Amazon DocumentDB

* Not officially supported by AWS.

Amazon DocumentDB in-place major version upgrade

Amazon DocumentDB makes new versions of database engines generally available only after extensive testing. You can choose how and when to upgrade your Amazon DocumentDB clusters to the new version.

Currently, Amazon DocumentDB supports three major versions: Amazon DocumentDB 3.6, 4.0, and 5.0. You can perform an in-place major version upgrade (MVU) of your database while keeping the same endpoints, storage, and tags of the clusters and can continue using your applications without any modifications. This feature is available for free in all regions where Amazon DocumentDB 5.0 is available.

> **Important**
>
> Your Amazon DocumentDB clusters will be unavailable during the in-place major version upgrade and your clusters will experience multiple reboots. Please refrain from connecting, reading, or writing to the cluster after starting the upgrade. Upgrade downtime can vary from cluster to cluster depending on number of collections, indexes, databases, and instances. We recommend performing the upgrade during your maintenance window or during low utilization hours. Once your cluster has been upgraded, you cannot downgrade the cluster to previous version, but you can choose to restore your pre-upgrade snapshot to a new cluster.

Topics

- MVU prerequisites and limitations
- Best practices for in-place major version upgrades
- Performing an in-place major version upgrade
- Differences between Amazon DocumentDB 3.6/4.0 to 5.0 upgraded clusters and new Amazon DocumentDB 5.0 clusters
- Troubleshooting an in-place major version upgrade

MVU prerequisites and limitations

The following are prerequisites and limitations to in-place major version upgrade that you may need to understand and act on before performing the upgrade:

- **Instance Type** — Amazon DocumentDB 4.0/5.0 does not support r4.* instances. In order to proceed with an in-place major version upgrade, modify r4.* instances to r5.* instances. See Modifying an Amazon DocumentDB instance for more information. Please refer to Supported instance classes by region for supported instances based on the Amazon DocumentDB engine version.
- **Instance OS patches** — An in-place major version upgrade needs the latest operating system (OS) patch to proceed. Please apply any pending OS maintenance actions on the instances before proceeding with the in-place upgrade. For more information, see Amazon DocumentDB operating system updates.

> **Note**
>
> In some situations, if you have pending cluster level engine patches, instance OS patches are not visible. You may need to apply cluster level engine patches before proceeding with applying instance OS patches and, subsequently, the in-place major version upgrade. See Performing a patch update to a cluster's engine version.

- In-place major version upgrade is available in all regions where Amazon DocumentDB 5.0 is available.
- In-place major version upgrade is not supported with Amazon DocumentDB 4.0 as the target version.
- Starting in Amazon DocumentDB 4.0, "." in usernames is not supported. If you are upgrading from Amazon DocumentDB 3.6 to 5.0 and have a username containing ".", please re-create your username without ".", before proceeding with in-place MVU.
- In-place major version upgrade is not currently supported on Amazon DocumentDB global clusters and elastic clusters.

> **Note**
>
> To upgrade your global clusters, delete your secondary clusters from the global cluster, convert the primary cluster to a regional cluster, perform an in-place major version upgrade on the regional (primary) cluster, then recreate the global cluster by adding secondary clusters using the same name in order to retain the same endpoints as earlier. Note that you will incur IO charges while your upgraded primary cluster replicates data to your newly added secondary clusters. For detailed steps on how to remove secondary

clusters from global cluster before deleting, see Removing a cluster from an Amazon DocumentDB global cluster.

- If you have a large amount of indexes (>3,000) operating in burstable performance instances (e.g. t3.medium or t4g.medium), you must scale up your primary instance to a larger instance (for example, at least r5.large) to perform the in-place major version upgrade. You can choose to scale down the instance size once your in-place major version upgrade is complete. See the table below for the maximum number of indexes supported on the db.t3 and db.t4g instance types for an in-place major version upgrade:

Instance	Maximum indexes supported for in-place MVU
db.t4g.medium	3K
db.t3.medium	10K

Best practices for in-place major version upgrades

Topics

- Test in-place major version upgrades using cloned clusters
- Before an in-place major version upgrade
- During an in-place major version upgrade
- After an in-place major version upgrade

Test in-place major version upgrades using cloned clusters

1. To test in-place major version upgrades, we recommend using fast cloning feature to create a clone of your target cluster. You will not incur any storage costs for testing in-place major version upgrade on a cloned volume, unless you modify any data on the cluster. For more information on volume clone, see Cloning a volume for an Amazon DocumentDB cluster.
2. To get a more realistic estimate of the time taken to complete the in-place major version upgrade, match the instance count of the cloned cluster to the targeted cluster.

3. We recommend fully testing the newly upgraded Amazon DocumentDB 5.0 cluster for any functional differences to ensure everything is working as expected.

Before an in-place major version upgrade

1. Have a version-compatible cluster parameter group ready.

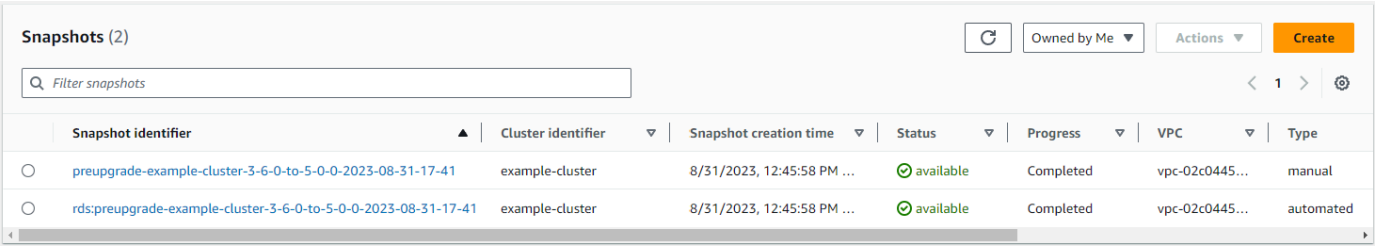
   Use the Amazon DocumentDB default cluster parameter group for the new engine version or create your own custom cluster parameter group for the new engine version.

   If you associate an Amazon DocumentDB cluster parameter group as a part of the upgrade request, the in-place major version upgrade will automatically reboot the cluster to apply the new parameter group.
2. Ensure that you've satisfied the prerequisites for an in-place major version upgrade as mentioned in the Prerequisites and limitations section.
3. Create a manual snapshot.

   The upgrade process creates a snapshot of your database cluster during upgrading. It is strongly recommended to create your own manual snapshot before the upgrade process. See Creating a manual cluster snapshot.

   > **Note**
   >
   > The auto snapshot created by the upgrade process will not be automatically deleted after the in-place major version upgrade has completed. This snapshot will not incur any charges as long as it is within the retention period. You can choose to delete this snapshot once you have verified a successful upgrade of your cluster.

   The snapshot is named as `preupgrade-<name>-<version>-<timestamp>`.



4. Check if you already scheduled an in-place major version upgrade of your cluster.

If you have modified the cluster and selected to apply it in the next maintenance window, in-place major version upgrade schedule will not be visible on console, but you can view it in the CLI. You can run the `describe-db-clusters` command to check if an in-place major version upgrade is already scheduled:

```
aws docdb describe-db-cluster \
  --region us-east-1 \
  --db-cluster-identifier mydocdbcluster
```

In the example above, replace each *`user input placeholder`* with your cluster's information.

The command returns the following output:

```
"PendingModifiedValues": {
    "EngineVersion": "5.0.0"
},
```

5. Perform multiple dry-runs using volume clone in lower environments to test the cluster post in-place major version upgrade on any execution plan and functional differences. We recommend cloning with the same number and size of instances to get a better estimate of in-place major version upgrade run time. For more information, see Cloning a volume for an Amazon DocumentDB cluster.
6. If the previous step is successful, proceed with in-place major version upgrade on the production cluster.

During an in-place major version upgrade

You can monitor progress of your in-place major version upgrade by subscribing to cluster maintenance events. When the upgrade completes, you will receive the “Database cluster major version has been upgraded” event. This and other events occurring during the upgrade appear in the ‘Events and Tags’ section of the cluster detail page in the Amazon DocumentDB console. The cluster status then changes from ‘upgrading’ to ‘available’.

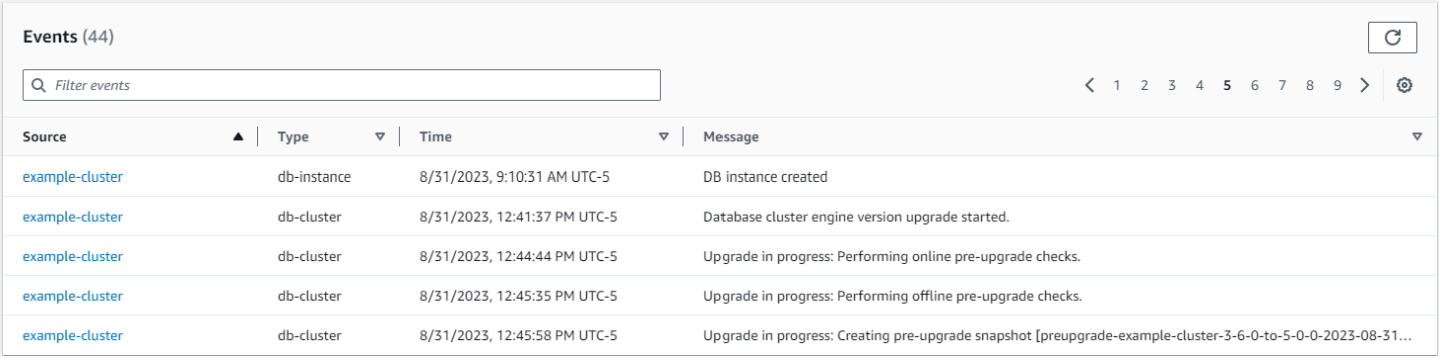
From CLI, you can run `aws docdb create-event-subscription` to create events and `aws docdb describe-events` to monitor progress. You can also setup event notifications for the

above events to Amazon SNS as the target to be notified via email, push messages, and other methods. For more information, see Subscribing to Amazon DocumentDB events.

In-place major version upgrade generates the following events during the upgrade:

- Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-<cluster-name>-<timestamp>]
- Upgrade in progress: Cloning volume.
- Upgrade in progress: Upgrading writer.
- Upgrade in progress: Upgrading readers.
- Database cluster major version has been upgraded.

Events are also visible on the console under the **Events** page:



Events (44)

Filter events

1 2 3 4 5 6 7 8 9

Source	Type	Time	Message
example-cluster	db-instance	8/31/2023, 9:10:31 AM UTC-5	DB instance created
example-cluster	db-cluster	8/31/2023, 12:41:37 PM UTC-5	Database cluster engine version upgrade started.
example-cluster	db-cluster	8/31/2023, 12:44:44 PM UTC-5	Upgrade in progress: Performing online pre-upgrade checks.
example-cluster	db-cluster	8/31/2023, 12:45:35 PM UTC-5	Upgrade in progress: Performing offline pre-upgrade checks.
example-cluster	db-cluster	8/31/2023, 12:45:58 PM UTC-5	Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-example-cluster-3-6-0-to-5-0-0-2023-08-31...

In the AWS CLI, you can run the `describe-events` command to track progress:

```
aws docdb describe-events
  --source-identifier mydocdbcluster
  --source-type db-cluster
```

In the example above, replace each *`user input placeholder`* with your cluster's information.

The command returns the following output:

```
{
    "Events": [
        {
            "SourceIdentifier": "mydocdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
```

```
                "maintenance"
            ],
            "Date": "2023-07-11T23:20:32.444000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:xxxx:cluster:mycluster"
        }
    ]
}
```

After an in-place major version upgrade

For Amazon DocumentDB 3.6, add a tag to the cluster to differentiate that the cluster was upgraded to Amazon DocumentDB 5.0 from Amazon DocumentDB 3.6 as opposed to a newly created Amazon DocumentDB 5.0 cluster. Refer to the section on differences between an upgraded Amazon DocumentDB 5.0 cluster and a new Amazon DocumentDB 5.0 cluster.

Take a manual snapshot after the in-place MVU finishes in case you need to restore to the post-upgrade state. The automatic snapshot process will resume as soon as in-place major version upgrade completes. The manual snapshot will not incur any charges as long as it is within the retention period.

To use the new features associated with Amazon DocumentDB 5.0, for example, client-side field level encryption, we recommend upgrading your driver version to the MongoDB 5.0 API version. For more information, see What's new in Amazon DocumentDB 5.0 for a list of Amazon DocumentDB 5.0 features.

> **Important**
>
> Immediately after performing in-place major version upgrade (MVU), your Amazon DocumentDB 5.0 cluster will repopulate the index metadata, based on which the database engine optimizes query execution plans. Expected query performance on your Amazon DocumentDB cluster will resume after the index metadata recalculation process is complete. Typically, this process completes in a few minutes but can last up to two hours depending on the number of indexes on your cluster. An immediate reboot, failover, or scale up/down of your writer instance after in-place MVU, may disrupt the index metadata calculation process on your cluster. After the in-place MVU completes, we recommend making such changes once you observe expected query performance on your Amazon DocumentDB 5.0 cluster.
>
> Additionally, after the in-place MVU completes, the available change stream data will be limited to the last 3 hours.

Please contact AWS support if you see this temporary performance drop persisting for more than two hours after in-place MVU.

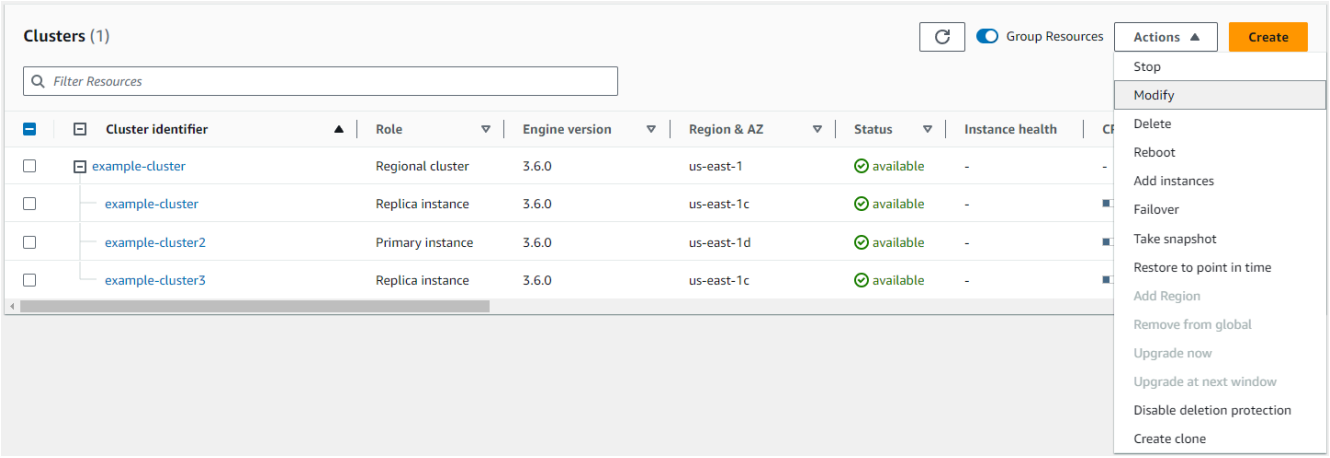
Fully test the upgraded Amazon DocumentDB 5.0 cluster to ensure everything is working as expected.

Performing an in-place major version upgrade

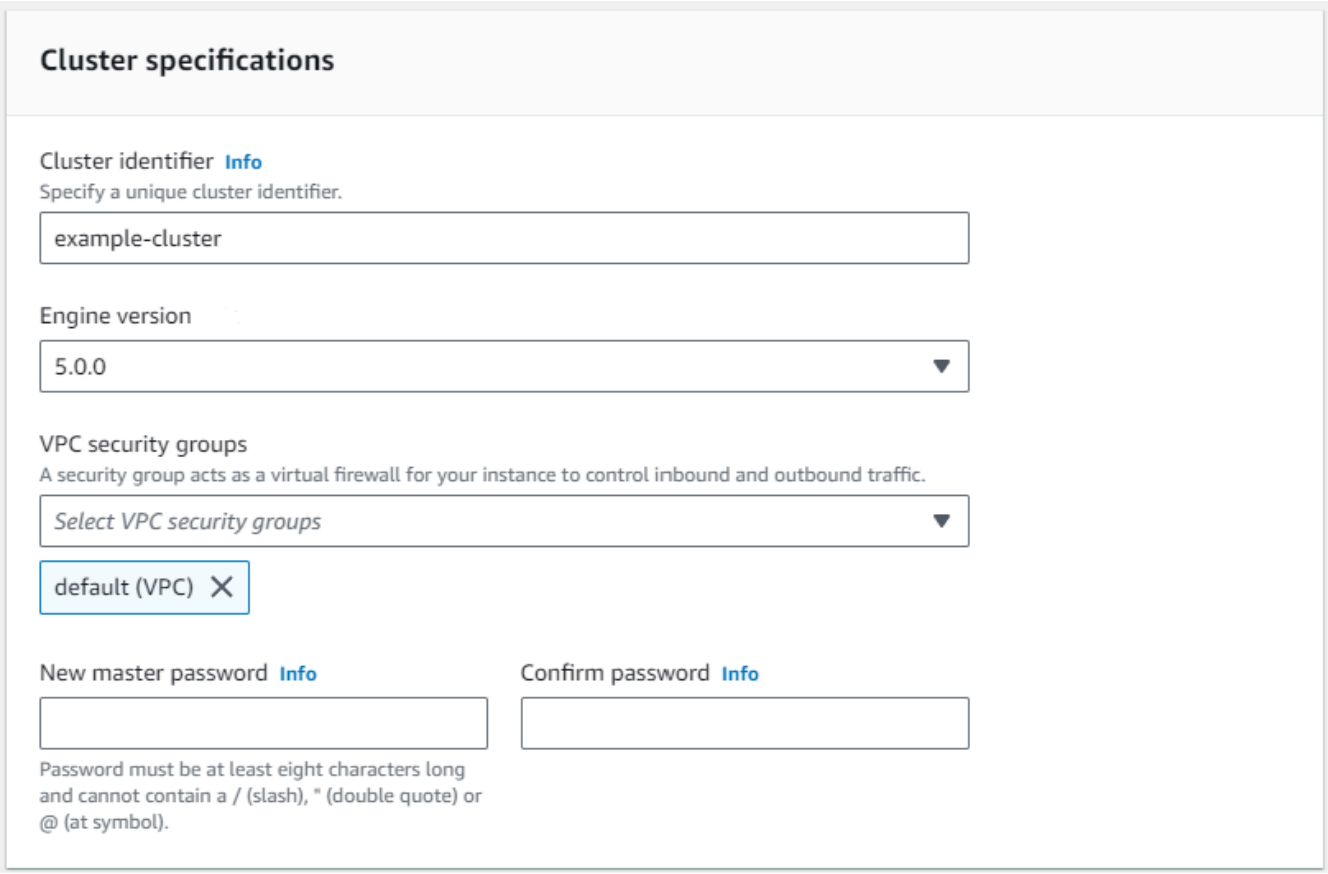
Using the AWS Management Console

To perform an in-place major version upgrade using the AWS Management Console:

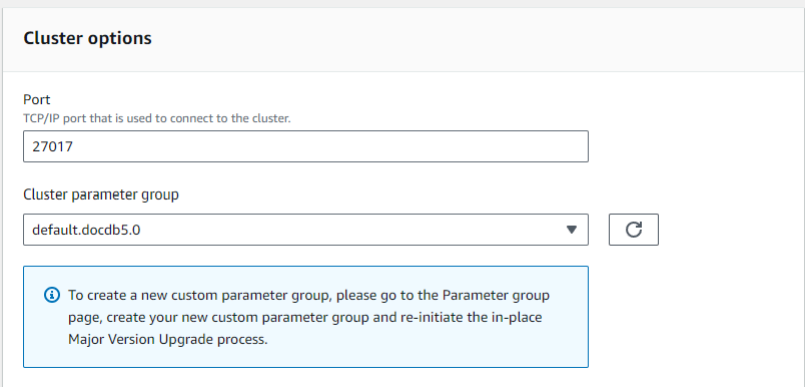
1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the **Clusters** table, select the source cluster, click **Actions**, and then **Modify**.



3. On the **Modify cluster** dialog in the **Cluster specifications** section, choose the targeted database version (**5.0.0**) from the **Engine version** drop down menu.

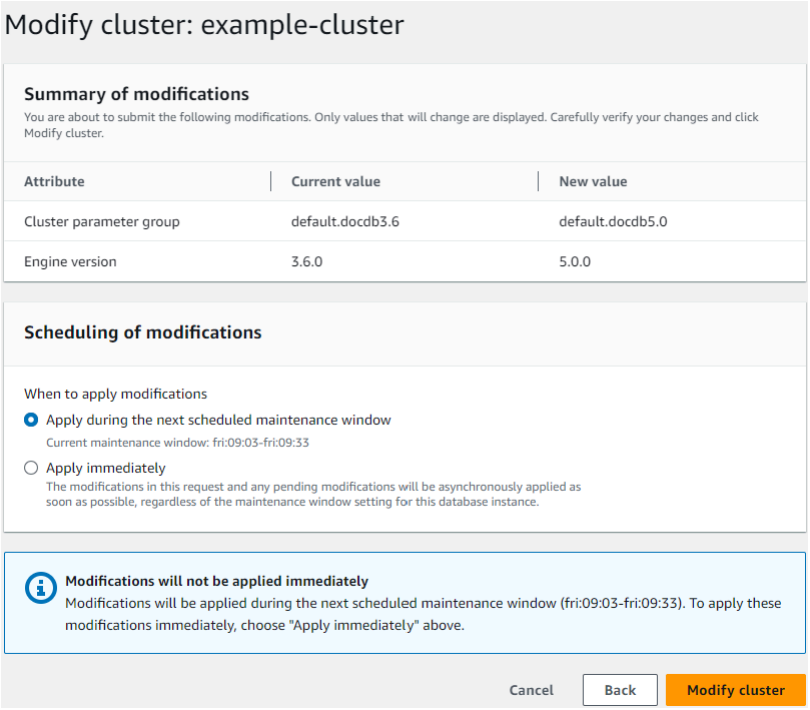


4. In the **Cluster options** section, choose the appropriate cluster parameter group (**default.docdb5.0**) or a custom created parameter group.

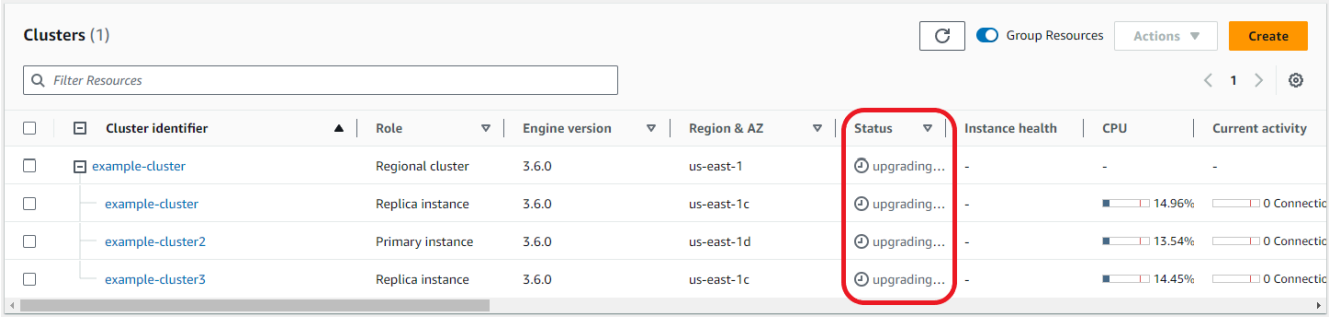


5. Once complete, scroll down and choose **Continue**.
6. In the **Scheduling of modifications** section, choose your preferred scheduling plan: apply immediately or apply in the next maintenance window.

   Then choose **Modify cluster**.



7. In the clusters table, note the status of your cluster as it is being upgraded:



Using the AWS CLI

Use the `modify-db-cluster` command with desired engine version option and `allow-major-version-upgrade` flag set:

```
aws docdb modify-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --allow-major-version-upgrade \
  --engine-version 5.0.0 \
  --apply-immediately \
  --cluster-parameter-group mydocdbparametergroup \
  --region us-east-1
```

In the example above, replace each *`user input placeholder`* with your cluster's information.

Differences between Amazon DocumentDB 3.6/4.0 to 5.0 upgraded clusters and new Amazon DocumentDB 5.0 clusters

- An in-place major version upgrade retains the original indexes on the upgraded cluster. With Amazon DocumentDB 5.0, we have enhanced the overall efficiency of index maintenance and the garbage collection process, especially for low cardinality indexes. As a general best practice, we recommend recreating your indexes using the reindex command after MVU successfully completes. Recreating indexes is not a requirement and will involve additional I/O. For more information, see Index maintenance using `reIndex`.
- Subdocument comparisons for multiple numeric data types:
  - If the cluster is migrated from Amazon DocumentDB 3.6, it will inherit the Amazon DocumentDB 3.6 subdocument comparison behavior. The functional difference is limited to numeric types (such as Long, Double, Decimal128) in a subdocument. For example, `{a: {b: {NumberLong(1)}}}` does not equal `{a: {b: 1}}` in Amazon DocumentDB 3.6, while they are compared as equal in Amazon DocumentDB 4.0 and after.
  - This subdocument comparison behavior exists only in Amazon DocumentDB 3.6, and in Amazon DocumentDB 5.0 clusters that were upgraded from version 3.6 using an in-place major version upgrade. This doesn't apply to newly created Amazon DocumentDB 5.0 clusters.

> **Note**
>
> For a list of functional differences between Amazon DocumentDB 3.6/4.0 and Amazon DocumentDB 5.0, see Amazon DocumentDB compatibility with MongoDB.

Troubleshooting an in-place major version upgrade

- In case of a failure, the in-place major version upgrade will attempt a rollback of the upgrade to assume the last operational state of the cluster before the upgrade started. A successful rollback will generate an event: "Database cluster is in a state that cannot be upgraded: DocumentDB cluster is in a state where major version upgrade cannot be completed successfully." At this point,

you should reach out to the AWS support team to troubleshoot and re-attempt the version upgrade. You can continue using your workload as before. In any other rare scenarios where the upgrade is taking longer than expected, please reach out to AWS support team for assistance.

- Once your in-place MVU completes successfully, your upgraded cluster may experience a temporary performance degradation and high CPU utilization for a small duration of time, while the index metadata refresh process is running. If you continue to experience performance degradation for more than 2 hours, please contact AWS support.

Upgrading your Amazon DocumentDB cluster using AWS Database Migration Service

> **Important**
>
> Amazon DocumentDB does not follow the same support lifecycles as MongoDB and MongoDB's end-of-life schedule does not apply to Amazon DocumentDB. There are no current plans for end-of-life for Amazon DocumentDB 3.6, and your existing MongoDB 3.6 drivers, applications, and tools, will continue to work with Amazon DocumentDB.

You can upgrade your Amazon DocumentDB cluster to a higher version with minimal downtime using AWS DMS. AWS DMS is a fully managed service that makes it easy to migrate from older Amazon DocumentDB versions, relational databases and non-relational databases to your target Amazon DocumentDB cluster.

Topics

- Step 1: Enable change streams
- Step 2: Modify the change streams retention duration
- Step 3: Migrate your indexes
- Step 4: Create an AWS DMS replication instance
- Step 5: Create an AWS DMS source endpoint
- Step 6: Create an AWS DMS target endpoint
- Step 7: Create and run a migration task
- Step 8: Changing the application endpoint to the target Amazon DocumentDB cluster

Step 1: Enable change streams

To perform a minimal downtime migration, AWS DMS requires access to the cluster's change streams. Amazon DocumentDB change streams provide a time-ordered sequence of update events that occur within your cluster's collections and databases. Reading from the change stream enables AWS DMS to perform change data capture (CDC) and apply incremental updates to the target Amazon DocumentDB cluster.

To enable change streams for all collections on a specific database, authenticate to your Amazon DocumentDB cluster using the mongo shell and execute the following commands:

```
db.adminCommand({modifyChangeStreams: 1,
    database: "db_name",
    collection: "",
    enable: true});
```

Step 2: Modify the change streams retention duration

Next, modify the change stream retention period based on how long you would like to retain change events in the change stream. For example, if you expect your Amazon DocumentDB cluster migration using AWS DMS to take 12 hours, you should set the change stream retention to a value greater than 12 hours. The default retention period for your Amazon DocumentDB cluster is three hours. You can modify the change stream log retention duration for your Amazon DocumentDB cluster to be between one hour and seven days using the AWS Management Console or the AWS CLI. For more details, refer to Modifying the Change Stream Log Retention Duration.

Step 3: Migrate your indexes

Create the same indexes on your target Amazon DocumentDB cluster that you have on your source Amazon DocumentDB cluster. Although AWS DMS handles the migration of data, it does not migrate indexes. To migrate the indexes, use the Amazon DocumentDB Index Tool to export indexes from the source Amazon DocumentDB cluster. You can get the tool by creating a clone of the Amazon DocumentDB tools GitHub repo and following the instructions in `README.md`. You can run the tool from an Amazon EC2 instance or an AWS Cloud9 environment running in the same Amazon VPC as your Amazon DocumentDB cluster.

In the following example, replace each *`user input placeholder`* with your own information.

The following code dumps indexes from your source Amazon DocumentDB cluster:

```
python migrationtools/documentdb_index_tool.py --dump-indexes
--uri mongodb://sample-user:user-password@sample-source-cluster.node.us-
east 1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
--dir ~/index.js/
```

```
2020-02-11 21:51:23,245: Successfully authenticated to database: admin2020-02-11
 21:46:50,432: Successfully connected to instance docdb-40-xx.cluster-xxxxxxxx.us-
east-1.docdb.amazonaws.com:27017
2020-02-11 21:46:50,432: Retrieving indexes from server...2020-02-11 21:46:50,440:
 Completed writing index metadata to local folder: /home/ec2-user/index.js/
```

Once your indexes are successfully exported, restore those indexes in your target Amazon DocumentDB cluster. To restore the indexes that you exported in the preceding step, use the Amazon DocumentDB Index Tool. The following command restores the indexes in your target Amazon DocumentDB cluster from the specified directory.

```
python migrationtools/documentdb_index_tool.py --restore-indexes
--uri mongodb://sample-user:user-password@sample-destination-
cluster.node.us-east 1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
--dir ~/index.js/

2020-02-11 21:51:23,245: Successfully authenticated to database: admin2020-02-11
 21:51:23,245: Successfully connected to instance docdb-50-xx.cluster-xxxxxxxx.us-
east-1.docdb.amazonaws.com:27017
2020-02-11 21:51:23,264: testdb.coll: added index: _id
```

To confirm that you restored the indexes correctly, connect to your target Amazon DocumentDB cluster with the mongo shell and list the indexes for a given collection. See the following code:

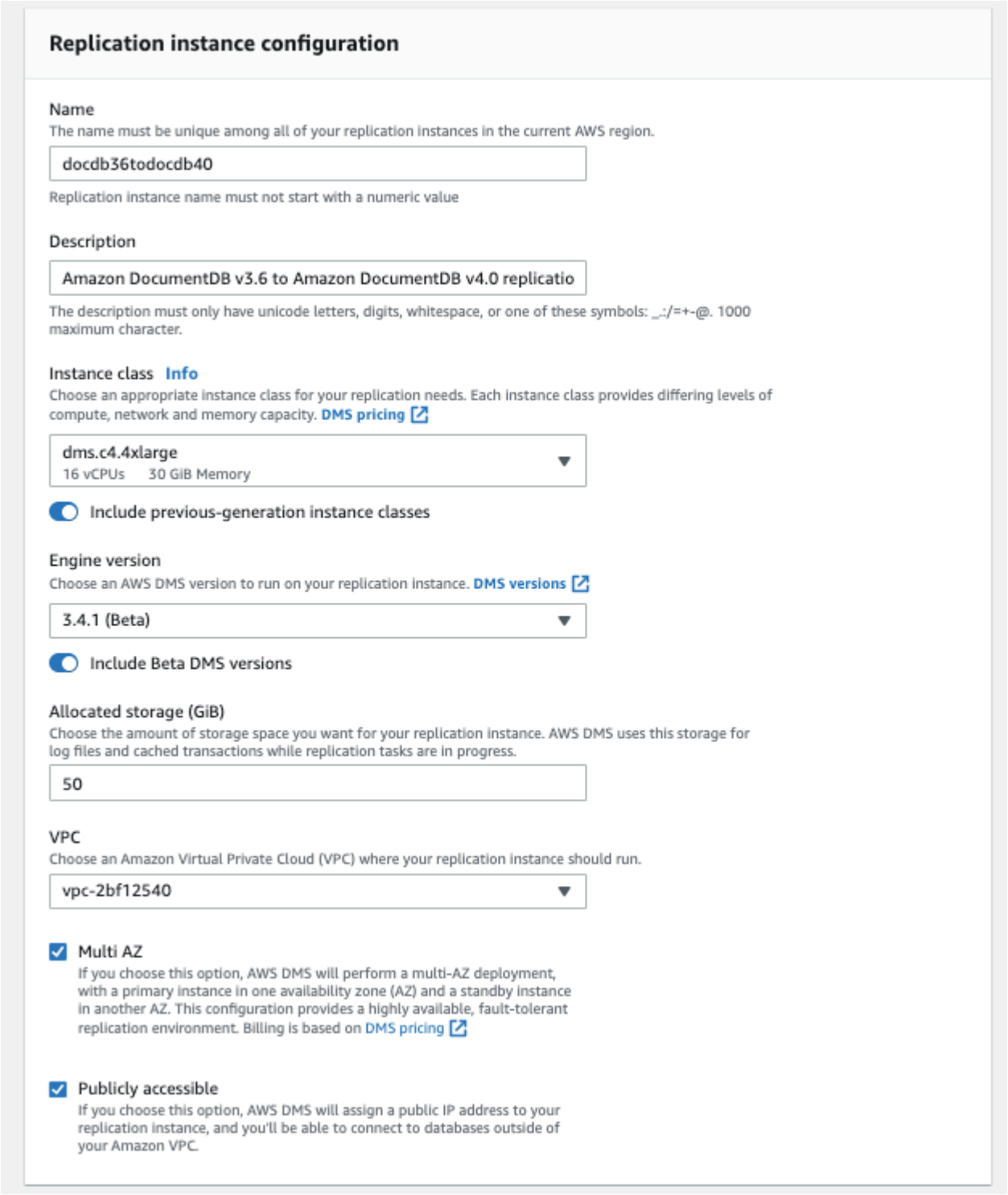
```
mongo --ssl
--host docdb-xx-xx.cluster-xxxxxxxx.us-east-1.docdb.amazonaws.com:27017
--sslCAFile rds-ca-2019-root.pem --username documentdb --password documentdb

db.coll.getIndexes()
```

Step 4: Create an AWS DMS replication instance

An AWS DMS replication instance connects and reads data from your source Amazon DocumentDB cluster and writes it your target Amazon DocumentDB cluster. The AWS DMS replication instance can perform both bulk load and CDC operations. Most of this processing happen in memory. However, large operations might require some buffering on disk. Cached transactions and log files are also written to disk. Once the data is migrated, the replication instance also streams any change events to make sure the source and target are in sync.

To create an AWS DMS replication instance:

1. Open the AWS DMS console.
2. In the navigation pane, choose **Replication instances**.
3. Choose **Create replication instance** and enter the following information:
   - For Name, enter a name of your choice. For example, `docdb36todocdb40`.
   - For **Description**, enter a description of your choice. For listitem, Amazon DocumentDB 3.6 to Amazon DocumentDB 4.0 replication instance.
   - For **Instance class**, choose the size based on your needs.
   - For **Engine version**, choose `3.4.1`.
   - For **Amazon VPC**, choose the Amazon VPC that houses your source and target Amazon DocumentDB clusters.
   - For **Allocated storage** (GiB), use the default of 50 GiB. If you have a high write throughput workload, increase this value to match your workload.
   - For **Multi-AZ**, choose **Yes** if you need high availability and failover support.
   - For **Publicly accessible**, enable this option.



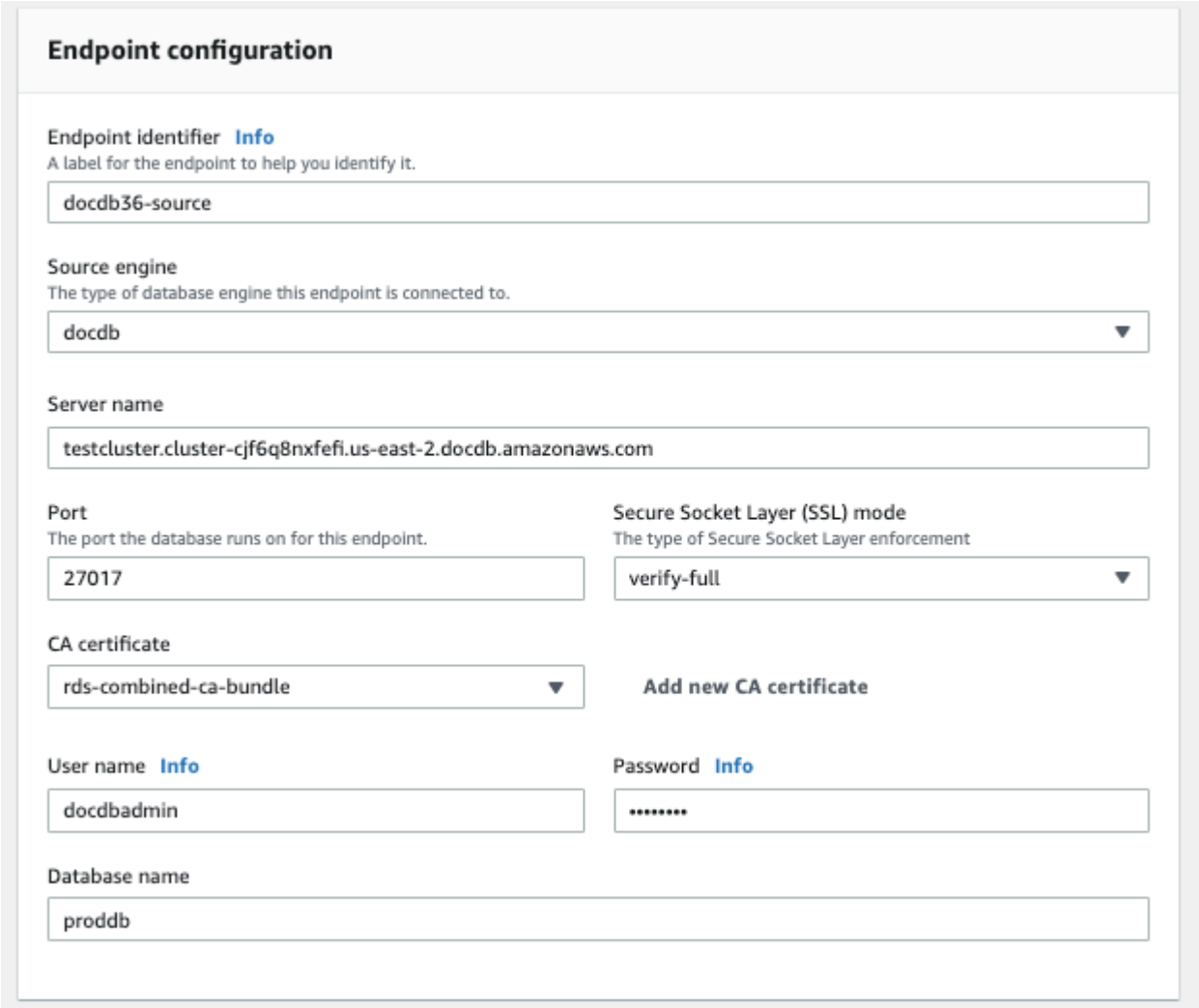
4. Choose **Create replication instance**.

Step 5: Create an AWS DMS source endpoint

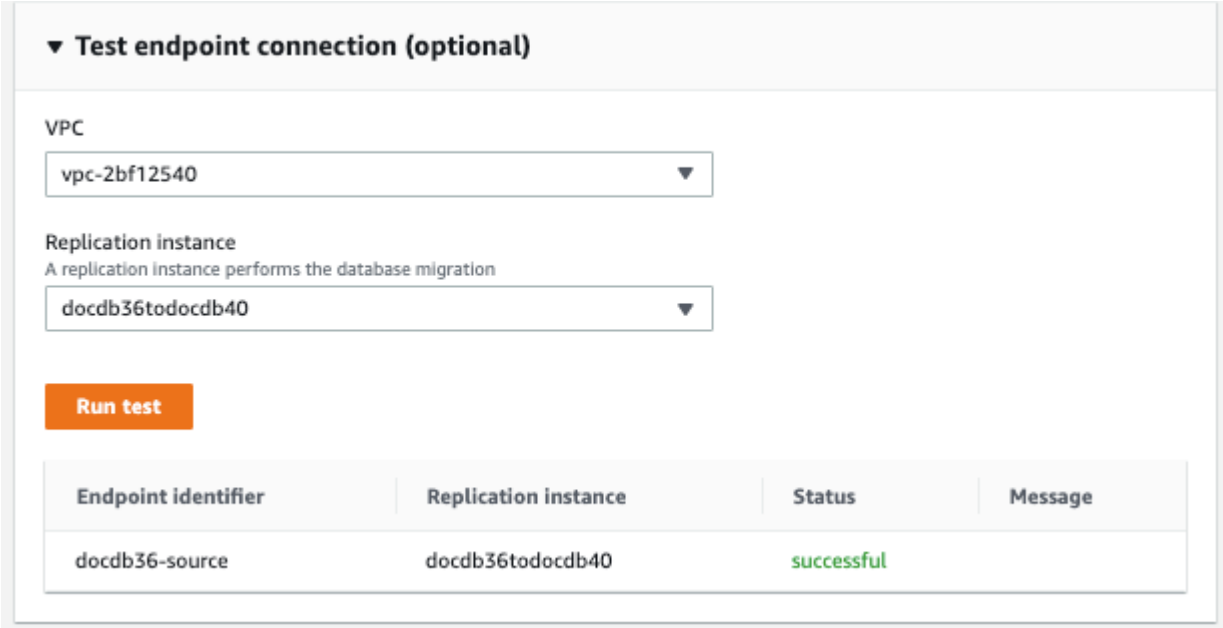
The source endpoint is used for the source Amazon DocumentDB cluster.

To create a source endpoint

1. Open the AWS DMS console.
2. In the navigation pane, choose **Endpoints**.
3. Choose `Create endpoint` and enter the following information:
   - For **Endpoint type**, choose **Source**.
   - >For **Endpoint identifier**, enter a name that's easy to remember, for example `docdb-source`.
   - For **Source engine**, choose `docdb`.
   - For **Server name**, enter the DNS name of your source Amazon DocumentDB cluster.
   - For **Port**, enter the port number of your source Amazon DocumentDB cluster.
   - For **SSL mode**, choose `verify-full`.
   - For **CA certificate**, choose **Add new CA certificate**. Download the new CA certificate to create TLS connections bundle. For **Certificate identifier**, enter `rds-combined-ca-bundle` . For **Import certificate file**, choose **Choose file** and navigate to the `.pem` file that you previously downloaded. Select and open the file. Choose **Import certificate**, then choose `rds-combined-ca-bundle` from the **Choose a certificate** drop down
   - For **User name**, enter the primary username of your source Amazon DocumentDB cluster.
   - For **Password**, enter the primary password of your source Amazon DocumentDB cluster.
   - For **Database name**, enter the database name you are looking to upgrade.



4. Test your connection to verify it was successfully setup.



5. Choose **Create Endpoint**.

> **Note**
>
> AWS DMS can only migrate one database at a time.

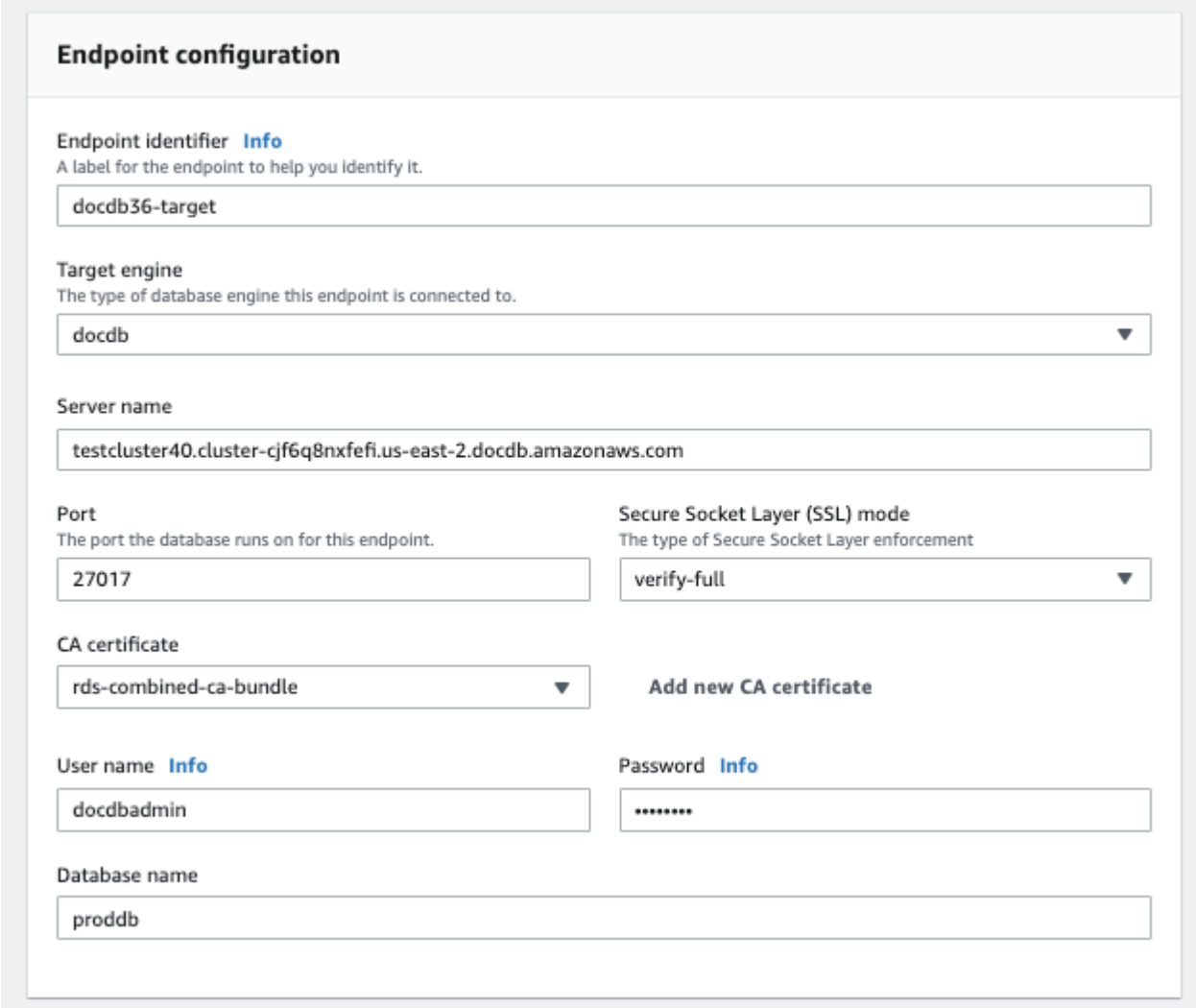
Step 6: Create an AWS DMS target endpoint

The target endpoint is for your target Amazon DocumentDB cluster.

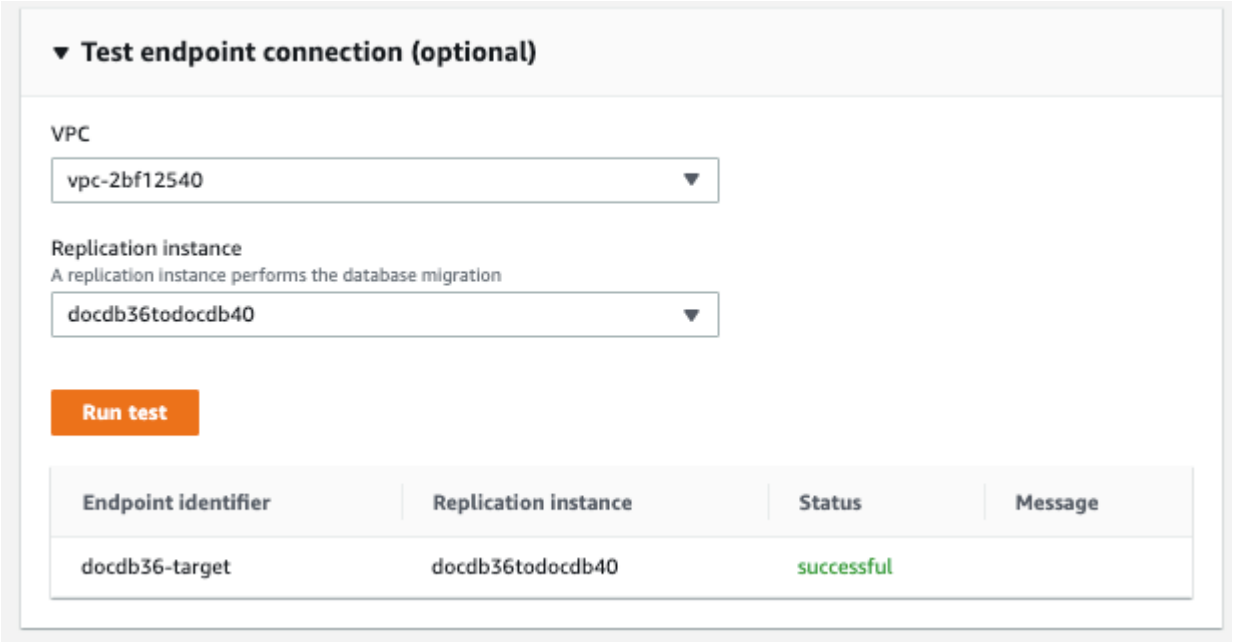
To create a target endpoint:

1. Open the AWS DMS console.
2. In the navigation pane, choose **Endpoints**.
3. Choose **Create endpoint** and enter the following information:
   - For **Endpoint type**, choose **Target**.
   - For **Endpoint identifier**, enter a name that's easy to remember, for example `docdb-target`.
   - For **Source engine**, choose `docdb`.
   - For **Server name**, enter the DNS name of your target Amazon DocumentDB cluster.
   - For **Port**, enter the port number of your target Amazon DocumentDB cluster.

- For **SSL mode**, choose `verify-full`.
- For **CA certificate**, choose the existing `rds-combined-ca-bundle` certificate from the **Choose a certificate** drop down.
- For **User name**, enter the primary username of your target Amazon DocumentDB cluster.
- For **Password**, enter the primary password of your target Amazon DocumentDB cluster.
- For **Database name**, enter the same database name you used to setup your source endpoint.



4. Test your connection to verify it was successfully set up.



5. Choose **Create Endpoint**.

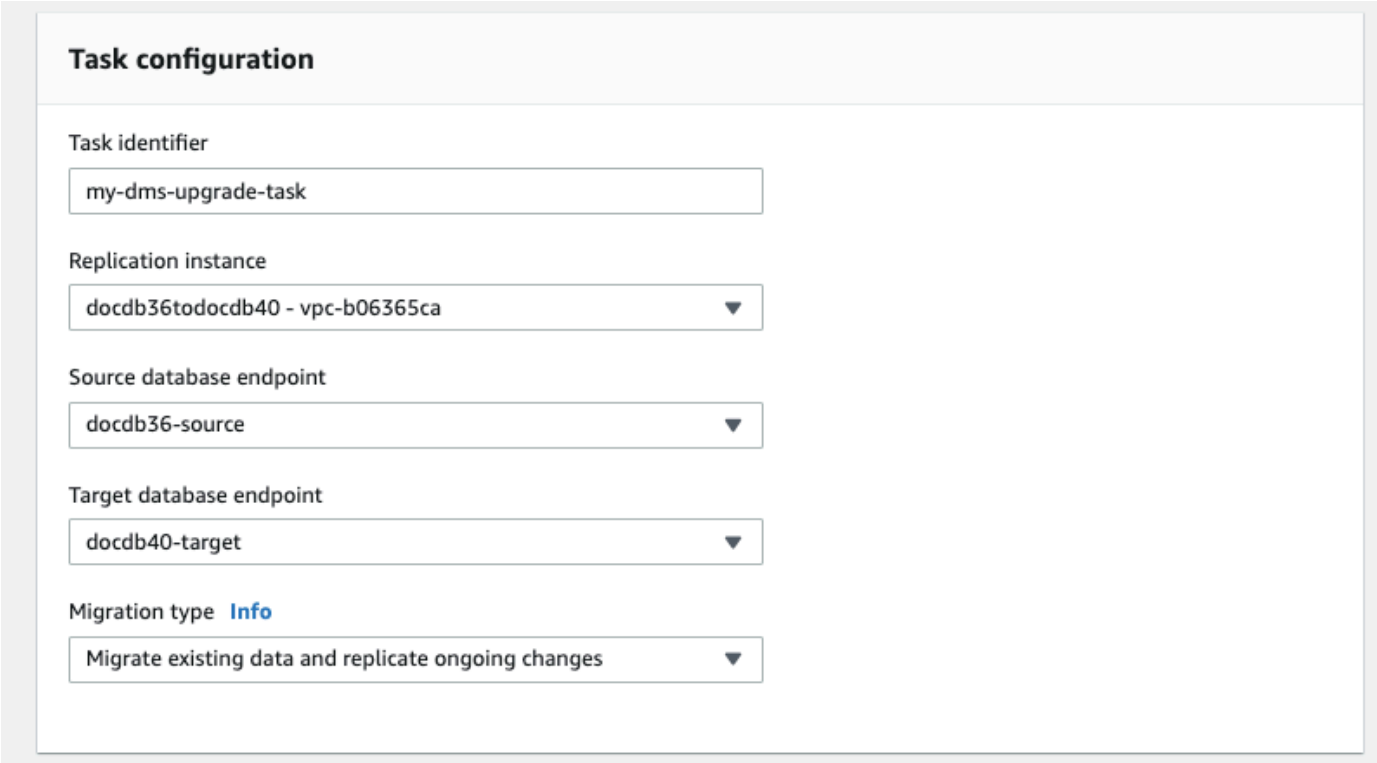
Step 7: Create and run a migration task

An AWS DMS task binds the replication instance with your source and target instance. When you create a migration task, you specify the source endpoint, target endpoint, replication instance, and any desired migration settings. An AWS DMS task can be created with three different migration types - migrate existing data, migrate existing data, and replicate ongoing changes or replicate data changes only. Since the purpose of this walk-through is to upgrade an Amazon DocumentDB cluster with minimal downtime, the steps utilize the option to migrate existing data and replicate ongoing changes. With this option, AWS DMS captures changes while migrating your existing data. AWS DMS continues to capture and apply changes even after the bulk data has been loaded. Eventually the source and target databases will be in sync, allowing for a minimal downtime migration.

Below are the steps to create a migration task for a minimal downtime migration:

1. Open the AWS DMS console.
2. In the navigation pane, choose **Database migration tasks**.
3. Choose **Create database migration task** and enter the following information in the **Task configuration** section:

- For **Task identifier**, enter a name that's easy to remember, for example `my-dms-upgrade-task`.
- For **Descriptive Amazon Resource Name (ARN)**, enter a user-friendly name to override the default DMS ARN.
- For **Replication instance**, choose the replication instance that you created in Step 4: Create an AWS DMS replication instance.
- For **Source database endpoint**, choose the source endpoint that you created in Step 5: Create an AWS DMS source endpoint.
- For **Target database endpoint**, choose the target endpoint that you created in Step 6: Create an AWS DMS target endpoint.
- For **Migration type**, choose **Migrate and replicate**.

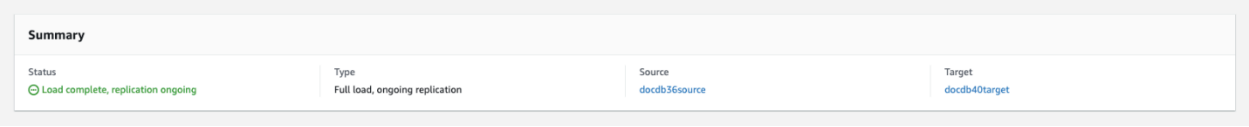


4. Enter the following information in the **Task settings** section:

- For **Target table preparation mode** section, choose **Do nothing**. This will ensure that the indexes created in step 3 will not be dropped.
- For **Task logs** sub-section, select **Turn on CloudWatch logs**.

- For **Migration task startup configuration**, choose **Automatically on create**. This will start the migration task automatically once you create it.
- Choose **Create database migration task**.

AWS DMS now begins migrating data from your source Amazon DocumentDB cluster to your target Amazon DocumentDB cluster. The task status should change from Starting to Running. You can monitor the progress by choosing Tasks in the AWS DMS console. After several minutes/hours (depending on the size of your migration), the status should change from to Load complete, replication ongoing. This means that AWS DMS has completed a full load migration of your source Amazon DocumentDB cluster to a target Amazon DocumentDB cluster and is now replicating change events.



Eventually your source and target will be in sync. You can verify whether they are in sync by running a `count()` operation on your collections to verify all change events have migrated.

Step 8: Changing the application endpoint to the target Amazon DocumentDB cluster

After the full load is complete and the CDC process is replicating continuously, you are ready to change your application's database connection endpoint from your source Amazon DocumentDB cluster to your target Amazon DocumentDB cluster.

Security in Amazon DocumentDB

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. This documentation helps you understand how to apply the shared responsibility model when using Amazon DocumentDB. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** — AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon DocumentDB (with MongoDB compatibility), see AWS Services in Scope by Compliance Program.
- **Security in the cloud** — Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

Amazon DocumentDB is authorized under Federal Risk and Authorization Management Program (FedRAMP). It has FedRAMP High authorization for AWS GovCloud (US) regions and FedRAMP Moderate authorization for AWS US East/West Regions. For details about AWS and compliance efforts, see AWS Services in Scope by Compliance Program.

> **Note**
>
> This chapter applies to both instance-based clusters and Elastic Clusters. For more information, see the topics below.

You also learn how to use other AWS services that help you monitor and secure your Amazon DocumentDB resources. The following topics show you how to configure Amazon DocumentDB to meet your security and compliance objectives.

Topics

- Password management with Amazon DocumentDB and AWS Secrets Manager
- Data protection in Amazon DocumentDB

- Identity and Access Management for Amazon DocumentDB
- Authentication using IAM identity
- Managing Amazon DocumentDB users
- Database access using Role-Based Access Control
- Logging and monitoring in Amazon DocumentDB
- Updating your Amazon DocumentDB TLS certificates
- Updating your Amazon DocumentDB TLS certificates — GovCloud
- Compliance validation in Amazon DocumentDB
- Resilience in Amazon DocumentDB
- Infrastructure security in Amazon DocumentDB
- Amazon DocumentDB API and interface VPC endpoints (AWS PrivateLink)
- Security best practices for Amazon DocumentDB
- Auditing Amazon DocumentDB events

Password management with Amazon DocumentDB and AWS Secrets Manager

Amazon DocumentDB integrates with Secrets Manager to manage primary user passwords for your clusters.

Topics

- Limitations for Secrets Manager integration with Amazon DocumentDB
- Overview of managing primary user passwords with AWS Secrets Manager
- Enforcing Amazon DocumentDB management of the primary user password in AWS Secrets Manager
- Managing the primary user password for a cluster with Secrets Manager

Limitations for Secrets Manager integration with Amazon DocumentDB

Managing primary user passwords with Secrets Manager isn't supported for the following features:

- Clusters that are part of an Amazon DocumentDB global database
- Amazon DocumentDB cross-Region read replicas

Overview of managing primary user passwords with AWS Secrets Manager

With AWS Secrets Manager, you can replace hard-coded credentials in your code, including database passwords, with an API call to Secrets Manager to retrieve the secret programmatically. For more information about Secrets Manager, see AWS Secrets Manager User Guide.

When you store database secrets in Secrets Manager, your AWS account incurs charges. For information about pricing, see AWS Secrets Manager pricing.

You can specify that Amazon DocumentDB manages the primary user password in Secrets Manager for an Amazon DocumentDB cluster when you perform one of the following operations:

- Create the cluster
- Modify the cluster

When you specify that Amazon DocumentDB manages the primary user password in Secrets Manager, Amazon DocumentDB generates the password and stores it in Secrets Manager. You can interact directly with the secret to retrieve the credentials for the primary user. You can also specify a customer managed key to encrypt the secret, or use the KMS key that is provided by Secrets Manager.

Amazon DocumentDB manages the settings for the secret and rotates the secret every seven days by default. You can modify some of the settings, such as the rotation schedule. If you delete a cluster that manages a secret in Secrets Manager, the secret and its associated metadata are also deleted.

To connect to a cluster with the credentials in a secret, you can retrieve the secret from Secrets Manager. For more information, see Get secrets from AWS Secrets Manager and Connect to a SQL database using JDBC with credentials in an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

Enforcing Amazon DocumentDB management of the primary user password in AWS Secrets Manager

You can use IAM condition keys to enforce Amazon DocumentDB management of the primary user password in AWS Secrets Manager. The following policy doesn't allow users to create or restore

instances or clusters unless the primary user password is managed by Amazon DocumentDB in Secrets Manager.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

Managing the primary user password for a cluster with Secrets Manager

You can configure Amazon DocumentDB management of the primary user password in Secrets Manager when you perform the following actions:

- Creating an Amazon DocumentDB cluster
- Modifying an Amazon DocumentDB cluster

You can use the Amazon DocumentDB console or the AWS CLI to perform these actions.

Data protection in Amazon DocumentDB

The AWS shared responsibility model applies to data protection in Amazon DocumentDB (with MongoDB compatibility). As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon DocumentDB or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- Client-side field level encryption
- Encrypting Amazon DocumentDB data at rest
- Encrypting data in transit
- Key management

Client-side field level encryption

Amazon DocumentDB client-side field level encryption (FLE) allows you to encrypt sensitive data in your client applications before it is transferred to a Amazon DocumentDB cluster. Sensitive

data remains encrypted when it is stored and processed in a cluster and is decrypted at the client application when retrieved.

Topics

- Getting started
- Querying in client-side FLE
- Limitations

Getting started

The initial configuration of client-side FLE in Amazon DocumentDB is a four-step process that includes creating an encryption key, associating a role to the application, configuring the application, and defining CRUD operation with encryption options.

Topics

- Step 1: Create the encryption keys
- Step 2: Associate a role with the application
- Step 3: Configure the application
- Step 4: Define a CRUD operation
- Example: client-side field level encryption configuration file

Step 1: Create the encryption keys

Using AWS Key Management Service, create a symmetric key that is used for encrypting and decrypting the sensitive data field and provide it the necessary IAM usage permissions. AWS KMS stores the Customer Key (CK) which is used to encrypt Data Keys (DKs). We recommend storing the Customer Key in KMS to strengthen your security posture. The Data Key is the secondary key which is stored in an Amazon DocumentDB collection and is required to encrypt sensitive fields before storing the document in Amazon DocumentDB. The Customer Key encrypts the Data Key which in turn encrypts and decrypts your data. If you are using a global cluster, you can create a multi-region key that can be used by different service roles in different regions.

For more information about the AWS Key Management Service, including how to create a key, see the AWS Key Management Service Developer Guide.

Step 2: Associate a role with the application

Create an IAM policy with appropriate AWS KMS permissions. This policy allows IAM identities to which it is attached to encrypt and decrypt the KMS key specified in resource field. Your application assumes this IAM role to authenticate with AWS KMS.

The policy should look similar to this:

```
{ "Effect": "Allow",
"Action": ["kms:Decrypt", "kms:Encrypt"],
"Resource": "Customer Key ARN"
}
```

Step 3: Configure the application

By now you defined a Customer Key in AWS KMS and created an IAM role and provided it the right IAM permissions to access the Customer Key. Import the required packages.

```
import boto3
import json
import base64
from pymongo import MongoClient
from pymongo.encryption import (Algorithm,
                                ClientEncryption)
```

```
# create a session object:
my_session = boto3.session.Session()

# get access_key and secret_key programmatically using get_frozen_credentials() method:
 current_credentials = my_session.get_credentials().get_frozen_credentials()
```

1. Specify ‘aws’ as KMS provider type and input your account credentials which were retrieved in the previous step.

   ```
   provider = "aws"
   kms_providers = {
       provider: {
           "accessKeyId": current_credentials.access_key,
           "secretAccessKey": current_credentials.secret_key
       }
   }
   ```

2. Specify the customer key which is used to encrypt the data key:

```
customer_key = {
“region”: “AWS region of the customer_key”,
    “key”: “customer_key ARN”
}

key_vault_namespace = "encryption.dataKeys"

key_alt_name = 'TEST_DATA_KEY'
```

3. Configure the MongoClient object:

```
client = MongoClient(connection_string)

coll = client.test.coll
coll.drop()

client_encryption = ClientEncryption(
    kms_providers, # pass in the kms_providers variable from the previous step
    key_vault_namespace = key_vault_namespace,
    client,
    coll.codec_options
)
```

4. Generate your Data Key:

```
data_key_id = client_encryption.create_data_key(provider,
    customer_key,
    key_alt_name = [key_alt_name])
```

5. Retrieve your existing Data Key:

```
data_key = DataKey("aws",
    master_key = customer_key)
key_id = data_key["_id"]
data_key_id = client[key_vault_namespace].find_one({"_id": key_id})
```

Step 4: Define a CRUD operation

Define the CRUD operation with encryption options.

1. Define the collection to write/read/delete a single document:

```
coll = client.gameinfo.users
```

2. Explicit Encryption - encrypt fields and insert:

> **Note**
>
> Exactly one of "key_id" or "key_alt_name" must be provided.

```
encrypted_first_name = client_encryption.encrypt(
    "Jane",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_alt_name=data_key_id
)
encrypted_last_name = client_encryption.encrypt(
    "Doe",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_alt_name=data_key_id
)
encrypted_dob = client_encryption.encrypt(
    "1990-01-01",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Random,
    key_alt_name=data_key_id
)

coll.insert_one(
    {"gamerTag": "jane_doe90",
    "firstName": encrypted_first_name,
    "lastName": encrypted_last_name,
    "dateOfBirth":encrypted_dob,
    "Favorite_games":["Halo","Age of Empires 2","Medal of Honor"]
})
```

Example: client-side field level encryption configuration file

In the following example, replace each *`user input placeholder`* with your own information.

```
# import python packages:
import boto3
```

```
import json
import base64
from pymongo import MongoClient
from pymongo.encryption import (Algorithm,
                                ClientEncryption)

def main():

    # create a session object:
    my_session = boto3.session.Session()

    # get aws_region from session object:
    aws_region = my_session.region_name

    # get access_key and secret_key programmatically using get_frozen_credentials()
 method:
    current_credentials = my_session.get_credentials().get_frozen_credentials()
    provider = "aws"

    # define the kms_providers which is later used to create the Data Key:
    kms_providers = {
        provider: {
            "accessKeyId": current_credentials.access_key,
            "secretAccessKey": current_credentials.secret_key
        }
    }

    # enter the kms key ARN. Replace the example ARN value.
    kms_arn = "arn:aws:kms:us-east-1:123456789:key/abcd-efgh-ijkl-mnop"
    customer_key = {
        "region": aws_region,
        "key":kms_arn
    }

    # secrets manager is used to strore and retrieve user credentials for connecting to
 an Amazon DocumentDB cluster.
    # retrieve the secret using the secret name. Replace the example secret key.
    secret_name = "/dev/secretKey"
    docdb_credentials = json.loads(my_session.client(service_name = 'secretsmanager',
 region_name = "us-east-1").get_secret_value(SecretId = secret_name)['SecretString'])

    connection_params = '/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
```

```
   conn_str = 'mongodb://' + docdb_credentials["username"] + ':' +
docdb_credentials["password"] + '@' + docdb_credentials["host"] + ':' +
str(docdb_credentials["port"]) + connection_params
   client = MongoClient(conn_str)

   coll = client.test.coll
   coll.drop()

   # store the encryption data keys in a key vault collection (having naming
convention as db.collection):
   key_vault_namespace = "encryption.dataKeys"
   key_vault_db_name, key_vault_coll_name = key_vault_namespace.split(".", 1)

   # set up the key vault (key_vault_namespace) for this example:
   key_vault = client[key_vault_db_name][key_vault_coll_name]
   key_vault.drop()
   key_vault.create_index("keyAltNames", unique=True)

   client_encryption = ClientEncryption(
       kms_providers,
       key_vault_namespace,
       client,
       coll.codec_options)

   # create a new data key for the encrypted field:
   data_key_id = client_encryption.create_data_key(provider, master_key=customer_key,
key_alt_names=["some_key_alt_name"], key_material = None)

   # explicitly encrypt a field:
   encrypted_first_name = client_encryption.encrypt(
   "Jane",
   Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
   key_id=data_key_id
   )
   coll.insert_one(
   {"gamerTag": "jane_doe90",
   "firstName": encrypted_first_name
   })
   doc = coll.find_one()
   print('Encrypted document: %s' % (doc,))

   # explicitly decrypt the field:
   doc["encryptedField"] = client_encryption.decrypt(doc["encryptedField"])
   print('Decrypted document: %s' % (doc,))
```

```
    # cleanup resources:
    client_encryption.close()
    client.close()

    if __name__ == "__main__":
        main()
```

Querying in client-side FLE

Amazon DocumentDB supports point equality queries with client-side FLE. Inequality and comparison queries can return inaccurate results. Read and write operations may have unexpected or incorrect behavior as compared to issuing that same operation against the decrypted value.

For example, to query filters for documents where gamerscore is greater than 500:

```
db.users.find( {
    "gamerscore" : { $gt : 500 }
})
```

The client uses an explicit encryption method to encrypt the query value:

```
encrypted_gamerscore_filter = client_encryption.encrypt(
    500,
        Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
        key_alt_name=data_key_id
        )

db.users.find( {
    "gamerscore" : { $gt : encrypted_gamerscore_filter }
} )
```

In the find operation, Amazon DocumentDB compares the encrypted value of 500 to the encrypted field values stored in each document using the greater than inequality check. The inequality check in the find operation may return a different result when performed using decrypted data and value, even though the operation succeeds in generating results.

Limitations

The following limitations apply to Amazon DocumentDB client-side field level encrytion:

- Amazon DocumentDB supports only point equality queries. Inequality and comparison queries can return inaccurate results. Read and write operations may have unexpected or incorrect behavior as compared to issuing that same operation against the decrypted value. To query filters for documents where gamerscore is greater than 500.

  ```
  db.users.find( {
      "gamerscore" : { $gt : 500 }
      })
  ```

  The client uses an explicit encryption method to encrypt the query value.

  ```
  encrypted_gamerscore_filter = client_encryption.encrypt(
      500,
      Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
      key_alt_name=data_key_id
  )

  db.users.find({
      "gamerscore" : { $gt : encrypted_gamerscore_filter }
  })
  ```

  In the find operation, Amazon DocumentDB compares the encrypted value of 500 to the encrypted field values stored in each document using the greater than inequality check. The inequality check in the find operation may return a different result when performed using decrypted data and value, even though the operation succeeds in generating results.

- Amazon DocumentDB does not support explicit client-side FLE from the Mongo Shell. However, the feature works with any of our supported drivers.

Encrypting Amazon DocumentDB data at rest

> **Note**
>
> AWS KMS is replacing the term *customer master key (CMK)* with *AWS KMS key* and *KMS key*. The concept has not changed. To prevent breaking changes, AWS KMS is keeping some variations of this term.

You encrypt data at rest in your Amazon DocumentDB cluster by specifying the storage encryption option when you create your cluster. Storage encryption is enabled cluster-wide and is applied to all instances, including the primary instance and any replicas. It is also applied to your cluster's storage volume, data, indexes, logs, automated backups, and snapshots.

Amazon DocumentDB uses the 256-bit Advanced Encryption Standard (AES-256) to encrypt your data using encryption keys stored in AWS Key Management Service (AWS KMS). When using an Amazon DocumentDB cluster with encryption at rest enabled, you don't need to modify your application logic or client connection. Amazon DocumentDB handles encryption and decryption of your data transparently, with minimal impact on performance.

Amazon DocumentDB integrates with AWS KMS and uses a method known as envelope encryption to protect your data. When an Amazon DocumentDB cluster is encrypted with an AWS KMS, Amazon DocumentDB asks AWS KMS to use your KMS key to generate a ciphertext data key to encrypt the storage volume. The ciphertext data key is encrypted using the KMS key that you define, and is stored along with the encrypted data and storage metadata. When Amazon DocumentDB needs to access your encrypted data, it requests AWS KMS to decrypt the ciphertext data key using your KMS key and caches the plaintext data key in memory to efficiently encrypt and decrypt data in the storage volume.

The storage encryption facility in Amazon DocumentDB is available for all supported instance sizes and in all AWS Regions where Amazon DocumentDB is available.

Enabling encryption at rest for an Amazon DocumentDB cluster

You can enable or disable encryption at rest on an Amazon DocumentDB cluster when the cluster is provisioned using either the AWS Management Console or the AWS Command Line Interface (AWS CLI). Clusters that you create using the console have encryption at rest enabled by default. Clusters that you create using the AWS CLI have encryption at rest disabled by default. Therefore, you must explicitly enable encryption at rest using the `--storage-encrypted` parameter. In either case, after the cluster is created, you can't change the encryption at rest option.

Amazon DocumentDB uses AWS KMS to retrieve and manage encryption keys, and to define the policies that control how these keys can be used. If you don't specify an AWS KMS key identifier, Amazon DocumentDB uses the default AWS managed service KMS key. Amazon DocumentDB creates a separate KMS key for each AWS Region in your AWS account. For more information, see AWS Key Management Service Concepts.

To get started on creating your own KMS key, see Getting Started in the *AWS Key Management Service Developer Guide*.

> **Important**
>
> You must use a symmetric encryption KMS key to encrypt your cluster as Amazon DocumentDB supports only symmetric encryption KMS keys. Do not use an asymmetric KMS key to attempt to encrypt the data in your Amazon DocumentDB clusters. For more information, see Asymmetric keys in AWS KMS in the *AWS Key Management Service Developer Guide*.

If Amazon DocumentDB can no longer gain access to the encryption key for a cluster — for example, when access to a key is revoked — the encrypted cluster goes into a terminal state. In this case, you can only restore the cluster from a backup. For Amazon DocumentDB, backups are always enabled for 1 day.

In addition, if you disable the key for an encrypted Amazon DocumentDB cluster, you will eventually lose read and write access to that cluster. When Amazon DocumentDB encounters a cluster that is encrypted by a key that it doesn't have access to, it puts the cluster into a terminal state. In this state, the cluster is no longer available, and the current state of the database can't be recovered. To restore the cluster, you must re-enable access to the encryption key for Amazon DocumentDB, and then restore the cluster from a backup.

> **Important**
>
> You cannot change the KMS key for an encrypted cluster after you have already created it. Be sure to determine your encryption key requirements before you create your encrypted cluster.

Using the AWS Management Console

You specify the encryption at rest option when you create a cluster. Encryption at rest is enabled by default when you create a cluster using the AWS Management Console. It can't be changed after the cluster is created.

To specify the encryption at rest option when creating your cluster

1. Create an Amazon DocumentDB cluster as described in the Getting Started section. However, in step 6, do not choose **Create cluster**.

2. Under the **Authentication** section, choose **Show advanced settings**.
3. Scroll down to the **Encryption-at-rest** section.
4. Choose the option that you want for encryption at rest. Whichever option you choose, you can't change it after the cluster is created.
   - To encrypt data at rest in this cluster, choose **Enable encryption**.
   - If you don't want to encrypt data at rest in this cluster, choose **Disable encryption**.
5. Choose the primary key that you want. Amazon DocumentDB uses the AWS Key Management Service (AWS KMS) to retrieve and manage encryption keys, and to define the policies that control how these keys can be used. If you don't specify an AWS KMS key identifier, Amazon DocumentDB uses the default AWS managed service KMS key. For more information, see AWS Key Management Service Concepts.

   > **Note**
   >
   > After you create an encrypted cluster, you can't change the KMS key for that cluster. Be sure to determine your encryption key requirements before you create your encrypted cluster.

6. Complete the other sections as needed, and create your cluster.

Using the AWS CLI

To encrypt an Amazon DocumentDB cluster using the AWS CLI, run the `create-db-cluster` command and specify the `--storage-encrypted` option. Amazon DocumentDB clusters created using the AWS CLI do not enable storage encryption by default.

The following example creates an Amazon DocumentDB cluster with storage encryption enabled.

In the following examples, replace each *`user input placeholder`* with your cluster's information.

Example

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster \
```

```
  --db-cluster-identifier mydocdbcluster \
  --port 27017 \
  --engine docdb \
  --master-username SampleUser1 \
  --master-user-password primaryPassword \
  --storage-encrypted
```

For Windows:

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted
```

When you create an encrypted Amazon DocumentDB cluster, you can specify an AWS KMS key identifier, as in the following example.

Example

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster \
  --db-cluster-identifier SampleUser1 \
  --port 27017 \
  --engine docdb \
  --master-username primaryUsername \
  --master-user-password yourPrimaryPassword \
  --storage-encrypted \
  --kms-key-id key-arn-or-alias
```

For Windows:

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted ^
```

```
--kms-key-id key-arn-or-alias
```

> **Note**
>
> After you create an encrypted cluster, you can't change the KMS key for that cluster. Be sure to determine your encryption key requirements before you create your encrypted cluster.

Limitations for Amazon DocumentDB encrypted clusters

The following limitations exist for Amazon DocumentDB encrypted clusters.

- You can enable or disable encryption at rest for an Amazon DocumentDB cluster only at the time that it is created, not after the cluster has been created. However, you can create an encrypted copy of an unencrypted cluster by creating a snapshot of the unencrypted cluster, and then restoring the unencrypted snapshot as a new cluster while specifying the encryption at rest option.

  For more information, see the following topics:

  - Creating a manual cluster snapshot
  - Restoring from a cluster snapshot
  - Copying Amazon DocumentDB cluster snapshots
- Amazon DocumentDB clusters with storage encryption enabled can't be modified to disable encryption.
- All instances, automated backups, snapshots, and indexes in an Amazon DocumentDB cluster are encrypted with the same KMS key.

Encrypting data in transit

You can use Transport Layer Security (TLS) to encrypt the connection between your application and an Amazon DocumentDB cluster. By default, encryption in transit is enabled for newly created Amazon DocumentDB clusters. It can optionally be disabled when the cluster is created, or at a later time. When encryption in transit is enabled, secure connections using TLS are required to connect to the cluster. For more information connecting to Amazon DocumentDB using TLS, see Connecting programmatically to Amazon DocumentDB.

Managing Amazon DocumentDB cluster TLS settings

Encryption in transit for an Amazon DocumentDB cluster is managed via the TLS parameter in a cluster parameter group. You can manage your Amazon DocumentDB cluster TLS settings using the AWS Management Console or the AWS Command Line Interface (AWS CLI). See the following sections to learn how to verify and modify your current TLS settings.

Using the AWS Management Console

Follow these steps to perform management tasks for TLS encryption using the console—such as identifying parameter groups, verifying the TLS value, and making needed modifications.

> **Note**
>
> Unless you specify differently when you create a cluster, your cluster is created with the default cluster parameter group. The parameters in the `default` cluster parameter group can't be modified (for example, `tls` enabled/disabled). So if your cluster is using a `default` cluster parameter group, you need to modify the cluster to use a non-default cluster parameter group. First, you might need to create a custom cluster parameter group. For more information, see Creating Amazon DocumentDB cluster parameter groups.

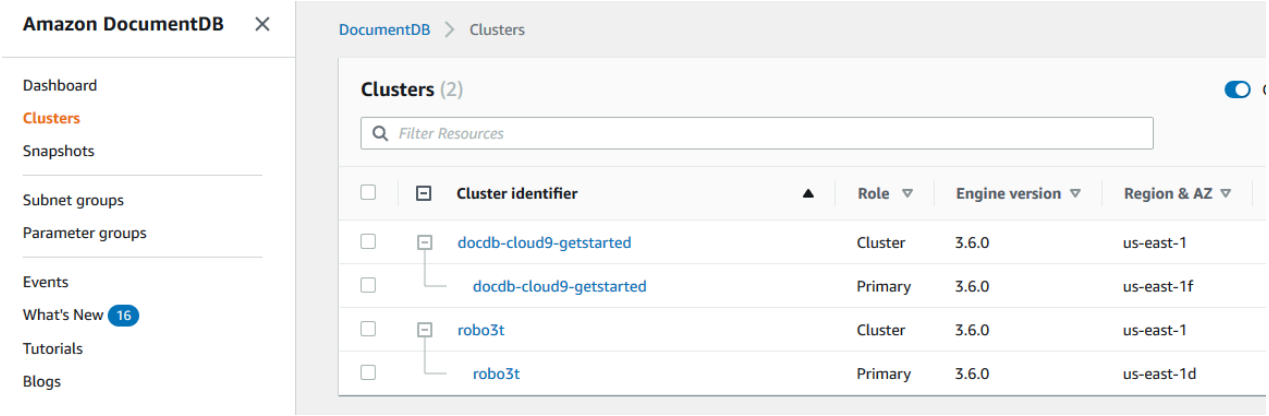
1. **Determine the cluster parameter group that your cluster is using.**

   a. Open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

   b. In the navigation pane, choose **Clusters**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

   c. Note that in the **Clusters** navigation box, the column **Cluster Identifier** shows both clusters and instances. Instances are listed underneath clusters. See the screenshot below for reference.



d. Choose the cluster that you're interested in.

e. Choose the **Configuration** tab and scroll down to the bottom of **Cluster details** and locate the **Cluster parameter group**. Note the name of the cluster parameter group.

   If the name of the cluster's parameter group is `default` (for example, `default.docdb3.6`), you must create a custom cluster parameter group and make it the cluster's parameter group before you continue. For more information, see the following:

   1. Creating Amazon DocumentDB cluster parameter groups — If you don't have a custom cluster parameter group that you can use, create one.
   2. Modifying an Amazon DocumentDB cluster — Modify your cluster to use the custom cluster parameter group.

2. **Determine the current value of the `tls` cluster parameter.**

   a. Open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

   b. In the navigation pane, choose **Parameter groups**.

   c. In the list of cluster parameter groups, choose the name of the cluster parameter group you are interested in.

   d. Locate the **Cluster parameters** section. In the list of cluster parameters, locate the `tls` cluster parameter row. At this point, the following four columns are important:

      - **Cluster parameter name** — The name of the cluster parameters. For managing TLS, you're interested in the `tls` cluster parameter.
      - **Values** — The current value of each cluster parameter.
      - **Allowed values** — A list of values that can be applied to a cluster parameter.

- **Apply type** — Either **static** or **dynamic**. Changes to static cluster parameters can be applied only when the instances are rebooted. Changes to dynamic cluster parameters can be applied either immediately or when the instances are rebooted.

3. **Modify the value of the `tls` cluster parameter.**

   If the value of `tls` is not what is needs to be, modify its value for this cluster parameter group. To change the value of the `tls` cluster parameter, continue from the preceding section by following these steps.

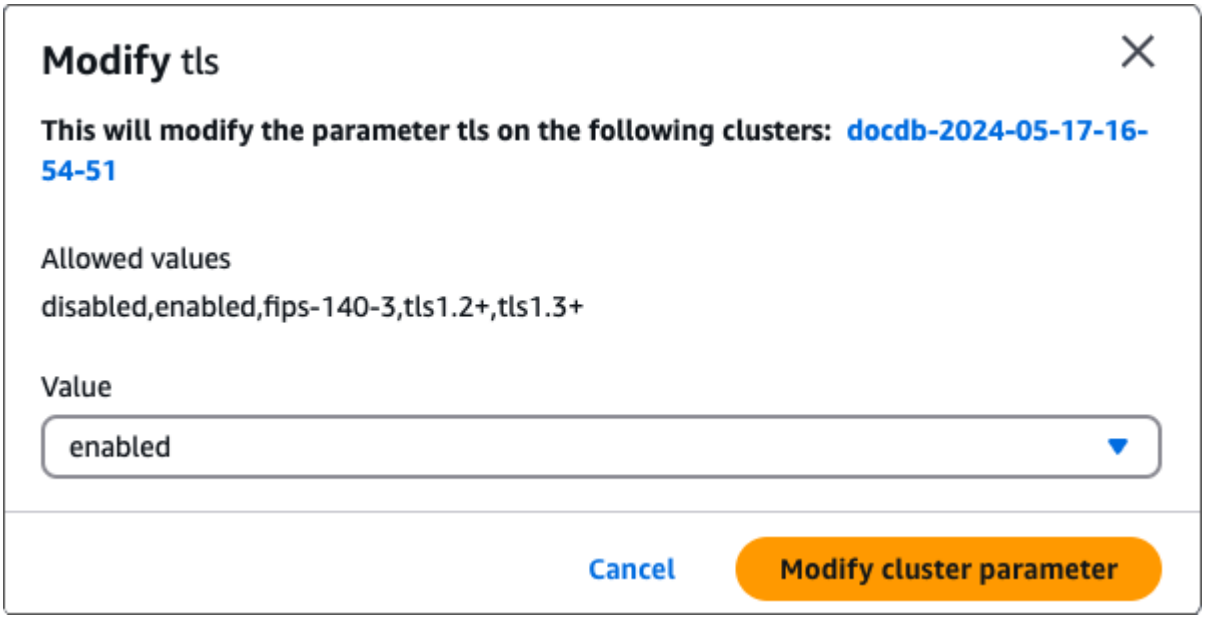
   a. Choose the button to the left of the cluster parameter's name (`tls`).

   b. Choose **Edit**.

   c. To change the value of `tls`, in the **Modify `tls`** dialog box, choose the value that you want for the cluster parameter in the drop-down list.

      Valid values are:

      - **disabled** — Disables TLS
      - **enabled** — Enables TLS versions 1.0 through 1.3.
      - **fips-140-3** — Enables TLS with FIPS. The cluster only accepts secure connections per the requirements of the Federal Information Processing Standards (FIPS) publication 140-3. This is only supported starting with Amazon DocumentDB 5.0 (engine version 3.0.3727) clusters in these regions: ca-central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1.
      - **tls1.2+** — Enables TLS version 1.2 and above. This is only supported starting with Amazon DocumentDB 4.0 (engine version 2.0.10980) and Amazon DocumentDB (engine version 3.0.11051).
      - **tls1.3+** — Enables TLS version 1.3 and above. This is only supported starting with Amazon DocumentDB 4.0 (engine version 2.0.10980) and Amazon DocumentDB (engine version 3.0.11051).



d. Choose **Modify cluster parameter**. The change is applied to each cluster instance when it is rebooted.

4. **Reboot the Amazon DocumentDB instance.**

   Reboot each instance of the cluster so that the change is applied to all instances in the cluster.

   a. Open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
   b. In the navigation pane, choose **Instances**.
   c. To specify an instance to reboot, locate the instance in the list of instances, and choose the button to the left of its name.
   d. Choose **Actions**, and then **Reboot**. Confirm that you want to reboot by choosing **Reboot**.

Using the AWS CLI

Follow these steps to perform management tasks for TLS encryption using the AWS CLI—such as identifying parameter groups, verifying the TLS value, and making needed modifications.

> **Note**
>
> Unless you specify differently when you create a cluster, the cluster is created with the default cluster parameter group. The parameters in the `default` cluster parameter group can't be modified (for example, `tls` enabled/disabled). So if your cluster is using a `default` cluster parameter group, you need to modify the cluster to use a non-default cluster parameter group. You might need to first create a custom cluster parameter group. For more information, see Creating Amazon DocumentDB cluster parameter groups.

1. **Determine the cluster parameter group that your cluster is using.**

   Run the `describe-db-clusters` command with the following options:

   - `--db-cluster-identifier`
   - `--query`

   In the following example, replace each *`user input placeholder`* with your cluster's information.

```
aws docdb describe-db-clusters \
  --db-cluster-identifier mydocdbcluster \
  --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

   Output from this operation looks something like the following (JSON format):

```
[
    [
        "mydocdbcluster",
        "myparametergroup"
    ]
]
```

   If the name of the cluster's parameter group is `default` (for example, `default.docdb3.6`), you must have a custom cluster parameter group and make it the cluster's parameter group before you continue. For more information, see the following topics:

1. Creating Amazon DocumentDB cluster parameter groups — If you don't have a custom cluster parameter group that you can use, create one.
2. Modifying an Amazon DocumentDB cluster — Modify your cluster to use the custom cluster parameter group.

2. **Determine the current value of the `tls` cluster parameter.**

   To get more information about this cluster parameter group, run the `describe-db-cluster-parameters` command with the following options:

   - `--db-cluster-parameter-group-name`
   - `--query`

     Limits the output to just the fields of interest: `ParameterName`, `ParameterValue`,`AllowedValues`, and `ApplyType`.

   In the following example, replace each *`user input placeholder`* with your cluster's information.

```
aws docdb describe-db-cluster-parameters \
  --db-cluster-parameter-group-name myparametergroup \
  --query 'Parameters[*].[ParameterName,ParameterValue,AllowedValues,ApplyType]'
```

   Output from this operation looks something like the following (JSON format):

```
[
    [
        "audit_logs",
        "disabled",
        "enabled,disabled",
        "dynamic"
    ],
    [
        "tls",
        "disabled",
        "disabled,enabled,fips-140-3,tls1.2+,tls1.3+",
        "static"
    ],
    [
        "ttl_monitor",
```

```
        "enabled",
        "disabled,enabled",
        "dynamic"
    ]
]
```

3. **Modify the value of the `tls` cluster parameter.**

   If the value of `tls` is not what it needs to be, modify its value for this cluster parameter group. To change the value of the `tls` cluster parameter, run the `modify-db-cluster-parameter-group` command with the following options:

   - `--db-cluster-parameter-group-name` — Required. The name of the cluster parameter group to modify. This cannot be a `default.*` cluster parameter group.
   - `--parameters` — Required. A list of the cluster parameter group's parameters to modify.
     - `ParameterName` — Required. The name of the cluster parameter to modify.
     - `ParameterValue` — Required. The new value for this cluster parameter. Must be one of the cluster parameter's `AllowedValues`.
       - `enabled` — The cluster accepts secure connections using TLS version 1.0 through 1.3.
       - `disabled` — The cluster does not accept secure connections using TLS.
       - `fips-140-3` — The cluster only accepts secure connections per the requirements of the Federal Information Processing Standards (FIPS) publication 140-3. This is only supported starting with Amazon DocumentDB 5.0 (engine version 3.0.3727) clusters in these regions: ca-central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1.
       - `tls1.2+` — The cluster accepts secure connections using TLS version 1.2 and above. This is only supported starting with Amazon DocumentDB 4.0 (engine version 2.0.10980) and Amazon DocumentDB 5.0 (engine version 3.0.11051).
       - `tls1.3+` — The cluster accepts secure connections using TLS version 1.3 and above. This is only supported starting with Amazon DocumentDB 4.0 (engine version 2.0.10980) and Amazon DocumentDB 5.0 (engine version 3.0.11051).
     - `ApplyMethod` — When this modification is to be applied. For static cluster parameters like `tle`, this value must be `pending-reboot`.

- `pending-reboot` — Change is applied to an instance only after it is rebooted. You must reboot each cluster instance individually for this change to take place across all of the cluster's instances.

In the following examples, replace each *`user input placeholder`* with your cluster's information.

The following code *disables* `tls`, applying the change to each instance when it is rebooted.

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

The following code *enables* `tls` (version 1.0 through 1.3) applying the change to each instance when it is rebooted.

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=enabled,ApplyMethod=pending-
reboot"
```

The following code *enables* TLS with `fips-140-3`, applying the change to each instance when it is rebooted.

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup2 \
  --parameters "ParameterName=tls,ParameterValue=fips-140-3,ApplyMethod=pending-
reboot"
```

Output from this operation looks something like the following (JSON format):

```
{
    "DBClusterParameterGroupName": "myparametergroup"
}
```

4. **Reboot your Amazon DocumentDB instance.**

Reboot each instance of the cluster so that the change is applied to all instances in the cluster. To reboot an Amazon DocumentDB instance, run the `reboot-db-instance` command with the following option:

- `--db-instance-identifier`

The following code reboots the instance `mydocdbinstance`.

In the following examples, replace each *`user input placeholder`* with your cluster's information.

Example

For Linux, macOS, or Unix:

```
aws docdb reboot-db-instance \
  --db-instance-identifier mydocdbinstance
```

For Windows:

```
aws docdb reboot-db-instance ^
  --db-instance-identifier mydocdbinstance
```

Output from this operation looks something like the following (JSON format):

```
{
    "DBInstance": {
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "PreferredMaintenanceWindow": "fri:09:32-fri:10:02",
        "PendingModifiedValues": {},
        "DBInstanceStatus": "rebooting",
        "DBSubnetGroup": {
            "Subnets": [
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetIdentifier": "subnet-4e26d263"
```

```
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1c"
                    },
                    "SubnetIdentifier": "subnet-afc329f4"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1e"
                    },
                    "SubnetIdentifier": "subnet-b3806e8f"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1d"
                    },
                    "SubnetIdentifier": "subnet-53ab3636"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetIdentifier": "subnet-991cb8d0"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1f"
                    },
                    "SubnetIdentifier": "subnet-29ab1025"
                }
            ],
            "SubnetGroupStatus": "Complete",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-91280df6",
            "DBSubnetGroupName": "default"
        },
        "PromotionTier": 2,
        "DBInstanceClass": "db.r5.4xlarge",
```

```
        "InstanceCreateTime": "2018-11-05T23:10:49.905Z",
        "PreferredBackupWindow": "00:00-00:30",
        "KmsKeyId": "arn:aws:kms:us-east-1:012345678901:key/0961325d-a50b-44d4-
b6a0-a177d5ff730b",
        "StorageEncrypted": true,
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "EngineVersion": "3.6.0",
        "DbiResourceId": "db-SAMPLERESOURCEID",
        "DBInstanceIdentifier": "mydocdbinstance",
        "Engine": "docdb",
        "AvailabilityZone": "us-east-1a",
        "DBInstanceArn": "arn:aws:rds:us-east-1:012345678901:db:sample-cluster-
instance-00",
        "BackupRetentionPeriod": 1,
        "Endpoint": {
            "Address": "mydocdbinstance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "Z2R2ITUGPM61AM"
        },
        "DBClusterIdentifier": "mydocdbcluster"
    }
}
```

It takes a few minutes for your instance to reboot. You can use the instance only when its status is *available*. You can monitor the instance's status using the console or AWS CLI. For more information, see Monitoring an Amazon DocumentDB instance's status.

Key management

Amazon DocumentDB uses AWS Key Management Service (AWS KMS) to retrieve and manage encryption keys. AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using AWS KMS, you can create encryption keys and define the policies that control how these keys can be used. AWS KMS supports AWS CloudTrail, so you can audit key usage to verify that keys are being used appropriately.

Your AWS KMS keys can be used in combination with Amazon DocumentDB and supported AWS services such as Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), Amazon Elastic Block Store (Amazon EBS), and Amazon Redshift. For a list of services that support AWS KMS, see How AWS Services use AWS KMS in the *AWS Key Management Service Developer Guide*. For information about AWS KMS, see What is AWS Key Management Service?

Identity and Access Management for Amazon DocumentDB

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon DocumentDB resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon DocumentDB works with IAM
- Identity-based policy examples for Amazon DocumentDB
- Troubleshooting Amazon DocumentDB identity and access
- Managing access permissions to your Amazon DocumentDB resources
- Using identity-based policies (IAM policies) for Amazon DocumentDB
- AWS managed policies for Amazon DocumentDB
- Amazon DocumentDB API permissions: actions, resources, and conditions reference

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon DocumentDB.

Service user – If you use the Amazon DocumentDB service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon DocumentDB features to do your work, you might need additional permissions. Understanding

how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon DocumentDB, see Troubleshooting Amazon DocumentDB identity and access.

Service administrator – If you're in charge of Amazon DocumentDB resources at your company, you probably have full access to Amazon DocumentDB. It's your job to determine which Amazon DocumentDB features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon DocumentDB, see How Amazon DocumentDB works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon DocumentDB. To view example Amazon DocumentDB identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon DocumentDB.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication

(MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or

store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's

permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon DocumentDB works with IAM

Before you use IAM to manage access to Amazon DocumentDB, learn what IAM features are available to use with Amazon DocumentDB.

IAM features you can use with Amazon DocumentDB

IAM feature	Instance-based clusters	Elastic clusters
Identity-based policies	Yes	Yes
Resource-based policies	No	No
Policy actions	Yes	Yes
Policy resources	Yes	Yes
Policy condition keys (service-specific)	Yes	Yes
ACLs	No	No
ABAC (tags in policies)	Partial	Yes
Temporary credentials	Yes	Yes
Principal permissions	Yes	Yes
Service roles	Yes	Yes

IAM feature	Instance-based clusters	Elastic clusters
Service-linked roles	No	Yes

To get a high-level view of how Amazon DocumentDB and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon DocumentDB

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon DocumentDB

To view examples of Amazon DocumentDB identity-based policies, see Identity-based policy examples for Amazon DocumentDB.

Resource-based policies within Amazon DocumentDB

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon DocumentDB

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

> **Note**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).
> To see a list of RDS actions, see Actions defined by Amazon Relational Database Service in the *Service Authorization Reference*.
> To view policy actions for Amazon DocumentDB elastic clusters, see Actions defined by Amazon DocumentDB elastic clusters in the *Service Authorization Reference*.

Policy actions in Amazon DocumentDB use the following prefix before the action:

```
aws
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

To view examples of Amazon DocumentDB identity-based policies, see Identity-based policy examples for Amazon DocumentDB.

Policy resources for Amazon DocumentDB

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

> **Note**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).
> To view a list of RDS resource types and their ARNs, see Resources defined by Amazon Relational Database Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Relational Database Service.
> To view resource types for Amazon DocumentDB eleastic clusters, see Resource types defined by Amazon DocumentDB elastic clusters in the *Service Authorization Reference*.

To view examples of Amazon DocumentDB identity-based policies, see Identity-based policy examples for Amazon DocumentDB.

Policy condition keys for Amazon DocumentDB

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

> **Note**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).
> To see a list of RDS condition keys, see Condition keys for Amazon Relational Database Service in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Relational Database Service.
> To view condition keys for Amazon DocumentDB eleastic clusters, see Condition keys for Amazon DocumentDB elastic clusters in the *Service Authorization Reference*.

To view examples of Amazon DocumentDB identity-based policies, see Identity-based policy examples for Amazon DocumentDB.

ACLs in Amazon DocumentDB

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon DocumentDB

> **Note**
>
> ABAC is only partially supported for instance-based clusters but is supported for elastic clusters.

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amazon DocumentDB

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amazon DocumentDB

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon DocumentDB

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon DocumentDB functionality. Edit service roles only when Amazon DocumentDB provides guidance to do so.

Service-linked roles for Amazon DocumentDB

> **Note**
>
> Service-linked roles are not supported for instance-based clusters but are supported for elastic clusters.

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon DocumentDB

By default, users and roles don't have permission to create or modify Amazon DocumentDB resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon DocumentDB, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon Relational Database Service in the *Service Authorization Reference*.

> **Note**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).
> For policy actions for Amazon DocumentDB elastic clusters, see Actions, resources, and condition keys for Amazon DocumentDB Elastic Clusters in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Amazon DocumentDB console
- Allow users to view their own permissions

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon DocumentDB resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon DocumentDB console

To access the Amazon DocumentDB (with MongoDB compatibility) console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon DocumentDB resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Amazon DocumentDB console, also attach the Amazon DocumentDB `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting Amazon DocumentDB identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon DocumentDB and IAM.

Topics

- I am not authorized to perform an action in Amazon DocumentDB
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amazon DocumentDB resources

I am not authorized to perform an action in Amazon DocumentDB

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `aws:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `aws:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon DocumentDB.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon DocumentDB. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon DocumentDB resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon DocumentDB supports these features, see How Amazon DocumentDB works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access permissions to your Amazon DocumentDB resources

Every AWS resource is owned by an AWS account, and permissions to create or access the resources are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

> **Note**
>
> An *account administrator* (or administrator user) is a user with administrator permissions. For more information, see IAM Best Practices in the *IAM User Guide*.

Topics

- Amazon DocumentDB Resources and operations

- Understanding resource ownership
- Managing access to resources
- Specifying policy elements: actions, effects, resources, and principals
- Specifying conditions in a policy

Amazon DocumentDB Resources and operations

In Amazon DocumentDB, the primary resource is a *cluster*. Amazon DocumentDB supports other resources that can be used with the primary resource such as *instances*, *parameter groups*, and *event subscriptions*. These resources are referred to as *subresources*.

These resources and subresources have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

Resource Type	ARN Format
Cluster	`arn:aws:rds:` *`region`*`:`*`account-id`* `:cluster:` *`db-cluster-name`*
Cluster parameter group	`arn:aws:rds:` *`region`*`:`*`account-id`* `:cluster-pg:` *`cluster-parameter-group-name`*
Cluster snapshot	`arn:aws:rds:` *`region`*`:`*`account-id`* `:cluster-snapshot:` *`cluster-snapshot-name`*
Instance	`arn:aws:rds:` *`region`*`:`*`account-id`* `:db:`*`db-instance-name`*
Security group	`arn:aws:rds:` *`region`*`:`*`account-id`* `:secgrp:`*`security-group-name`*
Subnet group	`arn:aws:rds:` *`region`*`:`*`account-id`* `:subgrp:`*`subnet-group-name`*

Amazon DocumentDB provides a set of operations to work with the Amazon DocumentDB resources. For a list of available operations, see Actions.

Understanding resource ownership

A *resource owner* is the AWS account that created a resource. That is, the resource owner is the AWS account of the *principal entity* (the root account, an IAM user, or an IAM role) that authenticates the request that creates the resource. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create an Amazon DocumentDB resource, such as an instance, your AWS account is the owner of the Amazon DocumentDB resource.
- If you create an IAM user in your AWS account and grant permissions to create Amazon DocumentDB resources to that user, the user can create Amazon DocumentDB resources. However, your AWS account, to which the user belongs, owns the Amazon DocumentDB resources.
- If you create an IAM role in your AWS account with permissions to create Amazon DocumentDB resources, anyone who can assume the role can create Amazon DocumentDB resources. Your AWS account, to which the role belongs, owns the Amazon DocumentDB resources.

Managing access to resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

> **Note**
>
> This section discusses using IAM in the context of Amazon DocumentDB. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWSIAM Policy Reference in the *IAM User Guide*.

Policies that are attached to an IAM identity are referred to as *identity-based* policies (IAM policies). Policies that are attached to a resource are referred to as *resource-based* policies. Amazon DocumentDB supports only identity-based policies (IAM policies).

Topics

- Identity-based policies (IAM policies)
- Resource-based policies

Identity-based policies (IAM policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – An account administrator can use a permissions policy that is associated with a particular user to grant permissions for that user to create an Amazon DocumentDB resource, such as an instance.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, an administrator can create a role to grant cross-account permissions to another AWS account or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows the users in Account B to create or access resources in Account A. The principal in the trust policy can also be an AWS service principal if you want to grant permissions to an AWS service to assume the role.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following is an example policy that allows the user with the ID `123456789012` to create instances for your AWS account. The new instance must use an option group and a parameter group that starts with `default`, and it must use the `default` subnet group.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateDBInstanceOnly",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:123456789012:db:test*",
```

```
                "arn:aws:rds:*:123456789012:pg:cluster-pg:default*",
                "arn:aws:rds:*:123456789012:subgrp:default"
            ]
        }
    ]
}
```

For more information about using identity-based policies with Amazon DocumentDB, see Using identity-based policies (IAM policies) for Amazon DocumentDB. For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-based policies

Other services, such as Amazon Simple Storage Service (Amazon S3), also support resource-based permissions policies. For example, you can attach a policy to an Amazon S3 bucket to manage access permissions to that bucket. Amazon DocumentDB doesn't support resource-based policies.

Specifying policy elements: actions, effects, resources, and principals

For each Amazon DocumentDB resource (see Amazon DocumentDB Resources and operations), the service defines a set of API operations. For more information, see Actions. To grant permissions for these API operations, Amazon DocumentDB defines a set of actions that you can specify in a policy. Performing an API operation can require permissions for more than one action.

The following are the basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies.
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, the `rds:DescribeDBInstances` permission allows the user to perform the `DescribeDBInstances` operation.
- **Effect** – You specify the effect when the user requests the specific action—this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). Amazon DocumentDB doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the Amazon DocumentDB API actions and the resources that they apply to, see Amazon DocumentDB API permissions: actions, resources, and conditions reference.

Specifying conditions in a policy

When you grant permissions, you can use the IAM policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. Amazon DocumentDB has no service-specific context keys that can be used in an IAM policy. For a list of global condition context keys that are available to all services, see Available Keys for Conditions in the *IAM User Guide*.

Using identity-based policies (IAM policies) for Amazon DocumentDB

> **Important**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon RDS. Amazon DocumentDB console, AWS CLI, and API calls are logged as calls made to the Amazon RDS API.
> We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Amazon DocumentDB resources. For more information, see Managing access permissions to your Amazon DocumentDB resources.

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

The following is an example of an IAM policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowCreateDBInstanceOnly",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:123456789012:db:test*",
                "arn:aws:rds:*:123456789012:pg:cluster-pg:default*",
                "arn:aws:rds:*:123456789012:subgrp:default"
            ]
        }
    ]
}
```

The policy includes a single statement that specifies the following permissions for the IAM user:

- The policy allows the IAM user to create an instance using the CreateDBInstance action (this also applies to the create-db-instance AWS CLI operation and the AWS Management Console).
- The `Resource` element specifies that the user can perform actions on or with resources. You specify resources using an Amazon Resource Name (ARN). This ARN includes the name of the service that the resource belongs to (`rds`), the AWS Region (`*` indicates any Region in this example), the user account number (`123456789012` is the user ID in this example), and the type of resource.

  The `Resource` element in the example specifies the following policy constraints on resources for the user:

  - The instance identifier for the new instance must begin with `test` (for example, `testCustomerData1`, `test-region2-data`).
  - The cluster parameter group for the new instance must begin with `default`.
  - The subnet group for the new instance must be the `default` subnet group.

The policy doesn't specify the `Principal` element because in an identity-based policy you don't specify the principal who gets the permission. When you attach policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon DocumentDB API operations and the resources that they apply to, see Amazon DocumentDB API permissions: actions, resources, and conditions reference.

Permissions required to use the Amazon DocumentDB console

For a user to work with the Amazon DocumentDB console, that user must have a minimum set of permissions. These permissions allow the user to describe the Amazon DocumentDB resources for their AWS account and to provide other related information, including Amazon EC2 security and network information.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy. To ensure that those users can still use the Amazon DocumentDB console, also attach the `AmazonDocDBConsoleFullAccess` managed policy to the user, as described in AWS managed policies for Amazon DocumentDB.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the Amazon DocumentDB API.

Customer managed policy examples

In this section, you can find example user policies that grant permissions for various Amazon DocumentDB actions. These policies work when you are using Amazon DocumentDB API actions, AWS SDKs, or the AWS CLI. When you are using the console, you need to grant additional permissions specific to the console, which is discussed in Permissions required to use the Amazon DocumentDB console.

For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS) and Amazon Neptune.

> **Note**
>
> All examples use the US East (N. Virginia) Region (`us-east-1`) and contain fictitious account IDs.

Examples

- Example 1: Allow a user to perform any describe action on any Amazon DocumentDB resource
- Example 2: Prevent a user from deleting an instance
- Example 3: Prevent a user from creating a cluster unless storage encryption is enabled

Example 1: Allow a user to perform any describe action on any Amazon DocumentDB resource

The following permissions policy grants permissions to a user to run all of the actions that begin with `Describe`. These actions show information about an Amazon DocumentDB resource, such as an instance. The wildcard character (*) in the `Resource` element indicates that the actions are allowed for all Amazon DocumentDB resources that are owned by the account.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowRDSDescribe",
         "Effect":"Allow",
         "Action":"rds:Describe*",
         "Resource":"*"
      }
   ]
}
```

Example 2: Prevent a user from deleting an instance

The following permissions policy grants permissions to prevent a user from deleting a specific instance. For example, you might want to deny the ability to delete your production instances to any user that is not an administrator.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyDelete1",
         "Effect":"Deny",
         "Action":"rds:DeleteDBInstance",
         "Resource":"arn:aws:rds:us-east-1:123456789012:db:my-db-instance"
      }
   ]
}
```

Example 3: Prevent a user from creating a cluster unless storage encryption is enabled

The following permissions policy denies permissions to a user from creating an Amazon DocumentDB cluster unless storage encryption is enabled.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "PreventUnencryptedDocumentDB",
         "Effect": "Deny",
         "Action": "RDS:CreateDBCluster",
         "Condition": {
         "Bool": {
         "rds:StorageEncrypted": "false"
      },
         "StringEquals": {
         "rds:DatabaseEngine": "docdb"
         }
      },
      "Resource": "*"
      }
   ]
}
```

AWS managed policies for Amazon DocumentDB

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *AWS Identity and Access Management User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ViewOnlyAccess` AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions

for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *AWS Identity and Access Management User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon DocumentDB:

- AmazonDocDBFullAccess – Grants full access to all Amazon DocumentDB resources for the root AWS account.
- AmazonDocDBReadOnlyAccess – Grants read-only access to all Amazon DocumentDB resources for the root AWS account.
- AmazonDocDBConsoleFullAccess – Grants full access to manage Amazon DocumentDB and Amazon DocumentDB elastic cluster resources using the AWS Management Console.
- AmazonDocDBElasticReadOnlyAccess – Grants read-only access to all Amazon DocumentDB elastic cluster resources for the root AWS account.
- AmazonDocDBElasticFullAccess – Grants full access to all Amazon DocumentDB elastic cluster resources for the root AWS account.

AmazonDocDBFullAccess

This policy grants administrative permissions that allow a principal full access to all Amazon DocumentDB actions. The permissions in this policy are grouped as follows:

- The Amazon DocumentDB permissions allow all Amazon DocumentDB actions.
- Some of the Amazon EC2 permissions in this policy are required to validate the passed resources in an API request. This is to make sure Amazon DocumentDB is able to successfully use the resources with a cluster. The rest of the Amazon EC2 permissions in this policy allow Amazon DocumentDB to create AWS resources that are needed to make it possible for you to connect to your clusters.
- The Amazon DocumentDB permissions are used during API calls to validate the passed resources in a request. They are required for Amazon DocumentDB to be able to use the passed key with the Amazon DocumentDB cluster.
- The CloudWatch Logs are required for Amazon DocumentDB to be able to ensure that the log delivery destinations are reachable, and that they are valid for broker log use.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Action": [
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBCluster",
                "rds:CreateDBClusterParameterGroup",
                "rds:CreateDBClusterSnapshot",
                "rds:CreateDBInstance",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:DeleteDBCluster",
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
                "rds:DeleteDBParameterGroup",
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeOptionGroups",
```

```
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
```

```
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDBReadOnlyAccess

This policy grants read-only permissions that allow users to view information in Amazon DocumentDB. Principals with this policy attached can't make any updates or delete exiting resources, nor can they create new Amazon DocumentDB resources. For example, principals with these permissions can view the list of clusters and configurations associated with their account, but cannot change the configuration or settings of any clusters. The permissions in this policy are grouped as follows:

- Amazon DocumentDB permissions allow you to list Amazon DocumentDB resources, describe them, and get information about them.
- Amazon EC2 permissions are used to describe the Amazon VPC, subnets, security groups, and ENIs that are associated with a cluster.
- An Amazon DocumentDB permission is used to describe the key that is associated with the cluster.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DownloadDBLogFilePortion",
                "rds:ListTagsForResource"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
```

```
                "ec2:DescribeVpcs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "kms:ListAliases",
                "kms:ListKeyPolicies"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "logs:DescribeLogStreams",
                "logs:GetLogEvents"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*",
                "arn:aws:logs:*:*:log-group:/aws/docdb/*:log-stream:*"
            ]
        }
    ]
}
```

AmazonDocDBConsoleFullAccess

Grants full access to manage Amazon DocumentDB resources using the AWS Management Console for following:

- The Amazon DocumentDB permissions to allow all Amazon DocumentDB and Amazon DocumentDB cluster actions.
- Some of the Amazon EC2 permissions in this policy are required to validate the passed resources in an API request. This is to make sure Amazon DocumentDB is able to successfully use the resources to provision and maintain the cluster. The rest of the Amazon EC2 permissions in this policy allow Amazon DocumentDB to create AWS resources that are needed to make it possible for you to connect to your clusters like VPCEndpoint.

- AWS KMS permissions are used during API calls to AWS KMS to validate the passed resources in a request. They are required for Amazon DocumentDB to be able to use the passed key to encrypt and decrypt the data at rest with the Amazon DocumentDB elastic cluster.
- The CloudWatch Logs are required for Amazon DocumentDB to be able to ensure that the log delivery destinations are reachable, and that they are valid for auditing and profiling log use.
- Secrets Manager permissions are required to validate given secret and use it setup the admin user for Amazon DocumentDB elastic clusters.
- Amazon RDS permissions are required for Amazon DocumentDB cluster management actions. For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon RDS.
- SNS permissions allows principals to Amazon Simple Notification Service (Amazon SNS) subscriptions and topics, and to publish Amazon SNS messages.
- IAM permissions are required for creating the service linked roles required for metrics and logs publishing.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DocdbSids",
            "Effect": "Allow",
            "Action": [
                "docdb-elastic:CreateCluster",
                "docdb-elastic:UpdateCluster",
                "docdb-elastic:GetCluster",
                "docdb-elastic:DeleteCluster",
                "docdb-elastic:ListClusters",
                "docdb-elastic:CreateClusterSnapshot",
                "docdb-elastic:GetClusterSnapshot",
                "docdb-elastic:DeleteClusterSnapshot",
                "docdb-elastic:ListClusterSnapshots",
                "docdb-elastic:RestoreClusterFromSnapshot",
                "docdb-elastic:TagResource",
                "docdb-elastic:UntagResource",
                "docdb-elastic:ListTagsForResource",
                "docdb-elastic:CopyClusterSnapshot",
                "docdb-elastic:StartCluster",
                "docdb-elastic:StopCluster",
                "docdb-elastic:GetPendingMaintenanceAction",
```

```
                "docdb-elastic:ListPendingMaintenanceActions",
                "docdb-elastic:ApplyPendingMaintenanceAction",
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBCluster",
                "rds:CreateDBClusterParameterGroup",
                "rds:CreateDBClusterSnapshot",
                "rds:CreateDBInstance",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:CreateGlobalCluster",
                "rds:DeleteDBCluster",
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
                "rds:DeleteDBParameterGroup",
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DeleteGlobalCluster",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
```

```
                "rds:DescribeGlobalClusters",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:ModifyGlobalCluster",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveFromGlobalCluster",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "DependencySids",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:AllocateAddress",
                "ec2:AssignIpv6Addresses",
                "ec2:AssignPrivateIpAddresses",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
```

```
                "ec2:AssociateSubnetCidrBlock",
                "ec2:AssociateVpcCidrBlock",
                "ec2:AttachInternetGateway",
                "ec2:AttachNetworkInterface",
                "ec2:CreateCustomerGateway",
                "ec2:CreateDefaultSubnet",
                "ec2:CreateDefaultVpc",
                "ec2:CreateInternetGateway",
                "ec2:CreateNatGateway",
                "ec2:CreateNetworkInterface",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateVpc",
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeCustomerGateways",
                "ec2:DescribeInstances",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribePrefixLists",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:ModifyVpcEndpoint",
                "kms:DescribeKey",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
```

```
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "DocdbSLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
                }
            }
        },
        {
            "Sid": "DocdbElasticSLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-
elastic.amazonaws.com/AWSServiceRoleForDocDB-Elastic",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "docdb-elastic.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDBElasticReadOnlyAccess

This policy grants read-only permissions that allow users to view elastic cluster information in Amazon DocumentDB. Principals with this policy attached can't make any updates or delete exiting resources, nor can they create new Amazon DocumentDB resources. For example, principals with these permissions can view the list of clusters and configurations associated with their account, but cannot change the configuration or settings of any clusters. The permissions in this policy are grouped as follows:

- Amazon DocumentDB elastic cluster permissions allow you to list Amazon DocumentDB elastic cluster resources, describe them, and get information about them.
- CloudWatch permissions are used to verify service metrics.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "docdb-elastic:ListClusters",
            "docdb-elastic:GetCluster",
            "docdb-elastic:ListClusterSnapshots",
            "docdb-elastic:GetClusterSnapshot",
            "docdb-elastic:ListTagsForResource"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics",
            "cloudwatch:GetMetricStatistics"
         ],
         "Resource": "*"
      }
   ]
}
```

AmazonDocDBElasticFullAccess

This policy grants administrative permissions that allow a principal full access to all Amazon DocumentDB actions for Amazon DocumentDB elastic cluster.

This policy uses AWS tags (https://docs.aws.amazon.com/tag-editor/latest/userguide/tagging.html) within conditions to scope access to resources. If you are using a secret, it must be tagged with tag key `DocDBElasticFullAccess` and a tag value. If you are using a customer managed key, it must be tagged with tag key `DocDBElasticFullAccess` and a tag value.

The permissions in this policy are grouped as follows:

- Amazon DocumentDB elastic cluster permissions allow all Amazon DocumentDB actions.
- Some of the Amazon EC2 permissions in this policy are required to validate the passed resources in an API request. This is to make sure Amazon DocumentDB is able to successfully use the resources to provision and maintain the cluster. The rest of the Amazon EC2 permissions in this policy allow Amazon DocumentDB to create AWS resources that are needed to make it possible for you to connect to your clusters like a VPC endpoint.
- AWS KMS permissions are required for Amazon DocumentDB to be able to use the passed key to encrypt and decrypt the data at rest within the Amazon DocumentDB elastic cluster.

> **Note**
>
> The customer managed key must have a tag with key `DocDBElasticFullAccess` and a tag value.

- SecretsManager permissions are required to validate given secret and use it setup the admin user for Amazon DocumentDB elastic clusters.

> **Note**
>
> The secret used must have a tag with key `DocDBElasticFullAccess` and a tag value.

- IAM permissions are required for creating the service linked roles required for metrics and logs publishing.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DocdbElasticSid",
            "Effect": "Allow",
            "Action": [
                "docdb-elastic:CreateCluster",
                "docdb-elastic:UpdateCluster",
                "docdb-elastic:GetCluster",
                "docdb-elastic:DeleteCluster",
                "docdb-elastic:ListClusters",
                "docdb-elastic:CreateClusterSnapshot",
                "docdb-elastic:GetClusterSnapshot",
                "docdb-elastic:DeleteClusterSnapshot",
```

```
                "docdb-elastic:ListClusterSnapshots",
                "docdb-elastic:RestoreClusterFromSnapshot",
                "docdb-elastic:TagResource",
                "docdb-elastic:UntagResource",
                "docdb-elastic:ListTagsForResource",
                "docdb-elastic:CopyClusterSnapshot",
                "docdb-elastic:StartCluster",
                "docdb-elastic:StopCluster",
                "docdb-elastic:GetPendingMaintenanceAction",
                "docdb-elastic:ListPendingMaintenanceActions",
                "docdb-elastic:ApplyPendingMaintenanceAction"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "EC2Sid",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeVpcEndpoints",
                "ec2:DeleteVpcEndpoints",
                "ec2:ModifyVpcEndpoint",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeAvailabilityZones",
                "secretsmanager:ListSecrets"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaFirst": "docdb-elastic.amazonaws.com"
                }
            }
        },
        {
            "Sid": "KMSSid",
            "Effect": "Allow",
            "Action": [
```

```
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "docdb-elastic.*.amazonaws.com"
                    ],
                    "aws:ResourceTag/DocDBElasticFullAccess": "*"
                }
            }
        },
        {
            "Sid": "KMSGrantSid",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/DocDBElasticFullAccess": "*",
                    "kms:ViaService": [
                        "docdb-elastic.*.amazonaws.com"
                    ]
                },
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        },
        {
            "Sid": "SecretManagerSid",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:ListSecretVersionIds",
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:GetResourcePolicy"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "StringLike": {
                    "secretsmanager:ResourceTag/DocDBElasticFullAccess": "*"
                },
                "StringEquals": {
                    "aws:CalledViaFirst": "docdb-elastic.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudwatchSid",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "SLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-
elastic.amazonaws.com/AWSServiceRoleForDocDB-Elastic",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "docdb-elastic.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDB-ElasticServiceRolePolicy

You can't attach `AmazonDocDBElasticServiceRolePolicy` to your AWS Identity and Access Management entities. This policy is attached to a service-linked role that allows Amazon DocumentDB to perform actions on your behalf. For more information, see Service-linked roles in elastic clusters.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

Amazon DocumentDB updates to AWS managed policies

Change	Description	Date
AmazonDocDBElasticFullAccess, AmazonDocDBConsoleFullAccess - Change	Policies updated to add pending maintenance actions.	2/11/2025
AmazonDocDBElasticFullAccess, AmazonDocDBConsoleFullAccess - Change	Policies updated to add start/stop cluster and copy cluster snapshot actions.	2/21/2024
AmazonDocDBElasticReadOnlyAccess, AmazonDocDBElasticFullAccess - Change	Policies updated to add `cloudwatch:GetMetricData` action.	6/21/2023
AmazonDocDBElasticReadOnlyAccess - New policy	New managed policy for Amazon DocumentDB elastic clusters.	6/8/2023

Change	Description	Date
AmazonDocDBElasticFullAccess - New policy	New managed policy for Amazon DocumentDB elastic clusters.	6/5/2023
AmazonDocDB-ElasticServiceRolePolicy – New policy	Amazon DocumentDB creates a new AWSServiceRoleForDocDB-Elastic service linked role for Amazon DocumentDB elastic clusters.	11/30/2022
AmazonDocDBConsoleFullAccess - Change	Policy updated to add Amazon DocumentDB global and elastic cluster permissions.	11/30/2022
AmazonDocDBConsoleFullAccess, AmazonDocDBFullAccess, AmazonDocDBReadOnlyAccess - New Policy	Service launch.	1/19/2017

Amazon DocumentDB API permissions: actions, resources, and conditions reference

Use the following sections as a reference when you set up Using identity-based policies (IAM policies) for Amazon DocumentDB and write permissions policies that you can attach to an IAM identity (identity-based policies).

The following lists each Amazon DocumentDB API operation. Included in the list are the corresponding actions for which you can grant permissions to perform the action, the AWS resource that you can grant the permissions for, and condition keys that you can include for fine-grained access control. You specify the actions in the policy's `Action` field, the resource value in the policy's `Resource` field, and conditions in the policy's `Condition` field. For more information about conditions, see Specifying conditions in a policy.

You can use AWS-wide condition keys in your Amazon DocumentDB policies to express conditions. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

You can test IAM policies with the IAM policy simulator. It automatically provides a list of resources and parameters required for each AWS action, including Amazon DocumentDB actions. The IAM policy simulator determines the permissions that are required for each of the actions that you specify. For information about the IAM policy simulator, see Testing IAM Policies with the IAM Policy Simulator in the *IAM User Guide*.

> **Note**
>
> To specify an action, use the `rds:` prefix followed by the API operation name (for example, `rds:CreateDBInstance`).

The following lists Amazon RDS API operations and their related actions, resources, and condition keys.

Topics

- Amazon DocumentDB actions that support resource-level permissions
- Amazon DocumentDB actions that don't support resource-level permissions

Amazon DocumentDB actions that support resource-level permissions

Resource-level permissions provide the ability to specify the resources on which users are allowed to perform actions. Amazon DocumentDB has partial support for resource-level permissions. This means that for certain Amazon DocumentDB actions, you can control when users are allowed to use those actions based on conditions that have to be fulfilled, or specific resources that users are allowed to use. For example, you can grant users permission to modify only specific instances.

The following lists Amazon DocumentDB API operations and their related actions, resources, and condition keys.

> **Note**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon RDS. For more Amazon DocumentDB actions and permissions,

refer to Actions, resources, and condition keys for Amazon RDS in the *Service Authorization Reference*.

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
AddTagsToResource `rds:AddTagsToResource`	Instance `arn:aws:rds:` *region*`:`*account-id* `:db:`*db-instance-name*	`rds:db-tag`
	Subnet group `arn:aws:rds:` *region*`:`*account-id* `:subgrp:`*subnet-group-name*	`rds:subgrp-tag`
ApplyPendingMaintenanceAction `rds:ApplyPendingMaintenanceAction`	Instance `arn:aws:rds:` *region*`:`*account-id* `:db:`*db-instance-name*	`rds:db-tag`
CopyDBClusterSnapshot `rds:CopyDBClusterSnapshot`	Cluster snapshot `arn:aws:rds:` *region*`:`*account-id* `:cluster-snapshot:` *cluster-snapshot-name*	`rds:cluster-snapshot-tag`
CreateDBCluster `rds:CreateDBCluster`	Cluster `arn:aws:rds:` *region*`:`*account-id* `:cluster:` *db-cluster-name*	`rds:cluster-tag`

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
	Subnet group arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
CreateDBClusterParameterGroup rds:CreateDBClusterParameterGroup	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
CreateDBClusterSnapshot rds:CreateDBClusterSnapshot	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	Cluster snapshot arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBInstance rds:CreateDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
CreateDBSubnetGroup rds:CreateDBSubnetGroup	Subnet group arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DeleteDBInstance rds:DeleteDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
DeleteDBSubnetGroup rds:DeleteDBSubnetGroup	Subnet group arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribeDBClusterParameterGroups rds:DescribeDBClusterParameterGroups	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
DescribeDBClusterParameters rds:DescribeDBClusterParameters	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
DescribeDBClusters rds:DescribeDBClusters	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
DescribeDBClusterSnapshotAttributes rds:DescribeDBClusterSnapshotAttributes	Cluster snapshot arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
DescribeDBSubnetGroups rds:DescribeDBSubnetGroups	Subnet group arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribePendingMaintenanceActions rds:DescribePendingMaintenanceActions	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
FailoverDBCluster rds:FailoverDBCluster	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
ListTagsForResource rds:ListTagsForResource	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	Subnet group arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ModifyDBCluster rds:ModifyDBCluster	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
ModifyDBClusterParameterGroup rds:ModifyDBClusterParameterGroup	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
ModifyDBClusterSnapshotAttribute rds:ModifyDBClusterSnapshotAttribute	Cluster snapshot arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
ModifyDBInstance rds:ModifyDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
RebootDBInstance rds:RebootDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
RemoveTagsFromResource rds:RemoveTagsFromResource	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	Subnet group arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ResetDBClusterParameterGroup rds:ResetDBClusterParameterGroup	Cluster parameter group arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API Operations and Actions	Resources	Condition Keys
RestoreDBClusterFromSnapshot `rds:RestoreDBClusterFromSnapshot`	Cluster `arn:aws:rds:` *region*`:`*account-id* `:cluster:` *db-cluster-instance-name*	`rds:cluster-tag`
	Cluster snapshot `arn:aws:rds:` *region*`:`*account-id* `:cluster-snapshot:` *cluster-snapshot-name*	`rds:cluster-snapshot-tag`
RestoreDBClusterToPointInTime `rds:RestoreDBClusterToPointInTime`	Cluster `arn:aws:rds:` *region*`:`*account-id* `:cluster:` *db-cluster-instance-name*	`rds:cluster-tag`
	Subnet group `arn:aws:rds:` *region*`:`*account-id* `:subgrp:`*subnet-group-name*	`rds:subgrp-tag`

Amazon DocumentDB actions that don't support resource-level permissions

You can use all Amazon DocumentDB actions in an IAM policy to either grant or deny users permission to use that action. However, not all Amazon DocumentDB actions support resource-level permissions, which enable you to specify the resources on which an action can be performed. The following Amazon DocumentDB API actions currently don't support resource-level permissions. Therefore, to use these actions in an IAM policy, you must grant users permission to use all resources for the action by using a `*` wildcard for the `Resource` element in your statement.

- `rds:DescribeDBClusterSnapshots`

- `rds:DescribeDBInstances`

Authentication using IAM identity

Amazon DocumentDB users and applications can use IAM users and roles to authenticate into an Amazon DocumentDB cluster. Amazon DocumentDB IAM authentication is a password-less authentication method. Also, client applications do not send the password secrets to the Amazon DocumentDB cluster when using IAM roles/users. Instead, client connections are authenticated by AWS STS using temporary security tokens. Non-administrative users and applications can now use the same IAM identity ARN when connecting to different Amazon DocumentDB clusters and other AWS services.

You can also choose to use both password-based and IAM authentication to authenticate users and applications to an Amazon DocumentDB cluster. IAM authentication is available only in Amazon DocumentDB instance-based cluster version 5.0. IAM authentication using IAM identity ARNs is not supported for the Amazon DocumentDB primary user.

> **Note**
>
> The primary user can only be authenticated using existing password-based authentication.

Topics

- Getting started with authentication using IAM users and roles
- Configuring AWS compute types to authenticate to Amazon DocumentDB using AWS IAM
- Monitoring IAM authentication requests
- Using IAM authentication
- Drivers supporting IAM
- IAM identity authentication FAQ

Getting started with authentication using IAM users and roles

Amazon DocumentDB users and roles with IAM identities are created and managed in an `$external` database.

Creating a user

Connect as the primary user, then create an IAM user and role:

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:user/iamuser",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

Alternatively, add an Amazon DocumentDB user using an IAM role:

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:role/iamrole",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

Modifying an IAM user or role

Modify an existing IAM user:

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:user/iamuser",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

Modify an existing IAM role:

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:role/iamrole",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
```

```
);
```

To grant or revoke roles from an IAM user:

```
use $external;
db.grantRolesToUser(
  "arn:aws:iam::123456789123:user/iamuser",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

```
use $external;
db.revokeRolesFromUser(
  "arn:aws:iam::123456789123:user/iamuser",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

To grant or revoke roles from an IAM role:

```
use $external;
db.grantRolesToUser(
  "arn:aws:iam::123456789123:user/iamrole",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

```
use $external;
db.revokeRolesFromUser(
  "arn:aws:iam::123456789123:user/iamrole",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

Dropping an IAM user or role

To drop an existing IAM user:

```
use $external;
db.dropUser("arn:aws:iam::123456789123:user/iamuser");
```

To drop an existing IAM role:

```
use $external;
```

```
db.dropUser("arn:aws:iam::123456789123:role/iamrole");
```

Configure a connection URI to authenticate using AWS IAM

To authenticate using AWS IAM, use the following URI parameters: `authSource` as `$external` and `authMechanism` as `MONGODB-AWS`. If you are using an IAM user, the username and password fields are replaced by an Access Key and Secret Key respectively. If you are assuming an IAM role, attached to the environment you are in (for example, AWS Lambda function, Amazon EC2 instance). You don't need to specifically pass any credential when authenticating using the `MONGODB-AWS` mechanism. If you are using MongoDB drivers that support the `MONGODB-AWS` authentication mechanism, the drivers also have the ability to retrieve IAM role credentials from the compute instance (for example, Amazon EC2, Lambda function, and others). The following example uses a mongo shell to authenticate using `MONGODB-AWS` by passing an Access Key and Secret Key (of an IAM user) manually to demonstrate authentication against Amazon DocumentDB.

The following example uses Python code to authenticate using `MONGODB-AWS` without explicitly passing any credentials (using an IAM Role attached to the environment) to demonstrate authentication against Amazon DocumentDB.

```
##Create a MongoDB client, open a connection to Amazon DocumentDB using an IAM role
    client = pymongo.MongoClient('mongodb://<DocDBEndpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false&authSource=
%24external&authMechanism=MONGODB-AWS')
```

The following example uses a mongo shell to authenticate using `MONGODB-AWS` mechanism by passing an Access Key and Secret Key (of an IAM user) manually to demonstrate authentication against Amazon DocumentDB.

```
$ mongo 'mongodb://<access_key>:<secret_key>@<cluster_endpoint>:<db_port>/test?
authSource=%24external&authMechanism=MONGODB-AWS'
```

The following example uses a mongo shell to authenticate using `MONGODB-AWS` without explicitly passing any credentials (using IAM Role attached to the environment) to demonstrate authentication against Amazon DocumentDB.

```
$ mongo 'mongodb://<cluster_endpoint>:<db_port>/test?authSource=
%24external&authMechanism=MONGODB-AWS'
```

Configuring AWS compute types to authenticate to Amazon DocumentDB using AWS IAM

Using Amazon EC2/AWS Lambda/AWS Fargate

Amazon EC2 uses the following environment variables. If you have a IAM role attached to the EC2 instance or an execution IAM role associated with a Lambda function or an Amazon ECS task, then these variables are automatically populated and the driver can fetch these values from environment:

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_SESSION_TOKEN
```

For more information about environment variables, see Using Lambda environment variables in the *AWS Lambda Developer Guide*.

Using Amazon EKS

Assigning a role to your Amazon Elastic Kubernetes Service (Amazon EKS) pods will automatically setup the following two environment variables:

```
AWS_WEB_IDENTITY_TOKEN_FILE - path of web identity token file
AWS_ROLE_ARN - Name of IAM role to connect with
```

With the help of these variables, manually assume the role from your code using the AWS SDK call for `AssumeRoleWithWebIdentity`:

- Omit the `ProviderID` parameter.
- Find the value of the `WebIdentityToken` parameter in the file described in the `AWS_WEB_IDENTITY_TOKEN_FILE` environment variable.

For more info on Amazon EKS, see What is Amazon EKS in the *Amazon EKS User Guide*.

Monitoring IAM authentication requests

Using Amazon DocumentDB auditing

Go to the audit log folder in Amazon CloudWatch, and use different search patterns to get the logs for IAM authentication. For example, use `{ $.param.mechanism = "MONGODB-AWS" }` as the search pattern for “Search all log streams”.

For more info on supported events in auditing, see Auditing Amazon DocumentDB events.

Using Amazon CloudWatch metrics

`StsGetCallerIdentityCalls`: This metric shows how many `GetCallerIdentity` calls an Amazon DocumentDB instance is making to the regionalized AWS Security Token Service (AWS STS) endpoint. Please refer to the `MONGODB-AWS` authentication specification on why database instances need to make STS `GetCallerIdentity` calls.

Using IAM authentication

When you don’t want to manage the username and password in your own database, you can use IAM authentication. IAM authentication is available only in Amazon DocumentDB instance-based cluster version 5.0.

IAM authentication has a dependency on the STS service. We recommend you evaluate whether you can lower your connection rate when you are using IAM authentication for the connection and getting an STS throttling exception.

For IAM quotas, see IAM and AWS STS quotas in the *IAM User Guide*.

Drivers supporting IAM

Drivers that support Amazon DocumentDB 5.0 and the `MONGODB-AWS` authentication mechanism should work with the IAM authentication implementation in Amazon DocumentDB.

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

IAM identity authentication FAQ

Are there any samples I can refer to?

See these pages for sample use cases and configurations:

- How human users can authenticate to Amazon DocumentDB using IAM Users and IAM Roles
- Password-less authentication to Amazon DocumentDB using IAM Roles

I am getting an error while using my Python driver: "pymongo.errors.ConfigurationError: MONGODB-AWS authentication requires pymongo-auth-aws". How can I resolve this?

Please make sure you use the following statement while installing the Python driver with IAM authentication:

```
pip install 'pymongo[aws]'
```

This will install the additional AWS dependencies required for IAM authentication to work.

Will my connection drop when my IAM role temporary credentials expire?

No, the temporary IAM credentials are only used for establishing connection and authentication. Then all further authorization happens in the Amazon DocumentDB cluster. Even if IAM credentials rotate/expire, the connection will not drop or get stale.

Managing Amazon DocumentDB users

In Amazon DocumentDB, users authenticate to a cluster in conjunction with a password. Each cluster has primary sign-in credentials that are established during cluster creation.

> **Note**
>
> All new users created before **March 26, 2020** have been granted the `dbAdminAnyDatabase`, `readWriteAnyDatabase`, and `clusterAdmin` roles. It is recommended that you reevaluate all users and modify the roles as necessary to enforce least privilege for all users in your clusters.
> For more information, see Database access using Role-Based Access Control.

Primary and `serviceadmin` user

A newly created Amazon DocumentDB cluster has two users: the primary user and the `serviceadmin` user.

The *primary user* is a single, privileged user that can perform administrative tasks and create additional users with roles. When you connect to an Amazon DocumentDB cluster for the first time, you must authenticate using the primary sign-in credentials. The primary user receives these administrative permissions for an Amazon DocumentDB cluster when that cluster is created, and is granted the role of `root`.

The `serviceadmin` user is created implicitly when the cluster is created. Every Amazon DocumentDB cluster has a `serviceadmin` user that provides AWS the ability to manage your cluster. You cannot log in as, drop, rename, change the password, or change the permissions for `serviceadmin`. Any attempt to do so results in an error.

> **Note**
>
> The primary and `serviceadmin` users for an Amazon DocumentDB cluster cannot be deleted and the role of `root` for the primary user cannot be revoked.
> If you forget your primary user password, you can reset it using the AWS Management Console or the AWS CLI.

Creating additional users

After you connect as the primary user (or any user that has the role `createUser`), you can create a new user, as shown below.

```
db.createUser(
    {
        user: "sample-user-1",
        pwd: "password123",
        roles:
            [{"db":"admin", "role":"dbAdminAnyDatabase" }]
    }
)
```

To view user details, you can use the `show users` command as follows. You can additionally remove users with the `dropUser` command. For more information, see Common commands.

```
show users
{
```

```
    "_id" : "serviceadmin",
    "user" : "serviceadmin",
    "db" : "admin",
    "roles" : [
     {
            "role" : "root",
            "db" : "admin"
        }
    ]
},

{
    "_id" : "myPrimaryUser",
    "user" : "myPrimaryUser",
    "db" : "admin",
    "roles" : [
     {
            "role" : "root",
            "db" : "admin"
        }
    ]
},

{
    "_id" : "sample-user-1",
    "user" : "sample-user-1",
    "db" : "admin",
    "roles" : [
     {
            "role" : "dbAdminAnyDatabase",
            "db" : "admin"
     }
    ]
}
```

In the example above, the new user `sample-user-1` is attributed to the `admin` database. This is always the case for a new user. Amazon DocumentDB does not have the concept of an `authenticationDatabase` and thus all authentication is performed in the context of the `admin` database.

When creating users, if you omit the db field when specifying the role, Amazon DocumentDB will implicitly attribute the role to the database in which the connection is being issued against. For

example, if your connection is issued against the database `sample-database` and you run the following command, the user `sample-user-2` will be created in the `admin` database and will have `readWrite` permissions to the database `sample-database`.

```
db.createUser(
    {
        user: "sample-user-2",
        pwd: "password123",
        roles:
            ["readWrite"]
    }
)
```

Creating users with roles that are scoped across all databases (for example, `readInAnyDatabase`) require that you are either in the context of the `admin` database when creating the user or you explicitly state the database for the role when creating the user.

To switch the context of your database, you can use the following command.

```
use admin
```

To learn more about Role Based Access Control and enforcing least privilege amongst the users in your cluster, see Database access using Role-Based Access Control.

Automatically rotating passwords for Amazon DocumentDB

With AWS Secrets Manager, you can replace hardcoded credentials in your code (including passwords) with an API call to Secrets Manager to retrieve the secret programmatically. This helps ensure that the secret can't be compromised by someone examining your code, because the secret simply isn't there. Also, you can configure Secrets Manager to automatically rotate the secret for you according to a schedule that you specify. This enables you to replace long-term secrets with short-term ones, which helps to significantly reduce the risk of compromise.

Using Secrets Manager, you can automatically rotate your Amazon DocumentDB passwords (that is, *secrets*) using an AWS Lambda function that Secrets Manager provides.

For more information about AWS Secrets Manager and native integration with Amazon DocumentDB, see the following:

- Blog: How to rotate Amazon DocumentDB and Amazon Redshift credentials in AWS Secrets Manager
- What is AWS Secrets Manager?
- Rotate AWS Secrets Manager secrets
- Amazon DocumentDB credentials in Secrets Manager

Database access using Role-Based Access Control

You can restrict access to the actions that users can perform on databases using *role-based access control* (RBAC) in Amazon DocumentDB (with MongoDB compatibility). RBAC works by granting one or more roles to a user. These roles determine the operations that a user can perform on database resources. Amazon DocumentDB currently supports both built-in roles that are scoped at the database level, such as `read`, `readWrite`, `readAnyDatabase`, `clusterAdmin`, and user-defined roles that can be scoped to specific actions and granular resources such as collections based on your requirements.

Common use cases for RBAC include enforcing least privileges by creating users with read-only access to the databases or collections in a cluster, and multi-tenant application designs that enable a single user to access a given database or collection in a cluster.

> **Note**
>
> All new users created before **March 26, 2020** have been granted the `dbAdminAnyDatabase`, `readWriteAnyDatabase`, and `clusterAdmin` roles. It is recommended that you reevaluate all existing users and modify the roles as necessary to enforce least privileges for your clusters.

Topics

- RBAC concepts
- Getting started with RBAC built-in roles
- Getting started with RBAC user-defined roles
- Connecting to Amazon DocumentDB as a User
- Common commands

- Functional differences
- Limits
- Database access using Role-Based Access Control

RBAC concepts

The following are important terms and concepts related to role-based access control. For more information on Amazon DocumentDB users, see Managing Amazon DocumentDB users.

- **User** — An individual entity that can authenticate to the database and perform operations.
- **Password** — A secret that is used to authenticate the user.
- **Role** — Authorizes a user to perform actions on one or more databases.
- **Admin Database** — The database in which users are stored and authorized against.
- **Database (db)** — The namespace within clusters that contains collections for storing documents.

The following command creates a user named `sample-user`.

```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

In this example:

- `user: "sample-user"` — Indicates the user name.
- `pwd: "abc123"` — Indicates the user password.
- `role: "read", "db: "sample-database"` — Indicates that the user `sample-user` will have read permissions in `sample-database`.

The following example shows the output after you get the user `sample-user` with `db.getUser(sample-user)`. In this example, the user `sample-user` resides in the `admin` database but has the read role for the database `sample-database`.

```
{
    "_id" : "sample-user",
    "user" : "sample-user",
    "db" : "admin",
    "roles" : [
        {
            "db" : "sample-database",
            "role" : "read"
        }
    ]
}
```

User ID

Username

All users created in the *admin* database

User *sample-user* has read permissions in database *sample-database*

When creating users, if you omit the db field when specifying the role, Amazon DocumentDB will implicitly attribute the role to the database in which the connection is being issued against. For example, if your connection is issued against the database `sample-database` and you run the following command, the user `sample-user` will be created in the `admin` database and will have `readWrite` permissions to the database `sample-database`.

```
db.createUser({user: "sample-user", pwd: "abc123", roles: ["readWrite"]})
```

Output from this operation looks something like the following.

```
{
   "user":"sample-user",
   "roles":[
      {
         "db":"sample-database",
         "role":"readWrite"
      }
   ]
}
```

Creating users with roles that are scoped across all databases (for example, `readAnyDatabase`) require that you either be in the context of the `admin` database when creating the user, or you explicitly state the database for the role when creating the user. To issue commands against the `admin` database, you can use the command `use admin`. For more information, see Common commands.

Getting started with RBAC built-in roles

To help you get started with role-based access control, this section walks you through an example scenario of enforcing least privilege by creating roles for three users with different job functions.

- `user1` is a new manager that needs to be able to view and access all databases in a cluster.
- `user2` is a new employee that needs access to only one database, `sample-database-1`, in that same cluster.
- `user3` is an existing employee that needs to view and access a different database, `sample-database-2` that they didn't have access to before, in the same cluster.

At a point later, both `user1` and `user2` leave the company and so their access must be revoked.

To create users and grant roles, the user that you authenticate to the cluster with must have an associated role that can perform actions for `createUser` and `grantRole`. For example, the roles `admin` and `userAdminAnyDatabase` can both grant such abilities, for example. For actions per role, see Database access using Role-Based Access Control.

> **Note**
>
> In Amazon DocumentDB, all user and role operations (for example, `create`, `get`, `drop`, `grant`, `revoke`, etc.) are implicitly performed in the `admin` database whether or not you are issuing commands against the `admin` database.

First, to understand what the current users and roles are in the cluster, you can run the `show users` command, as in the following example. You will see two users, `serviceadmin` and the primary user for the cluster. These two users always exist and cannot be deleted. For more information, see Managing Amazon DocumentDB users.

```
show users
```

For `user1`, create a role with read and write access to all databases in the entire cluster with the following command.

```
db.createUser({user: "user1", pwd: "abc123", roles: [{role: "readWriteAnyDatabase", db:
 "admin"}]})
```

Output from this operation looks something like the following.

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

For `user2`, create a role with read-only access to the database `sample-database-1` with the following command.

```
db.createUser({user: "user2", pwd: "abc123", roles: [{role: "read", db: "sample-
database-1"}]})
```

Output from this operation looks something like the following.

```
{
   "user":"user2",
   "roles":[
      {
         "role":"read",
         "db":"sample-database-1"
      }
   ]
}
```

To simulate the scenario that `user3` is an existing user, first create the user `user3`, and then assign a new role to `user3`.

```
db.createUser({user: "user3", pwd: "abc123", roles: [{role: "readWrite", db: "sample-
database-1"}]})
```

Output from this operation looks something like the following.

```
{
   "user":"user3",
   "roles":[
```

```
      {
         "role":"readWrite",
         "db":"sample-database-1"
      }
   ]
}
```

Now that the user `user3` has been created, assign `user3` the role `read` to `sample-database-2`.

```
db.grantRolesToUser("user3", [{role: "read", db: "sample-database-2"}])
```

Lastly, both `user1` and `user2` leave the company and need their access to the cluster revoked. You can do this by dropping the users, as follows.

```
db.dropUser("user1")
db.dropUser("user2")
```

To ensure that all users have the appropriate roles, you can list all users with the following command.

```
show users
```

Output from this operation looks something like the following.

```
{
   "_id":"serviceadmin",
   "user":"serviceadmin",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"master-user",
   "user":"master-user",
   "db":"admin",
   "roles":[
      {
```

```
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"user3",
   "user":"user3",
   "db":"admin",
   "roles":[
      {
         "db":"sample-database-2",
         "role":"read"
      },
      {
         "db":"sample-database-1",
         "role":"readWrite"
      }
   ]
}
```

Getting started with RBAC user-defined roles

To help you get started with user-defined roles, this section walks you through an example scenario of enforcing least privilege by creating roles for three users with different job functions.

In this example, the following applies:

- `user1` is a new manager that needs to be able to view and access all databases in a cluster.
- `user2` is a new employee that needs only the ‘find’ action to only one database, `sample-database-1`, in that same cluster.
- `user3` is an existing employee that needs to view and access a specific collection, col2 in a different database, `sample-database-2` that they didn't have access to before, in the same cluster.
- For `user1`, create a role with read and write access to all databases in the entire cluster with the following command.

```
db.createUser(
 {
     user: "user1", pwd: "abc123",
```

```
     roles: [{role: "readWriteAnyDatabase", db: "admin"}]
 }
 )
```

Output from this operation looks something like the following.

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

For `user2`, create a role with 'find' privileges to all collections in the database `sample-database-1` with the following command. Note that this role would ensure that any associated users can only run find queries.

```
db.createRole(
{
    role: "findRole",
    privileges: [
    {
        resource: {db: "sample-database-1", collection: ""}, actions: ["find"]
    }],
    roles: []
}
)
```

Output from this operation looks something like the following.

```
{
   "role":"findRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
```

```
            "find"
         ]
      }
   ],
   "roles":[

   ]
}
```

Next, create the user (`user2`) and attach the recently created role `findRole` to the user.

```
db.createUser(
{
    user: "user2",
    pwd: "abc123",
    roles: []
})

 db.grantRolesToUser("user2",["findRole"])
```

To simulate the scenario that `user3` is an existing user, first create the user `user3`, and then create a new role called collectionRole which we will in the next step assing to `user3`.

Now you can assign a new role to `user3`. This new role will allow `user3` to be able to insert, update, delete and find access to one specific collection col2 in `sample-database-2`.

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
    roles: []
})

db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
```

```
}
)
```

Output from this operation looks something like the following.

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

Now that the user `user3` has been created, you can grant `user3` the role `collectionFind`.

```
db.grantRolesToUser("user3",["collectionRole"])
```

Lastly, both `user1` and `user2` leave the company and need their access to the cluster revoked. You can do this by dropping the users, as follows.

```
db.dropUser("user1")
db.dropUser("user2")
```

To ensure that all users have the appropriate roles, you can list all users with the following command.

```
show users
```

Output from this operation looks something like the following.

```
{
   "_id":"serviceadmin",
   "user":"serviceadmin",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"master-user",
   "user":"master-user",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"user3",
   "user":"user3",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"collectionRole"
      }
   ]
}
```

Connecting to Amazon DocumentDB as a User

When connecting to an Amazon DocumentDB cluster, you connect in the context of a particular database. By default, if you don't specify a database in your connection string, you are automatically connected to the cluster in the context of the `test` database. All collection level commands like `insert` and `find` are issued against collections in the `test` database.

To see the database you are in the context of or — in other words — issuing commands against, use the db command in the mongo shell, as follows.

Query:

```
db
```

Output:

```
test
```

Although the default connection might be in the context of the `test` database, that does not necessarily mean that the user associated with the connection is authorized to perform actions on the `test` database. In the preceding example scenario, if you authenticate as the user `user3`, which has the `readWrite` role for the `sample-database-1` database, the default context of your connection is the `test` database. However, if you try to insert a document into a collection on the `test` database, you will receive an *Authorization failure* error message. This is because that user is not authorized to perform that command on that database, as shown below.

Query:

```
db
```

Output:

```
test
```

Query:

```
db.col.insert({x:1})
```

Output:

```
WriteCommandError({ "ok" : 0, "code" : 13, "errmsg" : "Authorization failure" })
```

If you change the context of your connection to the `sample-database-1` database, you can write to the collection for which the user has the authorization to do so.

Query:

```
use sample-database-1
```

Output:

```
switched to db sample-database-1
```

Query:

```
db.col.insert({x:1})
```

Output:

```
WriteResult({ "nInserted" : 1})
```

When you authenticate to a cluster with a particular user, you can also specify the database in the connection string. Doing so removes the necessity to perform the `use` command after the user has been authenticated to the `admin` database.

The following connection string authenticates the user against the `admin` database, but the context of the connection will be against the `sample-database-1` database.

```
mongo "mongodb://user3:abc123@sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
sample-database-2"
```

Common commands

This section provides examples of common commands using role-based access control in Amazon DocumentDB. You must be in the context of the `admin` database to create and modify users and roles. You can use the `use admin` command to switch to the `admin` database.

> **Note**
>
> Modifications to the users and roles will implicitly occur in the `admin` database. Creating users with roles that are scoped across all databases (for example, `readAnyDatabase`) requires that you are either in the context of the `admin` database (that is, `use admin`) when creating the user, or you explicitly state the database for the role when creating the user (as shown in Example 2 in this section).

Example 1: Create a user with `read` role for the database `foo`.

```
db.createUser({user: "readInFooBar", pwd: "abc123", roles: [{role: "read", db:
 "foo"}]})
```

Output from this operation looks something like the following.

```
{
   "user":"readInFooBar",
   "roles":[
      {
         "role":"read",
         "db":"foo"
      }
   ]
}
```

Example 2: Create a user with read access on all databases.

```
db.createUser({user: "readAllDBs", pwd: "abc123", roles: [{role: "readAnyDatabase", db:
 "admin"}]})
```

Output from this operation looks something like the following.

```
{
   "user":"readAllDBs",
   "roles":[
      {
         "role":"readAnyDatabase",
         "db":"admin"
      }
   ]
}
```

Example 3: Grant `read` role to an existing user on a new database.

```
db.grantRolesToUser("readInFooBar", [{role: "read", db: "bar"}])
```

Example 4: Update a user's role.

```
db.updateUser("readInFooBar", {roles: [{role: "read", db: "foo"}, {role: "read", db:
 "baz"}]})
```

Example 5: Revoke access to a database for a user.

```
db.revokeRolesFromUser("readInFooBar", [{role: "read", db: "baz"}])
```

Example 6: Describe a built-in role.

```
db.getRole("read", {showPrivileges:true})
```

Output from this operation looks something like the following.

```
{
   "role":"read",
   "db":"sample-database-1",
   "isBuiltin":true,
   "roles":[

   ],
   "inheritedRoles":[

   ],
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "changeStream",
            "collStats",
            "dbStats",
            "find",
            "killCursors",
            "listCollections",
            "listIndexes"
         ]
      }
   ],
   "inheritedPrivileges":[
      {
```

```
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "changeStream",
            "collStats",
            "dbStats",
            "find",
            "killCursors",
            "listCollections",
            "listIndexes"
         ]
      }
   }
```

Example 7: Drop a user from the cluster.

```
db.dropUser("readInFooBar")
```

Output from this operation looks something like the following.

```
true
```

Example 8: Create a role with read and write access to a specific collection

```
db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
}
)
```

Output from this operation looks something like the following.

```
{
```

```
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

Example 9: Create a user and assign a user defined role

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
    roles: []
})

db.grantRolesToUser("user3",["collectionRole"])
```

Example 10: Grant additional privileges to a user defined role

```
db.grantPrivilegesToRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col1" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

Example 11: Remove privileges from a user defined role

```
db.revokePrivilegesFromRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col2" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

Example 12: Update an existing user defined role

```
db.updateRole(
  "collectionRole",
  {
    privileges: [
    {
        resource: {db: "sample-database-3", collection: "sample-collection-3"},
 actions: ["find", "update", "insert", "remove"]
    }],
    roles: []
  }
)
```

Functional differences

In Amazon DocumentDB, user and role definitions are stored in the `admin` database and users are authenticated against the `admin` database. This functionality differs from the MongoDB Community Edition, but is consistent with MongoDB Atlas.

Amazon DocumentDB also supports change streams, which provide a time-ordered sequence of change events that occur within your cluster's collections. The `listChangeStreams` action is applied at the cluster level (that is, across all databases), and the `modifyChangeStreams` action can be applied at the database level and cluster level.

Limits

The following table contains the limits for Role-Based Access Control in Amazon DocumentDB.

Description	Limit
Number of users per cluster	1000
Number of roles associated with a user	1000
Number of user-defined roles	100
Number of resources associated with a privilege	100

Database access using Role-Based Access Control

With role-based access control, you can create a user and grant it one or more roles to determine what operations that user can perform in a database or cluster.

The following is a list of built-in roles that are currently supported in Amazon DocumentDB.

> **Note**
>
> In Amazon DocumentDB 4.0 and 5.0, the `ListCollection` and `ListDatabase` commands can optionally use the `authorizedCollections` and `authorizedDatabases` parameters to list the collections and databases that the user has permission to access with requiring the `listCollections` and `listDatabase` roles, respectively. Also, users now have the ability to kill their own cursors without requiring the `KillCursor` role.

Database user

Role name	Description	Actions

`read`	Grants a user read access to the specified database.	`changeStreams`
		`collStats`
		`dbStats`
		`find`

Role name	Description	Actions

		`killCursors`
		
		`listIndexes`
		
		`listCollections`

`readWrite`	Grants the user read and write access to the specified database.	All actions from `read` permissions.
		
		`createCollection`
		
		`dropCollection`
		
		`createIndex`
		
		`dropIndex`
		
		`insert`
		
		`killCursors`
		
		`listIndexes`
		
		`listCollections`
		
		`remove`
		
		`update`

Cluster user

Role name	Description	Actions
readAnyDatabase	Grants a user read access to all databases in the cluster.	All actions from `read` permissions.

Role name	Description	Actions

		`listChangeStreams`
		
		`listDatabases`

`readWriteAnyDatabase`	Grants a user read and write access to all databases in the cluster.	All actions from `readWrite` permissions.
		
		`listChangeStreams`
		
		`listDatabases`

`userAdminAnyDatabase`	Grants a user the ability to assign and modify the roles or privileges any user has to the specified database.	`changeCustomData`
		
		`changePassword`
		
		`createUser`
		
		`dropRole`
		
		`dropUser`
		
		`grantRole`
		
		`listDatabases`
		
		`revokeRole`
		
		`viewRole`
		
		`viewUser`

Role name	Description	Actions

`dbAdminAnyDatabase`	Grants a user the ability to perform database administration roles on any specified database.	All actions from `dbAdmin` permissions.
		
		`dropCollection`
		
		`listDatabases`
		
		`listChangeStreams`
		
		`modifyChangeStreams`

Superuser

Role name	Description	Actions
`root`	Grants a user access to the resources and operations of all the following roles combined: `readWriteAnyDatabase`, `dbAdminAnyDatabase`, `userAdminAnyDatabase`, `clusterAdmin`, `restore`, and `backup`.	All actions from `readWriteAnyDatabase`, `dbAdminAnyDatabase`, `userAdminAnyDatabase`, `clusterAdmin`, `restore`, and `backup`.

Database administrator

Role name	Description	Actions
dbAdmin	Grants a user the ability to perform administrative tasks on the specified database.	bypassDocumentValidation collMod collStats createCollection createIndex dropCollection dropDatabase dropIndex dbStats find killCursors listIndexes listCollections modifyChangeStreams
dbOwner	Grants a user the ability to perform any administrative tasks on the specified database by combining the roles dbAdmin and readWrite .	All actions from dbAdmin and readWrite .

Cluster administrator

role Name	Description	Actions

`clusterAdmin`	Grants a user the greatest cluster management access by combining the `clusterManager` , `clusterMonitor` , and `hostManager` roles.	All actions from `clusterManager` , `clusterMonitor` , and `hostManager` .
		
		`listChangeStreams`
		
		`dropDatabase`
		
		`modifyChangeStreams`

`clusterManager`	Grants a user the ability to take management and monitoring actions on the specified cluster.	`listChangeStreams`
		
		`listSessions`
		
		`modifyChangeStreams`
		
		`replSetGetConfig`

`clusterMonitor`	Grants a user the ability to have read-only access to monitoring tools.	`collStats`
		
		`dbStats`
		
		`find`
		
		`getParameter`
		
		`hostInfo`
		
		`indexStats`
		
		`killCursors`

role Name	Description	Actions
		listChangeStreams listCollections listDatabases listIndexes listSessions replSetGetConfig serverStatus top
hostManager	Grants a user the ability to monitor and manage servers.	auditConfigure killCursors killAnyCursor killAnySession killop

Backup administrator

Role name	Description	Actions
backup	Grants a user the access needed to back up data.	getParameter insert find

Role name	Description	Actions

		`listChangeStreams`
		`listCollections`
		`listDatabases`
		`listIndexes`
		`update`

Role name	Description	Actions

`restore`	Grants a user the access needed to restore data.	`bypassDocumentValidation`
		`changeCustomData`
		`changePassword`
		`collMod`
		`createCollection`
		`createIndex`
		`createUser`
		`dropCollection`
		`dropRole`
		`dropUser`
		`getParameter`
		`grantRole`
		`find`
		`insert`
		`listCollections`
		`modifyChangeStreams`
		`revokeRole`

Role name	Description	Actions

		`remove`
		`viewRole`
		`viewUser`
		`update`

Logging and monitoring in Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) provides a variety of Amazon CloudWatch metrics that you can monitor to determine the health and performance of your Amazon DocumentDB clusters and instances. You can view Amazon DocumentDB metrics using various tools, including the Amazon DocumentDB console, the AWS CLI, the Amazon CloudWatch console, and the CloudWatch API. For more information about monitoring, see Monitoring Amazon DocumentDB.

In addition to Amazon CloudWatch metrics, you can use the profiler to log the execution time and details of operations that were performed on your cluster. Profiler is useful for monitoring the slowest operations on your cluster to help you improve individual query performance and overall cluster performance. When enabled, operations are logged to Amazon CloudWatch Logs and you can use CloudWatch Insight to analyze, monitor, and archive your Amazon DocumentDB profiling data. For more information, see Profiling Amazon DocumentDB operations.

Amazon DocumentDB also integrates with AWS CloudTrail, a service that provides a record of actions taken by users, roles, or an AWS service in Amazon DocumentDB (with MongoDB compatibility). CloudTrail captures all AWS CLI API calls for Amazon DocumentDB as events, including calls from the Amazon DocumentDB AWS Management Console and from code calls to the Amazon DocumentDB SDK. For more information, see Logging Amazon DocumentDB API calls with AWS CloudTrail.

With Amazon DocumentDB, you can audit events that were performed in your cluster. Examples of logged events include successful and failed authentication attempts, dropping a collection in a database, or creating an index. By default, auditing is disabled on Amazon DocumentDB and requires that you opt in to this feature. For more information, see Auditing Amazon DocumentDB events.

Updating your Amazon DocumentDB TLS certificates

Topics

- Updating your application and Amazon DocumentDB cluster
- Automatic server certificate rotation
- Troubleshooting
- Frequently Asked Questions

The certificate authority (CA) certificate for Amazon DocumentDB clusters was updated in August of 2024. If you are using Amazon DocumentDB clusters with Transport Layer Security (TLS) enabled (the default setting) and you have not rotated your client application and server certificates, the following steps are required to mitigate connectivity issues between your application and your Amazon DocumentDB clusters.

- Step 1: Download the new CA certificate and update your application
- Step 2: Update the server certificate

The CA and server certificates were updated as part of standard maintenance and security best practices for Amazon DocumentDB. Client applications must add the new CA certificates to their trust stores, and existing Amazon DocumentDB instances must be updated to use the new CA certificates before this expiration date.

Updating your application and Amazon DocumentDB cluster

Follow the steps in this section to update your application's CA certificate bundle (Step 1) and your cluster's server certificates (Step 2). Before you apply the changes to your production environments, we strongly recommend testing these steps in a development or staging environment.

> **Note**
>
> You must complete Steps 1 and 2 in each AWS Region in which you have Amazon DocumentDB clusters.

Step 1: Download the new CA certificate and update your application

Download the new CA certificate and update your application to use the new CA certificate to create TLS connections to Amazon DocumentDB. Download the new CA certificate bundle from https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem. This operation downloads a file named `global-bundle.pem`.

> **Note**
>
> If you are accessing the keystore that contains both the old CA certificate (`rds-ca-2019-root.pem`) and the new CA certificates (`rds-ca-rsa2048-g1`, `rds-ca-rsa4096-g1`, `rds-ca-ecc384-g1`), verify that the keystore selects `global-bundle`.

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

Next, update your applications to use the new certificate bundle. The new CA bundle contains both the old CA certificate (rds-ca-2019) and the new CA certificates (rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, rds-ca-ecc384-g1). By having both CA certificates in the new CA bundle, you can update your application and cluster in two steps.

For Java applications, you must create a new trust store with the new CA certificate. For instructions, see the Java tab in the Connecting with TLS enabled topic.

To verify that your application is using the latest CA certificate bundle, see How can I be sure that I'm using the newest CA bundle? If you're already using the latest CA certificate bundle in your application, you can skip to Step 2.

For examples of using a CA bundle with your application, see Encrypting data in transit and Connecting with TLS enabled.

> **Note**
>
> Currently, the MongoDB Go Driver 1.2.1 only accepts one CA server certificate in `sslcertificateauthorityfile`. Please see Connecting with TLS enabled for connecting to Amazon DocumentDB using Go when TLS is enabled.

Step 2: Update the server certificate

After the application has been updated to use the new CA bundle, the next step is to update the server certificate by modifying each instance in an Amazon DocumentDB cluster. To modify instances to use the new server certificate, see the following instructions.

Amazon DocumentDB provides the following CAs to sign the DB server certificate for a DB instance:

- **rds-ca-ecc384-g1**—Uses a certificate authority with ECC 384 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation. This is only supported on Amazon DocumentDB 4.0 and 5.0.
- **rds-ca-rsa2048-g1**—Uses a certificate authority with RSA 2048 private key algorithm and SHA256 signing algorithm in most AWS regions. This CA supports automatic server certificate rotation.
- **rds-ca-rsa4096-g1**—Uses a certificate authority with RSA 4096 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation.

> **Note**
>
> If you are using the AWS CLI, you can see the validities of the certificate authorities listed above by using describe-certificates.

These CA certificates are included in the regional and global certificate bundle. When you use the rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, or rds-ca-ecc384-g1 CA with a database, Amazon DocumentDB manages the DB server certificate on the database. Amazon DocumentDB rotates the DB server certificate automatically before it expires.

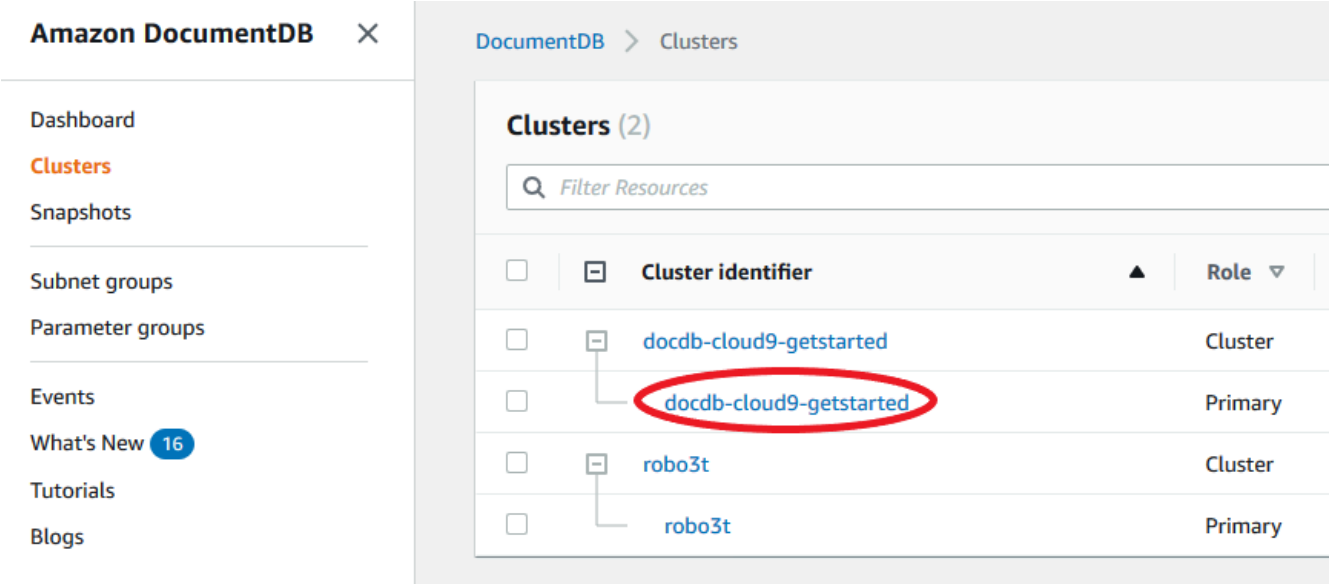
> **Note**
>
> Amazon DocumentDB does not require a reboot for certificate rotation if your cluster is running on the following engine patch versions:
>
> - Amazon DocumentDB 3.6: 1.0.208662 or greater
> - Amazon DocumentDB 4.0: 2.0.10179 or greater
> - Amazon DocumentDB 5.0: 3.0.4780 or greater

> You can determine the current Amazon DocumentDB engine patch version by running the following command: `db.runCommand({getEngineVersion: 1})`.
> Before updating the server certificate, ensure that you have completed Step 1.

Using the AWS Management Console

Complete the following steps to identify and rotate the old server certificate for your existing Amazon DocumentDB instances using the AWS Management Console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your clusters reside.
3. In the navigation pane on the left side of the console, choose **Clusters**.
4. You may need to identify which instances are still on the old server certificate (`rds-ca-2019`). You can do this in the **Certificate authority** column which is located on the far right of the **Clusters** table.
5. In the **Clusters** table, you'll see the column **Cluster identifier** on the far left. Your instances are listed under clusters, similar to the screenshot below.



6. Check the box to the left of the instance you are interested in.
7. Choose **Actions** and then choose **Modify**.

8. Under **Certificate authority**, select the new server certificate (`rds-ca-rsa2048-g1`) for this instance.
9. You can see a summary of the changes on the next page. Note that there is an extra alert to remind you to ensure that your application is using the latest certificate CA bundle before modifying the instance to avoid causing an interruption in connectivity.
10. You can choose to apply the modification during your next maintenance window or apply immediately. If your intention is to modify the server certificate immediately, use the **Apply Immediately** option.
11. Choose **Modify instance** to complete the update.

Using the AWS CLI

Complete the following steps to identify and rotate the old server certificate for your existing Amazon DocumentDB instances using the AWS CLI.

1. To modify the instances immediately, execute the following command for each instance in the cluster.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. To modify the instances in your clusters to use the new CA certificate during your cluster's next maintenance window, execute the following command for each instance in the cluster.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

Automatic server certificate rotation

Amazon DocumentDB supports automatic server certificate rotation. The server certificate is the leaf certificate issued to each cluster instance. As opposed to the root CA certificates, the server certificates have short (12 months) validity and Amazon DocumentDB automatically handles their rotation without any action from you. Amazon DocumentDB uses the same root CA for this automatic rotation, so you don't need to download a new CA bundle.

> **Important**
>
> When connecting to your Amazon DocumentDB cluster, we recommend that you trust the root CA bundle as opposed to directly trusting each server certificate. This will prevent connection errors after the server certificate is rotated. See Connecting with TLS enabled.

Amazon DocumentDB attempts to rotate your server certificate in your preferred maintenance window at the server certificate half life. The new server certificate is valid for 12 months.

Use the `describe-db-engine-versions` command and inspect the `SupportsCertificateRotationWithoutRestart` flag to identify whether the engine version supports rotating the certificate without restart.

> **Note**
>
> Amazon DocumentDB supports server certificate rotation without restarts if your cluster is running on the following engine patch versions:
>
> - Amazon DocumentDB 3.6: 1.0.208662 or greater
> - Amazon DocumentDB 4.0: 2.0.10179 or greater
> - Amazon DocumentDB 5.0: 3.0.4780 or greater
>
> You can determine the current Amazon DocumentDB engine patch version by running this command: `db.runCommand({getEngineVersion: 1})`.
> If you are using an older engine patch version, Amazon DocumentDB will rotate the server certificate and schedule a database restart event on your preferred maintenance window.

Troubleshooting

If you are having issues connecting to your cluster as part of the certificate rotation, we suggest the following:

- **Verify that your clients are using the latest certificate bundle.** See How can I be sure that I'm using the newest CA bundle?.

- **Verify that your instances are using the latest certificate.** See How do I know which of my Amazon DocumentDB instances are using the old/new server certificate?.
- **Verify that the latest certificate CA is being utilized by your application.** Some drivers, like Java and Go, require extra code to import multiple certificates from a certificate bundle to the trust store. For more information on connecting to Amazon DocumentDB with TLS, see Connecting programmatically to Amazon DocumentDB.
- **Contact support.** If you have questions or issues, contact Support.

Frequently Asked Questions

The following are answers to some common questions about TLS certificates.

What if I have questions or issues?

If you have questions or issues, contact Support.

How do I know whether I'm using TLS to connect to my Amazon DocumentDB cluster?

You can determine whether your cluster is using TLS by examining the `tls` parameter for your cluster's cluster parameter group. If the `tls` parameter is set to `enabled`, you are using the TLS certificate to connect to your cluster. For more information, see Managing Amazon DocumentDB cluster parameter groups.

Why are you updating the CA and server certificates?

The Amazon DocumentDB CA and server certificates are being updated as part of standard maintenance and security best practices for Amazon DocumentDB.

What happens if I don't take any action by the expiration date?

If you are using TLS to connect to your Amazon DocumentDB cluster with an expired CA certificate, your applications that connect via TLS will no longer be able to communicate with the Amazon DocumentDB cluster.

Amazon DocumentDB will not rotate your database certificates automatically before expiration. You must update your applications and clusters to use the new CA certificates before or after the expiration date.

How do I know which of my Amazon DocumentDB instances are using the old/new server certificate?

To identify the Amazon DocumentDB instances that still use the old server certificate, you can use either the Amazon DocumentDB AWS Management Console or the AWS CLI.

Using the AWS Management Console

To identify the instances in your clusters that are using the older certificate

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your instances reside.
3. In the navigation pane on the left side of the console, choose **Clusters**.
4. The **Certificate authority** column (near the far right of the table) shows which instances are still on the old server certificate (`rds-ca-2019`) and the new server certificate (`rds-ca-rsa2048-g1`).

Using the AWS CLI

To identify the instances in your clusters that are using the older server certificate, use the `describe-db-clusters` command with the following .

```
aws docdb describe-db-instances \
    --filters Name=engine,Values=docdb \
    --query 'DBInstances[*].
{CertificateVersion:CACertificateIdentifier,InstanceID:DBInstanceIdentifier}'
```

How do I modify individual instances in my Amazon DocumentDB cluster to update the server certificate?

We recommend that you update server certificates for all instances in a given cluster at the same time. To modify the instances in your cluster, you can use either the console or the AWS CLI.

> **Note**
>
> Before updating the server certificate, ensure that you have completed Step 1.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your clusters reside.
3. In the navigation pane on the left side of the console, choose **Clusters**.
4. The **Certificate authority** column (near the far right of the table) shows which instances are still on the old server certificate (`rds-ca-2019`).
5. In the **Clusters** table, under **Cluster identifier**, select an instance to modify.
6. Choose **Actions** and then choose **Modify**.
7. Under **Certificate authority**, select the new server certificate (`rds-ca-rsa2048-g1`) for this instance.
8. You can see a summary of the changes on the next page. Note that there is an extra alert to remind you to ensure that your application is using the latest certificate CA bundle before modifying the instance to avoid causing an interruption in connectivity.
9. You can choose to apply the modification during your next maintenance window or apply immediately.
10. Choose **Modify instance** to complete the update.

Using the AWS CLI

Complete the following steps to identify and rotate the old server certificate for your existing Amazon DocumentDB instances using the AWS CLI.

1. To modify the instances immediately, execute the following command for each instance in the cluster.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. To modify the instances in your clusters to use the new CA certificate during your cluster's next maintenance window, execute the following command for each instance in the cluster.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

What happens if I add a new instance to an existing cluster?

All new instances that are created use the old server certificate and require TLS connections using the old CA certificate. Any new Amazon DocumentDB instances created after January 25, 2024 will default to using the new certificate rds-ca-rsa2048-g1.

What happens if there is an instance replacement or failover on my cluster?

If there is an instance replacement in your cluster, the new instance that is created continues to use the same server certificate that the instance was previously using. We recommend that you update server certificates for all instances at the same time. If a failover occurs in the cluster, the server certificate on the new primary is used.

If I'm not using TLS to connect to my cluster, do I still need to update each of my instances?

We highly recommend enabling TLS. In the event that you do not enable TLS, we still recommend rotating the certificates on your Amazon DocumentDB instances in the event you plan to use TLS to connect to your clusters in the future. If you never plan to use TLS to connect to your Amazon DocumentDB clusters, no action is needed.

If I'm not using TLS to connect to my cluster but I plan to in the future, what should I do?

If you created a cluster before January, 2024, follow Step 1 and Step 2 in the previous section to ensure that your application is using the updated CA bundle, and that each Amazon DocumentDB instance is using the latest server certificate. If you create a cluster after January 25, 2024, your cluster will already have the latest server certificate (rds-ca-rsa2048-g1). To verify that your application is using the latest CA bundle, see If I'm not using TLS to connect to my cluster, do I still need to update each of my instances?

Can the deadline be extended beyond August, 2024?

If your applications are connecting via TLS, the deadline cannot be extended.

How can I be sure that I'm using the newest CA bundle?

To verify that you have the newest bundle, use the following command. To run this command, you must have java installed and the java tools need to be in the PATH variable of your shell. For more information, see Using Java

macOS and Amazon Linux

```
keytool -printcert -v -file global-bundle.pem
```

Windows

```
keytool -printcert -v -file global-bundle.p7b
```

Why do I see "RDS" in the name of the CA bundle?

For certain management features, such as certificate management, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).

When will the new certificate expire?

The new server certificate will expire (generally) as follows:

- **rds-ca-rsa2048-g1**—Expires 2061
- **rds-ca-rsa4096-g1**—Expires 2121
- **rds-ca-ecc384-g1**—Expires 2121

What kind of errors will I see if I don't take action before the certificate expires?

Error messages will vary depending on your driver. In general, you'll see certificate validation errors that contain the string "certificate has expired".

If I applied the new server certificate, can I revert it back to the old server certificate?

If you need to revert an instance to the old server certificate, we recommend that you do so for all instances in the cluster. You can revert the server certificate for each instance in a cluster by using the AWS Management Console or the AWS CLI.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your clusters reside.
3. In the navigation pane on the left side of the console, choose **Clusters**.
4. In the **Clusters** table, under **Cluster identifier**, select an instance to modify. Choose **Actions**, and then choose **Modify**.
5. Under **Certificate authority**, you can select the old server certificate (`rds-ca-2019`).
6. Choose **Continue** to view a summary of your modifications.
7. In this resulting page, you can choose to schedule your modifications to be applied in the next maintenance window or apply your modifications immediately. Make your selection, and choose **Modify instance**.

> **Note**
>
> If you choose to apply your modifications immediately, any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing this option can cause unexpected downtime.

Using the AWS CLI

```
aws docdb modify-db-instance --db-instance-identifier <db_instance_name> ca-
certificate-identifier rds-ca-2019 <--apply-immediately | --no-apply-immediately>
```

If you choose `--no-apply-immediately`, the changes will be applied during the cluster's next maintenance window.

If I restore from a snapshot or a point in time restore, will it have the new server certificate?

If you restore a snapshot or perform a point-in-time restore after August, 2024, the new cluster that is created will use the new CA certificate.

What if I'm having issues connecting directly to my Amazon DocumentDB cluster from any Mac OS?

Mac OS has updated the requirements for trusted certificates. Trusted certificates must now be valid for 397 days or fewer (see https://support.apple.com/en-us/HT211025).

> **Note**
>
> This restriction is observed in newer versions of Mac OS.

Amazon DocumentDB instance certificates are valid for over four years, longer than the Mac OS maximum. In order to connect directly to an Amazon DocumentDB cluster from a computer running Mac OS, you must allow invalid certificates when creating the TLS connection. In this case, invalid certificates mean that the validity period is longer than 397 days. You should understand the risks before allowing invalid certificates when connecting to your Amazon DocumentDB cluster.

To connect to an Amazon DocumentDB cluster from Mac OS using the AWS CLI, use the `tlsAllowInvalidCertificates` parameter.

```
mongo --tls --host <hostname> --username <username> --password <password> --port 27017
 --tlsAllowInvalidCertificates
```

Updating your Amazon DocumentDB TLS certificates — GovCloud

Topics

- Updating your application and Amazon DocumentDB cluster
- Troubleshooting
- Frequently Asked Questions

> **Note**
>
> This information applies to users in the GovCloud (US-West) and GovCloud (US-East) regions.

The certificate authority (CA) certificate for Amazon DocumentDB (with MongoDB compatibility) clusters will update on **May 18, 2022**. If you are using Amazon DocumentDB clusters with Transport Layer Security (TLS) enabled (the default setting) and you have not rotated your client application and server certificates, the following steps are required to mitigate connectivity issues between your application and your Amazon DocumentDB clusters.

- Step 1: Download the new CA certificate and update your application
- Step 2: Update the server certificate

The CA and server certificates were updated as part of standard maintenance and security best practices for Amazon DocumentDB. The previous CA certificate will expire on May 18, 2022. Client applications must add the new CA certificates to their trust stores, and existing Amazon DocumentDB instances must be updated to use the new CA certificates before this expiration date.

Updating your application and Amazon DocumentDB cluster

Follow the steps in this section to update your application's CA certificate bundle (Step 1) and your cluster's server certificates (Step 2). Before you apply the changes to your production environments, we strongly recommend testing these steps in a development or staging environment.

> **Note**
>
> You must complete Steps 1 and 2 in each AWS Region in which you have Amazon DocumentDB clusters.

Step 1: Download the new CA certificate and update your application

Download the new CA certificate and update your application to use the new CA certificate to create TLS connections to Amazon DocumentDB in your specific region:

- For GovCloud (US-West), download the new CA certificate bundle from https://truststore.pki.us-gov-west-1.rds.amazonaws.com/us-gov-west-1/us-gov-west-1-bundle.pem. This operation downloads a file named `us-gov-west-1-bundle.pem`.
- For GovCloud (US-East), download the new CA certificate bundle from https://truststore.pki.us-gov-west-1.rds.amazonaws.com/us-gov-east-1/us-gov-east-1-bundle.pem. This operation downloads a file named `us-gov-east-1-bundle.pem`.

> **Note**
>
> If you are accessing the keystore that contains both the old CA certificate (`rds-ca-2017-root.pem`) and the new CA certificates (`rds-ca-rsa2048-g1.pem`, `rds-ca-rsa4096-g1.pem`, or `rds-ca-ecc384-g1.pem`), verify that the keystore selects your certificate of choice. For details on each certificate, see Step 2 below.

```
wget https://truststore.pki.us-gov-west-1.rds.amazonaws.com/us-gov-west-1/us-gov-west-1-bundle.pem
```

```
wget https://truststore.pki.us-gov-west-1.rds.amazonaws.com/us-gov-east-1/us-gov-east-1-bundle.pem
```

Next, update your applications to use the new certificate bundle. The new CA bundle contains both the old CA certificate and the new CA certificate (`rds-ca-rsa2048-g1.pem`, `rds-ca-rsa4096-g1.pem`, or `rds-ca-ecc384-g1.pem`). By having both CA certificates in the new CA bundle, you can update your application and cluster in two steps.

Any downloads of the CA certificate bundle after December 21, 2021 should use the new CA certificate bundle. To verify that your application is using the latest CA certificate bundle, see How can I be sure that I'm using the newest CA bundle? If you're already using the latest CA certificate bundle in your application, you can skip to Step 2.

For examples of using a CA bundle with your application, see Encrypting data in transit and Connecting with TLS enabled.

> **Note**
>
> Currently, the MongoDB Go Driver 1.2.1 only accepts one CA server certificate in `sslcertificateauthorityfile`. Please see Connecting with TLS enabled for connecting to Amazon DocumentDB using Go when TLS is enabled.

Step 2: Update the server certificate

After the application has been updated to use the new CA bundle, the next step is to update the server certificate by modifying each instance in an Amazon DocumentDB cluster. To modify instances to use the new server certificate, see the following instructions.

Amazon DocumentDB provides the following CAs to sign the DB server certificate for a DB instance:

- **rds-ca-ecc384-g1**—Uses a certificate authority with ECC 384 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation. This is only supported on Amazon DocumentDB 4.0 and 5.0.
- **rds-ca-rsa2048-g1**—Uses a certificate authority with RSA 2048 private key algorithm and SHA256 signing algorithm in most AWS regions. This CA supports automatic server certificate rotation.
- **rds-ca-rsa4096-g1**—Uses a certificate authority with RSA 4096 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation.

> **Note**
>
> If you are using the AWS CLI, you can see the validities of the certificate authorities listed above by using describe-certificates.

> **Note**
>
> Amazon DocumentDB 4.0 and 5.0 instances do **not** require a reboot.

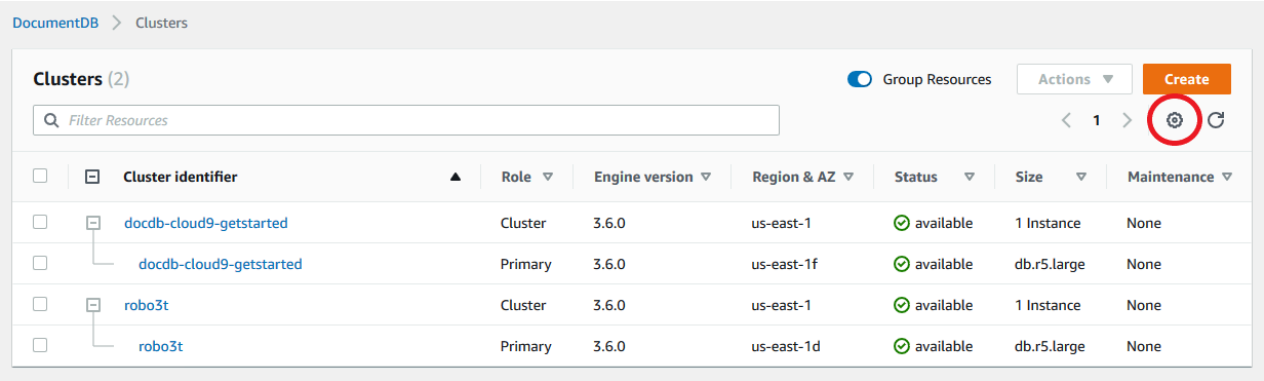
> Updating your Amazon DocumentDB 3.6 instances requires a reboot, which might cause service disruption. Before updating the server certificate, ensure that you have completed Step 1.

Using the AWS Management Console

Complete the following steps to identify and rotate the old server certificate for your existing Amazon DocumentDB instances using the AWS Management Console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your clusters reside.
3. In the navigation pane on the left side of the console, choose **Clusters**.
4. You may need to identify which instances are still on the old server certificate (`rds-ca-2017`). You can do this in the **Certificate authority** column which is hidden by default. To show the **Certificate authority column**, do the following:

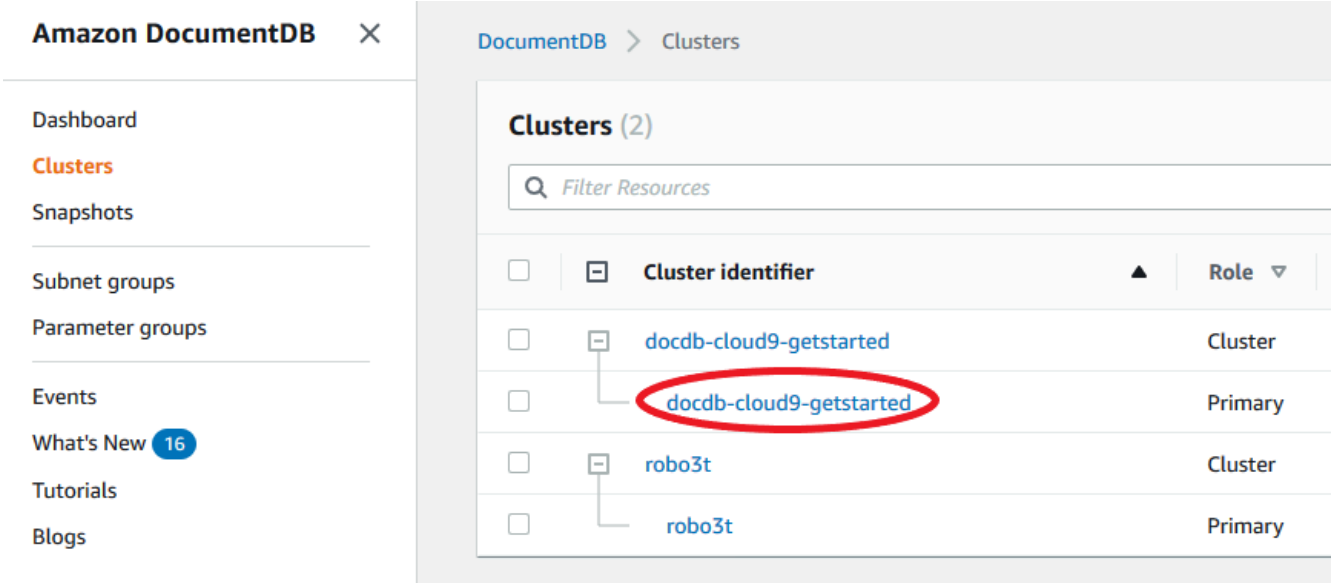
   a. Choose the **Settings** icon.



   b. Under the list of visible columns, choose the **Certificate authority** column.

   c. Choose **Confirm** to save your changes.

5. Now back in the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



6. Check the box to the left of the instance you are interested in.
7. Choose **Actions** and then choose **Modify**.
8. Under **Certificate authority**, select the new server certificate (`rds-ca-rsa2048-g1`, `rds-ca-rsa4096-g1`, or `rds-ca-ecc384-g1`) for this instance.
9. You can see a summary of the changes on the next page. Note that there is an extra alert to remind you to ensure that your application is using the latest certificate CA bundle before modifying the instance to avoid causing an interruption in connectivity.
10. You can choose to apply the modification during your next maintenance window or apply immediately. If your intention is to modify the server certificate immediately, use the **Apply Immediately** option.
11. Choose **Modify instance** to complete the update.

Using the AWS CLI

Complete the following steps to identify and rotate the old server certificate for your existing Amazon DocumentDB instances using the AWS CLI.

1. To modify the instances immediately, execute the following command for each instance in the cluster. Use one of the following certificates: `rds-ca-rsa2048-g1`,`rds-ca-rsa4096-g1`, or `rds-ca-ecc384-g1`.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa4096-g1 --apply-immediately
```

2. To modify the instances in your clusters to use the new CA certificate during your cluster's next maintenance window, execute the following command for each instance in the cluster. Use one of the following certificates: `rds-ca-rsa2048-g1`, `rds-ca-rsa4096-g1`, or `rds-ca-ecc384-g1`.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa4096-g1 --no-apply-immediately
```

Troubleshooting

If you are having issues connecting to your cluster as part of the certificate rotation, we suggest the following:

- **Reboot your instances.** Rotating the new certificate requires that you reboot each of your instances. If you applied the new certificate to one or more instances but did not reboot them, reboot your instances to apply the new certificate. For more information, see Rebooting an Amazon DocumentDB instance.
- **Verify that your clients are using the latest certificate bundle.** See How can I be sure that I'm using the newest CA bundle?.
- **Verify that your instances are using the latest certificate.** See How do I know which of my Amazon DocumentDB instances are using the old/new server certificate?.
- **Verify that the latest certificate CA is being utilized by your application.** Some drivers, like Java and Go, require extra code to import multiple certificates from a certificate bundle to the trust store. For more information on connecting to Amazon DocumentDB with TLS, see Connecting programmatically to Amazon DocumentDB.
- **Contact support.** If you have questions or issues, contact Support.

Frequently Asked Questions

The following are answers to some common questions about TLS certificates.

What if I have questions or issues?

If you have questions or issues, contact Support.

How do I know whether I'm using TLS to connect to my Amazon DocumentDB cluster?

You can determine whether your cluster is using TLS by examining the `tls` parameter for your cluster's cluster parameter group. If the `tls` parameter is set to `enabled`, you are using the TLS certificate to connect to your cluster. For more information, see Managing Amazon DocumentDB cluster parameter groups.

Why are you updating the CA and server certificates?

The Amazon DocumentDB CA and server certificates were updated as part of standard maintenance and security best practices for Amazon DocumentDB. The current CA and server certificates will expire on Wednesday, May 18, 2022.

What happens if I don't take any action by the expiration date?

If you are using TLS to connect to your Amazon DocumentDB cluster and you do not make the change by May 18, 2022, your applications that connect via TLS will no longer be able to communicate with the Amazon DocumentDB cluster.

Amazon DocumentDB will not rotate your database certificates automatically before expiration. You must update your applications and clusters to use the new CA certificates before or after the expiration date.

How do I know which of my Amazon DocumentDB instances are using the old/new server certificate?

To identify the Amazon DocumentDB instances that still use the old server certificate, you can use either the Amazon DocumentDB AWS Management Console or the AWS CLI.

Using the AWS Management Console

To identify the instances in your clusters that are using the older certificate

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your instances reside.
3. In the navigation pane on the left side of the console, choose **Instances**.

4. The **Certificate authority** column (hidden by default) shows which instances are still on the old server certificate (`rds-ca-2017`) and the new server certificate (`rds-ca-rsa2048-g1`, `rds-ca-rsa4096-g1`, or `rds-ca-ecc384-g1`). To show the **Certificate authority column**, do the following:

   a. Choose the **Settings** icon.
   b. Under the list of visible columns, choose the **Certificate authority** column.
   c. Choose **Confirm** to save your changes.

Using the AWS CLI

To identify the instances in your clusters that are using the older server certificate, use the `describe-db-clusters` command with the following .

```
aws docdb describe-db-instances \
    --filters Name=engine,Values=docdb \
    --query 'DBInstances[*].
{CertificateVersion:CACertificateIdentifier,InstanceID:DBInstanceIdentifier}'
```

How do I modify individual instances in my Amazon DocumentDB cluster to update the server certificate?

We recommend that you update server certificates for all instances in a given cluster at the same time. To modify the instances in your cluster, you can use either the console or the AWS CLI.

> **Note**
>
> Updating your instances requires a reboot, which might cause service disruption. Before updating the server certificate, ensure that you have completed Step 1.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your clusters reside.

3. In the navigation pane on the left side of the console, choose **Instances**.
4. The **Certificate authority** column (hidden by default) shows which instances are still on the old server certificate (`rds-ca-2017`). To show the **Certificate authority column**, do the following:
   a. Choose the **Settings** icon.
   b. Under the list of visible columns, choose the **Certificate authority** column.
   c. Choose **Confirm** to save your changes.
5. Select an instance to modify.
6. Choose **Actions** and then choose **Modify**.
7. Under **Certificate authority**, select one of the new server certificate (`rds-ca-rsa2048-g1`,`rds-ca-rsa4096-g1`, or `rds-ca-ecc384-g1`) for this instance.
8. You can see a summary of the changes on the next page. Note that there is an extra alert to remind you to ensure that your application is using the latest certificate CA bundle before modifying the instance to avoid causing an interruption in connectivity.
9. You can choose to apply the modification during your next maintenance window or apply immediately.
10. Choose **Modify instance** to complete the update.

Using the AWS CLI

Complete the following steps to identify and rotate the old server certificate for your existing Amazon DocumentDB instances using the AWS CLI.

1. To modify the instances immediately, execute the following command for each instance in the cluster.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa4096-g1 --apply-immediately
```

2. To modify the instances in your clusters to use the new CA certificate during your cluster's next maintenance window, execute the following command for each instance in the cluster.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa4096-g1 --no-apply-immediately
```

What happens if I add a new instance to an existing cluster?

All new instances that are created use the old server certificate and require TLS connections using the old CA certificate. Any new Amazon DocumentDB instances created after March 21, 2022 will default to using the new certificates.

What happens if there is an instance replacement or failover on my cluster?

If there is an instance replacement in your cluster, the new instance that is created continues to use the same server certificate that the instance was previously using. We recommend that you update server certificates for all instances at the same time. If a failover occurs in the cluster, the server certificate on the new primary is used.

If I'm not using TLS to connect to my cluster, do I still need to update each of my instances?

If you are not using TLS to connect to your Amazon DocumentDB clusters, no action is needed.

If I'm not using TLS to connect to my cluster but I plan to in the future, what should I do?

If you created a cluster before March 21, 2022, follow Step 1 and Step 2 in the previous section to ensure that your application is using the updated CA bundle, and that each Amazon DocumentDB instance is using the latest server certificate. If you create a cluster after March 21, 2022, your cluster will already have the latest server certificate. To verify that your application is using the latest CA bundle, see If I'm not using TLS to connect to my cluster, do I still need to update each of my instances?

Can the deadline be extended beyond May 18, 2022?

If your applications are connecting via TLS, the deadline cannot be extended beyond May 18, 2022.

How can I be sure that I'm using the newest CA bundle?

For compatibility reasons, both old and new CA bundle files are named `us-gov-west-1-bundle.pem`. You can also use tools like `openssl` or `keytool` to inspect the CA bundle.

Why do I see "RDS" in the name of the CA bundle?

For certain management features, such as certificate management, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).

When will the new certificate expire?

The new server certificate will expire (generally) as follows:

- **rds-ca-rsa2048-g1**—Expires 2061
- **rds-ca-rsa4096-g1**—Expires 2121
- **rds-ca-ecc384-g1**—Expires 2121

What kind of errors will I see if I don't take action before the certificate expires?

Error messages will vary depending on your driver. In general, you'll see certificate validation errors that contain the string "certificate has expired".

If I applied the new server certificate, can I revert it back to the old server certificate?

If you need to revert an instance to the old server certificate, we recommend that you do so for all instances in the cluster. You can revert the server certificate for each instance in a cluster by using the AWS Management Console or the AWS CLI.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the list of Regions in the upper-right corner of the screen, choose the AWS Region in which your clusters reside.
3. In the navigation pane on the left side of the console, choose **Instances**.
4. Select an instance to modify. Choose **Actions**, and then choose **Modify**.
5. Under **Certificate authority**, you can select the old server certificate (`rds-ca-2017`).
6. Choose **Continue** to view a summary of your modifications.
7. In this resulting page, you can choose to schedule your modifications to be applied in the next maintenance window or apply your modifications immediately. Make your selection, and choose **Modify instance**.

> **Note**
>
> If you choose to apply your modifications immediately, any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing this option can cause unexpected downtime.

Using the AWS CLI

```
aws docdb modify-db-instance --db-instance-identifier <db_instance_name> ca-
certificate-identifier rds-ca-2017 <--apply-immediately | --no-apply-immediately>
```

If you choose `--no-apply-immediately`, the changes will be applied during the cluster's next maintenance window.

If I restore from a snapshot or a point in time restore, will it have the new server certificate ?

If you restore a snapshot or perform a point-in-time restore after March 21, 2022, the new cluster that is created will use the new CA certificate.

What if I'm having issues connecting directly to my Amazon DocumentDB cluster from Mac OS X Catalina?

Mac OS X Catalina has updated the requirements for trusted certificates. Trusted certificates must now be valid for 825 days or fewer (see https://support.apple.com/en-us/HT210176). Amazon DocumentDB instance certificates are valid for over four years, longer than the Mac OS X maximum. In order to connect directly to an Amazon DocumentDB cluster from a computer running Mac OS X Catalina, you must allow invalid certificates when creating the TLS connection. In this case, invalid certificates mean that the validity period is longer than 825 days. You should understand the risks before allowing invalid certificates when connecting to your Amazon DocumentDB cluster.

To connect to an Amazon DocumentDB cluster from OS X Catalina using the AWS CLI, use the `tlsAllowInvalidCertificates` parameter.

```
mongo --tls --host <hostname> --username <username> --password <password> --port 27017
 --tlsAllowInvalidCertificates
```

Compliance validation in Amazon DocumentDB

The security and compliance of Amazon DocumentDB is assessed by third-party auditors as part of multiple AWS compliance programs, including the following:

- System and Organization Controls (SOC) 1, 2, and 3. For more information, see SOC.
- Federal Risk and Authorization Management Program (FedRAMP). For more information, see AWS Services in Scope by Compliance Program.
- Payment Card Industry Data Security Standard (PCI DSS). For more information, see PCI DSS.
- ISO 9001, 27001, 27017, and 27018. For more information, see ISO Certified.
- Health Insurance Portability and Accountability Act Business Associate Agreement (HIPAA BAA). For more information, see HIPAA Compliance

AWS provides a frequently updated list of AWS services in scope of specific compliance programs at AWS Services in Scope by Compliance Program.

Third-party audit reports are available for you to download using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

For more information about AWS compliance programs, see AWS Compliance Programs.

Your compliance responsibility when using Amazon DocumentDB is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. If your use of Amazon DocumentDB is subject to compliance with standards like HIPAA or PCI, AWS provides resources to help:

- AWS Compliance Resources – A collection of workbooks and guides that might apply to your industry and location.
- Security and Compliance Quick Start Guides – Deployment guides that discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- AWS Config – A service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – A comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- Architecting for HIPAA Security and Compliance Whitepaper – A whitepaper that describes how companies can use AWS to create HIPAA-compliant applications.

Resilience in Amazon DocumentDB

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

An Amazon DocumentDB cluster can only be created in an Amazon VPC that has at least two subnets in at least two Availability Zones. By distributing your cluster instances across at least two Availability Zones, Amazon DocumentDB helps ensure that there are instances available in your cluster in the unlikely event of an Availability Zone failure. The cluster volume for your Amazon DocumentDB cluster always spans three Availability Zones to provide durable storage with less possibility of data loss.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon DocumentDB offers several features to help support your data resiliency and backup needs.

Fault-tolerant and self-healing storage

Each 10 GB portion of your storage volume is replicated six ways, across three Availability Zones. Amazon DocumentDB uses fault-tolerant storage that transparently handles the loss of up to two copies of data without affecting database write availability, and up to three copies without affecting read availability. Amazon DocumentDB storage is also self-healing; data blocks and disks are continuously scanned for errors and replaced automatically.

Manual backups and restore

Amazon DocumentDB provides the capability to create full backups of your cluster for long-term retention and recovery. For more information, see Backing up and restoring in Amazon DocumentDB.

Point-in-time recovery

Point-in-time recovery helps protect your Amazon DocumentDB clusters from accidental write or delete operations. With point-in-time recovery, you don't have to worry about creating, maintaining, or scheduling on-demand backups. For more information, see Restoring to a point in time.

Infrastructure security in Amazon DocumentDB

As a managed service, Amazon DocumentDB is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon DocumentDB through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location. You can use Amazon DocumentDB policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon DocumentDB resource from only the specific VPC within the AWS network.

> **Note**
>
> Amazon DocumentDB does not support resource-based access policies.

Amazon DocumentDB API and interface VPC endpoints (AWS PrivateLink)

> **Note**
>
> Amazon DocumentDB elastic clusters do not support AWS PrivateLink VPC endpoints.

You can establish a private connection between your VPC and Amazon DocumentDB API endpoints by creating an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink.

While Amazon DocumentDB instance-based clusters do not require an interface VPC endpoint connection, AWS PrivateLink enables you to privately access Amazon DocumentDB API operations without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Amazon DocumentDB instances in your VPC don't need public IP addresses to communicate with Amazon DocumentDB API endpoints to launch, modify, or terminate database instances and database clusters. Your Amazon DocumentDB instances also don't need public IP addresses to use any of the available Amazon DocumentDB API operations. Traffic between your VPC and Amazon DocumentDB doesn't leave the Amazon network.

Each interface endpoint is represented by one or more elastic network interfaces in your subnets. For more information, see Elastic network interfaces in the *Amazon EC2 User Guide.*

For more information about VPC endpoints, see Access an AWS service using an interface VPC endpoint in the *Amazon Virtual Private Cloud (AWS PrivateLink) User Guide*. For more information about Amazon DocumentDB operations, see the Amazon DocumentDB cluster, instance, and resource management API reference.

Topics

- Considerations for VPC endpoints
- Region availability
- Creating an interface VPC endpoint for Amazon DocumentDB API
- Creating a VPC endpoint policy for Amazon DocumentDB API

Considerations for VPC endpoints

Before you set up an interface VPC endpoint for Amazon DocumentDB API endpoints, ensure that you review the interface endpoint prerequisites in the *Amazon Virtual Private Cloud (AWS PrivateLink) User Guide*.

All Amazon DocumentDB API operations relevant to managing Amazon DocumentDB resources are available from your VPC using AWS PrivateLink.

VPC endpoint policies are supported for Amazon DocumentDB API endpoints. By default, full access to Amazon DocumentDB API operations is allowed through the endpoint. For more

information, see Control access to VPC endpoints using endpoint policies in the *Amazon Virtual Private Cloud (AWS PrivateLink) User Guide*.

Region availability

Amazon DocumentDB API currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Spain)
- Europe (Milan)
- Middle East (UAE)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Creating an interface VPC endpoint for Amazon DocumentDB API

You can create a VPC endpoint for the Amazon DocumentDB API using either the Amazon VPC Console or the AWS Command Line Interface (AWS CLI). For more information, see Access an AWS service using an interface VPC endpoint in the *Amazon Virtual Private Cloud (AWS PrivateLink) User Guide*.

Create a VPC endpoint for the Amazon DocumentDB API using the service name `com.amazonaws.region.rds`.

Excluding AWS Regions in China, if you enable private DNS for the endpoint, you can make API requests to Amazon DocumentDB with the VPC endpoint using its default DNS name for the AWS Regions, for example rds.us-east-1.amazonaws.com. For the China (Beijing) and China (Ningxia) AWS Regions, you can make API requests with the VPC endpoint using rds-api.cn-north-1.amazonaws.com.cn and rds-api.cn-northwest-1.amazonaws.com.cn, respectively.

For more information, see Access an AWS service using an interface VPC endpoint in the *Amazon Virtual Private Cloud (AWS PrivateLink) User Guide*.

Creating a VPC endpoint policy for Amazon DocumentDB API

You can attach an endpoint policy to your VPC endpoint that controls access to the Amazon DocumentDB API. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Control access to VPC endpoints using endpoint policies in the *Amazon Virtual Private Cloud (AWS PrivateLink) User Guide*.

Example: VPC endpoint policy for Amazon DocumentDB API actions

The following is an example of an endpoint policy for Amazon DocumentDB API. When attached to an endpoint, this policy grants access to the listed Amazon DocumentDB API actions for all principals on all resources.

```
{
"Statement":[
```

```
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "docdb:CreateDBInstance",
            "docdb:ModifyDBInstance",
            "docdb:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

Example: VPC endpoint policy that denies all access from a specified AWS account

The following VPC endpoint policy denies AWS account 123456789012 all access to resources using the endpoint. The policy allows all actions from other accounts.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Security best practices for Amazon DocumentDB

For security best practices, you must use AWS Identity and Access Management (IAM) accounts to control access to Amazon DocumentDB API operations, especially operations that create, modify, or delete Amazon DocumentDB resources. Such resources include clusters, security groups, and parameter groups. You must also use IAM to control actions that perform common administrative

actions such as backing up restoring clusters. When creating IAM roles, employ the principle of least privilege.

- Enforce least privilege with role-based access control.
- Assign an individual IAM account to each person who manages Amazon DocumentDB resources. Do not use the AWS account root user to manage Amazon DocumentDB resources. Create an IAM user for everyone, including yourself.
- Grant each user the minimum set of permissions that are required to perform their duties.
- Use IAM groups to effectively manage permissions for multiple users. For more information about IAM, see the IAM User Guide. For information about IAM best practices, see IAM Best Practices.
- Regularly rotate your IAM credentials.
- Configure AWS Secrets Manager to automatically rotate the secrets for Amazon DocumentDB. For more information, see Rotating Your AWS Secrets Manager Secrets and Rotating Secrets for Amazon DocumentDB in the *AWS Secrets Manager User Guide*.
- Use Transport Layer Security (TLS) and encryption at rest to encrypt your data.

Auditing Amazon DocumentDB events

With Amazon DocumentDB (with MongoDB compatibility), you can audit events that were performed in your cluster. Examples of logged events include successful and failed authentication attempts, dropping a collection in a database, or creating an index. By default, auditing is disabled on Amazon DocumentDB and requires that you opt in to use this feature.

When auditing is enabled, Amazon DocumentDB records Data Definition Language (DDL), Data Manipulation Language (DML), authentication, authorization, and user management events to Amazon CloudWatch Logs. When auditing is enabled, Amazon DocumentDB exports your cluster's auditing records (JSON documents) to Amazon CloudWatch Logs. You can use Amazon CloudWatch Logs to analyze, monitor, and archive your Amazon DocumentDB auditing events.

Although Amazon DocumentDB does not charge an additional cost to enable auditing, you are charged standard rates for the usage of CloudWatch Logs. For information about CloudWatch Logs pricing, see Amazon CloudWatch pricing.

The Amazon DocumentDB auditing feature is distinctly different from the service resource usage that is monitored with AWS CloudTrail. CloudTrail records operations that are performed with the

AWS Command Line Interface (AWS CLI) or AWS Management Console on resources like clusters, instances, parameter groups, and snapshots. Auditing of resources with CloudTrail is on by default and cannot be disabled. The Amazon DocumentDB auditing feature is an opt-in feature. It records operations that take place within your cluster on objects, such as databases, collections, indexes, and users.

Topics

- Supported events
- Enabling auditing
- Disabling auditing
- Accessing your audit events
- Filtering DML audit events

Supported events

Amazon DocumentDB auditing supports the following event categories:

- **Data Definition Language (DDL)** - includes database management operations, connections, user management, and authorization.
- **Data Manipulation Language read events (DML reads)** - includes `find()` and the various aggregation operators, arithmetic operators, boolean operators, and other read query operators.
- **Data Manipulation Language write events (DML writes)** - includes `insert(), update(), delete()`, and `bulkWrite()` operators

The event types are as follows.

Event Type	Category	Description
`authCheck`	Authorization	Result code 0: Success
		Result code 13: Unauthorized attempts to perform an operation.

Event Type	Category	Description
`authenticate`	Connection	Successful or failed authentication attempts on a new connection.
`auditConfigure`	DDL	Audit filter configura tion.
`createDatabase`	DDL	Creation of a new database.
`createCollection`	DDL	Creation of a new collection within a database.
`createIndex`	DDL	Creation of a new index within a collection.
`dropCollection`	DDL	Dropping of a collection within a database.
`dropDatabase`	DDL	Dropping of a database.
`dropIndex`	DDL	Dropping of an index within a collection.
`modifyChangeStreams`	DDL	Change stream was created.
`renameCollection`	DDL	Renaming of a collection within a database.
`createRole`	Role Management	Creating a role.

Event Type	Category	Description
`dropAllRolesFromDatabase`	Role Management	Dropping all roles within a database.
`dropRole`	Role Management	Dropping a role.
`grantPrivilegesToRole`	Role Management	Granting privileges to a role.
`grantRolesToRole`	Role Management	Granting roles to a user-defined role.
`revokePrivilegesFromRole`	Role Management	Revoking privileges from a role.
`revokeRolesFromRole`	Role Management	Revoking roles from a user-defined role.
`updateRole`	Role Management	Updating a role.
`createUser`	User Management	Creation of a new user.
`dropAllUsersFromDatabase`	User Management	Dropping of all users within a database.
`dropUser`	User Management	Dropping of an existing user.
`grantRolesToUser`	User Management	Granting roles to a user.
`revokeRolesFromUser`	User Management	Revoking roles from a user.
`updateUser`	UserManagement	Updating of an existing user.

Event Type	Category	Description
`insert`	DML write	Inserts a document or documents into a collection.
`delete`	DML write	Deletes a document or documents from a collection.
`update`	DML write	Modifies an existing document or documents in a collection.
`bulkWrite`	DML write	Performs multiple write operations with controls for order of execution.
`setAuditConfig`	DML write	Set a new filter for DML auditing.
`count`	DML read	Returns the count of documents that would match a find() query for the collection or view.
`countDocuments`	DML read	Returns the count of documents that match the query for a collection or view.

Event Type	Category	Description
`find`	DML read	Selects documents in a collection or view and returns a cursor to the selected documents.
`getAuditConfig`	DML read	Retrieve the current filter for DML auditing.
`findAndModify`	DML read and DML write	Modifies and returns a single document.
`findOneAndDelete`	DML read and DML write	Deletes a single document based on the filter and sort criteria, returning the deleted document.
`findOneAndReplace`	DML read and DML write	Replaces a single document based on the specified filter.
`findOneAndUpdate`	DML read and DML write	Updates a single document based on the filter and sort criteria.
`aggregate`	DML read and DML write	Supports APIs in the aggregation pipeline.

Event Type	Category	Description
`distinct`	DML read	Finds the distinct values for a specified field across a single collection or view and returns the results in an array.

> **Note**
>
> Values in the DML event document parameter field have a 1KB size limit. Amazon DocumentDB truncates the value if it exceeds 1KB.

> **Note**
>
> TTL delete events are not audited at this time.

Enabling auditing

Enabling auditing on a cluster is a two-step process. Ensure that both steps are completed, or audit logs will not be sent to CloudWatch Logs.

Step 1. Enable the audit_logs cluster parameter

To enable auditing, you need to modify the `audit_logs` parameter in the parameter group. `audit_logs` is a comma-delimited list of events to log. Events must be specified in lowercase and there should be no white space between the list elements.

You can set the following values for the parameter group:

Value	Description	
`ddl`	Setting this will enable auditing	

Value	Description	
	for DDL events such as createDatabase, dropDatabase, createCollection, dropCollection, createIndex, dropIndex, authCheck, authenticate, createUser, dropUser, grantRolesToUser, revokeRolesFromUser, updateUser, and dropAllUsersFromDatabase	
`dml_read`	Setting this will enable auditing for DML read events such as find, sort count, distinct, group, projecta, unwind, geoNear, geoIntersects, geoWithin and other MongoDB read query operators.	
`dml_write`	Setting this will enable auditing for DML write events such as insert(), update(), delete(), and bulkWrite()	

Value	Description
`all`	Setting this will enable auditing for your database events, such as read queries, write queries, database actions and administrator actions.
none	Setting this will disable auditing
`enabled` (legacy)	This is a legacy parameter setting that is equivalent to 'ddl'. Setting this will enable auditing for DDL events such as createDatabase, dropDatabase, createCollection, dropCollection, createIndex, dropIndex, authCheck, authenticate, createUser, dropUser, grantRolesToUser, revokeRolesFromUser, updateUser, and dropAllUsersFromDatabase. We do not recommend using this setting because it is a legacy setting.

Value	Description
`disabled` (legacy)	This is a legacy parameter setting that is equivalent to 'none'. We do not recommend using this setting because it is a legacy setting.

> **Note**
>
> The default value for the audit_logs cluster parameter is none (legacy "`disabled`").

You can also use the above mentioned values in combinations.

Value	Description
`ddl, dml_read`	Setting this will enable auditing for DDL events and DML read events.
`ddl, dml_write`	Setting this will enable auditing for DDL events and DML write
`dml_read, dml_write`	Setting this will enable auditing for all DML events

> **Note**
>
> You cannot modify a default parameter group.

For more information, see the following:

- Creating Amazon DocumentDB cluster parameter groups

  After creating a custom parameter group, modify it by changing the `audit_logs` parameter value to `all`.
- Modifying Amazon DocumentDB cluster parameter groups

Step 2. Enable Amazon CloudWatch Logs export

When the value of the `audit_logs` cluster parameter is `enabled`, `ddl`, `dml_read`, or `dml_write`, you must also enable Amazon DocumentDB to export logs to Amazon CloudWatch. If you omit either of these steps, audit logs will not be sent to CloudWatch.

When creating a cluster, performing a point-in-time-restore, or restoring a snapshot, you can enable CloudWatch Logs by following these steps.

Using the AWS Management Console

To enable Amazon DocumentDB exporting logs to CloudWatch using the console, see the following topics:

- **When creating a cluster** — In Creating a cluster and primary instance using the AWS Management Console, see **Create a Cluster: Additional Configurations** (step 5, **Log exports**)
- **When modifying an existing cluster** — Modifying an Amazon DocumentDB cluster
- **When performing a cluster snapshot restore** — Restoring from a cluster snapshot
- **When performing a point-in-time restore** — Restoring to a point in time

Using the AWS CLI

To enable audit logs when creating a new cluster

The following code creates the cluster `sample-cluster` and enables CloudWatch audit logs.

Example

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster \
    --db-cluster-identifier sample-cluster \
    --port 27017 \
    --engine docdb \
    --master-username master-username \
    --master-user-password password \
    --db-subnet-group-name default \
    --enable-cloudwatch-logs-exports audit
```

For Windows:

```
aws docdb create-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --port 27017 ^
    --engine docdb ^
    --master-username master-username ^
    --master-user-password password ^
    --db-subnet-group-name default ^
    --enable-cloudwatch-logs-exports audit
```

To enable audit logs when modifying an existing cluster

The following code modifies the cluster `sample-cluster` and enables CloudWatch audit logs.

Example

For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

For Windows:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

Output from these operations looks something like the following (JSON format).

```
{
    "DBCluster": {
        "HostedZoneId": "ZNKXH85TT8WVW",
        "StorageEncrypted": false,
        "DBClusterParameterGroup": "default.docdb4.0",
        "MasterUsername": "<user-name>",
        "BackupRetentionPeriod": 1,
        "Port": 27017,
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "DBClusterArn": "arn:aws:rds:us-east-1:900083794985:cluster:sample-cluster",
        "Status": "creating",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "MultiAZ": false,
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1f"
        ],
        "DBSubnetGroup": "default",
        "DBClusterMembers": [],
        "ReaderEndpoint": "sample-cluster.cluster-ro-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "EnabledCloudwatchLogsExports": [
            "audit"
        ],
        "PreferredMaintenanceWindow": "wed:03:08-wed:03:38",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2019-02-13T16:35:04.756Z",
        "DbClusterResourceId": "cluster-YOS52CUXGDTNKDQ7DH72I4LED4",
        "Endpoint": "sample-cluster.cluster-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "PreferredBackupWindow": "07:16-07:46",
        "DBClusterIdentifier": "sample-cluster"
    }
}
```

Disabling auditing

You can disable auditing by disabling CloudWatch Logs export and disabling the `audit_logs` parameter.

Disabling CloudWatch Logs export

You can disable exporting audit logs using either the AWS Management Console or the AWS CLI.

Using the AWS Management Console

The following procedure uses the AWS Management Console to disable Amazon DocumentDB exporting logs to CloudWatch.

To disable audit logs

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**. Then choose the button to the left of the name of the cluster for which you want to disable exporting logs.
3. Choose **Actions**, and then choose **Modify**.
4. Scroll down to the **Log exports** section and choose **Disabled**.
5. Choose **Continue**.
6. Review your changes, and then choose when you want this change applied to your cluster.
   - **Apply during the next scheduled maintenance window**
   - **Apply immediately**
7. Choose **Modify cluster**.

Using the AWS CLI

The following code modifies the cluster `sample-cluster` and disables CloudWatch audit logs.

Example

For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster \
```

```
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

For Windows:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBCluster": {
        "DBClusterParameterGroup": "default.docdb4.0",
        "HostedZoneId": "ZNKXH85TT8WVW",
        "MasterUsername": "<user-name>",
        "Status": "available",
        "Engine": "docdb",
        "Port": 27017,
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1f"
        ],
        "EarliestRestorableTime": "2019-02-13T16:35:50.387Z",
        "DBSubnetGroup": "default",
        "LatestRestorableTime": "2019-02-13T16:35:50.387Z",
        "DBClusterArn": "arn:aws:rds:us-east-1:900083794985:cluster:sample-
cluster2",
        "Endpoint": "sample-cluster2.cluster-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster2.cluster-ro-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "BackupRetentionPeriod": 1,
        "EngineVersion": "4.0.0",
        "MultiAZ": false,
        "ClusterCreateTime": "2019-02-13T16:35:04.756Z",
        "DBClusterIdentifier": "sample-cluster2",
        "AssociatedRoles": [],
        "PreferredBackupWindow": "07:16-07:46",
        "DbClusterResourceId": "cluster-YOS52CUXGDTNKDQ7DH72I4LED4",
        "StorageEncrypted": false,
        "PreferredMaintenanceWindow": "wed:03:08-wed:03:38",
```

```
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ]
    }
}
```

Disabling the audit_logs parameter

To disable the `audit_logs` parameter for your cluster, you can modify the cluster so that it uses a parameter group where the `audit_logs` parameter value is `disabled`. Or you can modify the `audit_logs` parameter value in the cluster's parameter group so that it is `disabled`.

For more information, see the following topics:

- Modifying an Amazon DocumentDB cluster
- Modifying Amazon DocumentDB cluster parameter groups

Accessing your audit events

Use following steps to access your audit events on Amazon CloudWatch.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Make sure that you are in the same Region as your Amazon DocumentDB cluster.
3. In the navigation pane, choose **Logs**.
4. To find the audit logs for your cluster, from the list locate and choose **/aws/docdb/*yourClusterName*/audit**.

   The auditing events for each of your instances are available under each of the respective instance names.

Filtering DML audit events

Getting started with DML audit filtering

DML audit events can be filtered before they are written to Amazon CloudWatch. To utilize this feature, audit log and DML logging must be enabled. Amazon DocumentDB supports filtering on `atype`, `command`, `user`, `namespace`, and `auditAuthorizationSuccess`.

> **Note**
>
> DDL events are not filtered.

You can enable audit filtering at anytime by specifying the audit filter using the `setAuditConfig`, `filter`, and `auditAuthorizationSuccess` parameters in the `db.adminCommand( { command } )` operation:

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter:
         {
            //filter conditions
         },
      auditAuthorizationSuccess: true | false
   }
)
```

You can also retrieve the audit filter settings by running the following command:

```
db.admin.runCommand( { getAuditConfig: 1})
```

Security requirements

Only database users/roles with privileged action `auditConfigure` can execute the above commands against `admindb` when setting or listing DML audit filters. You can either use one of the built-in roles from [`clusterAdmin`, `hostManager`, `root`] or create custom roles that have `auditConfigure` privileges. The following is an example of using existing roles with the `auditConfigure` privilege and an example with custom roles.

User with built-in role:

```
use admin
db.createUser(
  {
    user: "myClusterAdmin",
    pwd: "password123",
    roles: [ { role: "clusterAdmin", db: "admin" } ]
  }
)
```

User with custom roles:

```
use admin
db.createRole(
   {
     role: "myRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "auditConfigure" ] }
     ],
     roles: []
   }
)
db.createUser(
  {
    user: "myUser",
    pwd: "myPassword",
    roles: [ { role: "myRole", db: "admin" } ]
  }
)
```

Filtering use cases

Example: filtering events by commands

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
        "$and": [
         {
            "param.command":
               {
```

```
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ]
      },
      auditAuthorizationSuccess: true
   }
)
```

Example: filtering events by user name

In this example, only user "myUser" will be logged:

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.user":
               {
                  $in: [ "myUser" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

Example: filtering by `atype`

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {atype: "authCheck"},
      auditAuthorizationSuccess: true
   })
```

> **Note**
>
> All DML logs have `authCheck` as `atype`. Only DDL has a different `atype`. If you put a value other than `authCheck` in the `filter`, it will not produce a DML log in CloudWatch.

Example: filtering by using multiple filters joined by operators

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.command":
               {
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ],
       "$nor": [
         {
            "param.command":
               {
                  $in: ["count", "insert", "delete", "update", "findandmodify" ]
               }
         }]
       },
      auditAuthorizationSuccess: true})
```

> **Note**
>
> At the top level, only `$and`, `$or`, and `$nor` are supported. Any other operators are not supported and will cause an error.

Example: filtering by events by `auditAuthorizationSuccess`

In this filter, all commands that have successfully passed authorization will not be logged:

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: false
   }
)
```

Example: filtering with `$in` and `$nin` conditions

When using both in `$in` and `$nin`, the command will not be logged as there will be an implicit "and" between the conditions. In this example, regex will block the `find` command so nothing will be logged:

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            atype: "authCheck",
            "param.command":
               {
                  $in: [ "find", "insert", "delete", "update", "findandmodify" ],
                  $nin: ["count", "insert", "delete", "update", "findandmodify" ],
                  $not: /^^find.*/
               }
         },
         ],
       "$or": [
         {
            "param.command":
               {
                  $nin: ["count", "insert", "delete", "update", "findandmodify" ]
               }
         }]
       },
      auditAuthorizationSuccess: true})
```

Example: filtering by `namespace`

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.ns":
               {
                  $in: [ "test.foo" ]
               }
```

```
         }
         ]},
      auditAuthorizationSuccess: true})
```

Example: resetting to default filter

Resetting to the default value means that every DML audit event will be logged. To reset filtering to the default value, run the following command:

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: true
   }
)
```

Backing up and restoring in Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) continuously backs up your data to Amazon Simple Storage Service (Amazon S3) for 1–35 days so that you can quickly restore to any point within the backup retention period. Amazon DocumentDB also takes automatic snapshots of your data as part of this continuous backup process.

> **Note**
>
> These are service-managed Amazon S3 buckets and you will not have access to the backup files. If you want to control your own backups, follow the instructions on Dumping, Restoring, Importing, and Exporting Data.

You can also retain backup data beyond the backup retention period by creating a manual snapshot of your cluster's data. The backup process does not impact your cluster's performance.

This section discusses the use cases for the backup capabilities in Amazon DocumentDB and shows you how to manage backups for your Amazon DocumentDB clusters.

Topics

- Back up and restore: concepts
- Understanding backup storage usage
- Dumping, restoring, importing, and exporting data
- Cluster snapshot considerations
- Comparing automatic and manual snapshots
- Creating a manual cluster snapshot
- Copying Amazon DocumentDB cluster snapshots
- Sharing Amazon DocumentDB cluster snapshots
- Restoring from a cluster snapshot
- Restoring to a point in time
- Deleting a cluster snapshot

Back up and restore: concepts

Noun	Descripti on	APIs (Verbs)

Backup retention period	A period of time between 1 and 35 days for which you can perform a point-in-time restore.	`create-db-cluster`
		
		`modify-db-cluster`
		
		`restore-db-cluster-to-point-in-time`

Amazon DocumentDB storage volume	Highly available and highly durable storage volume that replicate s data six ways across three Availability Zones. An Amazon DocumentD B cluster is highly durable	`create-db-cluster`
		
		`delete-db-cluster`

Noun	Description	APIs (Verbs)
	regardless of the number of instances in the cluster.	

Backup window	Period of time in the day in which automatic snapshots are taken.	`create-db-cluster`
		`describe-db-cluster`
		`modify-db-cluster`

Automatic snapshot	Daily snapshots that are full backups of cluster and are automatically created by the continuous backup process in Amazon DocumentDB.	`restore-db-cluster-from-snapshot`
		`describe-db-cluster-snapshot-attributes`
		`describe-db-cluster-snapshots`

Noun	Description	APIs (Verbs)

Manual snapshot	Snapshots you create manually to retain full backups of a cluster beyond the backup period.	`create-db-cluster-snapshot`
		`copy-db-cluster-snapshot`
		`delete-db-cluster-snapshot`
		`describe-db-cluster-snapshot-attributes`
		`describe-db-cluster-snapshots`
		`modify-db-cluster-snapshot-attribute`

Understanding backup storage usage

Amazon DocumentDB backup storage consists of continuous backups within the backup retention period and manual snapshots outside the retention period. To control your backup storage usage, you can reduce the backup retention interval, remove old manual snapshots when they are no longer needed, or both. For general information about Amazon DocumentDB backups, see Backing up and restoring in Amazon DocumentDB. For pricing information about Amazon DocumentDB backup storage, see Amazon DocumentDB Pricing.

To control your costs, you can monitor the amount of storage consumed by continuous backups and manual snapshots that persist beyond the retention period. Then you can reduce the backup retention interval and remove manual snapshots when they are no longer needed.

You can use the Amazon CloudWatch metrics `TotalBackupStorageBilled`, `SnapshotStorageUsed`, and `BackupRetentionPeriodStorageUsed` to review and monitor the amount of storage used by your Amazon DocumentDB backups, as follows:

- `BackupRetentionPeriodStorageUsed` represents the amount of backup storage used for storing continuous backups at the current time. This metric value depends on the size of the cluster volume and the number of changes you make during the retention period. However, for billing purposes the metric does not exceed the cumulative cluster volume size during the retention period. For example, if your cluster size is 100 GiB and your retention period is two

days, the maximum value for `BackRetentionPeriodStorageUsed` is 200 GiB (100 GiB + 100 GiB).

- `SnapshotStorageUsed` represents the amount of backup storage used for storing manual snapshots beyond the backup retention period. Manual snapshots taken within the retention period do not count against your backup storage. Similarly, automatic snapshots do not count against your backup storage. The size of each snapshot is the size of the cluster volume at the time you take the snapshot. The `SnapshotStorageUsed` value depends on the number of snapshots you keep and the size of each snapshot. For example, suppose that you have one snapshot outside the retention period and cluster volume size was 100 GiB when that snapshot was taken. The amount of `SnapshotStorageUsed` is 100 GiB.

- `TotalBackupStorageBilled` represents the sum of `BackupRetentionPeriodStorageUsed` and `SnapshotStorageUsed`, minus an amount of free backup storage equal to the size of cluster volume for one day. For example, if your cluster size is 100 GiB, you have one retention day, and you have one snapshot outside the retention period, the `TotalBackupStorageBilled` is 100 GiB (100 GiB + 100 GiB - 100 GiB).

- These metrics are computed independently for each Amazon DocumentDB cluster.

You can monitor your Amazon DocumentDB clusters and build reports using CloudWatch metrics through the CloudWatch console. For more information about how to use CloudWatch metrics, see Monitoring Amazon DocumentDB.

Dumping, restoring, importing, and exporting data

You can use the `mongodump`, `mongorestore`, `mongoexport`, and `mongoimport` utilities to move data in and out of your Amazon DocumentDB cluster. This section discusses the purpose of each of these tools and configurations to help you achieve better performance.

Topics

- mongodump
- mongorestore

- mongoexport
- mongoimport
- Tutorial

mongodump

The mongodump utility creates a binary (BSON) backup of a MongoDB database. The mongodump tool is the preferred method of dumping data from your source MongoDB deployment when looking to restore it into your Amazon DocumentDB cluster due to the size efficiencies achieved by storing the data in a binary format.

Depending on the resources available on the instance or machine you are using to perform the command, you can speed up your mongodump by increasing the number of parallel collections dumped from the default 1 using the `--numParallelCollections` option. A good rule of thumb is to start with one worker per vCPU on your Amazon DocumentDB cluster's primary instance.

> **Note**
>
> We recommend MongoDB Database Tools up to and including version 100.6.1 for Amazon DocumentDB. You can access the MongoDB Database Tools downloads here.

Example usage

The following is an example usage of the mongodump utility in the Amazon DocumentDB cluster, `sample-cluster`.

```
mongodump --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --out=sample-output-file \
    --numParallelCollections 4  \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

mongorestore

The `mongorestore` utility enables you to restore a binary (BSON) backup of a database that was created with the `mongodump` utility. You can improve restore performance by increasing the number of workers for each collection during the restore with the `--numInsertionWorkersPerCollection` option (the default is 1). A good rule of thumb is to start with one worker per vCPU on your Amazon DocumentDB cluster's primary instance.

Example usage

The following is an example usage of the `mongorestore` utility in the Amazon DocumentDB cluster, `sample-cluster`.

```
mongorestore --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem <fileToBeRestored>
```

mongoexport

The `mongoexport` tool exports data in Amazon DocumentDB to JSON, CSV, or TSV file formats. The `mongoexport` tool is the preferred method of exporting data that needs to be human or machine readable.

> **Note**
>
> `mongoexport` does not directly support parallel exports. However, it is possible to increase performance by executing multiple `mongoexport` jobs concurrently for different collections.

Example usage

The following is an example usage of the `mongoexport` tool in the Amazon DocumentDB cluster, `sample-cluster`.

```
mongoexport --ssl \
```

```
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --out=sample-output-file \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

mongoimport

The `mongoimport` tool imports the contents of JSON, CSV, or TSV files into an Amazon DocumentDB cluster. You can use the `--numInsertionWorkers` parameter to parallelize and speed up the import (the default is 1).

Example usage

The following is an example usage of the `mongoimport` tool in the Amazon DocumentDB cluster, `sample-cluster`.

```
mongoimport --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --file=<yourFile> \
    --numInsertionWorkers 4 \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

Tutorial

The following tutorial describes how to use the `mongodump`, `mongorestore`, `mongoexport`, and `mongoimport` utilities to move data in and out of an Amazon DocumentDB cluster.

1. **Prerequisites** — Before you begin, ensure that your Amazon DocumentDB cluster is provisioned and that you have access to an Amazon EC2 instance in the same VPC as your cluster. For more information, see Connect using Amazon EC2.

   To be able to use the mongo utility tools, you must have the mongodb-org-tools package installed in your EC2 instance, as follows.

```
sudo yum install mongodb-org-tools-4.0.18
```

Because Amazon DocumentDB uses Transport Layer Security (TLS) encryption by default, you must also download the Amazon RDS certificate authority (CA) file to use the mongo shell to connect, as follows.

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

2. **Download sample data** — For this tutorial, you will download some sample data that contains information about restaurants.

```
wget https://raw.githubusercontent.com/ozlerhakan/mongodb-json-files/master/
datasets/restaurant.json
```

3. **Import the sample data into Amazon DocumentDB** — Since the data is in a logical JSON format, you will use the `mongoimport` utility to import the data into your Amazon DocumentDB cluster.

```
mongoimport --ssl \
    --host="tutorialCluster.amazonaws.com:27017" \
    --collection=restaurants \
    --db=business \
    --file=restaurant.json \
    --numInsertionWorkers 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem
```

4. **Dump the data with `mongodump`** — Now that you have data in your Amazon DocumentDB cluster, you can take a binary dump of that data using the `mongodump` utility.

```
mongodump --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017"\
    --collection=restaurants \
    --db=business \
    --out=restaurantDump.bson \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
```

```
    --sslCAFile global-bundle.pem
```

5. **Drop the `restaurants` collection** — Before you restore the `restaurants` collection in the `business` database, you have to first drop the collection that already exists in that database, as follows.

```
use business
```

```
db.restaurants.drop()
```

6. **Restore the data with `mongorestore`** — With the binary dump of the data from Step 3, you can now use the `mongorestore` utility to restore your data to your Amazon DocumentDB cluster.

```
mongorestore --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017" \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem restaurantDump.bson
```

7. **Export the data using `mongoexport`** — To complete the tutorial, export the data from your cluster in the format of a JSON file, no different than the file you imported in Step 1.

```
mongoexport --ssl \
    --host="tutorialCluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=restaurants \
    --db=business \
    --out=restaurant2.json \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem
```

8. **Validation** — You can validate that the output of Step 5 yields the same result as Step 1 with the following commands.

```
wc -l restaurant.json
```

Output from this command:

```
2548 restaurant.json
```

```
wc -l restaurant2.json
```

Output from this command:

```
2548 restaurant2.json
```

Cluster snapshot considerations

Amazon DocumentDB creates daily automatic snapshots of your cluster during your cluster's backup window. Amazon DocumentDB saves the automatic snapshots of your cluster according to the backup retention period that you specify. If necessary, you can recover your cluster to any point in time during the backup retention period. Automatic snapshots don't occur while a copy operation is executing in the same Region for the same cluster.

Topics

- Backup storage
- Backup window
- Backup retention period
- Copy cluster snapshot encryption

In addition to automatic cluster snapshots, you can also manually create a cluster snapshot. You can copy both automatic and manual snapshots. For more information, see Creating a manual cluster snapshot and Copying Amazon DocumentDB cluster snapshots.

> **Note**
>
> Your cluster must be in the *available* state for an automatic snapshot to be taken.
> You can't share an Amazon DocumentDB automated cluster snapshot. As a workaround, you can create a manual snapshot by copying the automated snapshot, and then share that copy. For more information about copying a snapshot, see Copying Amazon DocumentDB cluster snapshots. For more information about restoring a cluster from a snapshot, see Restoring from a cluster snapshot.

Backup storage

Your Amazon DocumentDB backup storage for each AWS Region is composed of the backup storage needed for your backup retention period, which includes automatic and manual cluster snapshots in that Region. The default backup retention period is 1 day. For more information about backup storage pricing, see Amazon DocumentDB Pricing.

When you delete a cluster, all of its automatic snapshots are deleted and cannot be recovered. However, manual snapshots are not deleted when you delete a cluster. if you choose to have Amazon DocumentDB create a final snapshot (manual snapshot) before your cluster is deleted, you can use the final snapshot to recover your cluster.

For more information on snapshots and storage, see Understanding backup storage usage.

Backup window

Automatic snapshots occur daily during the preferred backup window. If the snapshot requires more time than allotted to the backup window, the backup process continues until it finishes, even though the backup window has ended. The backup window can't overlap with the weekly maintenance window for the cluster.

If you don't specify a preferred backup window when you create the cluster, Amazon DocumentDB assigns a default 30-minute backup window. This window is chosen at random from an 8-hour block of time associated with your cluster's Region. You can change your preferred backup window by modifying the cluster. For more information, see Modifying an Amazon DocumentDB cluster.

Region Name	Region	UTC Time Block
US East (Ohio)	us-east-2	03:00-11:00
US East (N. Virginia)	us-east-1	03:00-11:00
US West (Oregon)	us-west-2	06:00-14:00
Africa (Cape Town)	af-south-1	03:00–11:00
Asia Pacific (Hong Kong)	ap-east-1	06:00-14:00
Asia Pacific (Hyderabad)	ap-south-2	06:30–14:30

Region Name	Region	UTC Time Block
Asia Pacific (Mumbai)	ap-south-1	06:00-14:00
Asia Pacific (Seoul)	ap-northeast-2	13:00-21:00
Asia Pacific (Singapore)	ap-southeast-1	14:00-22:00
Asia Pacific (Sydney)	ap-southeast-2	12:00-20:00
Asia Pacific (Tokyo)	ap-northeast-1	13:00-21:00
Canada (Central)	ca-central-1	03:00-11:00
China (Beijing)	cn-north-1	06:00-14:00
China (Ningxia)	cn-northwest-1	06:00-14:00
Europe (Frankfurt)	eu-central-1	21:00-05:00
Europe (Ireland)	eu-west-1	22:00-06:00
Europe (London)	eu-west-2	22:00-06:00
Europe (Milan)	eu-south-1	02:00-10:00
Europe (Paris)	eu-west-3	23:59-07:29
Europe (Spain)	eu-south-2	02:00–10:00
Europe (Stockholm)	eu-north-1	04:00–12:00
Middle East (UAE)	me-central-1	05:00–13:00
South America (São Paulo)	sa-east-1	00:00-08:00
Israel (Tel Aviv)	il-central-1	04:00-12:00
AWS GovCloud (US-East)	us-gov-east-1	17:00-01:00
AWS GovCloud (US-West)	us-gov-west-1	06:00-14:00

Backup retention period

The backup retention period is the number of days an automatic backup is retained before being automatically deleted. Amazon DocumentDB supports a backup retention period of 1–35 days.

You can set the backup retention period when you create a cluster. If you don't explicitly set the backup retention period, the default backup retention period of 1 day is assigned to your cluster. After you create a cluster, you can modify the backup retention period by modifying the cluster using either the AWS Management Console or the AWS CLI. For more information, see Modifying an Amazon DocumentDB cluster.

Copy cluster snapshot encryption

Cluster and snapshot encryption is based on a KMS encryption key. The KMS key ID is the Amazon Resource Name (ARN), KMS key identifier, or the KMS key alias for the KMS encryption key.

The following guidelines and limitations apply:

- Encryption is inferred from the cluster when creating a snapshot. If the cluster is encrypted, the snapshot of that cluster is encrypted with the same KMS key. If the cluster is not encrypted, the snapshot is not encrypted.
- If you copy an encrypted cluster snapshot from your Amazon Web Services account, you can specify a value for `KmsKeyId` to encrypt the copy with a new KMS encryption key. If you don't specify a value for `KmsKeyId` , then the copy of the cluster snapshot is encrypted with the same KMS key as the source cluster snapshot.
- If you copy an encrypted cluster snapshot that is shared from another Amazon Web Services account, then you must specify a value for `KmsKeyId`.
- To copy an encrypted cluster snapshot to another Amazon Web Services Region, set `KmsKeyId` to the KMS key ID that you want to use to encrypt the copy of the cluster snapshot in the destination Region. KMS encryption keys are specific to the Amazon Web Services Region that they are created in, and you can't use encryption keys from one Amazon Web Services Region in another Amazon Web Services Region.
- If you copy an unencrypted cluster snapshot and specify a value for the `KmsKeyId` parameter, an error is returned.

Comparing automatic and manual snapshots

The following are key features of Amazon DocumentDB (with MongoDB compatibility) automatic and manual snapshots.

Amazon DocumentDB automatic snapshots have the following key features:

- **Automatic snapshot naming** — Automatic snapshot names follow the pattern `rds:<cluster-name>-yyyy-mm-dd-hh-mm`, with `yyyy-mm-dd-hh-mm` representing the date and time the snapshot was created.
- **Created automatically on a schedule** — When you create or modify a cluster, you can set the *backup retention period* to an integer value from 1 to 35 days. By default, new clusters have a backup retention period of 1 day. The backup retention period defines the number of days that automatic snapshots are kept before being automatically deleted. You can't disable automatic backups on Amazon DocumentDB clusters.

  In addition to setting the backup retention period, you also set the *backup window*, the time of day during which automatic snapshots are created.
- **Deleting automatic snapshots** — Automatic snapshots are deleted when you delete the automatic snapshot's cluster. You can't manually delete an automatic snapshot.
- **Incremental** — During the backup retention period, database updates are recorded so that there is an incremental record of changes.
- **Restoring from an automatic snapshot** — You can restore from an automatic snapshot using the AWS Management Console or the AWS CLI. When you restore from a snapshot using the AWS CLI, you must add instances separately after the cluster is *available*.
- **Sharing** — You can't share an Amazon DocumentDB automated cluster snapshot. As a workaround, you can create a manual snapshot by copying the automated snapshot, and then share that copy. For more information about copying a snapshot, see Copying Amazon DocumentDB cluster snapshots. For more information about restoring a cluster from a snapshot, see Restoring from a cluster snapshot.
- **You can restore from any point within the backup retention period** — Because database updates are incrementally recorded, you can restore your cluster to any point in time within the backup retention period.

  When you restore from an automatic snapshot or from a point-in-time restore using the AWS CLI, you must add instances separately after the cluster is *available*.

Amazon DocumentDB manual snapshots have the following key features:

- **Created on demand** — Amazon DocumentDB manual snapshots are created on demand using the Amazon DocumentDB Management Console or AWS CLI.
- **Deleting a manual snapshot** —A manual snapshot is deleted only when you explicitly delete it using either the Amazon DocumentDB console or AWS CLI. A manual snapshot is not deleted when you delete its cluster.
- **Full backups** — When a manual snapshot is taken, a full backup of your cluster's data is created and stored.
- **Manual snapshot naming** — You specify the manual snapshot name. Amazon DocumentDB does not add a `datetime` stamp to the name, so you must add that information if you want it included in the name.
- **Restoring from a manual snapshot** —You can restore from a manual snapshot using the console or the AWS CLI. When you restore from a snapshot using the AWS CLI, you must add instances separately after the cluster is *available*.
- **Service Quotas** — You are limited to a maximum of 100 manual snapshots per AWS Region.
- **Sharing** — You can share manual cluster snapshots, which can be copied by authorized AWS accounts. You can share encrypted or unencrypted manual snapshots. For more information about copying a snapshot, see Copying Amazon DocumentDB cluster snapshots.
- **You restore to when the manual snapshot was taken** —When you restore from a manual snapshot, you restore to when the manual snapshot was taken.

  When you restore from a snapshot using the AWS CLI, you must add instances separately after the cluster is *available*.

Creating a manual cluster snapshot

You can create a manual snapshot using either the AWS Management Console or AWS CLI. The amount of time it takes to create a snapshot varies with the size of your databases. When you create a snapshot, you must do the following:

1. Identify which cluster to back up.
2. Give your snapshot a name. This allows you to restore from it later.

Using the AWS Management Console

To create a manual snapshot using the AWS Management Console, you can follow either method below.

1. **Method 1:**

   1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
   2. In the navigation pane, choose **Snapshots**.

      > **Tip**
      >
      > If you don't see the navigation pane on the left side of your screen, choose the menu icon
      > (≡)
      > in the upper-left corner of the page.

   3. On the **Snapshots** page, choose **Create**.
   4. On the **Create cluster snapshot** page:

      a. **Cluster identifier** — From the drop-down list of clusters, choose the cluster that you want to create a snapshot of.

      b. **Snapshot identifier** — Enter a name for your snapshot.

         Snapshot naming constraints:

         - Length is [1–255] letters, numbers, or hyphens.
         - First character must be a letter.
         - Cannot end with a hyphen or contain two consecutive hyphens.
         - Must be unique for all clusters (across Amazon RDS, Amazon Neptune, and Amazon DocumentDB) per AWS account, per Region.

      c. Choose **Create**.

2. **Method 2:**

   1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
   2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. On the **Clusters** page, choose the button to the left of the cluster that you want to snapshot.
4. From the **Actions** menu, choose **Take snapshot**.
5. On the **Create cluster snapshot** page:
   a. **Snapshot identifier** — Enter a name for your snapshot.

      Snapshot naming constraints:

      - Length is [1–63] letters, numbers, or hyphens.
      - First character must be a letter.
      - Cannot end with a hyphen or contain two consecutive hyphens.
      - Must be unique for all clusters (across Amazon RDS, Amazon Neptune, and Amazon DocumentDB) per AWS account, per Region.

   b. Choose **Create**.

Using the AWS CLI

To create a cluster snapshot using the AWS CLI, use the `create-db-cluster-snapshot` operation with the following parameters.

Parameters

- **`--db-cluster-identifier`** — Required. The name of the cluster that you are taking a snapshot of. This cluster must exist and be *available*.
- **`--db-cluster-snapshot-identifier`** — Required. The name of the manual snapshot that you are creating.

The following example creates a snapshot named `sample-cluster-snapshot` for a cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

For Windows:

```
aws docdb create-db-cluster-snapshot ^
    --db-cluster-identifier sample-cluster ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Output from this operation looks something like the following.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-24T04:59:08.475Z",
        "Engine": "docdb",
        "Status": "creating",
        "Port": 0,
        "VpcId": "vpc-abc0123",
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:<accountID>:cluster-
snapshot:sample-cluster-snapshot"
    }
}
```

Copying Amazon DocumentDB cluster snapshots

In Amazon DocumentDB, you can copy manual and automatic snapshots within the same AWS Region or to a different AWS Region within the same account. You can also share snapshots owned by other AWS accounts in the same AWS Region. However, you can't copy a cluster snapshot across AWS Regions and AWS account in a single step. These actions must be performed individually.

As an alternative to copying, you can also share manual snapshots with other AWS accounts. For more information, see Sharing Amazon DocumentDB cluster snapshots.

> **Note**
>
> Amazon DocumentDB bills you based upon the amount of backup and snapshot data you keep and the period of time that you keep it. For more information about the storage associated with Amazon DocumentDB backups and snapshots, see Understanding backup storage usage. For pricing information about Amazon DocumentDB storage, see Amazon DocumentDB Pricing.

Topics

- Copying shared snapshots
- Copying snapshots across AWS Regions
- Limitations
- Handling encryption
- Parameter group considerations
- Copying a cluster snapshot

Copying shared snapshots

You can copy snapshots shared to you by other AWS accounts. If you are copying an encrypted snapshot that has been shared from another AWS account, you must have access to the AWS KMS encryption key that was used to encrypt the snapshot.

You can only copy a shared snapshot in the same AWS Region, whether the snapshot is encrypted or not. For more information, see Handling encryption.

Copying snapshots across AWS Regions

When you copy a snapshot to an AWS Region that is different from the source snapshot's AWS Region, each copy is a full snapshot. A full snapshot copy contains all of the data and metadata required to restore the Amazon DocumentDB cluster.

Depending on the AWS Regions involved and the amount of data to be copied, a cross-region snapshot copy can take hours to complete. In some cases, there might be a large number of cross-region snapshot copy requests from a given source AWS Region. In these cases, Amazon DocumentDB might put new cross-region copy requests from that source AWS Region into a queue until some in-progress copies complete. No progress information is displayed about copy requests while they are in the queue. Progress information is displayed when the copy starts.

Limitations

The following are some limitations when you copy snapshots:

- If you delete a source snapshot before the target snapshot becomes available, the snapshot copy may fail. Verify that the target snapshot has a status of `AVAILABLE` before you delete a source snapshot.
- You can have up to five snapshot copy requests in progress to a single destination Region per account.
- Depending on the regions involved and the amount of data to be copied, a cross-region snapshot copy can take hours to complete. For more information, see Copying snapshots across AWS Regions.

Handling encryption

You can copy a snapshot that has been encrypted using an AWS KMS encryption key. If you copy an encrypted snapshot, the copy of the snapshot must also be encrypted. If you copy an encrypted snapshot within the same AWS Region, you can encrypt the copy with the same AWS KMS encryption key as the original snapshot, or you can specify a different AWS KMS encryption key. If you copy an encrypted snapshot across Regions, you can't use the same AWS KMS encryption key for the copy as used for the source snapshot, because AWS KMS keys are Region-specific. Instead, you must specify an AWS KMS key valid in the destination AWS Regionn.

The source snapshot remains encrypted throughout the copy process. For more information, see Data protection in Amazon DocumentDB.

> **Note**
>
> For Amazon DocumentDB cluster snapshots, you can't encrypt an unencrypted cluster snapshot when you copy the snapshot.

Parameter group considerations

When you copy a snapshot across Regions, the copy doesn't include the parameter group used by the original Amazon DocumentDB cluster. When you restore a snapshot to create a new cluster, that cluster gets the default parameter group for the AWS Region it is created in. To give the new cluster the same parameters as the original, you must do the following:

1. In the destination AWS Region, create an Amazon DocumentDB cluster parameter group with the same settings as the original cluster. If one already exists in the new AWS Region, you can use that one.
2. After you restore the snapshot in the destination AWS Region, modify the new Amazon DocumentDB cluster and add the new or existing parameter group from the previous step. For more information, see Modifying an Amazon DocumentDB cluster.

Copying a cluster snapshot

You can copy an Amazon DocumentDB cluster using the AWS Management Console or the AWS CLI, as follows.

Using the AWS Management Console

To make a copy of a cluster snapshot using the AWS Management Console, complete the following steps. This procedure works for copying encrypted or unencrypted cluster snapshots, in the same AWS Region or across Regions.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Snapshots**, and then choose the button to the left of the snapshot that you want to copy.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. From the **Actions** menu, choose **Copy**.
4. In the resulting **Make Copy of cluster snapshot** page, complete the **Settings** section.
   a. **Destination Region** — Optional. To copy the cluster snapshot to a different AWS Region, choose that AWS Region for **Destination Region**.
   b. **New snapshot identifier** — Enter a name for the new snapshot.

      Target snapshot naming constraints:

      - Cannot be the name of an existing snapshot.
      - Length is [1—63] letters, numbers, or hyphens.
      - First character must be a letter.
      - Cannot end with a hyphen or contain two consecutive hyphens.
      - Must be unique for all clusters across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.
   c. **Copy tags** — To copy any tags you have on your source snapshot to your snapshot copy, choose **Copy tags**.
5. Complete the **Encryption-at-rest** section.
   a. **Encryption at rest** — If your snapshot is not encrypted, these options are not available to you because you cannot create an encrypted copy from an unencrypted snapshot. If your snapshot is encrypted, you can change the AWS KMS key used during encryption at rest.

      For more information about encrypting snapshot copies, see Copy cluster snapshot encryption.

      For more information about encryption at rest, see Encrypting Amazon DocumentDB data at rest.

b. **AWS KMS Key** — From the drop-down list, choose one of the following:

- **(default) aws/rds** — The account number and AWS KMS key ID are listed following this option.
- **<some-key-name>** — If you created a key, it is listed and available for you to choose.
- **Enter a key ARN** — In the **ARN** box, enter the Amazon Resource Name (ARN) for your AWS KMS key. The format of the ARN is `arn:aws:kms:<region>:<accountID>:key/<key-id>` .

6. To make a copy of the selected snapshot, choose **Copy snapshot**. Alternatively, you can choose **Cancel** to not make a copy of the snapshot.

Using the AWS CLI

To make a copy of an unencrypted cluster snapshot using the AWS CLI, use the `copy-db-cluster-snapshot` operation with the following parameters. If you are copying the snapshot to another AWS Region, run the command in the AWS Region to which the snapshot will be copied.

- **`--source-db-cluster-snapshot-identifier`** — Required. The identifier of the cluster snapshot to make a copy of. The cluster snapshot must exist and be in the *available* state. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region. This parameter is not case sensitive.
- **`--target-db-cluster-snapshot-identifier`** — Required. The identifier of the new cluster snapshot to create from the source cluster snapshot. This parameter is not case sensitive.

  Target snapshot naming constraints:

  - Cannot be the name of an existing snapshot.
  - Length is [1—63] letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all clusters across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.
- **`--source-region`** — If you are copying the snapshot to another AWS Region, specify the AWS Region that the encrypted cluster snapshot will be copied from.

If you're copying the snapshot to another AWS Region and you don't specify `--source-region`, you must specify the `pre-signed-url` option instead. The `pre-signed-url` value must be a URL that contains a Signature Version 4 signed request for the `CopyDBClusterSnapshot` action to be called in the source AWS Region where the cluster snapshot is copied from. To learn more about the `pre-signed-url`, see CopyDBClusterSnapshot.

- **`--kms-key-id`** — The KMS key identifier for the key to use to encrypt the copy of the cluster snapshot.

  If you are copying an encrypted cluster snapshot to another AWS Region, this parameter is required. You must specify a KMS key for the destination AWS Region.

  If you are copying an encrypted cluster snapshot in the same AWS Region, the AWS KMS key parameter is optional. The copy of the cluster snapshot is encrypted with the same AWS KMS key as the source cluster snapshot. If you want to specify a new AWS KMS encryption key to use to encrypt the copy, you can do so using this parameter.
- **`--copy-tags`** — Optional. The tags and values to be copied over.

To cancel a copy operation once it's in progress, you can delete the target cluster snapshot identified by `--target-db-cluster-snapshot-identifier` or `TargetDBClusterSnapshotIdentifier` while that cluster snapshot is in **copying** status.

Example

Example 1: Copy an unencrypted snapshot to the same Region

The following AWS CLI example creates a copy of `sample-cluster-snapshot` named `sample-cluster-snapshot-copy` in the same AWS Region as the source snapshot. When the copy is made, all tags on the original snapshot are copied to the snapshot copy.

For Linux, macOS, or Unix:

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
    --copy-tags
```

For Windows:

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
    --copy-tags
```

Output from this operation looks something like the following.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-03-27T08:40:24.805Z",
        "Engine": "docdb",
        "Status": "copying",
        "Port": 0,
        "VpcId": "vpc-abcd0123",
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:111122223333:key/sample-key-id",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot"
    }
}
```

Example

Example 2: Copy an unencrypted snapshot across AWS Regions

The following AWS CLI example creates a copy of `sample-cluster-snapshot`, which has the ARN `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:sample-cluster-`

snapshot. This copy is named named sample-cluster-snapshot-copy and is in the AWS Region in which the command is run.

For Linux, macOS, or Unix:

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
east-1:123456789012:cluster-snapshot:sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

For Windows:

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
east-1:123456789012:cluster-snapshot:sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

Output from this operation looks something like the following.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-29T16:45:51.239Z",
        "Engine": "docdb",
        "AllocatedStorage": 0,
        "Status": "copying",
        "Port": 0,
        "VpcId": "vpc-abc0123",
        "ClusterCreateTime": "2020-04-28T16:43:00.294Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "LicenseModel": "docdb",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": false,
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot-copy",
```

```
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot",
    }
}
```

Example

Example 3: Copy an encrypted snapshot across AWS Regions

The following AWS CLI example creates a copy of `sample-cluster-snapshot` from the us-west-2 Region to the us-east-1 Region. This command is called in the us-east-1 Region.

For Linux, macOS, or Unix:

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:cluster-snapshot:sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
    --source-region us-west-2 \
    --kms-key-id sample-us-east-1-key
```

For Windows:

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:cluster-snapshot:sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
    --source-region us-west-2 ^
    --kms-key-id sample-us-east-1-key
```

Output from this operation looks something like the following.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "ayhu-xrsc-test-ap-southeast-1-small-cluster-kms",
        "SnapshotCreateTime": "2020-04-29T16:45:53.159Z",
        "Engine": "docdb",
        "AllocatedStorage": 0,
        "Status": "copying",
        "Port": 0,
```

```
        "ClusterCreateTime": "2020-04-28T16:43:07.129Z",
        "MasterUsername": "chimera",
        "EngineVersion": "4.0.0",
        "LicenseModel": "docdb",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:111122223333:key/sample-key-id",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-west-2:111122223333:cluster-
snapshot:sample-cluster-snapshot",
    }
}
```

Note

For more information about encrypting snapshot copies, see Copy cluster snapshot encryption.
For more information about encryption at rest, see Encrypting Amazon DocumentDB data at rest.

Sharing Amazon DocumentDB cluster snapshots

Using Amazon DocumentDB, you can share a manual cluster snapshot in the following ways:

- Sharing a manual cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to copy the snapshot.
- Sharing a manual cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to directly restore a cluster from the snapshot instead of taking a copy of it and restoring from that.

Note

To share an automated cluster snapshot, create a manual cluster snapshot by copying the automated snapshot, and then share that copy. This process also applies to AWS Backup–generated resources.

You can share a manual snapshot with up to 20 other AWS accounts. You can also share an unencrypted manual snapshot as public, which makes the snapshot available to all accounts. When sharing a snapshot as public, ensure that none of your private information is included in any of your public snapshots.

When sharing manual snapshots with other AWS accounts, and you restore a cluster from a shared snapshot using the AWS CLI or the Amazon DocumentDB API, you must specify the Amazon Resource Name (ARN) of the shared snapshot as the snapshot identifier.

Sharing an encrypted snapshot

The following restrictions apply to sharing encrypted snapshots:

- You can't share encrypted snapshots as public.
- You can't share a snapshot that has been encrypted using the default AWS KMS encryption key of the account that shared the snapshot.

Follow these steps to share encrypted snapshots.

1. Share the AWS Key Management Service (AWS KMS) encryption key that was used to encrypt the snapshot with any accounts that you want to be able to access the snapshot.

   You can share AWS KMS encryption keys with another AWS accounts by adding the other accounts to the AWS KMS key policy. For details on updating a key policy, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*. For an example of creating a key policy, see Creating an IAM policy to enable copying of the encrypted snapshot later in this topic.
2. Use the AWS CLI, as shown below, to share the encrypted snapshot with the other accounts.

Allowing access to an AWS KMS encryption key

For another AWS account to copy an encrypted snapshot shared from your account, the account that you share your snapshot with must have access to the AWS KMS key that encrypted the snapshot. To allow another account access to an AWS KMS key, update the key policy for the AWS KMS key with the ARN of the account that you are sharing to as a principal in the AWS KMS key policy. Then allow the `kms:CreateGrant` action.

After you give an account access to your AWS KMS encryption key, to copy your encrypted snapshot, that account must create an AWS Identity and Access Management (IAM) user if it doesn't already have one. In addition, that account must also attach an IAM policy to that IAM user that allows the user to copy an encrypted snapshot using your AWS KMS key. The account must be an IAM user and cannot be a root AWS account identity due to AWS KMS security restrictions.

In the following key policy example, user 123451234512 is the owner of the AWS KMS encryption key. User 123456789012 is the account that the key is being shared with. This updated key policy gives the account access to the AWS KMS key. It does this by including the ARN for the root AWS account identity for user 123456789012 as a principal for the policy, and by allowing the `kms:CreateGrant` action.

```
{
    "Id": "key-policy-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {"AWS": [
                "arn:aws:iam::123451234512:user/KeyUser",
                "arn:aws:iam::123456789012:root"
            ]},
            "Action": [
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"},
            {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {"AWS": [
                "arn:aws:iam::123451234512:user/KeyUser",
                "arn:aws:iam::123456789012:root"
            ]},
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
```

```
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
        }
    ]
}
```

Creating an IAM policy to enable copying of the encrypted snapshot

When the external AWS account has access to your AWS KMS key, the owner of that account can create a policy to allow an IAM user that is created for the account to copy an encrypted snapshot that is encrypted with that AWS KMS key.

The following example shows a policy that can be attached to an IAM user for AWS account 123456789012. The policy enables the IAM user to copy a shared snapshot from account 123451234512 that has been encrypted with the AWS KMS key `c989c1dd-a3f2-4a5d-8d96-e793d082ab26` in the us-west-2 Region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey",
                "kms:CreateGrant",
                "kms:RetireGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:123451234512:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"]
        },
        {
            "Sid": "AllowAttachmentOfPersistentResources",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
```

```
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:123451234512:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"],
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

For details on updating a key policy, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Sharing a snapshot

You can share an Amazon DocumentDB manual cluster snapshot (or a copy of an automated snapshot) using the AWS Management Console or the AWS CLI:

Using the AWS Management Console

To share a snapshot using the AWS Management Console, complete the following steps:

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Snapshots**.
3. Select the manual snapshot that you want to share.
4. In the **Actions** drop-down menu, choose Share.
5. Choose one of the following options for **DB snapshot visibility**:
   - If the source is unencrypted, choose **Public** to permit all AWS accounts to restore a cluster from your manual snapshot. Or choose **Private** to permit only AWS accounts that you specify to restore a cluster from your manual snapshot.

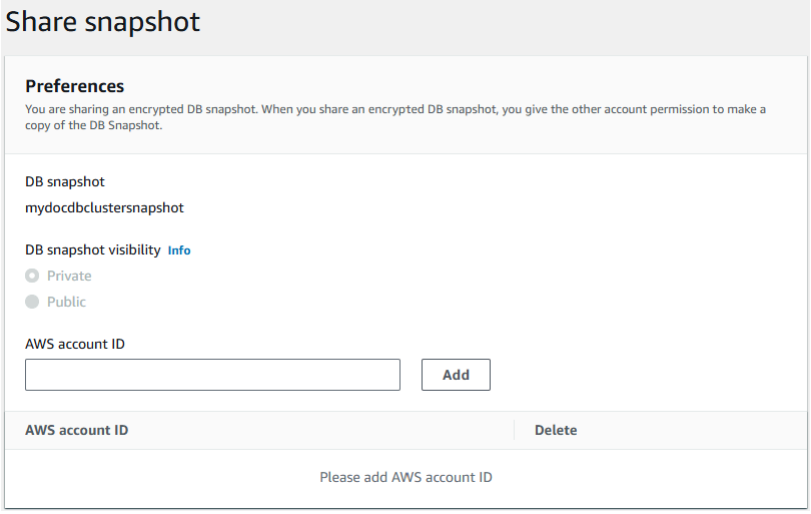
> **Warning**
>
> If you set **DB snapshot visibility** to **Public**, all AWS accounts can restore a cluster from your manual snapshot and have access to your data. Do not share any manual cluster snapshots that contain private information as **Public**.

- If the source is encrypted, **DB snapshot visibility** is set as **Private** because encrypted snapshots can't be shared as public.

> **Note**
>
> Snapshots that have been encrypted with the default AWS KMS key can't be shared.

6. For **AWS Account ID**, enter the AWS account identifier for an account that you want to permit to restore a cluster from your manual snapshot, and then choose **Add**. Repeat to include additional AWS account identifiers, up to 20 AWS accounts.

   If you make an error when adding an AWS account identifier to the list of permitted accounts, you can delete it from the list by choosing **Delete** at the right of the incorrect AWS account identifier.



7. After you have added identifiers for all of the AWS accounts that you want to permit to restore the manual snapshot, choose **Save** to save your changes.

Using the AWS CLI

To share a snapshot using the AWS CLI, use the Amazon DocumentDB `modify-db-snapshot-attribute` operation. Use the `--values-to-add` parameter to add a list of the IDs for the AWS accounts that are authorized to restore the manual snapshot.

The following example permits two AWS account identifiers, 123451234512 and 123456789012, to restore the snapshot named `manual-snapshot1`. It also removes the `all` attribute value to mark the snapshot as private.

For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-add '["123451234512","123456789012"]'
```

For Windows:

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-add '["123451234512","123456789012"]'
```

Output from this operation looks something like the following.

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512",
                    "123456789012"
                ]
            }
        ]
    }
}
```

To remove an AWS account identifier from the list, use the `--values-to-remove` parameter. The following example prevents AWS account ID 123456789012 from restoring the snapshot.

For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-remove '["123456789012"]'
```

For Windows:

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-remove '["123456789012"]'
```

Output from this operation looks something like the following.

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512"
                ]
            }
        ]
    }
}
```

Restoring from a cluster snapshot

Amazon DocumentDB (with MongoDB compatibility) creates a cluster snapshot of your storage volume. You can create a new cluster by restoring from a cluster snapshot. When you restore the cluster, you provide the name of the cluster snapshot to restore from and a name for the new cluster that is created by the restore. You can't restore from a snapshot to an existing cluster because a new cluster is created when you restore.

When you are restoring a cluster from a cluster snapshot:

- This action restores only the cluster, and not the instances for that cluster. You must invoke the `create-db-instance` action to create instances for the restored cluster, specifying the identifier of the restored cluster in `--db-cluster-identifier`. You can create instances only after the cluster is *available*.
- You cannot restore an encrypted snapshot to an unencrypted cluster. However, you can restore an unencrypted snapshot to an encrypted cluster by specifying the AWS KMS key.
- To restore a cluster from an encrypted snapshot, you must have access to the AWS KMS key.

> **Note**
>
> You cannot restore a 3.6 cluster to a 4.0 cluster but you can migrate from one cluster version to another. For more information, go to Migrating to Amazon DocumentDB.

Using the AWS Management Console

The following procedure shows how to restore an Amazon DocumentDB cluster from a cluster snapshot using the Amazon DocumentDB Management Console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Snapshots**, and then choose the button to the left of the snapshot that you want to use to restore a cluster.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. On the **Actions** menu, choose **Restore**.
4. On the **Restore snapshot** page, complete the **Configuration** section.

a. **Cluster identifier** — The name for the new cluster. You can accept the Amazon DocumentDB supplied name or type a name that you prefer. The Amazon DocumentDBsupplied name is in the format of `docdb-` plus a UTC timestamp; for example, `docdb-yyyy-mm-dd-hh-mm-ss`.

b. **Instance class** — The instance class for the new cluster. You can accept the default instance class or choose an instance class from the drop-down list.

c. **Number of instances** — The number of instances you want created with this cluster. You can accept the default of 3 instances (1 primary read/write and 2 read-only replicas) or choose the number of instances from the drop-down list.

5. For **Cluster storage configuration**, choose a storage option.

> **Note**
>
> **Amazon DocumentDB I/O-Optimized** storage configuration is only available on the Amazon DocumentDB 5.0 engine version.

6. If you are satisfied with the cluster configuration, choose **Restore cluster** and wait while your cluster is restored.

7. If you prefer to change some configurations, such as specifying a non-default Amazon VPC or security group, choose **Show advanced settings** from the bottom left of the page, and then continue with the following steps.

   a. Complete the **Network settings** section.

      - **Virtual Private Cloud (VPC)** — Accept the current VPC, or choose a VPC from the drop-down list.
      - **Subnet Group** — Accept the `default` subnet group, or choose one from the drop-down list.
      - **VPC Security Groups** — Accept the `default (VPC)` security group, or choose one from the list.

   b. Complete the **Cluster options** section.

      - **Database port** — Accept the default port, `27017`, or use the up or down arrow to set the port that you want to use for application connections.

   c. Complete the **Encryption** section.

- **Encryption at rest** — If your snapshot is encrypted, these options are not available to you. If it is not encrypted, you can choose one of the following:
  - To encrypt all your cluster's data, choose **Enable encryption-at-rest**. If you choose this option, you must designate a KMS key.
  - To not encrypt your cluster's data, choose **Disable encryption-at-rest**. If you choose this option, you are finished with the encryption section.
- **AWS KMS Key** — Choose one of the following from the drop-down list:
  - **(default) aws/rds** — The account number and AWS KMS key ID are listed following this option.
  - **Customer-managed key** — This option is available only if you created an IAM encryption key in the AWS Identity and Access Management (IAM) console. You can choose the key to encrypt your cluster.
  - **Enter a key ARN** — In the **ARN** box, enter the Amazon Resource Name (ARN) for your AWS KMS key. The format of the ARN is `arn:aws:kms:<region>:<accountID>:key/<key-id>`.

d. Complete the **Log exports** section.

- **Select the log types to publish to CloudWatch** — Choose one of the following:
  - **Enabled** — Enables your cluster to export DDL logging to Amazon CloudWatch Logs.
  - **Disabled** — Prevents your cluster from exporting DDL logs to Amazon CloudWatch Logs. **Disabled** is the default.
- **IAM role**—From the list, choose *RDS Service Linked Role*.

e. Complete the **Tags** section.

- **Add Tag** — In the *Key* box, enter the name for the tag for your cluster. In the *Value* box, optionally enter the tag value. Tags are used with AWS Identity and Access Management (IAM) policies to manage access to Amazon DocumentDB resources and to control what actions can be applied to the resources.

f. Complete the **Deletion protection** section.

- **Enable deletion protection** — Protects the cluster from being accidentally deleted. While this option is enabled, you can't delete the cluster.

8. Choose **Restore cluster**.

Using the AWS CLI

To restore a cluster from a snapshot using the AWS CLI, use the `restore-db-cluster-from-snapshot` operation with the following parameters. For more information, see RestoreDBClusterFromSnapshot.

- **`--db-cluster-identifier`** — Required. The name of the cluster that is created by the operation. A cluster by this name cannot exist before this operation.

  Cluster naming constraints:

  - Length is [1—63] letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all clusters across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.
- **`--snapshot-identifier`** — Required. The name of the snapshot used to restore from. A snapshot by this name must exist and be in the *available* state.
- **`--engine`** — Required. Must be `docdb`.
- **`--storage-type standard | iopt1`** — Optional. Default: `standard`.
- **`--kms-key-id`** — Optional. The ARN of the AWS KMS key identifier to use when restoring an encrypted snapshot or encrypting a cluster when restoring from an unencrypted snapshot. Supplying the AWS KMS key ID results in the restored cluster being encrypted with the AWS KMS key, whether or not the snapshot was encrypted.

  The format of the `--kms-key-id` is `arn:aws:kms:<region>:<accountID>:key/<key-id>`. If you do not specify a value for the `--kms-key-id` parameter, then the following occurs:

  - If the snapshot in `--snapshot-identifier` is encrypted, then the restored cluster is encrypted using the same AWS KMS key that was used to encrypt the snapshot.
  - If the snapshot in `--snapshot-identifier` is not encrypted, then the restored cluster is not encrypted.

For Linux, macOS, or Unix:

```
aws docdb restore-db-cluster-from-snapshot \
    --db-cluster-identifier sample-cluster-restore \
```

```
    --snapshot-identifier sample-cluster-snapshot \
    --engine docdb \
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/SAMPLE-KMS-KEY-ID
```

For Windows:

```
aws docdb restore-db-cluster-from-snapshot ^
    --db-cluster-identifier sample-cluster-restore ^
    --snapshot-identifier sample-cluster-snapshot ^
    --engine docdb ^
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/SAMPLE-KMS-KEY-ID
```

Output from this operation looks something like the following.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster-restore",
        "DBClusterParameterGroup": "default.docdb4.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "sample-cluster-restore.cluster-node.us-
east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster-restore.cluster-node.us-
east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "Port": 27017,
        "MasterUsername": "<master-user>",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcdefgh",
                "Status": "active"
            }
```

```
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/<sample-key-id>",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster-
restore",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-04-01T01:43:40.871Z",
        "DeletionProtection": true
    }
}
```

After the cluster status is *available*, create at least one instance for the cluster.

For Linux, macOS, or Unix:

```
aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster-restore  \
    --db-instance-identifier sample-cluster-restore-instance \
    --availability-zone us-east-1b \
    --promotion-tier 2 \
    --db-instance-class db.r5.large \
    --engine docdb
```

For Windows:

```
aws docdb create-db-instance ^
    --db-cluster-identifier sample-cluster-restore  ^
    --db-instance-identifier sample-cluster-restore-instance ^
    --availability-zone us-east-1b ^
    --promotion-tier 2 ^
    --db-instance-class db.r5.large ^
    --engine docdb
```

Output from this operation looks something like the following.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-cluster-restore-instance",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "creating",
```

```
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcdefgh",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-west-2b",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcdefgh",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    ...
                }
            ]
        },
        "PreferredMaintenanceWindow": "fri:09:43-fri:10:13",
        "PendingModifiedValues": {},
        "EngineVersion": "4.0.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster-restore",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/<sample-key-id>",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 2,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-cluster-
restore-instance"
    }
}
```

Restoring to a point in time

You can restore a cluster to any point in time that is within the cluster's backup retention period using the AWS Management Console or AWS Command Line Interface (AWS CLI).

> **Note**
>
> You cannot conduct a point-in-time restore of a 3.6 cluster to a 4.0 cluster but you can migrate from one cluster version to another. For more information, go to Migrating to Amazon DocumentDB.

Keep the following in mind when restoring a cluster to a point in time.

- The new cluster is created with the same configuration as the source cluster, except that the new cluster is created with the default parameter group. To set the new cluster's parameter group to the source cluster's parameter group, modify the cluster after it is *available*. For more information on modifying a cluster, see Modifying an Amazon DocumentDB cluster.

Using the AWS Management Console

You can restore a cluster to a point-in-time within its backup retention period by completing the following using the AWS Management Console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**. In the list of clusters, choose the button to the left of the cluster that you want to restore.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. On the **Actions** menu, choose **Restore to point in time**.
4. Complete the **Restore time** section, which specifies the date and time to restore to.

a. **Restore date**—Choose or enter a date that is between the **Earliest restore time** and the **Latest restore time**.

b. **Restore time**—Choose or enter the hour, minute, and seconds that are between the **Earliest restore time** and the **Latest restore time**.

5. Complete the **Configuration** section.

   a. **Cluster identifier** — Accept the default identifier, or enter an identifier that you prefer.

      Cluster naming constraints:

      - Length is [1—63] letters, numbers, or hyphens.
      - First character must be a letter.
      - Cannot end with a hyphen or contain two consecutive hyphens.
      - Must be unique for all clusters across Amazon RDS, Neptune and Amazon DocumentDB per AWS account, per Region.

   b. **Instance class** — From the drop-down list, choose the instance class that you want for the cluster's instances.

   c. **Number of instances** — From the drop-down list, choose the number of instances that you want created when the cluster is restored.

6. For **Cluster storage configuration**, choose a storage option.

   > **Note**
   >
   > **Amazon DocumentDB I/O-Optimized** storage configuration is only available on the Amazon DocumentDB 5.0 engine version.

7. Optional. To configure the network settings, cluster options, and enable log exports, choose **Show advanced settings**, and then complete the following sections. Otherwise, continue with the next step.

   - **Network settings**

     1. **Virtual Private Cloud (VPC)** — From the drop-down list, choose the VPC that you want to use for this cluster.
     2. **Subnet group** — From the drop-down list, choose the subnet group for this cluster.

3. **VPC security groups** — From the drop-down list, choose the VPC security groups for this cluster.

- **Cluster options**
  1. **Port** — Accept the default port (27017), or use the up and down arrows to set the port for communicating with this cluster.

- **Log exports**
  1. **Audit logs** — Select this option to enable exporting audit logs to Amazon CloudWatch Logs. If you select this option, you must enable `audit_logs` in the cluster's custom parameter group. For more information, see Auditing Amazon DocumentDB events.
  2. **Profiler logs** — Select this option to enable exporting operation profiler logs to Amazon CloudWatch Logs. If you select this option, you must also modify the following parameters in the cluster's custom parameter group:
     - `profiler` — Set to `enabled`.
     - `profiler_threshold_ms` — Set to a value `[0-INT_MAX]` to set the threshold for profiling operations.
     - `profiler_sampling_rate` — Set to a value `[0.0-1.0]` to set the percentage of slow operations to profile.

     For more information, see Profiling Amazon DocumentDB operations.
  3. **Profiler logs** — Export profiler logs to Amazon CloudWatch
  4. **IAM role** — From the drop-down list, choose *RDS Service Linked Role*.
- **Tags**
  1. **Add Tag** — In the *Key* box, enter the name for the tag for your cluster. In the *Value* box, optionally enter the tag value. Tags are used with AWS Identity and Access Management (IAM) policies to manage access to Amazon DocumentDB resources and to control what actions can be applied to the resources.
- **Deletion protection**
  1. **Enable deletion protection** — Protects the cluster from being accidentally deleted. While this option is enabled, you can't delete the cluster.

8. To restore the cluster, choose **Create cluster**. Alternatively, you can choose **Cancel** to cancel the operation.

Using the AWS CLI

To restore a cluster to a point in time using the snapshot's backup retention period, use the `restore-db-cluster-to-point-in-time` operation with the following parameters.

- **`--db-cluster-identifier`**— Required. The name of the new cluster to be created. This cluster cannot exist before the operation. The parameter value must meet the following constraints.

  Cluster naming constraints:

  - Length is [1—63] letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all clusters across Amazon RDS, Neptune and Amazon DocumentDB per AWS account, per Region.
- **`--restore-to-time`** — The UTC date and time to restore the cluster to. For example, `2018-06-07T23:45:00Z`.

  Time Constraints:

  - Must be before the latest restorable time for the cluster.
  - Must be specified if the `--use-latest-restorable-time` parameter is not provided.
  - Cannot be specified if the `--use-latest-restorable-time` parameter is `true`.
  - Cannot be specified if the `--restore-type` parameter value is `copy-on-write`.
- **`--source-db-cluster-identifier`** — The name of the source cluster from which to restore. This cluster must exist and be available.
- **`--use-latest-restorable-time`** or **`--no-use-latest-restorable-time`** — Whether to restore to the latest restorable backup time. Cannot be specified if the `--restore-to-time` parameter is provided.
- **`--storage-type standard | iopt1`** — Optional. Default: `standard`.

The AWS CLI operation `restore-db-cluster-to-point-in-time` only restores the cluster, not the instances for that cluster. You must invoke the `create-db-instance` operation to

create instances for the restored cluster, specifying the identifier of the restored cluster in `--db-cluster-identifier`. You can create instances only after the `restore-db-cluster-to-point-in-time` operation has completed and the restored cluster is *available*.

Example

The following example creates `sample-cluster-restored` from the snapshot `sample-cluster-snapshot` to the latest restorable time.

For Linux, macOS, or Unix:

```
aws docdb restore-db-cluster-to-point-in-time \
    --db-cluster-identifier sample-cluster-restored \
    --source-db-cluster-identifier sample-cluster-snapshot \
    --use-latest-restorable-time
```

For Windows:

```
aws docdb restore-db-cluster-to-point-in-time ^
    --db-cluster-identifier sample-cluster-restored ^
    --source-db-cluster-identifier sample-cluster-snapshot ^
    --use-latest-restorable-time
```

Example

The following example creates `sample-cluster-restored` from the snapshot `sample-cluster-snapshot` to 03:15 on December 11, 2018 (UTC), which is within the backup retention period of `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb restore-db-cluster-to-point-in-time \
    --db-cluster-identifier sample-cluster-restore \
    --source-db-cluster-identifier sample-cluster \
    --restore-to-time 2020-05-12T03:15:00Z
```

For Windows:

```
aws docdb restore-db-cluster-to-point-in-time ^
    --db-cluster-identifier sample-cluster-restore ^
    --source-db-cluster-identifier sample-cluster ^
```

```
    --restore-to-time 2020-05-12T03:15:00Z
```

Output from this operation looks something like the following.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-west-2b",
            "us-west-2a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster-restored",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "sample-cluster-restored.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster-restored.node.us-
east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "Port": 27017,
        "MasterUsername": "master-user",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abc0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID^>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster-
restored",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-04-24T20:14:36.713Z",
        "DeletionProtection": false
    }
```

```
}
```

Deleting a cluster snapshot

A manual snapshot is a full backup that is deleted only when you manually delete it using the AWS Management Console or AWS CLI. You cannot manually delete an automatic snapshot because automatic snapshots are deleted only when the snapshot's retention period expires or you delete the snapshot's cluster.

Using the AWS Management Console

To delete a manual cluster snapshot using the AWS Management Console, complete the following steps.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Snapshots**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the list of snapshots, choose the button to the left of the snapshot that you want to delete. The snapshot's type must be **manual**.

   1. You can verify that that the snapshot's type is **manual** by checking if it is listed as `manual` or `automatic` under the **Type** column.
4. From the **Actions** menu, choose **Delete**. If the **Delete** option is unavailable, you probably chose an automatic snapshot.
5. On the delete confirmation screen, to delete the snapshot, choose **Delete**. To keep the snapshot, choose **Cancel**.

Using the AWS CLI

An Amazon DocumentDB manual cluster snapshot is a full backup that you can manually delete using the AWS CLI. You cannot manually delete an automatic snapshot.

To delete a manual cluster snapshot using the AWS CLI, use the `delete-db-cluster-snapshot` operation with the following parameters.

Parameters

- **`--db-cluster-snapshot-identifier`** — Required. The name of the manual snapshot to delete.

The following example deletes the cluster snapshot `sample-cluster-snapshot`.

For Linux, macOS, or Unix:

```
aws docdb delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

For Windows:

```
aws docdb delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Output from this operation lists the details of the cluster snapshot you deleted.

Managing Amazon DocumentDB resources

These sections cover the various components and their related tasks for managing your Amazon DocumentDB (with MongoDB compatibility) implementation.

Topics

- Amazon DocumentDB Operational tasks overview
- Overview of Amazon DocumentDB global clusters
- Managing Amazon DocumentDB clusters
- Managing Amazon DocumentDB instances
- Managing Amazon DocumentDB subnet groups
- Amazon DocumentDB High availability and replication
- Managing Amazon DocumentDB indexes
- Managing collection-level document compression
- Managing Amazon DocumentDB events
- Choosing regions and availability zones
- Managing Amazon DocumentDB cluster parameter groups
- Understanding Amazon DocumentDB endpoints
- Understanding Amazon DocumentDB Amazon Resource Names (ARNs)
- Tagging Amazon DocumentDB resources
- Maintaining Amazon DocumentDB
- Understanding service-linked roles
- Using change streams with Amazon DocumentDB
- Using AWS Lambda with change streams

Amazon DocumentDB Operational tasks overview

This section covers operational tasks for your Amazon DocumentDB cluster, and how to accomplish these tasks using the AWS CLI.

Topics

- Adding a replica to an Amazon DocumentDB cluster

- Describing clusters and instances
- Creating a cluster snapshot
- Restoring from a snapshot
- Removing an instance from a cluster
- Deleting a cluster

Adding a replica to an Amazon DocumentDB cluster

After you create the primary instance for your Amazon DocumentDB cluster, you can add one or more *replicas*. A replica is a read-only instance that serves two purposes:

- **Scalability** — If you have a large number of clients that require concurrent access, you can add more replicas for read-scaling.
- **High availability** — If the primary instance fails, Amazon DocumentDB automatically fails over to a replica instance and designates it as the new primary. If a replica fails, other instances in the cluster can still serve requests until the failed node can be recovered.

Each Amazon DocumentDB cluster can support up to 15 replicas.

> **Note**
>
> For maximum fault tolerance, you should deploy replicas in separate Availability Zones. This helps ensure that your Amazon DocumentDB cluster can continue to function, even if an entire Availability Zone becomes unavailable.

The following AWS CLI example shows how to add a new replica. The `--availability-zone` parameter places the replica in the specified Availability Zone.

```
aws docdb create-db-instance \
    --db-instance-identifier sample-instance \
    --db-cluster-identifier sample-cluster \
    --engine docdb \
    --db-instance-class db.r5.large \
    --availability-zone us-east-1a
```

Describing clusters and instances

The following AWS CLI example lists all Amazon DocumentDB clusters in a Region. For certain management features such as cluster and instance lifecycle management, Amazon DocumentDB leverages operational technology that is shared with Amazon RDS. The `filterName=engine,Values=docdb` filter parameter returns only Amazon DocumentDB clusters.

For more information on describing and modifying clusters, see the Amazon DocumentDB cluster lifecycle.

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

Output from this operation looks something like the following.

```
{
    "DBClusters": [
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-1",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-2",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
```

```
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-3",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        }
    ]
}
```

The following AWS CLI example lists the instances in an Amazon DocumentDB cluster. For more information on describing and modifying clusters, see the Amazon DocumentDB instance lifecycle.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

The output looks like something like below. In this output, there are two instances. The primary instance is `sample-instance-1` (`"IsClusterWriter": true`). There is also a replica instance, `sample-instance2` (`"IsClusterWriter: false"`).

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-cluster-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
```

```
            }
        ]
    ]
]
```

Creating a cluster snapshot

A *cluster snapshot* is a complete backup of the data in your Amazon DocumentDB cluster. When the snapshot is being created, Amazon DocumentDB reads your data directly from the cluster volume. Because of this, you can create a snapshot even if your cluster doesn't have any instances running at the time. The amount of time it takes to create a snapshot depends on the size of your cluster volume.

Amazon DocumentDB supports automatic backups, which occur daily during the preferred backup window — a 30-minute period of time during the day. The following AWS CLI example shows how to view the backup window for your cluster:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].PreferredBackupWindow'
```

The output shows the backup window (in UTC):

```
[
    "00:18-00:48"
]
```

You can define the backup window when you create your Amazon DocumentDB cluster. You can also change the backup window, as shown in the following example. If you don't define a backup window, Amazon DocumentDB automatically assigns one to your cluster.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --preferred-backup-window "02:00-02:30"
```

In addition to automatic backups, you can manually create a cluster snapshot at any time. When you do this, you specify which cluster you want to back up, and a unique name for your snapshot so that you can restore from it later.

The following AWS CLI example shows how to create a snapshot of your data.

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Restoring from a snapshot

You can restore a cluster snapshot to a new Amazon DocumentDB cluster. To do this, you provide the name of the snapshot and the name of a new cluster. You can't restore from a snapshot to an existing cluster; instead, Amazon DocumentDB creates a new cluster when you restore and then populates it with your snapshot data.

The following example shows all the snapshots for the cluster `sample-cluster`.

```
aws docdb describe-db-cluster-snapshots \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusterSnapshots[*].[DBClusterSnapshotIdentifier,SnapshotType,Status]'
```

The output looks something like the following. A manual snapshot is one that you created manually, whereas an automated snapshot is created by Amazon DocumentDB within the cluster backup window.

```
[
        "sample-cluster-snapshot",
        "manual",
        "available"
    ],
    [
        "rds:sample-cluster",
        "automated",
        "available"
    ]
]
```

The following example shows how to restore an Amazon DocumentDB cluster from a snapshot.

```
aws docdb restore-db-cluster-from-snapshot \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster \
    --snapshot-identifier sample-cluster-snapshot
```

The new cluster does not have any instances associated with it; so if you want to interact with the cluster, you must add an instance to it.

```
aws docdb create-db-instance \
    --db-instance-identifier new-sample-instance \
    --db-instance-class db.r5.large \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster
```

You can use the following AWS CLI operations to monitor the progress of cluster and instance creation. When the cluster and instance statuses are available, you can connect to the new cluster's endpoint and access your data.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier new-sample-cluster  \
    --query 'DBClusters[*].[Status,Endpoint]'
```

```
aws docdb describe-db-instances \
    --db-instance-identifier new-sample-instance \
    --query 'DBInstances[*].[DBInstanceStatus]'
```

Removing an instance from a cluster

Amazon DocumentDB stores all of your data in the cluster volume. The data persists in that cluster volume, even if you remove all the instances from your cluster. If you need to access the data again, you can add an instance to the cluster at any time, and pick up where you left off.

The following example shows how to remove an instance from your Amazon DocumentDB cluster.

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

Deleting a cluster

Before you can delete an Amazon DocumentDB cluster, you must first remove all of its instances. The following AWS CLI example returns information about the instances in a cluster. If this operation returns any instance identifiers, you have to delete each of the instances. For more information, see Removing an instance from a cluster.

```
aws docdb describe-db-clusters \
```

```
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers[*].DBInstanceIdentifier'
```

When there are no more instances remaining, you can delete the cluster. At that time, you must choose one of the following options:

- **Create a final snapshot** — Capture all the cluster data in a snapshot so that you can re-create a new instance with that data later. The following example shows how to do this:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-snapshot
```

- **Skip the final snapshot** — Permanently discard all the cluster data. This cannot be reversed. The following example shows how to do this:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

Overview of Amazon DocumentDB global clusters

What is a global cluster?

A global cluster consists of one primary region and up to five read-only secondary regions. You issue write operations directly to the primary cluster in the primary region and Amazon DocumentDB automatically replicates the data to the secondary regions using dedicated infrastructure. Latency is typically under a second.

How are global clusters useful?

- **Recovery from region-wide outages** — In the event of a region-wide outage, you can promote one of the secondary clusters to a primary cluster within minutes, with a typical Recovery Time Objective (RTO) of under a minute. The Recovery Point Objective (RPO) is typically measured in seconds, but this depends on the lag across the network at the time of the failure.
- **Global reads with local latency** — If you have offices around the world, you can use a global cluster to keep your main sources of information updated in the primary region. Offices in your other regions can access the information in their own region, with local latency.

- **Scalable secondary clusters** — You can scale your secondary clusters by adding more read-only instances to a secondary region. The secondary cluster is read-only, so it can support up to 16 read-only replica instances rather than the usual limit of 15 for a single cluster.
- **Fast replication from primary to secondary clusters** — The replication performed by a global cluster has little performance impact on the primary database cluster. The resources of the DB instances are fully devoted to serve application read and write workloads.

What are the current limitations of global clusters?

- Global clusters are not supported on Amazon DocumentDB v3.6.
- Global clusters are not supported on t3, t4g, and r4 instance types.
- Global clusters are not available in the following Regions: South America (São Paulo), Europe (Milan), China (Beijing), and China (Ningxia).
- Switchover and global failover are not supported when regions are on different engine versions. Manual failover is supported when there's an engine version mismatch.
- Only the primary cluster performs write operations. Clients that perform write operations connect to the cluster endpoint of the primary cluster.
- You can have a maximum of five secondary regions and one primary region for your cluster.
- A secondary cluster cannot be stopped. A primary cluster cannot be stopped if it has secondary clusters associated with it. Only a regional cluster that has no secondary clusters can be stopped.
- Replicas attached to the secondary cluster can restart under certain circumstances. If the primary region's instance restarts or fails over, replicas in the secondary region also restart. The cluster is then unavailable until all replicas are back in sync with the primary database cluster's writer instance. This behavior is expected. Be sure that you understand the impact to your global cluster before making changes to your primary cluster.
- You cannot use change streams on secondary clusters.

Topics

- Quick start guide: global clusters
- Managing an Amazon DocumentDB global cluster
- Connect to an Amazon DocumentDB global cluster
- Monitoring Amazon DocumentDB global clusters
- Disaster recovery and Amazon DocumentDB global clusters

Quick start guide: global clusters

Topics

- Configuration
- Creating an Amazon DocumentDB global cluster
- Adding an AWS Region to an Amazon DocumentDB global cluster
- Using a snapshot for your Amazon DocumentDB global cluster

Configuration

Amazon DocumentDB global cluster spans at least two AWS Regions. The primary Region supports a cluster that has one primary (writer) instance and up to 15 replica instances, while a secondary Region runs a read-only cluster made up entirely of up to 16 replica instances. A global cluster can have up to five secondary Regions. The table lists the maximum clusters, instances, and replicas allowed in a global cluster.

Description	Primary AWS Region	Secondary AWS Region
Clusters	1	5 (maximum)
Writer instances	1	0
Read-only instances (Amazon DocumentDB replicas), per cluster	15 (max)	16 (total)
Read-only instances (max allowed, given actual number of secondary Regions)	15 - *s*	*s* = total number of secondary AWS Regions

The clusters have the following specific requirements:

- **Database instance class requirements** — You can only use the `db.r5` and `db.r6g` instance classes.
- **AWS Region requirements** — The primary cluster must be in one Region, and at least one secondary cluster must be in a different Region of the same account. You can create up to five

secondary (read-only) clusters, and each must be in a different Region. In other words, no two clusters can be in the same Region.

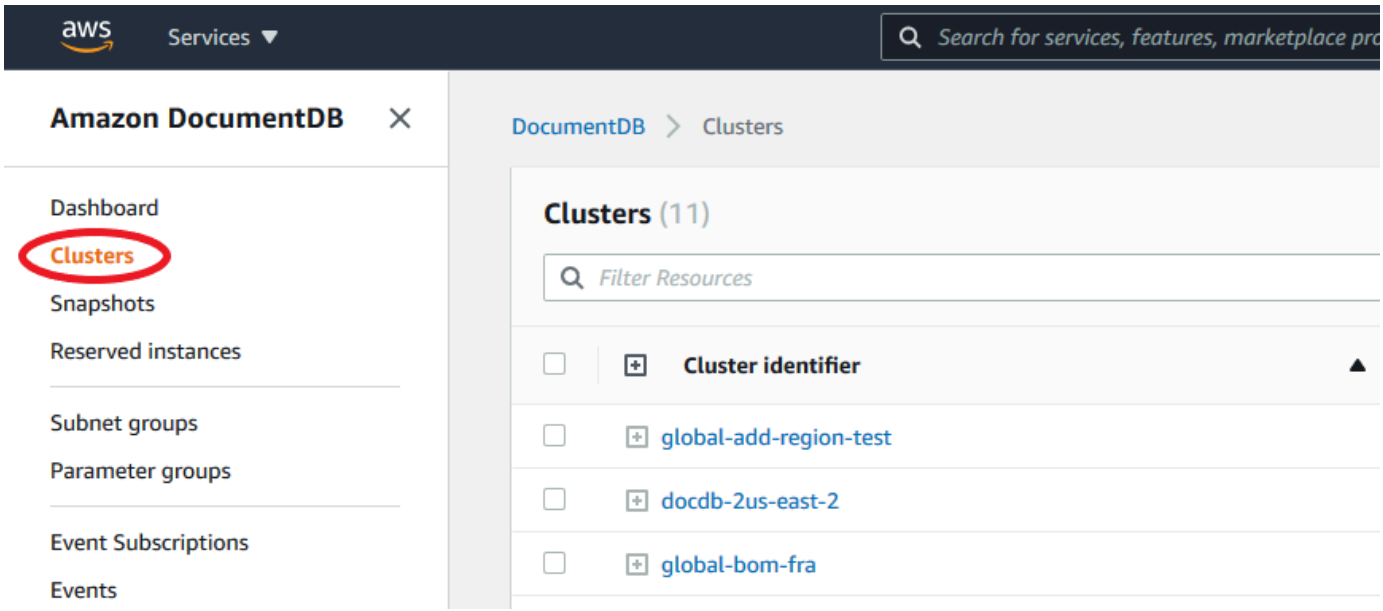
- **Naming requirements** — The names you choose for each of your clusters must be unique, across all Regions. You can't use the same name for different clusters even though they're in different Regions.

Creating an Amazon DocumentDB global cluster

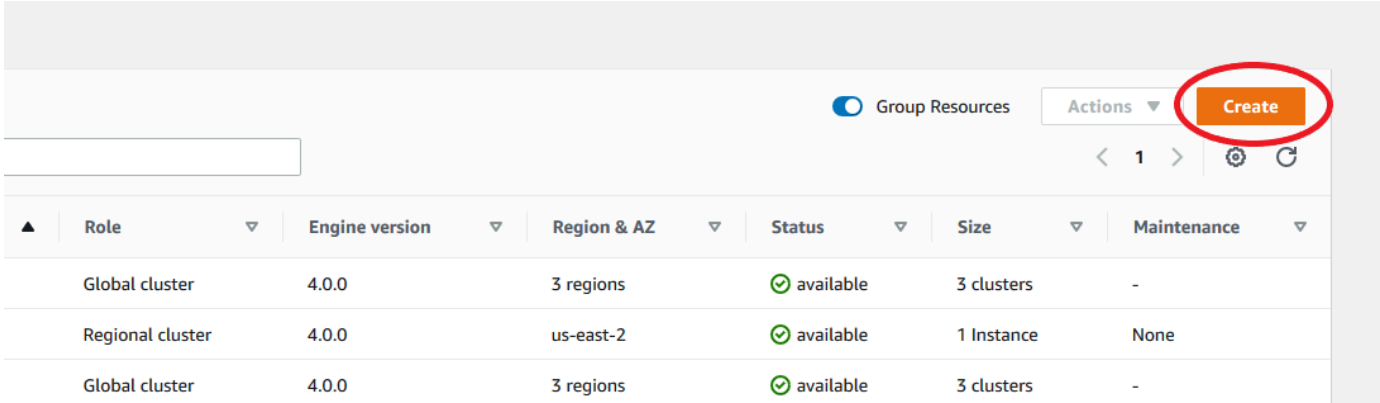
Are you ready to build your first global cluster? In this section we will explain how to create a brand new global cluster with new database clusters and instances, using either the AWS Management Console or AWS CLI with the following instructions.

Using the AWS Management Console

1. In the AWS Management Console, navigate to **Amazon DocumentDB**.
2. When you get to the Amazon DocumentDB console, choose **Clusters**.

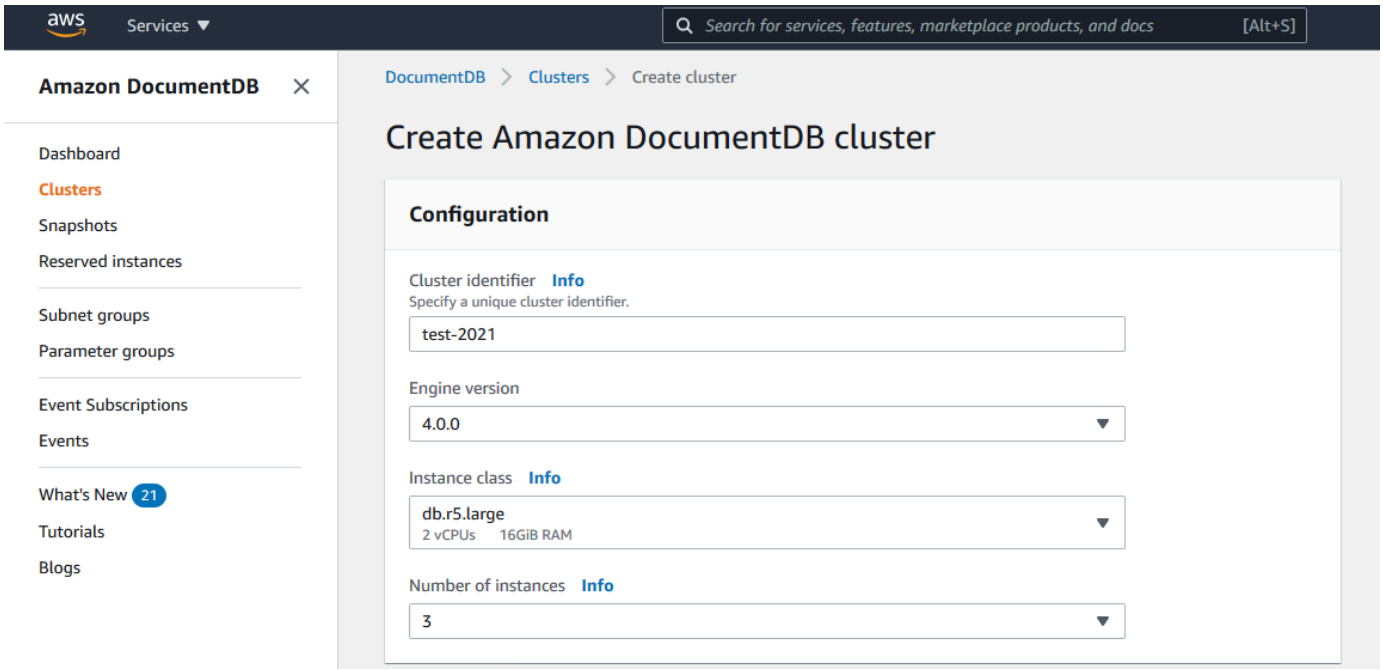


3. Choose **Create**.

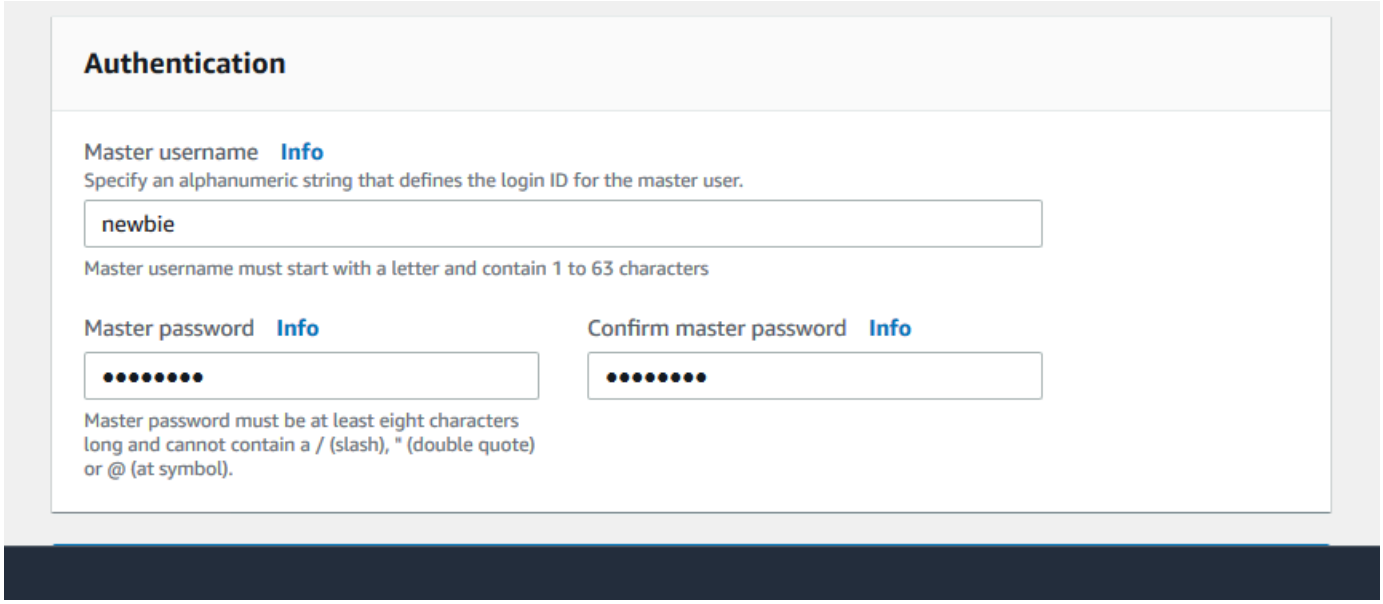


4. Fill out the **Configuration** section of the **Create Amazon DocumentDB Cluster** form accordingly:

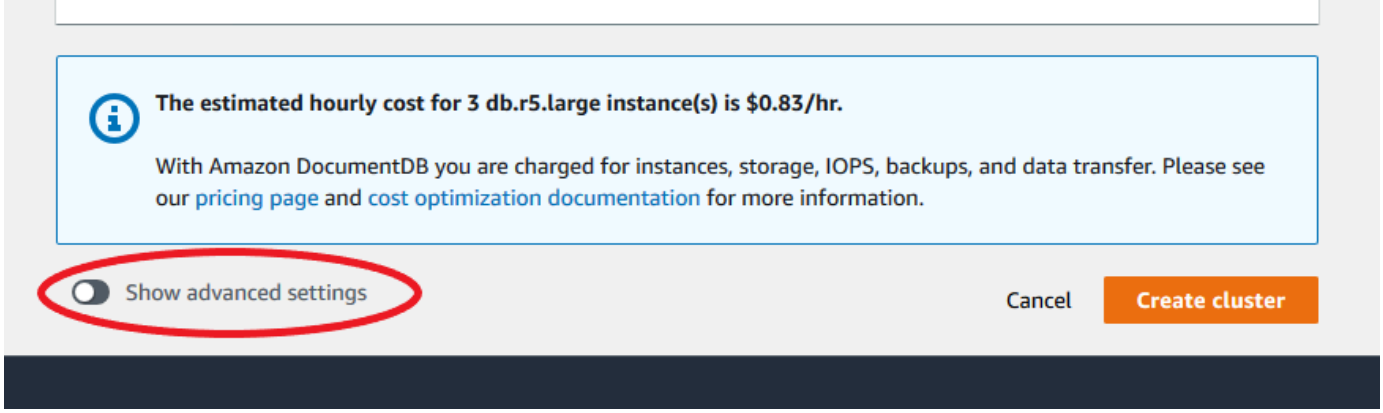
   - **Cluster identifier**: You can either enter a unique identifier for this instance or allow Amazon DocumentDB to provide the instance identifier based on the cluster identifier.
   - Engine version: Choose **4.0.0**
   - Instance class: Choose **db.r5.large**
   - Number of instances: Choose **3**.



5. In the **Authentication** section, fill in a master username and master password.

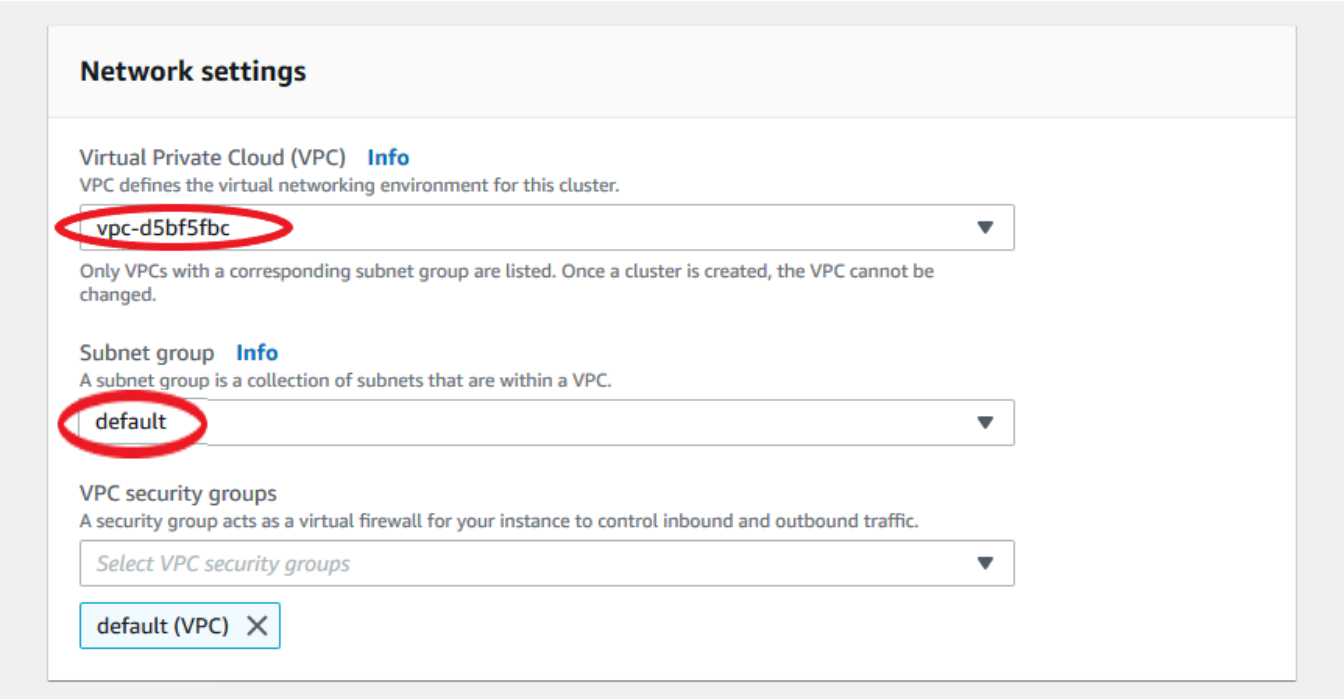


6. Choose **Show advanced settings**.

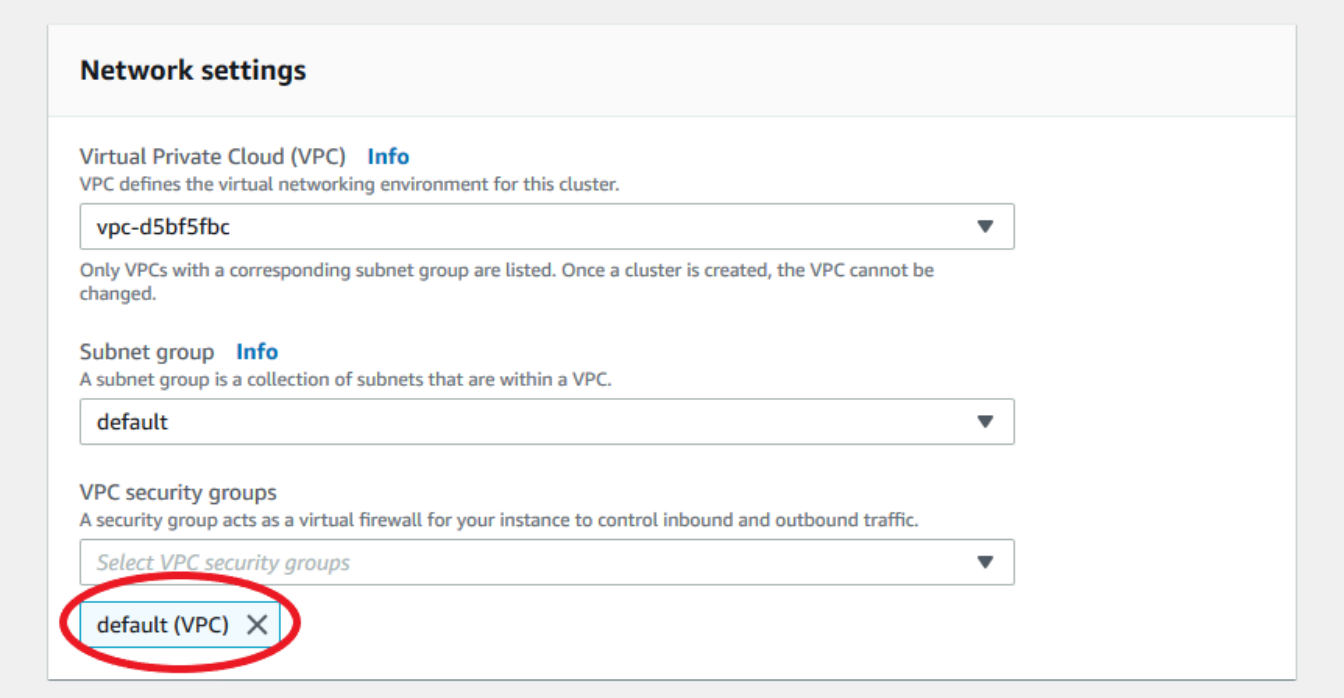


7. In the **Network settings** section:

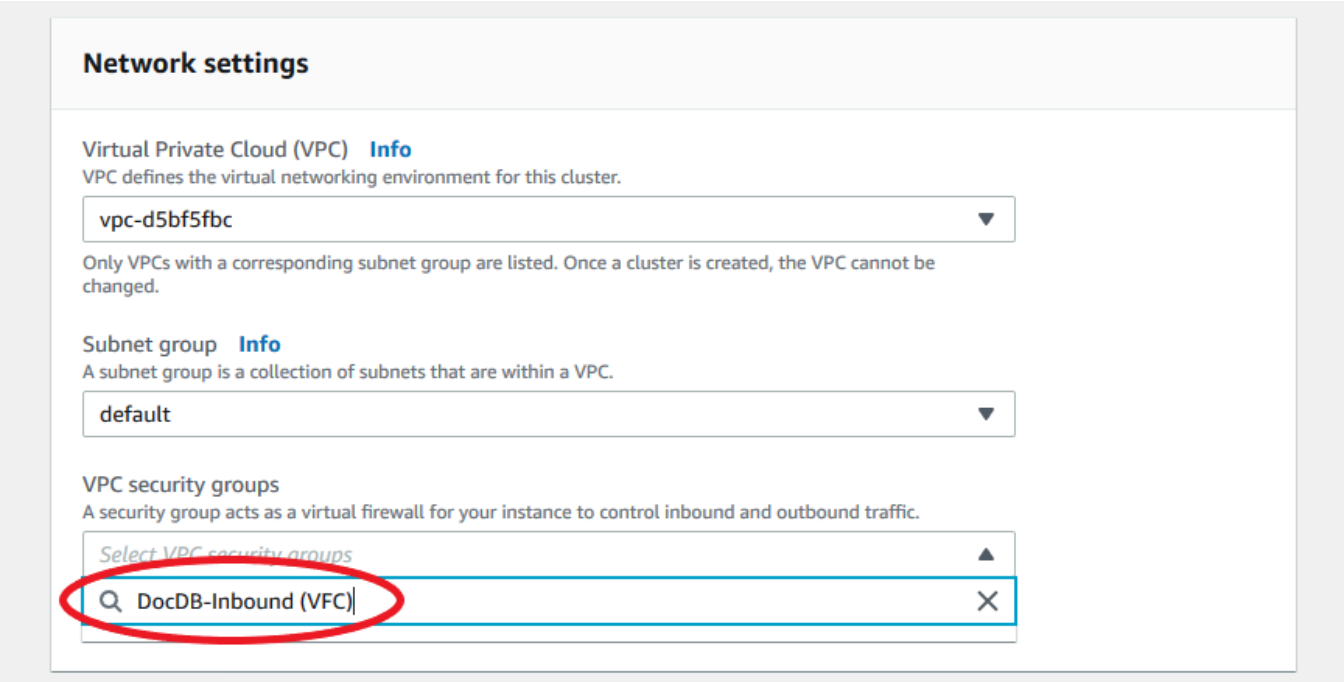
   - Keep default options for **Virtual Private Cloud (VPC)** and **Subnet group**.



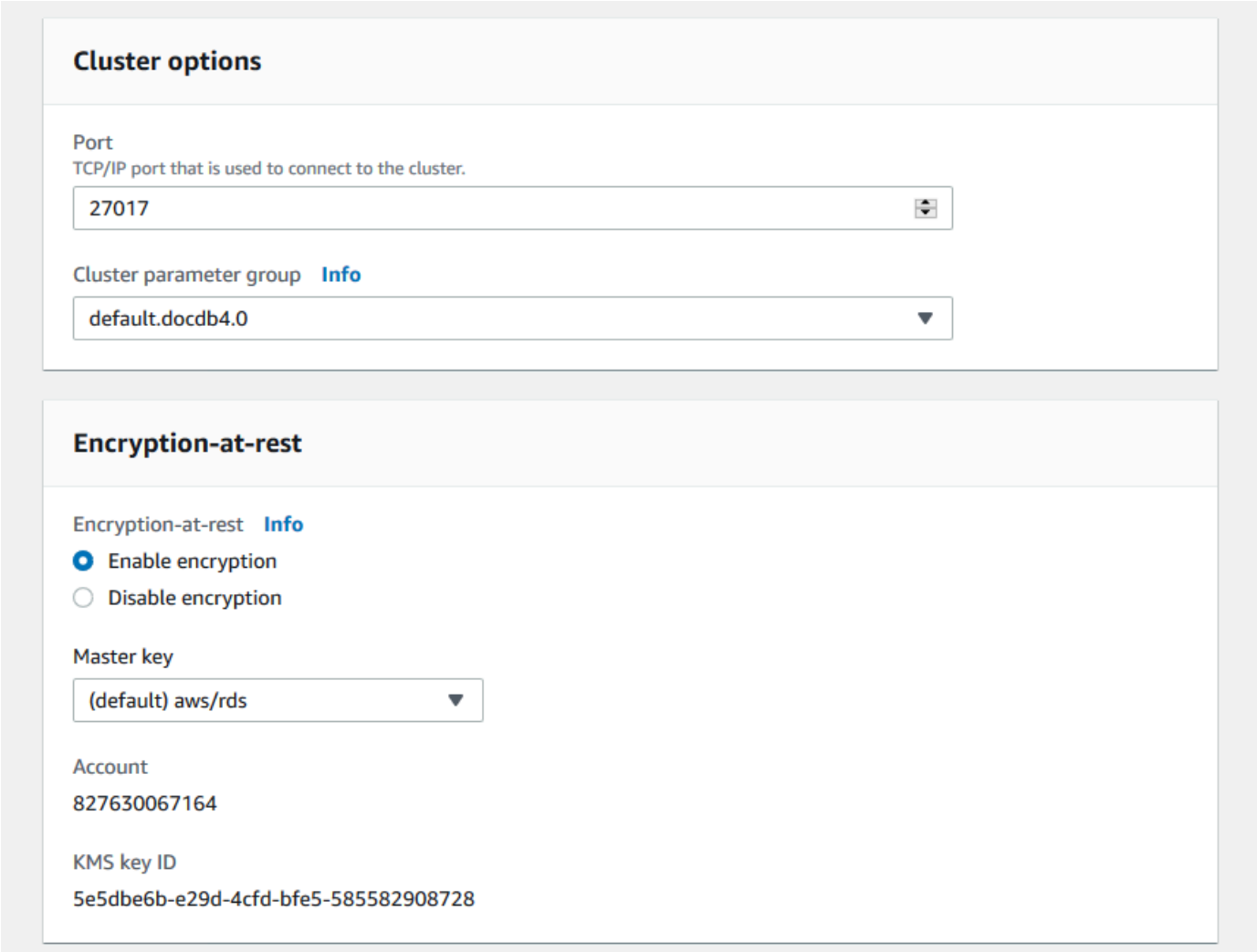
- For **VPC security groups**, **default (VPC)** should already be added.



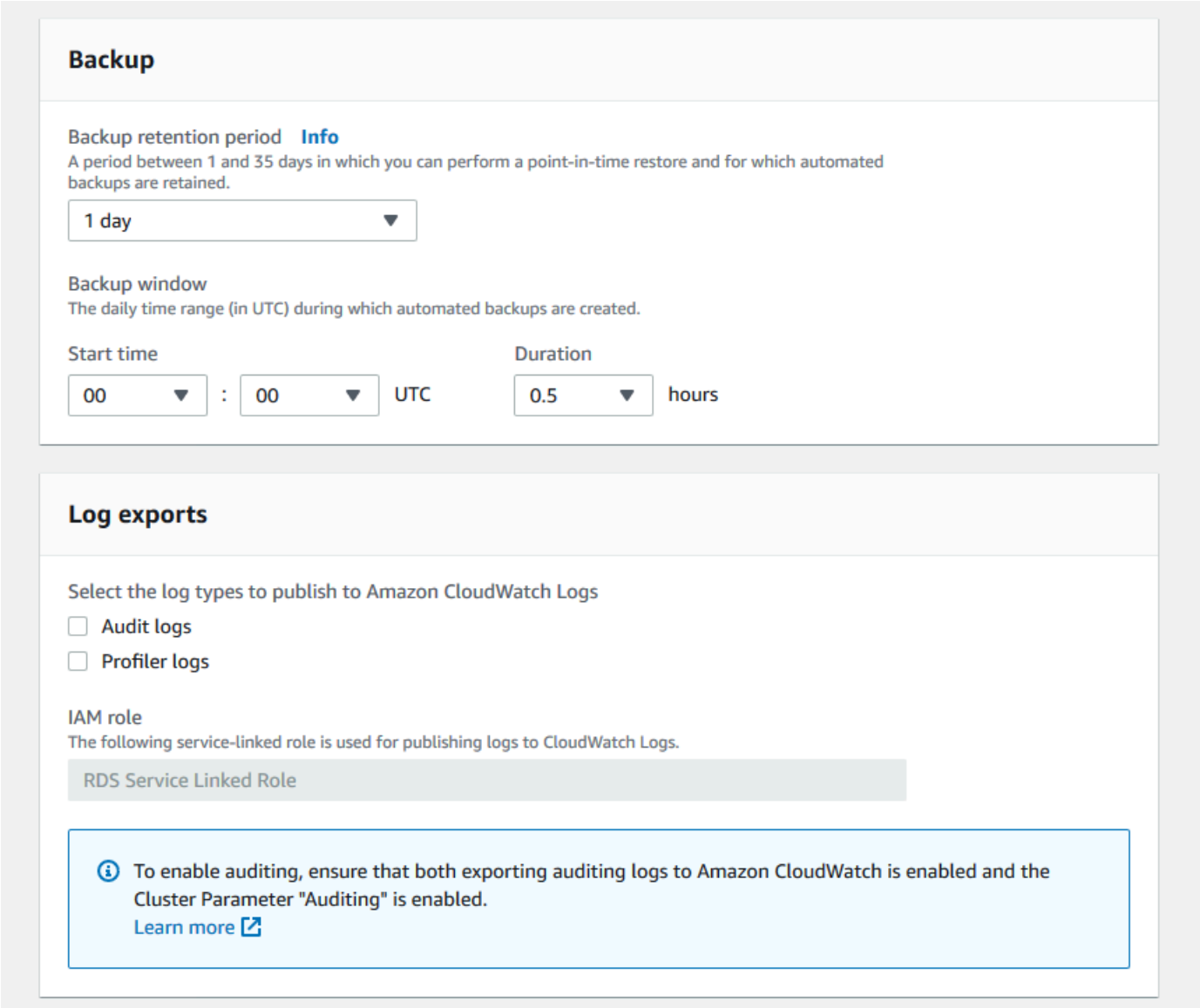
- Type DocDB into the **VPC security groups** field and select **DocDB-Inbound (VFC)**.



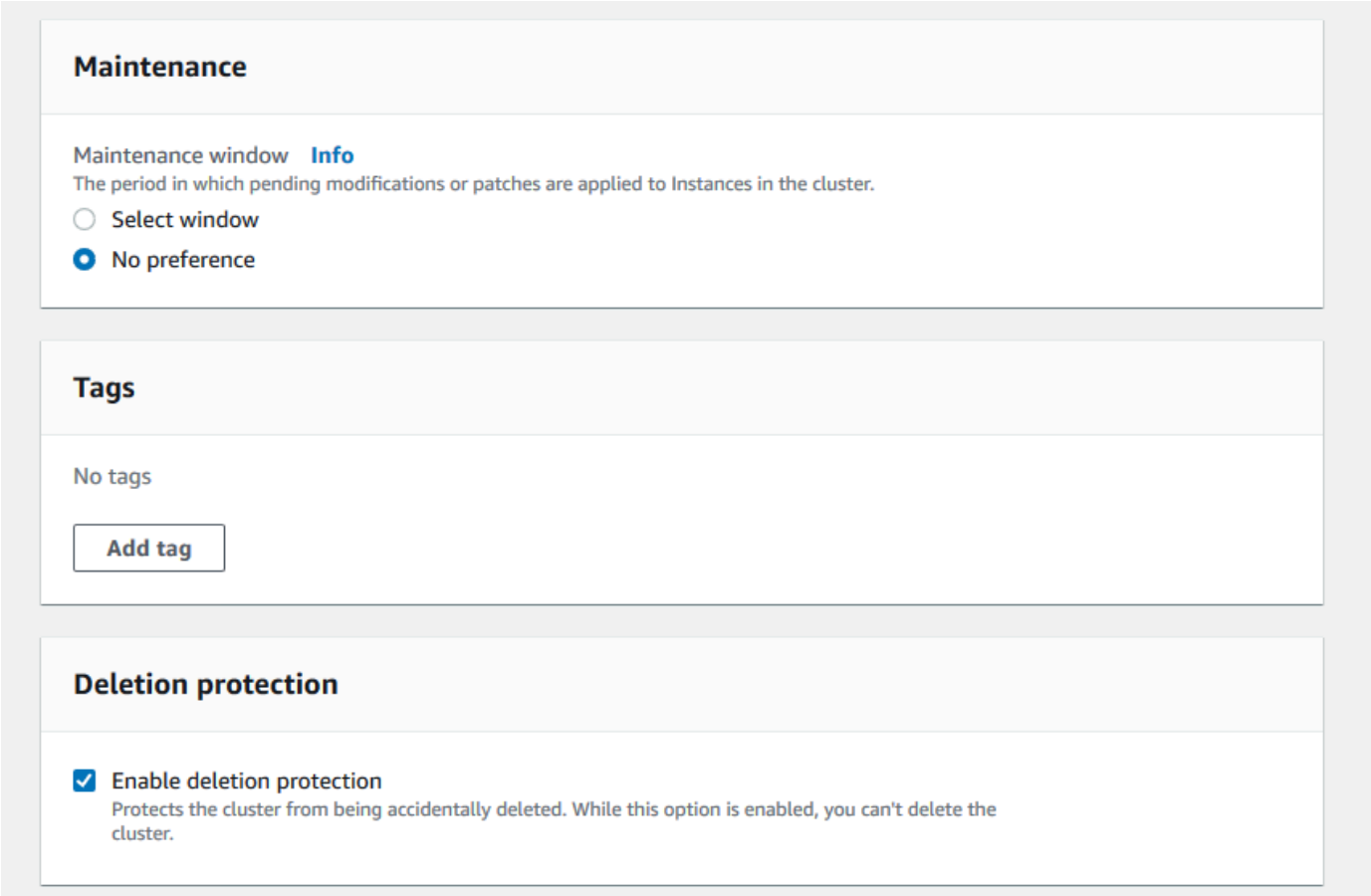
8. For **Cluster options** and **Encryption-at-rest**, leave at default selections.



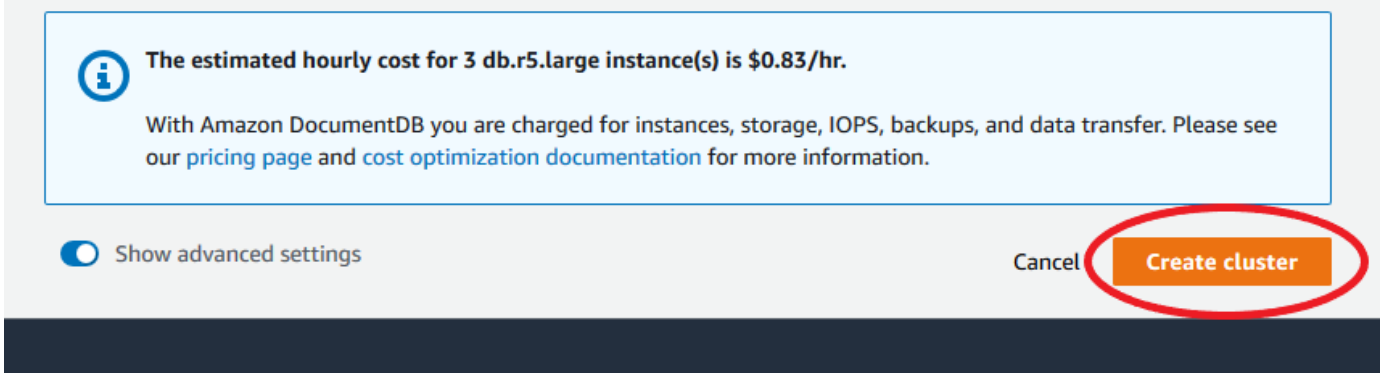
9. For **Backup** and **Log exports**, leave at default selections.



10. For **Maintenance**, **Tags**, and **Deletion protection**, leave at default selections.



11. Now click the button that says **Create cluster**.



Using the AWS CLI

To create an Amazon DocumentDB Regional cluster, call the create-global-cluster AWS CLI. The following AWS CLI command creates an Amazon DocumentDB cluster named `global-cluster-id`. For more information on deletion protection, see Deleting an Amazon DocumentDB cluster.

Also, `--engine-version` is an optional parameter that defaults to the latest major engine version. The current major engine version is `5.0.0`. When new major engine versions are released, the default engine version for `--engine-version` will be updated to reflect the last major engine version. As a result, for production workloads, and especially those that are dependent on scripting, automation, or AWS CloudFormation templates, we recommend that you explicitly specify the `--engine-version` to the intended major version.

If a `db-subnet-group-name` or `vpc-security-group-id` is not specified, Amazon DocumentDB will use the default subnet group and Amazon VPC security group for the given Region.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster \
      --global-cluster-identifier global-cluster-id \
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

For Windows:

```
aws docdb create-db-cluster ^
      --global-cluster-identifier global-cluster-id ^
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBCluster": {
        "StorageEncrypted": false,
        "DBClusterMembers": [],
        "Engine": "docdb",
        "DeletionProtection" : "enabled",
        "ClusterCreateTime": "2018-11-26T17:15:19.885Z",
        "DBSubnetGroup": "default",
        "EngineVersion": "4.0.0",
        "MasterUsername": "masteruser",
        "BackupRetentionPeriod": 1,
```

```
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:cluster-id",
        "DBClusterIdentifier": "cluster-id",
        "MultiAZ": false,
        "DBClusterParameterGroup": "default.docdb4.0",
        "PreferredBackupWindow": "09:12-09:42",
        "DbClusterResourceId": "cluster-KQSGI4MHU4NTDDRVNLNTU7XVAY",
        "PreferredMaintenanceWindow": "tue:04:17-tue:04:47",
        "Port": 27017,
        "Status": "creating",
        "ReaderEndpoint": "cluster-id.cluster-ro-sfcrlcjcoroz.us-
east-1.docdb.amazonaws.com",
        "AssociatedRoles": [],
        "HostedZoneId": "ZNKXTT8WH85VW",
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-77186e0d",
                "Status": "active"
            }
        ],
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1e"
        ],
        "Endpoint": "cluster-id.cluster-sfcrlcjcoroz.us-east-1.docdb.amazonaws.com"
    }
}
```

It takes several minutes to create the cluster. You can use the AWS Management Console or AWS CLI to monitor the status of your cluster. For more information, see Monitoring an Amazon DocumentDB cluster's status.

> **Important**
>
> When you use the AWS CLI to create an Amazon DocumentDB Regional cluster, no instances are created. Consequently, you must explicitly create a primary instance and any replica instances that you need. You can use either the console or AWS CLI to create the instances. For more information, see Adding an Amazon DocumentDB instance to a cluster and CreateDBCluster in the Amazon DocumentDB API Reference.

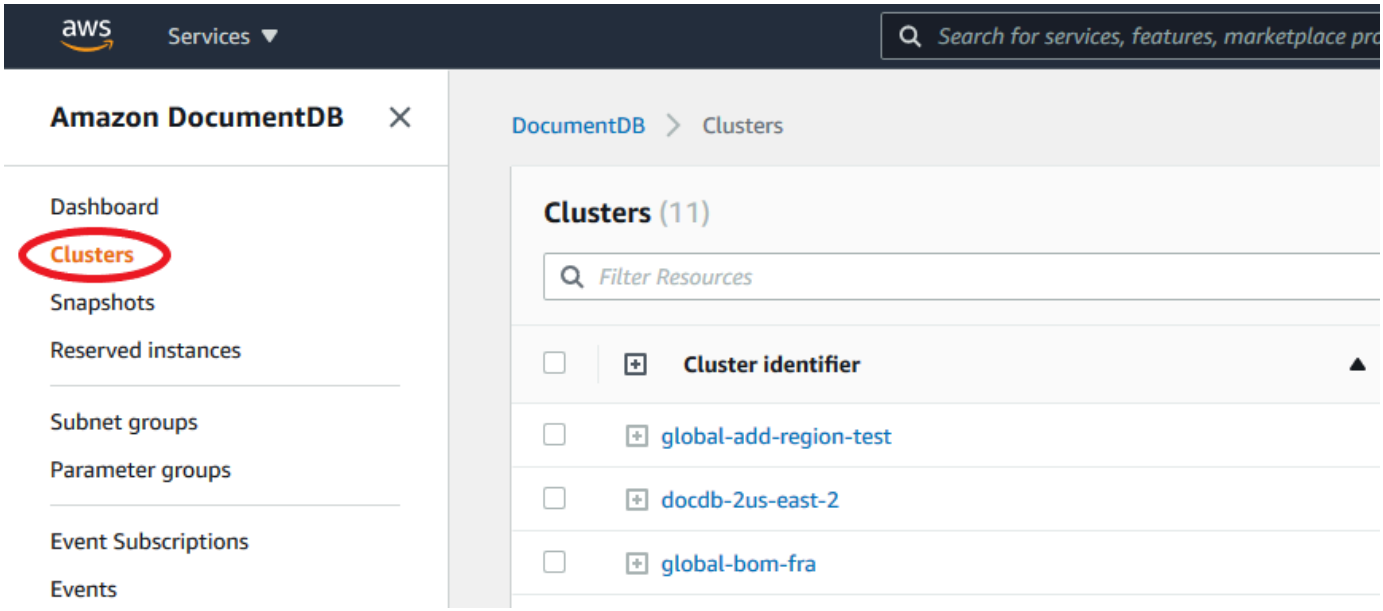
Once your Regional cluster is available, you can add a secondary cluster in another Region with the following instructions: Adding an AWS Region to an Amazon DocumentDB global cluster. When you add a Region, your Regional cluster becomes your primary cluster, and you have a new secondary cluster in the Region you chose.

Adding an AWS Region to an Amazon DocumentDB global cluster

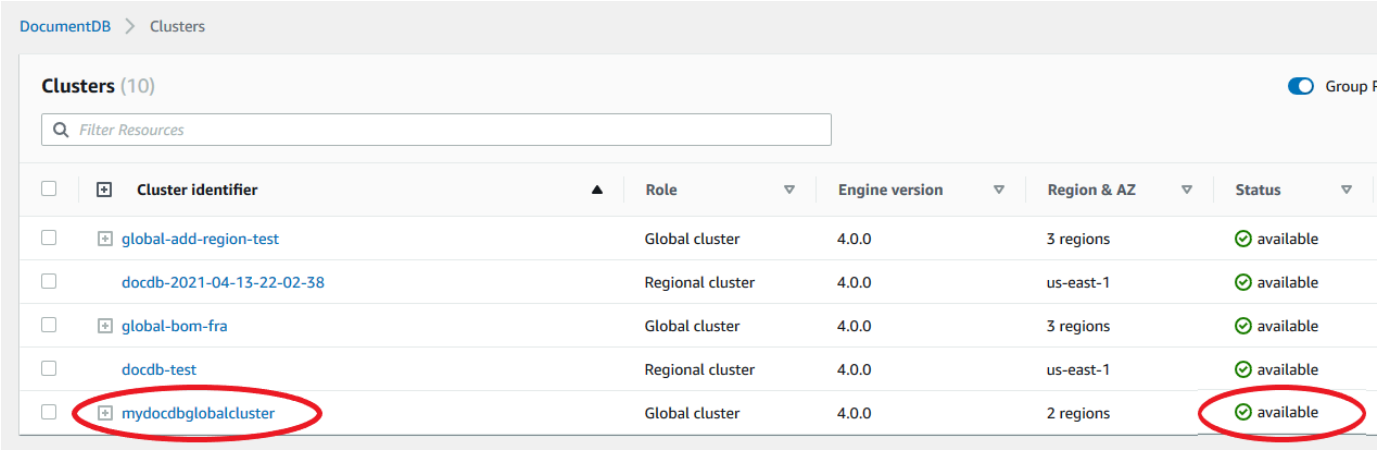
A global cluster needs at least one secondary cluster in a different Region than the primary cluster, and you can add up to five secondary clusters. Note that for each secondary cluster that you add, you must reduce the number of replicas allowed in the primary cluster by one. For example, if your global cluster has five secondary Regions, your primary cluster can have only 10 (rather than 15) replicas. For more information, see Configuration requirements of an Amazon DocumentDB global cluster.

Using the AWS Management Console

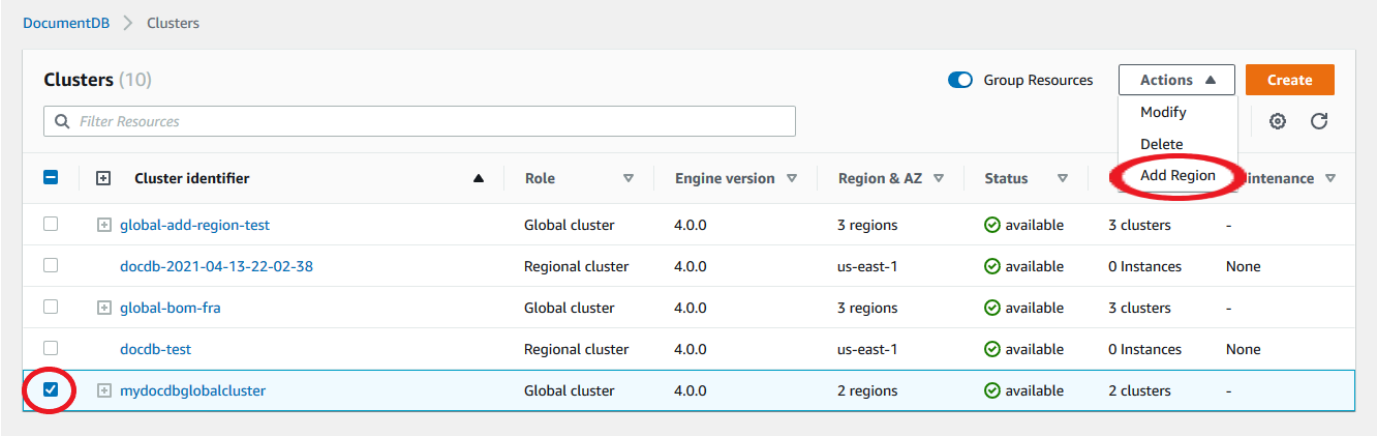
1. Sign in to the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Clusters**.



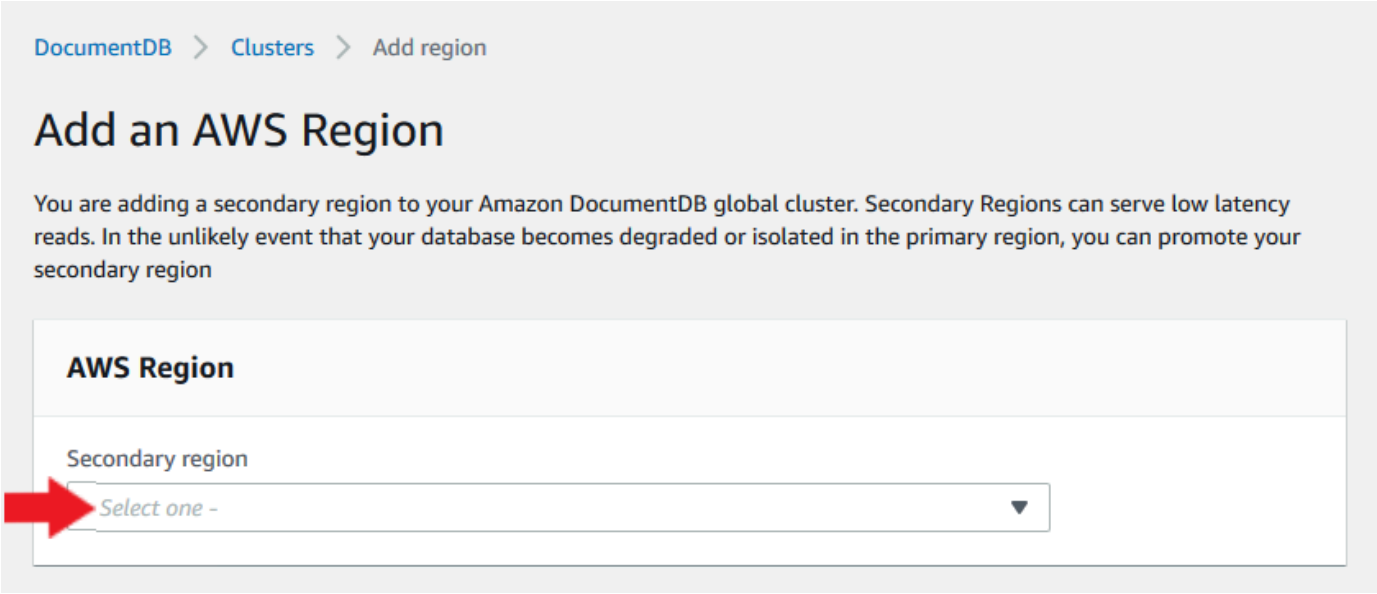
3. Choose the cluster that you would like to add a secondary cluster to. Ensure that the cluster is `Available`.



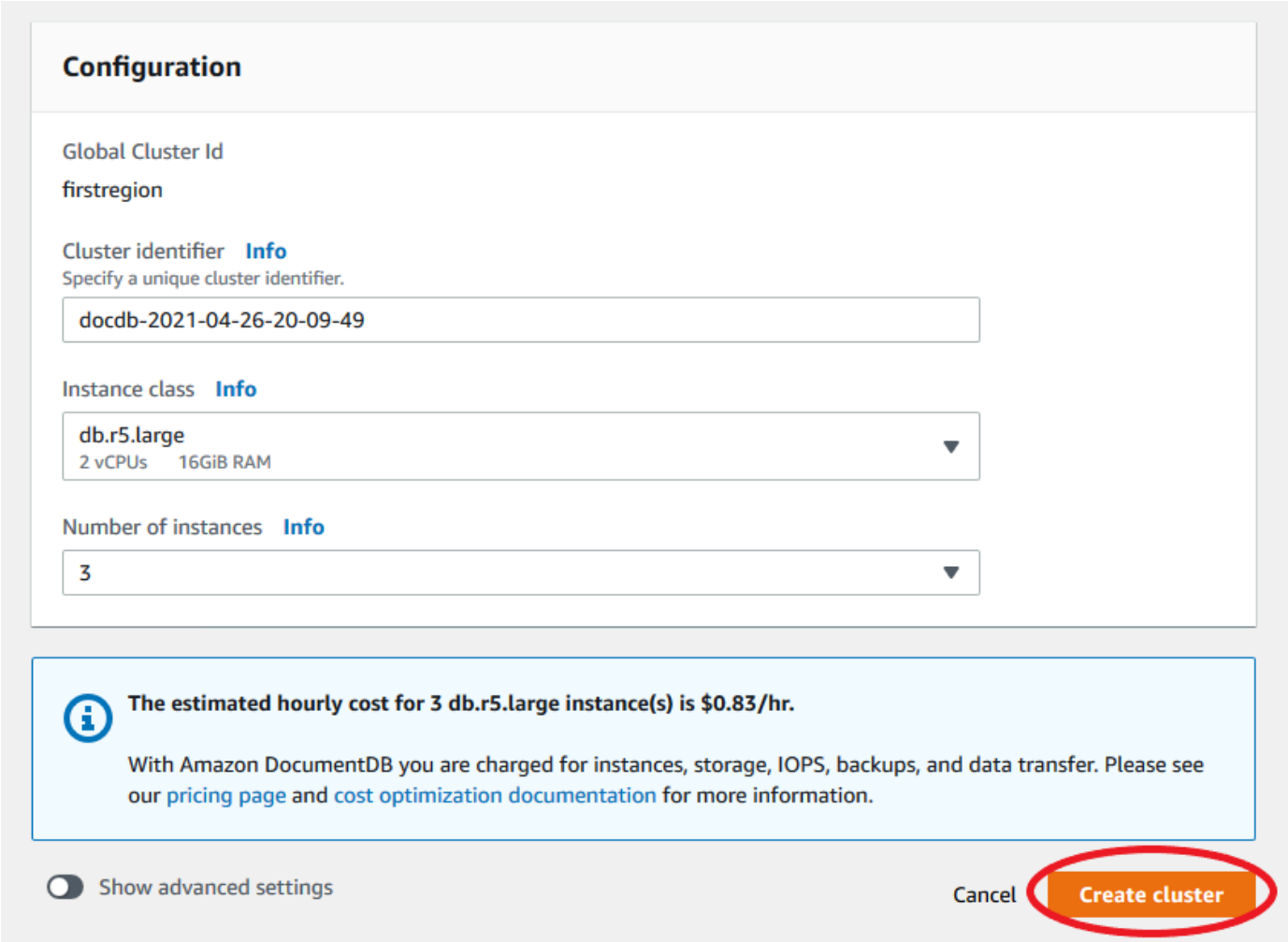
4. Select the dropdown list for **Actions** and then choose **Add Region**.



5. On the **Add an AWS Region** page, choose the secondary Region. Note that you can't choose a Region that already has a secondary cluster for the same global cluster. Also, it can't be the same Region as the primary cluster. If this is the first Region you are adding, you will also have to specify a global cluster identifier of your choice.



6. Complete the remaining fields for the secondary cluster in the new Region, then select **Create cluster**. After you finish adding the Region, you can see it in the list of **Clusters** in the AWS Management Console.



Using the AWS CLI

- Use the `create-db-cluster` CLI command with the name (`--global-cluster-identifier`) of your global cluster. For other parameters, do the following:
  - For `--region`, choose a different AWS Region than that of your primary Region.
  - Choose specific values for the `--engine` and `--engine-version` parameters.
  - For an encrypted cluster, specify your primary AWS Region as the `--source-region` for encryption.

The following example creates a new Amazon DocumentDB cluster and attaches it to the global cluster as a read-only secondary cluster. In the last step, the instance is added to the new cluster.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb --region secondary-region-id \
  create-db-cluster \
    --db-cluster-identifier cluster-id \
    --global-cluster-identifier global-cluster-id \
    --engine-version version \
    --engine docdb

aws docdb --region secondary-region-id \
  create-db-instance \
    --db-cluster-identifier cluster-id \
    --global-cluster-identifier global-cluster-id \
    --engine-version version \
    --engine docdb
```

For Windows:

```
aws docdb --region secondary-region-id ^
  create-db-cluster ^
    --db-cluster-identifier cluster-id ^
    --global-cluster-identifier global-cluster-id ^
    --engine-version version ^
    --engine docdb

aws docdb --region secondary-region-id ^
  create-db-instance ^
    --db-cluster-identifier cluster-id ^
    --global-cluster-identifier global-cluster-id ^
    --engine-version version ^
    --engine docdb
```

Using a snapshot for your Amazon DocumentDB global cluster

You can restore a snapshot of an Amazon DocumentDB cluster to use as the starting point for your global cluster. To do this, you must restore the snapshot and create a new cluster. This will serve as the primary cluster of your global cluster. You can then add another Region to the restored cluster, thus converting it into a global cluster.

Managing an Amazon DocumentDB global cluster

You perform most management operations on the individual clusters that make up a global cluster. When you choose **Group related resources** on the **Clusters** page in the console, you see the primary cluster and secondary clusters grouped under the associated global cluster.

The **Configuration** tab for a global cluster shows the AWS Regions where the clusters are running, the version, and the global cluster identifier.

Topics

- Modifying an Amazon DocumentDB global cluster
- Modifying parameters an Amazon DocumentDB global cluster
- Removing a cluster from an Amazon DocumentDB global cluster
- Deleting a cluster from an Amazon DocumentDB global cluster
- Creating a headless Amazon DocumentDB cluster in a secondary region

Modifying an Amazon DocumentDB global cluster

The **Clusters** page in the AWS Management Console lists all your global clusters, showing the primary cluster and secondary clusters for each one. The global cluster has its own configuration settings. Specifically, it has regions associated with its primary and secondary clusters.

When you make changes to the global cluster, you have a chance to cancel changes.

When you choose Continue, you confirm the changes.

Modifying parameters an Amazon DocumentDB global cluster

You can configure the cluster parameter groups independently for each cluster within the global cluster. Most parameters work the same as for other kinds of Amazon DocumentDB clusters. We recommend that you keep settings consistent among all the clusters in a global database. Doing this helps to avoid unexpected behavior changes if you promote a secondary cluster to be the primary.

For example, use the same settings for time zones and character sets to avoid inconsistent behavior if a different cluster takes over as the primary cluster.

Removing a cluster from an Amazon DocumentDB global cluster

There are several situations when you may want to remove clusters from your global cluster. For example, you might want to remove a cluster from a global cluster if the primary cluster becomes degraded or isolated. It then becomes a standalone provisioned cluster that could be used to create a new global cluster. To learn more, see Performing a manual failover for an Amazon DocumentDB global cluster.

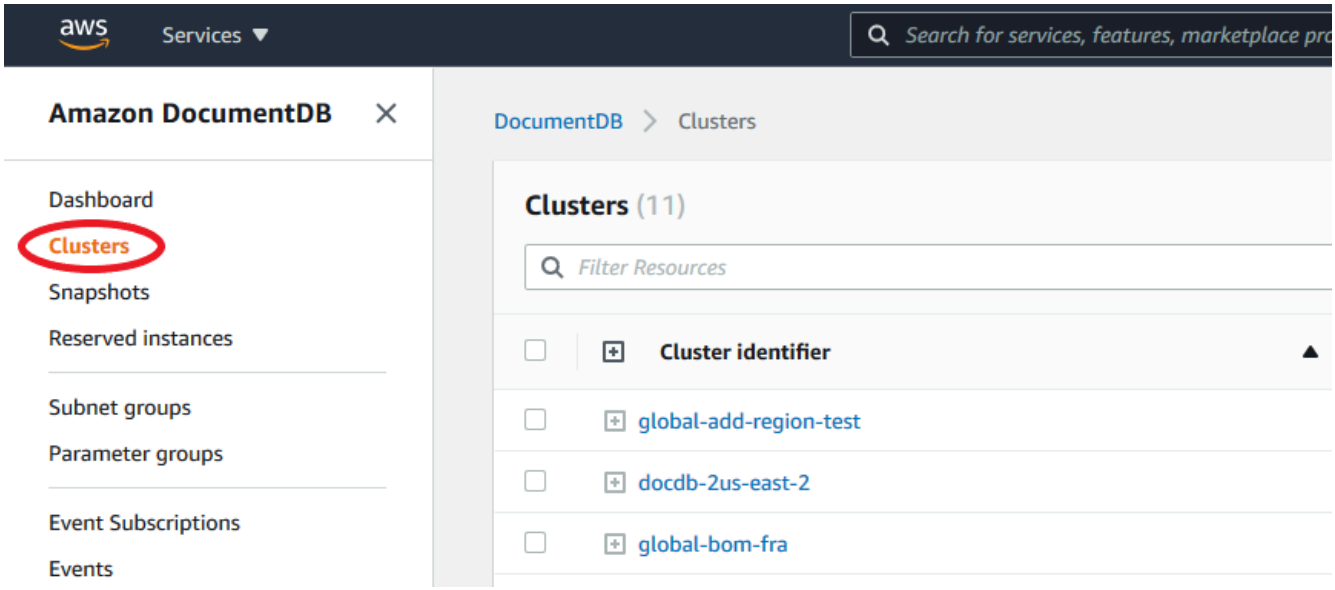
You also might want to remove clusters because you want to delete a global cluster that you no longer need. You can't delete the global cluster until after you detach all associated clusters, leaving the primary for last. For more information, see Deleting a cluster from an Amazon DocumentDB global cluster.

> **Note**
>
> When a cluster is detached from the global cluster, it's no longer synchronized with the primary. It becomes a standalone provisioned cluster with full read/write capabilities. Additionally, it is no longer visible in the Amazon DocumentDB console. It is only visible when you select the region in the console that the cluster was located in.

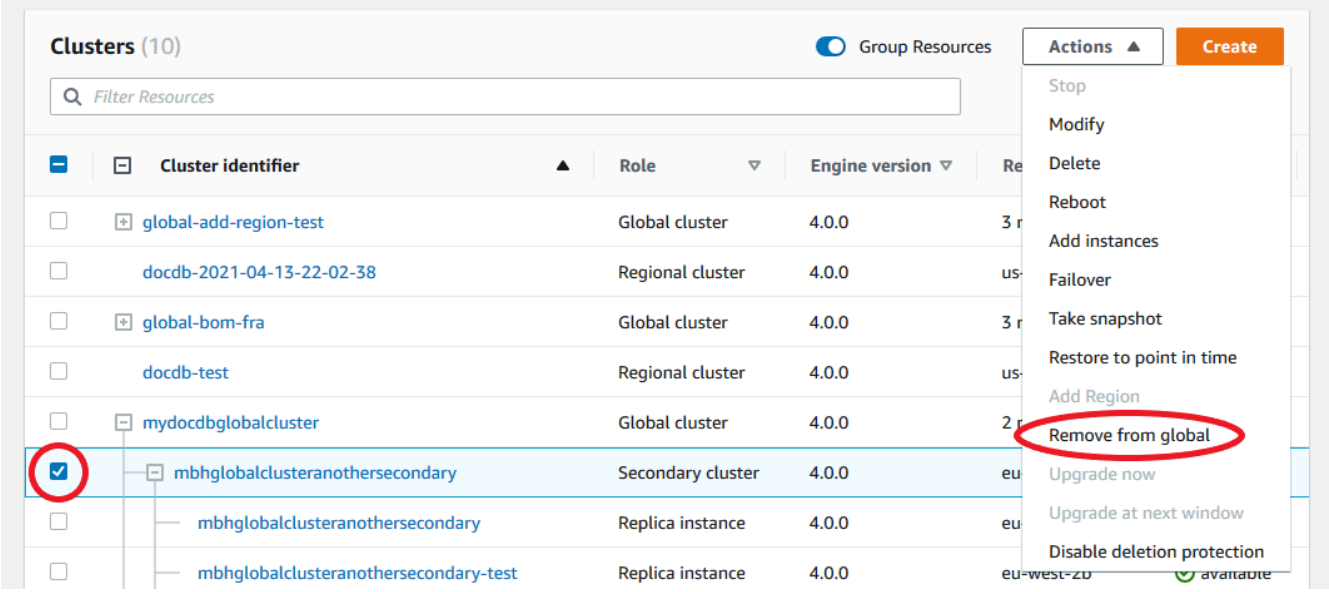
You can remove clusters from your global cluster using the AWS Management Console, the AWS CLI, or the RDS API.

Using the AWS Management Console

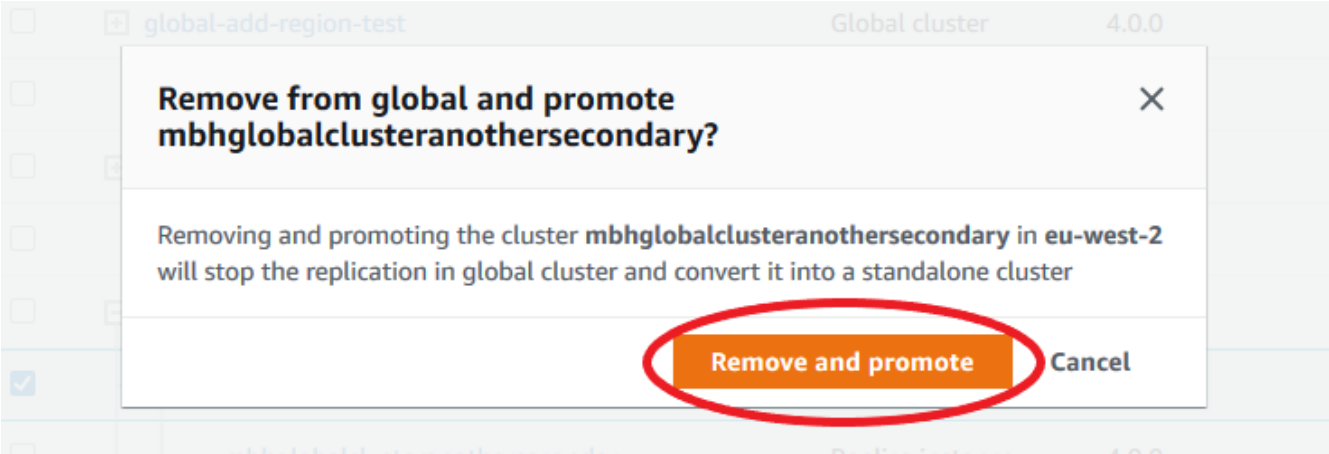
1. Sign in to the AWS Management Console and navigate to the Amazon DocumentDB console.
2. Choose **Clusters** on the left side navigation.



3. Expand the global cluster so you can see all the secondary clusters. Select the secondary clusters you wish to remove. Choose **Actions**, and in the menu that drops down, choose **Remove from Global**.



4. A prompt will appear, asking you to confirm that you want to detach the secondary from the global cluster. Choose **Remove and promote** to remove the cluster from the global cluster.



Now that cluster is no longer serving as a secondary and no longer synchronized with the primary cluster. It is a standalone cluster with full read/write capability.

After you remove or delete all secondary clusters, then you can remove the primary cluster the same way. You can't detach or remove the primary cluster from the global cluster until after you have removed all secondary clusters. The global cluster might remain in the Clusters list, with zero regions and AZs. You can delete if you no longer want to use this global cluster.

Using the AWS CLI

To remove a cluster from a global cluster, run the `remove-from-global-cluster` CLI command with the following parameters:

- `--global-cluster-identifier` — The name (identifier) of your global cluster.
- `--db-cluster-identifier` — The name of each cluster to remove from the global cluster.

The following examples first remove a secondary cluster and then the primary cluster from a global cluster.

For Linux, macOS, or Unix:

```
aws docdb --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_cluster_id

aws docdb --region primary_region \
  remove-from-global-cluster \
```

```
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_cluster_id
```

Repeat the `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` command for each secondary region in your global cluster.

For Windows:

```
aws docdb --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_cluster_id

aws docdb --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_cluster_id
```

Repeat the `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` command for each secondary region in your global cluster.

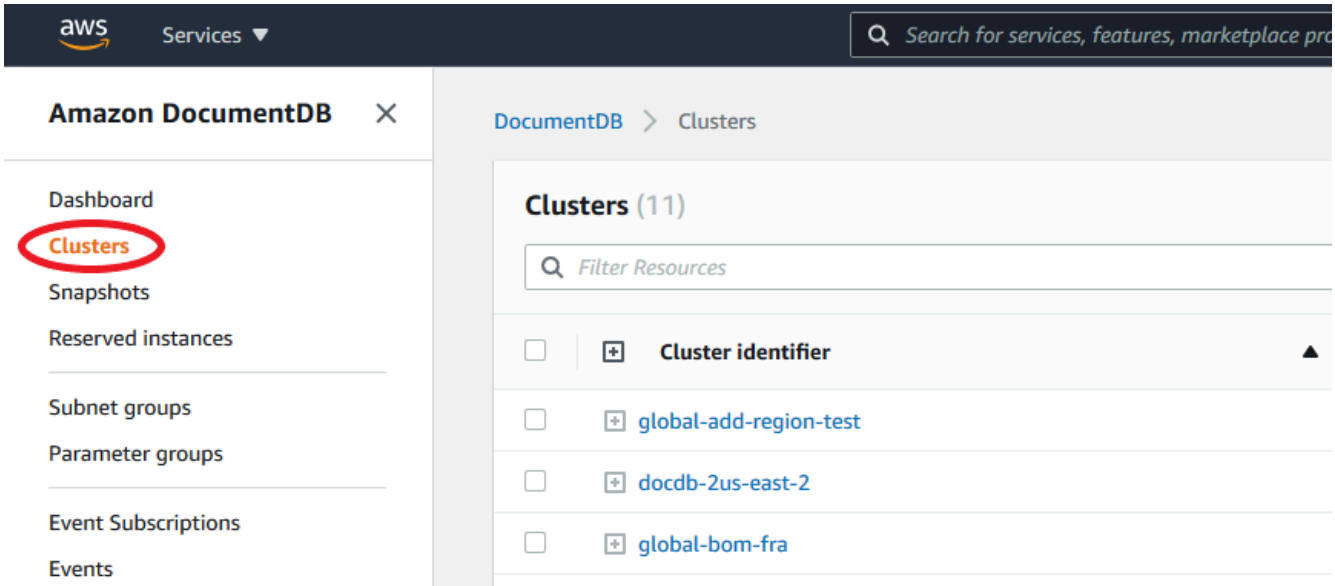
Deleting a cluster from an Amazon DocumentDB global cluster

To delete a global cluster, do the following:

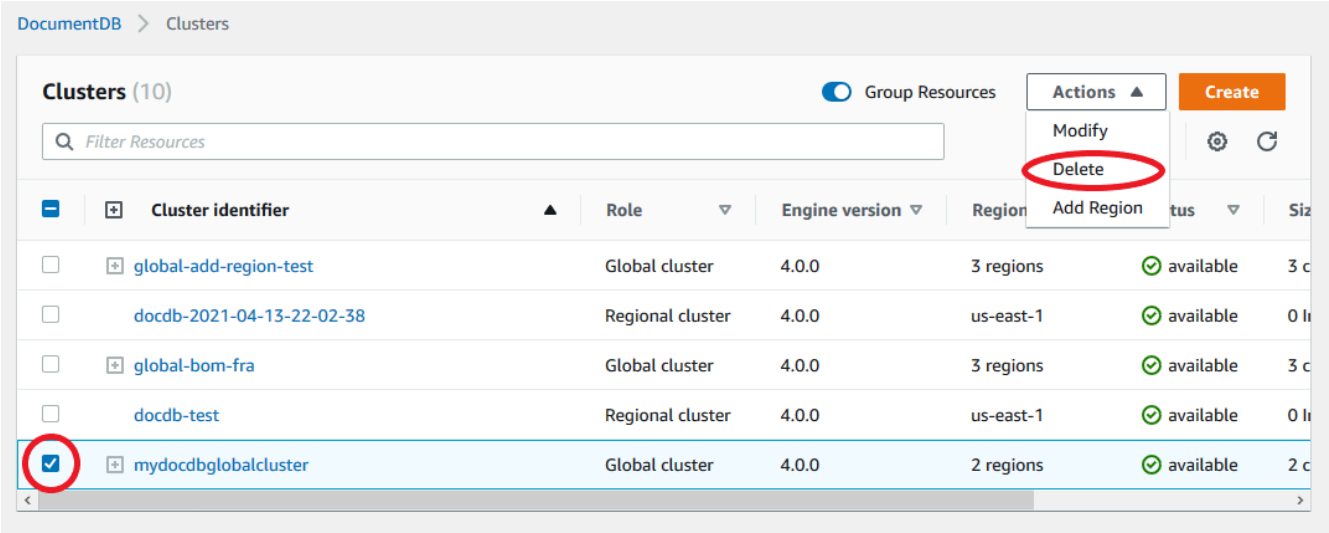
- Remove all secondary clusters from the global cluster. Each cluster becomes a standalone cluster. See the previous section, Removing a cluster from an Amazon DocumentDB global cluster.
- From each standalone cluster, delete all replicas.
- Remove the primary cluster from the global cluster. This becomes a standalone cluster.
- From the primary cluster, first delete all replicas, then delete the primary instance. Deleting the primary instance from the newly standalone cluster also typically removes both the cluster and the global cluster.

Using the AWS Management Console

1. Sign in to the AWS Management Console and navigate to the Amazon DocumentDB console.
2. Choose **Clusters** and find the global cluster you want to delete.



3. With your global cluster selected, choose **Delete** from the **Actions** menu.



Confirm that all clusters are removed from the global cluster. The global cluster should show zero regions and AZs and a size of zero clusters. If the global cluster contains any clusters, you can't delete it yet. You'll first have to follow the instructions in the previous step, **Removing a cluster from an Amazon DocumentDB global cluster**.

Using the AWS CLI

To delete a global cluster, run the `delete-global-cluster` CLI command with the name of the AWS Region and the global cluster identifier, as shown in the following example.

For Linux, macOS, or Unix:

```
aws docdb --region primary_region delete-global-cluster \
   --global-cluster-identifier global_cluster_id
```

For Windows:

```
aws docdb --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_cluster_id
```

Creating a headless Amazon DocumentDB cluster in a secondary region

Although an Amazon DocumentDB global cluster requires at least one secondary cluster in a different AWS Region than the primary, you can use a headless configuration for the secondary cluster. A headless secondary Amazon DocumentDB cluster is one without an instance. This type of configuration can lower expenses for a global cluster. In an Amazon DocumentDB cluster, compute and storage are decoupled. Without the instance, you're not charged for compute, only for storage. If it's set up correctly, a headless secondary's storage volume is kept in sync with the primary cluster.

You add the secondary cluster as you normally do when creating an Amazon DocumentDB global cluster. However, after the primary cluster begins replication to the secondary, you delete the read-only instance from the secondary cluster. This secondary cluster is now considered "headless" because it no longer has a Instance. Yet, the storage volume is kept in sync with the primary Amazon DocumentDB cluster.

> **Important**
>
> We only recommend headless clusters for customers who can tolerate region-wide failures for 15+ minutes. This is because recovering from a region-wide failure with a headless secondary cluster will require the user to create a new instance after failing over. A new instance can take ~10-15 minutes to become available.

How to Add a Headless Secondary Cluster to Your Global Cluster

1. Sign in to the AWS Management Console and open the Amazon DocumentDB console.
2. Choose **Clusters** on the left side navigation.

3. Choose the global cluster that needs a secondary cluster. Ensure that the primary cluster is `Available`.
4. For **Actions**, choose **Add region**.
5. On the **Add a region** page, choose the secondary region.

> **Note**
>
> You can't choose a region that already has a secondary cluster for the same global cluster. Also, it can't be the same region as the primary cluster.

6. Complete the remaining fields for the secondary cluster in the new region. These are the same configuration options as for any cluster instance.
7. Add a region. After you finish adding the region to your global cluster, you will see it in the list of `Clusters` in the AWS Management Console.
8. Check the status of the secondary cluster and its reader instance before continuing, by using the AWS Management Console or the AWS CLI. Here is a sample command if you use the AWS CLI:

```
$ aws docdb describe-db-clusters --db-cluster-identifier secondary-cluster-id --
query '*[].[Status]' --output text
```

   It can take several minutes for the status of a newly added secondary cluster to change from creating to available. When the cluster is available, you can delete the reader instance.

9. Select the reader instance in the secondary cluster, and then choose **Delete**.
10. After deleting the reader instance, the secondary cluster remains part of the global cluster. It should have no instance associated with it.

> **Note**
>
> You can use this headless secondary Amazon DocumentDB cluster to manually recover your Amazon DocumentDB global cluster from an unplanned outage in the primary region if such an outage occurs.

Connect to an Amazon DocumentDB global cluster

How you connect to a global cluster depends on whether you need to write to the cluster or read from the cluster:

- For read-only requests or queries, you connect to the reader endpoint for the cluster in your AWS Region.
- To run data manipulation language (DML) or data definition language (DDL) statements, you connect to the cluster endpoint for the primary cluster. This endpoint might be in a different AWS Region than your application.

When you view a global cluster in the console, you can see all the general-purpose endpoints associated with all of its clusters.

How you connect to a global cluster depends on whether you need to write to the database or read from the database. For DDL, DML and read operations that you would like to serve from the primary region, you should connect to your primary cluster. We recommend that you connect to your primary cluster using the cluster endpoint in replica set mode, with a read preference of `secondaryPreferred=true`. This will route write traffic to your primary cluster's writer instance and read traffic to your primary cluster's replica instance.

For cross region, read only traffic, you should connect to one of your secondary clusters. We recommend that you connect to your secondary cluster using the cluster endpoint in replica set mode. Since all instances are read-only replica instances, you do not need to specify a read preference. To minimize latency, choose whichever reader endpoint is in your region or the region closest to you.

Monitoring Amazon DocumentDB global clusters

Amazon DocumentDB (with MongoDB compatibility) integrates with CloudWatch so that you can gather and analyze operational metrics for your clusters. You can monitor these metrics using the CloudWatch console, the Amazon DocumentDB console, the AWS Command Line Interface (AWS CLI), or the CloudWatch API.

To monitor a global cluster, use the following CloudWatch metrics.

Metric	Description
`GlobalClusterReplicatedWriteIO`	The average number of billed write I/O operations replicated from the cluster volume in the primary AWS Region to the cluster volume in a secondary AWS Region, reported at 5-minute intervals. The number of replicated `ReplicatedWriteIOs` to each secondary region is the same as the number of in-region `VolumeWriteIOPs` performed by the primary region.
`GlobalClusterDataTransferBytes`	The amount of data transferred from the primary cluster's AWS Region to a secondary cluster's AWS Region, measure in bytes.
`GlobalClusterReplicationLag`	The amount of lag, in milliseconds, when replicating change events from the primary cluster's AWS Region to a secondary cluster's AWS Region

For more information on how to view these metrics, please see Viewing CloudWatch data.

Disaster recovery and Amazon DocumentDB global clusters

Topics

- Performing a managed failover for an Amazon DocumentDB global cluster
- Performing a manual failover for an Amazon DocumentDB global cluster
- Performing a switchover for an Amazon DocumentDB global cluster
- Unblocking a global cluster switchover or failover

By using a global cluster, you can recover from disasters such as region failures quickly. Recovery from disaster is typically measured using values for RTO and RPO.

- **Recovery time objective (RTO)** — The time it takes a system to return to a working state after a disaster. In other words, RTO measures downtime. For a global cluster, RTO in minutes.

- **Recovery point objective (RPO)** — The amount of data that can be lost (measured in time). For a global cluster, RPO is typically measured in seconds.
- To recover from an unplanned outage, you can perform a cross-region failover to one of the secondaries in your global cluster. When your global cluster has multiple secondary regions, make sure that you detach all the secondary regions that you wish to promote as primaries. Then, you promote one of those secondary regions to be the new primary AWS Region. Finally, you create new clusters in each of the other secondary regions and attach those clusters to your global cluster.

Performing a managed failover for an Amazon DocumentDB global cluster

This approach is intended for business continuity in the event of a true Regional disaster or complete service-level outage.

During a managed failover, your primary cluster is failed over to your choice of secondary Region while your Amazon DocumentDB global cluster's existing replication topology is maintained. The chosen secondary cluster promotes one of its read-only nodes to full writer status. This step allows the cluster to assume the role of primary cluster. Your database is unavailable for a short time while this cluster is assuming its new role. Data that wasn't replicated from the old primary to the chosen secondary cluster may be missing when this secondary becomes the new primary. The old primary volume makes a best effort attempt to take a snapshot before synchronizing with the new primary so unreplicated data is preserved on the snapshot.

> **Note**
>
> You can only perform a managed cross-Region cluster failover on an Amazon DocumentDB global cluster if the primary and all secondary clusters have the same engine versions. If your engine versions are incompatible, you can perform the failover manually by following the steps in Performing a manual failover for an Amazon DocumentDB global cluster.
> If the region's engine versions do not match, the failover will be blocked. Please check for any pending upgrades and apply them to ensure all region's engine versions match and the global cluster failover is unblocked. For more information, see Unblocking a global cluster switchover or failover.

To minimize data loss, we recommend that you do the following before using this feature:

- Take applications offline to prevent writes from being sent to the primary cluster of the Amazon DocumentDB global cluster.
- Check lag times for all Amazon DocumentDB secondary clusters. Choosing the secondary Region with the least replication lag can minimize data loss with the current failed primary Region. Check lag times for all Amazon DocumentDB secondary clusters in the global cluster by viewing the `GlobalClusterReplicationLag` metric in Amazon CloudWatch. These metrics show you how far behind (in milliseconds) replication to a secondary cluster is to the primary cluster.

  For more information about CloudWatch metrics for Amazon DocumentDB, see Amazon DocumentDB metrics.

During a managed failover, the chosen secondary cluster is promoted to its new role as primary. However, it doesn't inherit the various configuration options of the primary cluster. A mismatch in configuration can lead to performance issues, workload incompatibilities, and other anomalous behavior. To avoid such issues, we recommend that you resolve differences between your Amazon DocumentDB global clusters for the following:

- **Configure an Amazon DocumentDB cluster parameter group for the new primary, if necessary** — You can configure your Amazon DocumentDB cluster parameter groups independently for each cluster in your Amazon DocumentDB global clustere. Therefore, when you promote a secondary cluster to take over the primary role, the parameter group from the secondary might be configured differently than for the primary. If so, modify the promoted secondary cluster's parameter group to conform to your primary cluster's settings. To learn how, see Modifying Amazon DocumentDB cluster parameter groups.
- **Configure monitoring tools and options, such as Amazon CloudWatch events and alarms** — Configure the promoted cluster with the same logging ability, alarms, and so on as needed for the global cluster. As with parameter groups, configuration for these features isn't inherited from the primary during the failover process. Some CloudWatch metrics, such as replication lag, are only available for secondary Regions. Thus, a failover changes how to view those metrics and set alarms on them, and could require changes to any predefined dashboards. For more information about Amazon DocumentDB clusters and monitoring, see Monitoring Amazon DocumentDB.

Typically, the chosen secondary cluster assumes the primary role within a minute. As soon as the new primary Region's writer node is available, you can connect your applications to it and resume your workloads. After Amazon DocumentDB promotes the new primary cluster, it automatically rebuilds all additional secondary Region clusters.

Because Amazon DocumentDB global clusters use asynchronous replication, the replication lag in each secondary Region can vary. Amazon DocumentDB rebuilds these secondary Regions to have the exact same point-in-time data as the new primary Region cluster. The duration of the complete rebuilding task can take a few minutes to several hours, depending on the size of the storage volume and the distance between the Regions. When the secondary Region clusters finish rebuilding from the new primary Region, they become available for read access. As soon as the new primary writer is promoted and available, the new primary Region's cluster can handle read and write operations for the Amazon DocumentDB global cluster.

To restore the global cluster's original topology, Amazon DocumentDB monitors the availability of the old primary Region. As soon as that Region is healthy and available again, Amazon DocumentDB automatically adds it back to the global cluster as a secondary Region. Before creating the new storage volume in the old primary Region, Amazon DocumentDB tries to take a snapshot of the old storage volume at the point of failure. It does this so that you can use it to recover any of the missing data. If this operation is successful, Amazon DocumentDB places this snapshot named "rds:docdb-unplanned-global-failover-name-of-old-primary-DB-cluster-timestamp" in the snapshot section of the AWS Management Console. You can also see this snapshot listed in the information returned by the `DescribeDBClusterSnapshots` API operation.

> **Note**
>
> The snapshot of the old storage volume is a system snapshot that's subject to the backup retention period configured on the old primary cluster. To preserve this snapshot outside of the retention period, you can copy it to save it as a manual snapshot. To learn more about copying snapshots, including pricing, see Copying a cluster snapshot.

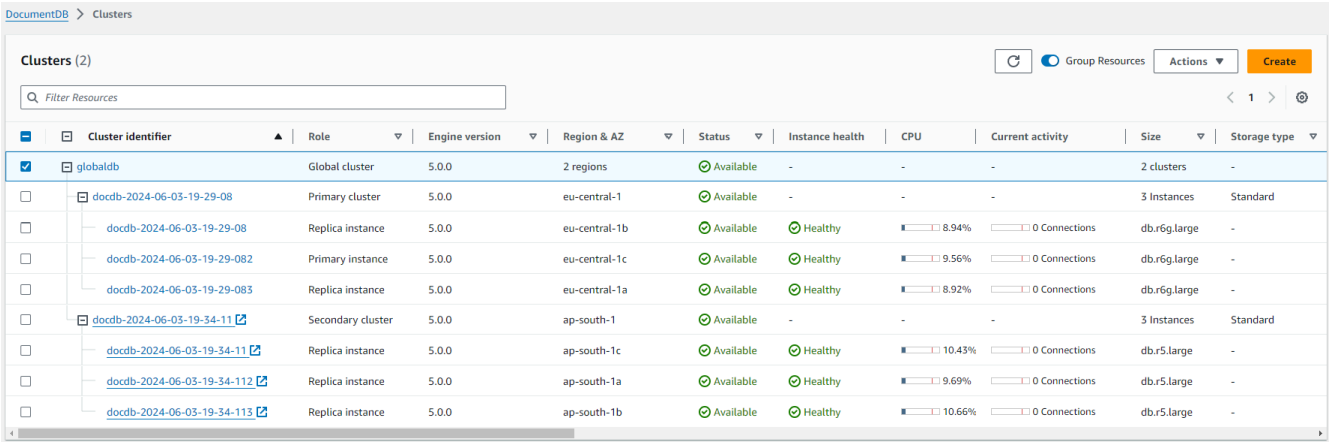
After the original topology is restored, you can fail back your global cluster to the original primary Region by performing a switchover operation when it makes the most sense for your business and workload. To do so, follow the steps in Performing a switchover for an Amazon DocumentDB global cluster.

You can fail over your Amazon DocumentDB global cluster using the AWS Management Console, the AWS CLI, or the Amazon DocumentDB API.

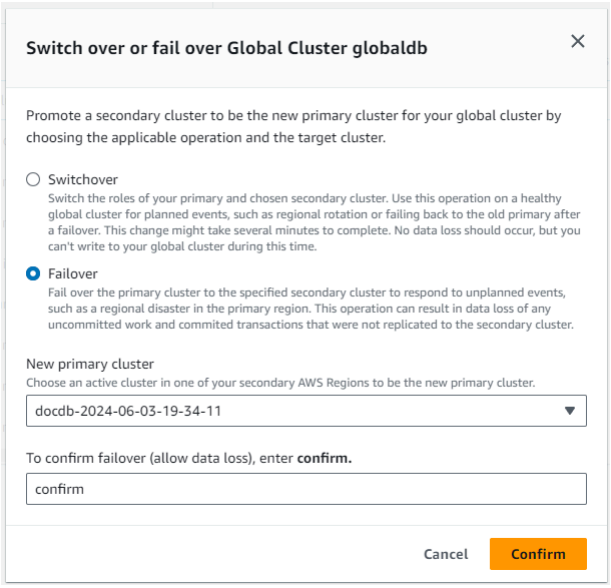
Using the AWS Management Console

Perform a managed failover on your Amazon DocumentDB global cluster

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. Find and choose the Amazon DocumentDB global cluster you want to fail over.



4. Choose **Switchover or Failover** from the **Actions** menu.
5. On the dialog box that appears, choose **Failover**, then choose the secondary cluster from the **New primary cluster** field drop down list.

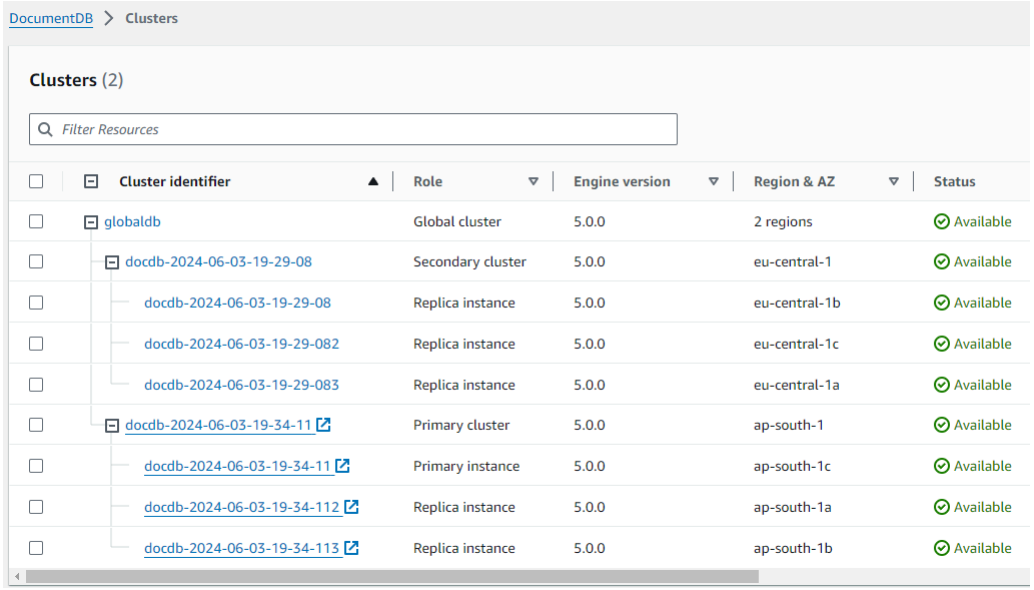


6. Type "confirm" in the last field. Then choose **Confirm**.

   The status of the primary cluster changes to "**Failing-over**". This condition should take approximately one minute. During this time, the status of the new primary cluster shows "**Modifying...**". Once the new primary is promoted, it will show "**Available**" and will be able to serve read and write transactions. The secondary regions including the old primary will

show "**Resyncing...**" while it resynchronizes to the new primary. Similar to the new primary, it will only be able to serve transaction once the status changes to "**Available**".

7. When complete, the original primary cluster becomes the secondary cluster. The selected secondary cluster becomes the primary cluster.



Using the AWS CLI

Perform a managed failover on your Amazon DocumentDB global cluster

Run the `failover-global-cluster` CLI command to fail over your Amazon DocumentDB global cluster. With the command, pass values for the following options:

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`
- `--allow-data-loss`

In the following examples, replace each *`user input placeholder`* with your cluster's information.

For Linux, macOS, or Unix:

```
aws docdb failover-global-cluster \
   --region region_of_selected_secondary \
```

```
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

For Windows:

```
aws docdb failover-global-cluster ^
   --region region_of_selected_secondary ^
   --global-cluster-identifier global_cluster_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

Performing a manual failover for an Amazon DocumentDB global cluster

If an entire cluster in one AWS Region becomes unavailable, you can promote another cluster in the global cluster to have read/write capability.

You can manually activate the global cluster failover mechanism if a cluster in a different AWS Region is a better choice to be the primary cluster. For example, you might increase the capacity of one of the secondary clusters and then promote it to be the primary cluster. Or the balance of activity among the AWS Regions might change, so that switching the primary cluster to a different AWS Region might give lower latency for write operations.

The following procedure outlines what to do to promote one of the secondary clusters in an Amazon DocumentDB global cluster.

To promote a secondary cluster:

1. Stop issuing DML statements and other write operations to the primary cluster in the AWS Region with the outage.
2. Identify a cluster from a secondary AWS Region to use as a new primary cluster. If you have two (or more) secondary AWS Regions in your global cluster, choose the secondary cluster that has the least lag time.
3. Detach your chosen secondary cluster from the global cluster.

   Removing a secondary cluster from a global cluster immediately stops the replication from the primary to this secondary and promotes it to standalone provisioned cluster with full read/write capabilities. Any other secondary cluster associated with the primary cluster in the

region with the outage are still available and can accept calls from your application. They also consume resources. Since you are recreating the global cluster, to avoid split-brain and other issues, remove the other secondary clusters before creating the new global cluster in the steps that follow.

For detailed steps for detaching, see Removing a cluster from an Amazon DocumentDB global cluster.

4. This cluster becomes the primary cluster of a new global cluster when you start adding Regions to it, in the next step.
5. Add an AWS Region to the cluster. When you do this, the replication process from primary to secondary begins.
6. Add more AWS Regions as needed to re-create the topology needed to support your application. Make sure that application writes are sent to the correct cluster before, during, and after making changes such as these, to avoid data inconsistencies among the clusters in the global cluster (split-brain issues).
7. When the outage is resolved and you're ready to assign your original AWS Region as the primary cluster again, perform the same steps in reverse.
8. Remove one of the secondary clusters from the global cluster. This will enable it to serve read/write traffic.
9. Redirect all the write traffic to the primary cluster in the original AWS Region.
10. Add an AWS Region to set up one or more secondary clusters in the same AWS Region as before.

Amazon DocumentDB global clusters can be managed using AWS SDKs, enabling you to create solutions to automate global cluster failover process for Disaster Recovery and Business Continuity Planning use cases. One such solution is made available for our customers under Apache 2.0 licensing and can be accessed from our tools repository here. This solution leverages Amazon Route 53 for endpoint management and provides AWS Lambda functions that can be triggered based appropriate events.

Performing a switchover for an Amazon DocumentDB global cluster

By using switchovers, you can change the Region of your primary cluster on a routine basis. This approach is intended for controlled scenarios, such as operational maintenance and other planned operational procedures.

There are three common use cases for using switchovers:

- For "regional rotation" requirements imposed on specific industries. For example, financial service regulations might want tier-0 systems to switch to a different Region for several months to ensure that disaster recovery procedures are regularly exercised.
- For multi-Region "follow-the-sun" applications. For example, a business might want to provide lower latency writes in different Regions based on business hours across different time zones.
- As a zero-data-loss method to fail back to the original primary Region after a failover.

> **Note**
>
> Switchovers are designed to be used on a healthy Amazon DocumentDB global cluster. To recover from an unplanned outage, follow the appropriate procedure in Performing a manual failover for an Amazon DocumentDB global cluster.
> To perform a switchover, all secondary regions must be running the exact same engine version as the primary. If the region's engine versions do not match, the switchover will be blocked. Please check for any pending upgrades and apply them to ensure all region's engine versions match and the global cluster switchover is unblocked. For more information, see Unblocking a global cluster switchover or failover.

During a switchover, Amazon DocumentDB switches over your primary cluster to your chosen secondary Region while it maintains your global cluster's existing replication topology. Before it starts the switchover process, Amazon DocumentDB waits for all secondary Region clusters to be fully synchronized with the primary Region cluster. Then, the DB cluster in the primary Region becomes read-only and the chosen secondary cluster promotes one of its read-only nodes to full writer status. Promoting this node to a writer allows that secondary cluster to assume the role of primary cluster. Because all secondary clusters were synchronized with the primary at the beginning of the process, the new primary continues operations for the Amazon DocumentDB global cluster without losing any data. Your database is unavailable for a short time while the primary and selected secondary clusters are assuming their new roles.

To optimize application availability, we recommend that you do the following before using this feature:

- Perform this operation during nonpeak hours or at another time when writes to the primary cluster are minimal.

- Take applications offline to prevent writes from being sent to the primary cluster of the Amazon DocumentDB global cluster.
- Check lag times for all Amazon DocumentDB secondary clusters in the global cluster by viewing the `GlobalClusterReplicationLag` metric in Amazon CloudWatch. This metric shows you how far behind (in milliseconds) replication to a secondary cluster is to the primary cluster. This value is directly proportional to the time it takes for Amazon DocumentDB to complete the switchover. Therefore, the larger the lag value, the longer the switchover will take.

  For more information about CloudWatch metrics for Amazon DocumentDB, see Amazon DocumentDB metrics.

During a switchover, the chosen secondary DB cluster is promoted to its new role as primary. However, it doesn't inherit the various configuration options of the primary DB cluster. A mismatch in configuration can lead to performance issues, workload incompatibilities, and other anomalous behavior. To avoid such issues, we recommend that you resolve differences between your Amazon DocumentDB global clusters for the following:

- **Configure Amazon DocumentDB DB cluster parameter group for the new primary, if necessary** — You can configure your Amazon DocumentDB cluster parameter groups independently for each cluster in your Amazon DocumentDB global cluster. That means that when you promote a secondary DB cluster to take over the primary role, the parameter group from the secondary might be configured differently than for the primary. If so, modify the promoted secondary DB cluster's parameter group to conform to your primary cluster's settings. To learn how, see Managing Amazon DocumentDB cluster parameter groups.
- **Configure monitoring tools and options, such as Amazon CloudWatch Events and alarms** — Configure the promoted cluster with the same logging ability, alarms, and so on as needed for the global cluster. As with parameter groups, configuration for these features isn't inherited from the primary during the switchover process. Some CloudWatch metrics, such as replication lag, are only available for primary Regions. Thus, a switchover changes how to view those metrics and set alarms on them, and could require changes to any predefined dashboards. For more information, see Monitoring Amazon DocumentDB.

> **Note**
>
> Typically, the role switchover can take up to several minutes.

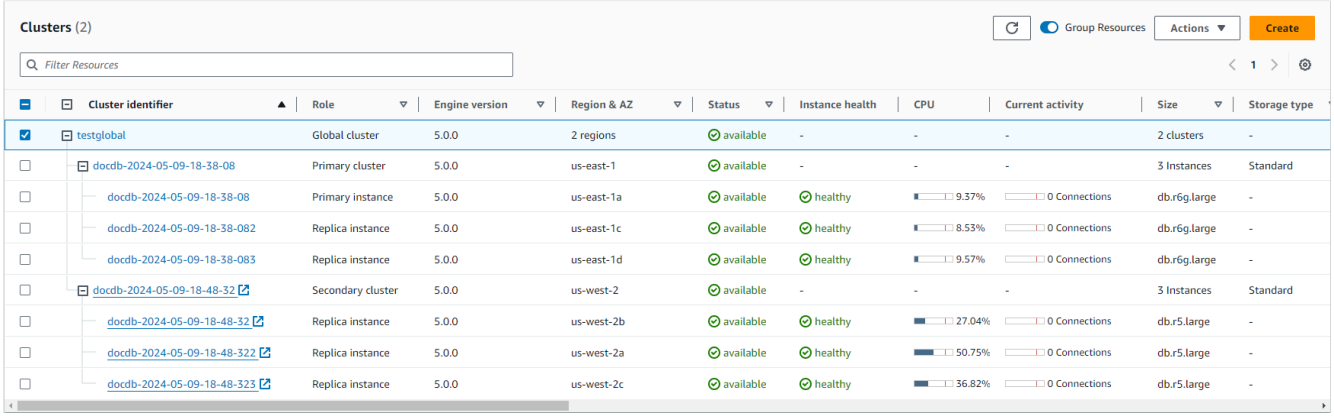
When the switchover process completes, the promoted Amazon DocumentDB cluster can handle write operations for the global cluster.

You can switch over your Amazon DocumentDB global cluster using the AWS Management Console or the AWS CLI:

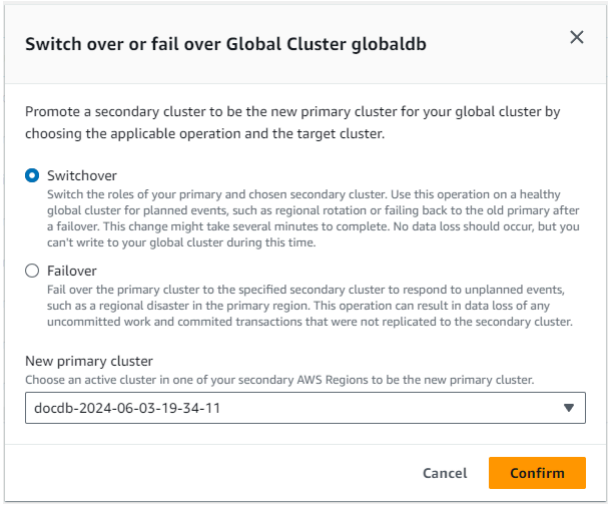
Using the AWS Management Console

Perform a switchover on your Amazon DocumentDB global cluster

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. Find and select the Amazon DocumentDB global cluster you want to switch over.



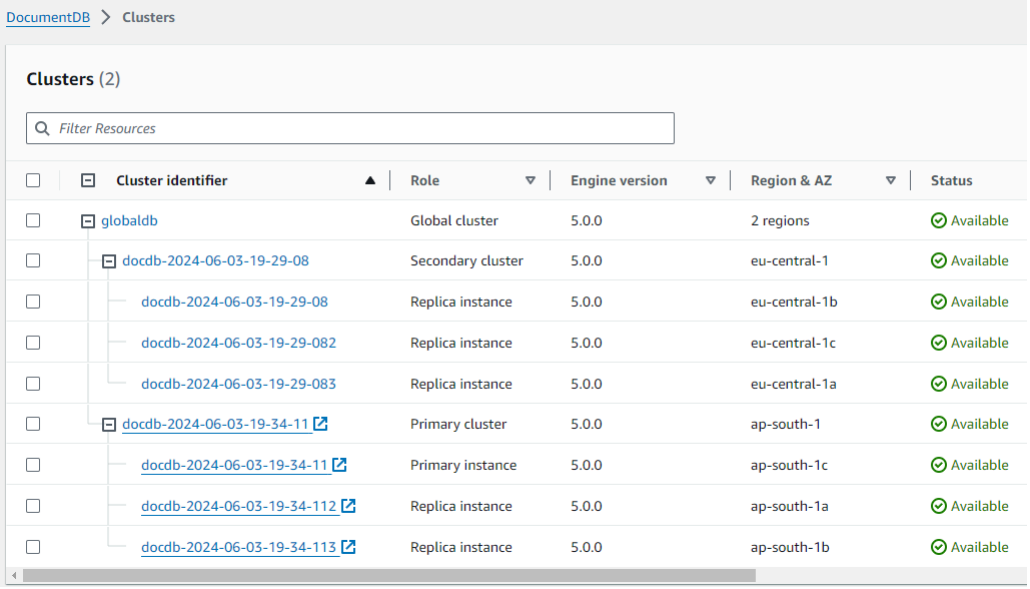
4. Choose **Switchover or Failover** from the **Actions** menu.
5. On the dialog box that appears, choose **Switchover**, then choose the secondary cluster from the **New primary cluster** field drop down list.



6. Choose **Confirm**.

   The status of the primary cluster changes to "**Switching-over**". This condition should take approximately three minutes. During this time, the status of all regional clusters show "**Modifying...**". Once the regions are synchronized and the new primary is promoted, it will show "**Available**" for all status fields and will be able to serve transactions.

7. When complete, the original primary cluster becomes the secondary cluster. The selected secondary cluster becomes the primary cluster.



Using the AWS CLI

Perform a switchover on your Amazon DocumentDB global cluster

Run the `switchover-global-cluster` CLI command to switch over your Amazon DocumentDB global cluster. With the command, pass values for the following options:

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`

In the following examples, replace each *`user input placeholder`* with your cluster's information.

For Linux, macOS, or Unix:

```
aws docdb switchover-global-cluster \
   --region region_of_primary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

For Windows:

```
aws docdb switchover-global-cluster ^
   --region region_of_primary ^
   --global-cluster-identifier global_cluster_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

Unblocking a global cluster switchover or failover

Global cluster switchovers and failovers are blocked when not all regional clusters in the global cluster are on the same engine version. If the versions do not match, you may see this error in response when calling a switchover or failover: The target DB cluster specified is running an engine version with a different patch level than the source DB cluster. We recommended routinely applying the latest engine versions to ensure you are running the latest updates to keep your global clusters in a healthy state.

To resolve this error, please update all secondary regions first, and then the primary region to the same engine version by applying any pending maintenance action items. To view pending maintenance action items, and to apply any needed changes to correct the issue, perform the instructions in one of the following tabs:

Using the AWS Management Console

To unblock a global cluster switchover or failover, you must determine if there are any pending maintenance actions for your clusters and apply them. Follow these steps to view and apply maintenance actions:

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. In the **Clusters** table, locate your global cluster in the **Cluster identifier** column. Under your global cluster, take note of each secondary cluster and the primary cluster for the given global cluster, and perform the following steps for each.

4. For each secondary cluster:

   a. If an update is available for your cluster, it is indicated as **Available**, **Required**, or **Next Window** in the **Maintenance** column.

   b. To take an action, choose the cluster to show it's details, then choose **Maintenance & backups**. The **Pending Maintenance** items appear.

   c. Under **Description**, if it indicates that a "New maintenance update is available", select it and then choose **Apply now**.

5. For your primary cluster:

   a. If an update is available for your cluster, it is indicated as **Available**, **Required**, or **Next Window** in the **Maintenance** column.

   b. To take an action, choose the cluster to show it's details, then choose **Maintenance & backups**. The **Pending Maintenance** items appear.

   c. Under **Description**, if it indicates that a "New maintenance update is available", select it and then choose **Apply now**.

Using the AWS CLI

To unblock a global cluster switchover or failover, you must determine if there are any pending maintenance actions for the cluster and apply them. Follow these steps to view and apply maintenance actions first on the secondary clusters then on the primary cluster for your global cluster:

1. Run the following on each secondary region's regional cluster first and then for the primary regions regional cluster.

2. Run the `describe-pending-maintenance-actions` CLI command with the `--resource-identifier` option to determine if any maintenance actions are available for your Amazon DocumentDB regional cluster.

   In the following examples, replace each *`user input placeholder`* with your cluster's information.

   For Linux, macOS, or Unix:

```
aws docdb describe-pending-maintenance-action \
```

```
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

For Windows:

```
aws docdb describe-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

The result looks similar to this:

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "CurrentApplyDate": "2025-04-11T03:01:00Z",
                    "Description": "db-version-upgrade",
                    "ForcedApplyDate": "2025-06-18T03:01:00Z",
                    "AutoAppliedAfterDate": "2025-05-11T03:01:00Z"
                    "OptInStatus": "pending"
                }
            ]
        }
    ]
}
```

3. If a maintenance action is needed, run the `apply-pending-maintenance-action` CLI command with the following options:

   - `--resource-identifier`
   - `--apply-action`
   - `--opt-in-type`
   - `--region`

   In the following examples, replace each *`user input placeholder`* with your cluster's information.

For Linux, macOS, or Unix:

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
   --apply-action system-update \
   --opt-in-type immediate \
   --region us-east-1
```

For Windows:

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --apply-action system-update ^
   --opt-in-type immediate ^
   --region us-east-1
```

4. Once the maintenance action has completed, run the `describe-pending-maintenance-actions` command again to ensure that there are no other actions pending for your cluster.

   The result you want is:

```
{
    "PendingMaintenanceActions": []
}
```

Using the Amazon DocumentDB API

To unblock a global cluster switchover or failover, you must determine if there are any pending maintenance actions for the cluster and apply them. Use the following APIs to view and apply maintenance actions:

1. Run the following on each secondary region's regional cluster first and then for the primary regions regional cluster.
2. Call the `PendingMaintenanceAction` API to determine if any maintenance actions are available for your Amazon DocumentDB global cluster.

3. Apply any changes by calling the `ApplyPendingMaintenanceAction` API.

Managing Amazon DocumentDB clusters

To manage an Amazon DocumentDB cluster, you must have an IAM policy with the appropriate Amazon DocumentDB control plane permissions. These permissions allow you to create, modify, and delete clusters and instances. The `AmazonDocDBFullAccess` policy provides all the required permissions for administering an Amazon DocumentDB cluster.

The following topics show how to perform various tasks when working with Amazon DocumentDB clusters, including creating, deleting, modifying, connecting to, and viewing clusters.

Topics

- Understanding clusters
- Amazon DocumentDB Cluster settings
- Amazon DocumentDB cluster storage configurations
- Determining a cluster's status
- Amazon DocumentDB cluster lifecycle
- Scaling Amazon DocumentDB clusters
- Cloning a volume for an Amazon DocumentDB cluster
- Understanding Amazon DocumentDB cluster fault tolerance

Understanding clusters

Amazon DocumentDB separates compute and storage, and offloads data replication and backup to the cluster volume. A cluster volume provides a durable, reliable, and highly available storage layer that replicates data six ways across three Availability Zones. Replicas enable higher data availability and read scaling. Each cluster can scale up to 15 replicas.

Noun	Description	API Operations (Verbs)

Cluster	Consists of one or more instances and a cluster storage volume that manages the data for those instances.	`create-db-cluster`
		`delete-db-cluster`
		`describe-db-clusters`

Noun	Description	API Operations (Verbs)
		`modify-db-cluster`

Instance	Reading and writing data to the cluster storage volume is done via instances. In a given cluster, there are two types of instances: primary and replica. A cluster always has one primary instance and can have 0–15 replicas.	`create-db-instance`
		`delete-db-instance`
		`describe-db-instances`
		`modify-db-instance`
		`describe-orderable-db-instance-options`
		`reboot-db-instance`

Cluster volume	A virtual database storage volume that spans three Availability Zones, with each Availability Zone having two copies of the cluster data.	N/A
Primary instance	Supports both read and write operations, and performs all data modifications to the cluster volume. Each cluster has one primary instance.	N/A

Noun	Description	API Operations (Verbs)
Replica instance	Supports only read operations. Each Amazon DocumentDB cluster can have up to 15 replica instances in addition to the primary instance. Multiple replicas distribute the read workload. By locating replicas in separate Availability Zones, you can also increase database availability.	N/A
Cluster endpoint	An endpoint for an Amazon DocumentDB cluster that connects to the current primary instance for the cluster. Each Amazon DocumentDB cluster has a cluster endpoint and one primary instance.	N/A
Reader endpoint	An endpoint for an Amazon DocumentDB cluster that connects to one of the available replicas for that cluster. Each Amazon DocumentDB cluster has a reader endpoint. If there is more than one replica, the reader endpoint directs each connection request to one of the Amazon DocumentDB replicas.	N/A

Noun	Description	API Operations (Verbs)
Instance endpoint	An endpoint for an instance in an Amazon DocumentDB cluster that connects to a specific instance. Each instance in a cluster, regardless of instance type, has its own unique instance endpoint.	N/A

Amazon DocumentDB Cluster settings

When you create or modify a cluster, it is important to understand which parameters are immutable and which are modifiable after the cluster has been created. The following table lists all the settings, or parameters, that are specific to a cluster. As specified in the table, some are modifiable, others are not.

> **Note**
>
> These settings should not be confused with Amazon DocumentDB cluster parameter groups and their parameters. For more information about cluster parameter groups, see Managing Amazon DocumentDB cluster parameter groups.

Parameter	Modifiable	Notes
`DBClusterIdentifier`	Yes	Naming constraints: • Length is [1—63] letters, numbers, or hyphens. • First character must be a letter. • Cannot end with a hyphen or contain two consecutive hyphens. • Must be unique for all clusters across Amazon RDS, Amazon Neptune, and

Parameter	Modifiable	Notes
		Amazon DocumentDB per AWS account, per Region.
`Engine`	No	Must be `docdb`.
`BackupRetentionPeriod`	Yes	Must be between [1-35] days.
`DBClusterParameterGroupName`	Yes	Naming constraints: • Length is [1—255] alphanumeric characters. • First character must be a letter. • Cannot end with a hyphen or contain two consecutive hyphens.
`DBSubnetGroupName`	No	After a cluster has been created, you cannot modify the cluster's subnet.
`EngineVersion`	No	Value can be `5.0.0` (default), `4.0.0`, or `3.6.0`.
`KmsKeyId`	No	If you choose to encrypt your cluster, you cannot change the AWS KMS key that you used to encrypt your cluster.
`MasterUsername`	No	After a cluster has been created, you cannot modify the `MasterUsername` . Naming constraints: • Length is [1—63] alphanumeric characters. • First character must be a letter. • Cannot be a word reserved by the database engine.

Parameter	Modifiable	Notes

`MasterUserPassword`	Yes	Constraints:
		• Length is [8—100] printable ASCII character s.
		• Can use any printable ASCII characters except for the following:
		  • **`/`** (forward slash)
		  • **`"`** (double quotation mark)
		  • **`@`** (at symbol)

`Port`	Yes	The port number applies to all instances in the cluster.
`PreferredBackupWindow`	Yes	
`PreferredMaintenanceWindow`	Yes	
`StorageEncrypted`	No	If you choose to encrypt your cluster, it cannot be unencrypted.
`StorageType`	Yes	The storage type for the DB cluster: Standard (`standard`) or I/O-Optimized (`iopt1`). Default: `standard` This parameter can be configured with `CreateDBCluster` and `ModifyDBCluster`. For more information, see Amazon DocumentDB cluster storage configurations.
`Tags`	Yes	

Parameter	Modifiable	Notes
`VpcSecurityGroupIds`	No	After a cluster has been created, you cannot modify the VPC that the cluster resides in.

Amazon DocumentDB cluster storage configurations

Starting from Amazon DocumentDB 5.0, instance-based clusters support two storage configuration types:

- **Amazon DocumentDB standard storage**: Designed for customers with low to moderate I/O consumption. If you expect your I/O costs to be less than 25% of your total Amazon DocumentDB cluster, this choice might be ideal for you. With the Amazon DocumentDB standard storage configuration, you're billed on a pay-per-request I/O basis in addition to instance and storage charges. This means your billing might vary from one cycle to another based on usage. The configuration is tailored to accommodate fluctuating I/O demands of your application.
- **Amazon DocumentDB I/O-optimized storage**: Designed for customers who prioritize price predictability or have I/O intensive applications. The I/O-optimized configuration offers improved performance, increased throughput, and reduced latency for customers with I/O intensive workloads. If you expect your I/O costs to exceed 25% of your total Amazon DocumentDB cluster costs, this option offers enhanced price performance. With the Amazon DocumentDB I/O-optimized storage configuration, you won't be charged based on I/O operations, ensuring predictable costs each billing cycle. The configuration stabilizes costs while improving performance.

You can switch your existing database clusters once every 30 days to Amazon DocumentDB I/O-optimized storage. You can switch back to Amazon DocumentDB standard storage at any time. The next date to modify the storage configuration to I/O-optimized can be tracked with the `describe-db-clusters` command using the AWS CLI or through the AWS Management Console in the cluster's configuration page.

You can create a new database cluster including the Amazon DocumentDB I/O-optimized configuration or convert your existing database clusters with a few clicks in the AWS Management Console, a single parameter change in the AWS Command Line Interface (AWS CLI), or through AWS SDKs. There is no downtime or reboot of instances required during or after modifying the storage configuration.

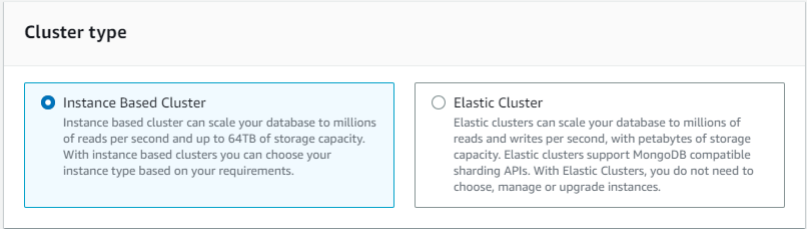
Requirement	Standard	I/O-Optimized	Usage
Default Storage Type	✓		
Low to Moderate I/O Workload	✓		Best if expected I/O charges are less than or equal to 25%
Price Predictability		✓	
High I/O Workload		✓	Best if expected I/O charges are greater than or equal to 25%
High Write Throughput		✓	Average 30%-50% observed improvement

Creating an I/O-optimized cluster

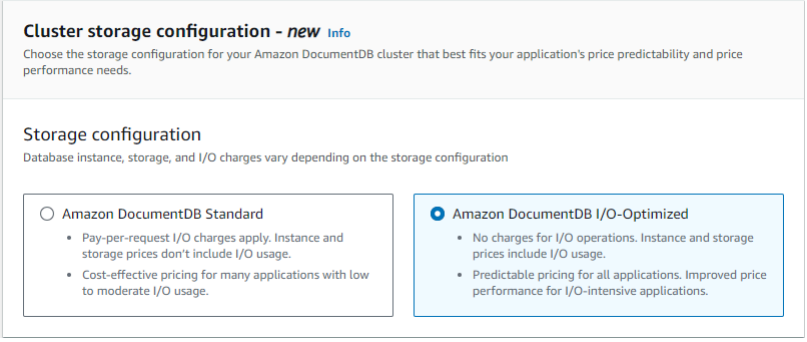
Using the AWS Management Console

To create or modify an I/O-optimized cluster using the AWS Management Console:

1. On the Amazon DocumentDB management console, under **Clusters**, choose either **Create** or select the cluster and choose **Actions**, and then choose **Modify**.
2. If you are creating a new cluster, make sure you choose **Instance Based Cluster** in the **Cluster type** section (this is the default option).



3. In the **Configuration** section, under **Cluster storage configuration**, choose **Amazon DocumentDB I/O-Optimized**.



4. Complete your cluster creation or modification and choose **Create cluster** or **Modify cluster**.

   For the complete Create cluster process, see Creating a cluster and primary instance using the AWS Management Console.

   For the complete Modify cluster process, see Modifying an Amazon DocumentDB cluster.

Using the AWS CLI

To create an I/O-optimized cluster using the AWS CLI:

In the following examples, replace each *user input placeholder* with your own information.

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
      --engine-version 5.0.0 \
      --storage-type iopt1 \
      --deletion-protection \
      --master-username username \
      --master-user-password password
```

For Windows:

```
aws docdb create-db-cluster ^
      --db-cluster-identifier sample-cluster ^
      --engine docdb ^
      --engine-version 5.0.0 ^
```

```
    --storage-type iopt1 ^
    --deletion-protection ^
    --master-username username ^
    --master-user-password password
```

Cost analysis for determining storage configuration

With Amazon DocumentDB, you have the flexibility to choose your storage configuration for every database cluster you have. In order to properly allocate your clusters between standard and I/O-optimized, you can track your Amazon DocumentDB costs cluster-wise. To do so, you can add tags to existing clusters, enable cost allocation tagging in your AWS Billing and Cost Management dashboard, and analyze your costs for a given cluster in the AWS Cost Explorer Service. For information on cost analysis, see our blog Using cost allocation tags.

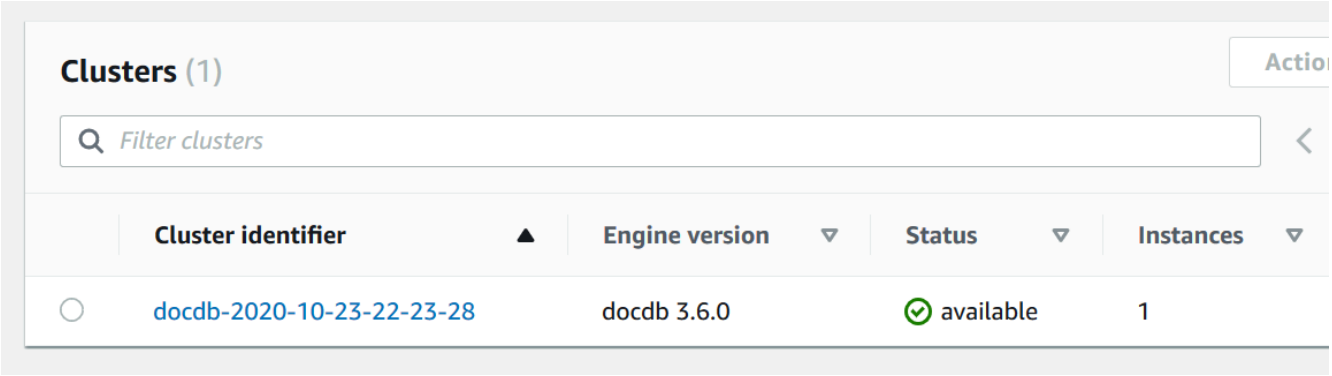
Determining a cluster's status

You can determine a cluster's status using the AWS Management Console or AWS CLI.

Using the AWS Management Console

Use the following procedure to see the status of your Amazon DocumentDB cluster using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. In the **Cluster identifier** column, find the name of the cluster that you are interested in. Then, to find the status of the cluster, read across that row to the **Status** column, as shown below.



Using the AWS CLI

Use the `describe-db-clusters` operation to see the the status of your Amazon DocumentDB cluster using the AWS CLI.

The following code finds the status of the cluster `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

For Windows:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

Output from this operation looks something like the following (JSON format).

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB cluster lifecycle

The lifecycle of an Amazon DocumentDB cluster includes creating, describing, modifying, and deleting the cluster. This section provides information about how to complete these processes.

Topics

- Creating an Amazon DocumentDB cluster
- Describing Amazon DocumentDB clusters
- Modifying an Amazon DocumentDB cluster
- Determining pending maintenance

- Performing a patch update to a cluster's engine version
- Stopping and starting an Amazon DocumentDB cluster
- Deleting an Amazon DocumentDB cluster

Creating an Amazon DocumentDB cluster

An Amazon DocumentDB cluster consists of instances and a cluster volume that represents the data for the cluster. The cluster volume is replicated six ways across three Availability Zones as a single, virtual volume. The cluster contains a primary instance and, optionally, up to 15 replica instances.

The following sections show how to create an Amazon DocumentDB cluster using either the AWS Management Console or the AWS CLI. You can then add additional replica instances for that cluster. When you use the console to create your Amazon DocumentDB cluster, a primary instance is automatically created for you at the same time. If you use the AWS CLI to create your Amazon DocumentDB cluster, after the cluster's status is *available*, you must then create the primary instance for that cluster.

Prerequisites

The following are prerequisites for creating an Amazon DocumentDB cluster.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

VPC prerequisites

You can only create an Amazon DocumentDB cluster in an Amazon Virtual Private Cloud (Amazon VPC). Your Amazon VPC must have at least one subnet in each of at least two Availability Zones in order for you to use it with an Amazon DocumentDB cluster. By distributing your cluster instances across Availability Zones, you ensure that instances are available in your cluster in the unlikely case of an Availability Zone failure.

Subnet prerequisites

When creating an Amazon DocumentDB cluster, you must choose a VPC and corresponding subnet group within that VPC to launch your cluster. Subnets determine the Availability Zone and IP range within that Availability Zone that you want to use to launch an instance. For the purposes of this discussion, the terms *subnet* and *Availability Zone* are used interchangeably. A subnet group is a named set of subnets (or Availability Zones). What a subnet group allows you to do is specify the Availability Zones that you want to use to for launching Amazon DocumentDB instances. For example, in a cluster with three instances, it is recommended for high availability that each of those instances is provisioned in separate Availability Zones. Thus, if a single Availability Zone goes down, it only affects a single instance.

Amazon DocumentDB instances can currently be provisioned in up to three Availability Zones. Even if a subnet group has more than three subnets, you can only use three of those subnets to create an Amazon DocumentDB cluster. As a result, it is suggested that when you create a subnet group, only choose the three subnets that you want to deploy your instances to. In US East (N. Virginia), your subnet group can have six subnets (or Availability Zones). However, when an Amazon DocumentDB cluster is provisioned, Amazon DocumentDB chooses three of those Availability Zones that it uses to provision instances.

For example, suppose that when you are creating a cluster, Amazon DocumentDB chooses the Availability Zones {1A, 1B, and 1C}. If you try to create an instance in Availability Zone {1D}, the API call fails. However, if you choose to create an instance without specifying a particular Availability Zone, then Amazon DocumentDB chooses an Availability Zone on your behalf. Amazon DocumentDB uses an algorithm to load balance the instances across Availability Zones to help you achieve high availability. For example, if three instances are provisioned, by default, they are provisioned across three Availability Zones and are not provisioned all in a single Availability Zone.

Recommendations:

- Unless you have a specific reason, always create a subnet group with three subnets. Doing so helps ensure that clusters with three or more instances can achieve higher availability as instances are provisioned across three Availability Zones.
- Always spread instances across multiple Availability Zones to achieve high availability. Never place all instances for a cluster in a single Availability Zone.
- Because failover events can happen at any time, you should not assume that a primary instance or replica instances are always in a particular Availability Zone.

Additional prerequisites

The following are some additional prerequisites for creating an Amazon DocumentDB cluster:

- If you are connecting to AWS using AWS Identity and Access Management (IAM) credentials, your IAM account must have IAM policies that grant the permissions that are required to perform Amazon DocumentDB operations.

  If you are using an IAM account to access the Amazon DocumentDB console, you must first sign in to the AWS Management Console with your IAM account. Then go to the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
- If you want to tailor the configuration parameters for your cluster, you must specify a cluster parameter group and parameter group with the required parameter settings. For information about creating or modifying a cluster parameter group or parameter group, see Managing Amazon DocumentDB cluster parameter groups.
- You must determine the TCP/IP port number that you want to specify for your cluster. The firewalls at some companies block connections to the default ports for Amazon DocumentDB. If your company firewall blocks the default port, choose another port for your cluster. All instances in a cluster use the same port.

Creating a cluster and primary instance using the AWS Management Console

The following procedures describe how to use the console to launch an Amazon DocumentDB cluster with one or more instances.

Create a cluster: using default settings

To create a cluster with instances using the default settings using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. If you want to create your cluster in an AWS Region other than the US East (N. Virginia) Region, choose the Region from the list in the upper-right section of the console.
3. In the navigation pane, choose **Clusters**, and then choose **Create**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

4. On the **Create Amazon DocumentDB cluster** page, complete the **Configuration** pane.

   a. **Cluster identifier**—Accept the Amazon DocumentDB provided name, or enter a name for your cluster; for example, **`sample-cluster`**.

      Cluster naming constraints:

      - Length is [1—63] letters, numbers, or hyphens.
      - First character must be a letter.
      - Cannot end with a hyphen or contain two consecutive hyphens.
      - Must be unique for all clusters across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.

   b. **Engine version**—Accept the default engine version of 5.0.0, or optionally choose 4.0.0 or 3.6.0.

   c. **Instance class**—Accept the default `db.r5.large`, or choose the instance class that you want from the list.

   d. **Number of instances**—In the list, choose the number of instances that you want to be created with this cluster. The first instance is the primary instance, and all other instances are read-only replica instances. You can add and delete instances later if you need to. By

default, an Amazon DocumentDB cluster launches with three instances (one primary and two replicas).

5. Complete the **Cluster storage configuration** section.

   Choose either **Amazon DocumentDB Standard** (default) or **Amazon DocumentDB I/O-Optimized**. For more information, see Amazon DocumentDB cluster storage configurations.

6. Complete the **Authentication** pane.

   a. **Username**—Enter a name for the primary user. To log in to your cluster, you must use the primary user name.

      Primary user naming constraints:

      - Length is [1—63] alphanumeric characters.
      - First character must be a letter.
      - Cannot be a word reserved by the database engine.

   b. Choose one of the following password options:

      - **Managed in AWS Secrets Manager**—Choose this option if you want AWS Secrets Manager to automatically manage your primary user password.

        If you choose this option, configure the KMS key by either creating your own or using a key that Secrets Manager creates.

      - **Self managed**—Choose this option if you want to self-manage your primary user password. If you choose this option, enter a password for the primary user, and then confirm it. To log in to your cluster, you must use the password for the primary user.

        Password constraints:

        - Length is [8-100] printable ASCII characters.
        - Can use any printable ASCII characters except for the following:
          - **/** (forward slash)
          - **"** (double quotation mark)
          - **@** (at symbol)

7. At the bottom of the screen, choose one of the following:

   - To create the cluster now, choose **Create cluster**.
   - To not create the cluster, choose **Cancel**.

- To further configure the cluster before creating, choose **Show additional configurations**, and then continue at Create a cluster: additional configurations.

  The configurations covered in the **Additional Configurations** section are as follows:

  - **Network settings**—The default is to use the `default` VPC security group.
  - **Cluster options**—The default is to use port is 27017 and the default parameter group.
  - **Encryption**—The default is to enable encryption using the `(default) aws/rds` key.

    > **Important**
    >
    > After a cluster is encrypted, it cannot be unencrypted.

  - **Backup**—The default is to retain backups for 1 day and let Amazon DocumentDB choose the backup window.
  - **Log exports**—The default is to not export audit logs to CloudWatch Logs.
  - **Maintenance**—The default is to let Amazon DocumentDB choose the maintenance window.
  - **Deletion protection**—Protect your cluster from accidental deletion. Default for cluster created using the console is *enabled*.

  If you accept the default settings now, you can change most of them later by modifying the cluster.

8. Enable inbound connection for your cluster's security group.

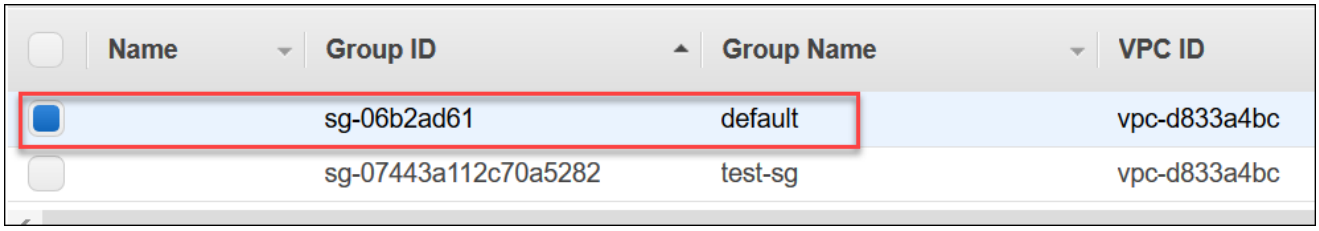
   If you did not change the defaults settings for your cluster, you created a cluster using the default security group for the default VPC in the given region. To connect to Amazon DocumentDB, you must enable inbound connections on port 27017 (or the port of your choice) for your cluster’s security group.

   To add an inbound connection to your cluster's security group

   a. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

   b. In the **Resources** section of the main window, choose **Security groups**.

c. From the list of security groups locate the security group you used when creating your cluster (it is most likely the *default* security group) and choose the box to the left of the security group's name.



d. From the **Actions** menu, choose **Edit inbound rules** then choose or enter the rule constraints.

   i. **Type**—From the list, choose the protocol to open to network traffic.

   ii. **Protocol**—From the list, choose the type of protocol.

   iii. **Port Range**—For a custom rule, enter a port number or port range. Be sure that the port number or range includes the port you specified when you created your cluster (default: 27017).

   iv. **Source**—Specifies the traffic that can reach your instance. From the list, choose the traffic source. If you choose **Custom**, specify a single IP address or an IP address range in CIDR notation (e.g., 203.0.113.5/32).

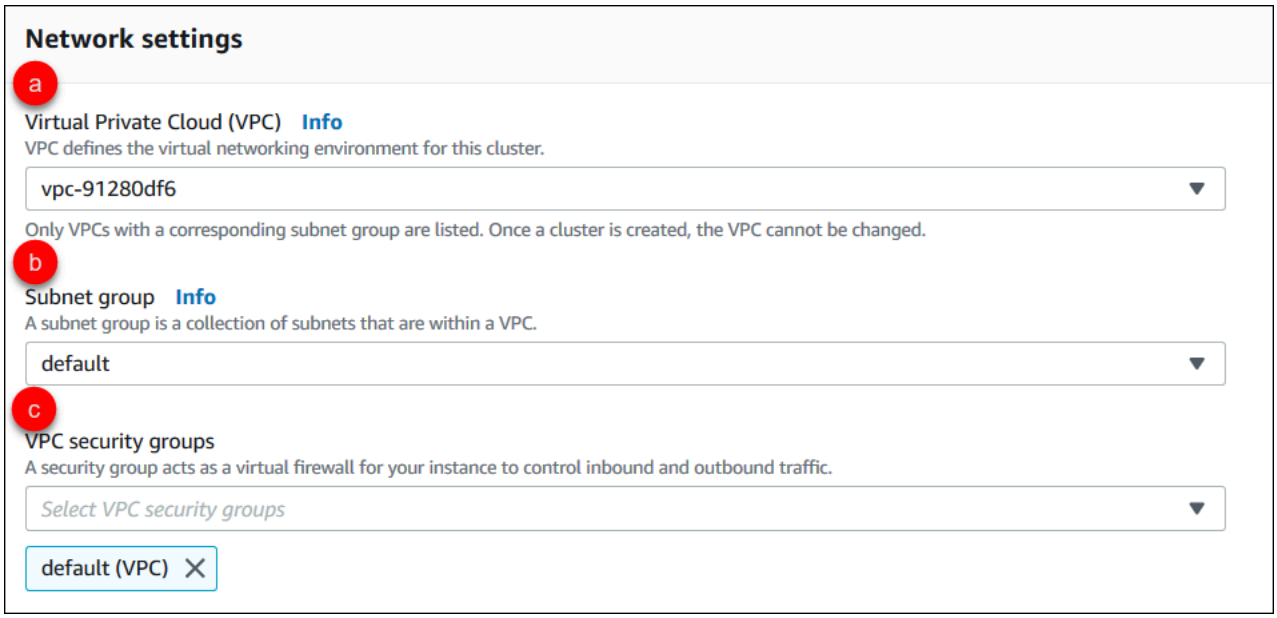
   v. **Description**—Enter a description for this rule.

   vi. When finished creating the rule, choose **Save**.

Create a cluster: additional configurations

If you want to accept the default settings for your cluster, you can skip the following steps and choose **Create cluster**.

1. Complete the **Network settings** pane.

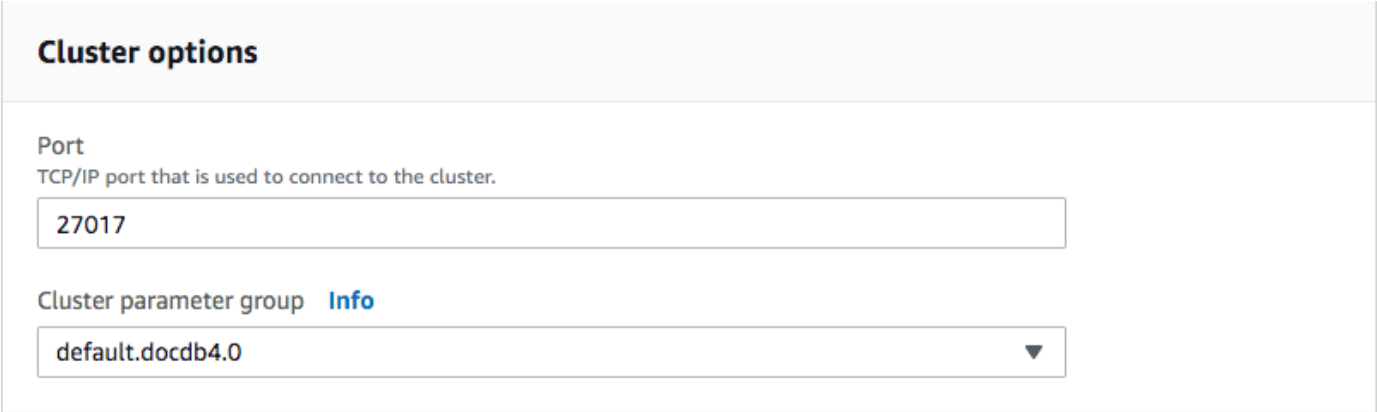


   a. **Virtual Private Cloud (VPC)**—In the list, choose the Amazon VPC that you want to launch this cluster in.

   b. **Subnet group**—In the list, choose the subnet group you want to use for this cluster.

   c. **VPC security groups**—In the list, choose the VPC security group for this cluster.

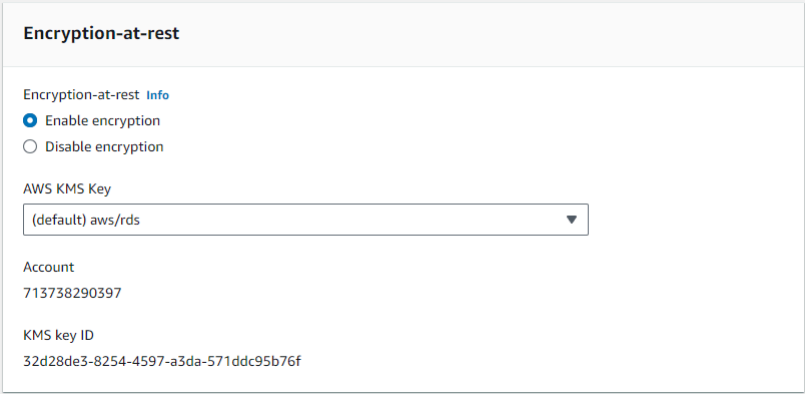
2. Complete the **Cluster options** pane.



   a. **Data base port**—Use the up and down arrows to set the TCP/IP port that applications will use to connect to your instance.

   b. **Cluster parameter group**—In the list of parameter groups, choose the cluster parameter group for this cluster.

3. Complete the **Encryption** pane.



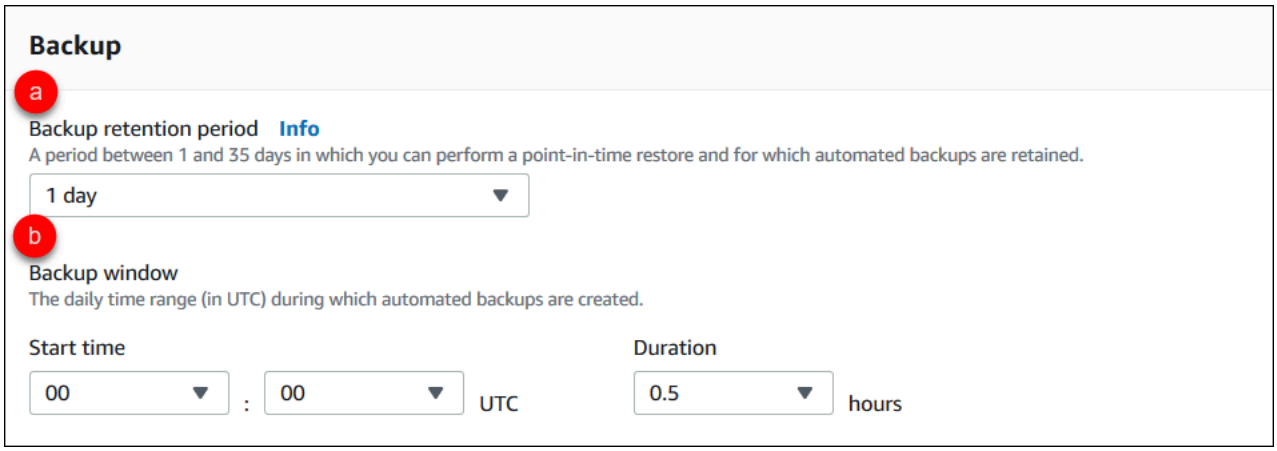
a. **Encryption-at-rest**—Choose one of the following:

   - **Enable encryption**—Default. All data at rest is encrypted. If you choose to encrypt your data, you cannot undo this action.
   - **Disable encryption**—Your data is not encrypted.

b. **AWS KMS key**—This is only available if you are encrypting your data. In the list, choose the key that you want to use for encrypting the data in this cluster. The default is `(default) aws/rds`.

   If you chose **Enter a key ARN**, you must enter an Amazon Resource Name (ARN) for the key.

4. Complete the **Backup** pane.



a. **Backup retention period**—In the list, choose the number of days to keep automatic backups of this cluster before deleting them.

b. **Backup window**—Set the daily time and duration during which Amazon DocumentDB is to make backups of this cluster.

i. **Start time**—In the first list, choose the start time hour (UTC) for starting your automatic backups. In the second list, choose the minute of the hour that you want automatic backups to begin.

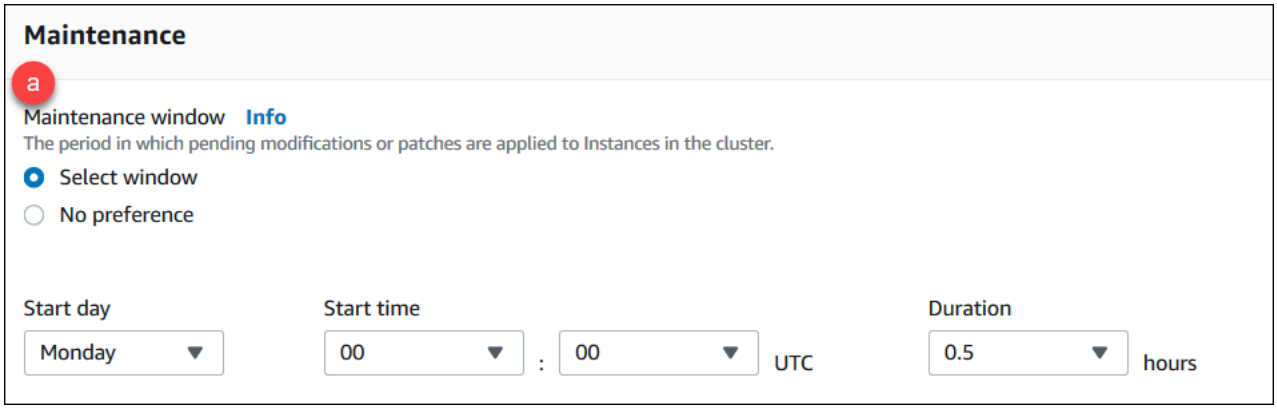
ii. **Duration**—In the list, choose the number of hours to be allocated to creating automatic backups.

5. Complete the **Log exports** pane by selecting the types of logs you want to export to CloudWatch Logs.

- **Audit logs**—Select this option to enable exporting audit logs to Amazon CloudWatch Logs. If you select **Audit logs**, you must enable `audit_logs` in the cluster's custom parameter group. For more information, see Auditing Amazon DocumentDB events.
- **Profiler logs**—Select this option to enable exporting operation profiler logs to Amazon CloudWatch Logs. If you select **Profiler logs**, you must also modify the following parameters in the cluster's custom parameter group:
  - `profiler`—Set to `enabled`.
  - `profiler_threshold_ms`—Set to a value `[0-INT_MAX]` to set the threshold for profiling operations.
  - `profiler_sampling_rate`—Set to a value `[0.0-1.0]` to set the percentage of slow operations to profile.

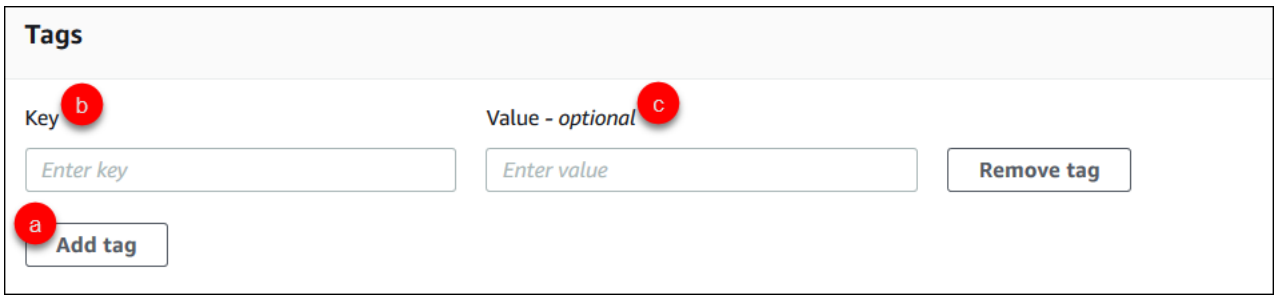
  For more information, see Profiling Amazon DocumentDB operations.

6. Complete the **Maintenance** pane.



- Choose one of the following
  - **Select window**—You can specify the day of the week, UTC start time, and duration for Amazon DocumentDB to perform maintenance on your cluster.
    a. **Start day**—In the list, choose the day of the week to start cluster maintenance.
    b. **Start time**—In the lists, choose the hour and minute (UTC) to start maintenance.
    c. **Duration**—In the list, choose how much time to allocate for cluster maintenance. If maintenance cannot be finished in the specified time, the maintenance process will continue past the specified time until finished.
  - **No preference**—Amazon DocumentDB chooses the day of the week, start time, and duration for performing maintenance.

7. If you want to add one or more tags to this cluster, complete the **Tags** pane.



For each tag you want to add to the cluster, repeat the following steps. You may have up to 10 on a cluster.

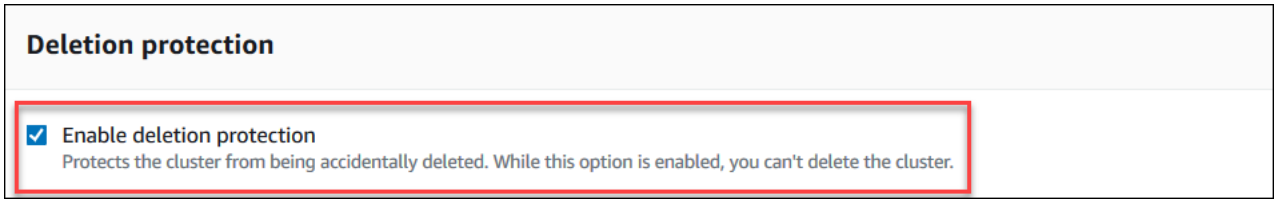
a. Choose **Add tags**.

b. Type the tag's **Key**.

c. Optionally type the tag's **Value**.

To remove a tag, choose **Remove tag**.

8. **Deletion Protection** is enabled by default when you create a cluster using the console. To disable deletion protection, clear **Enable deletion protection**. When enabled, deletion protection prevents a cluster from being deleted. To delete a deletion protected cluster, you must first modify the cluster to disable deletion protection.



For more information about deletion protection, see Deleting an Amazon DocumentDB cluster.

9. To create the cluster, choose **Create cluster**. Otherwise, choose **Cancel**.

Creating a cluster using the AWS CLI

The following procedures describe how to use the AWS CLI to launch an Amazon DocumentDB cluster and create an Amazon DocumentDB replica.

Parameters

- **`--db-cluster-identifier`**—Required. A lowercase string that identifies this cluster.

  Cluster Naming Constraints:

  - Length is [1–63] letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all clusters (across Amazon RDS, Amazon Neptune, and Amazon DocumentDB) per AWS account, per Region.

- **`--engine`**—Required. Must be **`docdb`**.
- **`--deletion-protection | --no-deletion-protection`**—Optional. When deletion protection is enabled, it prevents a cluster from being deleted. When you use the AWS CLI, the default setting is to have deletion protection disabled.

For more information about deletion protection, see Deleting an Amazon DocumentDB cluster.

- **`--storage-type standard | iopt1`**—Optional. Default: **`standard`**. The cluster's storage configuration. Valid values are `standard` (Standard) or `iopt1` (I/O-optimized).
- **`--master-username`**—Required. The user name used to authenticate the user.

  Master User Naming Constraints:

  - Length is [1-63] alphanumeric characters.
  - First character must be a letter.
  - Cannot be a word reserved by the database engine.
- **`--master-user-password`**—Optional. The user's password used to authenticate the user.

  Master Password Constraints:

  - Length is [8-100] printable ASCII characters.
  - Can use any printable ASCII characters except for the following:
    - **`/`** (forward slash)
    - **`"`** (double quotation mark)
    - **`@`** (at symbol)
- **`--manage-master-user-password`**—Optional. Amazon DocumentDB generates the master user password and manages it throughout its lifecycle in Secrets Manager.

For additional parameters, see CreateDBCluster.

To launch an Amazon DocumentDB cluster using the AWS CLI

To create an Amazon DocumentDB cluster, call the `create-db-cluster` AWS CLI. The following AWS CLI command creates an Amazon DocumentDB cluster named `sample-cluster` with deletion protection enabled. For more information on deletion protection, see Deleting an Amazon DocumentDB cluster.

Also, `--engine-version` is an optional parameter that defaults to the latest major engine version. The current major engine version is 5.0.0. When new major engine versions are released, the default engine version for `--engine-version` will be updated to reflect the last major engine

version. As a result, for production workloads, and especially those that are dependent on scripting, automation, or AWS CloudFormation templates, we recommend that you explicitly specify the `--engine-version` to the intended major version.

> **Note**
>
> If a `db-subnet-group-name` or `vpc-security-group-id` is not specified, Amazon DocumentDB will use the default subnet group and Amazon VPC security group for the given region.

For Linux, macOS, or Unix:

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
      --engine-version 5.0.0 \
      --deletion-protection \
      --master-username masteruser \
      --master-user-password password
```

For Windows:

```
aws docdb create-db-cluster ^
      --db-cluster-identifier sample-cluster ^
      --engine docdb ^
      --engine-version 5.0.0 ^
      --deletion-protection ^
      --master-username masteruser ^
      --master-user-password password
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBCluster": {
        "StorageEncrypted": false,
        "DBClusterMembers": [],
        "Engine": "docdb",
        "DeletionProtection" : "enabled",
        "ClusterCreateTime": "2018-11-26T17:15:19.885Z",
        "DBSubnetGroup": "default",
```

```
        "EngineVersion": "5.0.0",
        "MasterUsername": "masteruser",
        "BackupRetentionPeriod": 1,
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "DBClusterIdentifier": "sample-cluster",
        "MultiAZ": false,
        "DBClusterParameterGroup": "default.docdb5.0",
        "PreferredBackupWindow": "09:12-09:42",
        "DbClusterResourceId": "cluster-KQSGI4MHU4NTDDRVNLNTU7XVAY",
        "PreferredMaintenanceWindow": "tue:04:17-tue:04:47",
        "Port": 27017,
        "Status": "creating",
        "ReaderEndpoint": "sample-cluster.cluster-ro-sfcrlcjcoroz.us-
east-1.docdb.amazonaws.com",
        "AssociatedRoles": [],
        "HostedZoneId": "ZNKXTT8WH85VW",
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-77186e0d",
                "Status": "active"
            }
        ],
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1e"
        ],
        "Endpoint": "sample-cluster.cluster-sfcrlcjcoroz.us-east-1.docdb.amazonaws.com"
    }
}
```

It takes several minutes to create the cluster. You can use the AWS Management Console or AWS CLI to monitor the status of your cluster. For more information, see Monitoring an Amazon DocumentDB cluster's status.

> **Important**
>
> When you use the AWS CLI to create an Amazon DocumentDB cluster, no instances are created. Consequently, you must explicitly create a primary instance and any replica instances that you need. You can use either the console or AWS CLI to create the instances. For more information, see Adding an Amazon DocumentDB instance to a cluster.

For more information, see `CreateDBCluster` in the *Amazon DocumentDB API Reference*.

Describing Amazon DocumentDB clusters

You can use either Amazon DocumentDB Management Console or the AWS CLI to see details such as connection endpoints, security groups, VPCs, and parameter groups pertaining to your Amazon DocumentDB clusters.

For more information, see the following:

- Monitoring an Amazon DocumentDB cluster's status
- Finding a cluster's endpoints

Using the AWS Management Console

Use the following procedure to view the details of a specified Amazon DocumentDB cluster using the console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. In the list of clusters, choose the name of the cluster that you want to see the details of. The information about the cluster is organized into the following groupings:

   - **Summary** — General information about the cluster, including the engine version, cluster status, pending maintenance, and the status of its parameter group.
   - **Connectivity & Security** —The **Connect** section lists connection endpoints to connect to this cluster with the mongo shell or with an application. The **Security Groups** section lists the security groups associated with this cluster and their VPC ID and descriptions.

- **Configuration** — The **Cluster details** section lists details about the cluster, including the cluster's Amazon Resource Name (ARN), endpoint, and parameter group. It also lists the cluster's backup information, maintenance details, and security and network settings. The **Cluster instances** section lists the instances that belong to this cluster with each instance's role and cluster parameter group status.
- **Monitoring** — The Amazon CloudWatch Logs metrics for this cluster. For more information, see Monitoring Amazon DocumentDB with CloudWatch.
- **Events & tags** — The **Recent events** section lists the recent events for this cluster. Amazon DocumentDB keeps a record of events that relate to your clusters, instances, snapshots, security groups, and cluster parameter groups. This information includes the date, time, and message associated with each event. The **Tags** section lists the tags attached to this cluster.

Using the AWS CLI

To view the details of your Amazon DocumentDB clusters using the AWS CLI, use the `describe-db-clusters` command as shown in the examples below. For more information, see `DescribeDBClusters` in the *Amazon DocumentDB Resource Management API Reference*.

> **Note**
>
> For certain management features such as cluster and instance lifecycle management, Amazon DocumentDB leverages operational technology that is shared with Amazon RDS. The `filterName=engine,Values=docdb` filter parameter returns only Amazon DocumentDB clusters.

Example

Example 1: List all Amazon DocumentDB clusters

The following AWS CLI code lists the details for all Amazon DocumentDB clusters in a region.

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

Output from this operation looks something like the following.

```
{
```

```
    "DBClusters": [
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-1",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-2",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-3",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        }
    ]
}
```

Example

Example 2: List all details for a specified Amazon DocumentDB cluster

The following AWS CLI code lists the details for the cluster `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
   --filter Name=engine,Values=docdb \
   --db-cluster-identifier sample-cluster
```

For Windows:

```
aws docdb describe-db-clusters ^
   --filter Name=engine,Values=docdb ^
   --db-cluster-identifier sample-cluster
```

Output from this operation looks something like the following.

```
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1a",
                "us-east-1d"
            ],
            "BackupRetentionPeriod": 2,
            "DBClusterIdentifier": "sample-cluster",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            "EarliestRestorableTime": "2023-11-07T22:34:08.148000+00:00",
            "Endpoint": "sample-cluster.node.us-east-1.amazon.com",
            "ReaderEndpoint": "sample-cluster.node.us-east-1.amazon.com",
            "MultiAZ": false,
            "Engine": "docdb",
            "EngineVersion": "5.0.0",
            "LatestRestorableTime": "2023-11-10T07:21:16.772000+00:00",
            "Port": 27017,
            "MasterUsername": "chimeraAdmin",
```

```
            "PreferredBackupWindow": "22:22-22:52",
            "PreferredMaintenanceWindow": "sun:03:01-sun:03:31",
            "ReadReplicaIdentifiers": [],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "sample-instance-1",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                },
                {
                    "DBInstanceIdentifier": "sample-instance-2",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                },

            ],
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-9084c2ec",
                    "Status": "active"
                }
            ],
            "HostedZoneId": "Z06853723JYKYBXTJ49RB",
            "StorageEncrypted": false,
            "DbClusterResourceId": "cluster-T4LGLANHVAPGQYYULWUDKLVQL4",
            "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-
cluster",
            "AssociatedRoles": [],
            "IAMDatabaseAuthenticationEnabled": false,
            "ClusterCreateTime": "2023-11-06T18:05:41.568000+00:00",
            "EngineMode": "provisioned",
            "DeletionProtection": false,
            "HttpEndpointEnabled": false,
            "CopyTagsToSnapshot": false,
            "CrossAccountClone": false,
            "DomainMemberships": [],
            "TagList": [],
            "StorageType": "iopt1",
            "AutoMinorVersionUpgrade": false,
            "NetworkType": "IPV4",
            "IOOptimizedNextAllowedModificationTime":
 "2023-12-07T18:05:41.580000+00:00"
```

```
        }
    ]
}
```

Example

Example 3: List specific details for a Amazon DocumentDB cluster

To list a subset of the clusters' details using the AWS CLI, add a `--query` that specifies which cluster members the `describe-db-clusters` operation is to list. The `--db-cluster-identifier` parameter is the identifier for the particular cluster that you want to display the details of. For more information on queries, see How to Filter the Output with the `--query` Option in the *AWS Command Line Interface User Guide*.

The following example lists the instances in an Amazon DocumentDB cluster.

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
    --filter Name=engine,Values=docdb \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

For Windows:

```
aws docdb describe-db-clusters ^
    --filter Name=engine,Values=docdb ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterMembers]'
```

Output from this operation looks something like the following.

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
```

```
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

Modifying an Amazon DocumentDB cluster

To modify a cluster, the cluster must be in the *available* state. You cannot modify a cluster that is stopped. If the cluster is stopped, first start the cluster, wait for the cluster to become *available*, and then make the desired modifications. For more information, see Stopping and starting an Amazon DocumentDB cluster.

Using the AWS Management Console

Use the following procedure to modify a specific Amazon DocumentDB cluster using the console.

To modify an Amazon DocumentDB cluster

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. Specify the cluster that you want to modify by choosing the button to the left of the cluster's name.
4. Choose **Actions**, and then choose **Modify**.

5. In the **Modify Cluster: <cluster-name>** pane, make the changes that you want. You can make changes in the following areas:

   - **Cluster specifications**—The cluster's name, security groups, and credentials management.
   - **Cluster storage configuration**—The cluster's data storage mode. Choose between Standard and I/O-Optimized configuration.
   - **Cluster options**—The cluster's port and parameter group.
   - **Backup**—The cluster's backup retention period and backup window.
   - **Log exports**—Enable or disable exporting audit or profiler logs.
   - **Maintenance**—Set the cluster's maintenance window.
   - **Deletion protection**—Enable or disable deletion protection on the cluster. Deletion protection is enabled by default.
6. When you're finished, choose **Continue** to view a summary of your changes.
7. If you are satisfied with your changes, you can choose **Modify cluster** to modify your cluster. Alternatively, you can choose **Back** or **Cancel** to edit or cancel your changes, respectively.

It takes a few minutes for your changes to be applied. You can use the cluster only when its status is *available*. You can monitor the cluster's status using the console or AWS CLI. For more information, see Monitoring an Amazon DocumentDB cluster's status.

Using the AWS CLI

Use the `modify-db-cluster` operation to modify the specified cluster using the AWS CLI. For more information, see `ModifyDBCluster` in the *Amazon DocumentDB API Reference*.

Parameters

- **`--db-cluster-identifier`**—Required. The identifier of the Amazon DocumentDB cluster that you are going to modify.
- **`--backup-retention-period`**—Optional. The number of days for which automated backups are retained. Valid values are 1–35.
- **`--storage-type`**—Optional. The cluster's storage configuration. Valid values are `standard` (Standard) or `iopt1` (I/O-optimized).

- **`--db-cluster-parameter-group-name`**—Optional. The name of the cluster parameter group to use for the cluster.
- **`--manage-master-user-password`**—Optional. Amazon DocumentDB generates the master user password and manages it throughout its lifecycle in Secrets Manager.
- **`--rotate-master-user-password`**—Optional. Secrets Manager generates a new secret version for the existing secret. The new version of the secret contains the new primary user password. Amazon DocumentDB changes the primary user password for the cluster to match the password for the new secret version.

  You must specify the **`--apply-immediately`** option when you rotate the primary password.
- **`--master-user-password`**—Optional. The new password for the primary database user.

  Password constraints:
  - Length is [8—100] printable ASCII characters.
  - Can use any printable ASCII characters except for the following:
    - **`/`** (forward slash)
    - **`"`** (double quotation mark)
    - **`@`** (at symbol)
- **`--new-db-cluster-identifier`**—Optional. The new cluster identifier for the cluster when renaming a cluster. This value is stored as a lowercase string.

  Naming constraints:
  - Length is [1—63] letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all clusters across Amazon RDS, Amazon Neptune, and Amazon DocumentDB per AWS account, per region.
- **`--preferred-backup-window`**—Optional. The daily time range during which automated backups are created, in Universal Coordinated Time (UTC).
  - Format: `hh24:mm-hh24:mm`
- **`--preferred-maintenance-window`**—Optional. The weekly time range during which system maintenance can occur, in UTC.
  - Format: `ddd:hh24:mm-ddd:hh24:mm`
  - Valid days: `Sun`, `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, and `Sat`.

- **`--deletion-protection`** or **`--no-deletion-protection`**—Optional. Whether deletion protection should be enabled on this cluster. Deletion protection prevents a cluster from being accidentally deleted until the cluster is modified to disable deletion protection. For more information, see Deleting an Amazon DocumentDB cluster.
- **`--apply-immediately`** or **`--no-apply-immediately`**—Use `--apply-immediately` to make the change immediately. Use `--no-apply-immediately` to make the change during your cluster's next maintenance window.

Example

The following code changes the backup retention period for the cluster `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --apply-immediately \
       --backup-retention-period 7
```

For Windows:

```
aws docdb modify-db-cluster ^
       --db-cluster-identifier sample-cluster ^
       --apply-immediately ^
       --backup-retention-period 7
```

Output from this operation looks something like the following.

```
{
    "DBCluster": {
        "BackupRetentionPeriod": 7,
        "DbClusterResourceId": "cluster-VDP53QEWST7YHM36TTXOPJT5YE",
        "Status": "available",
        "DBClusterMembers": [
            {
                "PromotionTier": 1,
                "DBClusterParameterGroupStatus": "in-sync",
                "DBInstanceIdentifier": "sample-cluster-instance",
                "IsClusterWriter": true
```

```
            }
        ],
        "ReadReplicaIdentifiers": [],
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1a"
        ],
        "ReaderEndpoint": "sample-cluster.cluster-ro-ctevjxdlur57.us-
east-1.rds.amazonaws.com",
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "PreferredMaintenanceWindow": "sat:09:51-sat:10:21",
        "EarliestRestorableTime": "2018-06-17T00:06:19.374Z",
        "StorageEncrypted": false,
        "MultiAZ": false,
        "AssociatedRoles": [],
        "MasterUsername": "<your-master-user-name>",
        "DBClusterIdentifier": "sample-cluster",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "HostedZoneId": "Z2SUY0A1719RZT",
        "LatestRestorableTime": "2018-06-18T21:17:05.737Z",
        "AllocatedStorage": 1,
        "Port": 27017,
        "Engine": "docdb",
        "DBClusterParameterGroup": "default.docdb3.4",
        "Endpoint": "sample-cluster.cluster-ctevjxdlur57.us-
east-1.rds.amazonaws.com",
        "DBSubnetGroup": "default",
        "PreferredBackupWindow": "00:00-00:30",
        "EngineVersion": "3.4",
        "ClusterCreateTime": "2018-06-06T19:25:47.991Z",
        "IAMDatabaseAuthenticationEnabled": false
    }
}
```

It takes a few minutes for your changes to be applied. You can use the cluster only when its status is *available*. You can monitor the cluster's status using the console or AWS CLI. For more information, see Monitoring an Amazon DocumentDB cluster's status.

Determining pending maintenance

You can determine whether you have the latest Amazon DocumentDB engine version by determining whether you have pending cluster maintenance.

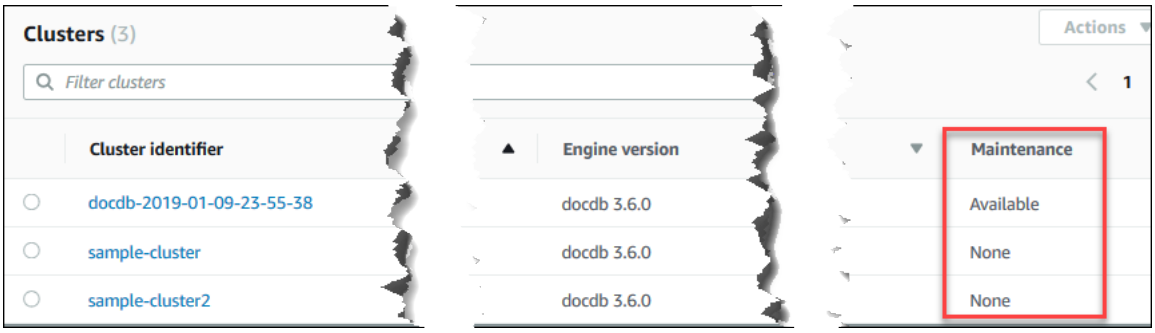
Using the AWS Management Console

You can use the AWS Management Console to determine whether a cluster has pending maintenance.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. Locate the **Maintenance** column to determine whether a cluster has pending maintenance.



None indicates that the cluster is running the latest engine version. **Available** indicates that the cluster has pending maintenance, which might mean that an engine upgrade is needed.

4. If your cluster has pending maintenance, continue with the steps at Performing a patch update to a cluster's engine version.

Using the AWS CLI

You can use the AWS CLI to determine whether a cluster has the latest engine version by using the `describe-pending-maintenance-actions` operation with the following parameters.

Parameters

- **`--resource-identifier`**—Optional. The ARN for the resource (cluster). If this parameter is omitted, pending maintenance actions for all clusters are listed.
- **`--region`**—Optional. The AWS Region that you want to run this operation in, for example, `us-east-1`.

Example

For Linux, macOS, or Unix:

```
aws docdb describe-pending-maintenance-actions \
   --resource-identifier arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster \
   --region us-east-1
```

For Windows:

```
aws docdb describe-pending-maintenance-actions ^
   --resource-identifier arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster ^
   --region us-east-1
```

Output from this operation looks something like the following.

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:123456789012:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Description": "New feature",
                    "Action": "db-upgrade",
                    "ForcedApplyDate": "2019-02-25T21:46:00Z",
```

```
                    "AutoAppliedAfterDate": "2019-02-25T07:41:00Z",
                    "CurrentApplyDate": "2019-02-25T07:41:00Z"
                }
            ]
        }
    ]
}
```

If your cluster has pending maintenance, continue with the steps at Performing a patch update to a cluster's engine version.

Performing a patch update to a cluster's engine version

In this section, we will explain how to deploy a patch update using the AWS Management Console or the AWS CLI. A patch update is an update within the same engine version (for example, updating a 3.6 engine version to a newer 3.6 engine version). You can update it immediately or during your cluster's next maintenance window. To determine whether your engine needs an update, see Determining pending maintenance. Please note that when you apply the update, your cluster will experience some downtime.

> **Note**
>
> If you are trying to upgrade from a major engine version to another, such as 3.6 to 5.0, see either Amazon DocumentDB in-place major version upgrade or Upgrading your Amazon DocumentDB cluster using AWS Database Migration Service. An in-place major version upgrade only supports docdb 5.0 as the target engine version.

There are two configuration requirements to get the latest patch updates for a cluster's engine version:

- The cluster's status must be *available*.
- The cluster must be running an earlier engine version.

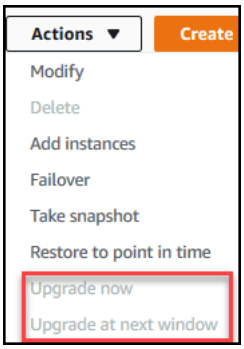
Using the AWS Management Console

The following procedure applies patch updates to your cluster's engine version using the console. You have the option to update immediately or during your cluster's next maintenance window.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**. In the list of clusters, choose the button to the left of the cluster that you want to upgrade. The status of the cluster must be *available*.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. From the **Actions** menu, choose one of the following options. These menu options are selectable only if the cluster you chose is not running the latest engine version.



   - **Upgrade now**—Immediately initiates the upgrade process. Your cluster will be offline for a time while the cluster is upgraded to the latest engine version.
   - **Upgrade at next window**—Initiates the upgrade process during the cluster's next maintenance window. Your cluster will be offline for a time while it is upgraded to the latest engine version.
4. When the confirmation window opens, choose one of the following:
   - **Upgrade**—To upgrade your cluster to the latest engine version according to the schedule chosen in the previous step.

- **Cancel**—To cancel the cluster's engine upgrade and continue with the cluster's current engine version.

Using the AWS CLI

You can apply patch updates to your cluster using the AWS CLI and the `apply-pending-maintenance-action` operation with the following parameters.

Parameters

- **`--resource-identifier`**—Required. The ARN of the Amazon DocumentDB cluster that you are going to upgrade.
- **`--apply-action`**—Required. The following values are permitted. To upgrade your cluster engine version, use `db-upgrade`.
  - **`db-upgrade`**
  - **`system-update`**
- **`--opt-in-type`**—Required. The following values are permitted.
  - `immediate`—Apply the maintenance action immediately.
  - `next-maintenance`—Apply the maintenance action during the next maintenance window.
  - `undo-opt-in`—Cancel any existing `next-maintenance` opt-in requests.

Example

The following example patch updates the engine version of `sample-cluster` to version 4.0.0.

For Linux, macOS, or Unix:

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-east-1:123456789012\:cluster:sample-cluster
 \
   --apply-action db-upgrade \
   --opt-in-type immediate
```

For Windows:

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster ^
```

```
    --apply-action db-upgrade ^
    --opt-in-type immediate
```

Output from this operation looks like the following.

```
{
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:us-
east-1:444455556666:cluster:docdb-2019-01-09-23-55-38",
        "PendingMaintenanceActionDetails": [
            {
                "CurrentApplyDate": "2019-02-20T20:57:06.904Z",
                "Description": "Bug fixes",
                "ForcedApplyDate": "2019-02-25T21:46:00Z",
                "OptInStatus": "immediate",
                "Action": "db-upgrade",
                "AutoAppliedAfterDate": "2019-02-25T07:41:00Z"
            }
        ]
    }
}
```

Stopping and starting an Amazon DocumentDB cluster

Stopping and starting Amazon DocumentDB clusters can help you manage costs for development and test environments. Instead of creating and deleting clusters and instances each time you use Amazon DocumentDB, you can temporarily stop all the instances in your cluster when they aren't needed. You can then start them again when you resume your testing.

Topics

- Overview of stopping and starting a cluster
- Operations you can perform on a stopped cluster

Overview of stopping and starting a cluster

During periods where you don't need an Amazon DocumentDB cluster, you can stop all instances in that cluster at once. You can then start the cluster again anytime you need to use it. Starting and stopping simplifies the setup and teardown processes for clusters that are used for development,

testing, or similar activities that don't require continuous availability. You can stop and start a cluster using the AWS Management Console or the AWS CLI with a single action, regardless of how many instances are in the cluster.

While your cluster is stopped, the cluster storage volume remains unchanged. You are charged only for storage, manual snapshots, and automated backup storage within your specified retention window. You aren't charged for any instance hours. Amazon DocumentDB automatically starts your cluster after seven days so that it doesn't fall behind any required maintenance updates. When your cluster starts after seven days, you will begin to be charged for the instances in the cluster again. While your cluster is stopped, you can't query your storage volume because querying requires that instances are in the available state.

When an Amazon DocumentDB cluster is stopped, neither the cluster nor its instances can be modified in any way. This includes adding or removing instances, or deleting the cluster.

Using the AWS Management Console

The following procedure shows you how to stop a cluster with one or more instances in the available state, or start a stopped cluster.

To stop or start an Amazon DocumentDB cluster

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

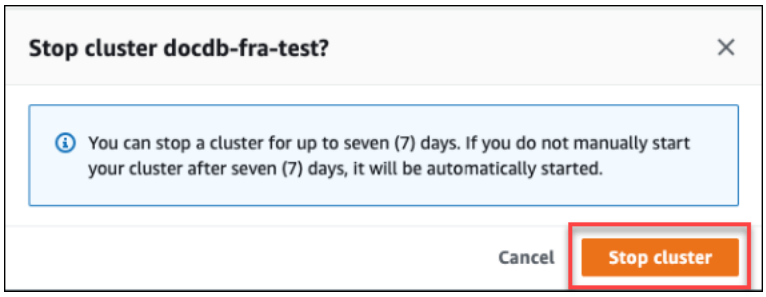
   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the list of clusters, choose the button to the left of the name of the cluster that you want to stop or start.
4. Choose **Actions**, and then choose the action that you want to perform on the cluster.
   - If you want to stop the cluster and the cluster is available:

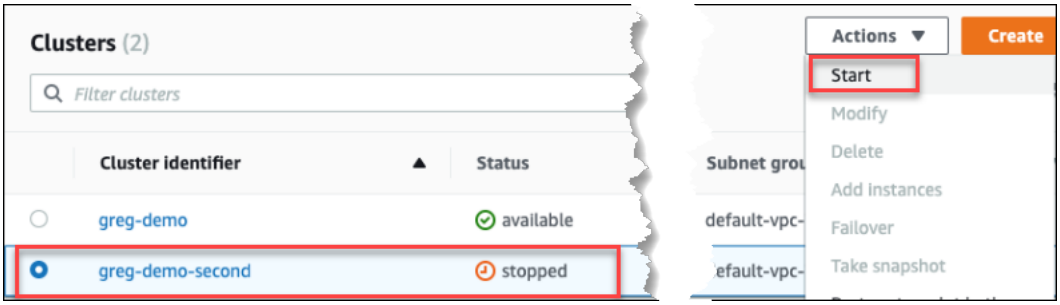
     a. Choose **Stop**.

To avoid activating the failover mechanism, the stop operation stops the replica instances first, and then the primary instance.

b. On the confirmation dialog, confirm that you want to stop the cluster by choosing **Stop cluster**, or to keep the cluster running, choose **Cancel**.



- If you want to start the cluster, and the cluster is stopped, choose **Start**.



5. Monitor the status of the cluster and its instances. If you started the cluster, you can resume using the cluster when the cluster and its instances are *available*. For more information, see Determining a cluster's status.

Using the AWS CLI

The following code examples show you how to stop a cluster with one or more instances in the available state, or start a stopped cluster.

To stop a cluster with one or more available instances using the AWS CLI, use the `stop-db-cluster` operation. To start a stopped cluster, use the `start-db-cluster` operation. Both operations use the `--db-cluster-identifier` parameter.

Parameter:

- **`--db-cluster-identifier`**—Required. The name of the cluster to stop or start.

Example — To stop a cluster using the AWS CLI

The following code stops the cluster `sample-cluster`. The cluster must have one or more instances in the available state.

For Linux, macOS, or Unix:

```
aws docdb stop-db-cluster \
   --db-cluster-identifier sample-cluster
```

For Windows:

```
aws docdb stop-db-cluster ^
   --db-cluster-identifier sample-cluster
```

Example — To start a cluster using the AWS CLI

The following code starts the cluster `sample-cluster`. The cluster must currently be stopped.

For Linux, macOS, or Unix:

```
aws docdb start-db-cluster \
   --db-cluster-identifier sample-cluster
```

For Windows:

```
aws docdb start-db-cluster ^
   --db-cluster-identifier sample-cluster
```

Operations you can perform on a stopped cluster

While an Amazon DocumentDB cluster is stopped, you can do a point-in-time restore to any point within your specified automated backup retention window. For details about doing a point-in-time restore, see Restoring to a point in time.

You can't modify the configuration of an Amazon DocumentDB cluster, or any of its instances, while the cluster is stopped. You also can't add or remove instances from the cluster, or delete the cluster if it still has any associated instances. You must start the cluster before performing any such administrative actions.

Amazon DocumentDB applies any scheduled maintenance to your stopped cluster only after it's started again. After seven days, Amazon DocumentDB automatically starts a stopped cluster so that it doesn't fall too far behind in its maintenance status. When the cluster restarts, you will begin to be charged for the instances in the cluster again.

While a cluster is stopped, Amazon DocumentDB does not perform any automated backups nor does it extend the backup retention period.

Deleting an Amazon DocumentDB cluster

You can delete an Amazon DocumentDB cluster using the AWS Management Console or the AWS CLI. To delete a cluster, the cluster must be in the *available* state and must not have any instances associated with it. If the cluster is stopped, first start the cluster, wait for the cluster to become *available*, and then delete the cluster. For more information, see Stopping and starting an Amazon DocumentDB cluster.

Deletion protection

To protect your cluster from accidental deletion, you can enable *deletion protection*. Deletion protection is enabled by default when you create a cluster using the console. However, deletion protection is disabled by default if you create a cluster using the AWS CLI.

Amazon DocumentDB enforces deletion protection for a cluster whether you perform the delete operation using the console or the AWS CLI. If deletion protection is enabled, you can't delete a cluster. To delete a cluster that has deletion protection enabled, you must first modify the cluster and disable deletion protection.

When using the console with deletion protection enabled on a cluster, you can't delete the cluster's last instance because doing so also deletes the cluster. You can delete the last instance of a

deletion protected cluster using the AWS CLI. However, the cluster itself still exists, and your data is preserved. You can access the data by creating new instances for the cluster. For more information about enabling and disabling deletion protection, see:

- Creating an Amazon DocumentDB cluster
- Modifying an Amazon DocumentDB cluster

Using the AWS Management Console

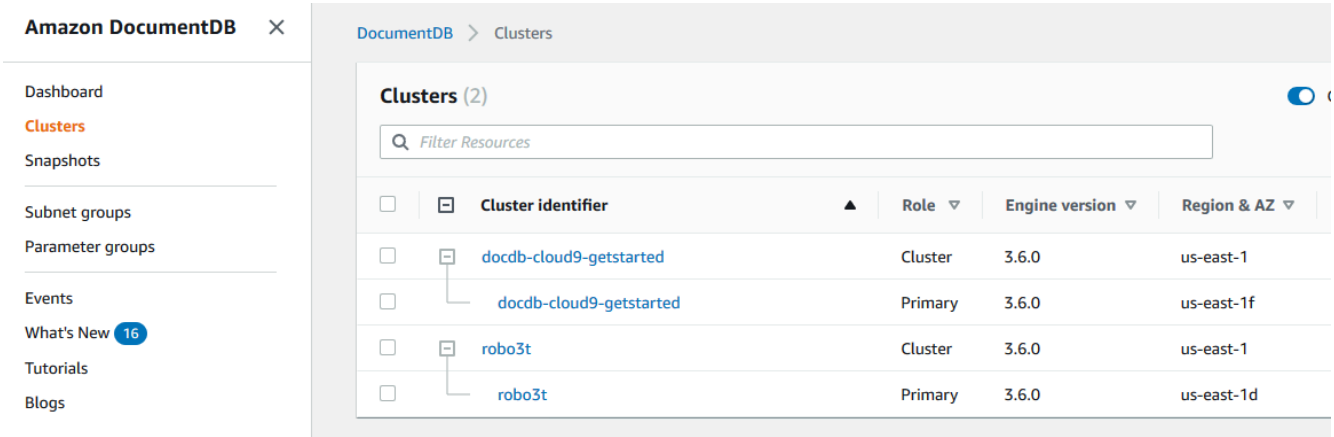
To delete a cluster using the AWS Management Console, deletion protection must be disabled.

To determine whether a cluster has deletion protection enabled:

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. Note that in the Clusters navigation box, the **Cluster identifier** column shows both clusters and instances. Instances are listed underneath clusters, similar to the screenshot below.



4. Choose the cluster's name, and select the **Configuration** tab. In the **Cluster details** section, locate **Deletion protection**. If deletion protection is enabled, modify the cluster to disable

deletion protection. For information about modifying a cluster, see Modifying an Amazon DocumentDB cluster.

After **Deletion protection** is disabled, you are ready to delete the cluster.

To delete a cluster:

1. In the navigation pane, choose **Clusters**.
2. Determine whether the cluster has any instances by checking the **Cluster identifier** column for instances listed below it. Before you can delete a cluster, you must delete all of its instances. For more information, see Deleting an Amazon DocumentDB instance.
3. Depending on whether your cluster has any instances, do one of the following steps.
   - If the cluster has no instances, select the button to the left of the cluster name and choose **Actions**. From the dropdown menu, choose **Delete**. Complete the **Delete <cluster-name>** dialog box, and then choose **Delete**.
   - If the cluster has one or more instances, do the following:
     a. In the navigation pane, choose **Clusters**.
     b. Delete each of the cluster's instances by selecting the checkbox to the left of the cluster's name. Select **Actions**, and then choose **Delete**. Complete the **Delete <cluster-name>** dialog box, and then choose **Delete**.

     When you delete the last instance, the cluster will also be deleted. For more information about deleting instances, see Deleting an Amazon DocumentDB instance.

It takes several minutes for the cluster to be deleted. To monitor the status of the cluster, see Monitoring an Amazon DocumentDB cluster's status.

Using the AWS CLI

You cannot delete a cluster that has any instances associated with it. To determine which instances are associated with your cluster, run the `describe-db-clusters` command and delete all of the cluster's instances. Then, if needed, disable deletion protection on your cluster, and finally, delete the cluster.

1. **First, delete all of the cluster's instances.**

To determine which instances you need to delete, run the following command.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].
[DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
```

Output from this operation looks something like the following (JSON format).

```
[
    [
        "sample-cluster",
        [
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
]
```

If the cluster you want to delete has any instances, delete them as shown below.

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

2. **Second, disable deletion protection.**

   Using the AWS CLI to delete all of a cluster's instances does not delete the cluster. You must also delete the cluster, but you can do this only if deletion protection is disabled.

   To determine whether the cluster has deletion protection enabled, run the following command.

   > **Tip**
   >
   > To see the deletion protection status of all your Amazon DocumentDB clusters, omit the `--db-cluster-identifier` parameter.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DeletionProtection]'
```

Output from this operation looks something like the following.

```
[
    [
        "sample-cluster",
        "true"
    ]
]
```

If the cluster has deletion protection enabled, modify the cluster and disable deletion protection. To disable deletion protection on the cluster, run the following command.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --no-deletion-protection \
    --apply-immediately
```

3. **Finally, delete the cluster.**

   After deletion protection is disabled, you are ready to delete the cluster. To delete a cluster, use the `delete-db-cluster` operation with the following parameters.

   - **`--db-cluster-identifier`**—Required. The identifier of the cluster that you want to delete.
   - **`--final-db-snapshot-identifier`**—Optional. If you want a final snapshot, you must include this parameter with a name for the final snapshot. You must include either `--final-db-snapshot-identifier` or `--skip-final-snapshot`.

     Naming constraints:

     - Length is [1—63] letters, numbers, or hyphens.
     - First character must be a letter.

  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all clusters across Amazon RDS, Amazon Neptune, and Amazon DocumentDB per AWS account, per Region.
- **`--skip-final-snapshot`**—Optional. Use this parameter only if you don't want to take a final snapshot before deleting your cluster. The default setting is to take a final snapshot. You must include either `--final-db-snapshot-identifier` or `--skip-final-snapshot`.

The following AWS CLI code deletes the cluster `sample-cluster` with a final snapshot. The operation fails if there are any instances associated with the cluster or if deletion protection is enabled.

Example

For Linux, macOS, or Unix:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

For Windows:

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Example

The following AWS CLI code deletes the cluster `sample-cluster` without taking a final snapshot.

For Linux, macOS, or Unix:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

For Windows:

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --skip-final-snapshot
```

The output of the `delete-db-cluster` operation is the cluster you are deleting.

It takes several minutes for the cluster to be deleted. To monitor the status of the cluster, see Monitoring a cluster's status.

Scaling Amazon DocumentDB clusters

Amazon DocumentDB enables you to scale the storage and compute in your clusters based on your needs. This section describes how you can use storage scaling, instance scaling, and read scaling to manage performance and scaling for your Amazon DocumentDB clusters and instances.

Topics

- Storage scaling
- Instance scaling
- Read scaling
- Write scaling

Storage scaling

Amazon DocumentDB storage automatically scales with the data in your cluster volume. As your data grows, your cluster volume storage grows in 10 GiB increments, up to 128 TiB.

Instance scaling

You can scale your Amazon DocumentDB cluster as needed by modifying the instance class for each instance in the cluster. Amazon DocumentDB supports several instance classes that are optimized for Amazon DocumentDB.

For more information, see Modifying an Amazon DocumentDB instance.

Read scaling

You can achieve read scaling for your Amazon DocumentDB cluster by creating up to 15 Amazon DocumentDB replicas in the cluster. Each Amazon DocumentDB replica returns the same data from the cluster volume with minimal replica lag—usually less than 100 milliseconds after the primary instance has written an update. As your read traffic increases, you can create additional Amazon DocumentDB replicas and connect to them directly to distribute the read load for your cluster. Amazon DocumentDB replicas don't have to be of the same instance class as the primary instance.

For more information, see Adding an Amazon DocumentDB instance to a cluster.

To read scale with Amazon DocumentDB, we recommend that you connect to your cluster as a replica set and distribute reads to replica instances using the built-in read preference capabilities of your driver. For more information, please see Connecting to Amazon DocumentDB as a replica set.

Write scaling

You can scale write capacity on your Amazon DocumentDB cluster by increasing the size of your cluster's primary instance. This section provides two methods for scaling your cluster's primary instance based on your needs. The first option seeks to minimize application impact but requires more steps to complete. The second option optimizes for simplicity as it has fewer steps, but it comes with the tradeoff of having more potential impact to your application.

Depending on your application, you can choose what approach below is best for you. For more information about available instance sizes and costs, see the Amazon DocumentDB Pricing page.

1. **Optimize for high availability and performance** — If you are connecting to your cluster in replica set mode (recommended), you can use the following process to minimize the impact to your application when scaling your primary instance. This method minimizes impact because it keeps your cluster at or above your high availability, and read scaling targets are added to the cluster as instances, instead of being updated in place.

    a. Add one or more replicas of the larger instance type to your cluster (see ???). We recommend all replicas be of the same or larger instance type as the primary. This avoids an unintentional reduction in write performance from failing over to a smaller instance type. For most customers, this means temporarily doubling the number of instances in their cluster, then removing the smaller replicas after scaling is complete.

    b. Set the failover tier on all new replicas to priority zero, ensuring a replica of the smaller instance type has the highest failover priority. For more information, see ???.

c. Initiate a manual failover, which will promote one of the new replicas to be the primary instance. For more information, see ???.

> **Note**
>
> This will incur ~30 seconds of downtime for your cluster. Please plan accordingly.

d. Remove all replicas of an instance type smaller than your new primary from the cluster.

e. Set the failover tier of all instances back to the same priority (usually, this means setting them back to 1).

As an example, suppose that you have a cluster that currently contains three `r5.large` instances (one primary and two replicas), and you want to scale to an `r5.xlarge` instance type. To do so, you would first add three `r5.xlarge` replica instances to your cluster and then set the failover tier of the new `r5.xlarge` replicas to zero. Next, you would initiate a manual failover (understanding that your application will experience ~30 seconds of downtime). Once the failover is complete, you would remove all three `r5.large` instances from your cluster, leaving the cluster scaled to `r5.xlarge` instances.

To help optimize costs, Amazon DocumentDB instances are billed in one second increments, with a ten minute minimum charge following a billable status change such as creating, modifying, or deleting an instance. For more information, see Cost optimization in the best practices documentation.

2. **Optimize for simplicity** — This approach optimizes for simplicity. It doesn't expand and contract the cluster, but it might temporarily reduce your read capacity.

   It is possible that changing the instance class of a replica will result in that instance not serving requests for a brief period of time, from a few seconds to less than 30 seconds. If you are connecting to your cluster in replica set mode (recommended), then this would reduce your read capacity by one replica (e.g., to 66% capacity in a 3-node cluster, or 75% capacity in a 4-node cluster, etc.) during the scaling operation.

   a. Scale one of the replica instances in your cluster. For more information, see Managing instance classes.

   b. Wait until the instance is available (see Monitoring an Amazon DocumentDB instance's status).

> **Note**
>
> This will incur ~30 seconds of downtime for your cluster. Please plan accordingly.

c. Continue executing steps 1 and 2 until all replicas instances have been scaled, one by one.

d. Initiate a manual failover. This will promote one of the replicas to be the primary instance. For more information, see Amazon DocumentDB Failover.

> **Note**
>
> This will incur up to 30 seconds of downtime for your cluster, but often takes less time than that. Please plan accordingly.

e. Scale the former primary (now a replica) instance.

Cloning a volume for an Amazon DocumentDB cluster

By using Amazon DocumentDB cloning, you can create a new cluster that uses the same Amazon DocumentDB cluster volume and has the same data as the original. The process is designed to be fast and cost-effective. The new cluster with its associated data volume is known as a *clone*. Creating a clone is faster and more space-efficient than physically copying the data using other techniques, such as restoring a snapshot.

Amazon DocumentDB supports creating an Amazon DocumentDB provisioned clone from a provisioned Amazon DocumentDB cluster. When you create a clone using a different deployment configuration than the source, the clone is created using the latest version of the source's Amazon DocumentDB engine.

When you create clones from your Amazon DocumentDB clusters, the clones are created in your AWS account—the same account that owns the source Amazon DocumentDB cluster.

Topics

- Overview of Amazon DocumentDB cloning
- Limitations of Amazon DocumentDB cloning
- How Amazon DocumentDB cloning works
- Creating an Amazon DocumentDB clone

Overview of Amazon DocumentDB cloning

Amazon DocumentDB uses a *copy-on-write protocol* to create a clone. This mechanism uses minimal additional space to create an initial clone. When the clone is first created, Amazon DocumentDB keeps a single copy of the data that is used by the source DB cluster and the new (cloned) Amazon DocumentDB cluster. Additional storage is allocated only when changes are made to data (on the Amazon DocumentDB storage volume) by the source Amazon DocumentDB cluster or the Amazon DocumentDB cluster clone. To learn more about the copy-on-write protocol, see How Amazon DocumentDB cloning works.

Amazon DocumentDB cloning is especially useful for quickly setting up test environments using your production data, without risking data corruption. You can use clones for many types of applications, such as the following:

- Experiment with potential changes (schema changes and parameter group changes, for example) to assess all impacts.
- Run workload-intensive operations, such as exporting data or running analytical queries on the clone.
- Create a copy of your production DB cluster for development, testing, or other purposes.

You can create more than one clone from the same Amazon DocumentDB cluster. You can also create multiple clones from another clone.

After creating an Amazon DocumentDB clone, you can configure the Amazon DocumentDB instances differently from the source Amazon DocumentDB cluster. For example, you might not need a clone for development purposes to meet the same high availability requirements as the source production Amazon DocumentDB cluster. In this case, you can configure the clone with a single Amazon DocumentDB instance rather than the multiple DB instances used by the Amazon DocumentDB cluster.

When you finish using the clone for your testing, development, or other purposes, you can delete it.

Limitations of Amazon DocumentDB cloning

Amazon DocumentDB; cloning currently has the following limitations:

- You can create as many clones as you want, up to the maximum number of DB clusters allowed in the AWS Region. However, after you create 15 clones, the next clone is a full copy. The cloning operation acts like a point-in-time recovery.
- You can't create a clone in a different AWS Region from the source Amazon DocumentDB cluster.
- You can't create a clone from an Amazon DocumentDB cluster that has no DB instances. You can only clone Amazon DocumentDB clusters that have at least one DB instance.
- You can create a clone in a different virtual private cloud (VPC) than that of the Amazon DocumentDB cluster. If you do, the subnets of the VPCs must map to the same Availability Zones.

How Amazon DocumentDB cloning works

Amazon DocumentDB cloning works at the storage layer of an Amazon DocumentDB cluster. It uses a *copy-on-write* protocol that's both fast and space-efficient in terms of the underlying durable media supporting the Amazon DocumentDB storage volume. You can learn more about Amazon DocumentDB cluster volumes in Managing Amazon DocumentDB clusters.

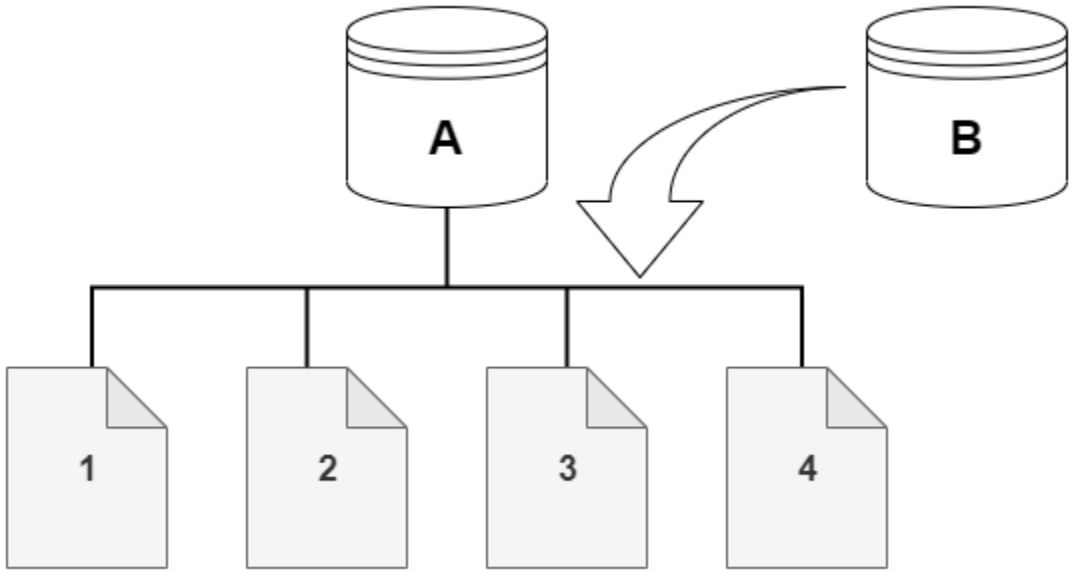
Topics

- Understanding the copy-on-write protocol
- Deleting a source cluster volume

Understanding the copy-on-write protocol

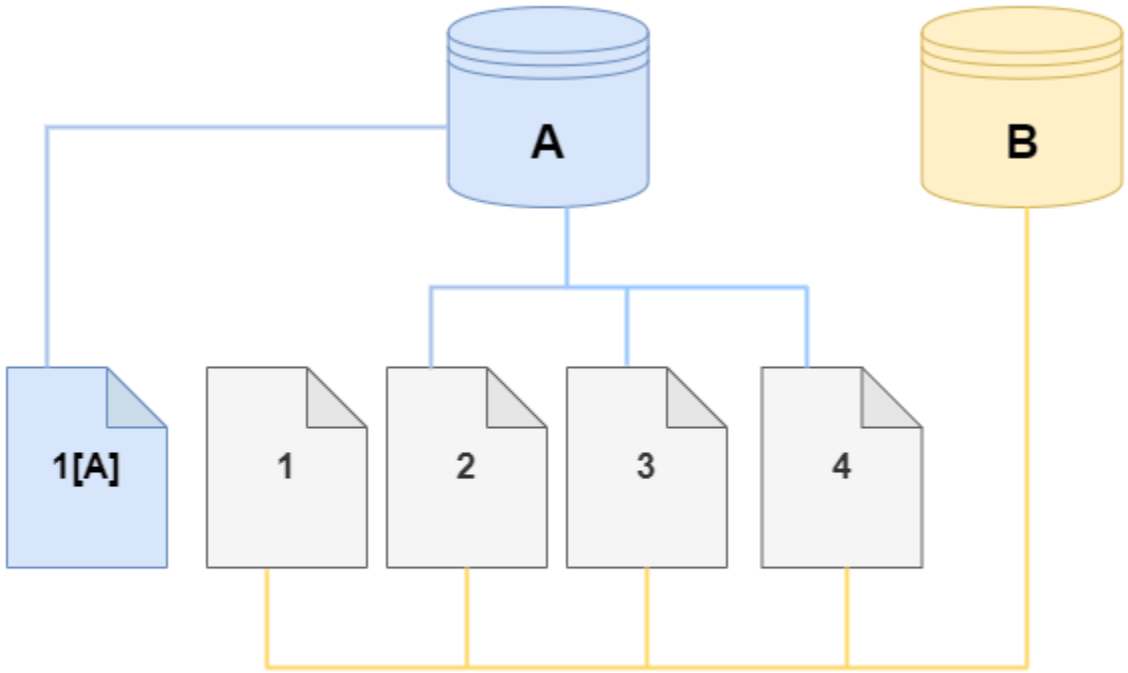
An Amazon DocumentDB cluster stores data in pages in the underlying Amazon DocumentDB storage volume.

For example, in the following diagram you can find an Amazon DocumentDB cluster (A) that has four data pages, 1, 2, 3, and 4. Imagine that a clone, B, is created from the Amazon DocumentDB cluster. When the clone is created, no data is copied. Rather, the clone points to the same set of pages as the source Amazon DocumentDB cluster.

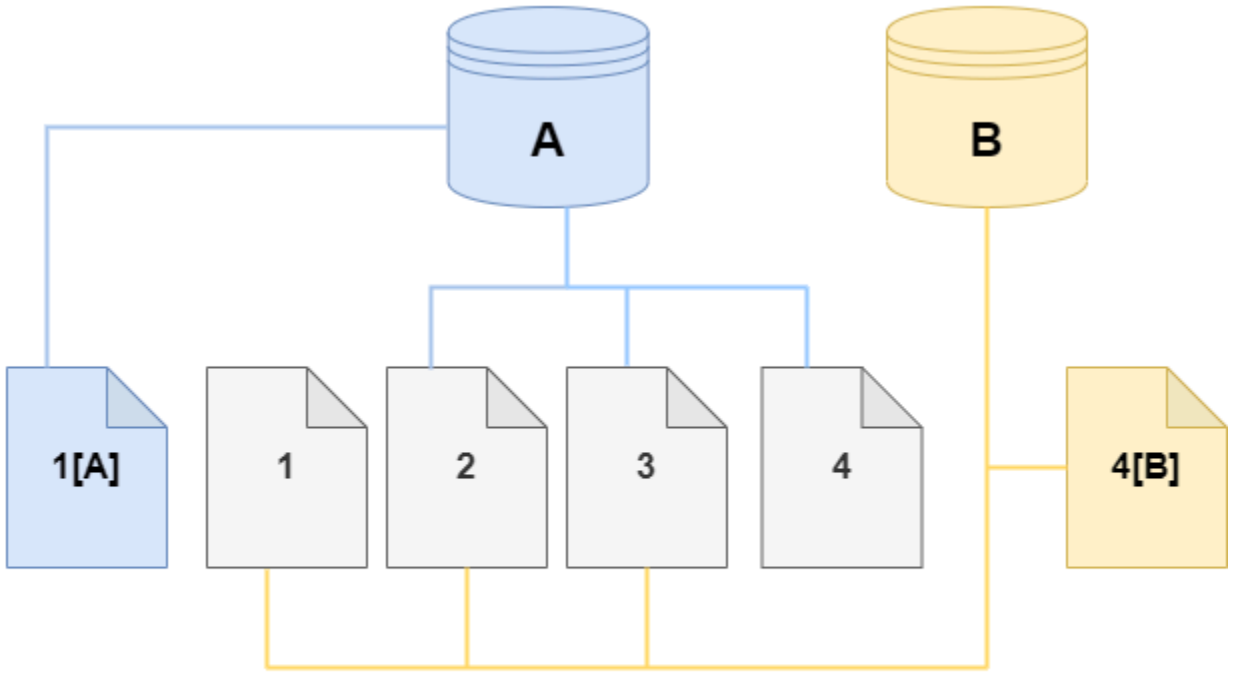


When the clone is created, no additional storage is usually needed. The copy-on-write protocol uses the same segment on the physical storage media as the source segment. Additional storage is required only if the capacity of the source segment isn't sufficient for the entire clone segment. If that's the case, the source segment is copied to another physical device.

In the following diagrams, you can find an example of the copy-on-write protocol in action using the same cluster A and its clone, B, as shown preceding. Let's say that you make a change to your Amazon DocumentDB cluster (A) that results in a change to data held on page 1. Instead of writing to the original page 1, Amazon DocumentDB creates a new page 1[A]. The Amazon DocumentDB cluster volume for cluster (A) now points to page 1[A], 2, 3, and 4, while the clone (B) still references the original pages.



On the clone, a change is made to page 4 on the storage volume. Instead of writing to the original page 4, Amazon DocumentDB creates a new page, 4[B]. The clone now points to pages 1, 2, 3, and to page 4[B], while the cluster (A) continues pointing to 1[A], 2, 3, and 4.



As more changes occur over time in both the source Amazon DocumentDB cluster volume and the clone, more storage is needed to capture and store the changes.

Deleting a source cluster volume

When you delete a source cluster volume that has one or more clones associated with it, the clones aren't affected. The clones continue to point to the pages that were previously owned by the source cluster volume.

Creating an Amazon DocumentDB clone

You can create a clone in the same AWS account as the source Amazon DocumentDB cluster. To do so, you can use the AWS Management Console or the AWS CLI and the procedures following.

By using Amazon DocumentDB cloning, you can create a provisioned Amazon DocumentDB cluster clone from a provisioned Amazon DocumentDB cluster.

Using the AWS Management Console

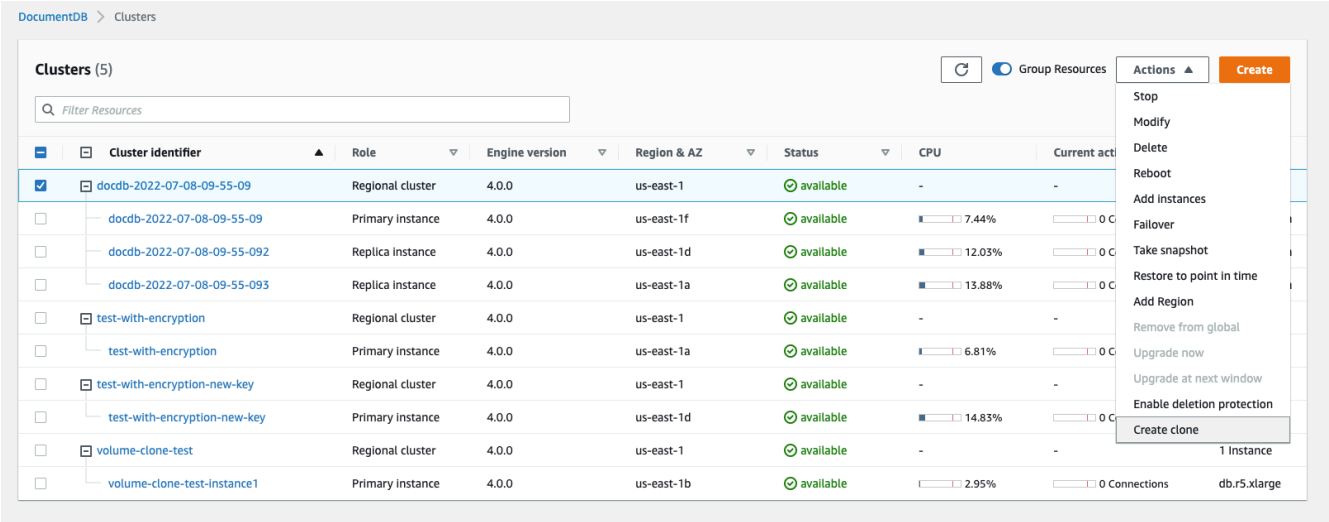
The following procedure describes how to clone an Amazon DocumentDB cluster using the AWS Management Console.

Creating a clone using the AWS Management Console results in an Amazon DocumentDB cluster with one Amazon DocumentDB instance.

These instructions apply for DB clusters owned by the same AWS account that is creating the clone. The DB cluster must be owned by the same AWS account as cross-account cloning is not supported in Amazon DocumentDB.

To create a clone of a DB cluster owned by your AWS account using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. Choose your Amazon DocumentDB cluster from the list, and for **Actions**, choose **Create clone**.

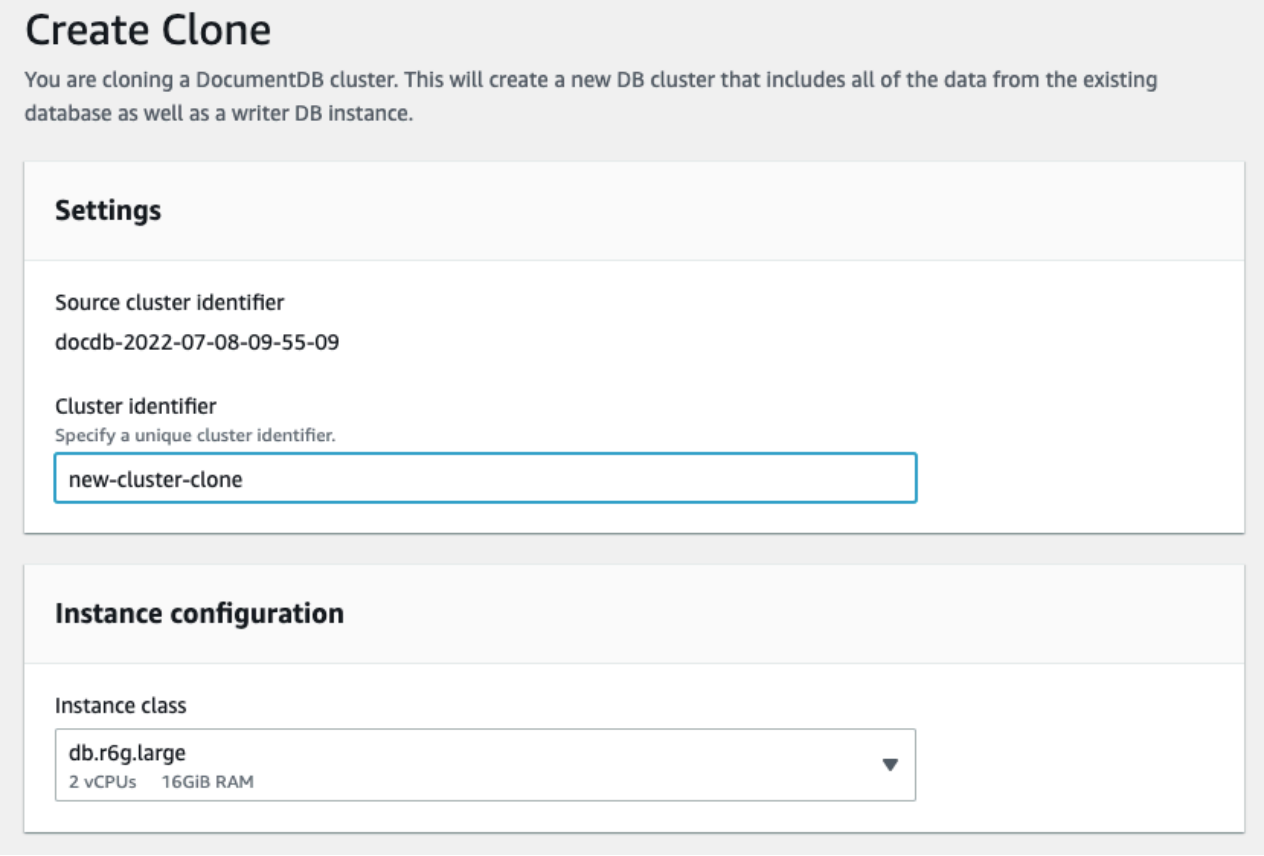


The Create clone page opens, where you can configure a **Cluster identifier** and an **Instance class**, and other options for the Amazon DocumentDB cluster clone.

4. In the **Settings** section, do the following:

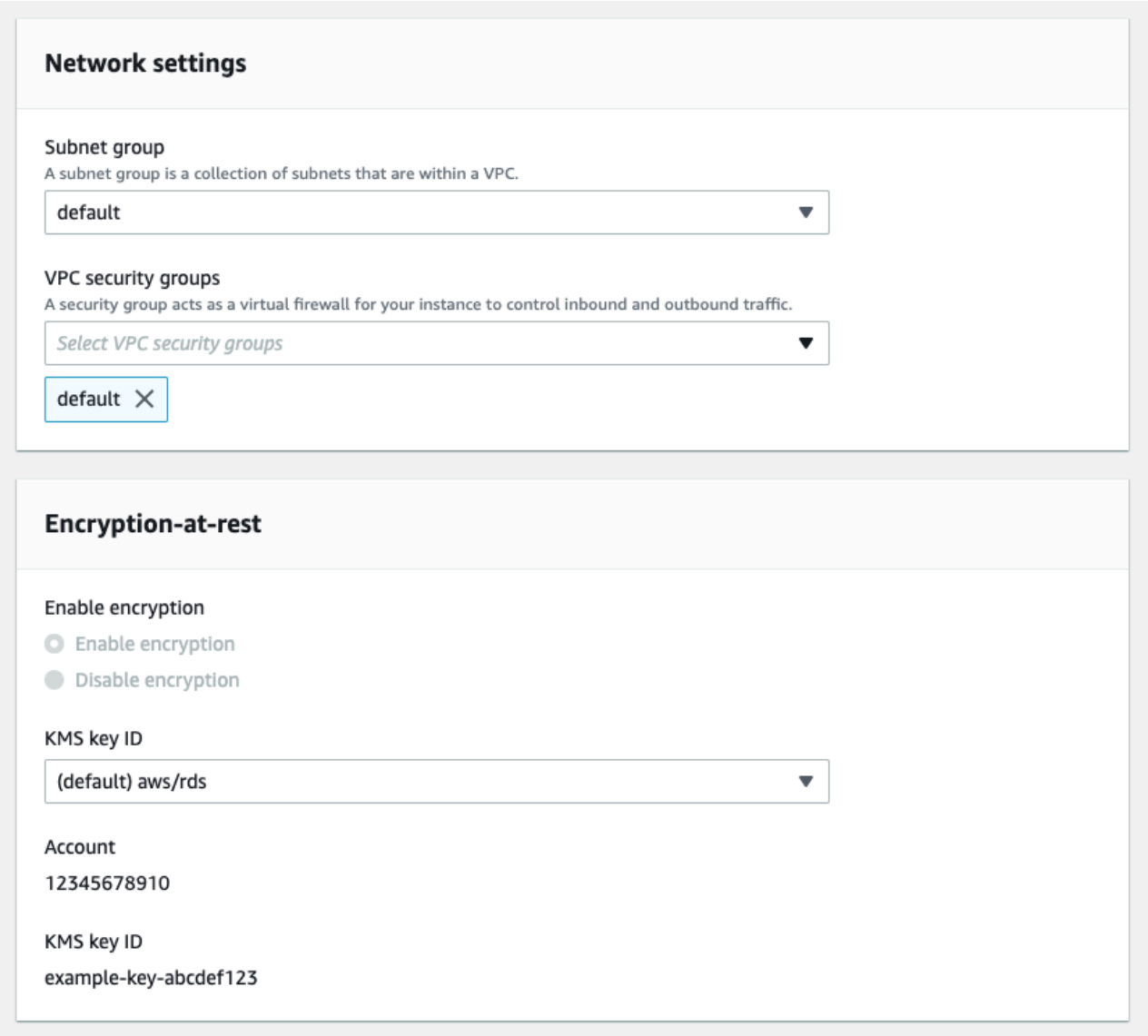
   a. For **Cluster identifier**, enter the name that you want to give to your cloned Amazon DocumentDB cluster.

b. For **Instance configuration**, select an appropriate **Instance class** for your cloned Amazon DocumentDB cluster.

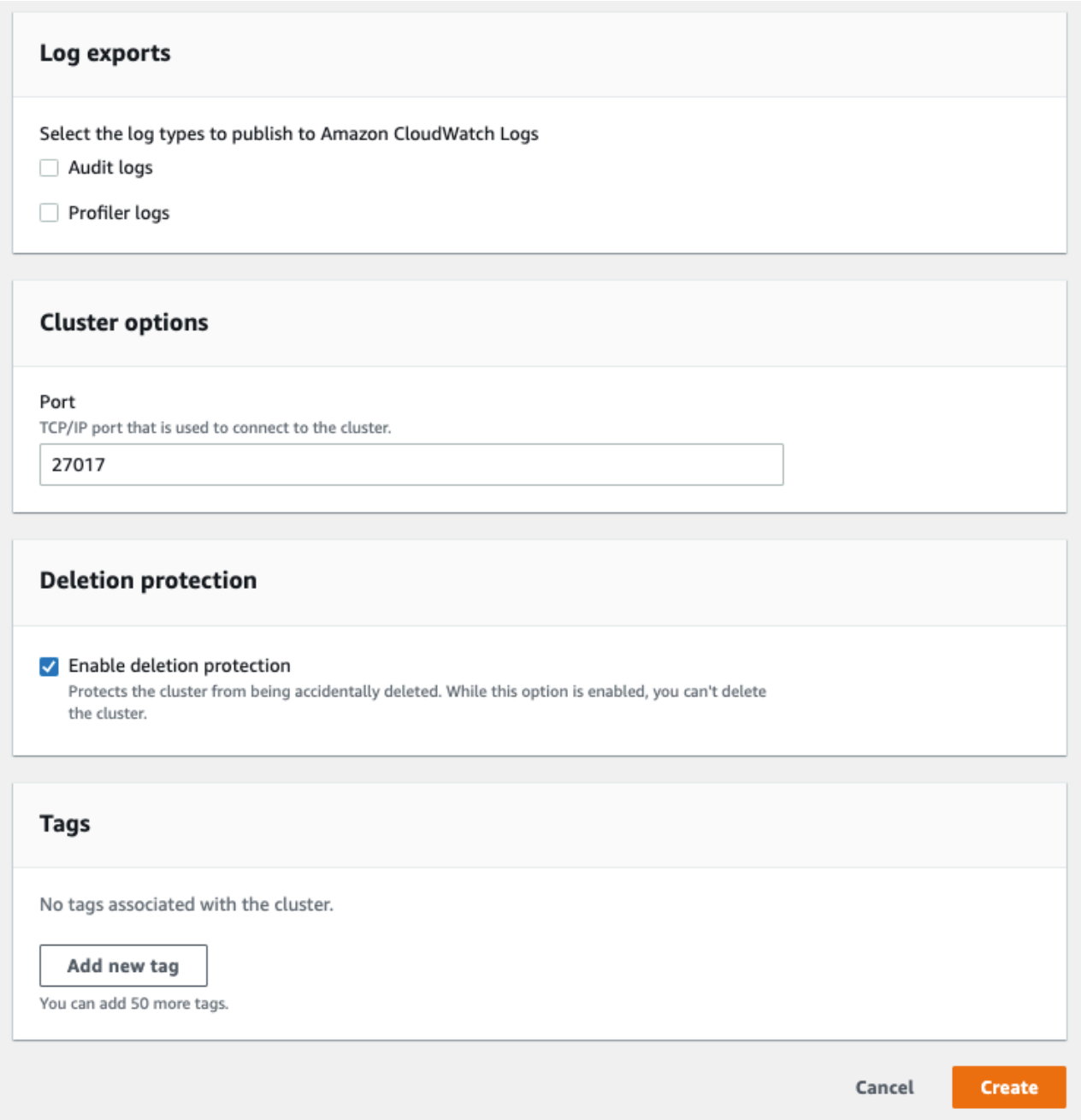


c. For **Network settings**, choose a **Subnet group** for your use case and the associated VPC security groups.

d. For **Encryption-at-rest**, if the source cluster (the cluster that is being cloned) has encryption enabled, the cloned cluster must also have encryption enabled. If this scenario is true, then the **Enable encryption** options are grayed out (disabled) but with the **Enable encryption** choice selected. Conversely, if the source cluster does not have encryption enabled, the **Enable encryption** options are available and you can choose to enable or disable encryption.



e. Complete the new cluster clone configuration by selecting the type of logs to export (optional), entering a specific port used to connect to the cluster, and enabling protection from accidentally deleting the cluster (enabled by default).



f. Finish entering all settings for your Amazon DocumentDB cluster clone. To learn more about Amazon DocumentDB cluster and instance settings, see Managing Amazon DocumentDB clusters.

5. Choose **Create clone** to launch the Amazon DocumentDB clone of your chosen Amazon DocumentDB cluster.

When the clone is created, it's listed with your other Amazon DocumentDB clusters in the console **Databases** section and displays its current state. Your clone is ready to use when its state is **Available**.

Using the AWS CLI

Using the AWS CLI for cloning your Amazon DocumentDB cluster involves a couple of steps.

The `restore-db-cluster-to-point-in-time` AWS CLI command that you use results in an empty Amazon DocumentDB cluster with 0 Amazon DocumentDB instances. That is, the command restores only the Amazon DocumentDB cluster, not the DB instances for that cluster. You do that separately after the clone is available. The two steps in the process are as follows:

1. Create the clone by using the restore-db-cluster-to-point-in-time CLI command. The parameters that you use with this command control the capacity type and other details of the empty Amazon DocumentDB cluster (clone) being created.
2. Create the Amazon DocumentDB instance for the clone by using the create-db-instance CLI command to recreate the Amazon DocumentDB instance in the restored Amazon DocumentDB cluster.

The commands following assume that the AWS CLI is set up with your AWS Region as the default. This approach saves you from passing the `--region` name in each of the commands. For more information, see Configuring the AWS CLI. You can also specify the `--region` in each of the CLI commands that follow.

Creating the clone

The specific parameters that you pass to the `restore-db-cluster-to-point-in-time` CLI command vary. What you pass depends on the type of clone that you want to create.

Use the following procedure to create a provisioned Amazon DocumentDB clone from a provisioned Amazon DocumentDB cluster.

To create a clone of the same engine mode as the source Amazon DocumentDB cluster

- Use the `restore-db-cluster-to-point-in-time` CLI command and specify values for the following parameters:
  - `--db-cluster-identifier` – Choose a meaningful name for your clone. You name the clone when you use the restore-db-cluster-to-point-in-time CLI command.

- `--restore-type` – Use `copy-on-write` to create a clone of the source DB cluster. Without this parameter, the `restore-db-cluster-to-point-in-time` restores the Amazon DocumentDB cluster rather than creating a clone. Default for `restore-type` is `full-copy`.
- `--source-db-cluster-identifier` – Use the name of the source Amazon DocumentDB cluster that you want to clone.
- `--use-latest-restorable-time` – This value points to the latest restorable volume data for the clone. This parameter is required for `restore-type copy-on-write`, however, you can not use the `restore-to-time parameter` with it.

The following example creates a clone named `my-clone` from a cluster named `my-source-cluster`.

For Linux, macOS, or Unix:

```
aws docdb restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

For Windows:

```
aws docdb restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

The command returns the JSON object containing details of the clone. Check to make sure that your cloned DB cluster is available before trying to create the DB instance for your clone. For more information, see Checking the status and getting clone details below:

Checking the status and getting clone details

You can use the following command to check the status of your newly created empty DB cluster.

```
$ aws docdb describe-db-clusters --db-cluster-identifier my-clone --query '*[].
[Status]' --output text
```

Or you can obtain the status and the other values that you need to create the DB instance for your clone by using the following AWS CLI query:

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}'
```

For Windows:

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}"
```

This query returns output similar to the following.

```
[
  {
        "Status": "available",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
    }
]
```

Creating the Amazon DocumentDB instance for your clone

Use the create-db-instance CLI command to create the DB instance for your clone.

The `--db-instance-class` parameter is used for provisioned Amazon DocumentDB clusters only.

For Linux, macOS, or Unix:

```
aws docdb create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class  db.r5.4xlarge \
```

```
    --engine docdb
```

For Windows:

```
aws docdb create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class  db.r5.4xlarge ^
    --engine docdb
```

Parameters to use for cloning

The following table summarizes the various parameters used with `restore-db-cluster-to-point-in-time` to clone Amazon DocumentDB clusters.

Parameter	Description
--source-db-cluster-identifier	Use the name of the source Amazon DocumentDB cluster that you want to clone.
--db-cluster-identifier	Choose a meaningful name for your clone. You name your clone with the `restore-db-cluster-to-point-in-time` command. Then you pass this name to the `create-db-instance` command.
--restore-type	Specify `copy-on-write` as the `--restore-type` to create a clone of the source DB cluster rather than restoring the source Amazon DocumentDB cluster.
--use-latest-restorable-time	This value points to the latest restorable volume data for the clone.

Understanding Amazon DocumentDB cluster fault tolerance

Amazon DocumentDB clusters are fault tolerant by design. Each cluster's volume spans multiple Availability Zones in a single AWS Region, and each Availability Zone contains a copy of the cluster's volume data. This functionality means that your cluster can tolerate an Availability Zone failure without any loss of data and only a brief interruption of service.

If the primary instance in a cluster fails, Amazon DocumentDB automatically performs a failover to a new primary instance in one of two ways:

- By promoting an existing Amazon DocumentDB replica to the new primary instance chosen based on the Promotion Tier setting of each replica, and then creating a replacement for the former primary. A failover to the replica instance typically takes less than 30 seconds. Read and write operations may experience brief interruption during this period. To increase the availability of your cluster, we recommend that you create at least one or more Amazon DocumentDB replicas in two or more different Availability Zones.
- By creating a new primary instance. This only happens if you do not have a replica instance in your cluster and can take a few minutes to complete.

If the cluster has one or more Amazon DocumentDB replicas, an Amazon DocumentDB replica is promoted to the primary instance during a failure event. A failure event results in a brief interruption, during which read and write operations fail with an exception. However, service is typically restored in less than 120 seconds, and often less than 60 seconds. To increase the availability of your cluster, we recommend that you create at least one or more Amazon DocumentDB replicas in two or more different Availability Zones.

You can customize the order in which your Amazon DocumentDB replicas are promoted to the primary instance after a failure by assigning each replica a priority. Priorities range from 0 for the highest priority to 15 for the lowest priority. If the primary instance fails, the Amazon DocumentDB replica with the highest priority is promoted to the new primary instance. You can modify the priority of an Amazon DocumentDB replica at any time. Modifying the priority doesn't trigger a failover. You can use the `modify-db-instance` operation with the `--promotion-tier` parameter. For more information about customizing the failover priority of an instance, see Amazon DocumentDB Failover.

More than one Amazon DocumentDB replica can share the same priority, resulting in promotion tiers. If two or more Amazon DocumentDB replicas share the same priority, then the replica that is largest in size is promoted to primary. If two or more Amazon DocumentDB replicas share the same priority and size, an arbitrary replica in the same promotion tier is promoted.

If the cluster doesn't contain any Amazon DocumentDB replicas, the primary instance is re-created during a failure event. A failure event results in an interruption, during which read and write operations fail with an exception. Service is restored when the new primary instance is created, which typically takes less than 10 minutes. Promoting an Amazon DocumentDB replica to the primary instance is much faster than creating a new primary instance.

Managing Amazon DocumentDB instances

The following topics provide information to help you manage your Amazon DocumentDB instances. They include details about instance classes and statuses, and how to create, delete, and modify an instance.

Topics

- Determining an instance's status
- Amazon DocumentDB instance lifecycle
- Managing instance classes
- NVMe-backed instances

Determining an instance's status

To see the valid instance statuses, their meaning, and how to determine the status of your instances, see Monitoring an Amazon DocumentDB instance's status.

Amazon DocumentDB instance lifecycle

The lifecycle of an Amazon DocumentDB instance includes creating, modifying, maintaining and upgrading, performing backups and restores, rebooting, and deleting the instance. This section provides information about how to complete these processes.

Topics

- Adding an Amazon DocumentDB instance to a cluster
- Describing Amazon DocumentDB instances
- Modifying an Amazon DocumentDB instance
- Rebooting an Amazon DocumentDB instance
- Deleting an Amazon DocumentDB instance

You can create a new Amazon DocumentDB instance using the AWS Management Console or the AWS CLI. To add an instance to a cluster, the cluster must be in an *available* state. You cannot add an instance to a cluster that is stopped. If the cluster is stopped, first start the cluster, wait for the cluster to become *available*, and then add an instance. For more information, see Stopping and starting an Amazon DocumentDB cluster.

> **Note**
>
> If you create an Amazon DocumentDB cluster using the console, an instance is automatically created for you at the same time. If you want to create additional instances, use one of the following procedures.

Adding an Amazon DocumentDB instance to a cluster

Using the AWS Management Console

Use the following procedure to create an instance for your cluster using the Amazon DocumentDB console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. To choose the cluster that you want to add an instance to, select the button to the left of the cluster's name.
4. Choose **Actions**, and then choose **Add instances**.
5. In the **Add instance to: <cluster-name>** page, repeat the following steps for each instance that you want to add to the cluster. You can have up to 15.

   a. **Instance identifier**— You can either enter a unique identifier for this instance or allow Amazon DocumentDB to provide the instance identifier based on the cluster identifier.

      Instance naming constraints:

      - Length is [1—63] letters, numbers, or hyphens.
      - First character must be a letter.

- Cannot end with a hyphen or contain two consecutive hyphens.
- Must be unique for all instances across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.

b. **Instance class** — From the drop-down list, choose the instance type you want for this instance.

c. **Promotion tier** — From the drop-down list, choose the promotion tier for your instance or choose *No preference* to allow Amazon DocumentDB to set the promotion tier for your instance. Lower numbers mean higher priority. For more information, see Controlling the failover target.

d. To add more instances, choose **Add additional instances** and repeat steps a, b, and c.

6. Finish the operation.

   - To add the instances to your cluster, choose **Create**.
   - To cancel the operation, choose **Cancel**.

It takes several minutes to create an instance. You can use the console or AWS CLI to view the instance's status. For more information, see Monitoring an instance's status.

Using the AWS CLI

Use the `create-db-instance` AWS CLI operation with the following parameters to create the primary instance for your cluster.

- **`--db-instance-class`** — Required. The compute and memory capacity of the instance, for example, `db.m4.large`. Not all instance classes are available in all AWS Regions.
- **`--db-instance-identifier`** — Required. A string that identifies the instance.

  Instance Naming Constraints:

  - Length is [1—63] letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all instances across Amazon RDS, Neptune, and Amazon DocumentDB per AWS account, per Region.
- **`--engine`** — Required. Must be `docdb`.
- **`--availability-zone`** — Optional. The Availability Zone that you want this instance to be created in. Use this parameter to locate your instances in different Availability Zones to

increase fault tolerance. For more information, see Amazon DocumentDB High availability and replication.

- **`--promotion-tier`** — Optional. The failover priority tier for this instance. Must be between 0 and 15 with lower numbers being higher priority. For more information, see Controlling the failover target.

1. **First, determine what Availability Zones you can create your instance in.**

   If you want to specify the Availability Zone before you create your instance, run the following command to determine which Availability Zones are available for your Amazon DocumentDB cluster.

   For Linux, macOS, or Unix:

   ```
   aws docdb describe-db-clusters \
         --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   For Windows:

   ```
   aws docdb describe-db-clusters ^
         --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   Output from this operation looks something like the following.

   ```
   [
       [
           "sample-cluster",
           [
               "us-east-1c",
               "us-east-1b",
               "us-east-1a"
           ]
       ]
   ]
   ```

2. **Second, determine what instance classes you can create in your Region.**

To determine which instance classes are available to you in your Region, run the following command. From the output, choose an instance class for the instance you want to add to your Amazon DocumentDB cluster.

For Linux, macOS, or Unix:

```
aws docdb describe-orderable-db-instance-options \
        --engine docdb \
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

For Windows:

```
aws docdb describe-orderable-db-instance-options ^
        --engine docdb ^
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

Output from this operation looks something like the following.

```
[
    "db.r5.16xlarge",
    "db.r5.2xlarge",
    "db.r5.4xlarge",
    "db.r5.8xlarge",
    "db.r5.large",
    "db.r5.xlarge"
]
```

3. **Finally, add an instance to your Amazon DocumentDB cluster.**

   To add an instance to your Amazon DocumentDB cluster, run the following command..

   For Linux, macOS, or Unix:

```
aws docdb create-db-instance \
       --db-cluster-identifier sample-cluster \
       --db-instance-identifier sample-instance-2 \
       --availability-zone us-east-1b \
       --promotion-tier 2 \
       --db-instance-class db.r5.xlarge \
       --engine docdb
```

For Windows:

```
aws docdb create-db-instance ^
       --db-cluster-identifier sample-cluster ^
       --db-instance-identifier sample-instance-2 ^
       --availability-zone us-east-1b ^
       --promotion-tier 2 ^
       --db-instance-class db.r5.xlarge ^
       --engine docdb
```

Output from this operation looks something like the following.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance-2",
        "DBInstanceClass": "db.r5.xlarge",
        "Engine": "docdb",
        "DBInstanceStatus": "creating",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1b",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
```

```
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:11:35-sun:12:05",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 2,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance-2"
    }
}
```

It takes several minutes to create the instance. You can use the console or AWS CLI to view the instance's status. For more information, see Monitoring an Amazon DocumentDB instance's status.

Describing Amazon DocumentDB instances

You can use either the Amazon DocumentDB Management Console or the AWS CLI to see details such as connection endpoints, security groups VPCs, certificate authority, and parameter groups pertaining to your Amazon DocumentDB instances.

Using the AWS Management Console

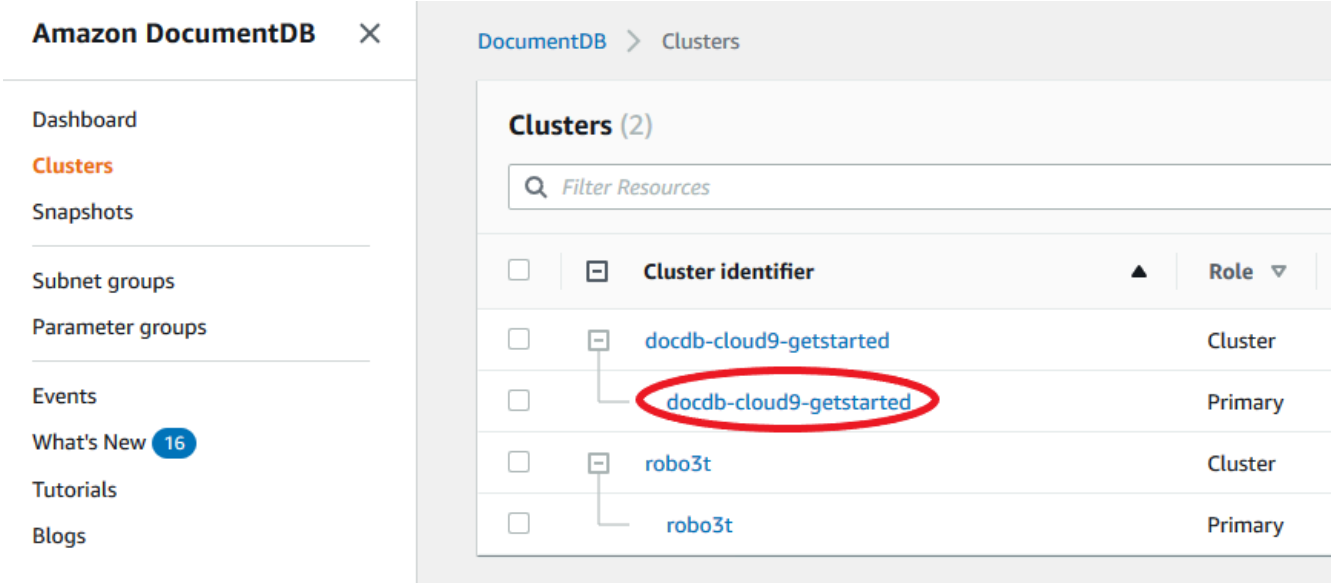
To view the details of your instances using the AWS Management Console, follow the steps below.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

2. In the navigation pane, choose **Clusters** .

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. In the list of instances, choose the name of the instance that you want to see its details. The information about the instance is organized into the following groupings:

   - **Summary**—General information about the instance, including the engine version, class, status, and any pending maintenance.
   - **Connectivity & Security** —The **Connect** section lists the connection endpoints to connect to this instance with the mongo shell or with an application. The **Security Groups** section lists the security groups associated with this instance and their VPC ID and descriptions.
   - **Configuration**—The **Details** section lists the configurations and status of the instance, including the instance's Amazon Resource Name (ARN), endpoint, role, class, and certificate authority. It also lists the instance's security and network settings, and backup information. The **Cluster details** section lists the details of the cluster that this instance

belongs to. The **Cluster instances** section lists all the instances that belong to your cluster with each instance's role and cluster parameter group status.

> **Note**
>
> You can modify the cluster associated with your instance by selecting **Modify** next to the **Cluster details** header. For more information, see Modifying an Amazon DocumentDB cluster.

- **Monitoring**—The CloudWatch Logs metrics for this instance. For more information, see Monitoring Amazon DocumentDB with CloudWatch.
- **Events & tags** —The **Recent events** section lists the recent events for this instance. Amazon DocumentDB keeps a record of events that relate to your clusters, instances, snapshots, security groups, and cluster parameter groups. This information includes the date, time, and message associated with each event. The **Tags** section lists the tags attached to this cluster. For more information, see Tagging Amazon DocumentDB resources.

Using the AWS CLI

To view the details of your Amazon DocumentDB instances using the AWS CLI, use the `describe-db-clusters` command as shown in the examples below. For more information, see `DescribeDBInstances` in the *Amazon DocumentDB Resource Management API Reference*.

> **Note**
>
> For certain management features such as cluster and instance lifecycle management, Amazon DocumentDB leverages operational technology that is shared with Amazon RDS. The `filterName=engine,Values=docdb` filter parameter returns only Amazon DocumentDB clusters.

1. **List all Amazon DocumentDB instances.**

   The following AWS CLI code lists the details for all Amazon DocumentDB instances in a region.

   For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --filter Name=engine,Values=docdb
```

For Windows:

```
aws docdb describe-db-instances \
    --filter Name=engine,Values=docdb
```

2. **List all details for a specified Amazon DocumentDB instance**

The following code lists the details for `sample-cluster-instance`. Including the `--db-instance-identifier` parameter with the name of an instance restricts the output to information on that particular instance.

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --db-instance-identifier sample-cluster-instance
```

For Windows:

```
aws docdb describe-db-instances \
    --db-instance-identifier sample-cluster-instance
```

Output from this operation looks like the following.

```
{
    "DBInstances": [
        {
            "DbiResourceId": "db-BJKKB54PIDV5QFKGVRX5T3S6GM",
            "DBInstanceArn": "arn:aws:rds:us-east-1:012345678901:db:sample-
cluster-instance-00",
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-77186e0d",
                    "Status": "active"
                }
            ],
            "DBInstanceClass": "db.r5.large",
            "DBInstanceStatus": "creating",
```

```
            "AutoMinorVersionUpgrade": true,
            "PreferredMaintenanceWindow": "fri:09:32-fri:10:02",
            "BackupRetentionPeriod": 1,
            "StorageEncrypted": true,
            "DBClusterIdentifier": "sample-cluster",
            "EngineVersion": "3.6.0",
            "AvailabilityZone": "us-east-1a",
            "Engine": "docdb",
            "PromotionTier": 2,
            "DBInstanceIdentifier": "sample-cluster-instance",
            "PreferredBackupWindow": "00:00-00:30",
            "PubliclyAccessible": false,
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-4e26d263",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1a"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-afc329f4",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1c"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-b3806e8f",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1e"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-53ab3636",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1d"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
```

```
                        "SubnetIdentifier": "subnet-991cb8d0",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1b"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-29ab1025",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1f"
                        },
                        "SubnetStatus": "Active"
                    }
                ],
                "VpcId": "vpc-91280df6",
                "DBSubnetGroupDescription": "default",
                "SubnetGroupStatus": "Complete"
            },
            "PendingModifiedValues": {},
            "KmsKeyId": "arn:aws:kms:us-east-1:012345678901:key/0961325d-
a50b-44d4-b6a0-a177d5ff730b"
        }
    ]
}
```

Modifying an Amazon DocumentDB instance

You can modify your Amazon DocumentDB instance using either the AWS Management Console or the AWS CLI. To modify an instance, the instance must be in the *available* state. You cannot modify an instance that is stopped. If the cluster is stopped, first start the cluster, wait for the instance to become *available*, and then make the desired modifications. For more information, see Stopping and starting an Amazon DocumentDB cluster.

Using the AWS Management Console

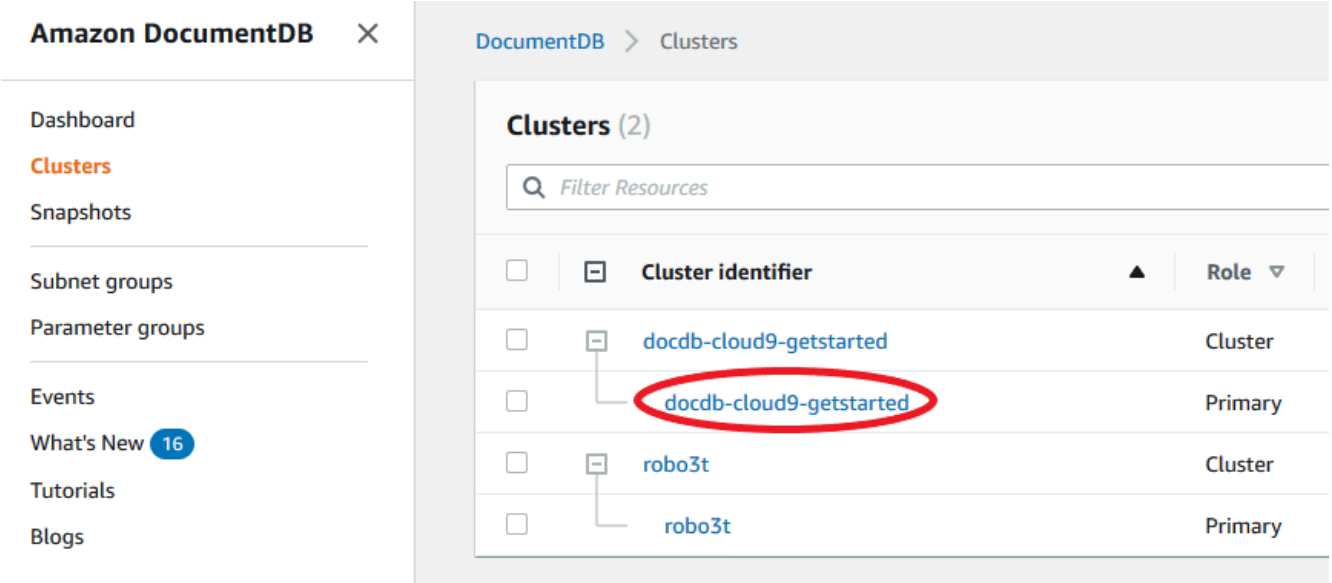
To modify a specific Amazon DocumentDB instance using the console, complete the following steps.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

2. In the navigation pane, choose **Clusters** .

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. Check the box to the left of the instance you wish to modify.
5. Choose **Actions**, and then choose **Modify**.
6. In the **Modify instance: <instance-name>** pane, make the changes that you want. You can make the following changes:

   - **Instance specifications** — The instance identifier and class. Instance identifier naming constraints:
     - **Instance identifier** — Enter a name that is unique for all instances owned by your AWS account in the current region. The instance identifier must contain [1—63] alphanumeric characters or hyphens, have a letter as the first character, and cannot end with a hyphen or contain two consecutive hyphens.
     - **Instance class** — From the drop-down menu, select an instance class for your Amazon DocumentDB instance. For more information, see Managing instance classes.

- **Certificate authority** — Server certificate for this instance. For more information, see Updating your Amazon DocumentDB TLS certificates.
- **Failover** — During failover, the instance with the highest promotion tier will be promoted to primary. For more information, see Amazon DocumentDB Failover.
- **Maintenance** — The maintenance window in which pending modifications or patches are applied to instances in the cluster.

7. When you have finished, choose **Continue** to see a summary of your changes.
8. After verifying your changes, you can apply them immediately or during the next maintenance window under **Scheduling of modifications**. Choose **Modify instance** to save your changes. Alternatively, you can choose **Cancel** to discard your changes.

It takes a few minutes for your changes to be applied. You can use the instance only when its status is *available*. You can monitor the instance's status using the console or AWS CLI. For more information, see Monitoring an Amazon DocumentDB instance's status.

Using the AWS CLI

To modify a specific Amazon DocumentDB instance using the AWS CLI, use the `modify-db-instance` with the following parameters. For more information, see ModifyDBInstance.
The following code modifies the instance class to `db.r5.large` for the instance `sample-instance`.

Parameters

- **`--db-instance-identifier`** — Required. The identifier for the instance to be modified.
- **`--db-instance-class`** — Optional. The new compute and memory capacity of the instance; for example, `db.r5.large`. Not all instance classes are available in all AWS Regions. If you modify the instance class, an outage occurs during the change. The change is applied during the next maintenance window, unless `ApplyImmediately` is specified as true for this request.
- **`--apply-immediately`** or **`--no-apply-immediately`** — Optional. Specifies whether this modification should be applied immediately or wait until the next maintenance window. If this parameter is omitted, the modification is performed during the next maintenance window.

Example

For Linux, macOS, or Unix:

```
aws docdb modify-db-instance \
       --db-instance-identifier sample-instance \
       --db-instance-class db.r5.large \
       --apply-immediately
```

For Windows:

```
aws docdb modify-db-instance ^
       --db-instance-identifier sample-instance ^
       --db-instance-class db.r5.large ^
       --apply-immediately
```

Output from this operation looks something like the following.

```
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "sample-instance-1",
            "DBInstanceClass": "db.r5.large",
            "Engine": "docdb",
            "DBInstanceStatus": "modifying",
            "Endpoint": {
                "Address": "sample-instance-1.node.us-east-1.docdb.amazonaws.com",
                "Port": 27017,
                "HostedZoneId": "ABCDEFGHIJKLM"
            },
            "InstanceCreateTime": "2020-01-10T22:18:55.921Z",
            "PreferredBackupWindow": "02:00-02:30",
            "BackupRetentionPeriod": 1,
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-abcd0123",
                    "Status": "active"
                }
            ],
            "AvailabilityZone": "us-east-1a",
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "DBSubnetGroupDescription": "default",
```

```
                "VpcId": "vpc-abcd0123",
                "SubnetGroupStatus": "Complete",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-abcd0123",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1a"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-abcd0123",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1b"
                        },
                        "SubnetStatus": "Active"
                    }
                ]
            },
            "PreferredMaintenanceWindow": "sun:10:57-sun:11:27",
            "PendingModifiedValues": {
                "DBInstanceClass": "db.r5.large"
            },
            "EngineVersion": "3.6.0",
            "AutoMinorVersionUpgrade": true,
            "PubliclyAccessible": false,
            "DBClusterIdentifier": "sample-cluster",
            "StorageEncrypted": true,
            "KmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/wJalrXUtnFEMI/
K7MDENG/bPxRfiCYEXAMPLEKEY",
            "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
            "CACertificateIdentifier": "rds-ca-2019",
            "PromotionTier": 1,
            "DBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:sample-
instance-1",
            "EnabledCloudwatchLogsExports": [
                "profiler"
            ]
        }
    ]
}
```

It takes a few minutes for your modifications to be applied. You can use the instance only when its status is *available*. You can monitor the instance's status using the AWS Management Console or AWS CLI. For more information, see Monitoring an Amazon DocumentDB instance's status.

Rebooting an Amazon DocumentDB instance

Occasionally, you might need to reboot your Amazon DocumentDB instance, usually for maintenance reasons. If you make certain changes, such as changing the cluster parameter group that is associated with a cluster, you must reboot the instances in the cluster for the changes to take effect. You can reboot a specified instance using the AWS Management Console or the AWS CLI.

Rebooting an instance restarts the database engine service. Rebooting results in a momentary outage, during which the instance status is set to `rebooting`. An Amazon DocumentDB event is created when the reboot is completed.

Rebooting an instance doesn't result in a failover. To failover an Amazon DocumentDB cluster, use the AWS Management Console or the AWS CLI operation `failover-db-cluster`. For more information, see Amazon DocumentDB Failover.

You can't reboot your instance if it isn't in the *available* state. Your database can be unavailable for several reasons, such as a previously requested modification, or a maintenance-window action. For more information on instance states, see Monitoring an Amazon DocumentDB instance's status.

Using the AWS Management Console

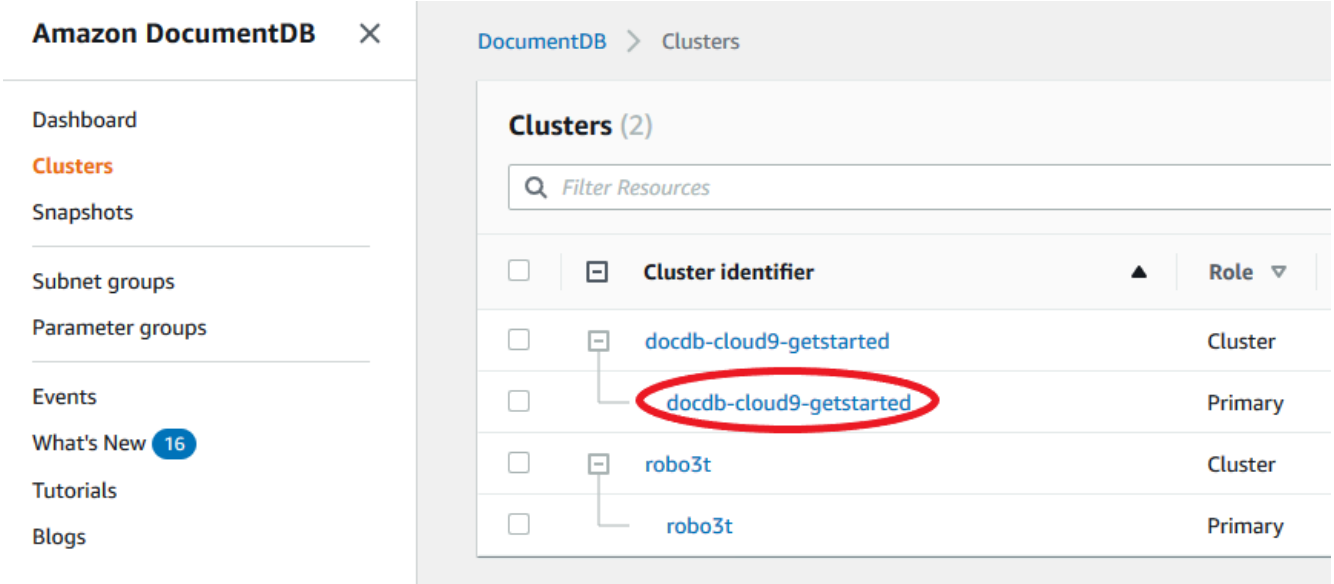
The following procedure reboots an instance that you specify using the console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters** .

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon

(≡)
in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. Check the box to the left of the instance you wish to reboot.
5. Choose **Actions**, choose **Reboot**, and then choose **Reboot** to confirm your reboot.

It takes a few minutes for your instance to reboot. You can use the instance only when its status is *available*. You can monitor the instance's status using the console or the AWS CLI. For more information, see Monitoring an Amazon DocumentDB instance's status.

Using the AWS CLI

To reboot an Amazon DocumentDB instance, use the `reboot-db-instance` operation with the `--db-instance-identifier` parameter. This parameter specifies the identifier for the instance to be rebooted.

The following code reboots the instance `sample-instance`.

Example

For Linux, macOS, or Unix:

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-instance
```

For Windows:

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-instance
```

Output from this operation looks something like the following.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "rebooting",
        "Endpoint": {
            "Address": "sample-instance.node.us-east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "ABCDEFGHIJKLM"
        },
        "InstanceCreateTime": "2020-03-27T08:05:56.314Z",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1c",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-abcd0123",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
```

```
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:06:53-sun:07:23",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ]
    }
}
```

It takes a few minutes for your instance to reboot. You can use the instance only when its status is *available*. You can monitor the instance's status using the console or AWS CLI. For more information, see Monitoring an Amazon DocumentDB instance's status.

Deleting an Amazon DocumentDB instance

You can delete your Amazon DocumentDB instance using either the AWS Management Console or the AWS CLI. To delete an instance, the instance must be in the *available* state. You cannot delete an instance that is stopped. If the Amazon DocumentDB cluster that contains your instance is stopped, first start the cluster, wait for the instance to become *available*, and then delete the instance. For more information, see Stopping and starting an Amazon DocumentDB cluster.

> **Note**
>
> Amazon DocumentDB stores all of your data in the cluster volume. The data persists in that cluster volume, even if you remove all the instances from your cluster. If you need to access the data again, you can add an instance to the cluster at any time and pick up where you left off.

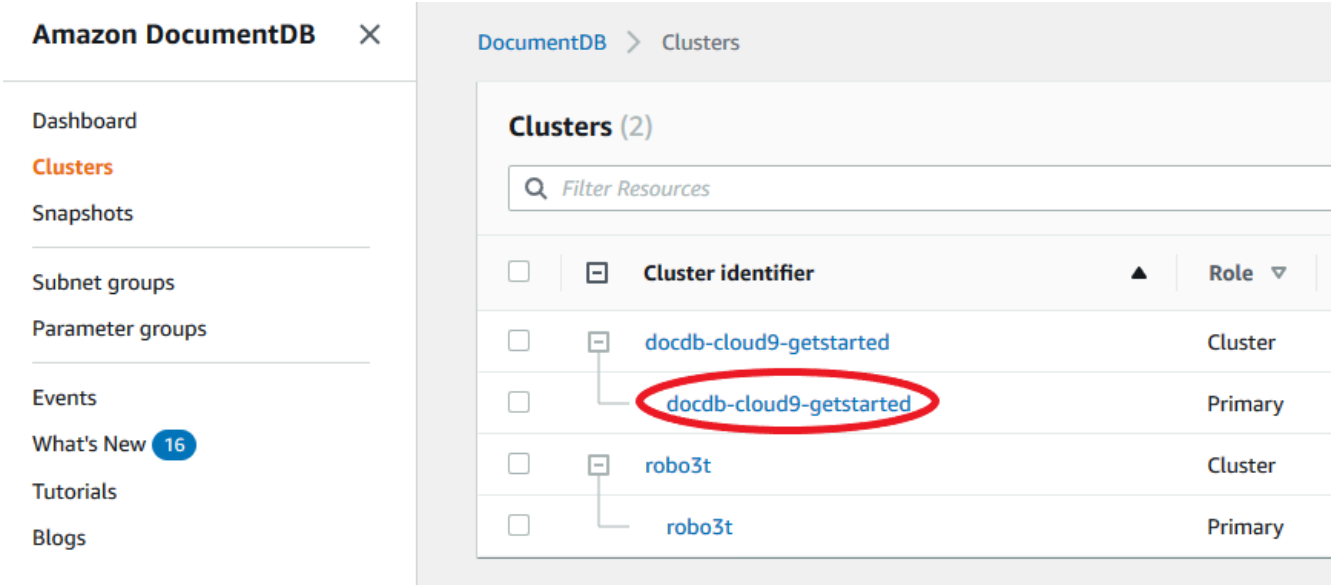
Using the AWS Management Console

The following procedure deletes a specified Amazon DocumentDB instance using the console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters** .

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. Check the box to the left of the instance you wish to delete.
5. Select **Actions**, and then choose **Delete**.

   1. If you are deleting the last instance in your cluster:
      - **Create final cluster snapshot?** — Choose **Yes** if you want to create a final snapshot before the cluster is deleted. Otherwise, choose **No**.
      - **Final snapshot name** — If you choose to create a final snapshot, enter the cluster snapshot identifier of the new cluster snapshot created.
      - **Delete <instance-name> instance?** — Enter the phrase **delete entire cluster** into the field to confirm the deletion.
   2. If you are not deleting the last instance in your cluster:
      - **Delete <instance-name> instance?** — Enter the phrase **delete me** into the field to confirm the deletion.
6. Select **Delete** to delete the instance.

It takes several minutes for an instance to be deleted. To monitor the status of an instance, see Monitoring an Amazon DocumentDB instance's status.

Using the AWS CLI

The following procedure deletes an Amazon DocumentDB instance using the AWS CLI.

1. **First, determine how many instances are in your Amazon DocumentDB cluster:**

   To determine how many instances are in your cluster, run the `describe-db-clusters` command, as follows.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].
[DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
```

   Output from this operation looks something like the following.

```
[
    [
        "sample-cluster",
        [
```

```
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
]
```

2. **If there are more than one instances in your Amazon DocumentDB cluster:**

   To delete a specified Amazon DocumentDB instance, use the `delete-db-instance` command with the `--db-instance-identifier` parameter, as shown below. It takes several minutes for an instance to be deleted. To monitor the status of an instance, see Monitoring an Amazon DocumentDB instance's status.

```
aws docdb delete-db-instance \
       --db-instance-identifier sample-instance-2
```

   Output from this operation looks something like the following.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance-2",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "deleting",
        "Endpoint": {
            "Address": "sample-instance-2.node.us-east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "ABCDEFGHIJKLM"
        },
        "InstanceCreateTime": "2020-03-27T08:05:56.314Z",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1c",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
```

```
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:06:53-sun:07:23",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance-2",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ]
    }
}
```

3. **If the instance your want to delete is the last instance in your Amazon DocumentDB cluster:**

   If you delete the last instance in an Amazon DocumentDB cluster, you also delete that cluster and the automatic snapshots and continuous backups associated with that cluster.

To delete the last instance in your cluster, you can delete the cluster and optionally create a final snapshot. For more information, see Deleting an Amazon DocumentDB cluster.

Deletion protection

Deleting the last instance of an Amazon DocumentDB cluster will also delete the cluster, as well as the automatic snapshots and continuous backups associated with that cluster. Amazon DocumentDB enforces deletion protection for a cluster whether you perform the delete operation using the AWS Management Console or the AWS CLI. If deletion protection is enabled, you can't delete a cluster.

To delete a cluster that has deletion protection enabled, you must first modify the cluster and disable deletion protection. For more information, see Deleting an Amazon DocumentDB cluster.

Managing instance classes

The instance class determines the computation and memory capacity of an Amazon DocumentDB (with MongoDB compatibility) instance. The instance class you need depends on your processing power and memory requirements.

Amazon DocumentDB supports the R4, R5, R6G, T3, and T4G families of instance classes. These classes are current-generation instance classes that are optimized for memory-intensive applications. For the specifications on these classes, see Instance class specifications.

Topics

- Determining an instance class
- Changing an instance's class
- Supported instance classes by region
- Instance class specifications

Determining an instance class

To determine the class of an instance, you can use the AWS Management Console or the `describe-db-instances` AWS CLI operation.

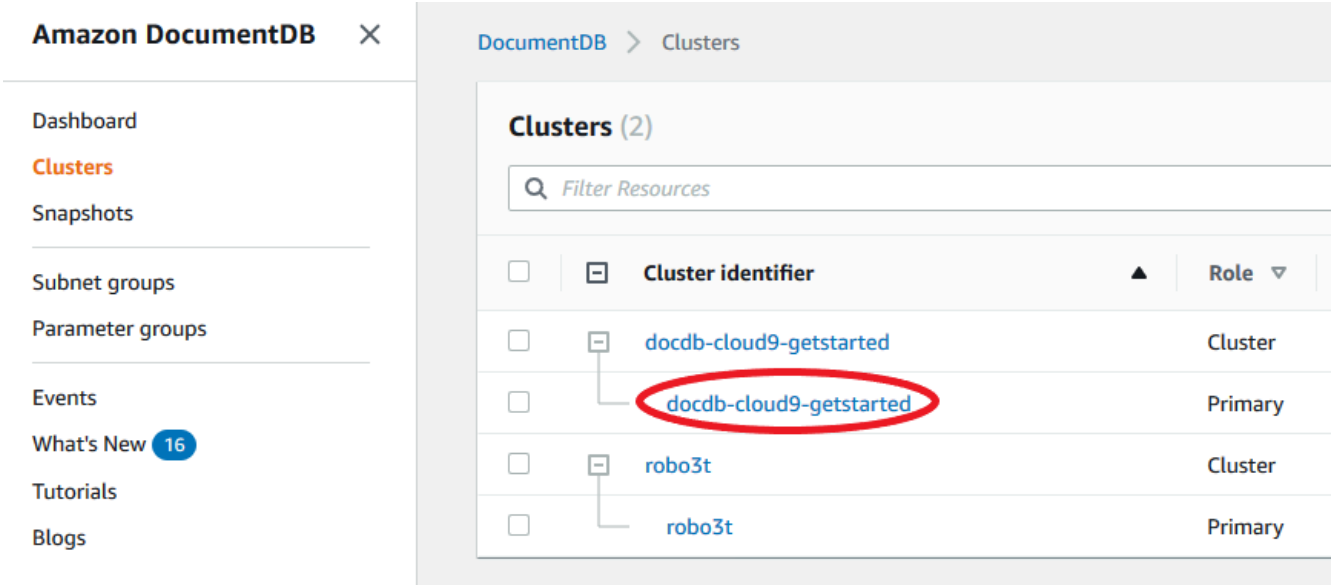
Using the AWS Management Console

To determine the instance class for your cluster's instances, complete the following steps in the console.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters** to find the instance that you're interested in.

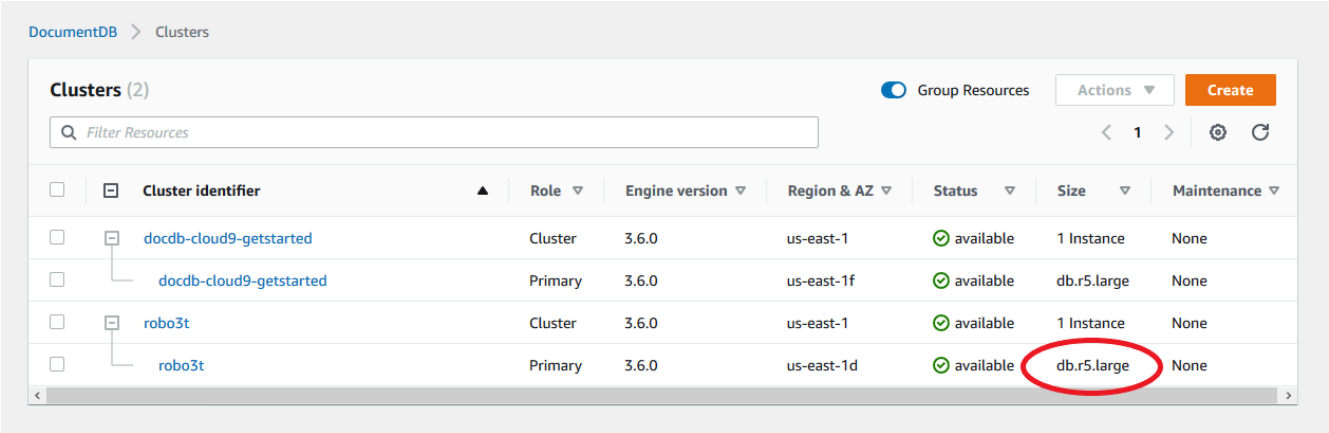
> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. In the list of instances, expand the cluster to find the instances you are interested in. Find the instance that you want. Then, look at the **Size** column of the instance's row to see its instance class.

   In the following image, the instance class for instance `robo3t` is `db.r5.4xlarge`.



Using the AWS CLI

To determine the class of an instance using the AWS CLI, use the `describe-db-instances` operation with the following parameters.

- **`--db-instance-identifier`** — Optional. Specifies the instance that you want to find the instance class for. If this parameter is omitted, `describe-db-instances` returns a description for up to 100 of your instances.
- **`--query`** — Optional. Specifies the members of the instance to include in the results. If this parameter is omitted, all instance members are returned.

Example

The following example finds the instance name and class for the instance `sample-instance-1`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --db-instance-identifier sample-instance-1
```

For Windows:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --db-instance-identifier sample-instance-1
```

Output from this operation looks something like the following.

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ]
```

Example

The following example finds the instance name and class for up to 100 Amazon DocumentDB instances.

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --filter Name=engine,Values=docdb
```

For Windows:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --filter Name=engine,Values=docdb
```

Output from this operation looks something like the following.

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ],
    [
        "sample-instance-2",
        "db.r5.large"
    ],
    [
        "sample-instance-3",
        "db.r5.4xlarge"
    ],
    [
```

```
        "sample-instance-4",
        "db.r5.4xlarge"
    ]
]
```

For more information, see Describing Amazon DocumentDB instances.

Changing an instance's class

You can change the instance class of your instance using the AWS Management Console or the AWS CLI. For more information, see Modifying an Amazon DocumentDB instance.

Supported instance classes by region

Amazon DocumentDB supports the following instance classes:

- R6G—Latest generation of memory-optimized instances powered by Arm-based AWS Graviton2 processors that provide up to 30% better performance over R5 instances at 5% less cost.
- R6GD—Memory-optimized R6G instances with local non-volatile memory express (NVMe)-based Solid-State Drive (SSD) storage for ephemeral data.
- R5—Memory-optimized instances that provide up to 100% better performance over R4 instances for the same instance cost.
- R4—Previous generation of memory-optimized instances.
- T4G—Latest-generation low cost burstable general-purpose instance type powered by Arm-based AWS Graviton2 processors that provides a baseline level of CPU performance, delivering up to 35% better price performance over T3 instances and ideal for running applications with moderate CPU usage that experience temporary spikes in usage.
- T3—Low cost burstable general-purpose instance type that provides a baseline level of CPU performance with the ability to burst CPU usage at any time for as long as required.

For detailed specifications on the instance classes, see Instance class specifications.

A particular instance class may or may not be supported in a given Region. The following table specifies which instance classes are supported by Amazon DocumentDB in each Region.

Supported instance classes by Region

Region	R6GD	R6G	R5	R4	T4G	T3
US East (Ohio)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
US East (N. Virginia)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
US West (Oregon)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
Africa (Cape Town)		Supporte	Supported		Supporte	Supported
South America (São Paulo)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Hong Kong)		Supporte	Supported		Supporte	Supported
Asia Pacific (Hyderabad)			Supported			Supported
Asia Pacific (Mumbai)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Seoul)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Sydney)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Singapore)	Supporte	Supporte	Supported		Supporte	Supported
Asia Pacific (Tokyo)	Supporte	Supporte	Supported		Supporte	Supported

Region	R6GD	R6G	R5	R4	T4G	T3
Canada (Central)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Frankfurt)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Ireland)	Supporte	Supporte	Supported	Supporte	Supporte	Supported
Europe (London)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Milan)		Supporte	Supported		Supporte	Supported
Europe (Paris)	Supporte	Supporte	Supported		Supporte	Supported
Europe (Spain)		Supporte	Supported		Supporte	Supported
Europe (Stockholm)		Supporte	Supported		Supporte	Supported
Middle East (UAE)		Supporte	Supported		Supporte	Supported
China (Beijing)	Supporte	Supporte	Supported		Supporte	Supported
China (Ningxia)		Supporte	Supported		Supporte	Supported
Israel (Tel Aviv)		Supporte	Supported		Supporte	Supported

Region	R6GD	R6G	R5	R4	T4G	T3
AWS GovCloud (US-West)	Supporte	Supporte	Supported		Supporte	Supported
AWS GovCloud (US-East)	Supporte	Supporte	Supported		Supporte	Supported

Instance class specifications

The following table provides details of the Amazon DocumentDB instance classes, including which instance types are supported in each class. You can find explanations for each table column below the table.

Instance class	vCPU[1]	Memory (GiB)[2]	NVMe SSD tiered cache (GiB)[3]	Max. temp. storage (GiB)[4]	Baseline / burst bandwidth (Gbps)[5]	Supporting Engines[6]
R6G – Current Generation Memory-Optimized Instance Class based on Graviton2						
`db.r6g.large`	2	16	-	32	0.75 / 10	4.0.0 and 5.0.0
`db.r6g.xlarge`	4	32	-	63	1.25 / 10	4.0.0 and 5.0.0
`db.r6g.2xlarge`	8	64	-	126	2.5 / 10	4.0.0 and 5.0.0
`db.r6g.4xlarge`	16	128	-	252	5.0 / 10	4.0.0 and 5.0.0
`db.r6g.8xlarge`	32	256	-	504	12	4.0.0 and 5.0.0

Instance class	vCPU[1]	Memory (GiB)[2]	NVMe SSD tiered cache (GiB)[3]	Max. temp. storage (GiB)[4]	Baseline / burst bandwidth (Gbps)[5]	Supporting Engines[6]
db.r6g.12xlarge	48	384	-	756	20	4.0.0 and 5.0.0
db.r6g.16xlarge	64	512	-	1008	25	4.0.0 and 5.0.0
R6GD – Current Generation NVMe-backed Instance Class based on Graviton2						
db.r6gd.xlarge	4	32	173	64	1.25 / 10	5.0.0 only
db.r6gd.2xlarge	8	64	346	128	2.5 / 10	5.0.0 only
db.r6gd.4xlarge	16	128	694	256	5.0 / 10	5.0.0 only
db.r6gd.8xlarge	32	256	1388	512	12	5.0.0 only
db.r6gd.12xlarge	48	384	2082	768	20	5.0.0 only
db.r6gd.16xlarge	64	512	2776	1024	25	5.0.0 only
R5 – Previous Generation Memory-Optimized Instance Class						
db.r5.large	2	16	-	31	0.75 / 10	3.6.0, 4.0.0, and 5.0.0

Instance class	vCPU[1]	Memory (GiB)[2]	NVMe SSD tiered cache (GiB)[3]	Max. temp. storage (GiB)[4]	Baseline / burst bandwidth (Gbps)[5]	Supporting Engines[6]
db.r5.xlarge	4	32	-	62	1.25 / 10	3.6.0, 4.0.0, and 5.0.0
db.r5.2xlarge	8	64	-	124	2.5 / 10	3.6.0, 4.0.0, and 5.0.0
db.r5.4xlarge	16	128	-	249	5.0 / 10	3.6.0, 4.0.0, and 5.0.0
db.r5.8xlarge	32	256	-	504	10	3.6.0, 4.0.0, and 5.0.0
db.r5.12xlarge	48	384	-	748	12	3.6.0, 4.0.0, and 5.0.0
db.r5.16xlarge	64	512	-	1008	20	3.6.0, 4.0.0, and 5.0.0
db.r5.24xlarge	96	768	-	1500	25	3.6.0, 4.0.0, and 5.0.0
R4 – Previous Generation Memory-Optimized Instance Class						
db.r4.large	2	15.25	-	30	0.75 / 10	3.6.0 only

Instance class	vCPU[1]	Memory (GiB)[2]	NVMe SSD tiered cache (GiB)[3]	Max. temp. storage (GiB)[4]	Baseline / burst bandwidth (Gbps)[5]	Supporting Engines[6]
db.r4.xlarge	4	30.5	-	60	1.25 / 10	3.6.0 only
db.r4.2xlarge	8	61	-	120	2.5 / 10	3.6.0 only
db.r4.4xlarge	16	122	-	240	5.0 /10	3.6.0 only
db.r4.8xlarge	32	244	-	480	10	3.6.0 only
db.r4.16xlarge	64	488	-	960	25	3.6.0 only
T4G – Latest Generation Burstable Performance Instance Classes based on Graviton2						
db.t4g.medium	2	4	-	8.13	0.256 / 5	4.0.0 and 5.0.0
T3 – Previous Generation Burstable Performance Instance Classes						
db.t3.medium	2	4	-	7.5	0.256 / 5	3.6.0, 4.0.0, and 5.0.0

Instance class	vCPU[1]	Memory (GiB)[2]	NVMe SSD tiered cache (GiB)[3]	Max. temp. storage (GiB)[4]	Baseline / burst bandwidth (Gbps)[5]	Supporting Engines[6]

1. **vCPU** — The number of virtual central processing units (CPUs). A virtual CPU is a unit of capacity that you can use to compare instance classes. Instead of purchasing or leasing a particular processor to use for several months or years, you are renting capacity by the hour. Our goal is to provide a consistent amount of CPU capacity no matter what the actual underlying hardware.
2. **Memory (GiB)** — The RAM, in gigabytes, that is allocated to the instance. There is often a consistent ratio between memory and vCPU.
3. **NVMe SSD tiered cache** — The space on the SSD volume, measured in gigabytes, allocated as extended cache for storing ephemeral data. This cache is only available in NVMe-backed instances.
4. **Max. temp. storage (GiB)** — The space, measured in gigabytes, allocated to the instance for non-persistent temporary file storage. For NVMe-backed instances, this storage is hosted on an NVMe-based SSD volume. In all other instances, it is hosted on Amazon Elastic Block Store (EBS).
5. **Baseline / burst bandwidth (Gbps)** — Burst bandwidth represents the maximum bandwidth in gigabits per second. Divide by 8 to get the expected throughput in gigabytes per second. Instances of size 4xlarge and smaller have a baseline bandwidth. To meet additional demand, they can use a network I/O credit mechanism to burst beyond their baseline bandwidth . Instances can use burst bandwidth for a limited time, typically from 5 to 60 minutes, depending on the instance size.
6. **Supporting Engines** — The Amazon DocumentDB engines that support the instance class.

NVMe-backed instances

NVMe-backed instances offer up to 7x faster query performance for applications with large datasets that exceed the memory of a regular instance. These instances leverage the local non-volatile memory express (NVMe)-based Solid-State Drive (SSD) storage available on r6gd instances to store ephemeral data, reducing network-based storage access, and improving read latency and throughput.

The local SSD space is divided into two sections:

- **Tiered cache** — Approximately 73% of the local SSD is allocated as a database cache, allowing the system to store up to five times more database pages than the main memory alone. The local SSD acts as a second-tier cache, while the existing in-memory buffer cache remains the first-tier cache. The query only accesses external storage if both the buffer cache and the SSD cache experience a miss.
- **Temporary storage** — The remaining 27% is reserved for non-persistent temporary file storage, used for complex queries involving sorts or resource-intensive operations like index builds. On regular instances, the temporary space resides on an Amazon Elastic Block Store (EBS) volume. The locally hosted temporary storage on the SSD reduces query latency involving sorts by up to two times and accelerates resource-intensive operations like index builds.

The details regarding the type of NVMe-backed instances supported and its specification can be found in Instance class specifications.

Topics

- Recommended use cases for NVMe-backed instances
- Using NVMe-backed instances with Amazon DocumentDB
- Monitoring NVMe-backed instances

Recommended use cases for NVMe-backed instances

We recommend you use NVMe-backed instances in the following scenarios:

- **Read-heavy workloads** — If your workload is read-intensive, and your dataset is larger than the buffer cache, indicated by low `BufferCacheHitRatio` and high `ReadIOPS` metrics, NVMe-backed instances can provide performance benefits.
- **Update-heavy workloads** — If your workload is update-intensive, and garbage collection is unable to keep up due to read latency on network storage, NVMe-backed instances could help mitigate the issue.

NVMe-backed instances can benefit various use cases, including:

- **Internet-scale applications** — Applications such as payment processing, billing, and e-commerce with strict performance Service Level Agreements (SLAs) can leverage the performance advantages of NVMe-backed instances.
- **Real-time reporting dashboards** — Dashboards that run hundreds of queries for metrics/data collection can benefit from the low latency and high throughput of NVMe-backed instances.
- **Generative AI applications** — Applications using vector search to find exact or nearest neighbors across millions of vector embeddings can leverage the high performance of NVMe-backed instances.

Using NVMe-backed instances with Amazon DocumentDB

To use NVMe-backed instances of Amazon DocumentDB:

- Create an Amazon DocumentDB cluster and add one of the NVMe-backed instance classes. For more information, see Creating an Amazon DocumentDB cluster.
- Alternatively, modify an existing Amazon DocumentDB cluster to use one of the NVMe-backed instance classes. For more information, see Modifying an Amazon DocumentDB cluster.

To check the availability of NVMe-backed instances across different AWS Regions, see Supported instance classes by region.

If you want to switch back from an NVMe-backed instance to a regular instance, modify the database instance class of your Amazon DocumentDB instance to a similar instance class without the NVMe storage. For example, if your current instance class is 'db.r6gd.4xlarge', choose 'db.r6g.4xlarge' to switch back. For more information, see Modifying an Amazon DocumentDB cluster.

Monitoring NVMe-backed instances

In addition to the regular instance metrics available in Amazon CloudWatch, NVMe-backed instances emit additional metrics specific to NVMe-based SSD storage, IOPS, and throughput.

```
NVMeStorageCacheHitRatio
FreeNVMeStorage
ReadIOPSNVMeStorage
ReadLatencyNVMeStorage
ReadThroughputNVMeStorage
WriteIOPSNVMeStorage
```

```
WriteLatencyNVMeStorage
WriteThroughputNVMeStorage
```

For more information about these metrics, see NVMe-backed instance metrics

Managing Amazon DocumentDB subnet groups

A virtual private cloud (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud. You can launch your AWS resources, such as Amazon DocumentDB clusters, into your Amazon VPC. You can specify an IP address range for the VPC, add subnets, associate security groups, and configure route tables.

A subnet is a range of IP addresses in your Amazon VPC. You can launch AWS resources into a specified subnet. Use a *public* subnet for resources that must be connected to the internet. Use a *private* subnet for resources that won't be connected to the internet. For more information about public and private subnets, see VPC and Subnet Basics in the *Amazon Virtual Private Cloud User Guide*.

A DB subnet group is a collection of subnets that you create in a VPC that you then designate for your clusters. A subnet group allows you to specify a particular VPC when creating clusters. If you use the `default` subnet group, it spans all subnets in the VPC.

Each DB subnet group should have subnets in at least two Availability Zones in a given Region. When creating a DB cluster in a VPC, you must select a DB subnet group. Amazon DocumentDB uses that DB subnet group and your preferred Availability Zone to select a subnet and an IP address within that subnet to associate with your cluster. If the primary instance fails, Amazon DocumentDB can promote a corresponding replica instance to be the new primary. It can then create a new replica instance using an IP address of the subnet in which the previous primary was located.

When Amazon DocumentDB creates an instance in a VPC, it assigns a network interface to your cluster by using an IP address selected from your DB subnet group. We strongly recommend that you use the DNS name because the underlying IP address can change during failover. For more information, see Amazon DocumentDB endpoints.

For information about creating your own VPC and subnets, see Working with VPCs and Subnets in the *Amazon Virtual Private Cloud User Guide*.

Topics

- Creating an Amazon DocumentDB subnet group
- Describing an Amazon DocumentDB subnet group
- Modifying an Amazon DocumentDB subnet group
- Deleting an Amazon DocumentDB subnet group

Creating an Amazon DocumentDB subnet group

When creating an Amazon DocumentDB cluster, you must choose a Amazon VPC and corresponding subnet group within that Amazon VPC to launch your cluster. Subnets determine the availability zone and IP range within the availability zone that you want to use to launch an instance.

A subnet group is a named set of subnets (or AZs) that allows you to specify the availability zones that you want to use to for launching Amazon DocumentDB instances. For example, in a cluster with three instances, it is recommended that each of those instances are provisioned in separate AZs—doing so optimizes for high availability. Thus, if a single AZ fails, it will only affect a single instance.

Currently, Amazon DocumentDB instances can be provisioned in up to three AZs. Even if a subnet group has more than three subnets, you will only be able to use three of those subnets to create an Amazon DocumentDB cluster. Therefore, we recommend that when you create a subnet group that you only choose the three subnets of which you want to deploy your instances.

For example: A cluster is created and Amazon DocumentDB choose AZs {1A, 1B, and 1C}. If you attempt to create an instance in AZ {1D} the API call will fail. However, if you choose to create an instance, without specifying the particular AZ, then Amazon DocumentDB will choose an AZ on your behalf. Amazon DocumentDB uses an algorithm to load balance the instances across AZs to help you achieve high availability. If three instances are provisioned, by default, they will be provisioned across three AZs and will not be provisioned all in a single AZ.

Best Practices

- Unless you have a specific reason, always create a subnet group with three subnets. This ensures that clusters with three or more instances will be able to achieve higher availability as instances will be provisioned across three AZs.
- Always spread instances across multiple AZs to achieve high availability. Never place all instances for a cluster in a single AZ.

- Because failover events can happen at any time, you should not assume that a primary instance or replica instances will always be in a particular AZ.

How to create a subnet group

You can use the AWS Management Console or AWS CLI to create an Amazon DocumentDB subnet group:

Using the AWS Management Console

Use the following steps to create an Amazon DocumentDB subnet group.

To create an Amazon DocumentDB subnet group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Subnet groups**, then choose **Create**.

> **Tip**
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. On the **Create subnet group** page:
   a. In the **Subnet group details** section:
      i. **Name**—Enter a meaningful name for the subnet group.
      ii. **Description**—Enter a description for the subnet group.
   b. In the **Add subnets** section:
      i. **VPC**—In the list, choose a VPC for this subnet group.
      ii. Do one of the following:
         - To include all subnets in the chosen VPC, choose **Add all the subnets related to this VPC**.

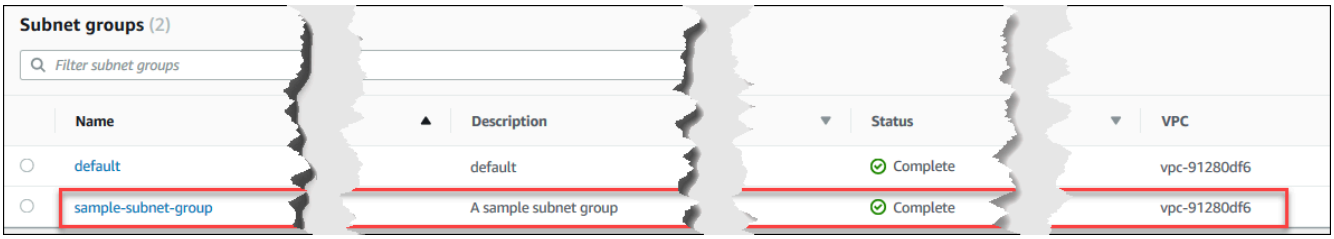
- To specify subnets for this subnet group, do the following for each Availability Zone for which you want to include subnets. You must include at least two Availability Zones.

  A. **Availability zone**—In the list, choose an Availability Zone.

  B. **Subnet**—In the list, choose a subnet from the chosen Availability Zone for this subnet group.

  C. Choose **Add subnet**.

4. Choose **Create**. When the subnet group is created, it is listed with your other subnet groups.



Using the AWS CLI

Before you can create a subnet group using the AWS CLI, you must first determine which subnets are available. Run the following AWS CLI operation to list the Availability Zones and their subnets.

Parameters:

- **`--db-subnet-group`**—Optional. Specifying a particular subnet group lists the Availability Zones and subnets for that group. Omitting this parameter lists Availability Zones and subnets for all your subnet groups. Specifying the `default` subnet group lists all the VPC's subnets.

Example

For Linux, macOS, or Unix:

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name default \
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

For Windows:

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name default ^
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

Output from this operation looks something like the following (JSON format).

```
[
    [
        "default",
        [
            [
                "us-east-1a",
                "subnet-4e26d263"
            ],
            [
                "us-east-1c",
                "subnet-afc329f4"
            ],
            [
                "us-east-1e",
                "subnet-b3806e8f"
            ],
            [
                "us-east-1d",
                "subnet-53ab3636"
            ],
            [
                "us-east-1b",
                "subnet-991cb8d0"
            ],
            [
                "us-east-1f",
                "subnet-29ab1025"
            ]
        ]
    ]
]
```

Using the output from the previous operation, you can create a new subnet group. The new subnet group must include subnets from at least two Availability Zones.

Parameters:

- **`--db-subnet-group-name`**—Required. The name for this subnet group.
- **`--db-subnet-group-description`**—Required. The description of this subnet group.
- **`--subnet-ids`**—Required. A list of subnets to include in this subnet group. Example: `subnet-53ab3636`.
- --Tags—Optional. A list of tags (key-value pairs) to attach to this subnet group.

The following code creates the subnet group `sample-subnet-group` with three subnets, `subnet-4e26d263`, `subnet-afc329f4`, and `subnet-b3806e8f`.

For Linux, macOS, or Unix:

```
aws docdb create-db-subnet-group \
    --db-subnet-group-name sample-subnet-group \
    --db-subnet-group-description "A sample subnet group" \
    --subnet-ids subnet-4e26d263 subnet-afc329f4 subnet-b3806e8f \
    --tags Key=tag1,Value=One Key=tag2,Value=2
```

For Windows:

```
aws docdb create-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group ^
    --db-subnet-group-description "A sample subnet group" ^
    --subnet-ids subnet-4e26d263 subnet-afc329f4 subnet-b3806e8f ^
    --tags Key=tag1,Value=One Key=tag2,Value=2
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupDescription": "A sample subnet group",
        "DBSubnetGroupName": "sample-subnet-group",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1a"
                },
                "SubnetIdentifier": "subnet-4e26d263",
```

```
                "SubnetStatus": "Active"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1c"
                },
                "SubnetIdentifier": "subnet-afc329f4",
                "SubnetStatus": "Active"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1e"
                },
                "SubnetIdentifier": "subnet-b3806e8f",
                "SubnetStatus": "Active"
            }
        ],
        "VpcId": "vpc-91280df6",
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "SubnetGroupStatus": "Complete"
    }
}
```

Describing an Amazon DocumentDB subnet group

You can use the AWS Management Console or the AWS CLI to get the details of an Amazon DocumentDB subnet group.

Using the AWS Management Console

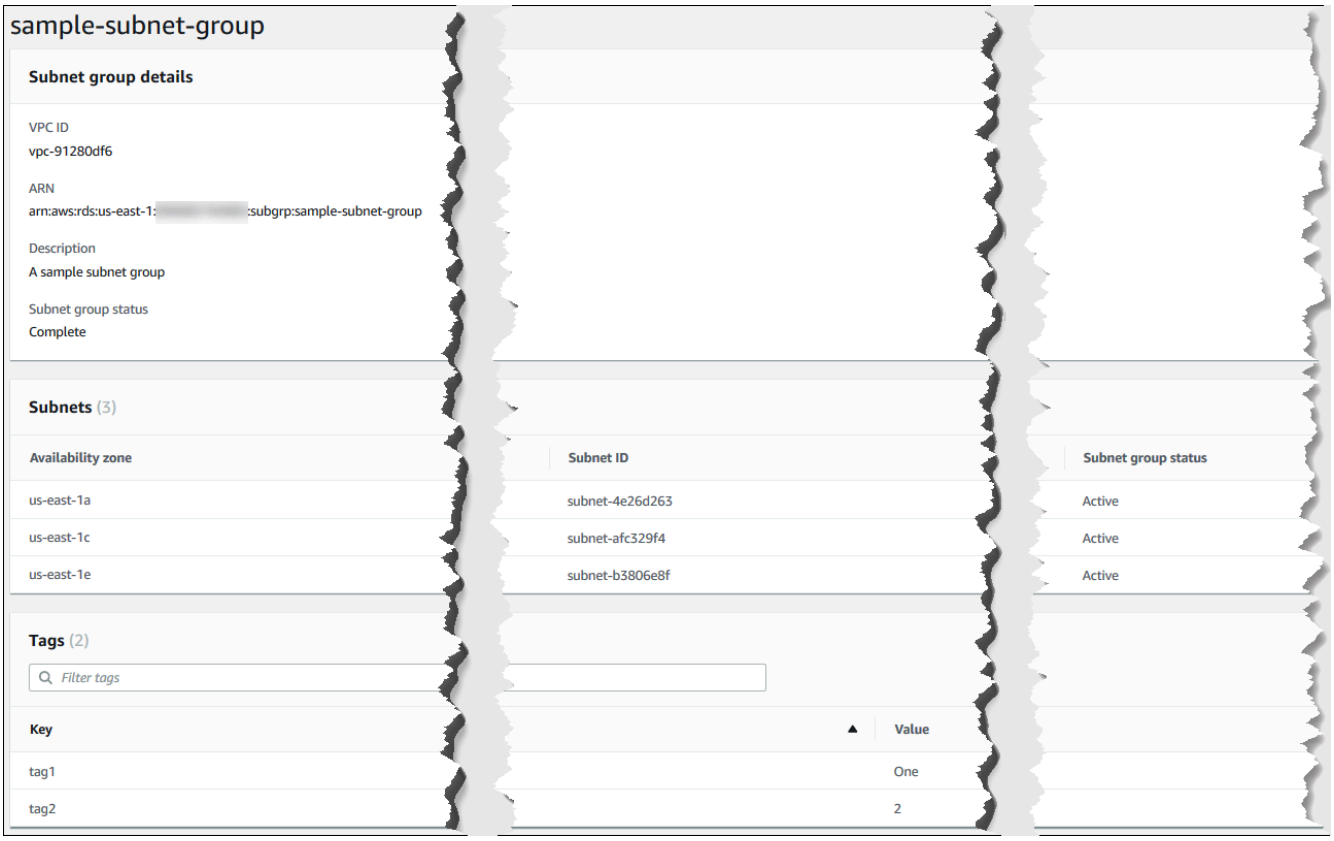
The following procedure shows you how to get the details of an Amazon DocumentDB subnet group.

To find the details of a subnet group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Subnet groups**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. To see the details of a subnet group, choose the name of that subnet group.



Using the AWS CLI

To find the details of an Amazon DocumentDB subnet group, use the `describe-db-subnet-groups` operation with the following parameter.

Parameter

- `--db-subnet=group-name`—Optional. If included, details for the named subnet group are listed. If omitted, details for up to 100 subnet groups are listed.

Example

The following code lists details for the `sample-subnet-group` subnet group that we created in the Creating an Amazon DocumentDB subnet group section.

For Linux, macOS, or Unix:

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name sample-subnet-group
```

For Windows:

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name sample-subnet-group
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "VpcId": "vpc-91280df6",
        "SubnetGroupStatus": "Complete",
        "DBSubnetGroupName": "sample-subnet-group",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1a"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-4e26d263"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1c"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-afc329f4"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1e"
                },
```

```
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-b3806e8f"
            }
        ],
        "DBSubnetGroupDescription": "A sample subnet group"
    }
}
```

Modifying an Amazon DocumentDB subnet group

You can use the AWS Management Console or AWS CLI to modify a subnet group's description or to add or remove subnets from an Amazon DocumentDB subnet group. However, you cannot modify the `default` subnet group.

Using the AWS Management Console

You can use the AWS Management Console to change a subnet group's description or to add and remove subnets. Remember that when you're finished, you must have at least two Availability Zones associated with your subnet group.

To modify your subnet group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Subnet groups**. Then choose the button to the left of the subnet group's name. Remember that you can't modify the `default` subnet group.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. Choose **Actions**, and then choose **Modify**.
4. **Description**—To change the description of your subnet group, enter a new description.
5. To change the subnets associated with your subnet group, in the **Add subnets** section, do any one or more of the following:

- To remove all subnets from this subnet group, choose **Remove all**.
- To remove specific subnets from this subnet group, choose **Remove** for each subnet you want to remove.
- To add all the subnets associated with this VPC, choose **Add all the subnets related to this VPC**.
- To add specific subnets to this subnet group, do the following for each Availability Zone for which you want to add a subnet.

  a. **Availability zone**—In the list, choose a new Availability Zone.

  b. **Subnet**—In the list, choose a subnet from the chosen Availability Zone for this subnet group.

  c. Choose **Add subnet**.

6. In the confirmation dialog box:
   - To make these changes to the subnet group, choose **Modify**.
   - To keep the subnet group unchanged, choose **Cancel**.

Using the AWS CLI

You can use the AWS CLI to change a subnet group's description or to add and remove subnets. Remember that when you're finished, you must have at least two Availability Zones associated with your subnet group. You can't modify the `default` subnet group.

Parameters:

- `--db-subnet-group-name`—Required. The name of the Amazon DocumentDB subnet group you are modifying.
- `--subnet-ids`—Required. A list of all the subnets that you want in the subnet group after this change is done.

> **Important**
>
> Any subnets currently in the subnet group that are not included in this list are removed from the subnet group. If you want to keep any of the subnets currently in the subnet group, you must include them in this list.

- `--db-subnet-group-description`—Optional. The description of the subnet group.

Example

The following code modifies the description and replaces the existing subnets with the subnets `subnet-991cb8d0`, `subnet-53ab3636`, and `subnet-29ab1025`.

For Linux, macOS, or Unix:

```
aws docdb modify-db-subnet-group \
    --db-subnet-group-name sample-subnet-group \
    --subnet-ids subnet-991cb8d0 subnet-53ab3636 subnet-29ab1025 \
    --db-subnet-group-description "Modified subnet group"
```

For Windows:

```
aws docdb modify-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group ^
    --subnet-ids subnet-991cb8d0 subnet-53ab3636 subnet-29ab1025 ^
    --db-subnet-group-description "Modified subnet group"
```

Output from this operation looks something like the following (JSON format). Notice that this is the same subnet group that was created in the [Creating an Amazon DocumentDB subnet group](#) section. However, the subnets in the subnet group are replaced with those listed in the `modify-db-subnet-group` operation.

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "DBSubnetGroupDescription": "Modified subnet group",
        "SubnetGroupStatus": "Complete",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1d"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-53ab3636"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1b"
                },
```

```
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-991cb8d0"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1f"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-29ab1025"
            }
        ],
        "VpcId": "vpc-91280df6",
        "DBSubnetGroupName": "sample-subnet-group"
    }
}
```

Deleting an Amazon DocumentDB subnet group

You can use the AWS Management Console or AWS CLI to delete an Amazon DocumentDB subnet group. However, you cannot delete the `default` subnet group.

Using the AWS Management Console

You can use the AWS Management Console to delete a subnet group. But you can't delete the `default` subnet group.

To delete a subnet group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Subnet groups**. Then choose the button to the left of the subnet group's name. Remember that you can't delete the `default` subnet group.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

3. Choose **Actions**, and then choose **Delete**.
4. In the confirmation dialog box:
   - To delete the subnet group, choose **Delete**.
   - To keep the subnet group, choose **Cancel**.

Using the AWS CLI

To delete an Amazon DocumentDB subnet group using the AWS CLI, use the `delete-db-subnet-group` operation with the following parameter.

Parameter

- `--db-subnet-group-name`—Required. The name of the Amazon DocumentDB subnet group to delete. Remember that you can't delete the `default` subnet group.

Example

The following code deletes `sample-subnet-group`.

For Linux, macOS, or Unix:

```
aws docdb delete-db-subnet-group \
    --db-subnet-group-name sample-subnet-group
```

For Windows:

```
aws docdb delete-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group
```

This operation produces no output.

Amazon DocumentDB High availability and replication

You can achieve high availability and read scaling in Amazon DocumentDB (with MongoDB compatibility) by using replica instances. A single Amazon DocumentDB cluster supports a single primary instance and up to 15 replica instances. These instances can be distributed across Availability Zones within the cluster's Region. The primary instance accepts read and write traffic, and replica instances accept only read requests.

The cluster volume is made up of multiple copies of the data for the cluster. However, the data in the cluster volume is represented as a single, logical volume to the primary instance and to Amazon DocumentDB replicas in the cluster. Replica instances are eventually consistent. They return query results with minimal replica lag—usually much less than 100 milliseconds after the primary instance has written an update. Replica lag varies depending on the rate of database change. That is, during periods in which a large number of write operations occur for the database, you might see an increase in the replica lag.

Read scaling

Amazon DocumentDB replicas work well for read scaling because they are fully dedicated to read operations on your cluster volume. Write operations are managed by the primary instance. The cluster volume is shared among all instances in your cluster. Therefore, you don't have to replicate and maintain a copy of the data for each Amazon DocumentDB replica.

High availability

When you create an Amazon DocumentDB cluster, depending upon the number of Availability Zones in the subnet group (there must be at least two), Amazon DocumentDB provisions instances across the Availability Zones. When you create instances in the cluster, Amazon DocumentDB automatically distributes the instances across the Availability Zones in a subnet group to balance the cluster. This action also prevents all instances from being located in the same Availability Zone.

Example

To illustrate the point, consider an example where you create a cluster that has a subnet group with three Availability Zones: *AZ1*, *AZ2*, and *AZ3*.

When the first instance in the cluster is created, it is the primary instance and is located in one of the Availability Zones. In this example, it's in *AZ1*. The second instance created is a replica instance and is located in one of the other two Availability Zones, say *AZ2*. The third instance created is a replica instance and is located in the remaining Availability Zone, *AZ3*. If you create more instances, they are distributed across the Availability Zones so that you achieve balance in the cluster.

If a failure occurs in the primary instance (AZ1), a failover is triggered, and one of the existing replicas is promoted to primary. When the old primary recovers, it becomes a replica in the same Availability Zone in which it was provisioned (AZ1). When you provision a three-instance cluster, Amazon DocumentDB continues to preserve that three-instance cluster. Amazon DocumentDB automatically handles detection, failover, and recovery of instance failures without any manual intervention.

When Amazon DocumentDB performs a failover and recovers an instance, the recovered instance remains in the Availability Zone in which it was originally provisioned. However, the role of the instance might change from primary to replica. Doing this prevents the scenario in which a series of failovers could result in all instances being in the same Availability Zone.

You can specify Amazon DocumentDB replicas as failover targets. That is, if the primary instance fails, the specified Amazon DocumentDB replica or replica from a tier is promoted to the primary instance. There is a brief interruption during which read and write requests made to the primary instance fail with an exception. If your Amazon DocumentDB cluster doesn't include any Amazon DocumentDB replicas, when the primary instance fails, it is re-created. Promoting an Amazon DocumentDB replica is much faster than re-creating the primary instance.

For high availability scenarios, we recommend that you create one or more Amazon DocumentDB replicas. These replicas should be of the same instance class as the primary instance and in different Availability Zones for your Amazon DocumentDB cluster.

For more information, see the following:

- Understanding Amazon DocumentDB cluster fault tolerance
- Amazon DocumentDB Failover
  - Controlling the failover target

High availability with global clusters

For high availability across multiple AWS Regions, you can set up Amazon DocumentDB global clusters. Each global cluster spans multiple regions, enabling low latency global reads and disaster recovery from outages across an AWS Region. Amazon DocumentDB automatically handles replicating all data and updates from the primary region to each of the secondary regions.

Adding replicas

The first instance added to the cluster is the primary instance. Every instance that is added after the first instance is a replica instance. A cluster can have up to 15 replica instances in addition to the primary.

When you create a cluster using the AWS Management Console, a primary instance is automatically created at the same time. To create a replica at the same time as you create the cluster and the primary instance, choose **Create replica in different zone**. For more information, see step 4.d in

Creating an Amazon DocumentDB cluster. To add more replicas to an Amazon DocumentDB cluster, see Adding an Amazon DocumentDB instance to a cluster.

When using the AWS CLI to create your cluster, you must explicitly create your primary and replica instances. For more information, see the "Using the AWS CLI" section in the following topics:

- Creating an Amazon DocumentDB cluster
- Adding an Amazon DocumentDB instance to a cluster

Amazon DocumentDB Failover

In certain cases, such as certain types of planned maintenance, or in the unlikely event of a primary node or Availability Zone failure, Amazon DocumentDB (with MongoDB compatibility) detects the failure and replaces the primary node. During a failover, write down time is minimized. This is because the role of primary node fails over to one of the read replicas instead of having to create and provision a new primary node. This failure detection and replica promotion ensure that you can resume writing to the new primary as soon as promotion is complete.

For failover to function, your cluster must have at least two instances — a primary and at least one replica instance.

> **Note**
>
> This topic only applies to original Amazon DocumentDB instance-based clusters. It does not apply to elastic or global clusters.

Controlling the failover target

Amazon DocumentDB provides you with failover tiers as a means to control which replica instance is promoted to primary when a failover occurs.

Failover Tiers

Each replica instance is associated with a failover tier (0–15). When a failover occurs due to maintenance or an unlikely hardware failure, the primary instance fails over to a replica with the highest priority (the lowest numbered tier). If multiple replicas have the same priority tier, the primary fails over to that tier's replica that is the closest in size to the previous primary.

By setting the failover tier for a group of select replicas to `0` (the highest priority), you can ensure that a failover will promote one of the replicas in that group. You can effectively prevent specific replicas from being promoted to primary in case of a failover by assigning a low-priority tier (high number) to these replicas. This is useful in cases where specific replicas are receiving heavy use by an application and failing over to one of them would negatively impact a critical application.

You can set the failover tier of an instance when you create it or later by modifying it. Setting an instance failover tier by modifying the instance does not trigger a failover. For more information see the following topics:

- Adding an Amazon DocumentDB instance to a cluster
- Modifying an Amazon DocumentDB instance

When manually initiating a failover, you have two means to control which replica instance is promoted to primary: the failover tiers as previously described, and the `--target-db-instance-identifier` parameter.

`--target-db-instance-identifier`

For testing, you can force a failover event using the `failover-db-cluster` operation. You can use the `--target-db-instance-identifier` parameter to specify which replica to promote to primary. Using the `--target-db-instance-identifier` parameter supersedes the failover priority tier. If you do not specify the `--target-db-instance-identifier` parameter, the primary failover is in accordance with the failover priority tier.

What happens during a failover

Failover is automatically handled by Amazon DocumentDB so that your applications can resume database operations as quickly as possible without administrative intervention.

- If you have an Amazon DocumentDB replica instance in the same or different Availability Zone when failing over: Amazon DocumentDB flips the canonical name record (CNAME) for your instance to point at the healthy replica, which is, in turn, promoted to become the new primary. Failover typically completes within 30 seconds from start to finish.
- If you don't have an Amazon DocumentDB replica instance (for example, a single instance cluster): Amazon DocumentDB will attempt to create a new instance in the same Availability Zone as the original instance. This replacement of the original instance is done on a best-

effort basis and may not succeed if, for example, there is an issue that is broadly affecting the Availability Zone.

Your application should retry database connections in the event of a connection loss.

Testing failover

A failover for a cluster promotes one of the Amazon DocumentDB replicas (read-only instances) in the cluster to be the primary instance (the cluster writer).

When the primary instance fails, Amazon DocumentDB automatically fails over to an Amazon DocumentDB replica, if one exists. You can force a failover when you want to simulate a failure of a primary instance for testing. Each instance in a cluster has its own endpoint address. Therefore, you need to clean up and re-establish any existing connections that use those endpoint addresses when the failover is complete.

To force a failover, use the `failover-db-cluster` operation with these parameters.

- `--db-cluster-identifier`—Required. The name of the cluster to fail over.
- `--target-db-instance-identifier`—Optional. The name of the instance to be promoted to the primary instance.

Example

The following operation forces a failover of the `sample-cluster` cluster. It does not specify which instance to make the new primary instance, so Amazon DocumentDB chooses the instance according to failover tier priority.

For Linux, macOS, or Unix:

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster
```

For Windows:

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster
```

The following operation forces a failover of the `sample-cluster` cluster, specifying that `sample-cluster-instance` is to be promoted to the primary role. (Notice `"IsClusterWriter": true` in the output.)

For Linux, macOS, or Unix:

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-cluster-instance
```

For Windows:

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --target-db-instance-identifier sample-cluster-instance
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBCluster": {
        "HostedZoneId": "Z2SUY0A1719RZT",
        "Port": 27017,
        "EngineVersion": "3.6.0",
        "PreferredMaintenanceWindow": "thu:04:05-thu:04:35",
        "BackupRetentionPeriod": 1,
        "ClusterCreateTime": "2018-06-28T18:53:29.455Z",
        "AssociatedRoles": [],
        "DBSubnetGroup": "default",
        "MasterUsername": "master-user",
        "Engine": "docdb",
        "ReadReplicaIdentifiers": [],
        "EarliestRestorableTime": "2018-08-21T00:04:10.546Z",
        "DBClusterIdentifier": "sample-cluster",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-cluster-instance",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": true
            },
            {
```

```
                "DBInstanceIdentifier": "sample-cluster-instance-00",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": false
            },
            {
                "DBInstanceIdentifier": "sample-cluster-instance-01",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": false
            }
        ],
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1a"
        ],
        "DBClusterParameterGroup": "default.docdb3.6",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "IAMDatabaseAuthenticationEnabled": false,
        "AllocatedStorage": 1,
        "LatestRestorableTime": "2018-08-22T21:57:33.904Z",
        "PreferredBackupWindow": "00:00-00:30",
        "StorageEncrypted": false,
        "MultiAZ": true,
        "Status": "available",
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-12345678"
            }
        ],
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    }
}
```

Replication lag

Replication lag is typically 50ms or less. The most common reasons for increased replica lag are:

- A high write rate on the primary that causes the read replicas to fall behind the primary.

- Contention on the read replicas between long running queries (e.g., large sequential scans, aggregation queries) and incoming write replication.
- Very large number of concurrent queries on the read replicas.

To minimize replication lag, try these troubleshooting techniques:

- If you have a high write rate or high CPU utilization, we recommend that you scale up the instances in your cluster.
- If there are long running queries on your read replicas, and very frequent updates to the documents being queried, consider altering your long running queries, or running them against the primary/write replica to avoid contention on the read replicas.
- If there is a very large number of concurrent queries or high CPU utilization only on the read replicas, another option is to scale out the number of read replicas to spread out the workload.
- Because replication lag is a result of high write throughput and long running queries, we recommend troubleshooting the replication lag by utilizing the DBClusterReplicaLagMaximum CW metric in combination with the slow query logger and `WriteThroughput/WriteIOPS` metrics.

In general, we recommend that all your replicas are of the same instance type, so that a cluster failover will not cause a degradation in performance.

If you are choosing between scaling up and scaling out (eg. six smaller instances vs three larger instances), we generally recommend trying to scale up first (larger instances) before scaling out, as you will get a larger buffer cache per DB instance.

Proactively, you should set a replication lag alarm and set its threshold to a value that you feel is the upper bound for how far behind (or “stale”) your data on replica instances can be before it starts affecting the functionality of your application. In general, we would advise that the replication lag threshold be exceeded for several data points before alarming, due to transient workloads.

> **Note**
>
> In addition, we recommend that you set another alarm for replication lags that exceed 10 seconds. If you surpass this threshold for multiple data points, we recommend that you scale up your instances or reduce your write throughput on the primary instance.

Managing Amazon DocumentDB indexes

Topics

- Amazon DocumentDB index creation
- Maintaining Amazon DocumentDB indexes

Amazon DocumentDB index creation

Building indexes in Amazon DocumentDB requires a number of decisions to be made:

- How quickly does it need to be completed?
- Can the collection be inaccessible while the build is occurring?
- How much of an instances compute power can be allocated to the build?
- What type of index should be created?

This section helps you answer these questions and provides the commands and monitoring examples to create and Amazon DocumentDB index on your instance-based cluster collection.

Guidelines

The following guidelines include basic limits and configuration tradeoffs when creating new indexes:

- **Amazon DocumentDB version support** - While single worker indexing is supported on all Amazon DocumentDB versions, multiple worker indexing is supported only on Amazon DocumentDB versions 4.0 and 5.0.
- **Performance trade-off** - Increasing the number of workers in the index creation process increases CPU utilization and read IO on the primary instance of your Amazon DocumentDB database. The resources needed to create a new index will not be available to your running workload.
- **Elastic clusters** - Parallel indexing is not supported on Amazon DocumentDB elastic clusters.
- **Maximum workers** - The maximum number of workers you can configure depends on the size of your primary instance in your database cluster. It is half the total number of vCPUs on the primary instance of your database cluster. For example, you can run a maximum of 32 workers on a db.r6g.16xlarge instance that has 64 vCPUs.

> **Note**
>
> Parallel workers are not supported on 2xlarge instance classes and lower.

- **Minimum workers** - The minimum number of workers you can configure is one. The default setting for index creation on instance-based clusters is two workers. However, you can reduce the number of workers to one by using the “worker threads” option. This will run the process with a single worker.
- **Index compression** - Amazon DocumentDB doesn't support index compression. Data sizes for indexes might be larger than when you use other options.
- **Indexing multiple collections** - Half the vCPUs on your database cluster's primary instance can be used for configured workers performing index creation on multiple collections.
- **Index types** - See this blog post for a complete explanation of supported index types on Amazon DocumentDB.

Getting started

To start index creation on a collection, use the `createIndexes` command. By default, the command will run two parallel workers that increases the speed of the index creation process by two times.

For example, the following command process demonstrates how to create an index for the “user_name” field in a document and increase the indexing process speed to four workers:

1. Create indexes using two parallel workers on the cluster:

   ```
   db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
    "name":"username_idx"}]})
   ```

2. To optimize the speed of the index creation process, you can specify the number of workers by using the “worker threads” option (`"workers":<number>`) in the `db.runCommand` `createIndexes` command.

   Increase the speed of the process to four parallel workers:

   ```
   db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
    "name":"username_idx", "workers":4}]})
   ```

> **Note**
>
> The higher the number of workers, the faster the index creation progresses. However, the higher the number of workers increases, the higher the load increases on the vCPUs and read IO of your primary instance. Ensure that your cluster is sufficiently provisioned to handle the increased burden without degrading other workloads.

Indexing progress status

The index creation process works by initializing, scanning collections, sorting keys, and, finally, inserting keys by way of an index builder. The process has up to six stages when you run it in the foreground, and up to nine stages when you run it in the background. You can view status metrics such as percentage completion, total number of scanned storage blocks, sorted keys, and inserted keys on stage by stage basis.

Monitor the progress on the indexing process by using the `db.currentOp()` command in the mongo shell. A 100% completion of the last stage shows that all the indexes have been successfully created:

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> Viewing the indexing progress status is only supported on Amazon DocumentDB 5.0.

Index build types

The four types of index builds are:

- **Foreground** — The foreground index build blocks all other database operations until the index is created. The Amazon DocumentDB foreground build is comprised of five stages.
- **Foreground (unique)** — Single document (unique) foreground index builds block other database operations like regular foreground builds. Unlike the basic foreground build, the unique build uses an additional stage (sorting keys 2) to look for duplicate keys. The foreground (unique) build is comprised of six stages.

- **Background** — The background index build allows other database operations to run in the foreground while the index is being created. The Amazon DocumentDB background build is comprised of eight stages.
- **Background (unique)** — Single document (unique) background index builds allow other database operations to run in the foreground while the index is being created. Unlike the basic background build, the unique build uses an additional stage (sorting keys 2) to look for duplicate keys. The background (unique) build is comprised of nine stages.

Index build stages

Stage	Foreground	Foreground (unique)	Background	Background (unique)
Initializing	1	1	1	1
building index: initializing	2	2	2	2
building index: scanning collection	3	3	3	3
building index: sorting keys 1	4	4	4	4
building index: sorting keys 2		5		5
building index: inserting keys	5	6	5	6
validating: scanning index			6	7
validating: sorting tuples			7	8

Stage	Foreground	Foreground (unique)	Background	Background (unique)
validating: scanning collection			8	9

- **initializing** - createIndex is preparing the index builder. This phase should be very brief.
- **building index: initializing** - The index builder is preparing to create the index. This phase should be very brief.
- **building index: scanning collection** - The index builder is performing a collection scan to collect index keys. The unit of measure is “blocks”.

> **Note**
>
> If more than one worker is configured for the index build, it is displayed in this stage. The “scanning collection” stage is the only stage that uses multiple workers during the index build process. All other stages will display a single worker.

- **building index: sorting keys 1** - The index builder is sorting the collected index keys. The unit of measure is “keys”.
- **building index: sorting keys 2** - The index builder is sorting the collected index keys that correspond to dead tuples. This phase only exists for unique index building. The unit of measure is “keys”.
- **building index: inserting keys** - The index builder is inserting index keys into the new index. The unit of measure is “keys”.
- **validating: scanning index** - createIndex is scanning the index to find keys that need to be validated. The unit of measure is “blocks”.
- **validating: sorting tuples** - createIndex is sorting the output of the index scanning phase.
- **validating: scanning collection** - createIndex is scanning the collection to validate the index keys found in the previous two phases. The unit of measure is “blocks”.

Index build output example

In the output example below (foreground index build), the status of the index creation is shown. The “msg” field summarizes the build progress by indicating the stage and the completion percentage of the build. The “workers” field indicates the number of workers used during that stage of the index build. The “progress” field shows the actual numbers used to calculate the percentage of completion.

> **Note**
>
> The “currentIndexBuildName”, “msg”, and “progress” fields are not supported on Amazon DocumentDB version 4.0.

```
{
    "inprog" : [{
    …
        "command": {
            "createIndexes": "test",
            "indexes": [{
                "v": 2,
                "key": {
                    "user_name": 1
                },
                "name": "user_name_1"
            }],
            "lsid": {
                "id": UUID(“094d0fba-8f41-4373-82c3-7c4c7b5ff13b”)
            },
            "$db": "test"
        },
        "currentIndexBuildName": user_name_1,
        "msg": "Index Build: building index number_1, stage 6/6 building index:
 656860/1003520 (keys) 65%",
        "workers": 1,
        "progress": {
            "done": 656861,
            "total": 1003520
        },
    …
    ],
```

```
    "ok" : 1
}
```

Maintaining Amazon DocumentDB indexes

Index maintenance using `reIndex`

`reIndex` is a command used to rebuild an index. It is typically used when an index has become corrupted or inefficient. Over time, indexes can accumulate unused space due to many updates, inserts, or deletes, leading to degraded performance. Reindexing helps to remove such unused space and restore the efficiency of the index.

`reIndex` guidelines

- `reIndex` is only supported on Amazon DocumentDB 5.0.
- Amazon DocumentDB supports `reindex` of a single index in the background, allowing for multiple workers. The old index is usable by queries when the `reIndex` process is running.
- Amazon DocumentDB supports indexing progress report through `currentOp`. You can see index build stages similar to the Index build stages viewed during index creation. The only difference is that `reIndex` always has eight stages, regardless if it's unique or not. There's no "building index: sorting keys 2" stage.
- `reIndex` can run concurrently with any command except index-related commands on the same collection: `createIndexes`, `dropIndexes`, `collMod`, and `renameCollection`.
- `reIndex` is currently not supported for text, geospatial, vector, and partial indexes.

`reIndex` build

Use the following command to rebuild your index:

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

Optionally, you can also control the number of workers assigned to the rebuild process:

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

Managing collection-level document compression

Amazon DocumentDB collection-level document compression allows you to lower storage and IO costs by compressing the documents in your collections. You can enable document compression at a collection level and view compression metrics as needed by measuring the storage gains through compression metrics such as storage size of compressed documents and compression status. Amazon DocumentDB uses the LZ4 compression algorithm to compress documents.

Amazon DocumentDB supports document compression starting with version 5.0. The following are collection-level document compression functions:

- **Default behavior** — The default compression setting for new collections on a cluster is determined by the cluster parameter `default_collection_compression`. This parameter is set to “disabled” by default.
- **Compressing existing collections** — The compression setting for existing collections can be changed using the `collMod` command.
- **Changing the compression threshold** — The default compression threshold is 2KB. This value can be specified for new collections using the `createCollection` command, and changed for existing collections using `collMod` command.

> **Note**
>
> Amazon DocumentDB document compression is not supported on Amazon DocumentDB versions 3.6 and 4.0.

Topics

- Managing document compression
- Monitoring document compression

Managing document compression

Enabling document compression in a collection

Enable document compression while creating a collection onAmazon DocumentDB by using `db.createCollection()` method:

```
db.createCollection( sample_collection,{
    storageEngine : {
        documentDB: {
            compression:{enable: <true | false>}
        }
    }
})
```

Enabling document compression in a cluster

Document compression can be enabled by default for all new collections on a cluster level by setting the cluster parameter `default_collection_compression` to “enabled”. When this parameter is set to “enabled”, newly created collections on the cluster will have compression enabled by default with a compression threshold of 2 KB.

Compressing existing collections

You can also modify the compression settings for an existing collection using the `collMod` operation and specifying the following `storageEngine` configuration. Please note that the change made using this command will only apply to newly inserted/updated documents and the compression on previously inserted documents will not change.

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {compression: {enable: <true | false>} }
    }
})
```

Setting the compression thresholds

By default, the compression threshold for compressed collections is 2032 bytes. This threshold value can be set in the `createCollection` command when creating a new collection with compression enabled:

```
db.createCollection( sample_collection, {
    storageEngine : {
        documentDB: {
            compression: {
                enable: true,
```

```
                threshold: <128 - 8000>
            }
        }
    }
})
```

You can also modify the compression threshold for an existing compressed collection using the `collMod` operation and specifying the following `storageEngine` configuration:

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

Please note that the compression threshold can only be set to a value between 128 to 8000 bytes. Additionally, the `enable` option needs to be set to “true” when specifying the compression threshold.

Monitoring document compression

You can check if a collection is compressed and calculate it's compression ratio as follows.

View compression statistics by running the `db.printCollectionStats()` or `db.collection.stats()` command from the mongo shell. The output shows you the original size and compressed size that you can compare to analyze the storage gains from document compression. In this example, statistics for a collection named “sample_collection” are shown below. A scaling factor of 1024*1024 is used below to output the `size` and `storageSize` values in MB.

```
db.sample_collection.stats(1024*1024)
```

The following is an example of the output for the above command:

```
{
```

```
    "ns" : "test.sample_collection",
    "count" : 1000000,
    "size" : 3906.3,
    "avgObjSize" : 4096,
    "storageSize" : 1953.1,
    compression:{"enabled" : true,"threshold" : 2032},
    ...
}
```

- **size** - The original size of the document collection.
- **avgObjSize** - The average document size before compression rounded off to first decimal. The unit of measure is bytes.
- **storageSize** - The storage size of the collection after compression. The unit of measure is bytes.
- **enabled** - Indicates if compression is enabled or disabled.

To calculate the actual compression ratio, divide the collection size by the storage size (size/storageSize). For the example above, the calculation is 3906.3/1953.1 which translates to a 2:1 compression ratio.

Managing Amazon DocumentDB events

Amazon DocumentDB (with MongoDB compatibility) keeps a record of events that relate to your clusters, instances, snapshots, security groups, and cluster parameter groups. This information includes the date and time of the event, the source name and source type of the event, and a message that is associated with the event.

> **Important**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon RDS and Amazon Neptune. Region limits, limits that are governed at the Region level, are shared between Amazon DocumentDB, Amazon RDS, and Amazon Neptune. For more information, see Regional quotas.

Topics

- Viewing Amazon DocumentDB event categories
- Viewing Amazon DocumentDB events

Viewing Amazon DocumentDB event categories

Each Amazon DocumentDB resource type has specific types of events that can be associated with it. You can use the AWS CLI `describe-event-categories` operation to view the mapping between event types and Amazon DocumentDB resource types.

Parameters

- **`--source-type`**—Optional. Use the `--source-type` parameter to see the event categories for a particular source type. The following are permitted values:
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`
  - `db-cluster-snapshot`
- **`--filters`**—Optional. To view the event categories for just Amazon DocumentDB, use the filter `--filter Name=engine,Values=docdb`.

Example

The following code lists the event categories associated with clusters.

For Linux, macOS, or Unix:

```
aws docdb describe-event-categories \
    --filter Name=engine,Values=docdb \
    --source-type db-cluster
```

For Windows:

```
aws docdb describe-event-categories ^
    --filter Name=engine,Values=docdb ^
    --source-type db-cluster
```

Output from this operation looks something like the following (JSON format).

```
{
    "EventCategoriesMapList": [
        {
```

```
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ],
            "SourceType": "db-cluster"
        }
    ]
}
```

The following code lists the event categories that are associated with each Amazon DocumentDB source type.

```
aws docdb describe-event-categories
```

Output from this operation looks something like the following (JSON format).

```
{
    "EventCategoriesMapList": [
        {
            "SourceType": "db-instance",
            "EventCategories": [
                "notification",
                "failure",
                "creation",
                "maintenance",
                "deletion",
                "recovery",
                "restoration",
                "configuration change",
                "read replica",
                "backtrack",
                "low storage",
                "backup",
                "availability",
                "failover"
            ]
        },
        {
            "SourceType": "db-security-group",
            "EventCategories": [
                "configuration change",
```

```
                "failure"
            ]
        },
        {
            "SourceType": "db-parameter-group",
            "EventCategories": [
                "configuration change"
            ]
        },
        {
            "SourceType": "db-cluster",
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ]
        },
        {
            "SourceType": "db-cluster-snapshot",
            "EventCategories": [
                "backup"
            ]
        }
    ]
}
```

Viewing Amazon DocumentDB events

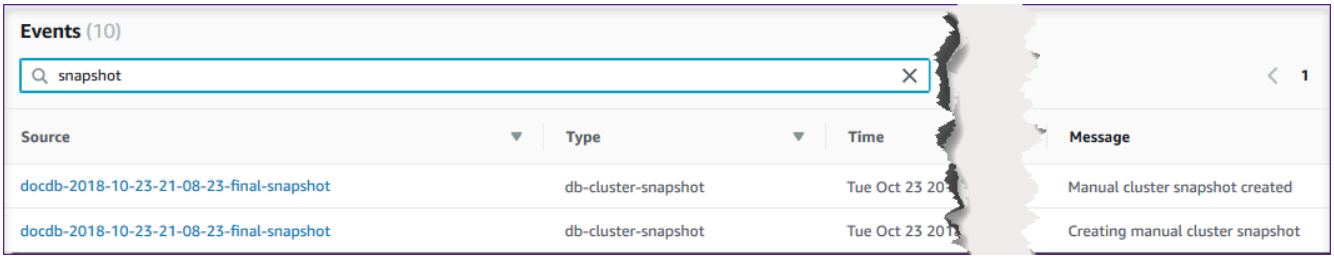
You can retrieve events for your Amazon DocumentDB resources through the Amazon DocumentDB console, which shows events from the past 24 hours. You can also retrieve events for your Amazon DocumentDB resources by using the describe-events AWS CLI command, or the DescribeEvents Amazon DocumentDB API operation. If you use the AWS CLI or the Amazon DocumentDB API to view events, you can retrieve events for up to the past 14 days.

Using the AWS Management Console

To view all Amazon DocumentDB instance events for the past 24 hours

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Events**. The available events appear in a list.

3. Use the **Filter** list to filter the events by type. Enter a term in the text box to further filter your results. For example, the following screenshot shows filtering all Amazon DocumentDB events for *snapshot* events.



Using the AWS CLI

To view all Amazon DocumentDB instance events for the past 7 days

You can view all Amazon DocumentDB instance events for the past 7 days by running the describe-events AWS CLI operation with the `--duration` parameter set to `10080` (10,080 minutes).

```
aws docdb describe-events --duration 10080
```

Filtering for Amazon DocumentDB Events

To see specific Amazon DocumentDB events, use the `describe-events` operation with the following parameters.

Parameters

- **`--filter`**—Required to limit returned values to Amazon DocumentDB events. Use **`Name=engine,Values=docdb`** to filter all events for Amazon DocumentDB only.
- **`--source-identifier`**—Optional. The identifier of the event source for which events are returned. If omitted, events from all sources are included in the results.
- **`--source-type`**—Optional, unless `--source-identifier` is provided, then required. If `--source-identifier` is provided, `--source-type` must agree with the type of the `--source-identifier`. The following are permitted values:
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`

- `db-cluster-snapshot`

The following example lists all your Amazon DocumentDB events.

```
aws docdb describe-events --filters Name=engine,Values=docdb
```

Output from this operation looks something like the following (JSON format).

```
{
    "Events": [
        {
            "SourceArn": "arn:aws:rds:us-east-1:123SAMPLE012:db:sample-cluster-
instance3",
            "Message": "instance created",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:17:40.023Z",
            "SourceIdentifier": "sample-cluster-instance3",
            "EventCategories": [
                "creation"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance shutdown",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:01.245Z",
            "SourceIdentifier": "docdb-2018-12-11-21-08-23",
            "EventCategories": [
                "availability"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance restarted",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:11.441Z",
            "SourceIdentifier": "docdb-2018-12-11-21-08-23",
            "EventCategories": [
                "availability"
            ]
```

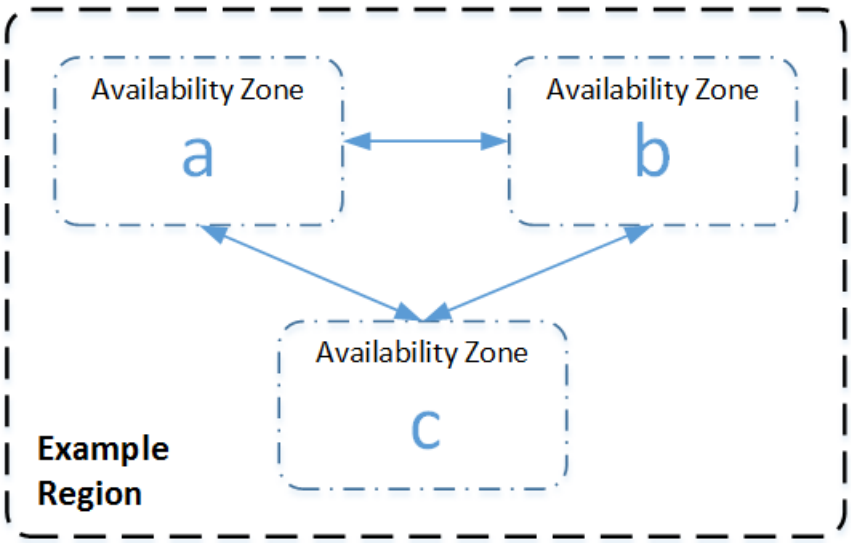
```
        }
    ]
}
```

For more information, see Auditing Amazon DocumentDB events.

Choosing regions and availability zones

Amazon cloud computing resources are hosted in multiple locations worldwide. These locations consist of AWS Regions and Availability Zones. Each *AWS Region* is a separate geographic area. Each Region has multiple, isolated locations known as *Availability Zones*. Amazon DocumentDB provides you the ability to place resources, such as instances, and data in multiple locations. Resources aren't replicated across AWS Regions unless you do so specifically.

Amazon operates advanced, highly available data centers. Although rare, failures can occur that affect the availability of instances that are in the same location. If you host all your instances in a single location that is affected by such a failure, none of your instances would be available. The following diagram shows an AWS Region with three Availability Zones.



It is important to remember that each Region is independent. Any Amazon DocumentDB activity that you initiate (for example, creating instances or listing available instances) runs only in your current default AWS Region. You can change the default Region on the console by setting the `EC2_REGION` environment variable. Or you can override it by using the `--region` parameter in the AWS CLI. For more information, see Configuring the AWS Command Line Interface, specifically, the sections on environment variables and command line options.

When you create a cluster using the Amazon DocumentDB console, and you choose to create a replica in a different Availability Zone, Amazon DocumentDB creates two instances. It creates the primary instance in one Availability Zone and the replica instance in a different Availability Zone. The cluster volume is always replicated across three Availability Zones.

To create or work with an Amazon DocumentDB instance in a specific AWS Region, use the corresponding regional service endpoint.

Region availability

Amazon DocumentDB is available in the following AWS Regions.

Regions supported by Amazon DocumentDB

Region Name	Region	Availability Zones (compute)
US East (Ohio)	`us-east-2`	3
US East (N. Virginia)	`us-east-1`	6
US West (Oregon)	`us-west-2`	4
Africa (Cape Town)	`af-south-1`	3
South America (São Paulo)	`sa-east-1`	3
Asia Pacific (Hong Kong)	`ap-east-1`	3
Asia Pacific (Hyderabad)	`ap-south-2`	3
Asia Pacific (Mumbai)	`ap-south-1`	3
Asia Pacific (Seoul)	`ap-northeast-2`	4
Asia Pacific (Singapore)	`ap-southeast-1`	3
Asia Pacific (Sydney)	`ap-southeast-2`	3

Region Name	Region	Availability Zones (compute)
Asia Pacific (Tokyo)	`ap-northeast-1`	3
Canada (Central)	`ca-central-1`	3
China (Beijing) Region	`cn-north-1`	3
China (Ningxia)	`cn-northwest-1`	3
Europe (Frankfurt)	`eu-central-1`	3
Europe (Ireland)	`eu-west-1`	3
Europe (London)	`eu-west-2`	3
Europe (Milan)	`eu-south-1`	3
Europe (Paris)	`eu-west-3`	3
Europe (Spain)	`eu-south-2`	3
Europe (Stockholm)	`eu-north-1`	3
Middle East (UAE)	`me-central-1`	3
Israel (Tel Aviv)	`il-central-1`	3
AWS GovCloud (US-West)	`us-gov-west-1`	3
AWS GovCloud (US-East)	`us-gov-east-1`	3

By default, the time zone for an Amazon DocumentDB cluster is Universal Time Coordinated (UTC).

For information on finding the connection endpoints for clusters and instances in a particular region, see Understanding Amazon DocumentDB endpoints.

Managing Amazon DocumentDB cluster parameter groups

You can manage Amazon DocumentDB engine configuration by using parameters in a cluster parameter group. A *cluster parameter group* is a collection of Amazon DocumentDB configuration values that make it easier to manage the parameters of your Amazon DocumentDB clusters. Cluster parameter groups act as a container for engine configuration values that are applied to all instances in the cluster.

This section describes how to create, view, and modify cluster parameter groups. It also shows how you can determine which cluster parameter group is associated with a given cluster.

Topics

- Describing Amazon DocumentDB cluster parameter groups
- Creating Amazon DocumentDB cluster parameter groups
- Modifying Amazon DocumentDB cluster parameter groups
- Modifying Amazon DocumentDB clusters to use custom cluster parameter groups
- Copying Amazon DocumentDB cluster parameter groups
- Resetting Amazon DocumentDB cluster parameter groups
- Deleting Amazon DocumentDB cluster parameter groups
- Amazon DocumentDB cluster parameters reference

Describing Amazon DocumentDB cluster parameter groups

A `default` cluster parameter group is created automatically when you create the first Amazon DocumentDB cluster in new region or are using a new engine. Subsequent clusters, which are created in the same region and have the same engine version, are created with the `default` cluster parameter group.

Topics

- Describing the details of an Amazon DocumentDB cluster parameter group
- Determining an Amazon DocumentDB cluster's parameter group

Describing the details of an Amazon DocumentDB cluster parameter group

To describe the details of a given cluster parameter group, complete the following steps using the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the **Cluster parameter groups** pane, select the name of the parameter group that you want to see the details of.
4. The resulting page shows the parameter group's parameters, recent activity, and tags.

   - Under **Cluster parameters**, you can see the parameter's name, current value, allowed values, whether the parameter is modifiable, its apply type, data type, and description. You can modify individual parameters by selecting the parameter and then choosing **Edit** in the **Cluster parameters** section. For more information, see Modifying Amazon DocumentDB cluster parameters.
   - Under **Recent events**, you can see the most recent events for this parameter group. You can filter through these events using the search bar in this section. For more information, see Managing Amazon DocumentDB events.
   - Under **Tags**, you can see the tags that are on this cluster parameter group. You can add or remove tags by choosing **Edit** in the **Tags** section. For more information, see Tagging Amazon DocumentDB resources.

Using the AWS CLI

You can use the `describe-db-cluster-parameter-groups` AWS CLI command to view the Amazon Resource Name (ARN), family, description, and name of a single cluster parameter group or all cluster parameter groups that you have for Amazon DocumentDB. You can also use the `describe-db-cluster-parameters` AWS CLI command to view the parameters and their details inside a single cluster parameter group.

- **`--describe-db-cluster-parameter-groups`** — To see a listing of all your cluster parameter groups and their details.
  - **`--db-cluster-parameter-group-name`** — Optional. The name of the cluster parameter group that you want described. If this parameter is omitted, all cluster parameter groups are described.
- **`--describe-db-cluster-parameters`** — To list all the parameters inside a parameter group and their values.
  - **`--db-cluster-parameter-group name`** — Required. The name of the cluster parameter group that you want described.

Example

The following code lists up to 100 cluster parameter groups and their ARN, family, description, and name.

```
aws docdb describe-db-cluster-parameter-groups
```

Output from this operation looks something like the following (JSON format).

```
{
          "DBClusterParameterGroups": [
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:012345678912:cluster-pg:default.docdb4.0",
                  "DBParameterGroupFamily": "docdb4.0",
                  "Description": "Default cluster parameter group for docdb4.0",
                  "DBClusterParameterGroupName": "default.docdb4.0"
              },
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:012345678912:cluster-pg:sample-parameter-group",
```

```
                    "DBParameterGroupFamily": "docdb4.0",
                    "Description": "Custom docdb4.0 parameter group",
                    "DBClusterParameterGroupName": "sample-parameter-group"
                }
            ]
}
```

Example

The following code lists the ARN, family, description, and name for `sample-parameter-group`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameter-groups \
          --db-cluster-parameter-group-name sample-parameter-group
```

For Windows:

```
aws docdb describe-db-cluster-parameter-groups ^
          --db-cluster-parameter-group-name sample-parameter-group
```

Output from this operation looks something like the following (JSON format).

```
{
          "DBClusterParameterGroups": [
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:cluster-pg:sample-parameter-group",
                  "Description": "Custom docdb4.0 parameter group",
                  "DBParameterGroupFamily": "docdb4.0",
                  "DBClusterParameterGroupName": "sample-parameter-group"
              }
          ]
}
```

Example

The following code lists the values of the parameters in `sample-parameter-group`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

For Windows:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

Output from this operation looks something like the following (JSON format).

```
{
   "Parameters": [
         {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
         },
         {
            "ParameterName": "change_stream_log_retention_duration",
            "ParameterValue": "17777",
            "Description": "Duration of time in seconds that the change stream log
 is retained and can be consumed.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "3600-86400",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
         }
   ]
}
```

Determining an Amazon DocumentDB cluster's parameter group

To determine which parameter group is associated with a particular cluster, complete the following steps using the AWS Management Console or the AWS CLI.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the left navigation pane, choose **Clusters**.
3. In the list of clusters, select the name of the cluster you are interested in.
4. The resulting page shows the details of the cluster that you selected. Scroll down to **Cluster details**. At the bottom of that section, locate the parameter group's name below **Cluster parameter group**.

Cluster details

Configurations and status

ARN

arn:aws:rds: :cluster:sample-cluster

Cluster identifier

sample-cluster (available)

Cluster creation time

1/10/2020, 2:13:38 PM UTC-8

Cluster endpoint

sample-cluster.
.docdb.amazonaws.com

Reader endpoint

sample-cluster.
.docdb.amazonaws.com

Master username

Port

27017

Status

available

Cluster parameter group

sample-parameter-group

Deletion protection

Enabled

CloudWatch logs enabled

None

Using the AWS CLI

The following AWS CLI code determines which parameter group is governing the cluster `sample-cluster`.

```
aws docdb describe-db-clusters \
      --db-cluster-identifier sample-cluster \
      --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

Output from this operation looks something like the following (JSON format).

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

Creating Amazon DocumentDB cluster parameter groups

Default cluster parameter groups such as `default.docdb5.0`, `default.docdb4.0`, or `default.docdb3.6`, are created when you create a cluster with a new engine version and in a new region. Subsequent clusters created in this region and with the same engine version inherit the `default` cluster parameter group. Once created, the `default` parameter groups cannot be deleted or renamed. You can modify the engine behavior of cluster instances by creating a custom parameter group with preferred parameter values and attaching it to your Amazon DocumentDB cluster.

The following procedure guides you through creating a custom cluster parameter group. You can then modify the parameters within that parameter group.

> **Note**
>
> After you create a cluster parameter group, you should wait at least 5 minutes before using that particular parameter group. This allows Amazon DocumentDB to fully complete the `create` action before the cluster parameter group is used for a new cluster. You can use the AWS Management Console or the `describe-db-cluster-parameter-groups`

AWS CLI operation to verify that your cluster parameter group has been created. For more information, see Describing Amazon DocumentDB cluster parameter groups.

Using the AWS Management Console

To create a cluster parameter group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. In the **Cluster parameter groups** pane, choose **Create**.
4. In the **Create cluster parameter group** pane, enter the following:

   a. **Group name** — Enter a name for the cluster parameter group. For example, `sample-parameter-group`. Cluster parameter groups have the following naming constraints:

      - Length is [1–255] alphanumeric characters.
      - First character must be a letter.
      - Cannot end with a hyphen or contain two consecutive hyphens.

   b. **Description** — Provide a description for this cluster parameter group.

5. To create the cluster parameter group, choose **Create**. To cancel the operation, choose **Cancel**.
6. After you choose **Create**, the following text appears at the top of the page to verify that your cluster parameter group has been successfully created:

```
Successfully created cluster parameter group 'sample-parameter-group'.
```

Using the AWS CLI

To create a new cluster parameter group for Amazon DocumentDB 4.0 clusters, use the AWS CLI `create-db-cluster-parameter-group` operation with the following parameters:

- **`--db-cluster-parameter-group-name`** — The name of the custom cluster parameter group. For example, `sample-parameter-group`.
- **`--db-cluster-parameter-group-family`** — The cluster parameter group family that is used as the template for the custom cluster parameter group. Currently, this must be `docdb4.0`.
- **`--description`** — The user-provided description for this cluster parameter group. The following example uses "`Custom docdb4.0 parameter group`".

For Linux, macOS, or Unix:

Example

```
aws docdb create-db-cluster-parameter-group \
 --db-cluster-parameter-group-name sample-parameter-group \
 --db-parameter-group-family docdb4.0 \
 --description "Custom docdb4.0 parameter group"
```

For Windows:

```
aws docdb create-db-cluster-parameter-group ^
 --db-cluster-parameter-group-name sample-parameter-group ^
 --db-parameter-group-family docdb4.0 ^
 --description "Custom docdb4.0 parameter group"
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "sample-parameter-group",
        "DBParameterGroupFamily": "docdb4.0",
        "Description": "Custom docdb4.0 parameter group",
        "DBClusterParameterGroupArn": "sample-parameter-group-arn"
    }
}
```

Modifying Amazon DocumentDB cluster parameter groups

This section explains how to modify a *custom* Amazon DocumentDB parameter group. In Amazon DocumentDB, you cannot modify a `default` cluster parameter group which is created when you first create a cluster with new engine version in a new region. If your Amazon DocumentDB cluster is using the default cluster parameter group and you want to modify a value in it, you must first create a new parameter group or copy an existing parameter group, modify it, and then apply the modified parameter group to your cluster.

Complete the following steps to modify a custom cluster parameter group. Modify actions could take a while to propagate. Please wait for the modified cluster paramater group to be available before attaching it to your cluster. You can use the AWS Management Console or the AWS CLI `describe-db-cluster-parameters` operation to verify that your cluster parameter group has been modified. For more information, see Describing cluster parameter groups.

Using the AWS Management Console

Follow these steps to modify a custom Amazon DocumentDB parameter group. You can't modify a `default` parameter group. If you want to modify a value in the `default` parameter group, you can copy the default cluster parameter group, modify it, and then apply the modified parameter group to your cluster. For more information about applying parameter groups to your cluster, see Modifying an Amazon DocumentDB cluster.

To modify a custom cluster parameter group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane on the left side of the console, choose **Parameter groups**. In the list of parameter groups, choose the name of the parameter group that you want to modify.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. For each parameter in the parameter group that you want to modify, do the following:

a. Locate the parameter that you want to modify, and verify that it is modifiable by checking if it is listed as `true` under the **Modifiable** column.

b. If it is modifiable, select the parameter and choose **Edit** from the top right of the console page.

c. In the **Modify <parameter-name>** dialog box, make the changes that you want. Then choose **Modify cluster parameter**, or choose **Cancel** to discard the changes.

Using the AWS CLI

You can modify the `ParameterValue`, `Description`, or `ApplyMethod` of any modifiable parameter in a custom Amazon DocumentDB cluster parameter group using the AWS CLI. You can't make modifications directly to a default cluster parameter group.

To modify a custom cluster parameter group's parameters, use the `modify-db-cluster-parameter-group` operation with the following parameters.

- **`--db-cluster-parameter-group-name`** — Required. The name of the cluster parameter group that you are modifying.
- **`--parameters`** — Required. The parameters that you are modifying. For a list of the parameters that apply to all instances in an Amazon DocumentDB cluster, see the Amazon DocumentDB cluster parameters reference. Each parameter entry must include the following:
  - **`ParameterName`** — The name of the parameter that you are modifying.
  - **`ParameterValue`** — The new value for this parameter.
  - **`ApplyMethod`** — How you want changes to this parameter applied. Permitted values are `immediate` and `pending-reboot`.

> **Note**
>
> Parameters with the `ApplyType` of `static` must have an `ApplyMethod` of `pending-reboot`.

Example - Modifying a parameter's value

In this example, you list the parameter values of `sample-parameter-group` and modify the `tls` parameter. Then, after waiting 5 minutes, you again list the parameter values of `sample-parameter-group` to see the changed parameter values.

1. List the parameters and their values of `sample-parameter-group`.

   For Linux, macOS, or Unix:

   ```
   aws docdb describe-db-cluster-parameters \
         --db-cluster-parameter-group-name sample-parameter-group
   ```

   For Windows:

   ```
   aws docdb describe-db-cluster-parameters ^
         --db-cluster-parameter-group-name sample-parameter-group
   ```

   Output from this operation looks something like the following (JSON format).

   ```
   {
         "Parameters": [
             {
                 "Source": "system",
                 "ApplyType": "static",
                 "AllowedValues": "disabled,enabled",
                 "ParameterValue": "enabled",
                 "ApplyMethod": "pending-reboot",
                 "DataType": "string",
                 "ParameterName": "tls",
                 "IsModifiable": true,
                 "Description": "Config to enable/disable TLS"
             },
             {
                 "Source": "user",
                 "ApplyType": "dynamic",
                 "AllowedValues": "disabled,enabled",
                 "ParameterValue": "enabled",
                 "ApplyMethod": "pending-reboot",
                 "DataType": "string",
                 "ParameterName": "ttl_monitor",
                 "IsModifiable": true,
   ```

```
                "Description": "Enables TTL Monitoring"
            }
        ]
}
```

2. Modify the `tls` parameter so that its value is `disabled` .

   You can't modify the `ApplyMethod` because the `ApplyType` is `static`.

   For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   For Windows:

```
aws docdb modify-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   Output from this operation looks something like the following (JSON format).

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

3. Wait at least 5 minutes.
4. List the parameter values of `sample-parameter-group` to verify that the `tls` parameter was modified.

   For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
```

   For Windows:

```
aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
```

Output from this operation looks something like the following (JSON format).

```
{
       "Parameters": [
           {
               "ParameterValue": "false",
               "ParameterName": "enable_audit_logs",
               "ApplyType": "dynamic",
               "DataType": "string",
               "Description": "Enables auditing on cluster.",
               "AllowedValues": "true,false",
               "Source": "system",
               "IsModifiable": true,
               "ApplyMethod": "pending-reboot"
           },
           {
               "ParameterValue": "disabled",
               "ParameterName": "tls",
               "ApplyType": "static",
               "DataType": "string",
               "Description": "Config to enable/disable TLS",
               "AllowedValues": "disabled,enabled",
               "Source": "system",
               "IsModifiable": true,
               "ApplyMethod": "pending-reboot"
           }
       ]
}
```

Modifying Amazon DocumentDB clusters to use custom cluster parameter groups

When you create an Amazon DocumentDB cluster, a `default.docdb4.0` parameter group is automatically created for that cluster. You can't modify the `default` cluster parameter group. Instead, you can modify your Amazon DocumentDB cluster to associate a new customized parameter group with it.

This section explains how to modify an existing Amazon DocumentDB cluster to use a custom cluster parameter group using the AWS Management Console and the AWS Command Line Interface (AWS CLI).

Using the AWS Management Console

To modify an Amazon DocumentDB cluster to use a new, non-default cluster parameter group

1. Before you begin, make sure you have created an Amazon DocumentDB cluster and a cluster parameter group. See Creating an Amazon DocumentDB cluster and Creating Amazon DocumentDB cluster parameter groups for further instructions.
2. After creating your cluster parameter group, open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb. In the navigation pane, choose **Clusters** to add your new parameter group to a cluster.
3. Choose the cluster that you want to associate your parameter group with. Choose **Actions**, and then choose **Modify** to modify your cluster.
4. Under **Cluster options**, choose the new parameter group that you want to associate your cluster with.
5. Choose **Continue** to view a summary of your modifications.
6. After verifying your changes, you can apply them immediately or during the next maintenance window under **Scheduling of modifications**.
7. Choose **Modify cluster** to update your cluster with your new parameter group.

Using the AWS CLI

Before you begin, make sure that you have created an Amazon DocumentDB cluster and a cluster parameter group. You can create an Amazon DocumentDB cluster using the AWS CLI `create-db-cluster` operation. You can create a cluster parameter group using the AWS CLI `create-db-cluster-parameter-group` operation.

To add your new cluster parameter group to your cluster, use the AWS CLI `modify-db-cluster` operation with the following parameters.

- **--db-cluster-identifier** — The name of your cluster (for example, `sample-cluster`).
- **--db-cluster-parameter-group-name** — The name of the parameter group that you want to associate your cluster with (for example, `sample-parameter-group`).

Example

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --db-cluster-parameter-group-name sample-parameter-group
```

Output from this operation looks something like the following (JSON format).

```
"DBCluster": {
    "AvailabilityZones": [
            "us-west-2c",
            "us-west-2b",
            "us-west-2a"
    ],
    "BackupRetentionPeriod": 1,
    "DBClusterIdentifier": "sample-cluster",
    "DBClusterParameterGroup": "sample-parameter-group",
    "DBSubnetGroup": "default",
   ...
}
```

Copying Amazon DocumentDB cluster parameter groups

You can make a copy of a cluster parameter group in Amazon DocumentDB using the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Using the AWS Management Console

The following procedure guides you through making a new cluster parameter group by making a copy of an existing cluster parameter group.

To copy a cluster parameter group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**.
3. In the **Cluster parameter groups** pane, choose the name of the cluster parameter group that you want to copy.
4. Choose **Actions**, and then choose **Copy** to copy that parameter group.

5. Under **Copy options**, enter a name and description for the new cluster parameter group. Then choose **Copy** to save your changes.

Using the AWS CLI

To make a copy of a cluster parameter group, use the `copy-db-cluster-parameter-group` operation with the following parameters.

- **`--source-db-cluster-parameter-group-identifier`** — Required. The name or Amazon Resource Name (ARN) of the cluster parameter group that you want to make a copy of.

  If the source and target cluster parameter groups are in the same AWS Region, the identifier can be either a name or an ARN.

  If the source and target cluster parameter groups are in different AWS Regions, the identifier must be an ARN.
- **`--target-db-cluster-parameter-group-identifier`** — Required. The name or ARN of the cluster parameter group copy.

  Constraints:
  - Cannot be null, empty, or blank.
  - Must contain 1–255 letters, numbers, or hyphens.
  - First character must be a letter.
  - Cannot end with a hyphen or contain two consecutive hyphens.
- **`--target-db-cluster-parameter-group-description`** — Required. A user-supplied description for the cluster parameter group copy.

Example

The following code makes a copy of `sample-parameter-group`, naming the copy `sample-parameter-group-copy`.

For Linux, macOS, or Unix:

```
aws docdb copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier sample-parameter-group \
    --target-db-cluster-parameter-group-identifier sample-parameter-group-copy \
```

```
    --target-db-cluster-parameter-group-description "Copy of sample-parameter-group"
```

For Windows:

```
aws docdb copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier sample-parameter-group ^
    --target-db-cluster-parameter-group-identifier sample-parameter-group-copy ^
    --target-db-cluster-parameter-group-description "Copy of sample-parameter-group"
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:sample-parameter-group-copy",
        "DBClusterParameterGroupName": "sample-parameter-group-copy",
        "DBParameterGroupFamily": "docdb4.0",
        "Description": "Copy of sample-parameter-group"
    }
}
```

Resetting Amazon DocumentDB cluster parameter groups

You can reset some or all of an Amazon DocumentDB cluster parameter group's parameter values to their default values by using the AWS Management Console or the AWS Command Line Interface (AWS CLI) to reset the cluster parameter group.

Using the AWS Management Console

Follow these steps to reset some or all of a cluster parameter group's parameter values to their default values.

To reset a cluster parameter group's parameter values

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane on the left side of the console, choose **Parameter groups**.
3. In the **Cluster parameter groups** pane, choose the name of the cluster parameter group that you want to reset.

4. Choose **Actions**, and then choose **Reset** to reset that parameter group.
5. On the resulting **Cluster parameter group reset confirmation** page, confirm that you want to reset all cluster parameters for that parameter group to their defaults. Then choose **Reset** to reset your parameter group. You can also choose **Cancel** to discard your changes.

Using the AWS CLI

To reset some or all of a cluster parameter group's parameter values to their default values, use the `reset-db-cluster-parameter-group` operation with the following parameters.

- **`--db-cluster-parameter-group-name`** — Required. The name of the cluster parameter group to reset.
- **`--parameters`** — Optional. A list of `ParameterName` and `ApplyMethod` in the cluster parameter group to reset to their default values. Static parameters must be set to `pending-reboot` to take effect on the next instance restart or `reboot-db-instance` request. You must call `reboot-db-instance` for every instance in your cluster that you want the updated static parameter to apply to.

  This parameter and `--reset-all-parameters` are mutually exclusive: you can use either one but not both.
- **`--reset-all-parameters`** or **`--no-reset-all-parameters`** — Optional. Specifies whether to reset all parameters (`--reset-all-parameters` or only some of the parameters (`--no-reset-all-parameters`) to their default values. The `--reset-all-parameters` parameter and `--parameters` are mutually exclusive: you can use either one but not both.

  When you reset the entire group, dynamic parameters are updated immediately. Static parameters are set to `pending-reboot` to take effect on the next instance restart or `reboot-db-instance` request. You must call `reboot-db-instance` for every instance in your cluster that you want the updated static parameter applied to.

Example

Example 1: Resetting all parameters to their default values

The following code resets all parameters in the cluster parameter group `sample-parameter-group` their default values.

For Linux, macOS, or Unix:

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --reset-all-parameters
```

For Windows:

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --reset-all-parameters
```

Example 2: Resetting specified parameters to their default values

The following code resets the `tls` parameter in the cluster parameter group `sample-parameter-group` to its default value.

For Linux, macOS, or Unix:

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --no-reset-all-parameters \
       --parameters ParameterName=tls,ApplyMethod=pending-reboot
```

For Windows:

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --no-reset-all-parameters ^
       --parameters ParameterName=tls,ApplyMethod=pending-reboot
```

Output from this operation looks something like the following (JSON format).

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

Rebooting a cluster instance

Before a static parameter's value is changed, the cluster instance must be rebooted. Reboot each instance in your cluster that you want the updated static parameter to apply to.

For Linux, macOS, or Unix:

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-cluster-instance
```

For Windows:

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-cluster-instance
```

Deleting Amazon DocumentDB cluster parameter groups

You can delete a custom Amazon DocumentDB cluster parameter group using the AWS Management Console or the AWS Command Line Interface (AWS CLI). You can't delete the `default.docdb4.0` cluster parameter group.

Using the AWS Management Console

To delete a cluster parameter group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. In the **Parameter groups** pane, choose the radio button to the left of the cluster parameter group that you want to delete.
4. Choose **Actions**, and then choose **Delete**.
5. In the **Delete** confirmation pane, choose **Delete** to delete the cluster parameter group. To keep the cluster parameter group, choose **Cancel**.

Using the AWS CLI

To delete a cluster parameter group, use the `delete-db-cluster-parameter-group` operation with the following parameter.

- **`--db-cluster-parameter-group-name`** — Required. The name of the cluster parameter group to delete. This must be an existing cluster parameter group. *You cannot delete the `default.docdb4.0` cluster parameter group.*

Example - Deleting a cluster parameter group

The following example walks you through the three steps for deleting a cluster parameter group:

1. Finding the name of the cluster parameter group that you want to delete.
2. Deleting the specified cluster parameter group.
3. Verifying that the cluster parameter group was deleted.

1. Find the name of the cluster parameter group that you want to delete.

The following code lists the names of all cluster parameter groups.

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

For Windows:

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

The output of the preceding operation is a list the names of cluster parameter groups similar to the following (JSON format).

```
[
       [
           "default.docdb4.0"
       ],
```

```
        [
            "sample-parameter-group"
        ],
        [
            "sample-parameter-group-copy"
        ]
    ]
```

2. Delete a specific cluster parameter group.

The following code deletes the cluster parameter group `sample-parameter-group-copy`.

For Linux, macOS, or Unix:

```
aws docdb delete-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

For Windows:

```
aws docdb delete-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

There is no output from this operation.

3. Verify that the specified cluster parameter group was deleted.

The following code lists the names of all remaining cluster parameter groups.

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

For Windows:

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

The output of the preceding operation is a list of cluster parameter groups similar to the following (JSON format). The cluster parameter group that you just deleted should not be in the list.

Output from this operation looks something like the following (JSON format).

```
[
        [
            "default.docdb4.0"
        ],
        [
            "sample-parameter-group"
        ]
    ]
```

Amazon DocumentDB cluster parameters reference

When you change a dynamic parameter and save the cluster parameter group, the change is applied immediately regardless of the *Apply immediately* setting. When you change a static parameter and save the cluster parameter group, the parameter change takes effect after you manually reboot the instance. You can reboot an instance using the Amazon DocumentDB console or by explicitly calling `reboot-db-instance`.

The following table shows the parameters that apply to all instances in an Amazon DocumentDB cluster.

Amazon DocumentDB cluster-level parameters

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
`audit_log s`	disabled	enabled, disabled, ddl, dml_read, dml_write, all, none	Yes	Dynamic	String	Defines whether Amazon CloudWatc h audit logs are enabled. • **`enabled`** — CloudWatc h audit

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
						logs are enabled. • **`disabled`** — CloudWatch audit logs are disabled. • **`ddl`** — auditing for DDL events is enabled. • **`dml_read`** — auditing for DML read events is enabled. • **`dml_write`** — auditing for DML write events is enabled. • **`all`** — auditing for all database events is enabled.

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
						• **none** — auditing is disabled.
`change_stream_log_retention_duration`	10800	3600-604800	Yes	Dynamic	Integer	Defines the duration of time (in seconds) that the change stream log is retained and can be consumed.

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
`default_collection_compression`	disabled	enabled, disabled	Yes	Dynamic	String	Defines the default compressi on setting for new collections in a cluster • **`enabled`** — compressi on is enabled by default. • **`disabled`** — compressi on is disabled by default.

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
`profiler`	disabled	enabled, disabled	Yes	Dynamic	String	Enables profiling for slow operations. • **`enabled`** — operation s that take longer than a customer-defined threshold value (e.g., 100ms) are logged toAmazon CloudWatc h Logs. • **`disabled`** — slow operation s are not logged to CloudWatc h Logs.

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
`profiler_sampling_rate`	1.0	0.0-1.0	Yes	Dynamic	Float	Defines the sampling rate for logged operations.
`profiler_threshold_ms`	100	50-2147483646	Yes	Dynamic	Integer	Defines the threshold for `profiler`. • All operations greater than `profiler_threshold_ms` are logged to CloudWatch Logs.

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
`tls`	enabled	enabled, disabled, fips-140-3, tls1.2+, tls1.3+	Yes	Static	String	Defines whether Transport Layer Security (TLS) connectio ns are required. • **`enabled`** — TLS connectio ns are required to connect. • **`disabled`** — TLS connectio ns cannot be used to connect. • **`fips-140-3`** — TLS connectio ns with Federal Informati on

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
						Processin g Standards (FIPS) attribute s are required to connect. The cluster only accepts secure connectio ns per FIPS Publicati on 140-3. This is only supported starting with Amazon DocumentD B 5.0 (engine version 3.0.3727) clusters in these regions:

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
						ca-centra l-1, us-west-2 , us-east-1 , us-east-2, us-gov-east-1, us-gov-west-1. • **`tls1.2+`** — TLS connectio ns using TLS version 1.2 and above are required to connect. This is only supported starting with Amazon DocumentD B 4.0 (engine version

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
						2.0.10980) and Amazon DocumentD B 5.0 (engine version 3.0.11051). • **`tls1.3+`** — TLS connectio ns using TLS version 1.3 and above are required to connect. This is only supported starting with Amazon DocumentD B 4.0 (engine version 2.0.10980) and Amazon

Parameter	Default Valu	Valid Values	Modifiable	Apply Type	Data Type	Descripti on
						DocumentD B 5.0 (engine version 3.0.11051).
`ttl_monit or`	enabled	enabled, disabled	Yes	Dynamic	String	Defines whether Time to Live (TTL) monitoring is enabled for the cluster. • **`enabled`** — TTL monitorin g is enabled. • **`disabled`** — TTL monitorin g is disabled.

Modifying Amazon DocumentDB cluster parameters

In Amazon DocumentDB, *cluster parameter groups* consist of *parameters* that apply to all of the instances that you create in the cluster. For custom cluster parameter groups, you can modify a parameter value at any time or reset all the parameter values to their defaults for parameter groups that you create. This section describes how to view the parameters that make up an

Amazon DocumentDB cluster parameter group and their values, and how you can change or update these values.

Parameters can be *dynamic* or *static*. When you change a dynamic parameter and save the cluster parameter group, the change is applied immediately regardless of the `Apply Immediately` setting. When you change a static parameter and save the cluster parameter group, the parameter change takes effect only after you manually reboot the instances.

Viewing an Amazon DocumentDB cluster parameter group's parameters

You can see an Amazon DocumentDB cluster's parameters and their values using the AWS Management Console or AWS CLI.

Using the AWS Management Console

To view the details of a cluster parameter group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the **Parameter groups** pane, choose the name of the cluster parameter group that you want to see the details of.
4. The resulting page shows the following values for each parameter: the parameter's name, current value, allowed values, whether the parameter is modifiable, apply type, data type, and description.

	Cluster parameter name	Values	Allowed values
○	audit_logs	disabled	enabled,disabled
○	tls	enabled	disabled,enabled
○	ttl_monitor	enabled	disabled,enabled

Using the AWS CLI

To see a cluster's parameter group's parameters and their values, use the `describe-db-cluster-parameters` operation with the following parameters.

- **`--db-cluster-parameter-group-name`** — Required. The name of the cluster parameter group for which you want a detailed parameter list.
- **`--source`** — Optional. If supplied, returns only parameters for a specific source. Parameter sources can be `engine-default`, `system`, or `user`.

Example

The following code lists the parameters and their values for the `custom3-6-param-grp` parameter group. For more information about the parameter group, omit the `--query` line. For information about all parameter groups, omit the `--db-cluster-parameter-group-name` line.

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameters \
   --db-cluster-parameter-group-name custom3-6-param-grp \
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

For Windows:

```
aws docdb describe-db-cluster-parameters ^
   --db-cluster-parameter-group-name custom3-6-param-grp ^
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

Output from this operation looks something like the following (JSON format).

```
[
    [
        "audit_logs",
        "disabled"
    ],
    [
        "tls",
        "enabled"
    ],
    [
```

```
        "ttl_monitor",
        "enabled"
    ]
]
```

Modifying an Amazon DocumentDB cluster parameter group's parameters

You can modify a parameter group's parameters using the AWS Management Console or AWS CLI.

Using the AWS Management Console

To update the parameters of a cluster parameter group

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon (≡) in the upper-left corner of the page.

3. In the **Parameter groups** pane, choose the cluster parameter group that you want to update the parameters of.
4. The resulting page shows the parameters and their corresponding details for this cluster parameter group. Select a parameter to update.
5. On the top right of the page, choose **Edit** to change the value of the parameter. For more information about the types of cluster parameters, see Amazon DocumentDB cluster parameters reference.
6. Make your change, and then choose **Modify cluster parameter** to save the changes. To discard your changes, choose **Cancel**.

Using the AWS CLI

To modify a cluster parameter group's parameters, use the `modify-db-cluster-parameter-group` operation with the following parameters:

- **`--db-cluster-parameter-group-name`** — Required. The name of the cluster parameter group that you are modifying.
- **`--parameters`** — Required. The parameter or parameters that you are modifying. Each parameter entry must include the following:
  - **`ParameterName`** — The name of the parameter that you are modifying.
  - **`ParameterValue`** — The new value for this parameter.
  - **`ApplyMethod`** — How you want changes to this parameter applied. Permitted values are `immediate` and `pending-reboot`.

> **Note**
>
> Parameters with the `ApplyType` of `static` must have an `ApplyMethod` of `pending-reboot`.

To change the values of a cluster parameter group's parameters (AWS CLI)

The following example changes the `tls` parameter.

1. **List the parameters and their values of `sample-parameter-group`**

   For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

   For Windows:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

   Output from this operation looks something like the following (JSON format).

```
{
    "Parameters": [
        {
            "Source": "system",
            "ApplyType": "static",
```

```
            "AllowedValues": "disabled,enabled",
            "ParameterValue": "enabled",
            "ApplyMethod": "pending-reboot",
            "DataType": "string",
            "ParameterName": "tls",
            "IsModifiable": true,
            "Description": "Config to enable/disable TLS"
        },
        {
            "Source": "user",
            "ApplyType": "dynamic",
            "AllowedValues": "disabled,enabled",
            "ParameterValue": "enabled",
            "ApplyMethod": "pending-reboot",
            "DataType": "string",
            "ParameterName": "ttl_monitor",
            "IsModifiable": true,
            "Description": "Enables TTL Monitoring"
        }
    ]
}
```

2. **Modify the `tls` parameter so that its value is `disabled`**. You can't modify the `ApplyMethod` because the `ApplyType` is `static`.

   For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 "ParameterName"=tls,ParameterValue=disabled,ApplyMethod=pending-reboot"
```

   For Windows:

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name sample-parameter-group ^
    --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

   Output from this operation looks something like the following (JSON format).

```
{
```

```
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

3. **Wait at least 5 minutes.**
4. **List the parameter values of `sample-parameter-group`.**

   For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

   For Windows:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

   Output from this operation looks something like the following (JSON format).

```
{
    "Parameters": [
        {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        },
        {
            "ParameterName": "tls",
            "ParameterValue": "disabled",
            "Description": "Config to enable/disable TLS",
            "Source": "user",
            "ApplyType": "static",
            "DataType": "string",
            "AllowedValues": "disabled,enabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        }
```

```
    ]
}
```

Understanding Amazon DocumentDB endpoints

You can use Amazon DocumentDB (with MongoDB compatibility) endpoints to connect to a cluster or instance. Amazon DocumentDB has three different types of endpoints, each with its own purpose.

Topics

- Finding a cluster's endpoints
- Finding an instance's endpoint
- Connecting to endpoints

Cluster endpoint

A cluster endpoint is an endpoint for an Amazon DocumentDB cluster that connects to the current primary instance for the cluster. Each Amazon DocumentDB cluster has a single cluster endpoint and one primary instance. In case of a failover, the cluster endpoint is remapped to the new primary instance.

Reader endpoint

A reader endpoint is an endpoint for an Amazon DocumentDB cluster that connects to one of the available replicas for that cluster. Each Amazon DocumentDB cluster has a reader endpoint. If there is more than one replica, the reader endpoint directs each connection request to one of the Amazon DocumentDB replicas.

Instance endpoint

An instance endpoint is an endpoint that connects to a specific instance. Each instance in a cluster, regardless of whether it is a primary or replica instance, has its own unique instance endpoint. It is best to not use instance endpoints in your application. This is because they can change roles in case of a failover, thus requiring code changes in your application.

Finding a cluster's endpoints

You can find a cluster's cluster endpoint and reader endpoint using the Amazon DocumentDB console or AWS CLI.

Using the AWS Management Console

To find a cluster's endpoints using the console:

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. From the list of clusters, choose the name of the cluster you are interested in.
4. On the cluster details page, select the **Configuration** tab. In the **Configurations and status** section, you will find the **Cluster endpoint** and **Reader endpoint**.

   Configurations and status

   ARN
   arn:aws:rds:us-east-2:816069136184:cluster:docdb-2025-01-31-15-22-38

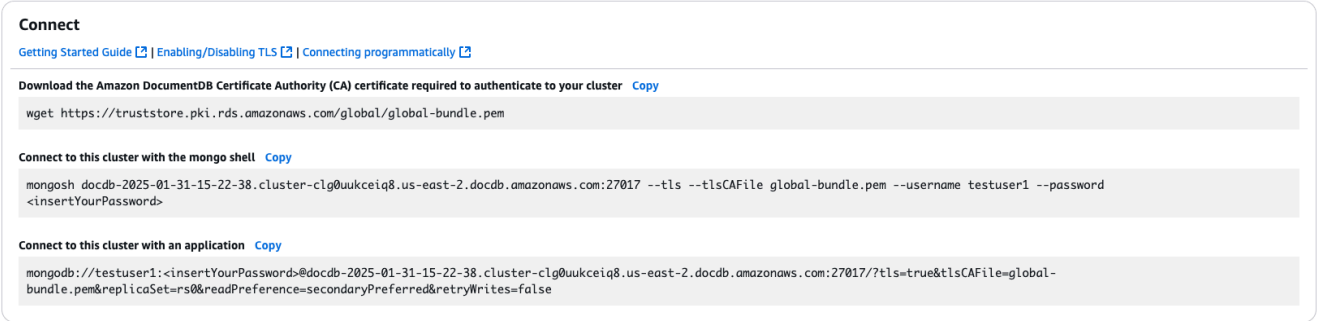
   Cluster identifier
   docdb-2025-01-31-15-22-38 (available)

   Cluster creation time
   1/31/2025, 10:23:09 AM UTC-5

   Cluster endpoint
   docdb-2025-01-31-15-22-38.cluster-clg0uukceiq8.us-east-2.docdb.amazonaws.com

   Reader endpoint
   docdb-2025-01-31-15-22-38.cluster-ro-clg0uukceiq8.us-east-2.docdb.amazonaws.com

5. To connect to this cluster, select the **Connectivity & security** tab. Locate the connection string for the `mongo` shell and the connection string that can be used in the application code to connect to your cluster.



Using the AWS CLI

To find the cluster and reader endpoints for your cluster using the AWS CLI, run the `describe-db-clusters` command with these parameters.

Parameters

- **`--db-cluster-identifier`**—Optional. Specifies the cluster to return endpoints for. If omitted, returns endpoints for up to 100 of your clusters.
- **`--query`**—Optional. Specifies the fields to display. Helpful by reducing the amount of data that you need to view to find the endpoints. If omitted, all information about a cluster is returned.
- **`--region`**—Optional. Use the `--region` parameter to specify the Region that you want to apply the command to. If omitted, your default Region is used.

Example

The following example returns the `DBClusterIdentifier`, endpoint (cluster endpoint), and `ReaderEndpoint` for `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
   --region us-east-1 \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

For Windows:

```
aws docdb describe-db-clusters ^
```

```
    --region us-east-1 ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

Output from this operation looks something like the following (JSON format).

```
[
  [
     "sample-cluster",
     27017,
     "sample-cluster.cluster-corlsfccjozr.us-east-1.docdb.amazonaws.com",
     "sample-cluster.cluster-ro-corlsfccjozr.us-east-1.docdb.amazonaws.com"
  ]
]
```

Now that you have the cluster endpoint, you can connect to the cluster using either `mongo` or `mongodb`. For more information, see Connecting to endpoints.

Finding an instance's endpoint

You can find the endpoint for an instance using the Amazon DocumentDB console or the AWS CLI.

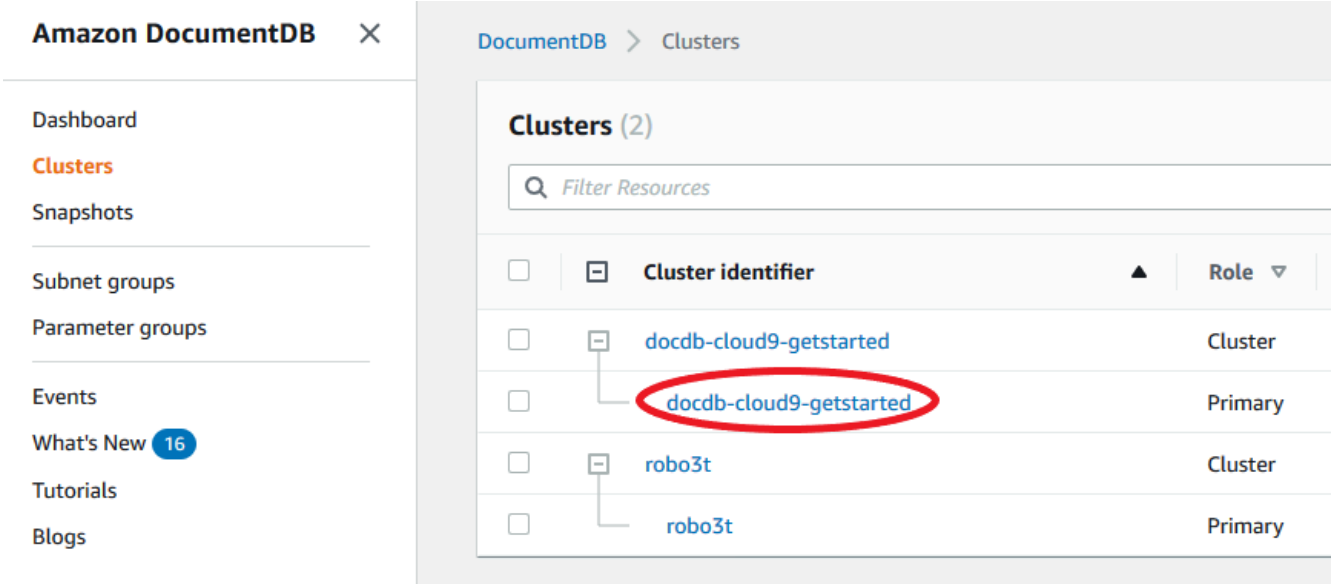
Using the AWS Management Console

To find an instance's endpoint using the console

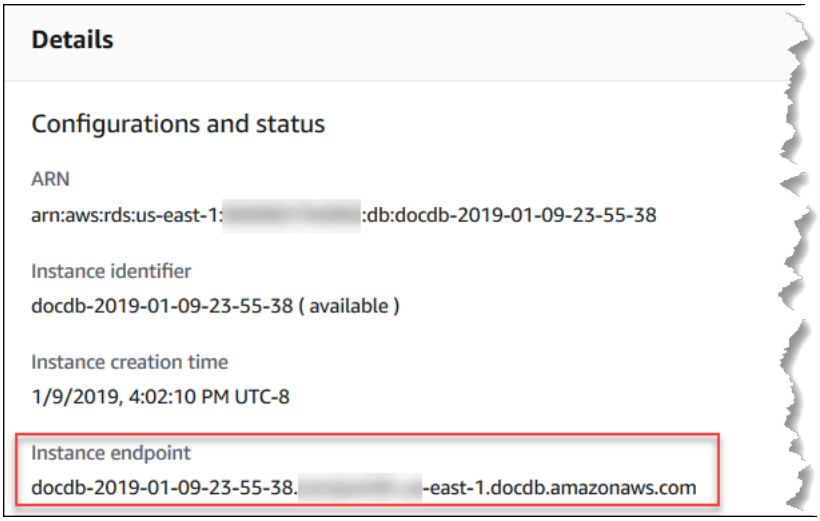
1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. Check the box to the left of the instance you are interested in.
5. Scroll down to the **Details** section then locate the instance endpoint.



6. To connect to this instance, scroll up to the **Connect** section. Locate the connection string for the `mongo` shell and a connection string that can be used in your application code to connect to your instance.

Using the AWS CLI

To find the instance endpoint using the AWS CLI, run the following command with these arguments.

Arguments

- **`--db-instance-identifier`**—Optional. Specifies the instance to return the endpoint for. If omitted, returns the endpoint for up to 100 of your instances.
- **`--query`**—Optional. Specifies the fields to display. Helpful by reducing the amount of data that you need to view to find the endpoints. If omitted, all information on an instance is returned. The `Endpoint` field has three members, so listing it in the query as in the following example returns all three members. If you're only interested in some of the `Endpoint` members, replace `Endpoint` in the query with the members you're interested in, as in the second example.
- **`--region`**—Optional. Use the `--region` parameter to specify the Region that you want to apply the command to. If omitted, your default Region is used.

Example

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

For Windows:

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

Output from this operation looks something like the following (JSON format).

```
[
    [
```

```
        "sample-cluster-instance",
        {
            "Port": 27017,
            "Address": "sample-cluster-instance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "HostedZoneId": "Z2R2ITUGPM61AM"
        }
    ]
]
```

Reducing the output to eliminate the endpoint's `HostedZoneId`, you can modify your query by specifying `Endpoint.Port` and `Endpoint.Address`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

For Windows:

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

Output from this operation looks something like the following (JSON format).

```
[
    [
        "sample-cluster-instance",
        27017,
        "sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com"
    ]
]
```

Now that you have the instance endpoint, you can connect to the instance using either `mongo` or `mongodb`. For more information, see Connecting to endpoints.

Connecting to endpoints

When you have your endpoint, either cluster or instance, you can connect to it using the mongo shell or a connection string.

Connecting using the mongo shell

Use the following structure to construct the string that you need to connect to your cluster or instance using the mongo shell:

```
mongo \
    --ssl \
    --host Endpoint:Port \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

mongo shell examples

Connect to a cluster:

```
mongo \
    --ssl \
    --host sample-cluster.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

Connect to an instance:

```
mongo \
    --ssl \
    --host sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

Connecting using a connection string

Use the following structure to construct the connection string that you need to connect to your cluster or instance.

```
mongodb://UserName:Password@endpoint:port?replicaSet=rs0&ssl_ca_certs=global-
bundle.pem
```

Connection string examples

Connect to a cluster:

```
mongodb://UserName:Password@sample-cluster.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

Connect to an instance:

```
mongodb://UserName:Password@sample-cluster-instance.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

Understanding Amazon DocumentDB Amazon Resource Names (ARNs)

Resources that you create in AWS are each uniquely identified with an Amazon Resource Name (ARN). For certain Amazon DocumentDB (with MongoDB compatibility) operations, you must uniquely identify an Amazon DocumentDB resource by specifying its ARN. For example, when you add a tag to a resource, you must provide the resource's ARN.

Topics

- Constructing an ARN for an Amazon DocumentDB resource
- Finding an Amazon DocumentDB resource ARN

Constructing an ARN for an Amazon DocumentDB resource

You can construct an ARN for an Amazon DocumentDB resource using the following syntax. Amazon DocumentDB shares the format of Amazon Relational Database Service (Amazon RDS) ARNs. Amazon DocumentDB ARNs contain `rds` and not docdb.

```
arn:aws:rds:region:account_number:resource_type:resource_id
```

Region Name	Region	Availability Zones (compute)
US East (Ohio)	`us-east-2`	3
US East (N. Virginia)	`us-east-1`	6
US West (Oregon)	`us-west-2`	4
Africa (Cape Town)	`af-south-1`	3
South America (São Paulo)	`sa-east-1`	3
Asia Pacific (Hong Kong)	`ap-east-1`	3
Asia Pacific (Hyderabad)	`ap-south-2`	3
Asia Pacific (Mumbai)	`ap-south-1`	3
Asia Pacific (Seoul)	`ap-northeast-2`	4
Asia Pacific (Singapore)	`ap-southeast-1`	3
Asia Pacific (Sydney)	`ap-southeast-2`	3
Asia Pacific (Tokyo)	`ap-northeast-1`	3
Canada (Central)	`ca-central-1`	3
China (Beijing) Region	`cn-north-1`	3
China (Ningxia)	`cn-northwest-1`	3
Europe (Frankfurt)	`eu-central-1`	3
Europe (Ireland)	`eu-west-1`	3
Europe (London)	`eu-west-2`	3

Region Name	Region	Availability Zones (compute)
Europe (Milan)	`eu-south-1`	3
Europe (Paris)	`eu-west-3`	3
Europe (Spain)	`eu-south-2`	3
Europe (Stockholm)	`eu-north-1`	3
Middle East (UAE)	`me-central-1`	3
Israel (Tel Aviv)	`il-central-1`	3
AWS GovCloud (US-West)	`us-gov-west-1`	3
AWS GovCloud (US-East)	`us-gov-east-1`	3

> **Note**
>
> The Amazon DocumentDB architecture separates storage and compute. For the storage layer, Amazon DocumentDB replicates six copies of your data across three AWS Availability Zones (AZs). The AZs listed in the table above are the number of AZs that you can use in a given region to provision compute instances. As an example, if you are launching an Amazon DocumentDB cluster in ap-northeast-1, your storage will be replicated six ways across three AZs but your compute instances will only be available in two AZs.

The following table shows the format that you should use when constructing an ARN for a particular Amazon DocumentDB resource. Amazon DocumentDB shares the format of Amazon RDS ARNs. Amazon DocumentDB ARNs contain `rds` and not `docdb`.

Resource Type	ARN Format / Example
Instance (db)	`arn:aws:rds:` *`region`*`:`*`account_number`* `:db:`*`resource_id`*

Resource Type	ARN Format / Example
	`arn:aws:rds:us-east-1: 1234567890 :db:sample-db-instance`

Cluster (`cluster`)	`arn:aws:rds: region:account_number :cluster: resource_id`
	
	`arn:aws:rds:us-east-1: 1234567890 :cluster: sample-db-cluster`

Cluster parameter group (`cluster-pg`)	`arn:aws:rds: region:account_number :cluster-pg: resource_id`
	
	`arn:aws:rds:us-east-1: 1234567890 :cluster-pg: sample-db-cluster-parameter-group`

Security group (`secgrp`)	`arn:aws:rds: region:account_number :secgrp:resource_id`
	
	`arn:aws:rds:us-east-1: 1234567890 :secgrp:sample-public-secgrp`

Cluster snapshot (`cluster-snapshot`)	`arn:aws:rds: region:account_number :cluster-snapshot: resource_id`
	
	`arn:aws:rds:us-east-1: 1234567890 :cluster-snapshot: sample-db-cluster-snapshot`

Subnet group (`subgrp`)	`arn:aws:rds: region:account_number :subgrp:resource_id`
	
	`arn:aws:rds:us-east-1: 1234567890 :subgrp:sample-subnet-10`

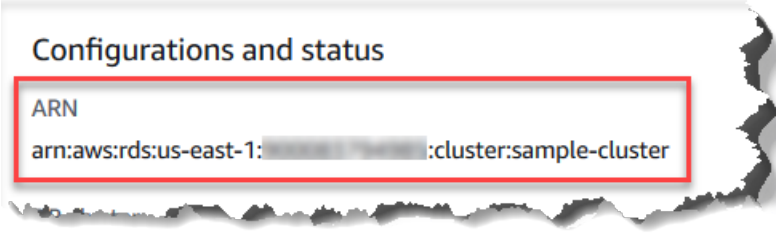
Finding an Amazon DocumentDB resource ARN

You can find the ARN of an Amazon DocumentDB resource using the AWS Management Console or the AWS CLI.

Using the AWS Management Console

To find an ARN using the console, navigate to the resource that you want an ARN for, and view the details for that resource.

For example, you can get the ARN for a cluster by selecting the **Configuration** tab on the cluster details page. The ARN can be found in the **Configurations and status** section, as shown in the following screenshot.



Using the AWS CLI

To get an ARN using the AWS CLI for a particular Amazon DocumentDB resource, use the `describe` operation for that resource. The following table shows each AWS CLI operation and the ARN property that is used with the operation to get an ARN.

AWS CLI Command	ARN Property
`describe-db-instances`	`DBInstanceArn`
`describe-db-clusters`	`DBClusterArn`
`describe-db-parameter-groups`	`DBParameterGroupArn`
`describe-db-cluster-parameter-groups`	`DBClusterParameterGroupArn`
`describe-db-security-groups`	`DBSecurityGroupArn`
`describe-db-snapshots`	`DBSnapshotArn`

AWS CLI Command	ARN Property
`describe-db-cluster-snapshots`	`DBClusterSnapshotArn`
`describe-db-subnet-groups`	`DBSubnetGroupArn`

Example - Finding the ARN for a cluster

The following AWS CLI operation finds the ARN for the cluster `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

For Windows:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

Output from this operation looks something like the following (JSON format).

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster"
]
```

Example - Finding ARNs for multiple parameter groups

For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameter-groups \
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

For Windows:

```
aws docdb describe-db-cluster-parameter-groups ^
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

Output from this operation looks something like the following (JSON format).

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:custom3-6-param-grp",
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:default.aurora5.6",
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:default.docdb3.6"
]
```

Tagging Amazon DocumentDB resources

You can use Amazon DocumentDB (with MongoDB compatibility) tags to add metadata to your Amazon DocumentDB resources. These tags can be used with AWS Identity and Access Management (IAM) policies to manage access to Amazon DocumentDB resources and to control what actions can be applied to the resources. You can also use tags to track costs by grouping expenses for similarly tagged resources.

You can tag the following Amazon DocumentDB resources:

- Clusters
- Instances
- Snapshots
- Cluster snapshots
- Parameter groups
- Cluster parameter groups
- Security groups
- Subnet groups

Overview of Amazon DocumentDB resource tags

An Amazon DocumentDB tag is a name-value pair that you define and associate with an Amazon DocumentDB resource. The name is referred to as the *key*. Supplying a value for the key is optional. You can use tags to assign arbitrary information to an Amazon DocumentDB resource. You can use a tag key, for example, to define a category, and the tag value might be an item in that category. For example, you might define a tag key of `project` and a tag value of `Salix`, indicating that the Amazon DocumentDB resource is assigned to the Salix project. You can also use tags to designate Amazon DocumentDB resources as being used for test or production by using a key such as

`environment=test` or `environment=production`. We recommend that you use a consistent set of tag keys to make it easier to track metadata that is associated with Amazon DocumentDB resources.

You can use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Each Amazon DocumentDB resource has a tag set, which contains all the tags that are assigned to that resource. A tag set can contain as many as 10 tags, or it can be empty. If you add a tag to an Amazon DocumentDB resource that has the same key as an existing tag on resource, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags; tags are interpreted strictly as character strings. Amazon DocumentDB can set tags on an instance or other Amazon DocumentDB resources, depending on the settings that you use when you create the resource. For example, Amazon DocumentDB might add a tag indicating that an instance is for production or for testing.

You can add a tag to a snapshot, but your bill will not reflect this grouping.

You can use the AWS Management Console or the AWS CLI to add, list, and delete tags on Amazon DocumentDB resources. When using the AWS CLI, you must provide the Amazon Resource Name (ARN) for the resource that you want to work with. For more information about Amazon DocumentDB ARNs, see Understanding Amazon DocumentDB Amazon Resource Names (ARNs).

Tag constraints

The following constraints apply to Amazon DocumentDB tags:

- Maximum number of tags per resource - 10
- Maximum **Key** length - 128 Unicode characters
- Maximum **Value** length - 256 Unicode characters
- Valid characters for **Key** and **Value** - uppercase and lowercase letters in the UTF-8 character set, digits, space, and the following characters: `_ . : / = + -` and @ (Java regex: `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"`)
- Tag keys and values are case sensitive.

- The prefix `aws:` cannot be used for tag keys or values; it is reserved for AWS.

Adding and updating tags on an Amazon DocumentDB resource

You can add up to 10 tags to a resource using the AWS Management Console or the AWS CLI.

Using the AWS Management Console

The process for adding a tag to a resource is similar regardless of what resource you're adding the tag to. In this example, you add a tag to a cluster.

To add or update tags to a cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. From the navigation pane, choose **clusters**.
3. Choose the name of the cluster that you want to add tags to.
4. Scroll down to the **Tags** section, and then choose **Edit**.
5. For each tag you that want to add to this resource, do the following:
   a. To add a new tag, enter in the name of the tag in the **Key** box. To change a tag's value, find the tag's name in the **Key** column.
   b. To give the tag a new or updated value, enter a value for the tag in the **Value** box.
   c. If you have more tags to add, choose **Add**. Otherwise, when finished, choose **Save**.

Using the AWS CLI

The process for adding a tag to a resource is similar regardless of what resource you're adding the tags to. In this example, you add three tags to a cluster. The second tag, `key2`, has no value.

Use the AWS CLI operation `add-tags-to-resource` with these parameters.

Parameters

- **`--resource-name`**—The ARN of the Amazon DocumentDB resource that you want to add tags to.
- **`--tags`**—A list the tags (key-value pair) that you want to add to this resource in the format `Key=`*`key-name`*`,Value=`*`tag-value`*.

Example

For Linux, macOS, or Unix:

```
aws docdb add-tags-to-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tags Key=key1,Value=value1 Key=key2 Key=key3,Value=value3
```

For Windows:

```
aws docdb add-tags-to-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tags Key=key1,Value=value1 Key=key2 Key=key3,Value=value3
```

The `add-tags-to-resource` operation produces no output. To see the results of the operation, use the `list-tags-for-resource` operation.

Listing tags on an Amazon DocumentDB resource

You can use the AWS Management Console or the AWS CLI to get a listing of the tags for an Amazon DocumentDB resource.

Using the AWS Management Console

The process for listing tags on a resource is similar regardless of what resource you're adding the tag to. In this example, you list the tags for a cluster.

To list the tags on a cluster using the console

1. Open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. From the navigation pane, choose **clusters**.
3. Choose the name of the cluster that you want to list tags for.
4. To see a listing of the tags on this resource, scroll down to the **Tags** section.

Using the AWS CLI

The process for listing the tags on a resource is similar regardless of what resource you're listing the tag for. In this example, you list the tags on a cluster.

Use the AWS CLI operation `list-tags-for-resource` with these parameters.

Parameters

- **`--resource-name`**—Required. The ARN of the Amazon DocumentDB resource that you want to list tags for.

Example

For Linux, macOS, or Unix:

```
aws docdb list-tags-for-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

For Windows:

```
aws docdb list-tags-for-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

Output from this operation looks something like the following (JSON format).

```
{
    "TagList": [
        {
            "Key": "key1",
            "Value": "value1"
        },
        {
            "Key": "key2",
            "Value": ""
        },
        {
            "Key": "key3",
            "Value": "value3"
        }
    ]
}
```

Removing tags from an Amazon DocumentDB resource

You can use the AWS Management Console or the AWS CLI to remove tags from Amazon DocumentDB resources.

Using the AWS Management Console

The process for removing tags from a resource is similar regardless of what resource you're adding the tag to. In this example, you remove tags from a cluster.

To remove tags from a cluster using the console

1. Open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. From the navigation pane, choose **clusters**.
3. Choose the name of the cluster that you want to remove tags from.
4. Scroll down to the **Tags** section, and then choose **Edit**.
5. If you want to remove all tags from this resource, choose **Remove all**. Otherwise, for each tag that you want to remove from this resource, do the following:
   a. Locate the name of the tag in the **Key** column.
   b. Choose **Remove** on the same row as the tag key.
   c. When finished, choose **Save**.

Using the AWS CLI

The process for removing a tag from a resource is similar regardless of what resource you're removing the tag from. In this example, you remove a tag from a cluster.

Use the AWS CLI operation `remove-tags-from-resource` with these parameters.

- **`--resource-name`**—Required. The ARN of the Amazon DocumentDB resource that you want to remove tags from.
- **`--tag-keys`**—Required. A list the tag keys that you want removed from this resource.

Example

For Linux, macOS, or Unix:

```
aws docdb remove-tags-from-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

For Windows:

```
aws docdb remove-tags-from-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

The `removed-tags-from-resource` operation produces no output. To see the results of the operation, use the `list-tags-for-resource` operation.

Maintaining Amazon DocumentDB

Periodically, Amazon DocumentDB performs maintenance on Amazon DocumentDB resources. Maintenance most often involves updates to the database engine (cluster maintenance) or the instance's underlying operating system (OS) (instance maintenance). Database engine updates are required patches and include security fixes, bug fixes, and enhancements to the database engine. While most operating system patches are optional, if you don't apply them for a while, the patch may be required and auto applied to maintain your security posture. So, we recommend that you apply operating system updates to your Amazon DocumentDB instances as soon as they are available.

Database engine patches require that you take your Amazon DocumentDB clusters offline for a short time. Once available, these patches are automatically scheduled to apply during an upcoming scheduled maintenance window of your Amazon DocumentDB cluster.

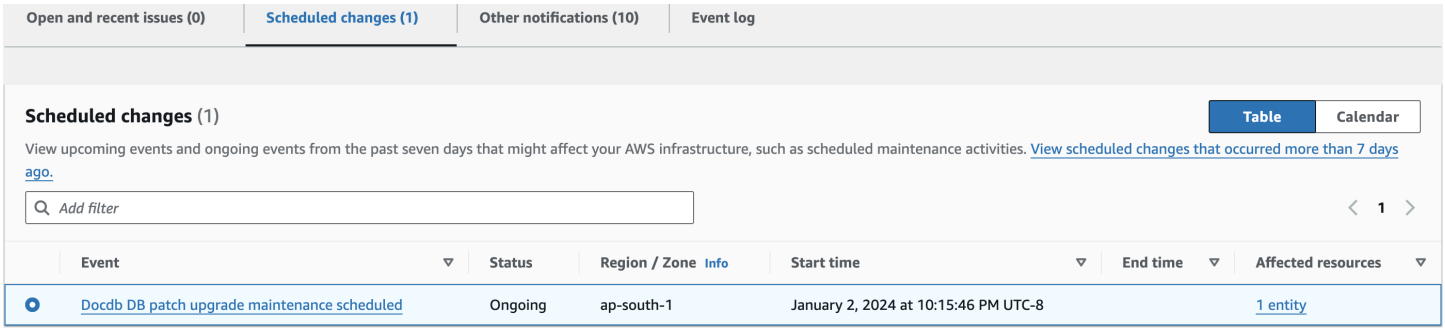
Both cluster and instances maintenance have their own respective maintenance windows. Cluster and instance modifications that you have chosen not to apply immediately, are also applied during the maintenance window. By default, when you create a cluster, Amazon DocumentDB assigns a maintenance window for both a cluster and each individual instance. You can choose the maintenance window when creating a cluster or an instance. You can also modify the maintenance windows at any time to fit your business schedules or practices. It is generally advised to choose maintenance windows that minimize the impact of the maintenance on your application (for example, on evenings or weekends).

Topics

- Notifications for Amazon DocumentDB engine patches
- Viewing pending Amazon DocumentDB maintenance actions
- Amazon DocumentDB engine updates
- User-initiated updates
- Managing your Amazon DocumentDB maintenance windows
- Amazon DocumentDB operating system updates

Notifications for Amazon DocumentDB engine patches

You will receive maintenance notifications for required database engine patches through health events in the AWS Health Dashboard (AHD) in the AWS console and through e-mails. When an Amazon DocumentDB engine maintenance patch becomes available in a particular AWS region, all impacted Amazon DocumentDB user accounts in the region will receive an AHD and email notification for each Amazon DocumentDB version affected by the patch. You can view these notifications under the **Scheduled changes** section of the AHD in the AWS console. The notification will have details about timing of patch availability, auto apply schedule, list of impacted clusters, and release notes. This notification will also be delivered via e-mail to the AWS account's root user email address.



Once you receive this notification, you can choose to self-apply these engine patches to your Amazon DocumentDB clusters before the scheduled auto-apply date. Or you can wait for the engine patches to get auto-applied during an upcoming maintenance window (default option).

> **Note**
>
> The **Status** for the notification in the AHD will be set to 'Ongoing' until a new Amazon DocumentDB engine patch with a new engine patch version is released.

Once the engine patch is applied to your Amazon DocumentDB cluster, the cluster's engine patch version will be updated to reflect the version in the notification. You can run the `db.runCommand({getEngineVersion: 1})` command to verify this update.

AWS Health also integrates with Amazon EventBridge which uses events to build scalable event-driven applications and integrates with over 20 targets, including AWS Lambda, Amazon Simple Queue Service (SQS), and others. You can use `AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_SCHEDULED` event code to setup Amazon EventBridge before engine patches become available. You can setup EventBridge to respond to the event and auto-perform actions such as capturing event information, initiating additional events, sending notifications via additional channels such as push notifications to the AWS Console Mobile Application, and taking corrective or other actions, when an Amazon DocumentDB engine patch becomes available in your region.

In the rare scenario of Amazon DocumentDB cancelling an engine patch, you will receive an AHD notification as well as an e-mail informing you about the cancellation. Accordingly, you can use the `AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_CANCELLED` event code to setup Amazon EventBridge to respond to this event. View the *Amazon EventBridge User Guide* to learn more about using Amazon EventBridge rules.

Viewing pending Amazon DocumentDB maintenance actions

You can view whether a maintenance update is available for your cluster by using the AWS Management Console or the AWS CLI.

If an update is available, you can do one of the following:

- Defer a maintenance action that is currently scheduled for next maintenance window (for OS patches only).
- Apply the maintenance actions immediately.
- Schedule the maintenance actions to start during your next maintenance window.

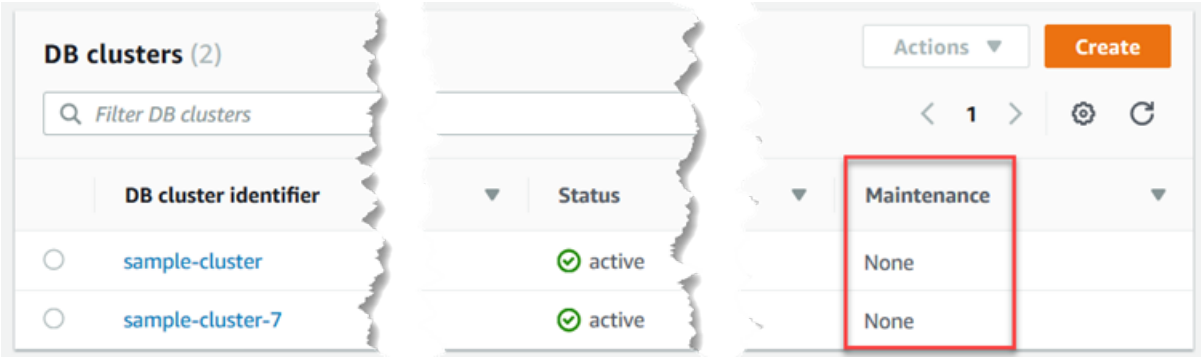
Note

If you take no action, required maintenance actions such as engine patches will be auto applied in an upcoming scheduled maintenance window.

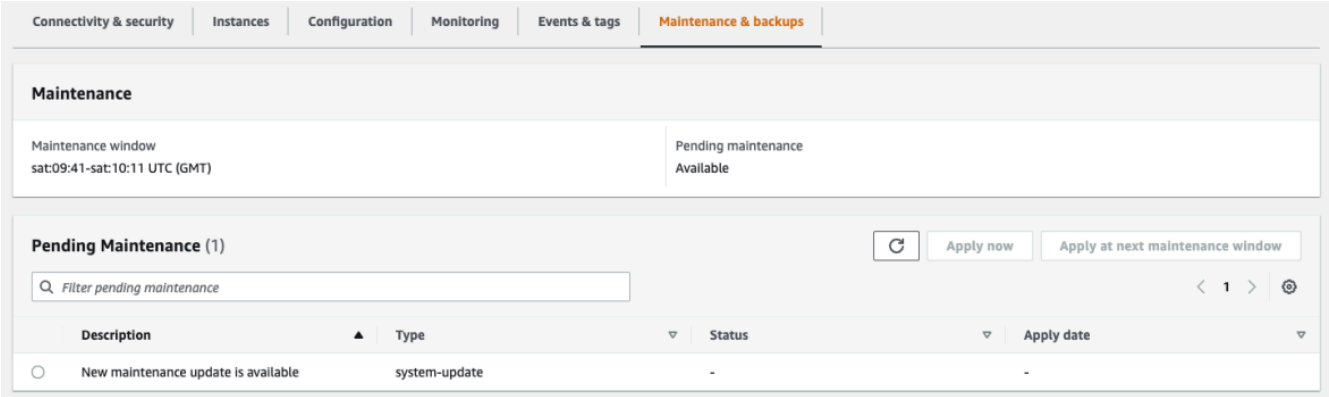
The maintenance window determines when pending operations start, but it does not limit the total execution time of these operations.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. If an update is available, it is indicated by the word **Available**, **Required**, or **Next Window** in the **Maintenance** column for the cluster on the Amazon DocumentDB console, as shown here:



4. To take an action, choose the cluster to show its details, then choose **Maintenance & backups**. The **Pending Maintenance** items appear.



Using the AWS CLI

Use the following AWS CLI operation to determine what maintenance actions are pending. The output here shows no pending maintenance actions.

```
aws docdb describe-pending-maintenance-actions
```

Output from this operation looks something like the following (JSON format).

```
{
    "PendingMaintenanceActions": []
}
```

Amazon DocumentDB engine updates

With Amazon DocumentDB, you can choose when to apply maintenance operations. You can decide when Amazon DocumentDB applies updates using the AWS Management Console or AWS CLI.

Use the procedures in this topic to immediately upgrade or schedule an upgrade for your cluster.

Using the AWS Management Console

You can use the console to manage updates for your Amazon DocumentDB clusters.

To manage an update for a cluster

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. In the list of clusters, choose the button next to the name of the cluster that you want to apply the maintenance operation to.
4. On the **Actions** menu, choose one of the following:
   - **Upgrade now** to immediately perform the pending maintenance tasks.
   - **Upgrade at next window** to perform the pending maintenance tasks during the cluster's next maintenance window.

   Alternatively, you can click **Apply now** or **Apply at next maintenance window** in the Pending Maintenance section of the cluster **Maintenance & backups** tab (see **Using the AWS Management Console** in the previous section).

> **Note**
>
> If there are no pending maintenance tasks, all of the preceding options are inactive.

Using the AWS CLI

To apply a pending update to a cluster, use the `apply-pending-maintenance-action` AWS CLI operation.

Parameters

- **`--resource-identifier`**—The Amazon DocumentDB Amazon Resource Name (ARN) of the resource that the pending maintenance action applies to.
- **`--apply-action`**—The pending maintenance action to apply to this resource.

  Valid values: `system-update` and `db-upgrade`.
- **`--opt-in-type`**—A value that specifies the type of opt-in request, or undoes an opt-in request. An opt-in request of type `immediate` can't be undone.

  Valid values:
  - `immediate`—Apply the maintenance action immediately.
  - `next-maintenance`—Apply the maintenance action during the next maintenance window for the resource.
  - `undo-opt-in`—Cancel any existing `next-maintenance` opt-in requests.

Example

For Linux, macOS, or Unix:

```
aws docdb apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:docdb \
    --apply-action system-update \
    --opt-in-type immediate
```

For Windows:

```
aws docdb apply-pending-maintenance-action ^
```

```
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:docdb ^
    --apply-action system-update ^
    --opt-in-type immediate
```

To return a list of resources that have at least one pending update, use the `describe-pending-maintenance-actions` AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws docdb describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-east-1:001234567890:db:docdb
```

For Windows:

```
aws docdb describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-east-1:001234567890:db:docdb
```

Output from this operation looks something like the following (JSON format).

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "CurrentApplyDate": "2019-01-11T03:01:00Z",
                    "Description": "db-version-upgrade",
                    "ForcedApplyDate": "2019-01-18T03:01:00Z",
                    "AutoAppliedAfterDate": "2019-01-11T03:01:00Z"
                }
            ]
        }
    ]
}
```

You can also return a list of resources for a cluster by specifying the `--filters` parameter of the`describe-pending-maintenance-actions` AWS CLI operation. The format for the `--filters` operation is `Name=`*`filter-name`*`,Values=`*`resource-id`*`,...`.

`db-cluster-id` is the acceptable values for the `Name` parameter of the filter. This value accepts a list of cluster identifiers or ARNs. The returned list only includes pending maintenance actions for the clusters identified by these identifiers or ARNs.

The following example returns the pending maintenance actions for the `sample-cluster1` and `sample-cluster2` clusters.

Example

For Linux, macOS, or Unix:

```
aws docdb describe-pending-maintenance-actions \
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

For Windows:

```
aws docdb describe-pending-maintenance-actions ^
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Apply dates

Each maintenance action has a respective apply date that you can find when describing the pending maintenance actions. When you read the output of pending maintenance actions from the AWS CLI, three dates are listed. These date values are `null` when the maintenance is optional. The values populate once the corresponding maintenance action is scheduled or applied.

- **`CurrentApplyDate`**—The date the maintenance action will get applied either immediately or during the next maintenance window.
- **`ForcedApplyDate`**—The date when the maintenance will be automatically applied, independent of your maintenance window.
- **`AutoAppliedAfterDate`**—The date after which the maintenance will be applied during the cluster's maintenance window.

User-initiated updates

As an Amazon DocumentDB user, you can initiate updates to your clusters or instances. For example, you can modify an instance's class to one with more or less memory, or you can change a cluster's parameter group. Amazon DocumentDB views these changes differently from Amazon

DocumentDB initiated updates. For more information about modifying a cluster or instance, see the following:

- Modifying an Amazon DocumentDB cluster
- Modifying an Amazon DocumentDB instance

To see a list of pending user initiated modifications, run the following command.

Example

To see pending user initiated changes for your instances

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

For Windows:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

Output from this operation looks something like the following (JSON format).

In this case, `sample-cluster-instance` has a pending change to a `db.r5.xlarge` instance class, while `sample-cluster-instance-2` has no pending changes.

```
[
    [
        "sample-cluster",
        "sample-cluster-instance",
        {
            "DBInstanceClass": "db.r5.xlarge"
        }
    ],
    [
        "sample-cluster",
        "sample-cluster-instance-2",
        {}
    ]
```

```
]
```

Managing your Amazon DocumentDB maintenance windows

Each instance and cluster has a weekly maintenance window during which any pending changes are applied. The maintenance window is an opportunity to control when modifications and software patching occur, in the event either are requested or required. If a maintenance event is scheduled for a given week, it is initiated during the 30-minute maintenance window that you identify. Most maintenance events also complete during the 30-minute maintenance window, although larger maintenance events might take more than 30 minutes to complete.

The 30-minute maintenance window is selected at random from an 8-hour block of time per Region. If you don't specify a preferred maintenance window when you create the instance or cluster, Amazon DocumentDB assigns a 30-minute maintenance window on a randomly selected day of the week.

The following table lists the time blocks for each Region from which default maintenance windows are assigned.

Region Name	Region	UTC Time Block
US East (Ohio)	us-east-2	03:00-11:00
US East (N. Virginia)	us-east-1	03:00-11:00
US West (Oregon)	us-west-2	06:00-14:00
Africa (Cape Town)	af-south-1	03:00–11:00
Asia Pacific (Hong Kong)	ap-east-1	06:00-14:00
Asia Pacific (Hyderabad)	ap-south-2	06:30–14:30
Asia Pacific (Mumbai)	ap-south-1	06:00-14:00
Asia Pacific (Seoul)	ap-northeast-2	13:00-21:00
Asia Pacific (Singapore)	ap-southeast-1	14:00-22:00
Asia Pacific (Sydney)	ap-southeast-2	12:00-20:00

Region Name	Region	UTC Time Block
Asia Pacific (Tokyo)	ap-northeast-1	13:00-21:00
Canada (Central)	ca-central-1	03:00-11:00
China (Beijing)	cn-north-1	06:00-14:00
China (Ningxia)	cn-northwest-1	06:00-14:00
Europe (Frankfurt)	eu-central-1	21:00-05:00
Europe (Ireland)	eu-west-1	22:00-06:00
Europe (London)	eu-west-2	22:00-06:00
Europe (Milan)	eu-south-1	02:00-10:00
Europe (Paris)	eu-west-3	23:59-07:29
Europe (Spain)	eu-south-2	02:00–10:00
Europe (Stockholm)	eu-north-1	04:00–12:00
Middle East (UAE)	me-central-1	05:00–13:00
South America (São Paulo)	sa-east-1	00:00-08:00
Israel (Tel Aviv)	il-central-1	04:00-12:00
AWS GovCloud (US-East)	us-gov-east-1	17:00-01:00
AWS GovCloud (US-West)	us-gov-west-1	06:00-14:00

Changing your Amazon DocumentDB maintenance windows

The maintenance window should fall at the time of lowest usage and thus might need changing from time to time. Your cluster or instance is unavailable during this time only if system changes (such as a scale storage operation or an instance class change) are being applied and require an outage. And then it is unavailable only for the minimum amount of time required to make the necessary changes.

For upgrades to the database engine, Amazon DocumentDB uses the cluster's preferred maintenance window and not the maintenance window for individual instances.

To change the maintenance window

- For a cluster, see Modifying an Amazon DocumentDB cluster.
- For an instance, see Modifying an Amazon DocumentDB instance.

Amazon DocumentDB operating system updates

Instances in Amazon DocumentDB clusters occasionally require operating system updates. Amazon DocumentDB upgrades the operating system to a newer version to improve database performance and customers' overall security posture. Operating system updates don't change the cluster engine version or instance class of an Amazon DocumentDB instance.

We recommend that you update the reader instances in a cluster first, then the writer instance to maximize the availability of your cluster. We don't recommend updating reader and writer instances at the same time, because you might incur longer downtime in the event of a failover.

Most operating system updates for Amazon DocumentDB are optional and don't have a set date to apply them. However, if you don't apply these updates for a while, they may eventually become required and automatically applied during your instance's maintenance window. This is to help maintain the security posture of your database. To avoid any unexpected downtime, we recommend that you apply operating system updates to your Amazon DocumentDB instances as soon as they become available and set your instance maintenance window at a time of your convenience as per your business needs.

To be notified when a new optional update becomes available, you can subscribe to RDS-EVENT-0230 in the security patching event category. For information about subscribing to Amazon DocumentDB events, see Subscribing to Amazon DocumentDB Event Subscriptions.

You should expect that when maintenance is performed on your cluster or instance, if the instance is a primary instance, it will fail over. To improve your availability, we recommend that you use more than one instance for your Amazon DocumentDB clusters. For more information, see Amazon DocumentDB Failover.

> **Note**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).

> **Important**
>
> Your Amazon DocumentDB instance will be taken offline during the operating system upgrade. You can minimize cluster downtime by having a multi-instance cluster. If you do not have a multi-instance cluster then you can choose to temporarily create one by adding secondary instance(s) to perform this maintenance, then deleting the additional reader instance(s) once the maintenance is completed (regular charges for the secondary instance will apply).

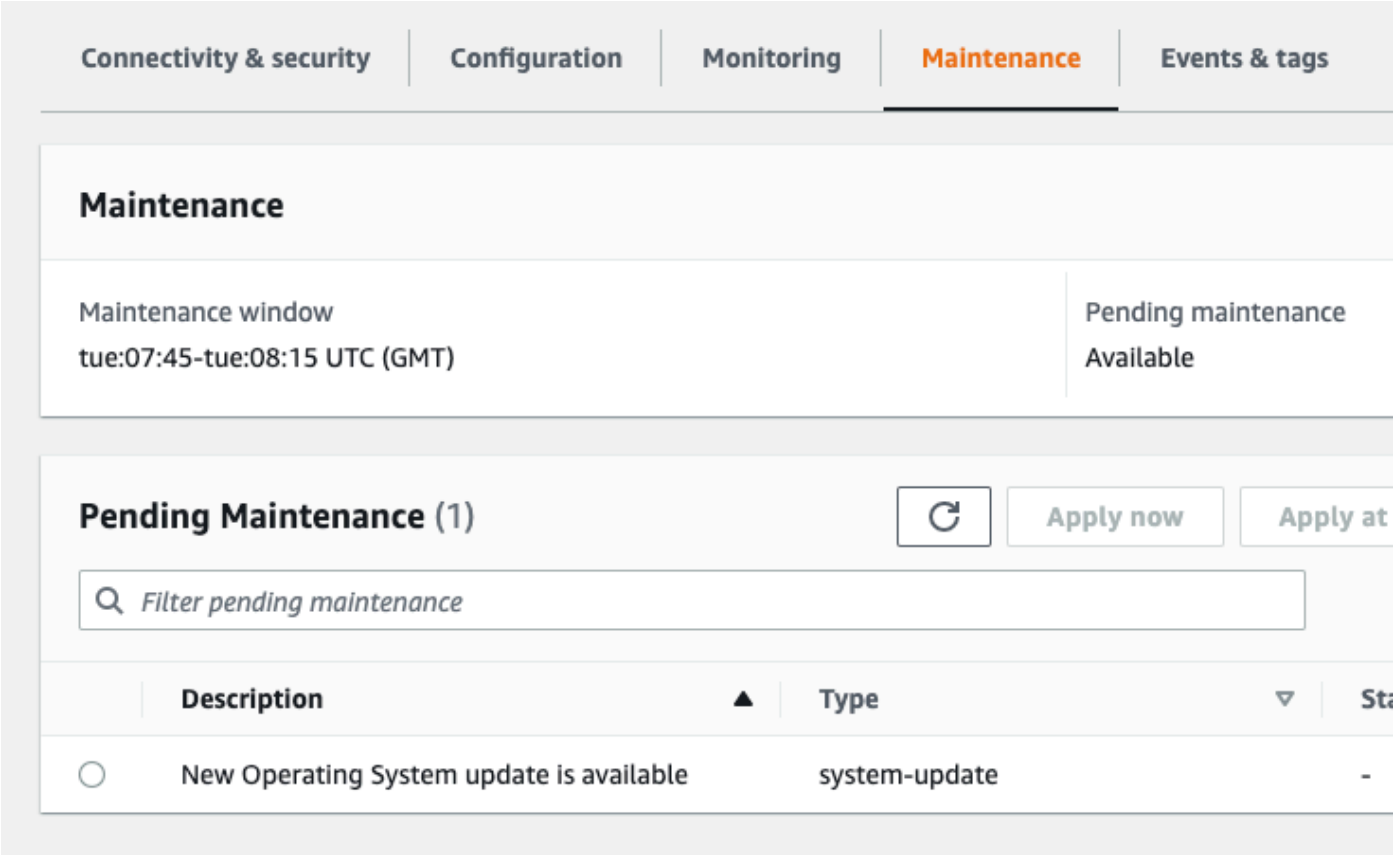
> **Note**
>
> Staying current on all optional and mandatory updates might be required to meet various compliance obligations. We recommend that you apply all updates made available by Amazon DocumentDB routinely during your maintenance windows.

You can use the AWS Management Console or the AWS CLI to determine whether an update is available.

Using the AWS Management Console

To determine whether an update is available using the AWS Management Console:

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**, and then select the instance.
3. Choose **Maintenance**.
4. In the **Pending Maintenance** section, find the operating system update.



You can select the operating system update and click **Apply now** or **Apply at next maintenance window** in the **Pending Maintenance** section. If the maintenance value is **next window**, defer the maintenance items by choosing **Defer upgrade**. You can't defer a maintenance action if it has already started.

Alternatively, you can choose the instance from a list of clusters by clicking on **Clusters** in the navigation pane and select **Apply now** or **Apply at next maintenance window** from the **Actions** menu.

Using the AWS CLI

To determine whether an update is available using the AWS CLI, call the `describe-pending-maintenance-actions` command:

```
aws docdb describe-pending-maintenance-actions
```

```
{
  "ResourceIdentifier": "arn:aws:docdb:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
```

```
    {
      "Action": "system-update",
      "Description": "New Operating System update is available"
    }
  ]
}
```

Operating system updates are specific to Amazon DocumentDB engine versions and instance classes. Therefore, Amazon DocumentDB instances receive or require updates at different times. When an operating system update is available for your instance based on its engine version and instance class, the update appears in the console. It can also be viewed by running the AWS CLI `describe-pending-maintenance-actions` command or by calling the `DescribePendingMaintenanceActions` API operation.

If you are not running the latest cluster patch release of your Amazon DocumentDB engine, you may not see operating system update listed as available maintenance. In order to view and manage the operating system update, you should first upgrade to the latest engine patch version.

Understanding service-linked roles

Amazon DocumentDB (with MongoDB compatibility) uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon DocumentDB. Service-linked roles are predefined by Amazon DocumentDB and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes using Amazon DocumentDB easier because you don’t have to manually add the necessary permissions. Amazon DocumentDB defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon DocumentDB can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your Amazon DocumentDB resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Amazon DocumentDB service-linked role permissions

Amazon DocumentDB (with MongoDB compatibility) uses the service-linked role named **AWSServiceRoleForRDS** to allow Amazon DocumentDB to call AWS services on behalf of your clusters.

The AWSServiceRoleForRDS service-linked role trusts the following services to assume the role:

- `docdb.amazonaws.com`

The role permissions policy allows Amazon DocumentDB to complete the following actions on the specified resources:

- Actions on `ec2`:
  - `AssignPrivateIpAddresses`
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeInternetGateways`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcAttribute`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`
  - `RevokeSecurityGroupIngress`
  - `UnassignPrivateIpAddresses`
- Actions on `sns`:
  - `ListTopic`
  - `Publish`
- Actions on `cloudwatch`:

- `PutMetricData`
- `GetMetricData`
- `CreateLogStream`
- `PullLogEvents`
- `DescribeLogStreams`
- `CreateLogGroup`

> **Note**
>
> You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. You might encounter the following error message:
> **Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.**
> If you see this error, ensure that you have the following permissions enabled:
>
> ```
> {
>  "Action": "iam:CreateServiceLinkedRole",
>  "Effect": "Allow",
>  "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>  "Condition": {
>  "StringLike": {
>  "iam:AWSServiceName":"rds.amazonaws.com"
>  }
>  }
> }
> ```

For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating an Amazon DocumentDB service-linked role

You don't need to manually create a service-linked role. When you create a cluster, Amazon DocumentDB creates the service-linked role for you.

If you delete this service-linked role and then need to create it again, you can use the same process to re-create the role in your account. When you create a cluster, Amazon DocumentDB creates the service-linked role for you again.

Modifying an Amazon DocumentDB service-linked role

Amazon DocumentDB does not allow you to modify the AWSServiceRoleForRDS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can modify the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting an Amazon DocumentDB service-linked role

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all of your clusters before you can delete the service-linked role.

Cleaning up an Amazon DocumentDB service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session using the console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose the name (not the check box) of the **AWSServiceRoleForRDS** role.
3. On the **Summary** page for the selected role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review the recent activity for the service-linked role.

> **Note**
>
> If you are unsure whether Amazon DocumentDB is using the AWSServiceRoleForRDS role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being

used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

If you want to remove the AWSServiceRoleForRDS role, you must first delete *all* your instances and clusters. For information about deleting instances and clusters, see the following topics:

- Deleting an Amazon DocumentDB instance
- Deleting an Amazon DocumentDB cluster

Supported regions for Amazon DocumentDB service-linked roles

Amazon DocumentDB supports using service-linked roles in all of the Regions where the service is available. For more information, see https://docs.aws.amazon.com/documentdb/latest/developerguide/regions-and-azs.html#regions-and-azs-availability.

Using change streams with Amazon DocumentDB

The change streams feature in Amazon DocumentDB (with MongoDB compatibility) provides a time-ordered sequence of change events that occur within your cluster's collections. You can read events from a change stream to implement many different use cases, including the following:

- Change notification
- Full-text search with Amazon OpenSearch Service (OpenSearch Service)
- Analytics with Amazon Redshift

Applications can use change streams to subscribe to data changes on individual collections. Change streams events are ordered as they occur on the cluster and are stored for 3 hours (by default) after the event has been recorded. The retention period can be extended up to 7 days using the `change_stream_log_retention_duration` parameter. To modify the change stream retention period, please see Modifying the Change Stream Log Retention Duration .

Topics

- Supported operations

- Billing
- Limitations
- Enabling change streams
- Example: using change streams with Python
- Full document lookup
- Resuming a change stream
- Resuming a change stream with startAtOperationTime
- Resuming a change stream with postBatchResumeToken
- Transactions in change streams
- Modifying the change stream log retention duration
- Using change streams on secondary instances

Supported operations

Amazon DocumentDB supports the following operations for change streams:

- All change events supported in the MongoDB `db.collection.watch()`, `db.watch()` and `client.watch()` API.
- Full document lookup for updates.
- Aggregation stages: `$match`, `$project`, `$redact`, and `$addFieldsand` `$replaceRoot`.
- Resuming a change stream from a resume token
- Resuming a change stream from a timestamp using `startAtOperation` (applicable to Amazon DocumentDB 4.0+)

Billing

The Amazon DocumentDB change streams feature is disabled by default and does not incur any additional charges until the feature is enabled. Using change streams in a cluster incurs additional read and write IOs and storage costs. You can use the `modifyChangeStreams` API operation to enable this feature for your cluster. For more information on pricing, see Amazon DocumentDB pricing.

Limitations

Change streams have the following limitations in Amazon DocumentDB:

- On Amazon DocumentDB 3.6. and Amazon DocumentDB 4.0, change streams can only be opened from a connection to the primary instance of an Amazon DocumentDB cluster. Reading from change streams on a replica instance is not supported on Amazon DocumentDB 3.6. and Amazon DocumentDB 4.0. When invoking the `watch()` API operation, you must specify a `primary` read preference to ensure that all reads are directed to the primary instance (see the Example section).
- On Amazon DocumentDB 5.0, change streams can be opened from both primary instance and secondary instances, including global clusters. You can specify a secondary read preference to redirect the change streams to secondary instances. See Using change streams on secondary instances for additional best practices and limitations.
- Events written to a change stream for a collection are available for up to 7 days (the default is 3 hours). Change streams data is deleted after the log retention duration window, even if no new changes have occurred.
- A long-running write operation on a collection like `updateMany` or `deleteMany` can temporarily stall the writing of change streams events until the long running write operation is complete.
- Amazon DocumentDB does not support the MongoDB operations log (`oplog`).
- With Amazon DocumentDB, you must explicitly enable change streams on a given collection.
- If the total size of a change streams event (including the change data and full document, if requested) is greater than `16 MB`, the client will experience a read failure on the change streams.
- The Ruby driver is currently not supported when using `db.watch()` and `client.watch()` with Amazon DocumentDB 3.6.
- The output from the `updateDescription` command in change streams is different in Amazon DocumentDB than in MongoDB when the updated value of the field is the same as the previous one:
  - Amazon DocumentDB doesn't return a field in the `updateDescription` output if the provided field is specified in the `$set` command and its target value is already equal to the source value.
  - MongoDB returns the field in the output, even if the specified value is equal to the current value.

Enabling change streams

You can enable Amazon DocumentDB change streams for all collections within a given database, or only for selected collections. The following are examples of how to enable change streams for different use cases using the mongo shell. Empty strings are treated as wildcards when specifying database and collection names.

```
//Enable change streams for the collection "foo" in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "foo",
    enable: true});
```

```
//Disable change streams on collection "foo" in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "foo",
    enable: false});
```

```
//Enable change streams for all collections in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "",
    enable: true});
```

```
//Enable change streams for all collections in all databases in a cluster
db.adminCommand({modifyChangeStreams: 1,
    database: "",
    collection: "",
    enable: true});
```

Change streams will be enabled for a collection if any of the following are true:

- Both the database and collection are explicitly enabled.
- The database containing the collection is enabled.
- All databases are enabled.

Dropping a collection from a database does not disable change streams for that collection if the parent database also has change streams enabled, or if all databases in the cluster are enabled. If

a new collection is created with the same name as the deleted collection, change streams will be enabled for that collection.

You can list all of your cluster's enabled change streams by using the `$listChangeStreams` aggregation pipeline stage. All aggregation stages supported by Amazon DocumentDB can be used in the pipeline for additional processing. If a previously enabled collection has been disabled, it will not appear in the `$listChangeStreams` output.

```
//List all databases and collections with change streams enabled
cursor = new DBCommandCursor(db,
    db.runCommand(
        {aggregate: 1,
        pipeline: [{$listChangeStreams: 1}],
        cursor:{}}));
```

```
//List of all databases and collections with change streams enabled
{ "database" : "test", "collection" : "foo" }
{ "database" : "bar", "collection" : "" }
{ "database" : "", "collection" : "" }
```

```
//Determine if the database “bar” or collection “bar.foo” have change streams enabled
cursor = new DBCommandCursor(db,
  db.runCommand(
      {aggregate: 1,
       pipeline: [{$listChangeStreams: 1},
                  {$match: {$or: [{database: "bar", collection: "foo"},
                                  {database: "bar", collection: ""},
                                  {database: "", collection: ""}]}}
                 ],
      cursor:{}}));
```

Example: using change streams with Python

The following is an example of using an Amazon DocumentDB change stream with Python at the collection level.

```
import os
import sys
from pymongo import MongoClient, ReadPreference

username = "DocumentDBusername"
```

```
password = <Insert your password>

clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']

#While ‘Primary’ is the default read preference, here we give an example of
#how to specify the required read preference when reading the change streams
coll = db.get_collection('foo', read_preference=ReadPreference.PRIMARY)
#Create a stream object
stream = coll.watch()
#Write a new document to the collection to generate a change event
coll.insert_one({'x': 1})
#Read the next change event from the stream (if any)
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf94f6000000020100000020000 9025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'insert'}
"""

#A subsequent attempt to read the next change event returns nothing, as there are no
 new changes
print(stream.try_next())

"""
Expected Output:
None
"""

#Generate a new change event by updating a document
result = coll.update_one({'x': 1}, {'$set': {'x': 2}})
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf99d4000000010100000010000 9025'},
```

```
'clusterTime': Timestamp(1571789268, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 2}}}
"""
```

The following is an example of using an Amazon DocumentDB change stream with Python at the database level.

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']
#Create a stream object
stream = db.watch()
coll = db.get_collection('foo')
#Write a new document to the collection foo to generate a change event
coll.insert_one({'x': 1})

#Read the next change event from the stream (if any)
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf94f6000000020100000002000009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'insert'}
"""
#A subsequent attempt to read the next change event returns nothing, as there are no
 new changes
print(stream.try_next())
```

```
"""
Expected Output:
None
"""

coll = db.get_collection('foo1')

#Write a new document to another collection to generate a change event
coll.insert_one({'x': 1})
print(stream.try_next())

"""
Expected Output: Since the change stream cursor was the database level you can see
 change events from different collections in the same database
{'_id': {'_data': '015daf94f6000000020100000020000902 5'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo1', 'db': 'bar'},
'operationType': 'insert'}
"""
```

Full document lookup

The update change event does not include the full document; it includes only the change that was made. If your use case requires the complete document affected by an update, you can enable full document lookup when opening the stream.

The `fullDocument` document for an update change streams event represents the most current version of the updated document at the time of document lookup. If changes occurred between the update operation and the `fullDocument` lookup, the `fullDocument` document might not represent the document state at update time.

To create a stream object with update lookup enabled, use this example:

```
stream = coll.watch(full_document='updateLookup')

#Generate a new change event by updating a document
result = coll.update_one({'x': 2}, {'$set': {'x': 3}})

stream.try_next()
```

The output of the stream object will look something like this:

```
{'_id': {'_data': '015daf9b7c0000000101000000100009025'},
'clusterTime': Timestamp(1571789692, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'fullDocument': {'_id': ObjectId('5daf9502ea258751778163d7'), 'x': 3},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 3}}}
```

Resuming a change stream

You can resume a change stream later by using a resume token, which is equal to the `_id` field of the last retrieved change event document.

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem', retryWrites='false')

db = client['bar']
coll = db.get_collection('foo')
#Create a stream object
stream = db.watch()
coll.update_one({'x': 1}, {'$set': {'x': 4}})
event = stream.try_next()
token = event['_id']
print(token)

"""
Output: This is the resume token that we will later us to resume the change stream
{'_data': '015daf9c5b0000000101000000100009025'}
"""
#Python provides a nice shortcut for getting a stream’s resume token
print(stream.resume_token)

"""
```

```
Output
{'_data': '015daf9c5b0000000101000000100009025'}
"""
#Generate a new change event by updating a document
result = coll.update_one({'x': 4}, {'$set': {'x': 5}})
#Generate another change event by inserting a document
result = coll.insert_one({'y': 5})
#Open a stream starting after the selected resume token
stream = db.watch(full_document='updateLookup', resume_after=token)
#Our first change event is the update with the specified _id
print(stream.try_next())

"""
#Output: Since we are resuming the change stream from the resume token, we will see all
 events after the first update operation. In our case, the change stream will resume
 from the update operation {x:5}

{'_id': {'_data': '015f7e8f0c0000000601000000006000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602129676, 6),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e8f0ac423bafbfd9adba2')},
'fullDocument': {'_id': ObjectId('5f7e8f0ac423bafbfd9adba2'), 'x': 5},
'updateDescription': {'updatedFields': {'x': 5}, 'removedFields': []}}
"""
#Followed by the insert
print(stream.try_next())

"""
#Output:
{'_id': {'_data': '015f7e8f0c0000000701000000007000fe038'},
'operationType': 'insert',
'clusterTime': Timestamp(1602129676, 7),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e8f0cbf8c233ed577eb94')},
'fullDocument': {'_id': ObjectId('5f7e8f0cbf8c233ed577eb94'), 'y': 5}}
"""
```

Resuming a change stream with `startAtOperationTime`

You can resume a change stream later from a particular time stamp by using `startAtOperationTime`.

> **Note**
>
> The ability to use `startAtOperationTime` is available in Amazon DocumentDB 4.0+. When using `startAtOperationTime`, the change stream cursor will only return changes that occurred at or after the specified Timestamp. The `startAtOperationTime` and `resumeAfter` commands are mutually exclusive and thus cannot be used together.

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='rds-root-ca-2020.pem',retryWrites='false')
db = client['bar']
coll = db.get_collection('foo')
#Create a stream object
stream = db.watch()
coll.update_one({'x': 1}, {'$set': {'x': 4}})
event = stream.try_next()
timestamp = event['clusterTime']
print(timestamp)
"""
Output
Timestamp(1602129114, 4)
"""
#Generate a new change event by updating a document
result = coll.update_one({'x': 4}, {'$set': {'x': 5}})
result = coll.insert_one({'y': 5})
#Generate another change event by inserting a document
#Open a stream starting after specified time stamp

stream = db.watch(start_at_operation_time=timestamp)
print(stream.try_next())

"""
#Output: Since we are resuming the change stream at the time stamp of our first update
 operation (x:4), the change stream cursor will point to that event
{'_id': {'_data': '015f7e941a0000000301000000003000fe038'},
```

```
'operationType': 'update',
'clusterTime': Timestamp(1602130970, 3),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e9417c423bafbfd9adbb1')},
'updateDescription': {'updatedFields': {'x': 4}, 'removedFields': []}}
"""

print(stream.try_next())
"""
#Output: The second event will be the subsequent update operation (x:5)
{'_id': {'_data': '015f7e9502000000050100000005000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602131202, 5),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e94ffc423bafbfd9adbb2')},
'updateDescription': {'updatedFields': {'x': 5}, 'removedFields': []}}
"""

print(stream.try_next())

"""
#Output: And finally the last event will be the insert operation (y:5)
{'_id': {'_data': '015f7e9502000000060100000006000fe038'},
'operationType': 'insert',
'clusterTime': Timestamp(1602131202, 6),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e95025c4a569e0f6dde92')},
'fullDocument': {'_id': ObjectId('5f7e95025c4a569e0f6dde92'), 'y': 5}}
"""
```

Resuming a change stream with `postBatchResumeToken`

Amazon DocumentDB change stream now returns an additional field called `postBatchResumeToken`. This field is returned from the `$changestream` command and `getMore` command.

Example of the `$changestream` command in Python:

```
db.command({"aggregate": "sales", "pipeline": [{ "$changeStream": {}}], "cursor":
 {"batchSize": 1}})
```

Expected output:

```
cursor" : {
   "firstBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

Example of the `getMore` command in Python:

```
db.command({"getMore": NumberLong(<cursor id>), "collection": "sales", "batchSize":
 1 })
```

Expected output

```
cursor" : {
   "nextBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

The `postBatchResumeToken` field can be used to open new change stream cursors in the `resumeAfter` field, similar to how the resume token is used.

Open a stream starting after the selected `postBatchResumeToken`:

```
post_batch_resume_token = output['cursor']['postBatchResumeToken']
stream = db.watch(full_document='updateLookup', resume_after=post_batch_resume_token)
```

Unlike a regular resume token that always corresponds to an operations log (oplog) entry that reflects an actual event, `postBatchResumeToken` corresponds to an oplog entry the change stream has scanned up to on the server, which is not necessarily a matching change.

Attempting to resume with an old regular resume token will force the database to scan all the oplog entries between the specified time stamp and the current time. This may generate a lot of queries internally with each sub-query scanning for a small period of time. This will cause a spike in CPU usage and degrade the database performance. Resuming with the last `postBatchResumeToken` skips the scanning of unmatched oplog entries.

Transactions in change streams

Change stream events will not contain events from uncommitted and/or aborted transactions. For example, if you start a transaction with one `INSERT` operation and one `UPDATE` operation, and if your `INSERT` operation succeeds, but the `UPDATE` operation fails, the transaction will be rolled back. Since this transaction was rolled back, your change stream will not contain any events for this transaction.

Modifying the change stream log retention duration

You can modify the change stream log retention duration to be between 1 hour and 7 days using the AWS Management Console or the AWS CLI.

Using the AWS Management Console

To modify the change stream log retention duration

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups** .

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon
   > (≡)
   > in the upper-left corner of the page.

3. In the **Parameter groups** pane, choose the cluster parameter group that is associated with your cluster. To identify the cluster parameter group that is associated with your cluster, see Determining an Amazon DocumentDB cluster's parameter group.
4. The resulting page shows the parameters and their corresponding details for your cluster parameter group. Select the parameter `change_stream_log_retention_duration`.
5. On the top right of the page, choose **Edit** to change the value of the parameter. The `change_stream_log_retention_duration` parameter can be modified to be between 1 hour and 7 days.
6. Make your change, and then choose **Modify cluster parameter** to save the changes. To discard your changes, choose **Cancel**.

Using the AWS CLI

To modify your cluster parameter group's `change_stream_log_retention_duration` parameter, use the `modify-db-cluster-parameter-group` operation with the following parameters:

- **`--db-cluster-parameter-group-name`** — Required. The name of the cluster parameter group that you are modifying. To identify the cluster parameter group that is associated with your cluster, see Determining an Amazon DocumentDB cluster's parameter group.
- **`--parameters`** — Required. The parameter that you are modifying. Each parameter entry must include the following:
  - **`ParameterName`** — The name of the parameter that you are modifying. In this case, it is `change_stream_log_retention_duration`
  - **`ParameterValue`** — The new value for this parameter.
  - **`ApplyMethod`** — How you want changes to this parameter applied. Permitted values are `immediate` and `pending-reboot`.

> **Note**
>
> Parameters with the `ApplyType` of `static` must have an `ApplyMethod` of `pending-reboot`.

1. To change the values of the parameter `change_stream_log_retention_duration`, run the following command and replace `parameter-value` with the value you want to modify the parameter to.

   For Linux, macOS, or Unix:

   ```
   aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
    "ParameterName=change_stream_log_retention_duration,ParameterValue=<parameter-
   value>,ApplyMethod=immediate"
   ```

   For Windows:

   ```
   aws docdb modify-db-cluster-parameter-group ^
   ```

```
    --db-cluster-parameter-group-name sample-parameter-group ^
    --parameters
 "ParameterName=change_stream_log_retention_duration,ParameterValue=<parameter-
value>,ApplyMethod=immediate"
```

Output from this operation looks something like the following (JSON format).

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

2. Wait at least 5 minutes.
3. List the parameter values of `sample-parameter-group` to ensure that your changes have been made.

   For Linux, macOS, or Unix:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

   For Windows:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

   Output from this operation looks something like the following (JSON format).

```
{
    "Parameters": [
        {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        },
        {
```

```
            "ParameterName": "change_stream_log_retention_duration",
            "ParameterValue": "12345",
            "Description": "Duration of time in seconds that the change stream
 log is retained and can be consumed.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "3600-86400",
            "IsModifiable": true,
            "ApplyMethod": "immediate"
        }
    ]
}
```

> **Note**
>
> Change stream log retention will not delete logs older than the configured `change_stream_log_retention_duration` value until log size is greater than (>) 51,200MB.

Using change streams on secondary instances

To get started on using change stream on secondary instances, open the change stream cursor with `readPreference` as the secondary.

You can open a change stream cursor to watch for change events on a specific collection or all collections in a cluster or database. You can open a change stream cursor on any Amazon DocumentDB instance and fetch change stream documents from both writer and reader instances. You can share change stream tokens (such as `resumeToken` or `startOperationTime`) across different change stream cursors opened on a writer and reader instance.

Example

```
import os
import sys
from pymongo import MongoClient, ReadPreference

username = "DocumentDBusername"
password = <Your password>
```

```
clusterendpoint = "DocumentDBClusterEndpoint"

client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']

# Make sure to use SECONDARY to redirect cursor reads from secondary instances
coll = db.get_collection('foo', read_preference=ReadPreference.SECONDARY)

# Create a stream object on RO. The token needs to generated from PRIMARY.
stream = coll.watch(resumeAfter=token)

for event in stream:
   print(event)
```

Guidelines and limitations for change streams on secondary instances

- Change stream events need to be replicated from the primary instance to the secondary instances. You can monitor the lag from the `DBInstanceReplicaLag` metric in Amazon CloudWatch.
- Timestamps on secondary instances may not always be in sync with the primary instance. In this case, expect delays on the secondary instance timestamp so it can catch up. As a best practice, we recommend using `startAtOperationTime` or `resumeToken` to start the watch on the secondary instance.
- You might experience lower throughput on secondary instances compared to the primary instance if your document size is large, you are doing `fullDocumentLookup`, and there is high concurrent write workload on the primary instance. As a best practice, we recommend you monitor your buffer cache hit ratio on the secondary and make sure that buffer cache hit ratio is high.

Using AWS Lambda with change streams

Amazon DocumentDB is integrated with AWS Lambda so you can use Lambda functions to process records in a change stream. Lambda event source mapping is a resource that can be used to invoke Lambda functions in order to process Amazon DocumentDB events that don’t directly invoke Lambda. With Amazon DocumentDB change stream as an event source, you can build event-driven

applications that respond to changes in your data. For example, you can use Lambda functions to process new documents, track updates to existing documents, or log deleted documents.

You can configure an event source mapping to send records from your Amazon DocumentDB change stream to a Lambda function. Events can be sent one at a time or batched for improved efficiency and will processed in-order. You can configure your event source mapping's batching behavior based on a specific time window duration (0 - 300 sec) or batch record count (max limit of 10,000 records). You can create multiple event source mappings to process the same data with multiple Lambda functions, or to process distinct items from multiple streams with a single function.

If your function returns an error, Lambda retries the batch until it processes successfully. In case the events in the change stream have expired, Lambda will disable the event source mapping. In this case, you can create a new event source mapping and configure it with a starting position of your choice. Lambda event source mappings process events at least once due to the distributed nature of its pollers. As a result, your Lambda function may receive duplicate events in rare situations. Follow best practices for working with AWS Lambda functions and build idempotent functions to avoid issues related to duplicate events. For more information see Using AWS Lambda console with Amazon DocumentDB in the *AWS Lambda Developer Guide*.

As performance best practices, the Lambda function needs to be short lived. To avoid introducing unnecessary processing delays, it also should not execute complex logic. For a high velocity stream in particular, it is better to trigger an asynchronous post-processing step function workflows than synchronous long running Lambdas. For more information about AWS Lambda, see the AWS Lambda Developer Guide.

Limitations

The following are limitations to consider when working with Amazon DocumentDB and AWS Lambda:

- AWS Lambda is currently supported only on Amazon DocumentDB 4.0 and 5.0.
- AWS Lambda is not currently supported on elastic clusters or global clusters.
- AWS Lambda payload sizes cannot exceed 6MB. For more information about Lambda batch sizes, see "Batching behavior" in Lambda event source mappings section in the *AWS Lambda Developer Guide*.

Using Amazon DocumentDB elastic clusters

Amazon DocumentDB elastic clusters support workloads with millions of reads/writes per second and petabytes of storage capacity. Elastic clusters also simplify how developers interact with Amazon DocumentDB by eliminating the need to choose, manage or upgrade instances.

Amazon DocumentDB elastic clusters were created to:

- Provide a solution for customers looking for a database that provides virtually limitless scale with rich query capabilities and MongoDB API compatibility.
- Give customers higher connection limits, and to reduce downtime from patching.
- Continue investing in a cloud-native, elastic, and class leading architecture for JSON workloads.

Topics

- Elastic cluster use cases
- Advantages of elastic clusters
- Elastic cluster region and version availability
- Limitations
- Amazon DocumentDB elastic clusters: how it works
- Get started with Amazon DocumentDB elastic clusters
- Amazon DocumentDB elastic cluster best practices
- Managing Amazon DocumentDB elastic clusters
- Data encryption at rest for Amazon DocumentDB elastic clusters
- Service-linked roles in elastic clusters

Elastic cluster use cases

Document databases are useful for workloads that require a flexible schema for fast, iterative development. For example Amazon DocumentDB use cases, see Document database use Cases.

The following are some examples of use cases for which elastic clusters can provide significant advantages:

User profiles

Because document databases have a flexible schema, they can store documents that have different attributes and data values at scale. Elastic clusters are a practical solution to online profiles in which different users provide different types of information. Suppose that your applications support hundreds of millions of user profiles. You can use elastic clusters to support such applications because they can be scaled up and out to support millions of writes and reads to these user profiles. You can also scale down for off-peak hours to reduce cost.

Content management and historical records

To effectively manage content, you must be able to collect and aggregate content from a variety of sources, and then deliver it to the customer. Due to their flexible schema, document databases are perfect for collecting and storing any type of data. You can use them to create and incorporate new types of content, including user-generated content such as images, comments, and videos. Over time, your database may require more storage. With elastic clusters, you can distribute your data over more storage volumes enabling you to store petabytes of data in a single cluster.

Advantages of elastic clusters

AWS service integration

Amazon DocumentDB elastic clusters integrate with other AWS services in the same way Amazon DocumentDB does:

- **Migration** — You can use AWS Database Migration Service (DMS) to migrate from MongoDB and other relational databases to Amazon DocumentDB elastic clusters.
- **Monitoring** — You can monitor the health and performance of your elastic cluster using Amazon CloudWatch.
- **Security** — You can set up authentication and authorization through AWS Identity and Access Management (IAM) to manage your elastic cluster resources such as creating and updating an elastic cluster, and use Amazon VPC for secure VPC-only connections. IAM authentication login for elastic clusters is not supported.
- **Data management** — You can use AWS Glue to import and export data from/to other AWS services such as Amazon S3, Amazon Redshift and Amazon OpenSearch Service.

Elastic cluster region and version availability

Region availability for elastic clusters

The following table shows the AWS regions where Amazon DocumentDB elastic clusters are currently available and the endpoint for each region.

Region name	Region	Availability zones
US East (N. Virginia)	`us-east-1`	5
US East (Ohio)	`us-east-2`	3
US West (Oregon)	`us-west-2`	3
Asia Pacific (Hong Kong)	`ap-east-1`	3
Asia Pacific (Mumbai)	`ap-south-1`	3
Asia Pacific (Seoul)	`ap-northeast-2`	3
Asia Pacific (Singapore)	`ap-southeast-1`	3
Asia Pacific (Sydney)	`ap-southeast-2`	3
Asia Pacific (Tokyo)	`ap-northeast-1`	3
Canada (Central)	`ca-central-1`	3
South America (São Paulo)	`sa-east-1`	3
Europe (Frankfurt)	`eu-central-1`	3
Europe (Ireland)	`eu-west-1`	3
Europe (London)	`eu-west-2`	3
Europe (Milan)	`eu-south-1`	3
Europe (Paris)	`eu-west-3`	3

Version availability

Elastic clusters support the MongoDB 5.0-compatable wire protocol. For differences between Amazon DocumentDB 4.0 instance-based clusters and elastic clusters, see Functional differences between Amazon DocumentDB 4.0 and elastic clusters.

Limitations

Elastic cluster management

The following cluster management features and capabilities are not supported in this release:

- Ability to use IAM users and roles to authenticate into the database
- Ability to create global clusters
- Existing Amazon DocumentDB events and subscribing to events
- Range sharding
- Shard existing collection
- Multi-field shard key
- Change shard key
- Point-in-time restore
- Cloning
- Performance Insights

> **Note**
>
> For information about elastic cluster limits, see Amazon DocumentDB Quotas and limits.

Query and write operations

The following query and write operation commands and capabilities are not supported in this release:

- DDL commands during scaling operations

- Profiler
- Parameter groups
- AWS Config
- AWS Backup

Collection and index management

The following collection and index management features are not supported in this release:

- Unique indexes
- Partial indexes
- Text indexes
- Vector indexes
- Document compression

Administration and diagnostics

The following administration and diagnostic commands and capabilities are not supported in this release:

- AWS Secrets Manager
- Role-based-access-control (RBAC) custom roles.
- When connecting, write concern of 0 is not supported.
- Changing subnets belonging to an VPC that is not currently assigned to an existing elastic cluster.

Opt-in features

The following Amazon DocumentDB opt-in features are not supported in this release:

- ACID transactions
- DDL/DML auditing
- Change streams
- Session commands

Amazon DocumentDB elastic clusters: how it works

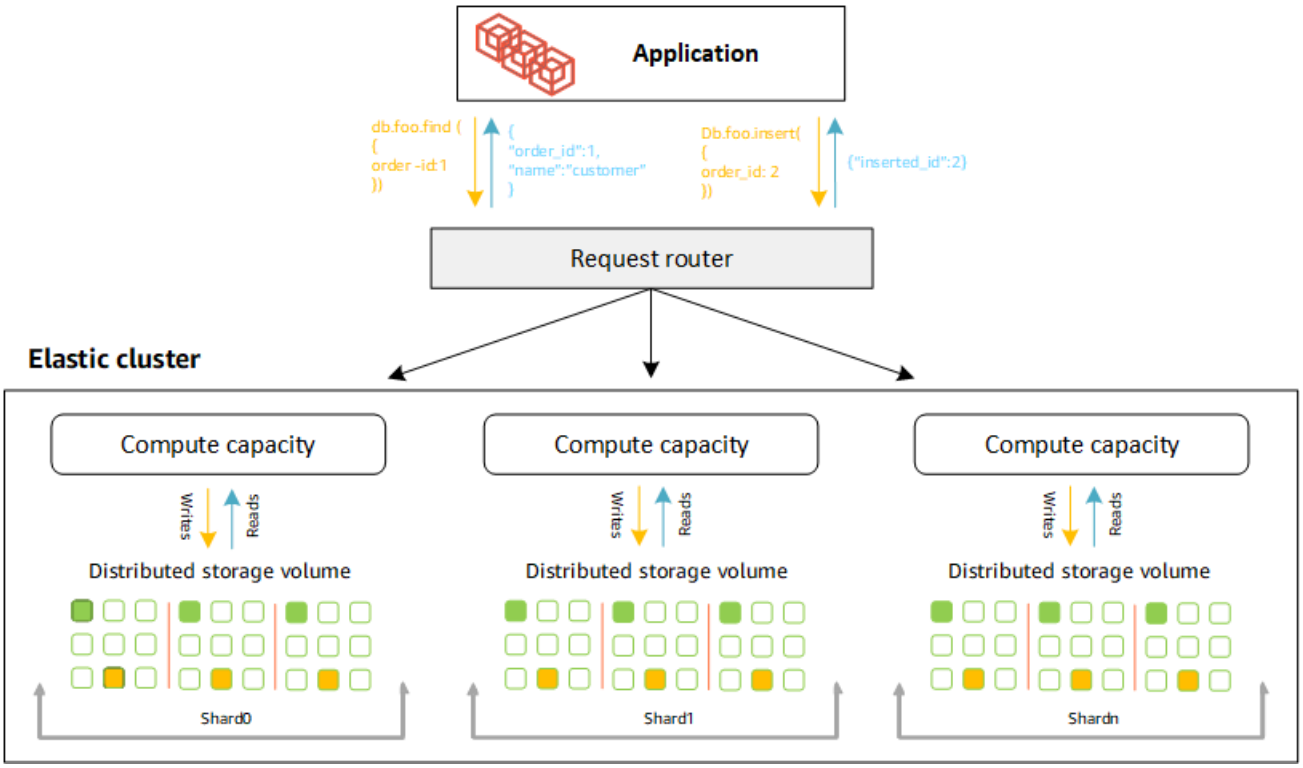
The topics in this section provide information about the mechanisms and functions that power Amazon DocumentDB elastic clusters.

Topics

- Amazon DocumentDB elastic cluster sharding
- Elastic cluster migration
- Elastic cluster scaling
- Elastic cluster reliability
- Elastic cluster storage and availability
- Functional differences between Amazon DocumentDB 4.0 and elastic clusters

Amazon DocumentDB elastic cluster sharding

Amazon DocumentDB elastic clusters use hash-based sharding to partition data across a distributed storage system. Sharding, also known as partitioning, splits large data sets into small data sets across multiple nodes enabling you to scale out your database beyond vertical scaling limits. Elastic clusters use the separation, or “decoupling,” of compute and storage in Amazon DocumentDB, enabling you to scale independently of each other. Rather than re-partitioning collections by moving small chunks of data between compute nodes, elastic clusters copy data efficiently within the distributed storage system.



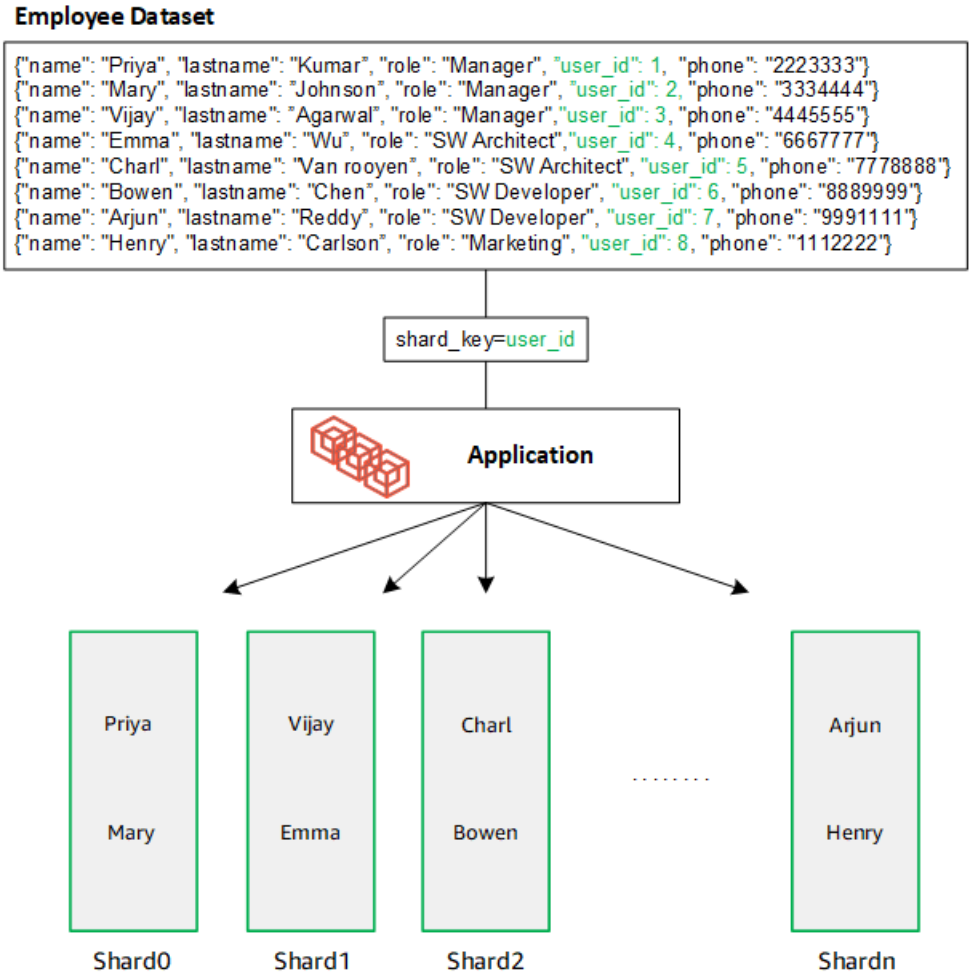
Shard definitions

Definitions of shard nomenclature:

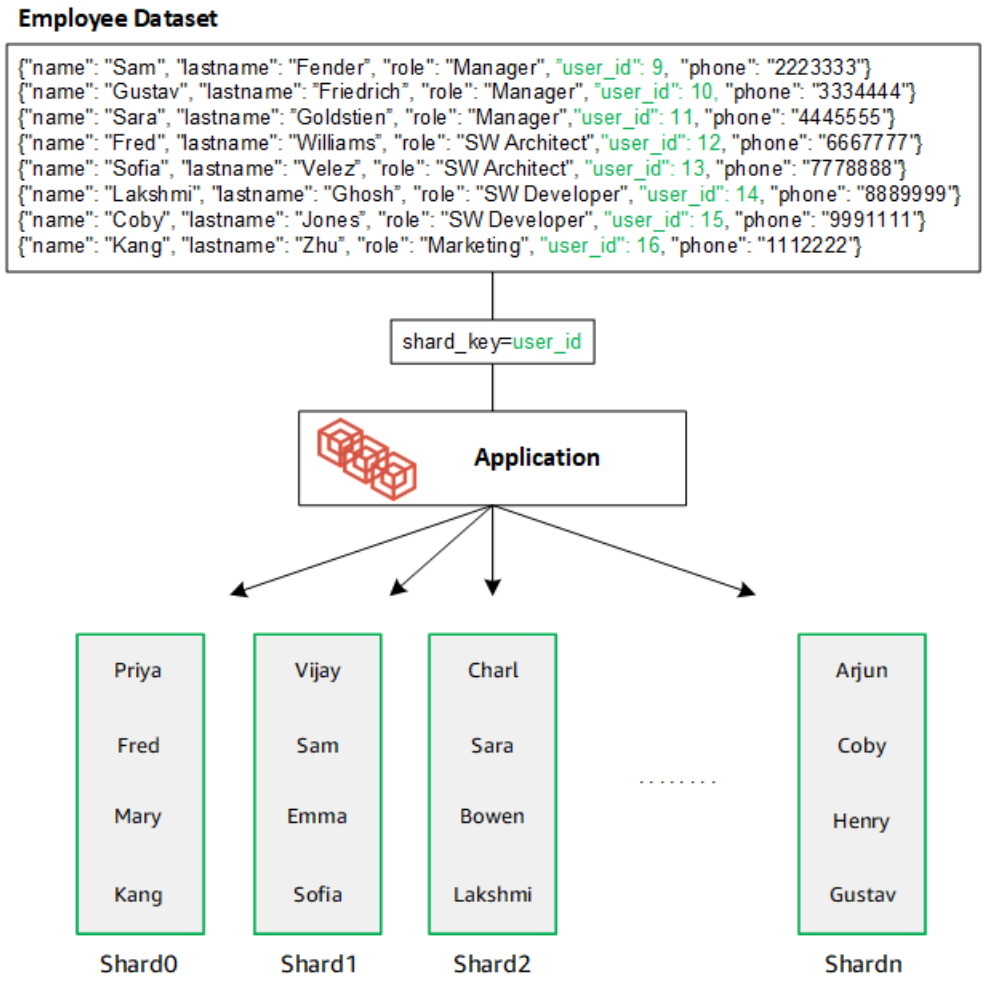
- **Shard** — A shard provides compute for an elastic cluster. It will have a single writer instance and 0–15 read replicas. By default, a shard will have two instances: a writer and a single read replica. You can configure a maximum of 32 shards and each shard instance can have a maximum of 64 vCPUs.
- **Shard key** — A shard key is a required field in your JSON documents in sharded collections that elastic clusters use to distribute read and write traffic to the matching shard.
- **Sharded collection** — A sharded collection is a collection whose data is distributed across an elastic cluster in data partitions.
- **Partition** — A partition is a logical portion of sharded data. When you create a sharded collection, the data is organized into partitions within each shard automatically based on the shard key. Each shard has multiple partitions.

Distributing data across configured shards

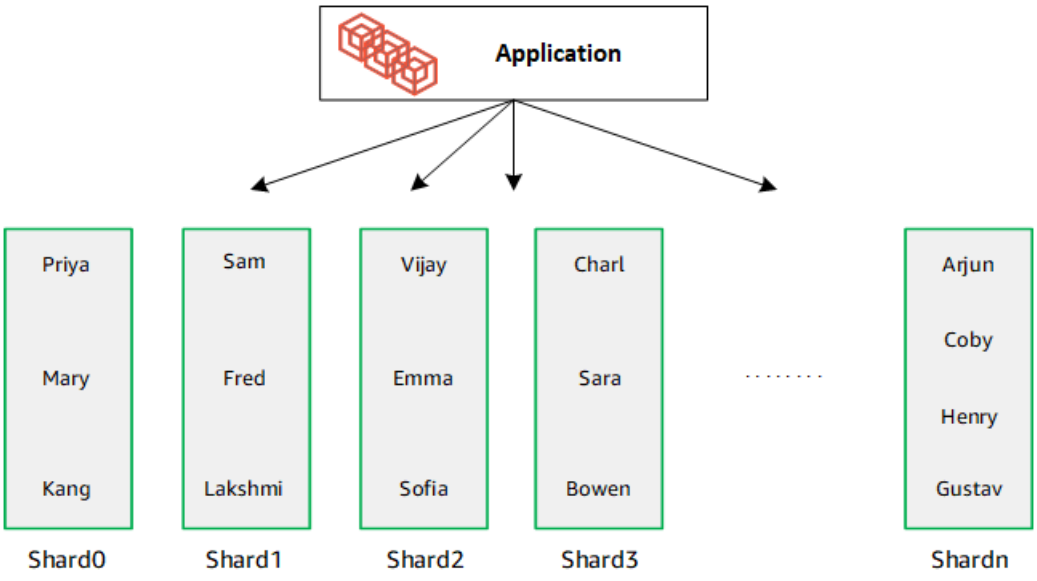
Create a shard key that has many unique values. A good shard key will evenly partition your data across the underlying shards, giving your workload the best throughput and performance. The following example is employee name data that uses a shard key named "user_id":



DocumentDB uses hash sharding to partition your data across underlying shards. Additonal data is inserted and distributed the same way:



When you scale out your database by adding additional shards, Amazon DocumentDB automatically redistributes the data:



Elastic cluster migration

Amazon DocumentDB supports migrating MongoDB sharded data to elastic clusters. Offline, online, and hybrid migration methods are supported. For more information, see Migrating to Amazon DocumentDB.

Elastic cluster scaling

Amazon DocumentDB elastic clusters provide the ability to increase the number of shards (scale out) in your elastic cluster, and the number of vCPUs applied to each shard (scale up). You can also reduce the number of shards and compute capacity (vCPUs) as needed.

For scaling best practices, see Scaling elastic clusters.

> **Note**
>
> Cluster-level scaling is also available. For more information, see Scaling Amazon DocumentDB clusters.

Elastic cluster reliability

Amazon DocumentDB is designed to be reliable, durable, and fault-tolerant. To improve availability, elastic clusters deploy two nodes per shard placed across different Availability Zones. Amazon DocumentDB includes several automatic features that make it a reliable database solution. For more information, see Amazon DocumentDB reliability.

Elastic cluster storage and availability

Amazon DocumentDB data is stored in a cluster volume, which is a single, virtual volume that uses solid state drives (SSDs). A cluster volume consists of six copies of your data, which are replicated automatically across multiple Availability Zones in a single AWS Region. This replication helps ensure that your data is highly durable, with less possibility of data loss. It also helps ensure that your cluster is more available during a failover because copies of your data already exist in other Availability Zones. For more details on storage, high availability, and replication see Amazon DocumentDB: how it works.

Functional differences between Amazon DocumentDB 4.0 and elastic clusters

The following functional differences exist between Amazon DocumentDB 4.0 and elastic clusters.

- Results from `top` and `collStats` are partitioned by shards. For sharded collections, data is distributed among multiple partitions and `collStats` reports aggregated `collScans` from the partitions.
- Collection statistics from `top` and `collStats` for sharded collections are reset when the cluster shard count is changed.
- The backup built-in role now supports `serverStatus`. Action - Developers and applications with backup role can collect statistics about the state of the Amazon DocumentDB cluster.
- The `SecondaryDelaySecs` field replaces `slaveDelay` in `replSetGetConfig` output.
- The `hello` command replaces `isMaster` - `hello` returns a document that describes the role of the elastic cluster.
- The `$elemMatch` operator in elastic clusters only matches documents in the first nesting level of an array. In Amazon DocumentDB 4.0, the operator traverses all levels before returning matched documents. For example:

```
db.foo.insert(
[
    {a: {b: 5}},
    {a: {b: [5]}},
    {a: {b: [3, 7]}},
    {a: [{b: 5}]},
    {a: [{b: 3}, {b: 7}]},
    {a: [{b: [5]}]},
    {a: [{b: [3, 7]}]},
    {a: [[{b: 5}]]},
    {a: [[{b: 3}, {b: 7}]]},
    {a: [[{b: [5]}]]},
    {a: [[{b: [3, 7]}]]}
]);
// Elastic clusters
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
```

```
// Docdb 4.0: traverse more than one level deep
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
{ "a" : [ [ { "b" : [ 5 ] } ] ] }
```

- The "$" projection in Amazon DocumentDB 4.0 returns all documents with all fields. With elastic clusters, the `find` command with a "$" projection returns documents that match the query parameter containing only the field that matched the "$" projection.
- In elastic clusters, the `find` commands with `$regex` and `$options` query parameters return an error: "Cannot set options in both $regex and $options."

- With elastic clusters, `$indexOfCP` now returns "-1" when:
  - the substring is not found in the `string expression`, or
  - `start` is a number greater than end, or
  - `start` is a number greater than the byte length of the string.

  In Amazon DocumentDB 4.0, `$indexOfCP` returns "0" when the `start` position is a number greater than end or the byte length of the string.
- With elastic clusters, projection operations in `_id fields`, e.g., `{"_id.nestedField" : 1}`, return documents that only include the projected field. Meanwhile, in Amazon DocumentDB 4.0, nested field projection commands do not filter out any document.

Get started with Amazon DocumentDB elastic clusters

This getting started section walks you through on how you can create and query your first elastic cluster.

There are many ways to connect and get started with Amazon DocumentDB. The following procedure is the quickest, simplest, and easiest way for users to get started using our powerful document database. This guide uses AWS CloudShell to connect and query your Amazon DocumentDB cluster directly from the AWS Management Console. New customers who are eligible for the AWS Free Tier can use Amazon DocumentDB and CloudShell for free. If your AWS CloudShell environment or Amazon DocumentDB elastic cluster makes use of resources beyond the free tier, you are charged the normal AWS rates for those resources. This guide will get you started with Amazon DocumentDB in less than 5 minutes.

Topics

- Prerequisites
- Step 1: Create an elastic cluster
- Step 2: Connect to your elastic cluster
- Step 3: Shard your collection, insert and query data
- Step 4: Explore

Prerequisites

Before you create your first Amazon DocumentDB cluster, you must do the following:

Create an Amazon Web Services (AWS) account

Before you can begin using Amazon DocumentDB, you must have an Amazon Web Services (AWS) account. The AWS account is free. You pay only for the services and resources that you use.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

Set up the needed AWS Identity and Access Management (IAM) permissions.

Access to manage Amazon DocumentDB resources such as clusters, instances, and cluster parameter groups requires credentials that AWS can use to authenticate your requests. For more information, see Identity and Access Management for Amazon DocumentDB.

1. In the search bar of the AWS Management Console, type in IAM and select **IAM** in the drop down menu.
2. Once you're in the IAM console, select **Users** from the navigation pane.
3. Select your username.
4. Click **Add permissions**.
5. Select **Attach policies directly**.
6. Type `AmazonDocDBElasticFullAccess` in the search bar and select it once it appears in the search results.
7. Click **Next**.
8. Click **Add permissions**.

> **Note**
>
> Your AWS account includes a default VPC in each region. If you choose to use an Amazon VPC, complete the steps in the Create a Amazon VPC topic in the *Amazon VPC User Guide*.

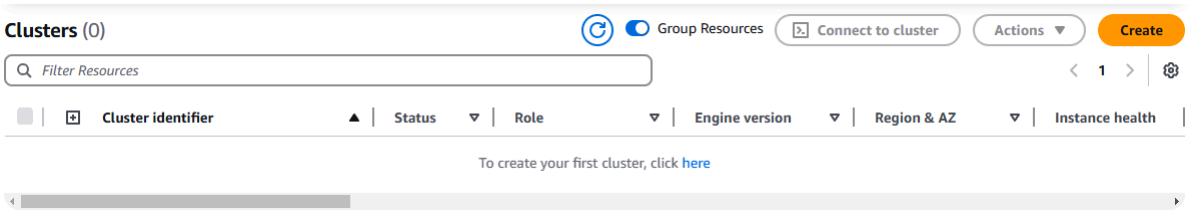
Step 1: Create an elastic cluster

In this section we explain how to create a brand new elastic cluster, using either the AWS Management Console or AWS CLI with the following instructions.

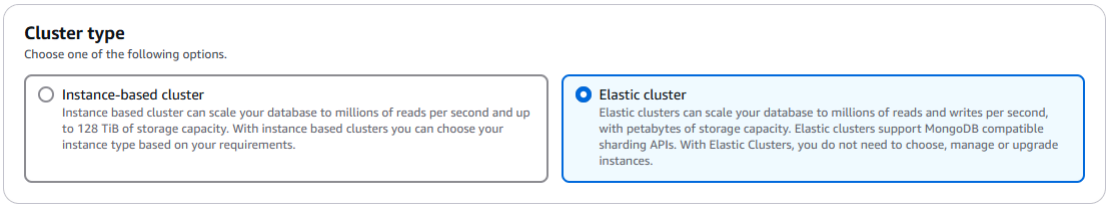
Using the AWS Management Console

To create an elastic cluster configuration using the AWS Management Console:

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. On the **Amazon DocumentDB Management Console**, under **Clusters**, choose **Create**.



3. On the **Create Amazon DocumentDB cluster** page, in the **Cluster type** section, choose **Elastic cluster**.



4. In the **Configuration** section, configure the following:

   a. In the **Cluster name** field, enter a unique cluster identifier (following the naming requirements below the field).

   b. In the **Shard count** field, enter the number of shards you want in your cluster. The maximum number of shards per cluster is 32.

   > **Note**
   >
   > Two nodes will be deployed for each shard. Both nodes will have the same shard capacity.

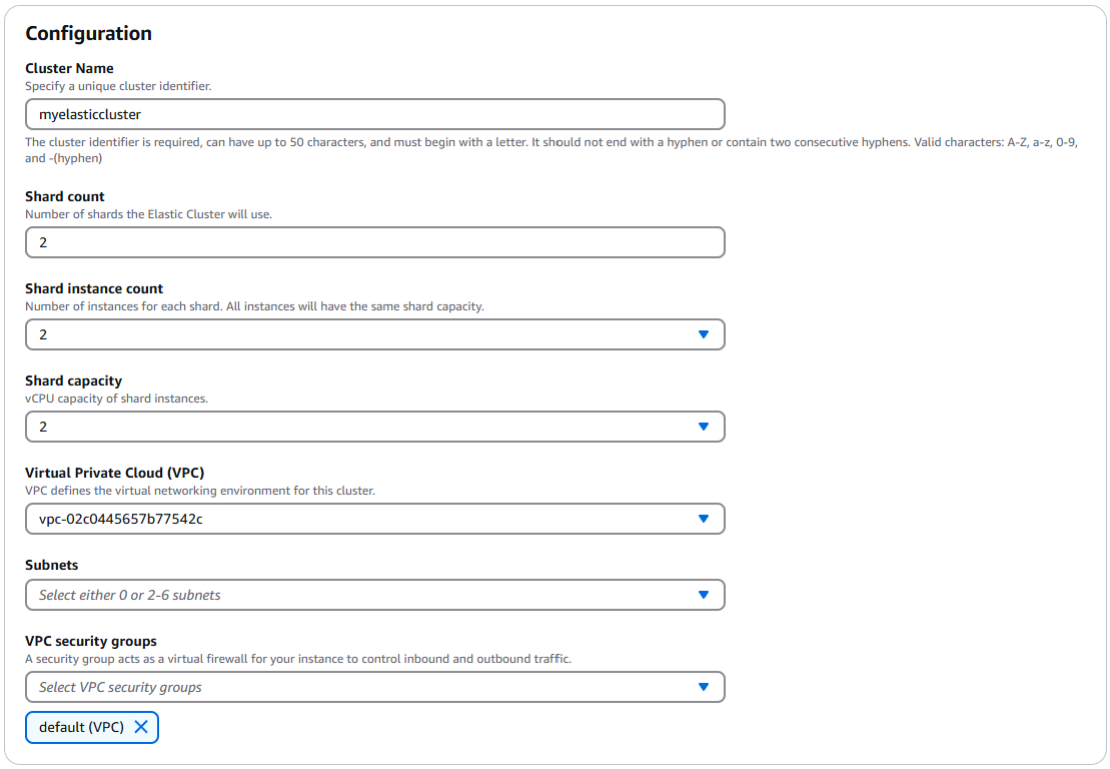
   c. In the **Shard instance count** field, choose the number of replica instances you want associated with each shard. The maximum number of shard instances is 16, in increments of 1. All replica instances have the same shard capacity as defined in the following field. For testing purposes, the default value of 2 should suffice.

   > **Note**
   >
   > The number of replica instances applies to all shards in the elastic cluster. A shard instance count value of 1 means there is one writer instance, and any additional instances are replicas that can be used for reads and to improve availability. For testing purposes, the default value of 2 should suffice.

   d. In the **Shard capacity** field, choose the number of virtual CPUs (vCPUs) you want associated with each shard instance. The maximum number of vCPUs per shard instance is 64. Allowed values are 2, 4, 8, 16, 32, 64. For testing purposes, the default value of 2 should suffice.

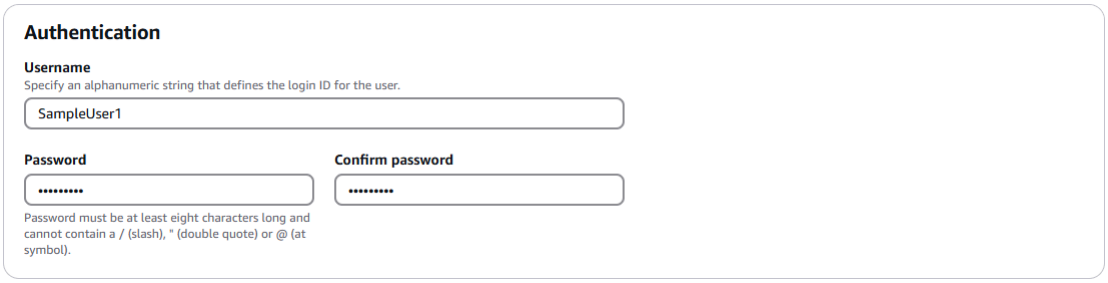
   e. In the **Virtual Private Cloud (VPC)** field, choose a VPC from the dropdown list.

f. For **Subnets** and **VPC security groups**, you can use the defaults or select three subnets of your choice and up to three **VPC security groups** (minimum one).



5. In the **Authentication** section, enter a string that identifies the login name of the primary user in the **Username** field.

   In the **Password** field, enter a unique password that complies with the instructions, then confirm it.



6. In the **Encryption** section, keep the default setting (**Default Key**).

   Optionally, you can enter a AWS KMS key ARN you created. For more information, see Data encryption at rest for Amazon DocumentDB elastic clusters.

> **Important**
>
> Encryption must be enabled for elastic clusters.

7. In the **Backup** section, edit the fields according to your backup requirements. For test purposes, you can retain the default settings.

   a. **Backup retention period**—In the list, choose the number of days to keep automatic backups of this cluster before deleting them.

   b. **Backup window**—Set the daily time and duration during which Amazon DocumentDB is to make backups of this cluster.

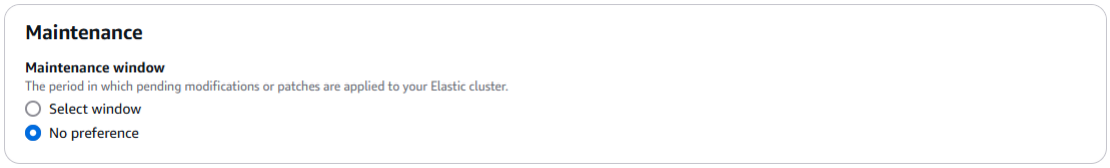
      i. Choose **Select window** if you want to configure the time and duration when backups are created.

         Start time—In the first list, choose the start time hour (UTC) for starting your automatic backups. In the second list, choose the minute of the hour that you want automatic backups to begin.

         Duration—In the list, choose the number of hours to be allocated to creating automatic backups.

      ii. Choose **No preference** if you want Amazon DocumentDB to choose the time and duration when backups are created.

8. In the **Maintenance** section, choose the day, time, and duration when modifications or patches are applied to your cluster. For test purposes, you can retain the default settings.



9. Choose **Create cluster**.

The elastic cluster is now provisioning. This can take up to a few minutes to finish. You can connect to your cluster when the elastic cluster status shows as **Available** in the **Clusters** list.

Using the AWS CLI

To create an elastic cluster using the AWS CLI, use the `create-cluster` operation with the following parameters:

- **--cluster-name**—Required. The current name of the elastic scale cluster as entered during creation or last modified.
- **--shard-capacity**—Required. The number of vCPUs assigned to each shard. Maximum is 64. Allowed values are 2, 4, 8, 16, 32, 64.
- **--shard-count**—Required. The number of shards assigned to the cluster. Maximum is 32.
- **--shard-instance-count**—Optional. The number of replica instances applying to all shards in this cluster. Maximum is 16.
- **--admin-user-name**—Required. The username associated with the admin user.
- **--admin-user-password**—Required. The password associated with the admin user.
- **--auth-type**—Required. The authentication type used to determine where to fetch the password used for accessing the elastic cluster. Valid types are `PLAIN_TEXT` or `SECRET_ARN`.
- **--vpc-security-group-ids**—Optional. Configure a list of EC2 VPC security groups to associate with this cluster.
- **--preferred-maintenance-window**—Optional. Configure the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

  The format is: `ddd:hh24:mi-ddd:hh24:mi`. Valid days (ddd): Mon, Tue, Wed, Thu, Fri, Sat, Sun

  The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Web Services Region, occurring on a random day of the week.

  Minimum 30-minute window.
- **--kms-key-id**—Optional. Configure the KMS key identifier for an encrypted cluster.

  The KMS key identifier is the Amazon Resource Name (ARN) for the AWS KMS encryption key. If you are creating a cluster using the same Amazon Web Services account that owns the KMS encryption key that is used to encrypt the new cluster, you can use the KMS key alias instead of the ARN for the KMS encryption key.

If an encryption key is not specified in KmsKeyId and if the `StorageEncrypted` parameter is true, Amazon DocumentDB uses your default encryption key.

- **--preferred-backup-window**—Optional. The daily preferred time range during which automated backups are created. The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region.
- **--backup-retention-period**—Optional. The number of days for which automated backups are retained. The default value is 1.
- **--storage-encrypted**—Optional. Configues whether the cluster is encrypted or not encrypted.

  `--no-storage-encrypted` specifies that the cluster is not encrypted.
- **--subnet-ids**—Optional. Configure network subnet Ids.

In the following example, replace each *`user input placeholder`* with your own information.

> **Note**
>
> The following examples include creation of a specific KMS key. To use the default KMS key, do not include the `--kms-key-id` parameter.

For Linux, macOS, or Unix:

```
aws docdb-elastic create-cluster \
     --cluster-name sample-cluster-123 \
     --shard-capacity 8 \
     --shard-count 4 \
     --shard-instance-count 3 \
     --auth-type PLAIN_TEXT \
     --admin-user-name testadmin \
     --admin-user-password testPassword \
     --vpc-security-group-ids ec-65f40350 \
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9 \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

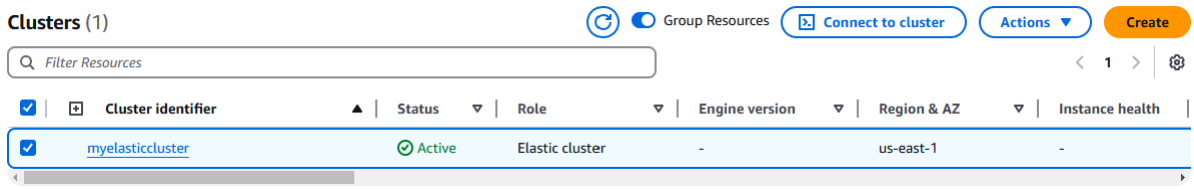
For Windows:

```
aws docdb-elastic create-cluster ^
     --cluster-name sample-cluster-123 ^
     --shard-capacity 8 ^
     --shard-count 4 ^
     --shard-instance-count 3 ^
     --auth-type PLAIN_TEXT ^
     --admin-user-name testadmin ^
     --admin-user-password testPassword ^
     --vpc-security-group-ids ec-65f40350 ^
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9 \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

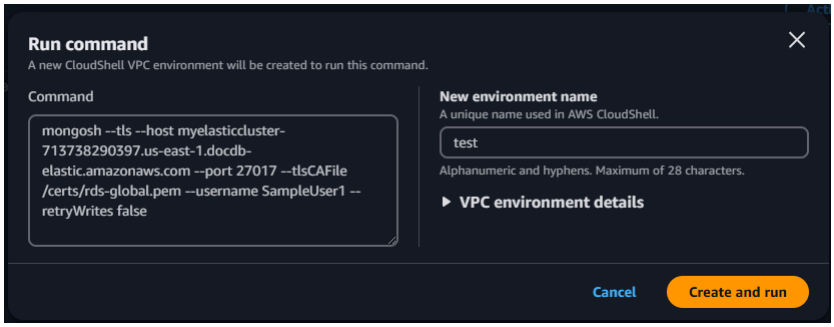
Step 2: Connect to your elastic cluster

Connect to your Amazon DocumentDB elastic cluster using AWS CloudShell.

1. On the Amazon DocumentDB management console, under **Clusters**, locate the elastic cluster you created. Choose your cluster by clicking the check box next to it.



2. Click **Connect to cluster** (which is next to the **Actions** dropdown menu. This button is enabled only after you have clicked the checkbox next to your cluster, and the status of the cluster shows as **Available**. The CloudShell **Run command** screen appears.
3. In the **New environment name** field, enter a unique name, such as "test" and click **Create and run**. VPC environment details are automatically configured for your Amazon DocumentDB database.



4. When prompted, enter the password you created in Step 1: Create an Amazon DocumentDB elastic cluster (sub-step 5).

After you enter your password and your prompt becomes `direct: mongos] <env-name>>`, you are successfully connected to your Amazon DocumentDB cluster

> **Note**
>
> For information about troubleshooting, see Troubleshooting Amazon DocumentDB.

Step 3: Shard your collection, insert and query data

Elastic clusters add support for sharding in Amazon DocumentDB. Now that you are connected to your cluster, you can shard the cluster, insert data and run a few queries.

1. To shard a collection, enter the following:

```
sh.shardCollection("db.Employee1" , { "Employeeid" : "hashed" })
```

2. To insert a single document, enter the following:

```
db.Employee1.insertOne({"Employeeid":1, "Name":"Joe", "LastName": "Bruin", "level":
 1 })
```

The following output is displayed:

```
WriteResult({ "nInserted" : 1 })
```

3. To read the document that you wrote, enter the `findOne()` command (it returns a single document):

```
db.Employee1.findOne()
```

The following output is displayed:

```
{
"_id" : ObjectId("61f344e0594fe1a1685a8151"),
"EmployeeID" : 1,
"Name" : "Joe",
"LastName" : "Bruin",
"level" : 1
}
```

4. To perform a few more queries, consider a gaming profile use case. First, insert a few entries into a collection titled "Employee". Enter the following:

```
db.profiles.insertMany([ { "_id": 1, "name": "Matt", "status": "active", "level":
 12, "score": 202 },
     { "_id": 2, "name": "Frank", "status": "inactive", "level": 2, "score": 9 },
     { "_id": 3, "name": "Karen", "status": "active", "level": 7, "score": 87 },
     { "_id": 4, "name": "Katie", "status": "active", "level": 3, "score": 27 }
])
```

The following output is displayed:

```
{ acknowledged: true,
     insertedIds: {
        '0': ObjectId('679d02cd6b5a0581be78bcbd'),
        '1': ObjectId('679d02cd6b5a0581be78bcbe'),
        '2': ObjectId('679d02cd6b5a0581be78bcbf'),
        '3': ObjectId('679d02cd6b5a0581be78bcc0')
```

```
    }
}
```

5. To return all the documents in the profiles collection, enter the `find()` command:

```
db.Employee.find()
```

The data you entered in step 4 is displayed.

6. To query a single document, include a filter (for example: "Katie"). Enter the following:

```
db.Employee.find({name: "Katie"})
```

The following output is displayed:

```
[

   {

     _id: ObjectId('679d02cd6b5a0581be78bcc0'),

     Employeeid: 4,

     name: 'Katie',

     lastname: 'Schaper',

     level: 3

   }

]
```

7. To find a profile and modify it, enter the `findAndModify` command. In this example, the employee "Matt" is given a higher level of "14":

```
db.Employee.findAndModify({
    query: { "Employeeid" : 1, "name" : "Matt"},
    update: { "Employeeid" : 1, "name" : "Matt", "lastname" : "Winkle", "level" :
 14 }
})
```

The following output is displayed (note that the level has not changed yet):

```
{

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 12

}
```

8. To verify the level increase, enter the following query:

```
db.Employee.find({name: "Matt"})
```

The following output is displayed:

```
[
   {

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 14

   }
]
```

Step 4: Explore

Congratulations! You have successfully completed the Get started procedure for Amazon DocumentDB elastic clusters.

What's next? Learn how to fully leverage this database with some of its popular features:

- Amazon DocumentDB elastic cluster best practices
- Managing Amazon DocumentDB elastic clusters

> **Note**
>
> The elastic cluster you created from this get started procedure will continue to accrue costs unless you delete it. For directions, see Deleting an elastic cluster.

Amazon DocumentDB elastic cluster best practices

Learn best practices for working with Amazon DocumentDB elastic clusters. All best practices for instance-based Amazon DocumentDB clusters also apply for elastic clusters. This section is continually updated as new best practices are identified.

Topics

- Choosing shard keys
- Connection management
- Unsharded collections
- Scaling elastic clusters
- Monitoring elastic clusters

Choosing shard keys

The following list describes guidelines for creating shard keys.

- Use an evenly distributed hash key to distribute your data across all of the shards in your cluster (avoid hot keys).
- Use your shard key in all read/update/delete requests to avoid scatter gather queries.

- Avoid nested shard keys when doing read/update/delete operations.
- When making batch operations, set `ordered` to false so all shards can run in parallel and improve latencies.

Connection management

The following list describes guidelines for managing your connections to your database.

- Monitor your connection counts and how frequently new connections are opened closed.
- Distribute your connections across all of the subnets in your application's configuration. If your cluster is configured in multiple subnets but you only utilize a subset of the subnets, you may be bottlenecked on your maximum connections.

Unsharded collections

The following describes a guideline for unsharded collections.

- When working with unsharded collections, to distribute load, try keeping highly utilized unsharded collections on different databases. Amazon DocumentDB elastic clusters place databases across different shards and co-locates unsharded collections for the same database on the same shard.

Scaling elastic clusters

The following list describes guidelines for scaling your elastic clusters.

- Scaling operations may cause a brief period of intermittent database and network errors. When possible, avoid scaling during peak hours. Try to scale during maintenance windows.
- Scaling shard capacity up and down (changing vCPU count per shard) to increase compute is preferred over increasing or decreasing the shard-count as it is faster and has a shorter duration of intermittent database and network errors.
- When anticipating growth, favor increasing the shard count instead scaling the shard capacity. This enables you to scale your cluster by increasing the shard capacity for scenarios where you need to quickly scale.
- Monitor your client-side retry policies and retry with exponential backoff and jitter to avoid overloading your database when getting errors while scaling.

Monitoring elastic clusters

The following list describes guidelines for monitoring your elastic clusters.

- Track the peak-to-average ratio of your per-shard metrics to determine if you are driving un-even traffic (have a hot-key/hot-spot). Key metrics to track peak-to-average ratios are:
  - `PrimaryInstanceCPUUtilization`
    - This can be monitored at the per-shard level.
    - At the cluster level you can monitor the average to p99 skew.
  - `PrimaryInstanceFreeableMemory`
    - This can be monitored at the per-shard level.
    - At the cluster level you can monitor the average to p99 skew.
  - `DatabaseCursorsMax`
    - This should be monitored at the per-shard level to determine skew.
  - `Documents-Inserted/Updated/Returned/Deleted`
    - This should be monitored at the per-shard level to determine skew.

Managing Amazon DocumentDB elastic clusters

To manage an Amazon DocumentDB elastic cluster, you must have an IAM policy with the appropriate Amazon DocumentDB control plane permissions. These permissions allow you to create, modify, and delete clusters. The Amazon DocumentDBFullAccess policy provides all the required permissions for administering an Amazon DocumentDB elastic cluster.

The following topics show how to perform various tasks when working with Amazon DocumentDB elastic clusters.

Topics

- Modifying elastic cluster configurations
- Monitoring an elastic cluster
- Deleting an elastic cluster
- Managing elastic cluster snapshots
- Stopping and starting an Amazon DocumentDB elastic cluster
- Maintaining Amazon DocumentDB elastic clusters

Modifying elastic cluster configurations

In this section we explain how to modify elastic cluster, using either the AWS Management Console or AWS CLI with the following instructions.

A primary use of modifying the cluster is to scale shards by increasing or decreasing the shard count and/or shard compute capacity.

Using the AWS Management Console

To modify an elastic cluster configuration using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Clusters**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. Choose the name of the cluster you want to modify in the **Cluster identifier** column.
4. Choose **Modify**.
5. Edit the fields you want changed and then select **Modify cluster**.

Configuration

Cluster identifier

SampleCluster

Shard count

Number of shards the Elastic Cluster will use.

2

Shard instance count

Number of instances for each shard. All instances will have the same shard capacity.

2

Shard capacity

vCPU capacity of each shard.

2

Maintenance

Maintenance window

The period in which pending modifications or patches are applied to your Elastic cluster.

- ○ Select window
- ◉ No preference

Authentication

Username

SampleUser

New password

Confirm new password

Password must be at least eight characters long and cannot contain a / (slash), " (double quote) or @ (at symbol).

Network settings

Subnets

Select either 0 or 2-6 subnets

subnet-0b2962f92a0f5a8fb ✕ subnet-08c6d849efd4dfe96 ✕

VPC security groups

> **Note**
>
> Alternatively, you can access the **Modify cluster** dialog by going to the **Clusters** page, checking the box next to your cluster, choosing **Actions**, then **Modify**.

Using the AWS CLI

To modify an elastic cluster configuration using the AWS CLI, use the `update-cluster` operation with the following parameters:

- **`--cluster-arn`**—Required. The ARN identifier of the cluster that you want to modify.
- **`--shard-capacity`**—Optional. The number of vCPUs assigned to each shard. Maximum is 64. Allowed values are 2, 4, 8, 16, 32, 64.
- **`--shard-count`**—Optional. The number of shards assigned to the cluster. Maximum is 32.
- **`--shard-instance`**-count—Optional. The number of replica instances applying to all shards in this cluster. Maximum is 16.
- **`--auth-type`**—Optional. The authentication type used to determine where to fetch the password used for accessing the elastic cluster. Valid types are `PLAIN_TEXT` or `SECRET_ARN`.
- **`--admin-user-password`**—Optional. The password associated with the admin user.
- **`--vpc-security-group-ids`**—Optional. Configure a list of Amazon EC2 and Amazon Virtual Private Cloud (VPC) security groups to associate with this cluster.
- **`--preferred-maintenance-window`**—Optional. Configure the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC)

  The format is: `ddd:hh24:mi-ddd:hh24:mi`. Valid days (ddd): Mon, Tue, Wed, Thu, Fri, Sat, Sun

  The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Web Services Region, occurring on a random day of the week.

  Minimum 30-minute window.
- **`--subnet-ids`**—Optional. Configure network subnet Ids.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic update-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
    --shard-capacity 8 \
    --shard-count 4 \
    --shard-instance-count 3 \
    --admin-user-password testPassword \
    --vpc-security-group-ids ec-65f40350 \
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

For Windows:

```
aws docdb-elastic update-cluster ^
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
    --shard-capacity 8 ^
    --shard-count 4 ^
    --shard-instance-count 3 ^
    --admin-user-password testPassword ^
    --vpc-security-group-ids ec-65f40350 ^
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

To monitor the status of the elastic cluster after your modification, see Monitoring an elastic cluster.

Monitoring an elastic cluster

In this section, we explain how to monitor your elastic cluster, using either the AWS Management Console or AWS CLI with the following instructions.

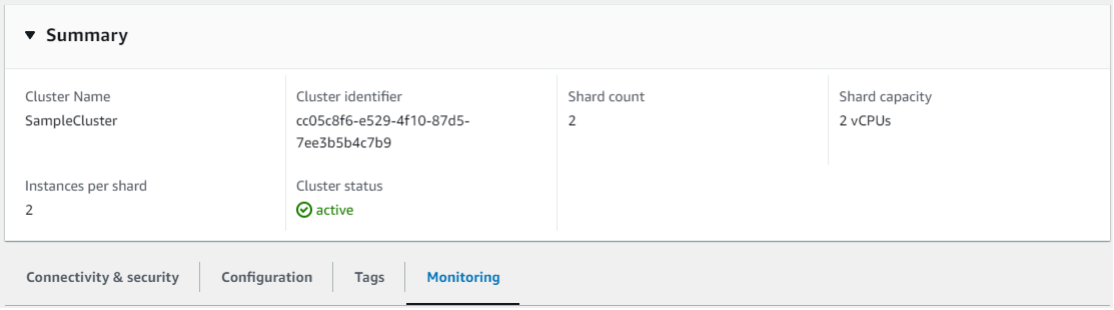
Using the AWS Management Console

To monitor an elastic cluster configuration using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. Choose the name of the cluster you want to monitor in the **Cluster identifier** column.
4. Choose the **Monitoring** tab.



A number of charts from Amazon CloudWatch are displayed for the following monitoring categories:

- Resource Utilization
- Throughput
- Operations
- System

You can also access Amazon CloudWatch through the AWS Management Console to set up your own monitoring environment for your elastic clusters.

Using the AWS CLI

To monitor a specific elastic cluster configuration using the AWS CLI, use the `get-cluster` operation with the following parameters:

- **`--cluster-arn`**—Required. The ARN identifier of the cluster for which you want information.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic get-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-west-2:123456789012:cluster:/68ffcdf8-
e3af-40a3-91e4-24736f2dacc9
```

For Windows:

```
aws docdb-elastic get-cluster ^
    --cluster-arn arn:aws:docdb:-elastic:us-west-2:123456789012:cluster:/68ffcdf8-
e3af-40a3-91e4-24736f2dacc9
```

The output from this operation looks something like the following:

```
"cluster": {
        ...
        "clusterArn": "arn:aws:docdb-elastic:us-
west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9",
        "clusterEndpoint": "stretch-11-477568257630.us-east-1.docdb-
elastic.amazonaws.com",
        "readerEndpoint": "stretch-11-477568257630-ro.us-east-1.docdb-
elastic.amazonaws.com",
        "clusterName": "stretch-11",
        "shardCapacity": 2,
        "shardCount": 3,
        "shardInstanceCount: 5,
        "status": "ACTIVE",
        ...
 }
```

For more information, see `DescribeClusterSnapshot` in the Amazon DocumentDB Resource Management API Reference.

To view the details of all elastic clusters using the AWS CLI, use the `list-clusters` operation with the following parameters:

- **`--next-token`**—Optional. If the number of items output (`--max-results`) is fewer than the total number of items returned by the underlying API calls, the output includes a `NextToken` that you can pass to a subsequent command to retrieve the next set of items.

- **`--max-results`**—Optional. The total number of items to return in the command's output. If more results exist than the specified `max-results` value, a pagination token (`next-token`) is included in the response so that the remaining results can be retrieved.
  - Default: 100
  - Minimum 20, maximum 100

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic list-clusters \
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== \
    --max-results 2
```

For Windows:

```
aws docdb-elastic list-clusters ^
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== ^
    --max-results 2
```

The output from this operation looks something like the following:

```
{
   "Clusters": [
      {
         "ClusterIdentifier":"mycluster-1",
         "ClusterArn":"arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster"
         "Status":"available",
         "ClusterEndpoint":"sample-cluster.sharded-cluster-corcjozrlsfc.us-
west-2.docdb.amazonaws.com"
       }
       {
         "ClusterIdentifier":"mycluster-2",
         "ClusterArn":"arn:aws:docdb:us-west-2:987654321098:sharded-cluster:sample-
cluster"
         "Status":"available",
         "ClusterEndpoint":"sample-cluster2.sharded-cluster-corcjozrlsfc.us-
west-2.docdb.amazonaws.com"
```

```
        }
    ]
}
```

Deleting an elastic cluster

In this section we explain how to delete an elastic cluster, using either the AWS Management Console or AWS CLI with the following instructions.

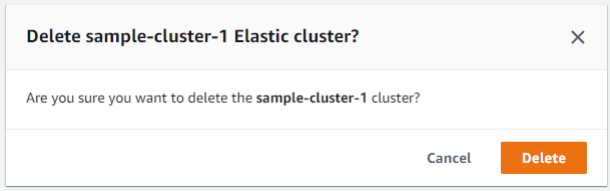
Using the AWS Management Console

To delete an elastic cluster configuration using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. In the cluster list table, select the check box to the left of the cluster name you want to delete and then choose **Actions**. From the dropdown menu, choose **Delete**.
4. In the **Delete "cluster-name" elastic cluster?** dialog box, choose **Delete**.



It takes several minutes for the cluster to be deleted. To monitor the status of the cluster, see Monitoring an Amazon DocumentDB Cluster's Status.

Using the AWS CLI

To delete an elastic cluster using the AWS CLI, use the `delete-cluster` operation with the following parameters::

- **`--cluster-arn`**—Required. The ARN identifier of the cluster that you want to delete.

- **`--no-skip-final-backup`**—Optional. If you want a final backup, you must include this parameter with a name for the final backup. You must include either `--final-backup-identifier` or `--skip-final-backup`.
- **`--skip-final-backup`**—Optional. Use this parameter only if you don't want to take a final backup before deleting your cluster. The default setting is to take a final snapshot.

The following AWS CLI code examples delete a cluster with an ARN of arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster with a final backup.

In the following example, replace each *`user input placeholder`* with your own information..

For Linux, macOS, or Unix:

```
aws docdb-elastic delete-cluster \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster \
    --no-skip-final-backup \
    --final-backup-identifier finalArnBU-arn:aws:docdb:us-
west-2:123456789012:sharded-cluster:sample-cluster
```

For Windows:

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --no-skip-final-backup ^
    --final-backup-identifier finalArnBU-arn:aws:docdb:us-
west-2:123456789012:sharded-cluster:sample-cluster
```

The following AWS CLI code examples delete a cluster with an ARN of arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster without taking a final backup.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic delete-cluster \
```

```
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster \
    --skip-final-backup \
```

For Windows:

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --skip-final-backup ^
```

The output of the `delete-cluster` operation is a display of the cluster you are deleting.

It takes several minutes for the cluster to be deleted. To monitor the status of the cluster, see Monitoring an Amazon DocumentDB Cluster's Status.

Managing elastic cluster snapshots

Manual snapshots can be taken after an elastic cluster has been created. Automated backups are created the moment the elastic cluster snapshot is created.

> **Note**
>
> Your elastic cluster must be in the `Available` state for a manual snapshot to be taken.

This section explains how you can create, view, restore from, and delete elastic cluster snapshots.

The following topics show how to perform various tasks when working with Amazon DocumentDB elastic cluster snapshots.

Topics

- Creating a manual elastic cluster snapshot
- Viewing an elastic cluster snapshot
- Restoring an elastic cluster from a snapshot
- Copying an elastic cluster snapshot
- Deleting an elastic cluster snapshot

- Managing an elastic cluster snapshot automatic backup

Creating a manual elastic cluster snapshot

In this section we explain how to create a manual elastic cluster snapshot, using either the AWS Management Console or AWS CLI with the following instructions.

Using the AWS Management Console

To create a manual elastic cluster snapshot using the AWS Management Console:

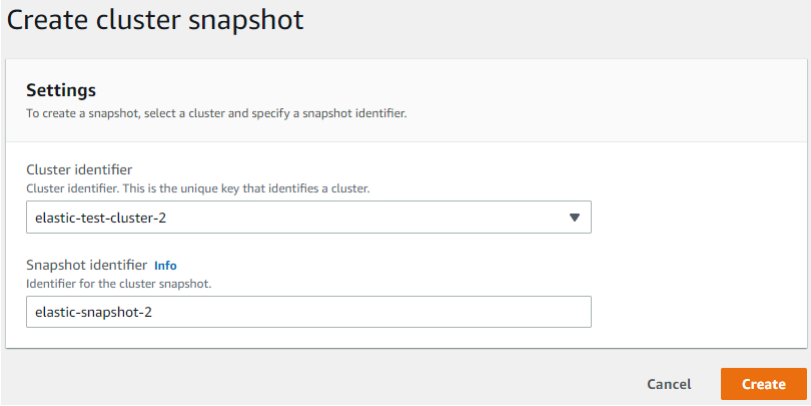
1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Snapshots**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. On the **Snapshots** page, choose **Create**.
4. On the **Create cluster snapshot** page, in the **Cluster identifier** field, choose your elastic cluster from the drop down list.

   In the **Snapshot identifier** field, enter a unique identifier for your elastic cluster.

   Choose **Create**.



> **Note**
>
> Alternatively, you can access the **Create cluster snapshot** dialog by going to the **Clusters** page, checking the box next to your cluster, then choosing **Actions**, then **Take snapshot**.

Your elastic cluster snapshot is now provisioning. This can take up to a few minutes to finish. You can view and restore from your snapshot when the status shows as `Available` in the **Snapshots** list.

Using the AWS CLI

To create a manual elastic cluster snapshot using the AWS CLI, use the `create-cluster-snapshot` operation with the following parameters:

- **`--snapshot-name`**—Required. The name of the cluster snapshot you want to create.
- **`--cluster-arn`**—Required. The ARN identifier of the cluster of which you want to create a snapshot.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic create-cluster-snapshot \
    --snapshot-name sample-snapshot-1 \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

For Windows:

```
aws docdb-elastic create-cluster-snapshot ^
    --snapshot-name sample-snapshot-1 ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

Viewing an elastic cluster snapshot

In this section we explain how to view elastic cluster snapshot information, using either the AWS Management Console or AWS CLI with the following instructions.

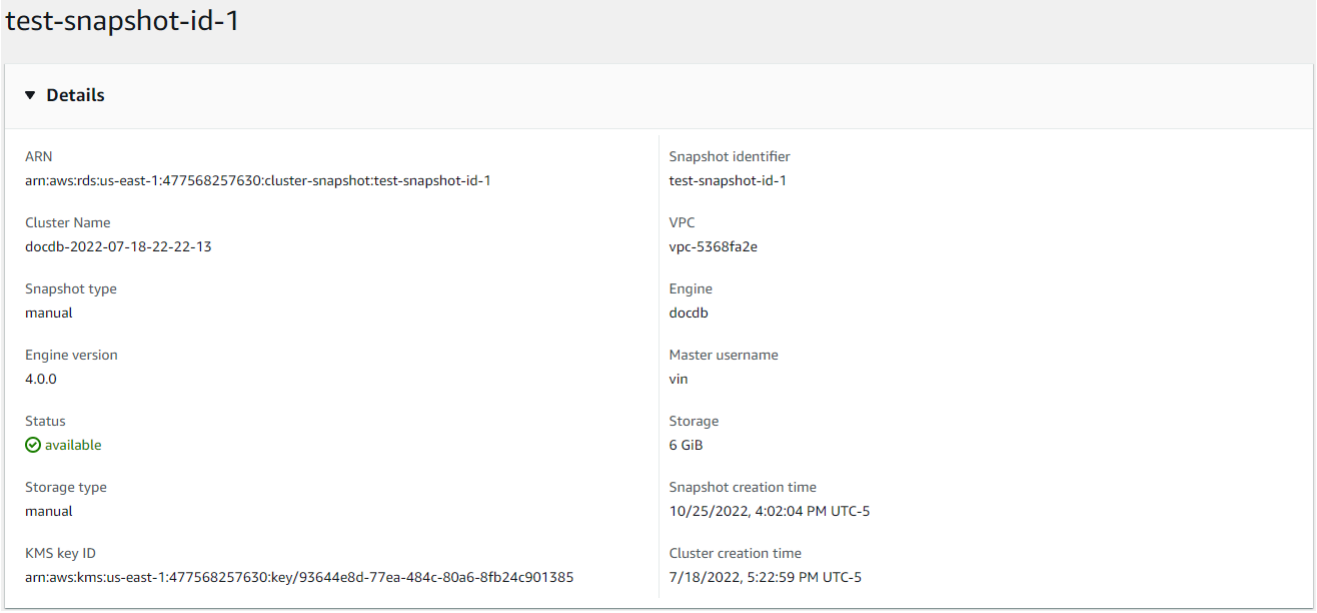
Using the AWS Management Console

To view information about a specific elastic cluster snapshot using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Snapshots**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. On the **Snapshots** page, choose your snapshot from the list by clicking on the name in **Snapshot identifier** column.
4. View your snapshot's information in **Details**.



Using the AWS CLI

To view information about a specific elastic cluster snapshot using the AWS CLI, use the `get-cluster-snapshot` operation with the following parameters:

- **`--snapshot-arn`**—Required. The ARN identifier of the snapshot for which you want information.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

For Windows:

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

To view information about a specific elastic cluster snapshot using the AWS CLI, use the `get-cluster-snapshot` operation with the following parameters:

- **`--snapshot-arn`**—Required. The ARN identifier of the snapshot for which you want information.

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

For Windows:

```
aws docdb-elastic get-cluster-snapshot ^
```

```
    --snapshot-arn sampleResourceName
```

To view information about all elastic cluster snapshots using the AWS CLI, use the `list-cluster-snapshots` operation with the following parameters:

- **`--snapshot-type`**—Optional. The type of cluster snapshots to be returned. You can specify one of the following values:
  - `automated` - Return all cluster snapshots that Amazon DocumentDB has automatically created for your AWS account.
  - `manual` - Return all cluster snapshots that you have manually created for your AWS account.
  - `shared` - Return all manual cluster snapshots that have been shared to your AWS account.
  - `public` - Return all cluster snapshots that have been marked as public.
- **`--next-token`**—Optional. An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the this token, up to the value specified by `max-results`.
- **`--max-results`**—Optional. The maximum number of results to include in the response. If more results exist than the specified `max-results` value, a pagination token (`next-token`) is included in the response so that the remaining results can be retrieved.
  - Default: 100
  - Minimum 20, maximum 100

In the following example, replace each *`user input placeholder`* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic list-cluster-snapshots \
    --snapshot-type  value \
    --next-token  value \
    --max-results 50
```

For Windows:

```
aws docdb-elastic list-cluster-snapshots ^
    --snapshot-type  value ^
```

```
    --next-token  value ^
    --max-results 50
```

Restoring an elastic cluster from a snapshot

In this section we explain how to restore an elastic cluster from a snapshot, using either the AWS Management Console or AWS CLI with the following instructions.

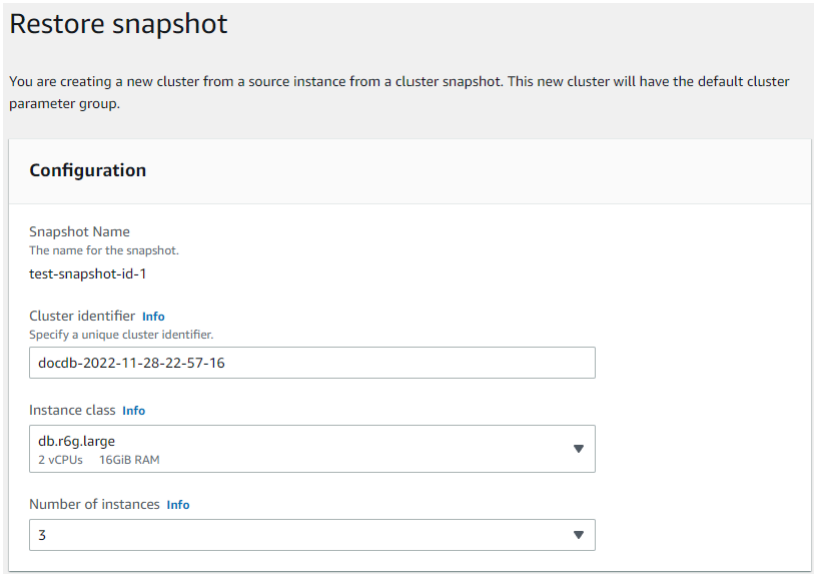
Using the AWS Management Console

To restore an elastic cluster from a snapshot using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Snapshots**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. Choose the button to the left of the snapshot, that you want to use to restore a cluster, in the **Snapshot identifier** column.
4. Choose **Actions**, then **Restore**.



5. On the **Restore snapshot** page, enter a name for the new cluster in the **Cluster identifier** field.

> **Note**
>
> For any manual snapshot restoration, you must create a new cluster.

6. In the **Virtual Private Cloud (VPC)** field, choose a VPC from the drop-down list.
7. For **Subnets** and **VPC security groups**, you can use the defaults or select three subnets of your choice and upto three VPC security groups (minimum one).
8. If you are satisfied with the cluster configuration, choose **Restore cluster** and wait while your cluster is restored.

Using the AWS CLI

To restore an elastic cluster from a snapshot using the AWS CLI, use the `restore-cluster-from-snapshot` operation with the following parameters:

- **`--cluster-name`**—Required. The current name of the elastic cluster as entered during creation or last modified.
- **`--snapshot-arn`**—Required. The ARN identifier of the snapshot being used to restore the cluster.
- **`--vpc-security-group-ids`**—Optional. One or more Amazon EC2 and Amazon Virtual Private Cloud (VPC) security groups to associate with the cluster.
- **`--kms-key-id`**—Optional. Configure the KMS key identifier for an encrypted cluster.

  The KMS key identifier is the Amazon Resource Name (ARN) for the AWS KMS encryption key. If you are creating a cluster using the same Amazon Web Services account that owns the KMS encryption key that is used to encrypt the new cluster, you can use the KMS key alias instead of the ARN for the KMS encryption key.

  If an encryption key is not specified in KmsKeyId and if the `StorageEncrypted` parameter is true, Amazon DocumentDB uses your default encryption key.
- **`--subnet-ids`**—Optional. Network subnet Ids.

In the following example, replace each *`user input placeholder`* with your own information..

For Linux, macOS, or Unix:

```
aws docdb-elastic restore-cluster-from-snapshot \
     --cluster-name elastic-sample-cluster \
     --snapshot-arn sampleResourceName \
     --vpc-security-group-ids value ec-65f40350 \
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

For Windows:

```
aws docdb-elastic restore-cluster-from-snapshot ^
     --cluster-name elastic-sample-cluster ^
     --snapshot-arn sampleResourceName ^
     --vpc-security-group-ids value ec-65f40350 ^
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

Copying an elastic cluster snapshot

In Amazon DocumentDB, you can copy manual and automatic elastic cluster snapshots within the same region and within the same account. In this section we explain how to copy an elastic cluster snapshot, using either the AWS Management Console or AWS CLI.

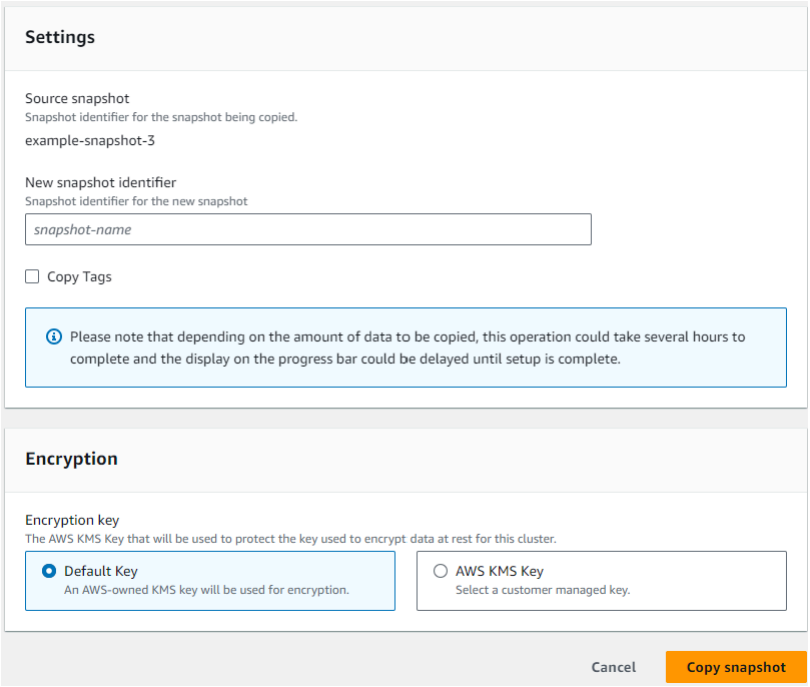
Using the AWS Management Console

To copy an elastic cluster snapshot using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Snapshots**.

   > **Tip**
   >
   > If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. Choose the button to the left of the snapshot, that you want to copy, in the **Snapshot identifier** column.
4. Choose **Actions**, then **Copy**.



5. For **New snapshot identifier**, enter the new snapshot's name.
6. For **Copy Tags**, check the box if you want to copy all tags from the source elastic cluster snapshot to the target elastic cluster snapshot.
7. For **Encryption**, choose either a default AWS KMS key or a KMS key of your choosing. The second option allows you to select an existing KMS key that you already created or allows you to create a new one.
8. Choose **Copy snapshot** when complete.

Using the AWS CLI

To copy an elastic cluster snapshot using the AWS CLI, use the `copy-cluster-snapshot` operation with the following parameters:

- **`--source-db-cluster-snapshot-identifier`**—Required. The identifier of the existing elastic cluster snapshot being copied. The elastic cluster snapshot must exist and be in the available state. This parameter is not case sensitive.
- **`--target-db-cluster-snapshot-identifier`**—Required. The identifier of the new elastic cluster snapshot to create from the existing cluster snapshot. This parameter is not case sensitive.

  Target snapshot name constraints:

- Cannot be the name of an existing snapshot.
- Length is [1—63] letters, numbers, or hyphens.
- First character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

In the following example, replace each *user input placeholder* with your own information.

For Linux, macOS, or Unix:

```
aws docdb-elastic copy-cluster-snapshot \
     --source-cluster-snapshot-arn <sample ARN> \
     --target-cluster-snapshot-name my-target-copied-snapshot
```

For Windows:

```
aws docdb-elastic copy-cluster-snapshot ^
     --source-cluster-snapshot-arn <sample ARN> ^
     --target-cluster-snapshot-name my-target-copied-snapshot
```

Deleting an elastic cluster snapshot

In this section we explain how to delete an elastic cluster snapshot, using either the AWS Management Console or AWS CLI.

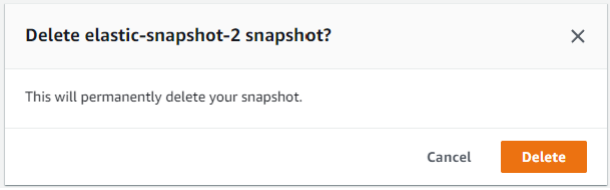
Using the AWS Management Console

To restore an elastic cluster from a snapshot using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Snapshots**.

> **Tip**
> If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. Choose the button to the left of the snapshot, that you want to use to restore a cluster, in the **Snapshot identifier** column.
4. Choose **Actions**, then **Delete**.



5. On the **Delete "snapshot-name" snapshot** dialog, choose **Delete**.

Using the AWS CLI

To delete an elastic cluster snapshot using the AWS CLI, use the `delete-cluster-snapshot` operation with the following parameters:

- **`--snapshot-arn`**—Required. The ARN identifier of the snapshot being used to restore the cluster.

In the following example, replace each *`user input placeholder`* with your own information..

For Linux, macOS, or Unix:

```
aws docdb-elastic delete-cluster-snapshot \
     --snapshot-arn sampleResourceName
```

For Windows:

```
aws docdb-elastic delete-cluster-snapshot ^
     --snapshot-arn sampleResourceName
```

Managing an elastic cluster snapshot automatic backup

Amazon DocumentDB takes daily snapshots of your elastic clusters. You can specify the preferred backup window and the backup retention period in a new or existing elastic cluster snapshot configuration. In this section we explain how to set automatic backup parameters in an elastic cluster snapshot, using either the AWS Management Console or AWS CLI.

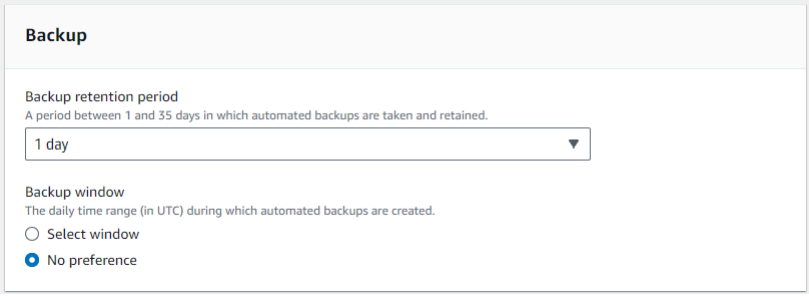
Using the AWS Management Console

To set an automatic backup for a new elastic cluster snapshot using the AWS Management Console:

1. Sign into the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon in the upper-left corner of the navigation pane.

3. Choose the button to the left of the cluster, that you want to change the backup settings for, in the **Cluster identifier** column.
4. Choose **Actions**, then **Modify**.
5. In the **Backup** section, edit the fields according to your backup requirements.



   a. **Backup retention period**—In the list, choose the number of days to keep automatic backups of this cluster before deleting them.
   b. **Backup window**—Set the daily time and duration during which Amazon DocumentDB is to make backups of this cluster.

      i. Choose **Select window** if you want to configure the time and duration when backups are created.

         Start time—In the first list, choose the start time hour (UTC) for starting your automatic backups. In the second list, choose the minute of the hour that you want automatic backups to begin.

         Duration—In the list, choose the number of hours to be allocated to creating automatic backups.

ii. Choose **No preference** if you want Amazon DocumentDB to choose the time and duration when backups are created.

6. Choose **Modify cluster** when complete.

Using the AWS CLI

To set an automatic backup for a new elastic cluster snapshot using the AWS CLI, use the `create-cluster-snapshot` operation with the following parameters:

- **`--preferred-backup-window`**—Optional. The daily preferred time range during which automated backups are created. The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region.

  Constraints:

  - Must be in the format `hh24:mi-hh24:mi`.
  - Must be in Universal Coordinated Time (UTC).
  - Must not conflict with the preferred maintenance window.
  - Must be at least 30 minutes.
- **`--backup-retention-period`**—Optional. The number of days for which automated backups are retained. The default value is 1.

  Constraints:

  - Must specify a minimum value of 1.
  - Range is from 1 to 35.

> **Note**
>
> Automated backups are only be taken when the cluster is in an 'active' state.

> **Note**
>
> You can also modify the `preferred-backup-window` and `backup-retention-period` parameters of an existing elastic cluster using the `aws docdb-elastic update-cluster` command.

In the following example, replace each *user input placeholder* with your own information.

The following `create-cluster` example creates the Amazon DocumentDB elastic cluster *sample-cluster* with the retention period for automatic backups of *7* days, and a preferred backup window of *18:00-18:30 UTC*.

For Linux, macOS, or Unix:

```
aws docdb-elastic create-cluster \
     --cluster-name sample-cluster \
     --shard-capacity 2 \
     --shard-count 2 \
     --admin-user-name SampleAdmin \
     --auth-type PLAIN_TEXT \
     --admin-user-password SamplePass123! \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

For Windows:

```
aws docdb-elastic create-cluster ^
     --cluster-name sample-cluster ^
     --shard-capacity 2 ^
     --shard-count 2 ^
     --admin-user-name SampleAdmin ^
     --auth-type PLAIN_TEXT ^
     --admin-user-password SamplePass123! ^
     --preferred-backup-window 18:00-18:30 ^
     --backup-retention-period 7
```

Stopping and starting an Amazon DocumentDB elastic cluster

Stopping and starting Amazon DocumentDB elastic clusters can help you manage costs for development and test environments. Instead of creating and deleting elastic clusters each time you use Amazon DocumentDB, you can temporarily stop your cluster when it isn't needed. You can then start it again when you resume your testing.

Topics

- Overview of stopping and starting an elastic cluster

- Operations you can perform on a stopped elastic cluster

Overview of stopping and starting an elastic cluster

During periods where you don't need an Amazon DocumentDB elastic cluster, you can stop the cluster. You can then start the cluster again anytime you need to use it. Starting and stopping simplifies the setup and teardown processes for elastic clusters that are used for development, testing, or similar activities that don't require continuous availability. You can stop and start an elastic cluster using the AWS Management Console or the AWS CLI with a single action.

While your elastic cluster is stopped, the cluster storage volume remains unchanged. You are charged only for storage, manual snapshots, and automated backup storage within your specified retention window. Amazon DocumentDB automatically starts your elastic cluster after seven days so that it doesn't fall behind any required maintenance updates. When your cluster starts after seven days, you will begin to be charged for the use of the elastic cluster again. While your cluster is stopped, you can't query your storage volume because querying requires that the cluster is in the available state.

When an Amazon DocumentDB elastic cluster is stopped, the cluster cannot be modified in any way. This includes deleting the cluster.

Using the AWS Management Console

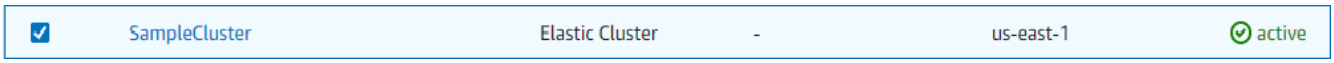
The following procedure shows you how to stop an elastic cluster in the available state, or start a stopped elastic cluster.

To stop or start an Amazon DocumentDB elastic cluster

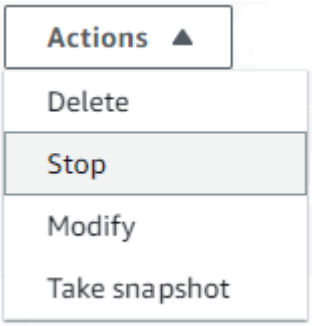
1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

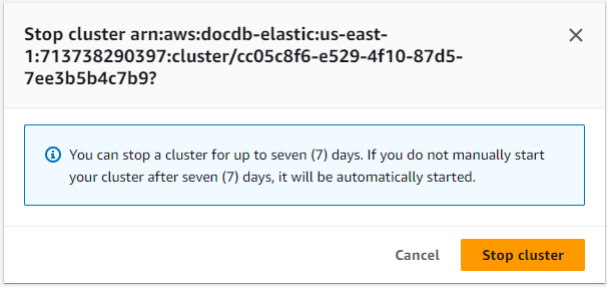
3. In the list of clusters, choose the button to the left of the name of the cluster that you want to stop or start.



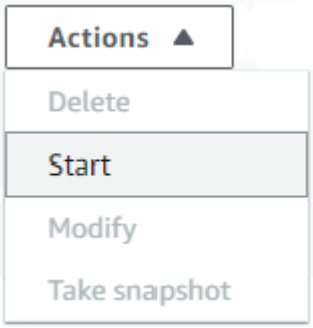
4. Choose **Actions**, and then choose the action that you want to perform on the cluster.
   - If you want to stop the cluster and the cluster is available:
     a. Choose **Stop**.



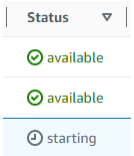
     b. On the confirmation dialog, confirm that you want to stop the elastic cluster by choosing **Stop cluster**, or to keep the cluster running, choose **Cancel**.



   - If you want to start the cluster, and the cluster is stopped, choose **Start**.



5. Monitor the status of the elastic cluster. If you started the cluster, you can resume using the cluster when the cluster is *available*. For more information, see Determining a cluster's status.



Using the AWS CLI

The following code examples show you how to stop an elastic cluster in the active or available state, or start a stopped elastic cluster.

To stop an elastic cluster using the AWS CLI, use the `stop-cluster` operation. To start a stopped cluster, use the `start-cluster` operation. Both operations use the `--cluster-arn` parameter.

Parameter:

- **`--cluster-arn`**—Required. The ARN identifier of the elastic cluster that you want to stop or start.

Example — To stop an elastic cluster using the AWS CLI

In the following example, replace each *`user input placeholder`* with your own information.

The following code stops the elastic cluster with an ARN of `arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2`.

> **Note**
>
> The elastic cluster must be in the active or available state.

For Linux, macOS, or Unix:

```
aws docdb-elastic stop-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

For Windows:

```
aws docdb-elastic stop-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Example — To start an elastic cluster using the AWS CLI

In the following example, replace each *`user input placeholder`* with your own information.

The following code starts the elastic cluster with an ARN of `arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2`.

> **Note**
>
> The elastic cluster must currently be stopped.

For Linux, macOS, or Unix:

```
aws docdb-elastic start-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

For Windows:

```
aws docdb-elastic start-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Operations you can perform on a stopped elastic cluster

You can't modify the configuration of an Amazon DocumentDB elastic cluster while the cluster is stopped. You must start the cluster before performing any such administrative actions.

Amazon DocumentDB applies any scheduled maintenance to your stopped elastic cluster only after it's started again. After seven days, Amazon DocumentDB automatically starts a stopped elastic cluster so that it doesn't fall too far behind in its maintenance status. When the elastic cluster restarts, you will begin to be charged for the shards in the cluster again.

While an elastic cluster is stopped, Amazon DocumentDB does not perform any automated backups nor does it extend the backup retention period.

Maintaining Amazon DocumentDB elastic clusters

Topics

- Viewing pending elastic cluster maintenance actions
- Elastic cluster engine updates
- Elastic cluster operating system updates

Periodically, Amazon DocumentDB performs maintenance on Amazon DocumentDB elastic cluster resources. Maintenance most often involves updates to the database engine (elastic cluster maintenance) or the elastic cluster's underlying operating system (OS updates). Database engine updates are required patches and include security fixes, bug fixes, and enhancements to the database engine. While most operating system patches are optional, if you don't apply them for a while, the patch may be required and auto applied to maintain your security posture. So, we recommend that you apply operating system updates to your Amazon DocumentDB elastic clusters as soon as they are available.

Database engine patches require that you take your Amazon DocumentDB elastic clusters offline for a short time. Once available, these patches are automatically scheduled to apply during an upcoming scheduled maintenance window of your Amazon DocumentDB elastic cluster.

Elastic clusters have their own respective maintenance windows. Elastic cluster modifications that you have chosen not to apply immediately, are applied during the maintenance window. By default, when you create an elastic cluster, Amazon DocumentDB assigns a maintenance window for your elastic cluster. You can choose the maintenance window when creating an elastic cluster. You can also modify the maintenance windows at any time to fit your business schedules or practices. It is generally advised to choose maintenance windows that minimize the impact of the maintenance on your application (for example, on evenings or weekends).

Viewing pending elastic cluster maintenance actions

You can view whether a maintenance update is available for your elastic cluster by using the AWS CLI.

If an update is available, you can do one of the following:

- Defer a maintenance action that is currently scheduled for next maintenance window (for OS patches only).
- Apply the maintenance actions immediately.
- Schedule the maintenance actions to start during your next maintenance window.
- Schedule the maintenance actions to start during your selected apply-on window.

The maintenance window determines when pending operations start, but it does not limit the total execution time of these operations.

Use the following AWS CLI operation to determine what maintenance actions are pending. List all pending maintenance actions:

```
aws docdb-elastic list-pending-maintenance-actions
```

Output from this operation looks something like the following (JSON format):

```
{
'ResourcePendingMaintenanceActions': [
    {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
    },
],
'NextToken': 'string'
}
```

Get pending maintenance action (if any) on a given `resourceArn`:

```
aws docdb-elastic get-pending-maintenance-action --resource-arn string-arn
```

Output from this operation looks something like the following (JSON format).

```
{
    'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            }
        ]
    }
}
```

Parameters:

- `ResourceArn`—The Amazon DocumentDB Amazon Resource Name (ARN) of the resource to which the pending maintenance action applies.
- `Action`—The pending maintenance action being applied to the resource.

  Valid values:

  - `ENGINE_UPDATE`
  - `ENGINE_UPGRADE`
  - `SECURITY_UPDATE`
  - `OS_UPDATE`
  - `MASTER_USER_PASSWORD_UPDATE`
- `AutoAppliedAfterDate`—First maintenance window after this date. `NEXT_MAINTENANCE` `OPT_IN` is ignored in this case.
- `ForcedApplyDate`—Applied regardless of maintenance window. `IMMEDIATE` `OPT_IN` is ignored in this case.
- `OptInStatus`—A value that specifies the type of opt-in request, or undoes an opt-in request. An opt-in request of type `IMMEDIATE` can't be undone.

  Valid values:

- `IMMEDIATE`—Apply the maintenance action immediately.
- `NEXT_MAINTENANCE`—Apply the maintenance action during the next maintenance window for the resource.
- `APPLY_ON`—Apply the maintenance action on specified apply date regardless of next maintenance window for the resource.
- `UNDO_OPT_IN`—Cancel any existing `NEXT_MAINTENANCE` or `APPLY_ON` opt in requests.
- `CurrentApplyDate`—Displayed if opt-in-type is `APPLY_ON`.
- `Description`—An option description for the maintenance action.

Elastic cluster engine updates

With Amazon DocumentDB, you can choose when to apply maintenance operations. You can decide when Amazon DocumentDB applies updates using the AWS CLI.

Apply pending maintenance actions:

```
aws docdb-elastic apply-pending-maintenance-action
--resource-arn string-arn
--apply-action string-enum
--opt-in-type string-enum
[--apply-on string-date-range]
```

Parameters:

- **`--resource-arn`**—The Amazon DocumentDB Amazon Resource Name (ARN) of the resource to which the pending maintenance action applies.
- **`--apply-action`**—The pending maintenance action to apply to this resource.

  Valid values:

  - `ENGINE_UPDATE`
  - `ENGINE_UPGRADE`
  - `SECURITY_UPDATE`
  - `OS_UPDATE`
  - `MASTER_USER_PASSWORD_UPDATE`
- **`--opt-in-type`**—A value that specifies the type of opt-in request, or undoes an opt-in request. An opt-in request of type `IMMEDIATE` can't be undone.

Valid values:

- `IMMEDIATE`—Apply the maintenance action immediately.
- `NEXT_MAINTENANCE`—Apply the maintenance action during the next maintenance window for the resource.
- `APPLY_ON`—Apply the maintenance action on specified apply date regardless of next maintenance window for the resource.
- `UNDO_OPT_IN`—Cancel any existing `NEXT_MAINTENANCE` or `APPLY_ON` opt in requests.

- **`[--apply-on]`**—Required if opt-in-type is `APPLY_ON`. Format: `yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm` (This option uses UTC time. The start time can be any time in the future from a minimum of 30 minutes and a maximum of 14 days, or force/apply date on pending action, whichever is earlier. Start to end time window can be a minimum of 30 minutes to a maximum of 8 hours long.)

Output from this operation looks something like the following (JSON format):

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'SECURITY_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'IMMEDIATE',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

Parameters:

- `ResourceArn`—The Amazon DocumentDB Amazon Resource Name (ARN) of the resource to which the pending maintenance action applies.
- `Action`—The pending maintenance action being applied to the resource.

Valid values:

- `ENGINE_UPDATE`
- `ENGINE_UPGRADE`
- `SECURITY_UPDATE`
- `OS_UPDATE`
- `MASTER_USER_PASSWORD_UPDATE`

- `AutoAppliedAfterDate`—First maintenance window after this date. `NEXT_MAINTENANCE` `OPT_IN` is ignored in this case.
- `ForcedApplyDate`—Applied regardless of maintenance window. `IMMEDIATE OPT_IN` is ignored in this case.
- `OptInStatus`—A value that specifies the type of opt-in request, or undoes an opt-in request. An opt-in request of type `IMMEDIATE` can't be undone.

  Valid values:

  - `IMMEDIATE`—Apply the maintenance action immediately.
  - `NEXT_MAINTENANCE`—Apply the maintenance action during the next maintenance window for the resource.
  - `APPLY_ON`—Apply the maintenance action on specified apply date regardless of next maintenance window for the resource.
  - `UNDO_OPT_IN`—Cancel any existing `NEXT_MAINTENANCE` or `APPLY_ON` opt in requests.
- `CurrentApplyDate`—Displayed if opt-in-type is `APPLY_ON`.
- `Description`—An option description for the maintenance action.

Apply dates

Each maintenance action has a respective apply date that you can find when describing the pending maintenance actions. When you read the output of pending maintenance actions from the AWS CLI, three dates are listed:

- `CurrentApplyDate`—The date the maintenance action will get applied either immediately or during the next maintenance window. If the maintenance is optional, this value can be null.
- `ForcedApplyDate`—The date when the maintenance will be automatically applied, independent of your maintenance window.

- `AutoAppliedAfterDate`—The date after which the maintenance will be applied during the cluster's maintenance window.

User-created maintenance actions

As an Amazon DocumentDBelastic cluster user, you can initiate updates to your clusters configurations.

Updating cluster primary password

```
aws docdb-elastic update-cluster
--cluster-arn string-arn
[--admin-user-password string]
[--auth-type string-enum]
[--apply-method string-enum]
[--apply-on string-date-range]
#... other parameters of the API that follow here are not relevant for this
 configuration
```

Parameters:

- **`--cluster-arn`**—The Amazon DocumentDB Amazon Resource Name (ARN) of the resource to which the maintenance action will be applied.
- **`[--admin-user-password]`**—The password associated with the admin user.
- **`[--auth-type]`**—The authentication type used to determine where to fetch the password used for accessing the elastic cluster. Valid types are `PLAIN_TEXT` or `SECRET_ARN`.
- **`[--apply-method]`**—A value that specifies the type of method being applied. Allowed values are `IMMEDIATE` and `APPLY_ON`. Default is `IMMEDIATE`.
- **`[--apply-on]`**—Required if `apply-method` is `APPLY_ON`. Format: `yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm` (This option uses UTC time. The start time can be any time in the future from a minimum of 30 minutes and a maximum of 14 days. Start to end time window can be a minimum of 30 minutes to a maximum of 8 hours long.)

Output from this operation looks something like the following (JSON format):

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
```

```
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'MASTER_USER_PASSWORD_UPDATE',
                'OptInStatus': 'APPLY_ON',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

Changing your Amazon DocumentDB maintenance windows

The maintenance window should fall at the time of lowest usage and thus might need changing from time to time. Your elastic cluster is unavailable during this time only if system changes (such as a scale storage operation change) are being applied and require an outage. It is unavailable only for the minimum amount of time required to make the necessary changes.

The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Web Services Region, occurring on a random day of the week.

To change the maintenance window, see Modifying elastic cluster configurations.

Elastic cluster operating system updates

Amazon DocumentDB elastic clusters occasionally require operating system updates. Amazon DocumentDB upgrades the operating system to a newer version to improve database performance and customers' overall security posture. Operating system updates don't change the cluster engine version of an Amazon DocumentDB elastic cluster.

Most operating system updates for Amazon DocumentDB elastic clusters are optional and don't have a set date to apply them. However, if you don't apply these updates for a while, they may eventually become required and automatically applied during your clusters maintenance window. This is to help maintain the security posture of your database. To avoid any unexpected downtime, we recommend that you apply operating system updates to your Amazon DocumentDB elastic cluster as soon as they become available and set your cluster maintenance window at a time of your convenience as per your business needs.

Data encryption at rest for Amazon DocumentDB elastic clusters

The following topics help you learn about, create, and monitor AWS Key Management Service encryption keys for Amazon DocumentDB elastic clusters:

Topics

- How Amazon DocumentDB elastic clusters use grants in AWS KMS
- Create a customer managed key
- Monitoring your encryption keys for Amazon DocumentDB elastic clusters
- Learn more

Amazon DocumentDB elastic clusters automatically integrate with AWS Key Management Service (AWS KMS) for key management and uses a method known as envelope encryption to protect your data. For more information about envelope encryption, see Envelope encryption in the *AWS Key Management Service Developer Guide*.

An AWS KMS key is a logical representation of a key. The KMS key includes metadata, such as the key ID, creation date, description, and key state. The KMS key also contains the key material used to encrypt and decrypt data. For more information about KMS keys, see AWS KMS keys in the *AWS Key Management Service Developer Guide*.

Amazon DocumentDB elastic clusters support encryption with two types of keys:

- **AWS owned keys —** Amazon DocumentDB elastic clusters use these keys by default to automatically encrypt personally identifiable data. You can't view, manage, or use AWS-owned keys, or audit their use. However, you don't have to take any action or change any programs to protect the keys that encrypt your data. For more information, see AWS owned keys in the *AWS Key Management Service Developer Guide*.
- **Customer-managed keys —** Symmetric AWS KMS keys that you create, own, and manage. Because you have full control of this layer of encryption, you can perform such tasks as:
  - Establishing and maintaining key policies
  - Establishing and maintaining IAM policies and grants
  - Enabling and disabling key policies
  - Rotating key cryptographic material

- Adding tags
- Creating key aliases
- Scheduling keys for deletion

For more information, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

> **Important**
>
> You must use a symmetric encryption KMS key to encrypt your cluster as Amazon DocumentDB supports only symmetric encryption KMS keys. Do not use an asymmetric KMS key to attempt to encrypt the data in your Amazon DocumentDB elastic clusters. For more information, see Asymmetric keys in AWS KMS in the *AWS Key Management Service Developer Guide*.
>
> If Amazon DocumentDB can no longer gain access to the encryption key for a cluster — for example, when access to a key is revoked — the encrypted cluster goes into a terminal state. In this case, you can only restore the cluster from a backup. For Amazon DocumentDB, backups are always enabled for 1 day. In addition, if you disable the key for an encrypted Amazon DocumentDB cluster, you will eventually lose read and write access to that cluster. When Amazon DocumentDB encounters a cluster that is encrypted by a key that it doesn't have access to, it puts the cluster into a terminal state. In this state, the cluster is no longer available, and the current state of the database can't be recovered. To restore the cluster, you must re-enable access to the encryption key for Amazon DocumentDB, and then restore the cluster from a backup.

> **Important**
>
> You cannot change the KMS key for an encrypted cluster after you have already created it. Be sure to determine your encryption key requirements before you create your encrypted elastic cluster.

How Amazon DocumentDB elastic clusters use grants in AWS KMS

Amazon DocumentDB elastic clusters require a grant to use your customer managed key.

When you create a cluster encrypted with a customer managed key, Amazon DocumentDB elastic clusters create a grant on your behalf by sending a `CreateGrant` request to AWS KMS. Grants in AWS KMS are used to give Amazon DocumentDB elastic clusters access to a KMS key in a customer account.

Amazon DocumentDB elastic clusters require the grant to use your customer managed key for the following internal operations:

- Send `DescribeKey` requests to AWS KMS to verify that the symmetric customer managed KMS key ID, entered when creating a tracker or geofence collection, is valid.
- Send `GenerateDataKey` requests to AWS KMS to generate data keys encrypted by your customer managed key.
- Send `Decrypt` requests to AWS KMS to decrypt the encrypted data keys so that they can be used to encrypt your data.
- You can revoke access to the grant, or remove the service's access to the customer managed key at any time. If you do, Amazon DocumentDB elastic clusters won't be able to access any of the data encrypted by the customer managed key, which affects operations that are dependent on that data.

Create a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console or the AWS KMS API.

Symmetric customer managed key creation

Follow the steps for Creating symmetric customer managed key in the *AWS Key Management Service Developer Guide*.

Key policy

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key, you can specify a key policy. For more information, see the KMS key access information located in the AWS Key Management Service overview of the *AWS Key Management Service Developer Guide*.

To use your customer managed key with Amazon DocumentDB elastic cluster resources, the following API operations must be permitted in the key policy:

- `kms:CreateGrant` – Adds a grant to a customer managed key. Grants control access to a specified KMS key, which allows access to grant operations Amazon Location Service requires. For more information about using grants, see Grants in AWS KMS in the *AWS Key Management Service Developer Guide*.
- `kms:DescribeKey` – Provides the customer managed key details to allow Docdb Elastic to validate the key.
- `kms:Decrypt` – Allows Docdb Elastic to use the stored encrypted data key to access encrypted data.
- `kms:GenerateDataKey` – Allows Docdb Elastic to generate an encrypted data key and store it because the data key isn't immediately used to encrypt.

For more information, see Permissions for AWS services in key policies and Troubleshooting key access in the *AWS Key Management Service Developer Guide*.

Restricting customer managed key access via IAM policies

In addition to KMS key policies, you can also restrict KMS key permissions in an IAM policy.

You can make the IAM policy stricter in various ways. For example, to allow the customer managed key to be used only for requests that originate in Amazon DocumentDB elastic clusters, you can use the `kms:ViaService` condition key with the `docdb-elastic.<region-name>.amazonaws.com` value.

For more information, see Allowing users in other accounts to use a KMS key in the *AWS Key Management Service Developer Guide*.

Monitoring your encryption keys for Amazon DocumentDB elastic clusters

When you use an AWS KMS key customer managed key with your Docdb Elastic resources, you can use AWS CloudTrail or Amazon CloudWatch Logs to track requests that Docdb Elastic sends to AWS KMS.

The following examples are AWS CloudTrail events for `CreateGrant`, `GenerateDataKeyWithoutPlainText`, `Decrypt`, and `DescribeKey` to monitor AWS KMS key operations called by Amazon DocumentDB elastic clusters to access data encrypted by your customer managed key:

CreateGrant


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-09T23:04:20Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-09T23:55:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "retiringPrincipal": "docdb-elastic.us-east-1.amazonaws.com",
        "granteePrincipal": "docdb-elastic.us-east-1.amazonaws.com",
        "operations": [
            "Decrypt",
            "Encrypt",
            "GenerateDataKey",
            "GenerateDataKeyWithoutPlaintext",
            "ReEncryptFrom",
            "ReEncryptTo",
            "CreateGrant",
            "RetireGrant",
```

```
            "DescribeKey"
        ],
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKey

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
```

```
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-10T18:02:59Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-10T18:03:25Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

```
{
```

```
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-10T18:05:49Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-10T18:06:19Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

DescribeKey

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-09T23:04:20Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-09T23:55:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "keyId": "alias/SampleKmsKey"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
```

```
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Learn more

The following resources provide more information about data encryption at rest:

- For more information about AWS KMS concepts, see AWS Key Management Service basic concepts in the *AWS Key Management Service Developer Guide*.
- For more information about AWS KMS security, see Security best practices for AWS Key Management Service in the *AWS Key Management Service Developer Guide*.

Service-linked roles in elastic clusters

Amazon DocumentDB elastic clusters use AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon DocumentDB elastic clusters. Service-linked roles are predefined by Amazon DocumentDB elastic clusters and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes using Amazon DocumentDB elastic clusters easier because you don't have to manually add the necessary permissions. Amazon DocumentDB elastic clusters defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon DocumentDB elastic clusters can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity. You can delete the roles only after first deleting their related resources. This protects your Amazon

DocumentDB elastic clusters resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that are marked with **Yes** in the **Service-Linked Role** column. Choose a Yes with a link to view the service-linked role documentation for that service.

Service-linked role permissions for elastic clusters

Amazon DocumentDB elastic clusters uses the service-linked role named `AWSServiceRoleForDocDB-Elastic` to allow Amazon DocumentDB elastic clusters to call AWS services on behalf of your clusters.

This service-linked role has a permissions policy attached to it called `AmazonDocDB-ElasticServiceRolePolicy` that grants it permissions to operate in your account. The role permissions policy allows Amazon DocumentDB elastic clusters to complete the following actions on the specified resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. If you encounter the following error message: **"Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later."**, make sure you have the following permissions enabled:

```
{
"Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-elastic.amazonaws.com/
AWSServiceRoleForDocDB-Elastic",
    "Condition": {
"StringLike": {
"iam:AWSServiceName":"docdb-elastic.amazonaws.com"
        }
    }
}
```

For more information, see Service-linked role permissions in the *AWS Identity and Access Management User Guide*.

Creating a service-linked role for Amazon DocumentDB elastic clusters

You don't need to manually create a service-linked role. When you create a DB instance, Amazon DocumentDB elastic clusters creates the service-linked role for you.

Editing a service-linked role for Amazon DocumentDB elastic clusters

Amazon DocumentDB elastic clusters do not allow you to edit the `AWSServiceRoleForDocDB-Elastic` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *AWS Identity and Access Management User Guide*.

Deleting a service-linked role for Amazon DocumentDB elastic clusters

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all of your clusters before you can delete the service-linked role.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console:

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane of the IAM console, choose **Roles**. Then choose the name (not the check box) of the `AWSServiceRoleForDocDB-Elastic` role.
3. On the **Summary** page for the chosen role, choose the **Access Advisor** tab.

> **Note**
>
> If you are unsure whether Amazon DocumentDB elastic clusters is using the `AWSServiceRoleForDocDB-Elastic` role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the AWS Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.
> If you want to remove the `AWSServiceRoleForDocDB-Elastic` role, you must first delete all of your clusters.

Deleting all of your clusters

To delete a cluster in the Amazon DocumentDB console:

1. Sign in to the AWS Management Console and open the Amazon DocumentDB console.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster that you want to delete.
4. For **Actions**, choose **Delete**.

5. If you are prompted to **Create final Snapshot?**, choose **Yes** or **No**.
6. If you chose **Yes** in the previous step, for **Final snapshot name** enter the name of your final snapshot.
7. Choose **Delete**.

> **Note**
>
> You can use the IAM console, the IAM CLI, or the IAM API to delete the `AWSServiceRoleForDocDB-Elastic` service-linked role. For more information, see Deleting a service-linked role in the *AWS Identity and Access Management User Guide*.

Monitoring Amazon DocumentDB

Monitoring your AWS services is an important part of keeping your systems healthy and functioning optimally. It's wise to collect monitoring data from all parts of your AWS solution so that you can more easily debug and fix failures or degradations, should they occur. Before you begin monitoring your AWS solutions, we recommend that you consider and formulate answers for the following questions:

- What are your monitoring goals?
- What resources are you going to monitor?
- How frequently will you monitor these resources?
- What monitoring tools will you use?
- Who is responsible for doing the monitoring?
- Who is to be notified and by what means if something goes wrong?

To understand your current performance patterns, identify performance anomalies, and formulate methods to address issues, you should establish baseline performance metrics for various times and under differing load conditions. As you monitor your AWS solution, we recommend that you store your historical monitoring data for future reference and for establishing your baselines.

In general, acceptable values for performance metrics depend on what your baseline looks like and what your application is doing. Investigate consistent or trending variances from your baseline. The following is advice about specific types of metrics:

- **High CPU or RAM use** — High values for CPU or RAM use might be appropriate, provided that they are in keeping with your goals for your application (like throughput or concurrency) and are expected.
- **Storage volume consumption** —Investigate storage consumption (`VolumeBytesUsed`) if space that is used is consistently at or above 85 percent of the total storage volume space. Determine whether you can delete data from the storage volume or archive data to a different system to free up space. For more information, see Amazon DocumentDB storage and Amazon DocumentDB Quotas and limits.
- **Network traffic** — For network traffic, talk with your system administrator to understand what the expected throughput is for your domain network and internet connection. Investigate network traffic if throughput is consistently lower than expected.

- **Database connections** — Consider constraining database connections if you see high numbers of user connections in conjunction with decreases in instance performance and response time. The best number of user connections for your instance will vary based on your instance class and the complexity of the operations being performed.
- **IOPS metrics**—The expected values for IOPS metrics depend on disk specification and server configuration, so use your baseline to know what is typical. Investigate if values are consistently different from your baseline. For best IOPS performance, make sure that your typical working set fits into memory to minimize read and write operations.

Amazon DocumentDB (with MongoDB compatibility) provides a variety of Amazon CloudWatch metrics that you can monitor to determine the health and performance of your Amazon DocumentDB clusters and instances. You can view Amazon DocumentDB metrics using various tools, including the Amazon DocumentDB console, AWS CLI, CloudWatch API, and Performance Insights.

Topics

- Monitoring an Amazon DocumentDB cluster's status
- Monitoring an Amazon DocumentDB instance's status
- Viewing Amazon DocumentDB recommendations
- Using Amazon DocumentDB event subscriptions
- Monitoring Amazon DocumentDB with CloudWatch
- Logging Amazon DocumentDB API calls with AWS CloudTrail
- Profiling Amazon DocumentDB operations
- Monitoring with Performance Insights

Monitoring an Amazon DocumentDB cluster's status

The status of a cluster indicates the health of the cluster. You can view the status of a cluster by using the Amazon DocumentDB console or the AWS CLI `describe-db-clusters` command.

Topics

- Cluster status values
- Monitoring a cluster's status

Cluster status values

The following table lists the valid values for a cluster's status.

Cluster Status	Description
`active`	The cluster is active. This status applies to elastic clusters only.
`available`	The cluster is healthy and available. This status applies to instance-based clusters only.
`backing-up`	The cluster is currently being backed up.
`creating`	The cluster is being created. It is inaccessible while it is being created.
`deleting`	The cluster is being deleted. It is inaccessible while it is being deleted.
`failing-over`	A failover from the primary instance to an Amazon DocumentDB replica is being performed.
`inaccessible-encryption-credentials`	The AWS KMS key used to encrypt or decrypt the cluster can't be accessed.
`maintenance`	A maintenance update is being applied to the cluster. This status is used for cluster-level maintenance that Amazon

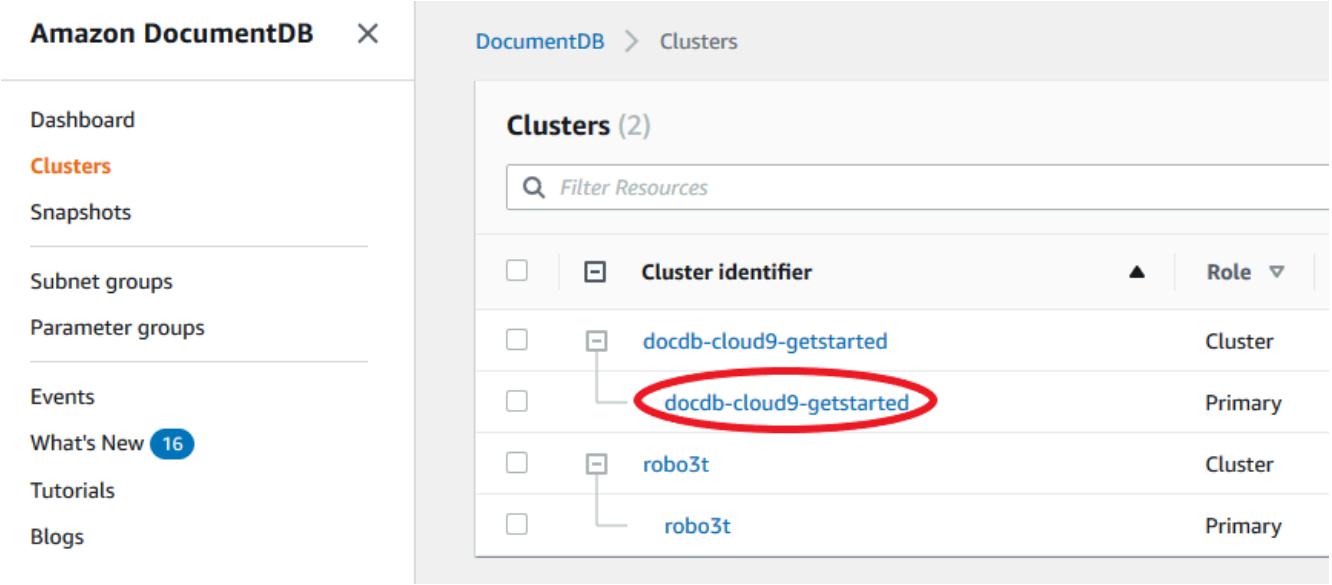
Cluster Status	Description
	DocumentDB schedules well in advance.
`migrating`	A cluster snapshot is being restored to a cluster.
`migration-failed`	A migration failed.
`modifying`	The cluster is being modified because of a customer request to modify the cluster.
`renaming`	The cluster is being renamed because of a customer request to rename it.
`resetting-master-credentials`	The master credentials for the cluster are being reset because of a customer request to reset them.
`upgrading`	The cluster engine version is being upgraded.

Monitoring a cluster's status

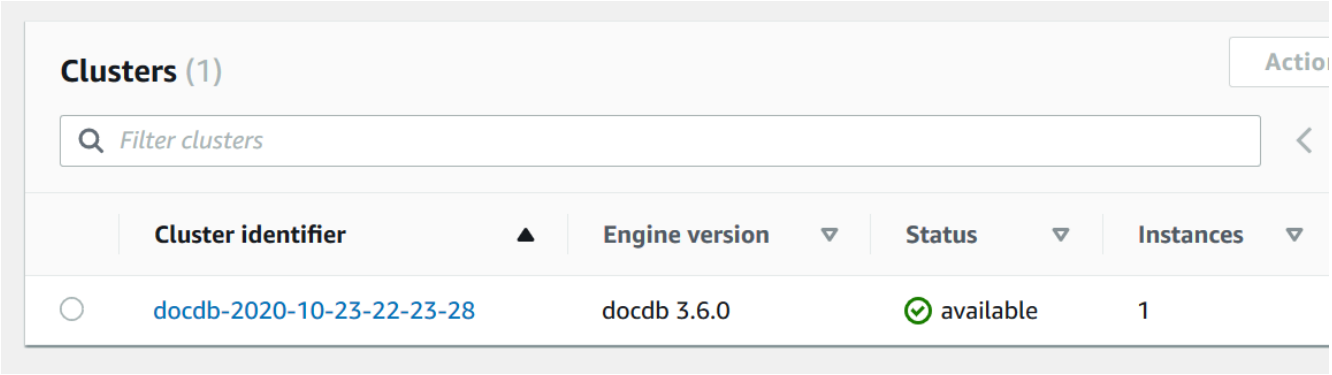
Using the AWS Management Console

When using the AWS Management Console to determine the status of a cluster, use the following procedure.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.
3. In the Clusters navigation box, you'll see the column **Cluster identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. In the **Cluster identifier** column, find the name of the instance that you are interested in. Then, to find the status of the instance, read across that row to the **Status** column, as shown below.



Using the AWS CLI

When using the AWS CLI to determine the status of a cluster, use the `describe-db-clusters` operation. The following code finds the status of the cluster `sample-cluster`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

For Windows:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

Output from this operation looks something like the following.

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Monitoring an Amazon DocumentDB instance's status

Amazon DocumentDB provides information about the current condition of each configured instance in the database.

There are three types of status that you can view for an Amazon DocumentDB instance:

- Instance status: This status is shown in the **Status** column of the cluster table in the AWS Management Console and shows the current life cycle condition of the instance. The values shown in the **Status** column are derived from the `Status` field of the `DescribeDBCluster` API response.
- Instance health status: This status is shown in the **Instance health** column of the cluster table in the AWS Management Console and shows whether the database engine, the component responsible for managing and retrieving data, is running. The values shown in the **Instance health** column are based on the Amazon CloudWatch `EngineUptime` system metric.
- Maintenance status: This status is shown in the **Maintenance** column of the cluster table in the AWS Management Console and indicates the status of any maintenance event that needs to be applied to an instance. Maintenance status is independent of the other instance status' and is derived from the `PendingMaintenanceAction` API. For more information about maintenance status, see Maintaining Amazon DocumentDB.

Topics

- Instance status values

- Monitoring instance status using the AWS Management Console or AWS CLI
- Instance health status values
- Monitoring instance health status using the AWS Management Console

Instance status values

The following table lists the possible status values for instances and how you are billed for each status. It shows if you will be billed for the instance and storage, only storage, or not billed. For all instance statuses, you are always billed for backup usage.

Instance status	Billed	Description
`available`	Billed	The instance is healthy and available.
`backing-up`	Billed	The instance is currently being backed up.
`configuring-log-exports`	Billed	Publishing log files to Amazon CloudWatch Logs is being enabled or disabled for this instance.
`creating`	Not billed	The instance is being created. The instance is not accessible while it is being created.
`deleting`	Not billed	The instance is being deleted.
`failed`	Not billed	The instance has failed and Amazon DocumentDB was unable to recover it. To recover the data, perform a point-in-time restore to the latest restorable time of the instance.
`inaccessible-encryption-credentials`	Not billed	The AWS KMS key that is used to encrypt or decrypt the instance could not be accessed.
`incompatible-network`	Not billed	Amazon DocumentDB is attempting to perform a recovery action on an instance but is unable to do so because the VPC is in

Instance status	Billed	Description
		a state that is preventing the action from being completed. This status can occur if, for example, all available IP addresses in a subnet were in use and Amazon DocumentDB was unable to get an IP address for the instance.
`maintenance`	Billed	Amazon DocumentDB is applying a maintenance update to the instance. This status is used for instance-level maintenance that Amazon DocumentDB schedules well in advance. We're evaluating ways to expose additional maintenance actions to customers through this status.
`modifying`	Billed	The instance is being modified because of a request to modify the instance.
`rebooting`	Billed	The instance is being rebooted because of a request or an Amazon DocumentDB process that requires the rebooting of the instance.
`renaming`	Billed	The instance is being renamed because of a request to rename it.
`resetting-master-credentials`	Billed	The master credentials for the instance are being reset because of a request to reset them.
`restore-error`	Billed	The instance encountered an error attempting to restore to a point-in-time or from a snapshot.
`starting`	Billed for storage	The instance is starting.
`stopped`	Billed for storage	The instance is stopped.

Instance status	Billed	Description
`stopping`	Billed for storage	The instance is being stopped.
`storage-full`	Billed	The instance has reached its storage capacity allocation. This is a critical status and should be remedied immediately; scale up your storage by modifying the instance. Set Amazon CloudWatch alarms to warn you when storage space is getting low so you don't run into this situation.

Monitoring instance status using the AWS Management Console or AWS CLI

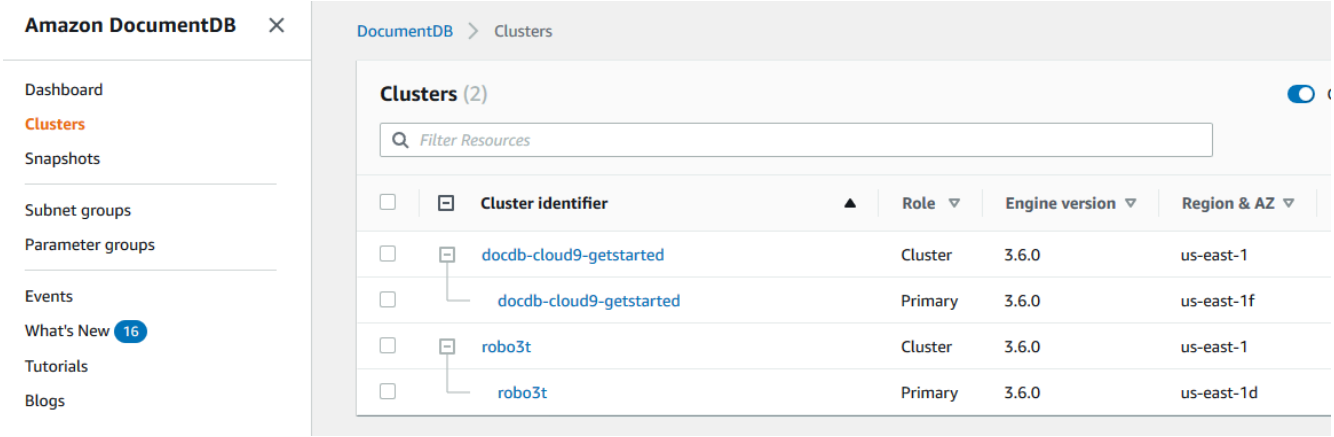
Use the AWS Management Console or AWS CLI to monitor the status of you instance.

Using the AWS Management Console

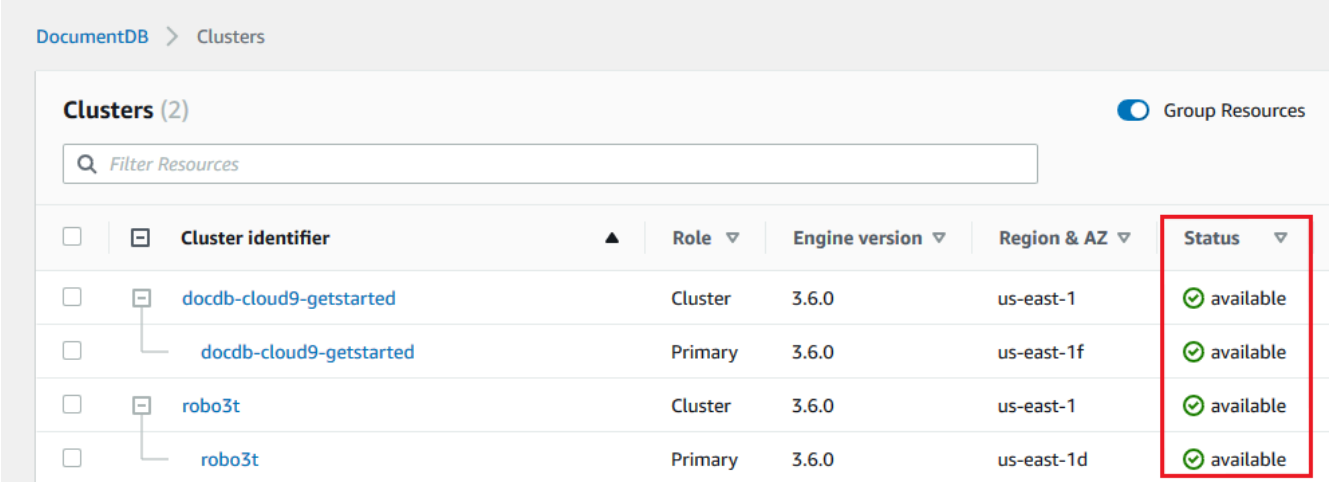
When using the AWS Management Console to determine the status of a cluster, use the following procedure.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

> **Note**
>
> Note that in the Clusters navigation box, the column **Cluster identifier** shows both clusters and instances. Instances are listed underneath clusters, similar to the image below.



3. Find the name of the instance that you are interested in. Then, to find the status of the instance, read across that row to the **Status** column, as shown following.



Using the AWS CLI

When using the AWS CLI to determine the status of a cluster, use the `describe-db-instances` operation. The following code finds the status of the instance `sample-cluster-instance-01`.

For Linux, macOS, or Unix:

```
aws docdb describe-db-instances \
          --db-instance-identifier sample-cluster-instance-01  \
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

For Windows:

```
aws docdb describe-db-instances ^
          --db-instance-identifier sample-cluster-instance-01  ^
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

Output from this operation looks something like the following.

```
[
          [
              "sample-cluster-instance-01",
              "available"
          ]
      ]
```

Instance health status values

The following table lists the possible health status values for instances. The **Instance health** column, located in the **Clusters** table in the AWS Management Console, shows whether the database engine, the component responsible for storing, managing, and retrieving data, is operating normally. This column also indicates if the `EngineUptime` system metric, available in CloudWatch, is showing the health status of each instance.

Instance health status	Description
healthy	Database engine is running in the Amazon DocumentDB instance.
unhealthy	Database engine is not running or has restarted less than a minute ago.

Monitoring instance health status using the AWS Management Console

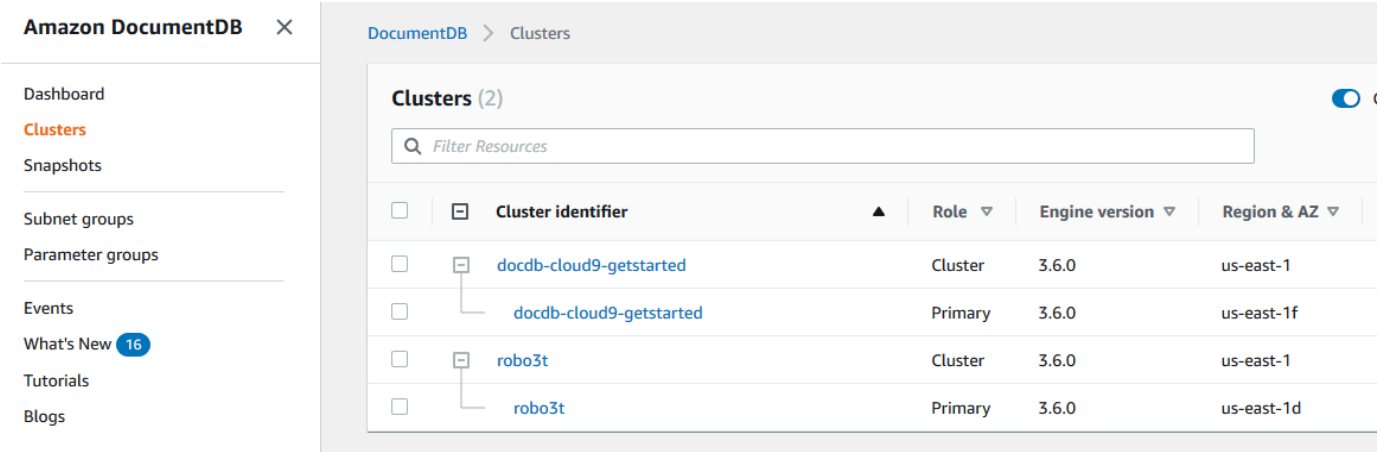
Use the AWS Management Console to monitor the health status of you instance.

When using AWS Management Console, use the following steps to understand the instance's health status.

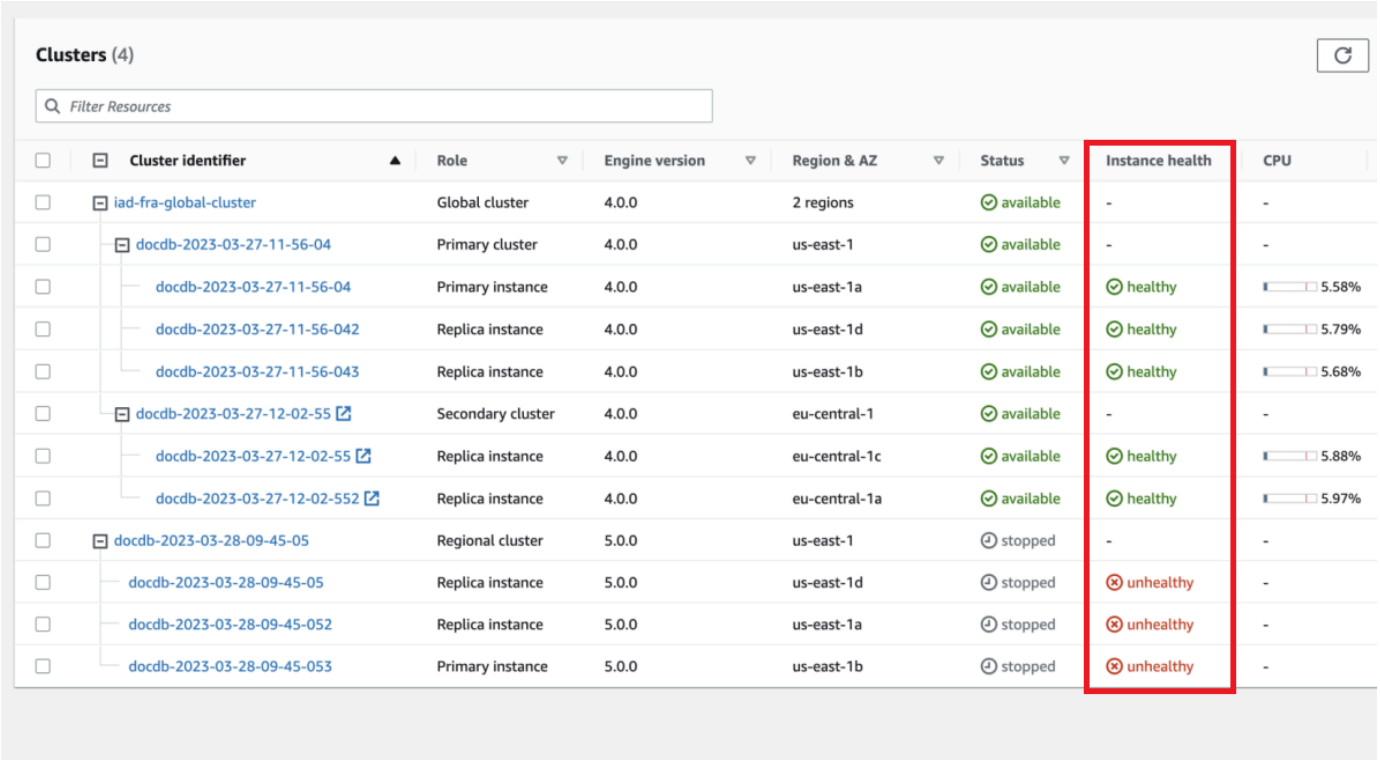
1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

2. In the navigation pane, choose **Clusters**.

> **Note**
>
> In the **Clusters** navigation box, the column **Cluster identifier** shows both clusters and instances. Instances are listed underneath clusters, similar to the image below.



3. Find the name of the instance that you are interested in. Then, to find the status of the instance, read across that row to the **Instance health** column, as shown in the following image:



> **Note**
>
> Instance health status polling occurs every 60 seconds and is based on the CloudWatch `EngineUptime` system metric. The values in the **Instance health** column are automatically updated.

Viewing Amazon DocumentDB recommendations

Amazon DocumentDB provides a list of automated recommendations for database resources, such as instances and clusters. These recommendations provide best practice guidance by analyzing your cluster and instance configurations.

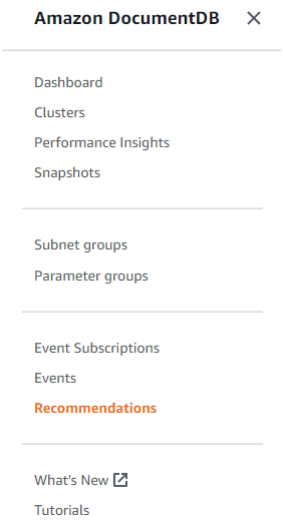
As an example of these recommendations, see the following:

Type	Description	Recommendation	Additional information
One instance	Cluster only contains one instance	Performance and availability: we recommend adding another instance with the same instance class in a different Availability Zone.	Amazon DocumentDB High Availability and Replication

Amazon DocumentDB generates recommendations for a resource when the resource is created or modified. Amazon DocumentDB also periodically scans your resources and generates recommendations.

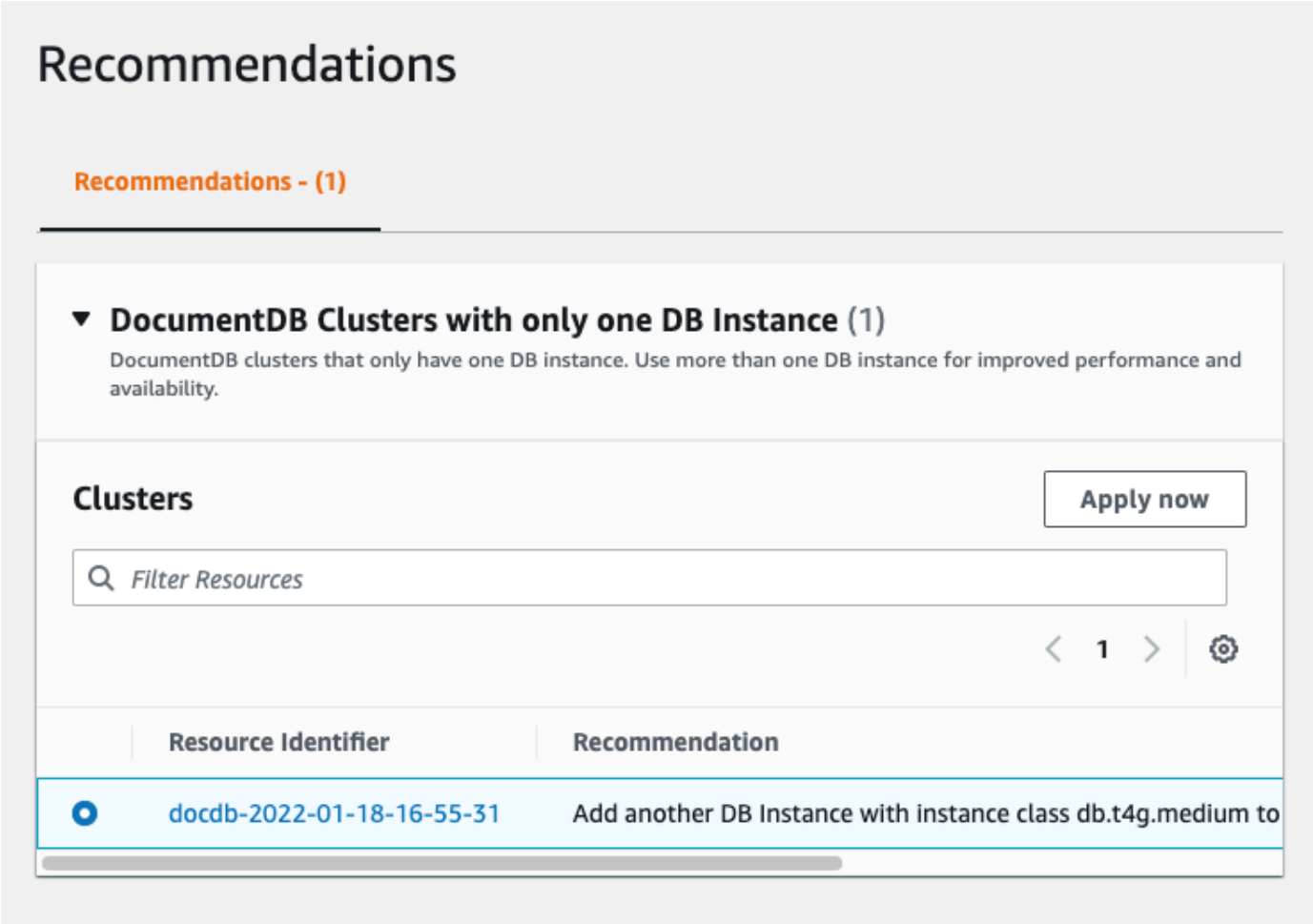
To view and take action on Amazon DocumentDB recommendations

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Recommendations**:



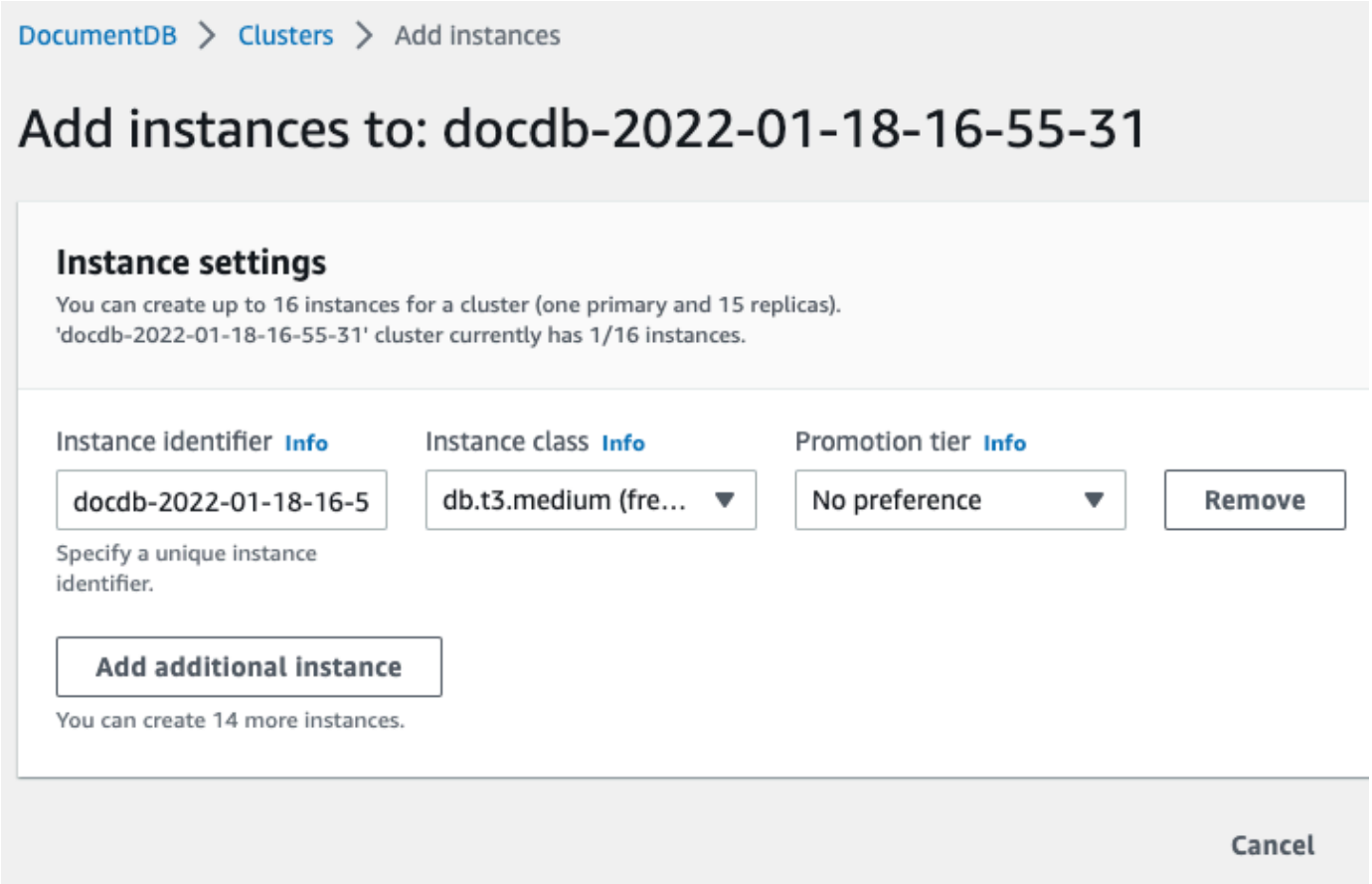
3. In the **Recommendations** dialog, expand the section of interest and select the recommended task.

   In the example below, the recommended task applies to an Amazon DocumentDB cluster with only one instance. The recommendation is to add another instance to improve performance and availability.



4. Click **Apply now**.

   For this example, the **Add instances** dialog appears:



5. Modify your new instance's settings and click **Create**.

Using Amazon DocumentDB event subscriptions

Amazon DocumentDB uses Amazon Simple Notification Service (Amazon SNS) to provide notifications when an Amazon DocumentDB event occurs. These notifications can be in any form that is supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint.

Amazon DocumentDB groups these events into categories that you can subscribe to so that you can be notified when an event in that category occurs. You can subscribe to an event category for an instance, cluster, snapshot, cluster snapshot, or for a parameter group. For example, if you subscribe to the Backup category for a given instance, you are notified whenever a backup-related event occurs that affects the instance. You also receive notification when an event subscription changes.

Events occur at both the cluster and the instance level, so you can receive events if you subscribe to a cluster or an instance.

Event subscriptions are sent to the addresses you provide when you create the subscription. You might want to create several different subscriptions, such as a subscription that receives all event notifications and another subscription that includes only critical events for your production instances. You can easily turn off notification without deleting a subscription. To do so, set the **Enabled** radio button to **No** in the Amazon DocumentDB console.

> **Important**
>
> Amazon DocumentDB doesn't guarantee the order of events sent in an event stream. The event order is subject to change.

Amazon DocumentDB uses the Amazon Resource Name (ARN) of an Amazon SNS topic to identify each subscription. The Amazon DocumentDB console creates the ARN for you when you create the subscription.

Billing for Amazon DocumentDB event subscriptions is through Amazon SNS. Amazon SNS fees apply when using event notification. For more information, see Amazon Simple Notification Service Pricing. Other than Amazon SNS charges, Amazon DocumentDB does not bill for event subscriptions.

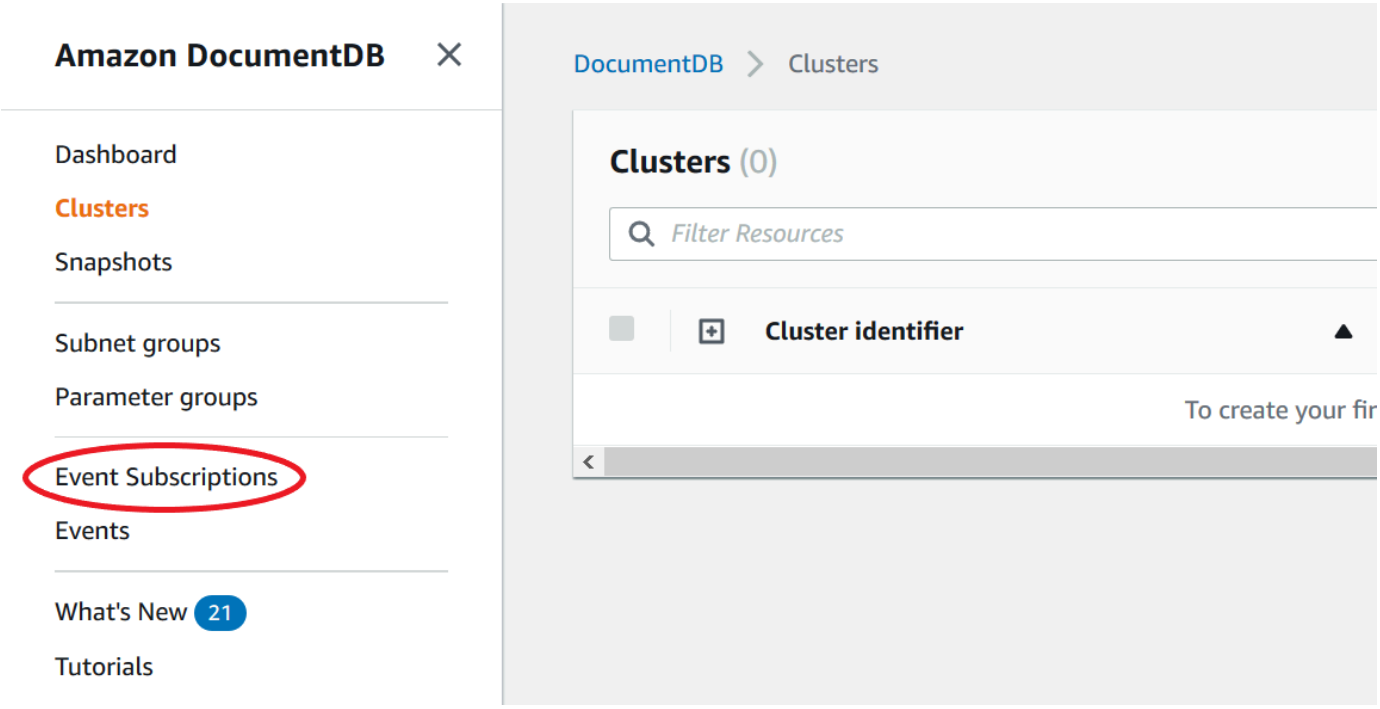
Topics

- Subscribing to Amazon DocumentDB events
- Managing Amazon DocumentDB event notification subscriptions
- Amazon DocumentDB event categories and messages

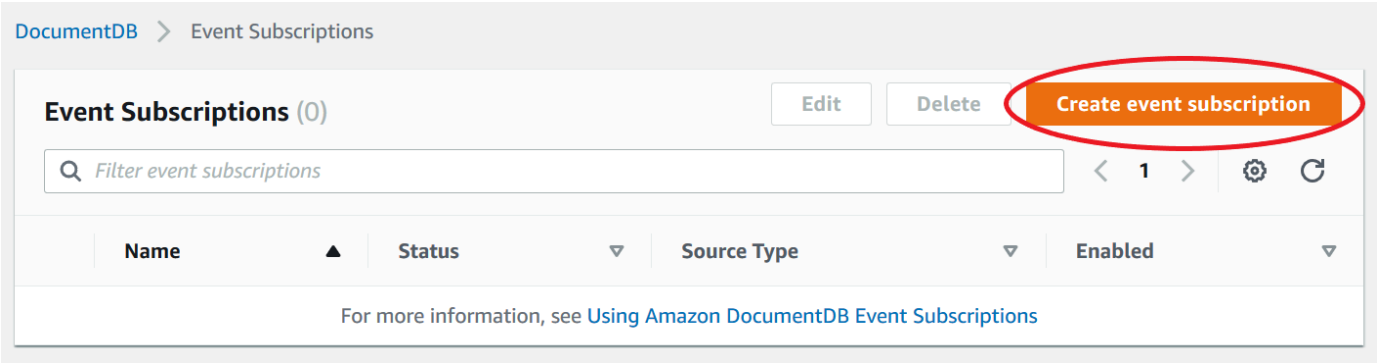
Subscribing to Amazon DocumentDB events

You can use the Amazon DocumentDB console to subscribe to event subscriptions, as follows:

1. Sign in to the AWS Management Console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Event subscriptions**.

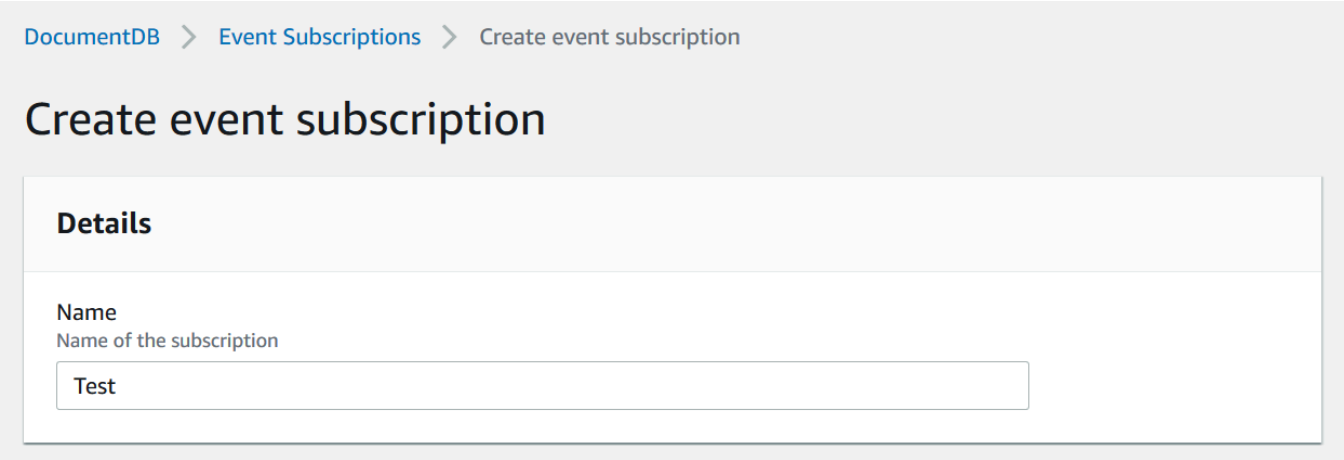


3. In the **Event subscriptions** pane, choose **Create event subscription**.

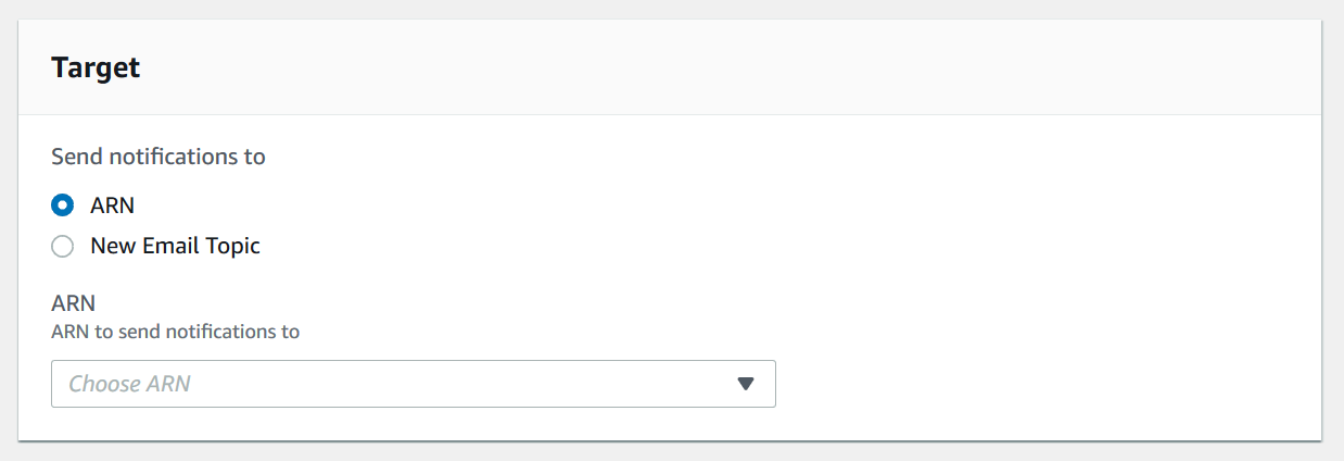


4. In the **Create event subscription** dialog box, do the following:

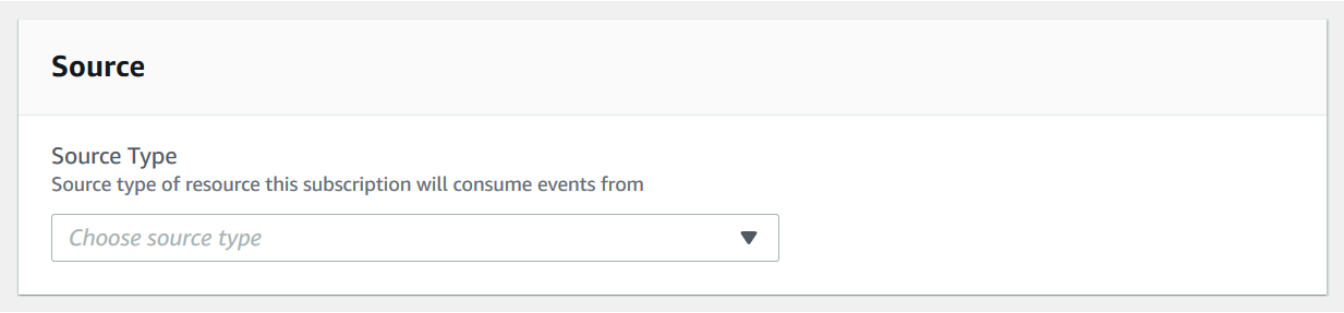
   - For **Name**, enter a name for the event notification subscription.



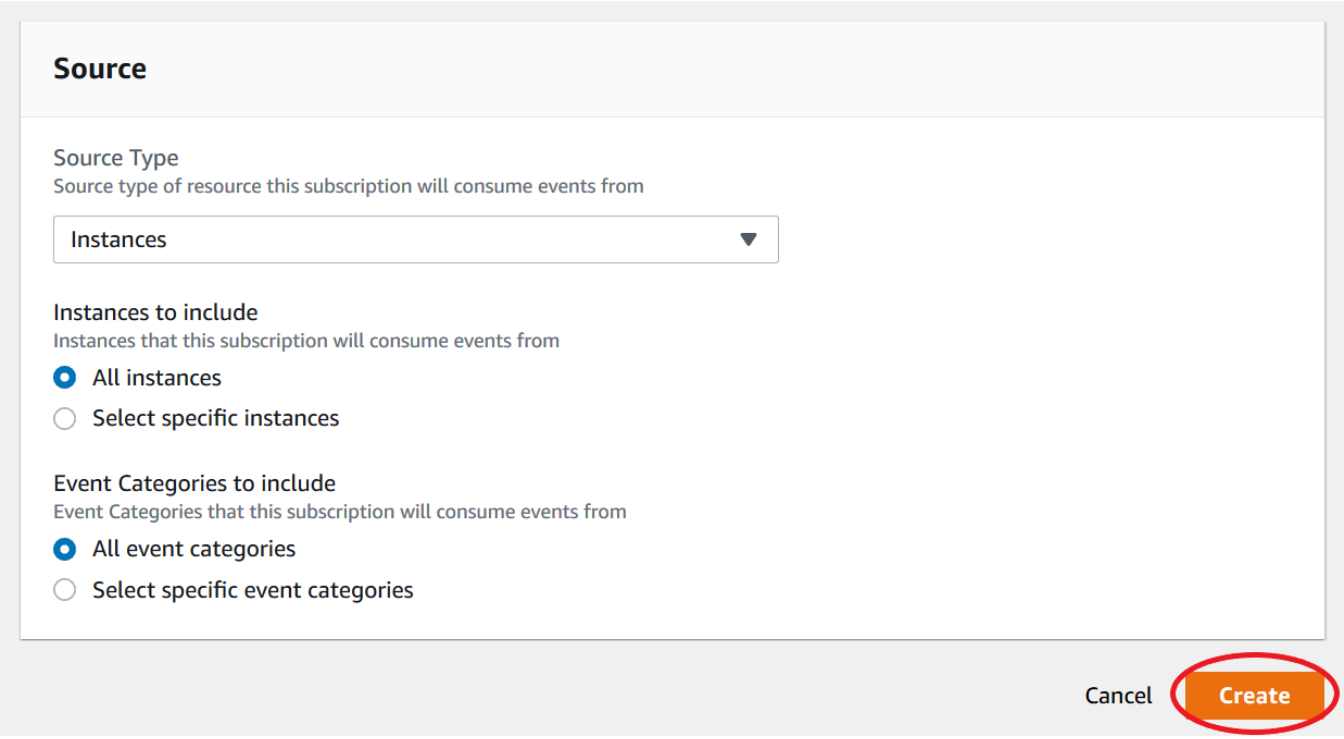
- For **Target**, choose where you want to send notifications to. You can choose an existing **ARN** or choose **New Email Topic** to enter the name of a topic and a list of recipients.



- For **Source**, choose a source type. Depending on the source type you selected, choose the event categories and the sources that you want to receive event notifications from.



- Choose **Create**.

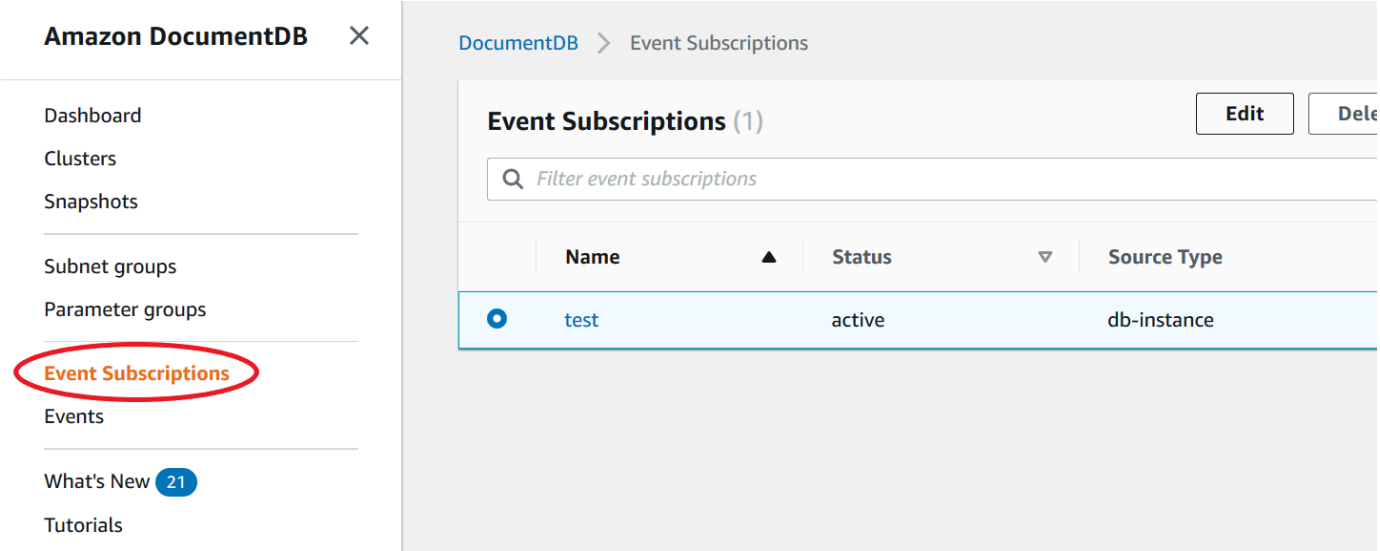


Managing Amazon DocumentDB event notification subscriptions

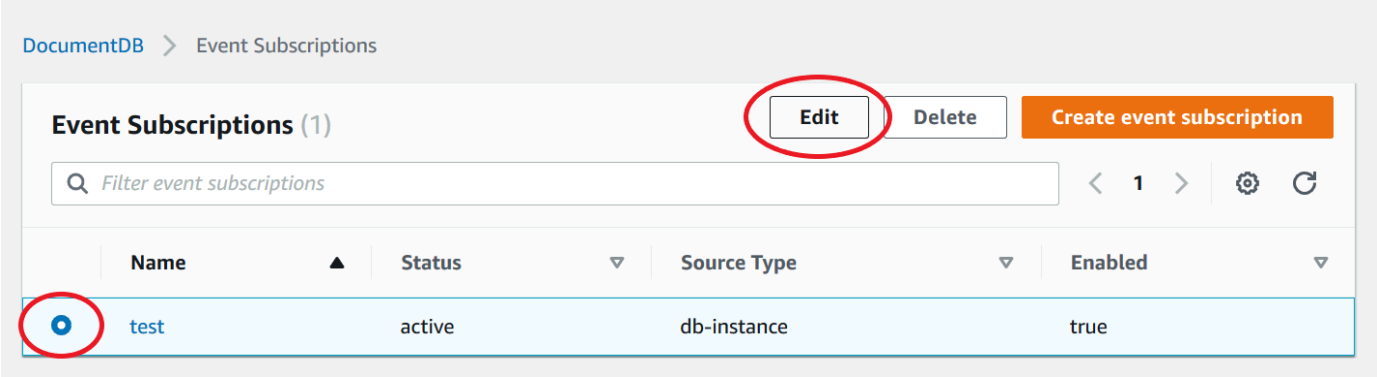
If you choose **Event subscriptions** in the navigation pane of the Amazon DocumentDB console, you can view subscription categories and a list of your current subscriptions. You can also modify or delete a specific subscription.

To modify your current Amazon DocumentDB event notification subscriptions

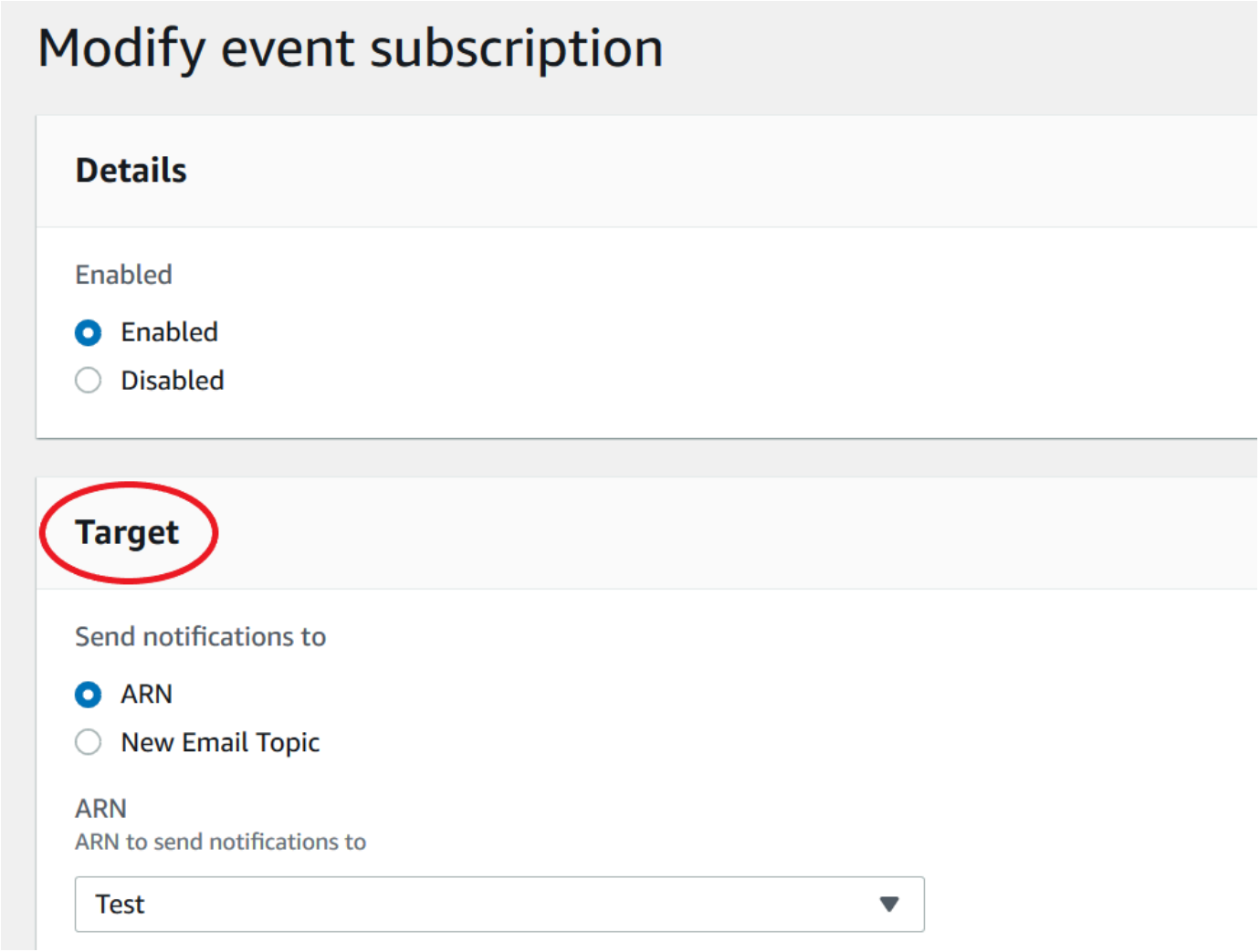
1. Sign in to the AWS Management Console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Event subscriptions**. The **Event subscriptions** pane shows all your event notification subscriptions.



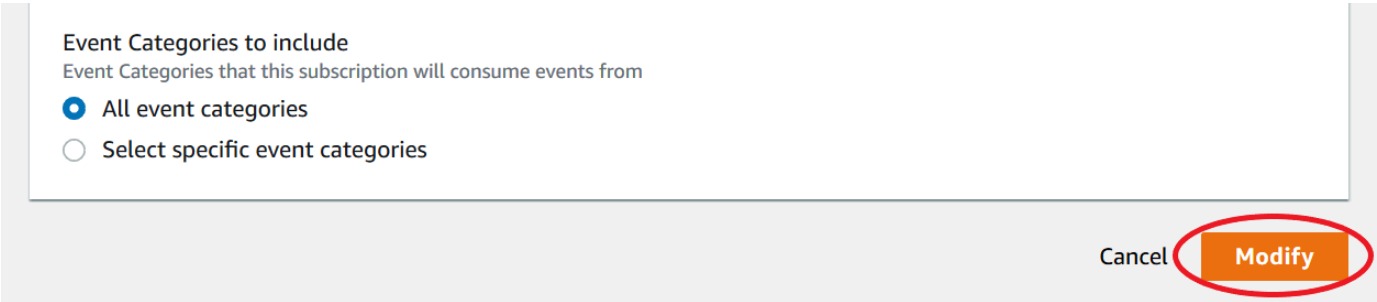
3. In the **Event subscriptions** pane, choose the subscription that you want to modify and choose **Edit**.



4. Make your changes to the subscription in either the **Target** or **Source** section. You can add or remove source identifiers by selecting or deselecting them in the Source section.



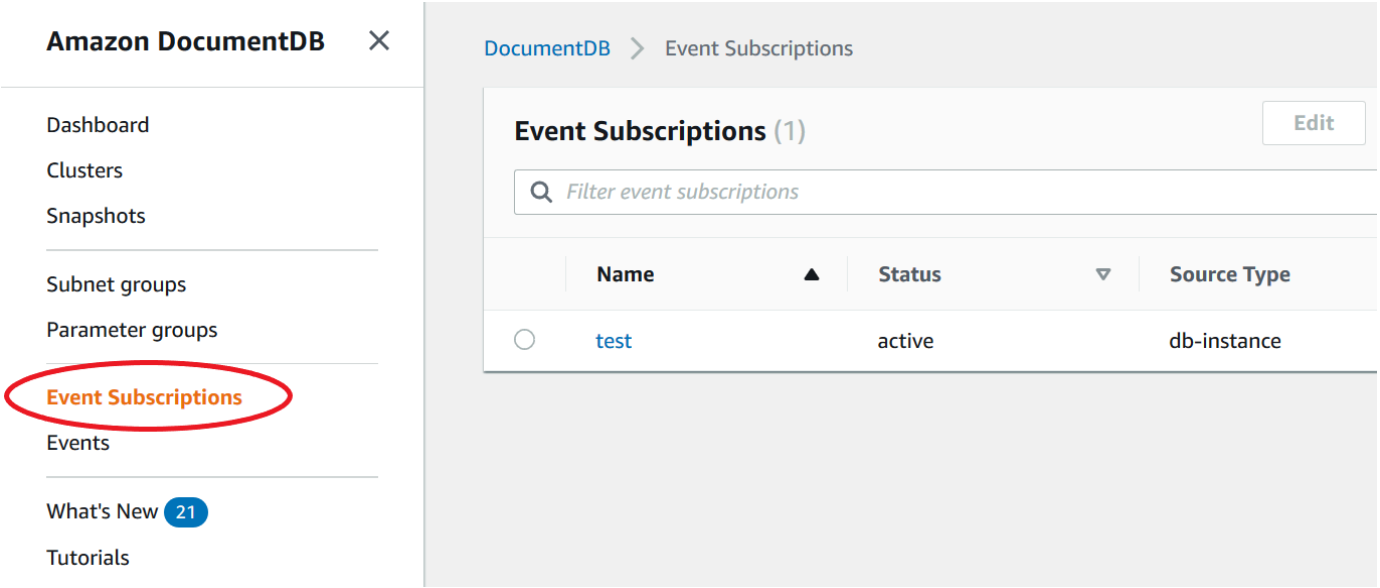
5. Choose **Modify**. The Amazon DocumentDB console indicates that the subscription is being modified.



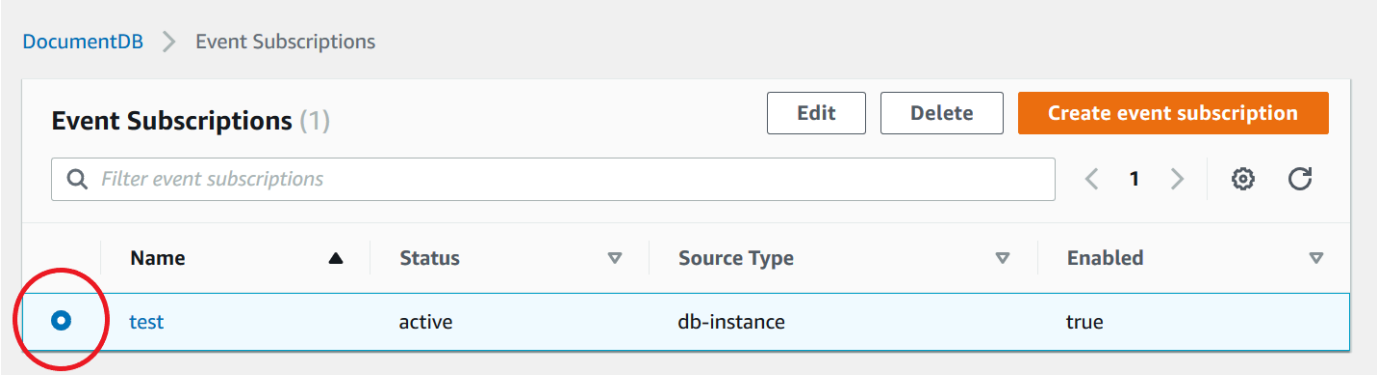
Deleting an Amazon DocumentDB event notification subscription

You can delete a subscription when you no longer need it. All subscribers to the topic will no longer receive event notifications specified by the subscription.

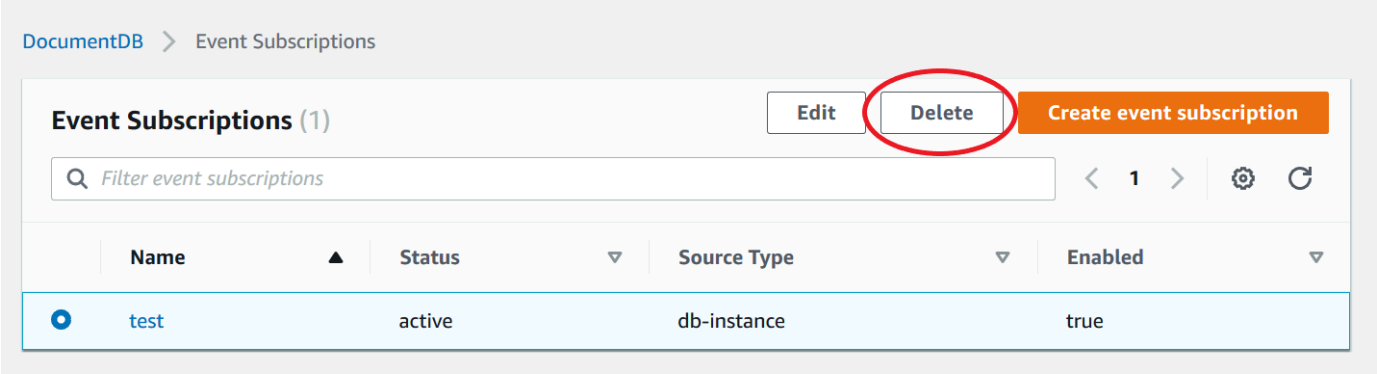
1. Sign in to the AWS Management Console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Event subscriptions**.



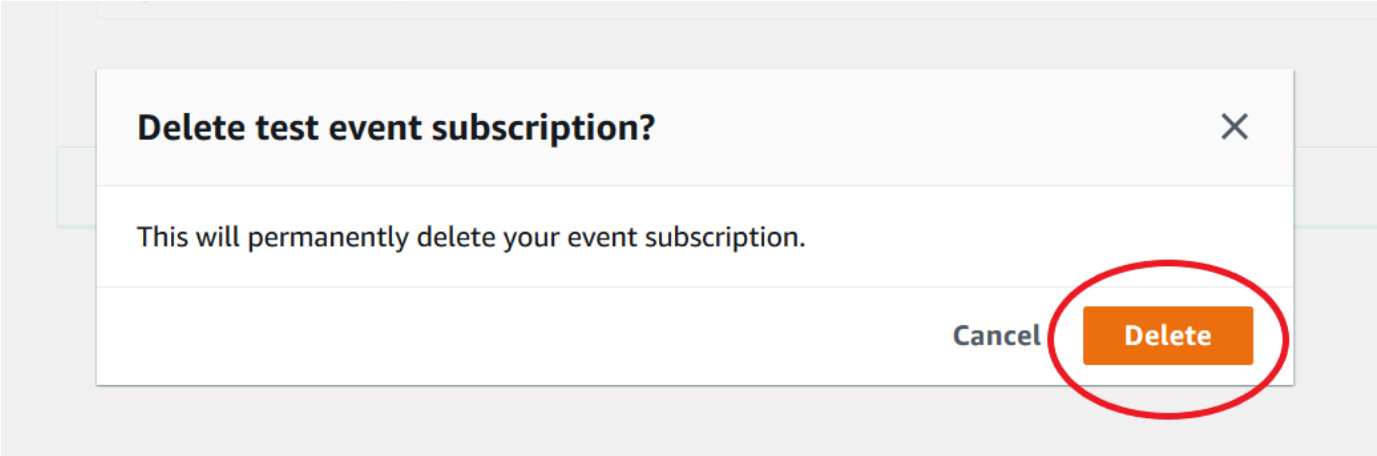
3. In the **Event subscriptions** pane, choose the subscription that you want to delete.



4. Choose **Delete**.



5. A pop-up window will appear asking you if you want to permanently delete this notification. Choose **Delete**.



Amazon DocumentDB event categories and messages

Amazon DocumentDB generates a significant number of events in categories that you can subscribe to using the console. Each category applies to a source type, which can be an instance, cluster, snapshot, or parameter group.

> **Note**
>
> Amazon DocumentDB uses existing Amazon RDS event definitions and IDs.

Amazon DocumentDB events originating from instances

Category	Description
availability	The instance restarted.
availability	The instance shutdown.
configuration change	Applying modification to an instance class.
configuration change	Finished applying modification to an instance class.
configuration change	Reset primary credentials.
creation	Instance created.

Category	Description
deletion	Instance deleted
failure	The instance has failed due to an incompatible configuration or an underlying storage issue. Begin a point-in-time-restore for the instance.
notification	Instance stopped.
notification	Instance started.
notification	Instance is being started due to it exceeding the maximum allowed time being stopped.
recovery	Recovery of the instance has started. Recovery time will vary with the amount of data to be recovered.
recovery	Recovery of the instance is complete.
security patching	The operating system update is available for your instance. For information about applying updates, see Maintaining Amazon DocumentDB.

Amazon DocumentDB events originating from a cluster

Category	Description
creation	Cluster created
deletion	Cluster deleted.
failover	Promoting previous primary again.
failover	Completed failover to instance.
failover	Started failover to DB instance: %s

Category	Description
failover	Started same AZ failover to DB instance: %s
failover	Started cross AZ failover to DB instance: %s
maintenance	Cluster has been patched.
maintenance	Database cluster is in a state that cannot be upgraded: %s
notification	The cluster stopped.
notification	The cluster started.
notification	The cluster stop failed.
notification	The cluster is being started due to it exceeding the maximum allowed time being stopped.
notification	Renamed cluster from %s to %s.

Amazon DocumentDB events originating from cluster snapshot

The following table shows the event category and a list of events when an Amazon DocumentDB cluster snapshot is the source type.

Category	Description
backup	Creating manual cluster snapshot.
backup	Manual cluster snapshot created.
backup	Creating automated cluster snapshot.
backup	Automated cluster snapshot created.

Amazon DocumentDB events originating from parameter group

The following table shows the event category and a list of events when a parameter group is the source type.

Category	Description
configuration change	Updated parameter %s to %s with apply method %s

Monitoring Amazon DocumentDB with CloudWatch

Amazon DocumentDB (with MongoDB compatibility) integrates with Amazon CloudWatch so that you can gather and analyze operational metrics for your clusters. You can monitor these metrics using the CloudWatch console, the Amazon DocumentDB console, the AWS Command Line Interface (AWS CLI), or the CloudWatch API.

CloudWatch also lets you set alarms so that you can be notified if a metric value breaches a threshold that you specify. You can even set up Amazon CloudWatch Events to take corrective action if a breach occurs. For more information about using CloudWatch and alarms, see the Amazon CloudWatch documentation.

Topics

- Amazon DocumentDB metrics
- Viewing CloudWatch data
- Amazon DocumentDB dimensions
- Monitoring Opcounter metrics
- Monitoring database connections

Amazon DocumentDB metrics

To monitor the health and performance of your Amazon DocumentDB cluster and instances, you can view the following metrics in the Amazon DocumentDB console.

> **Note**
>
> Metrics in the following tables apply to both instance-based and elastic clusters.

Topics

- Resource utilization metrics
- Latency metrics
- NVMe-backed instance metrics
- Operations metrics
- Throughput metrics
- System metrics
- T3 instance metrics

Resource utilization metrics

Metric	Description	
`BackupRetentionPeriodStorageUsed`	The total amount of backup storage in bytes used to support the point-in-time restore feature within the Amazon DocumentDB's retention window. Included in the total reported by the `TotalBackupStorageBilled` metric. Computed separately for each Amazon DocumentDB cluster.	
`ChangeStreamLogSize`	The amount of storage used by your cluster to store the change stream log in megabytes. This value is a subset of the total storage	

Metric	Description
	for the cluster (`VolumeBytesUsed`) and affects the cost of the cluster. For storage pricing information, see the Amazon DocumentDB product page. The change stream log size is a function of how much change is happening on your cluster and the change stream long retention duration. For more information on change streams, see Using change streams with Amazon DocumentDB.
`CPUUtilization`	The percentage of CPU used by an instance.
`DatabaseConnections`	The number of connections open on an instance taken at a 1-minute frequency.
`DatabaseConnectionsMax`	The maximum number of open database connections on an instance in a 1-minute period.
`DatabaseCursors`	The number of cursors open on an instance taken at a 1-minute frequency.
`DatabaseCursorsMax`	The maximum number of open cursors on an instance in a 1-minute period.

Metric	Description
`DatabaseCursorsTimedOut`	The number of cursors that timed out in a 1-minute period.
`FreeableMemory`	The amount of available random access memory, in bytes.
`FreeLocalStorage`	This metric reports the amount of storage available to each instance for temporary tables and logs. This value depends on the instance class. You can increase the amount of free storage space for an instance by choosing a larger instance class for your instance.
`LowMemThrottleQueueDepth`	The queue depth for requests that are throttled due to low available memory taken at a 1-minute frequency.
`LowMemThrottleMaxQueueDepth`	The maximum queue depth for requests that are throttled due to low available memory in a 1-minute period.
`LowMemNumOperationsThrottled`	The number of requests that are throttled due to low available memory in a 1-minute period.

Metric	Description
`SnapshotStorageUsed`	The total amount of backup storage in bytes consumed by all snapshots for a given Amazon DocumentDB cluster outside its backup retention window. Included in the total reported by the `TotalBackupStorageBilled` metric. Computed separately for each Amazon DocumentDB cluster.
`SwapUsage`	The amount of swap space used on the instance.
`TotalBackupStorageBilled`	The total amount of backup storage in bytes for which you are billed for a given Amazon DocumentDB cluster. Includes the backup storage measured by the `BackupRetentionPeriodStorageUsed` and `SnapshotStorageUsed` metrics. Computed separately for each Amazon DocumentDB cluster.
`TransactionsOpen`	The number of transactions open on an instance taken at a 1-minute frequency.
`TransactionsOpenMax`	The maximum number of transactions open on an instance in a 1-minute period.

Metric	Description
`VolumeBytesUsed`	The amount of storage used by your cluster in bytes. This value affects the cost of the cluster. For pricing information, see the Amazon DocumentDB product page.

Latency metrics

Metric	Description
`DBClusterReplicaLagMaximum`	The maximum amount of lag, in milliseconds, between the primary instance and each Amazon DocumentDB instance in the cluster.
`DBClusterReplicaLagMinimum`	The minimum amount of lag, in milliseconds, between the primary instance and each replica instance in the cluster.
`DBInstanceReplicaLag`	The amount of lag, in milliseconds, when replicating updates from the primary instance to a replica instance.
`ReadLatency`	The average amount of time taken per disk I/O operation.
`WriteLatency`	The average amount of time, in milliseconds, taken per disk I/O operation.

NVMe-backed instance metrics

Metric	Description
`NVMeStorageCacheHitRatio`	The percentage of requests that are served by the tiered cache.
`FreeNVMeStorage`	The amount of available Ephemeral NVMe storage.
`ReadIOPSNVMeStorage`	The average number of disk read I/O operations to Ephemeral NVMe storage.
`ReadLatencyNVMeStorage`	The average amount of time taken per disk read I/O operation for Ephemeral NVMe storage.
`ReadThroughputNVMeStorage`	The average number of bytes read from disk per second for Ephemeral NVMe storage.
`WriteIOPSNVMeStorage`	The average number of disk write I/O operations to Ephemeral NVMe storage.
`WriteLatencyNVMeStorage`	The average amount of time taken per disk write I/O operation for Ephemeral NVMe storage.
`WriteThroughputNVMeStorage`	The average number of bytes written to disk per second for Ephemeral NVMe storage.

Operations metrics

Metric	Description
`DocumentsDeleted`	The number of deleted documents in a 1-minute period.
`DocumentsInserted`	The number of inserted documents in a 1-minute period.
`DocumentsReturned`	The number of returned documents in a 1-minute period.
`DocumentsUpdated`	The number of updated documents in a 1-minute period.
`OpcountersCommand`	The number of commands issued in a 1-minute period.
`OpcountersDelete`	The number of delete operations issued in a 1-minute period.
`OpcountersGetmore`	The number of getmores issued in a 1-minute period.
`OpcountersInsert`	The number of insert operations issued in a 1-minute period.
`OpcountersQuery`	The number of queries issued in a 1-minute period.

Metric	Description
`OpcountersUpdate`	The number of update operations issued in a 1-minute period.
`TransactionsStarted`	The number of transactions started on an instance in a 1-minute period.
`TransactionsCommitted`	The number of transactions committed on an instance in a 1-minute period.
`TransactionsAborted`	The number of transactions aborted on an instance in a 1-minute period.
`TTLDeletedDocuments`	The number of documents deleted by a TTLMonitor in a 1-minute period.

Throughput metrics

Metric	Description
`NetworkReceiveThroughput`	The amount of network throughput, in bytes per second, received from clients by each instance in the cluster. This throughput doesn't include network traffic between instances in the cluster and the cluster volume.

Metric	Description
`NetworkThroughput`	The amount of network throughput, in bytes per second, both received from and transmitted to clients by each instance in the Amazon DocumentDB cluster. This throughput doesn't include network traffic between instances in the cluster and the cluster volume.
`NetworkTransmitThroughput`	The amount of network throughput, in bytes per second, sent to clients by each instance in the cluster. This throughput doesn't include network traffic between instances in the cluster and the cluster volume.
`ReadIOPS`	The average number of disk read I/O operations per second. Amazon DocumentDB reports read and write IOPS separately, and on 1-minute intervals.
`ReadThroughput`	The average number of bytes read from disk per second.

Metric	Description
`StorageNetworkReceiveThroughput`	The amount of network throughput, in bytes per second, received from the Amazon DocumentDB cluster storage volume by each instance in the cluster.
`StorageNetworkTransmitThroughput`	The amount of network throughput, in bytes per second, sent to the Amazon DocumentDB cluster storage volume by each instance in the cluster.
`StorageNetworkThroughput`	The amount of network throughput, in bytes per second, received and sent to the Amazon DocumentDB cluster storage volume by each instance in the Amazon DocumentDB cluster.

Metric	Description	
`VolumeReadIOPs`	The average number of billed read I/O operations from a cluster volume, reported at 5-minute intervals. Billed read operations are calculate d at the cluster volume level, aggregated from all instances in the cluster, and then reported at 5-minute intervals. The value is calculated by taking the value of the read operations metric over a 5-minute period. You can determine the amount of billed read operations per second by taking the value of the billed read operation s metric and dividing by 300 seconds. For example, if the `VolumeReadIOPs` returns 13,686, then the billed read operations per second is 45 (13,686 / 300 = 45.62). You accrue billed read operations for queries that request database pages that are not present in the buffer cache and therefore must be loaded from storage. You might see spikes in billed read operations as query results are read from storage and	

Metric	Description	
	then loaded into the buffer cache.	

Metric	Description	
`VolumeWriteIOPs`	The average number of billed write I/O operations from a cluster volume, reported at 5-minute intervals. Billed write operations are calculate d at the cluster volume level, aggregated from all instances in the cluster, and then reported at 5-minute intervals. The value is calculated by taking the value of the write operations metric over a 5-minute period. You can determine the amount of billed write operations per second by taking the value of the billed write operation s metric and dividing by 300 seconds. For example, if the `VolumeWriteIOPs` returns 13,686, then the billed write operations per second is 45 (13,686 / 300 = 45.62). Note that `VolumeReadIOPs` and `VolumeWriteIOPs` metrics are calculated by the DocumentDB storage layer and it includes IOs performed by the primary and replica instances. The data is aggregated every 20-30 minutes and then reported	

Metric	Description
	at 5-minute intervals, thus emitting the same data point for the metric in the time period. If you are looking for a metric to correlate to your insert operations over a 1-minute interval, you can use the instance level WriteIOPS metric. The metric is available in the monitoring tab of your Amazon DocumentDB primary instance.
`WriteIOPS`	The average number of disk write I/O operations per second. When used on a cluster level, `WriteIOPs` are evaluated across all instances in the cluster. Read and write IOPS are reported separately, on 1-minute intervals.
`WriteThroughput`	The average number of bytes written to disk per second.

System metrics

Metric	Description
`BufferCacheHitRatio`	The percentage of requests that are served by the buffer cache.

Metric	Description
`DiskQueueDepth`	The number of I/O operations that are waiting to be written to or read from disk.
`EngineUptime`	The amount of time, in seconds, that the instance has been running.
`IndexBufferCacheHitRatio`	The percentage of index requests that are served by the buffer cache. You might see a spike greater than 100 percent for the metric right after you drop an index, collection or database. This will automatically be corrected after 60 seconds. This limitation will be fixed in a future patch update.

T3 instance metrics

Metric	Description
`CPUCreditUsage`	The number of CPU credits spent during the measurement period.
`CPUCreditBalance`	The number of CPU credits that an instance has accrued. This balance is depleted when the CPU bursts and CPU credits are spent more quickly than they are earned.

Metric	Description
`CPUSurplusCreditBalance`	The number of surplus CPU credits spent to sustain CPU performance when the CPUCreditBalance value is zero.
`CPUSurplusCreditsCharged`	The number of surplus CPU credits exceeding the maximum number of CPU credits that can be earned in a 24-hour period, and thus attracting an additional charge. For more information, see Monitoring your CPU credits.

Viewing CloudWatch data

You can view Amazon CloudWatch data using the CloudWatch console, the Amazon DocumentDB console, AWS Command Line Interface (AWS CLI), or the CloudWatch API.

Using the AWS Management Console

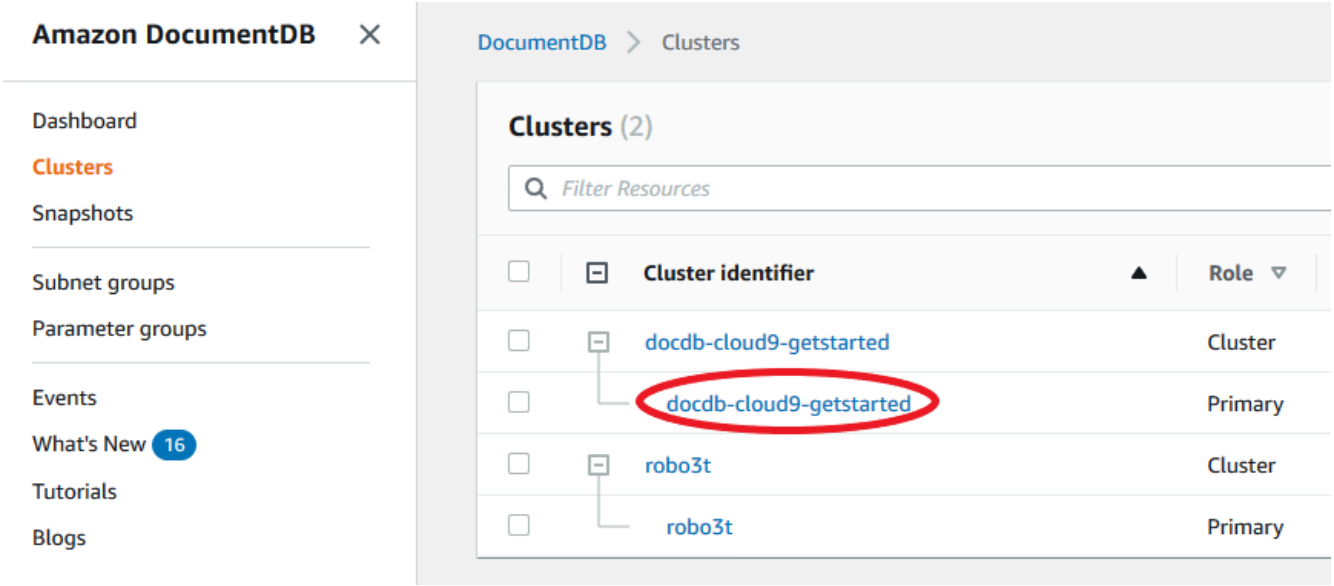
To view CloudWatch metrics using the Amazon DocumentDB Management Console, complete the following steps.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**.

> **Tip**
> If you don't see the navigation pane on the left side of your screen, choose the menu icon

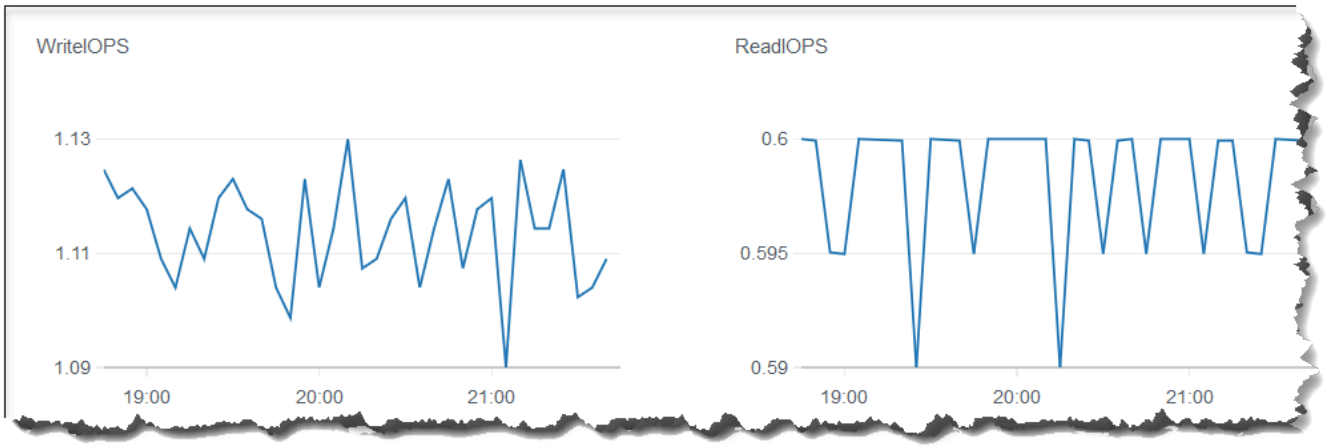
(≡)
in the upper-left corner of the page.

3. In the Clusters navigation box, you'll see the column **Cluster Identifier**. Your instances are listed under clusters, similar to the screenshot below.



4. From the list of instances, choose the name of the instance that you want metrics for.
5. In the resulting instance summary page, choose the **Monitoring** tab to view graphical representations of your Amazon DocumentDB instance's metrics. Because a graph must be generated for each metric, it might take a few minutes for the **CloudWatch** graphs to populate.

   The following image shows the graphical representations of two CloudWatch metrics in the Amazon DocumentDB console, `WriteIOPS` and `ReadIOPS`.



Using the CloudWatch Management Console

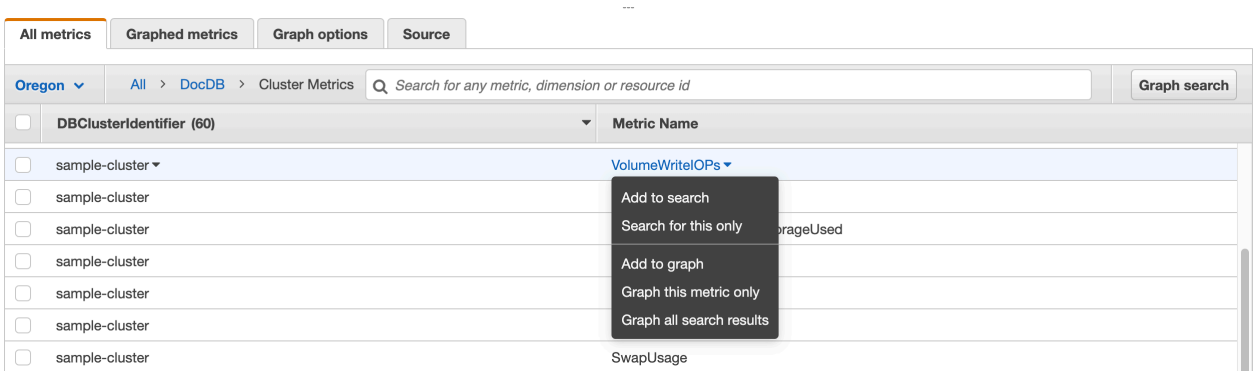
To view CloudWatch metrics using the CloudWatch Management Console, complete the following steps.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/cloudwatch.
2. In the navigation pane, choose **Metrics**. Then, from the list of service names, choose **DocDB**.
3. Choose a metric dimension (for example, **Cluster Metrics**).
4. The **All metrics** tab displays all metrics for that dimension in **DocDB**.

   a. To sort the table, use the column heading.

   b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

   c. To filter by metric, hover over the metric name and select the dropdown arrow next to the metric name. Then, choose **Add to search**, as shown in the image below.



Using the AWS CLI

To view CloudWatch data for Amazon DocumentDB, use the CloudWatch `get-metric-statistics` operation with the following parameters.

Parameters

- **`--namespace`** — Required. The service namespace for which you want CloudWatch metrics. For Amazon DocumentDB, this must be `AWS/DocDB`.
- **`--metric-name`** — Required. The name of the metric for which you want data.
- **`--start-time`** — Required. The timestamp that determines the first data point to return.

The value specified is inclusive; results include data points with the specified timestamp. The timestamp must be in ISO 8601 UTC format (for example, 2016-10-03T23:00:00Z).

- **`--end-time`** — Required. The timestamp that determines the last data point to return.

  The value specified is inclusive; results include data points with the specified timestamp. The timestamp must be in ISO 8601 UTC format (for example, 2016-10-03T23:00:00Z).

- **`--period`** — Required. The granularity, in seconds, of the returned data points. For metrics with regular resolution, a period can be as short as one minute (60 seconds) and must be a multiple of 60. For high-resolution metrics that are collected at intervals of less than one minute, the period can be 1, 5, 10, 30, 60, or any multiple of 60.
- **`--dimensions`** — Optional. If the metric contains multiple dimensions, you must include a value for each dimension. CloudWatch treats each unique combination of dimensions as a separate metric. If a specific combination of dimensions was not published, you can't retrieve statistics for it. You must specify the same dimensions that were used when the metrics were created.
- **`--statistics`** — Optional. The metric statistics, other than percentile. For percentile statistics, use `ExtendedStatistics`. When calling `GetMetricStatistics`, you must specify either `Statistics` or `ExtendedStatistics`, but not both.

  Permitted values:

  - `SampleCount`
  - `Average`
  - `Sum`
  - `Minimum`
  - `Maximum`

- **`--extended-statistics`** — Optional. The `percentile` statistics. Specify values between p0.0 and p100. When calling `GetMetricStatistics`, you must specify either `Statistics` or `ExtendedStatistics`, but not both.
- **`--unit`** — Optional. The unit for a given metric. Metrics may be reported in multiple units. Not supplying a unit results in all units being returned. If you specify only a unit that the metric does not report, the results of the call are null.

  Possible values:

  - `Seconds`

- `Microseconds`
- `Milliseconds`
- `Bytes`
- `Kilobytes`
- `Megabytes`
- `Gigabytes`
- `Terabytes`
- `Bits`
- `Kilobytes`
- `Megabits`
- `Gigabits`
- `Terabits`
- `Percent`
- `Count`
- `Bytes/Second`
- `Kilobytes/Second`
- `Megabytes/Second`
- `Gigabytes/Second`
- `Terabytes/Second`
- `Bits/Second`
- `Kilobits/Second`
- `Megabits/Second`
- `Gigabits/Second`
- `Terabits/Second`
- `Count/Second`
- `None`

Example

The following example finds the maximum `CPUUtilization` for a 2-hour period taking a sample every 60 seconds.

For Linux, macOS, or Unix:

```
aws cloudwatch get-metric-statistics \
       --namespace AWS/DocDB \
       --dimensions \
           Name=DBInstanceIdentifier,Value=docdb-2019-01-09-23-55-38 \
       --metric-name CPUUtilization \
       --start-time 2019-02-11T05:00:00Z \
       --end-time 2019-02-11T07:00:00Z \
       --period 60 \
       --statistics Maximum
```

For Windows:

```
aws cloudwatch get-metric-statistics ^
       --namespace AWS/DocDB ^
       --dimensions ^
           Name=DBInstanceIdentifier,Value=docdb-2019-01-09-23-55-38 ^
       --metric-name CPUUtilization ^
       --start-time 2019-02-11T05:00:00Z ^
       --end-time 2019-02-11T07:00:00Z ^
       --period 60 ^
       --statistics Maximum
```

Output from this operation look something like the following.

```
{
       "Label": "CPUUtilization",
       "Datapoints": [
           {
               "Unit": "Percent",
               "Maximum": 4.49152542374361,
               "Timestamp": "2019-02-11T05:51:00Z"
           },
           {
               "Unit": "Percent",
               "Maximum": 4.25000000000485,
               "Timestamp": "2019-02-11T06:44:00Z"
           },

           ********* some output omitted for brevity *********

           {
```

```
                "Unit": "Percent",
                "Maximum": 4.33333333331878,
                "Timestamp": "2019-02-11T06:07:00Z"
            }
        ]
    }
```

Amazon DocumentDB dimensions

The metrics for Amazon DocumentDB are qualified by the values for the account or operation. You can use the CloudWatch console to retrieve Amazon DocumentDB data filtered by any of the dimensions in the following table.

Dimension	Description
`DBClusterIdentifier`	Filters the data that you request for a specific Amazon DocumentDB cluster.
`DBClusterIdentifier, Role`	Filters the data that you request for a specific Amazon DocumentDB cluster, aggregating the metric by instance role (WRITER/READER). For example, you can aggregate metrics for all READER instances that belong to a cluster.
`DBInstanceIdentifier`	Filters the data that you request for a specific database instance.

Monitoring Opcounter metrics

Opcounter metrics have a non-zero value (usually ~50) for idle clusters. This is because Amazon DocumentDB performs periodic health checks, internal operations, and metrics collection tasks.

Monitoring database connections

When you view the number of connections by using database engine commands such as `db.runCommand( { serverStatus: 1 })`, you might see up to 10 more connections than you see in `DatabaseConnections` through CloudWatch. This occurs because Amazon DocumentDB performs periodic health checks and metrics collection tasks that don't get accounted for in

`DatabaseConnections`. `DatabaseConnections` represents customer-initiated connections only.

Logging Amazon DocumentDB API calls with AWS CloudTrail

Amazon DocumentDB (with MongoDB compatibility) is integrated with AWS CloudTrail, a service that provides a record of actions taken by users, roles, or an AWS service in Amazon DocumentDB (with MongoDB compatibility). CloudTrail captures all AWS CLI API calls for Amazon DocumentDB as events, including calls from the Amazon DocumentDB console and from code calls to the Amazon DocumentDB SDK. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon DocumentDB. If you don't configure a trail, you can still view the most recent events on the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon DocumentDB (with MongoDB compatibility), the IP address from which the request was made, who made the request, when it was made, and other details.

> **Important**
>
> For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS). Amazon DocumentDB console, AWS CLI, and API calls are logged as calls made to the Amazon RDS API.

To learn more about AWS CloudTrail, see AWS CloudTrail User Guide.

Amazon DocumentDB information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon DocumentDB (with MongoDB compatibility), that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon DocumentDB (with MongoDB compatibility), create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further

analyze and act upon the event data collected in CloudTrail logs. For more information, see the following topics in the *AWS CloudTrail User Guide*:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions
- Receiving CloudTrail Log Files from Multiple Accounts

Every event or log entry includes information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Profiling Amazon DocumentDB operations

You can use the profiler in Amazon DocumentDB (with MongoDB compatibility) to log the execution time and details of operations that were performed on your cluster. The profiler is useful for monitoring the slowest operations on your cluster to help you improve individual query performance and overall cluster performance.

By default, the profiler feature is disabled. When enabled, the profiler logs operations that are taking longer than a customer-defined threshold value (for example, 100 ms) to Amazon CloudWatch Logs. Logged details include the profiled command, time, plan summary, and client metadata. After the operations are logged to CloudWatch Logs, you can use CloudWatch Logs Insights to analyze, monitor, and archive your Amazon DocumentDB profiling data. Common queries are provided in the section Common queries.

When enabled, the profiler uses additional resources in your cluster. We recommend that you start with a high threshold value (for example, 500 ms) and gradually lower the value to identify slow operations. Starting with a threshold value of 50 ms can cause performance issues on your cluster for high throughput applications. The profiler is enabled at the cluster level and works on all

instances and databases in a cluster. Amazon DocumentDB logs operations to Amazon CloudWatch Logs on a best-effort basis.

Although Amazon DocumentDB imposes no additional charge to enable the profiler, you are charged the standard rates for the usage of CloudWatch Logs. For information about CloudWatch Logs pricing, see Amazon CloudWatch pricing.

Topics

- Supported operations
- Limitations
- Enabling the Amazon DocumentDB profiler
- Disabling the Amazon DocumentDB profiler
- Disabling profiler logs export
- Accessing your Amazon DocumentDB profiler logs
- Common queries

Supported operations

Amazon DocumentDB profiler supports the following operations:

- `aggregate`
- `count`
- `delete`
- `distinct`
- `find` (OP_QUERY and command)
- `findAndModify`
- `insert`
- `update`

Limitations

The slow query profiler is only able to emit profiler logs if the entire result set of the query is able to fit in one batch, and if the result set is under 16MB (maximum BSON size). Result sets greater than 16MB are automatically split into multiple batches.

Most drivers or shells may set a default batch size that is small. You can specify the batch size as part of your query. For the purpose of capturing slow query logs, we recommend a batch size that exceeds the size of your expected result set. If you are unsure of the result set size, or if it varies, you can also set the batch size to a large number (e.g., 100k).

However, using a larger batch size means more results will have to be retrieved from the database before a response is sent to the client. For some queries, that may create longer delays before you get results. If you do not plan to consume the entire result set, it is possible that you will spend more I/Os to process the query and throw away the result.

Enabling the Amazon DocumentDB profiler

Enabling the profiler on a cluster is a three-step process. Ensure that all steps are completed, or profiling logs will not be sent to CloudWatch Logs. Profiler is set at the cluster level and is performed on all of the cluster's databases and instances.

To enable the profiler on a cluster

1. Because you can't modify a default cluster parameter group, ensure that you have an available custom cluster parameter group. For more information, see Creating Amazon DocumentDB cluster parameter groups.
2. Using an available custom cluster parameter group, modify the following parameters: `profiler`, `profiler_threshold_ms`, and `profiler_sampling_rate`. For more information, see Modifying Amazon DocumentDB cluster parameter groups.
3. Create or modify your cluster to use the custom cluster parameter group and to enable exporting `profiler` logs to CloudWatch Logs.

The following sections show how to implement these steps using the AWS Management Console and the AWS Command Line Interface (AWS CLI).

Using the AWS Management Console

1. Before you begin, create a Amazon DocumentDB cluster and a custom cluster parameter group if you don't already have one. For more information, see Creating Amazon DocumentDB cluster parameter groups and Creating an Amazon DocumentDB cluster.
2. Using an available custom cluster parameter group, modify the following parameters. For more information, see Modifying Amazon DocumentDB cluster parameter groups.

- `profiler` — Enables or disables query profiling. Permitted values are `enabled` and `disabled`. The default value is `disabled`. To enable profiling, set the value to `enabled`.
- `profiler_threshold_ms` — When `profiler` is set to `enabled`, all commands that are taking longer than `profiler_threshold_ms` are logged to CloudWatch. Permitted values are `[50-INT_MAX]`. The default value is `100`.
- `profiler_sampling_rate` — The fraction of slow operations that should be profiled or logged. Permitted values are `[0.0-1.0]`. The default value is `1.0`.

3. Modify your cluster to use the custom cluster parameter group and set the profiler log exports to publish to Amazon CloudWatch.

   a. In the navigation pane, choose **Clusters** to add your custom parameter group to a cluster.

   b. Choose the button to the left of the name of the cluster that you want to associate your parameter group with. Select **Actions**, and then **Modify** to modify your cluster.

   c. Under **Cluster options**, choose the custom parameter group from the step above to add it to your cluster.

   d. Under **Log exports**, select **Profiler logs** to publish to Amazon CloudWatch.

   e. Choose **Continue** to view a summary of your modifications.

   f. After verifying your changes, you can apply them immediately or during the next maintenance window under **Scheduling of modifications** .

   g. Choose **Modify cluster** to update your cluster with your new parameter group.

Using the AWS CLI

The following procedure enables the profiler on all supported operations for the cluster `sample-cluster`.

1. Before you begin, ensure that you have an available custom cluster parameter group by running the following command, and reviewing the output for a cluster parameter group that doesn't have `default` in the name and has `docdb3.6` as the parameter group family. If you don't have a non-default cluster parameter group, see Creating Amazon DocumentDB cluster parameter groups.

```
aws docdb describe-db-cluster-parameter-groups \
```

```
    --query 'DBClusterParameterGroups[*].
[DBClusterParameterGroupName,DBParameterGroupFamily]'
```

In the following output, only `sample-parameter-group` meets both criteria.

```
[
       [
           "default.docdb3.6",
           "docdb3.6"
       ],
       [
           "sample-parameter-group",
           "docdb3.6"
       ]
]
```

2. Using your custom cluster parameter group, modify the following parameters:

   - `profiler` — Enables or disables query profiling. Permitted values are `enabled` and `disabled`. The default value is `disabled`. To enable profiling, set the value to `enabled`.
   - `profiler_threshold_ms` — When `profiler` is set to `enabled`, all commands taking longer than `profiler_threshold_ms` are logged to CloudWatch. Permitted values are `[50-INT_MAX]`. The default value is `100`.
   - `profiler_sampling_rate` — The fraction of slow operations that should be profiled or logged. Permitted values are `[0.0-1.0]`. The default value is `1.0`.

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=enabled,ApplyMethod=immediate \

 ParameterName=profiler_threshold_ms,ParameterValue=100,ApplyMethod=immediate \

 ParameterName=profiler_sampling_rate,ParameterValue=0.5,ApplyMethod=immediate
```

3. Modify your Amazon DocumentDB cluster so that it uses the `sample-parameter-group` custom cluster parameter group from the previous step and sets the parameter `--enable-cloudwatch-logs-exports` to `profiler`.

The following code modifies the cluster `sample-cluster` to use the `sample-parameter-group` from the previous step, and adds `profiler` to the enabled CloudWatch Logs exports.

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --db-cluster-parameter-group-name sample-parameter-group \
       --cloudwatch-logs-export-configuration '{"EnableLogTypes":["profiler"]}'
```

Output from this operation looks something like the following.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "available",
        "EarliestRestorableTime": "2020-04-07T02:05:12.479Z",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "3.6.0",
        "LatestRestorableTime": "2020-04-08T22:08:59.317Z",
        "Port": 27017,
        "MasterUsername": "test",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
```

```
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-
cluster",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ],
        "DeletionProtection": true
    }
}
```

Disabling the Amazon DocumentDB profiler

To disable the profiler, you disable both the `profiler` parameter and the export of `profiler` logs to CloudWatch Logs.

Disabling the Profiler

You can disable the `profiler` parameter using either the AWS Management Console or AWS CLI, as follows.

Using the AWS Management Console

The following procedure uses the AWS Management Console to disable Amazon DocumentDB `profiler`.

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Parameter groups**. Then choose the name of the cluster parameter group that you want to disable the profiler on.
3. In the resulting **Cluster parameters** page, select the button to the left of the `profiler` parameter and choose **Edit**.
4. In the **Modify profiler** dialog box, choose `disabled` in the list.
5. Choose **Modify cluster parameter**.

Using the AWS CLI

To disable `profiler` on a cluster using the AWS CLI, modify the cluster as follows.

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=disabled,ApplyMethod=immediate
```

Disabling profiler logs export

You can disable exporting `profiler` logs to CloudWatch Logs by using either the AWS Management Console or AWS CLI, as follows.

Using the AWS Management Console

The following procedure uses the AWS Management Console to disable Amazon DocumentDB exporting logs to CloudWatch.

1. Open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Clusters**. Choose the button to the left of the name of the cluster for which you want to disable exporting logs.
3. On the **Actions** menu, choose **Modify**.

4. Scroll down to the **Log exports** section and unselect **Profiler logs**.
5. Choose **Continue**.
6. Review your changes, and then choose when you want this change applied to your cluster:
   - **Apply during the next scheduled maintenance window**
   - **Apply immediately**
7. Choose **Modify cluster**.

Using the AWS CLI

The following code modifies the cluster `sample-cluster` and disables CloudWatch profiler logs.

Example

For Linux, macOS, or Unix:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

For Windows:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

Output from this operation looks something like the following.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "available",
```

```
        "EarliestRestorableTime": "2020-04-08T02:05:17.266Z",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "3.6.0",
        "LatestRestorableTime": "2020-04-09T05:14:44.356Z",
        "Port": 27017,
        "MasterUsername": "test",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "DeletionProtection": true
    }
}
```

Accessing your Amazon DocumentDB profiler logs

Follow these steps to access your profile logs on Amazon CloudWatch.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Make sure that you are in the same Region as your Amazon DocumentDB cluster.
3. In the navigation pane, choose **Logs**.
4. To find the profiler logs for your cluster, in the list, choose `/aws/docdb/yourClusterName/profiler`.

   The profile logs for each of your instances are available under each of the respective instance names.

Common queries

The following are some common queries you can use to analyze your profiled commands. For more information about CloudWatch Logs Insights, see Analyzing Log Data with CloudWatch Logs Insights and Sample Queries.

Get the 10 slowest operations on a specified collection

```
filter ns="test.foo" | sort millis desc | limit 10
```

Get all the update operations on a collection that took more than 60 ms

```
filter millis > 60 and op = "update"
```

Get the 10 slowest operations in the last month

```
sort millis desc | limit 10
```

Get all the queries with a COLLSCAN plan summary

```
filter planSummary="COLLSCAN"
```

Monitoring with Performance Insights

Performance Insights adds to the existing Amazon DocumentDB monitoring features to illustrate your cluster performance and help you analyze any issues that affect it. With the Performance Insights dashboard, you can visualize the database load and filter the load by waits, query statements, hosts, or application.

> **Note**
>
> Performance Insights is only available for Amazon DocumentDB 3.6, 4.0, and 5.0 instance-based clusters.

How is it useful?

- Visualize database performance — Visualize the load to determine when and where the load is on the database
- Determine what is causing load on database — Determine which queries, hosts, and applications are contributing to the load on your instance
- Determine when there is load on your database — Zoom in on the Performance Insights dashboard to focus on specific events or zoom out to look at trends across a larger time span
- Alert on database load — Access new database load metrics automatically from CloudWatch where you can monitor the DB load metrics alongside other Amazon DocumentDB metrics and set alerts on them

What are the limitations of Amazon DocumentDB Performance Insights?

- Performance Insights in the AWS GovCloud (US-East) and AWS GovCloud (US-West) regions are not available
- Performance Insights for Amazon DocumentDB retains up to 7 days of performance data
- Queries longer than 1,024 bytes are not aggregated in Performance Insights

Topics

- Performance Insights concepts
- Enabling and disabling Performance Insights

- Configuring access policies for Performance Insights
- Analyzing metrics with the Performance Insights dashboard
- Retrieving metrics with the Performance Insights API
- Amazon CloudWatch metrics for Performance Insights
- Performance Insights for counter metrics

Performance Insights concepts

Topics

- Average active sessions
- Dimensions
- Max vCPU

Average active sessions

Database load (DB load) measures the level of activity in your database. The key metric in Performance Insights is DB Load, which is collected every second. The unit for the DBLoad metric is the *Average Active Sessions (AAS)* for an Amazon DocumentDB instance.

An *active* session is a connection that has submitted work to the Amazon DocumentDB instance and is waiting for a response. For example, if you submit a query to an Amazon DocumentDB instance, the database session is active while the instance is processing the query.

To obtain the average active sessions, Performance Insights samples the number of sessions concurrently running a query. The AAS is the total number of sessions divided by the total number of samples. The following table shows five consecutive samples of a running query.

Sample	Number of sessions running query	AAS	Calculation
1	2	2	2 sessions / 1 sample
2	0	1	2 sessions / 2 samples

Sample	Number of sessions running query	AAS	Calculation
3	4	2	6 sessions / 3 samples
4	0	1.5	6 sessions / 4 samples
5	4	2	10 sessions / 5 samples

In the preceding example, the DB Load for the time interval from 1-5 is 2 AAS. An increase in DB load means that, on average, more sessions are running on the database.

Dimensions

The DB Load metric is different from the other time-series metrics because you can break it into subcomponents called dimensions. You can think of dimensions as categories for the different characteristics of the DB Load metric. When you are diagnosing performance issues, the most useful dimensions are **wait states** and **top query**.

wait states

A *wait state* causes a query statement to wait for a specific event to happen before it can continue running. For example, a query statement might wait until a locked resource is unlocked. By combining DB Load with wait states, you can get a complete picture of the session state. Here are various Amazon DocumentDB wait states:

Amazon DocumentDB wait state	Wait State Description
Latch	The Latch wait state occurs when the session is waiting to page the buffer pool. Frequent paging in and out of the buffer pool can happen more often when there are frequent large queries being processed by the system, collection scans, or when the buffer pool is too small to handle the working set.

Amazon DocumentDB wait state	Wait State Description
CPU	The CPU wait state occurs when the session is waiting on CPU.
CollectionLock	The CollectionLock wait state occurs when the session is waiting to acquire a lock on the collection. These events occur when there are DDL operations on the collection.
DocumentLock	The DocumentLock wait state occurs when the session is waiting to acquire a lock on a document. High number of concurrent writes to the same document will contribute to more DocumentLock wait states on that document.
SystemLock	The SystemLock wait state occurs when the session is waiting on the system. This can occur when there are frequent long running queries, long running transactions, or high concurrency on the system.
IO	The IO wait state occurs when the session waiting on IO to complete.
BufferLock	The BufferLock wait state occurs when the session is waiting to acquire a lock on a shared page in the buffer. BufferLock wait states can be prolonged if other processes are holding open cursors on the requested pages.
LowMemThrottle	The LowMemThrottle wait state occurs when the session is waiting due to heavy memory pressure on the Amazon DocumentD B instance. If this state persists for a long time, consider scaling up the instance to provide additional memory. For more information, see Resource Governor.

Amazon DocumentDB wait state	Wait State Description
BackgroundActivity	The BackgroundActivity wait state occurs when the session is waiting on internal system processes.
Other	The Other wait state is an internal wait state. If this state persists for a long time, consider terminating this query. For more informati on, see How Do I Find and Terminate Long Running or Blocked Queries?

Top queries

Whereas wait states show bottlenecks, top queries show which queries are contributing the most to DB load. For example, many queries might be currently running on the database, but a single query might consume 99% of the DB load. In this case, the high load might indicate a problem with the query.

Max vCPU

In the dashboard, the **Database load** chart collects, aggregates, and displays session information. To see whether active sessions are exceeding the maximum CPU, look at their relationship to the **Max vCPU** line. The **Max vCPU** value is determined by the number of vCPU (virtual CPU) cores for your Amazon DocumentDB instance.

If the DB load is often above the **Max vCPU** line, and the primary wait state is CPU, the CPU is overloaded. In this case, you might want to throttle connections to the instance, tune any queries with a high CPU load, or consider a larger instance class. High and consistent instances of any wait state indicate that there might be bottlenecks or resource contention issues to resolve. This can be true even if the DB load doesn't cross the **Max vCPU** line.

Enabling and disabling Performance Insights

To use Performance Insights, enable it on your DB instance. You can disable it later if necessary. Enabling and disabling Performance Insights doesn't cause downtime, a reboot, or a failover.

The Performance Insights agent consumes limited CPU and memory on the DB host. When the DB load is high, the agent limits the performance impact by collecting data less frequently.

Enabling Performance Insights when creating a cluster

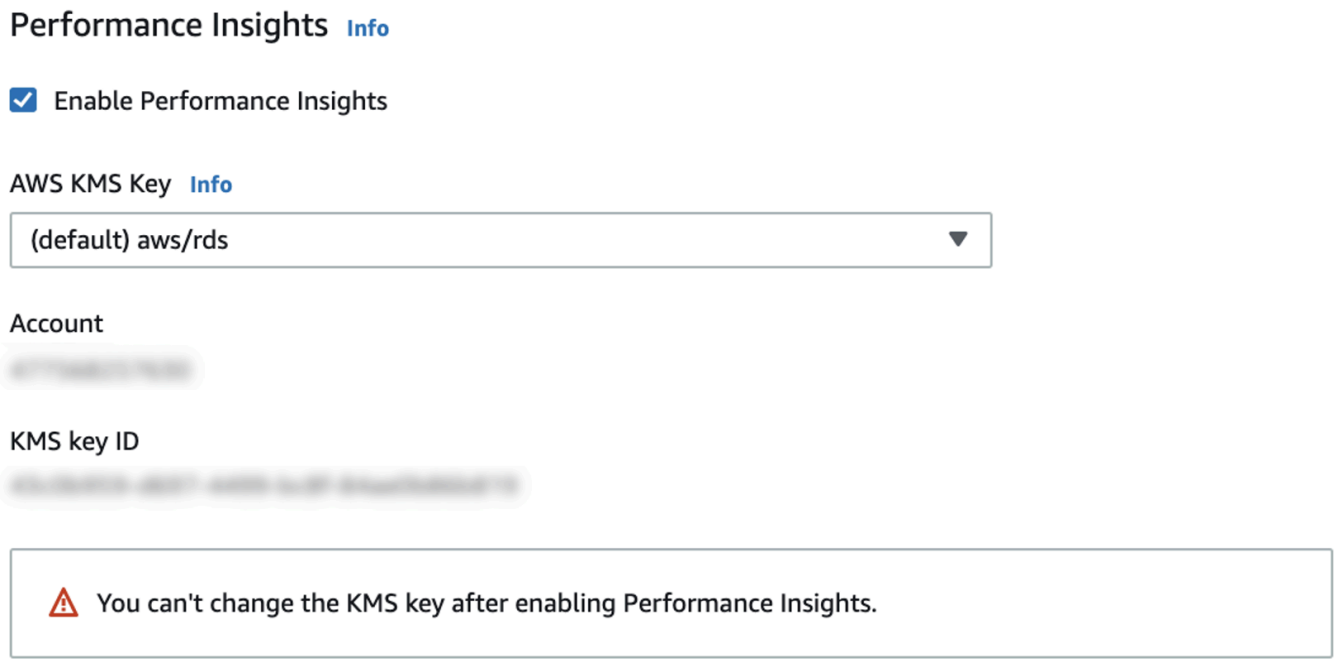
In the console, you can enable or disable Performance Insights when you create or modify a new DB instance.

Using the AWS Management Console

In the console, you can enable Performance Insights when you create an Amazon DocumentDB cluster. When you create a new Amazon DocumentDB cluster, enable Performance Insights by choosing **Enable Performance Insights** in the **Performance Insights** section.

Console instructions

1. To create a cluster, follow the instructions for Creating an Amazon DocumentDB cluster.
2. Select **Enable Performance Insights** in the Performance Insights section.



> **Note**
>
> The Performance Insights data retention period will be seven days.

AWS KMS key — Specify your AWS KMS key. Performance Insights encrypts all potentially sensitive data using your AWS KMS key. Data is encrypted in flight and at rest. For more information, see Configuring an AWS AWS KMS policy for Performance Insights.

Enabling and disabling when modifying an instance

You can modify a DB instance to enable or disable Performance Insights using the console or AWS CLI.

Using the AWS Management Console

Console instructions

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. Choose **Clusters**.
3. Choose a DB instance, and choose **Modify**.
4. In the Performance Insights section, choose either **Enable Performance Insights** or **Disable Performance Insights**.

   > **Note**
   >
   > If you choose **Enable Performance Insights**, you can specify your AWS AWS KMS key. Performance Insights encrypts all potentially sensitive data using your AWS KMS key. Data is encrypted in flight and at rest. For more information, see Encrypting Amazon DocumentDB Data at Rest.

5. Choose **Continue**.
6. For **Scheduling of Modifications**, choose **Apply immediately**. If you choose **Apply during the next scheduled maintenance window**, your instance ignores this setting and enables Performance Insights immediately.
7. Choose **Modify instance**.

Using the AWS CLI

When you use the `create-db-instance` or `modify-db-instance` AWS AWS CLI commands, you can enable Performance Insights by specifying `--enable-performance-insights`, or disable it by specifying `--no-enable-performance-insights`.

The following procedure describes how to enable or disable Performance Insights for a DB instance using the AWS AWS CLI.

AWS AWS CLI instructions

Call the `modify-db-instance` AWS AWS CLI command and provide the following values:

- `--db-instance-identifer` — The name of the DB instance
- `--enable-performance-insights` to enable or `--no-enable-performance-insights` to disable

Example

The following example enables Performance Insights for `sample-db-instance`:

For Linux, macOS, or Unix:

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights
```

For Windows:

```
aws docdb modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights
```

Configuring access policies for Performance Insights

To access Performance Insights, you must have the appropriate permissions from AWS Identity and Access Management (IAM). You have the following options for granting access:

- Attach the `AmazonRDSPerformanceInsightsReadOnly` managed policy to a permission set or role.
- Create a custom IAM policy and attach it to a permission set or role.

Also, if you specified a customer managed key when you turned on Performance Insights, make sure that users in your account have the `kms:Decrypt` and `kms:GenerateDataKey` permissions on the KMS key.

> **Note**
>
> For encryption-at-rest with AWS KMS keys and security groups management, Amazon DocumentDB leverages operational technology that is shared with Amazon RDS.

Attaching the AmazonRDSPerformanceInsightsReadOnly policy to an IAM principal

`AmazonRDSPerformanceInsightsReadOnly` is an AWS-managed policy that grants access to all read-only operations of the Amazon DocumentDB Performance Insights API. Currently, all operations in this API are read-only. If you attach `AmazonRDSPerformanceInsightsReadOnly` to a permission set or role, the recipient can use Performance Insights with other console features.

Creating a custom IAM policy for Performance Insights

For users who don't have the `AmazonRDSPerformanceInsightsReadOnly` policy, you can grant access to Performance Insights by creating or modifying a user-managed IAM policy. When you attach the policy to a permission set or role, the recipient can use Performance Insights.

To create a custom policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Create Policy** page, choose the JSON tab.
5. Copy and paste the following text, replacing *us-east-1* with the name of your AWS Region and *111122223333* with your customer account number.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBInstances",
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "rds:DescribeDBClusters",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:DescribeDimensionKeys",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetDimensionKeyDetails",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetResourceMetadata",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetResourceMetrics",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:ListAvailableResourceDimensions",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:ListAvailableResourceMetrics",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        }
    ]
}
```

6. Choose **Review policy**.
7. Provide a name for the policy and optionally a description, and then choose **Create policy**.

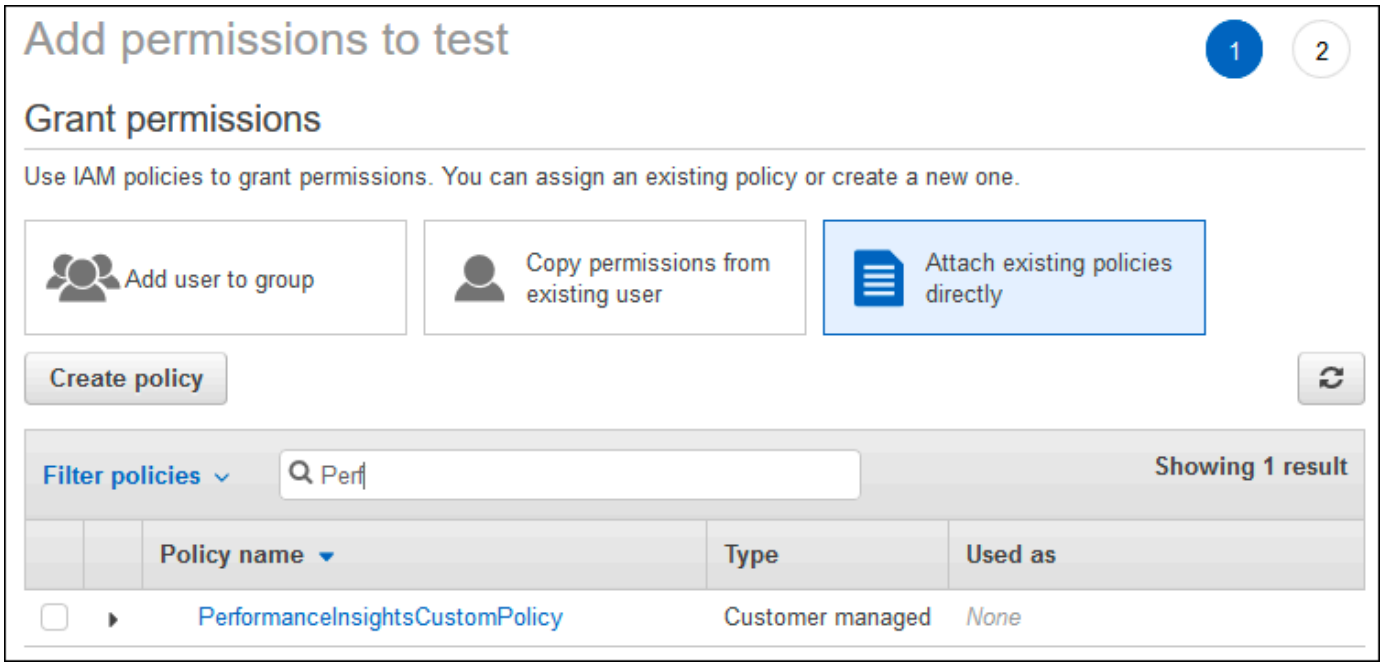
You can now attach the policy to a permission set or role. The following procedure assumes that you already have a user available for this purpose.

To attach the policy to a user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose an existing user from the list.

> **Important**
>
> To use Performance Insights, make sure that you have access to Amazon DocumentDB in addition to the custom policy. For example, the **AmazonDocDBReadOnlyAccess** predefined policy provides read-only access to Amazon DocDB.For more information, see Managing access using policies.

4. On the **Summary** page, choose **Add permissions**.
5. Choose **Attach existing policies directly**. For **Search**, type the first few characters of your policy name, as shown following.



6. Choose your policy, and then choose **Next: Review**.
7. Choose **Add permissions**.

Configuring an AWS KMS policy for Performance Insights

Performance Insights uses an AWS KMS key to encrypt sensitive data. When you enable Performance Insights through the API or the console, you have the following options:

- Choose the default AWS managed key.

  Amazon DocumentDB uses the AWS managed key for your new DB instance. Amazon DocumentDB creates an AWS managed key for your AWS account. Your AWS account has a different AWS managed key for Amazon DocumentDB for each AWS Region.

- Choose a customer managed key.

  If you specify a customer managed key, users in your account that call the Performance Insights API need the `kms:Decrypt` and `kms:GenerateDataKey` permissions on the KMS key. You can configure these permissions through IAM policies. However, we recommend that you manage these permissions through your KMS key policy. For more information, see Using key policies in AWS KMS.

Example

The following sample key policy shows how to add statements to your KMS key policy. These statements allow access to Performance Insights. Depending on how you use the AWS KMS, you might want to change some restrictions. Before adding statements to your policy, remove all comments.

```
{
 "Version" : "2012-10-17",
 "Id" : "your-policy",
 "Statement" : [ {
    //This represents a statement that currently exists in your policy.
 }
 ....,
 //Starting here, add new statement to your policy for Performance Insights.
 //We recommend that you add one new statement for every RDS/DocumentDB instance
 {
    "Sid" : "Allow viewing RDS Performance Insights",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            //One or more principals allowed to access Performance Insights
```

```
            "arn:aws:iam::444455556666:role/Role1"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition" :{
        "StringEquals" : {
            //Restrict access to only RDS APIs (including Performance Insights).
            //Replace *region* with your AWS Region.
            //For example, specify us-west-2.
            "kms:ViaService" : "rds.*region*.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
            //Restrict access to only data encrypted by Performance Insights.
            "kms:EncryptionContext:aws:pi:service": "rds",
            "kms:EncryptionContext:service": "pi",

            //Restrict access to a specific DocDB instance.
            //The value is a DbiResourceId.
           "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBBCCCCDDDDDEEEEE"
        }
    }
}
```

Analyzing metrics with the Performance Insights dashboard

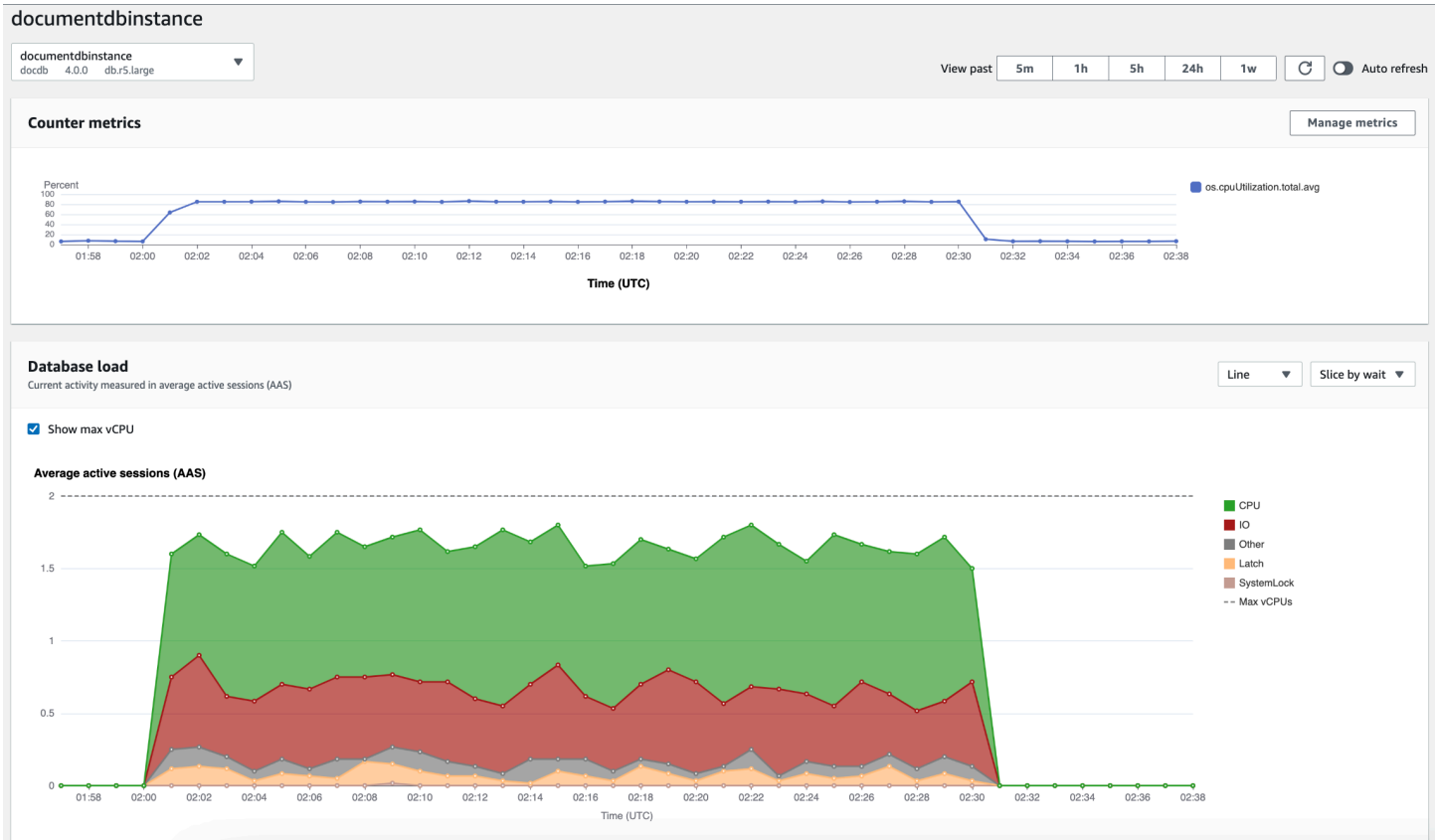
The Performance Insights dashboard contains database performance information to help you analyze and troubleshoot performance issues. On the main dashboard page, you can view information about the database load (DB load). You can "slice" DB load by dimensions such as wait states or query.

Topics

- Overview of the Performance Insights dashboard
- Opening the Performance Insights dashboard
- Analyzing database load by wait states
- Overview of the Top queries tab
- Zooming in on the database load chart

Overview of the Performance Insights dashboard

The dashboard is the easiest way to interact with Performance Insights. The following example shows the dashboard for an Amazon DocumentDB instance. By default, the Performance Insights dashboard shows data for the last hour.



The dashboard is divided into the following parts:

1. **Counter metrics** – Shows data for specific performance counter metrics.
2. **Database load** – Shows how the DB load compares to DB instance capacity as represented by the **Max vCPU** line.
3. **Top dimensions** – Shows the top dimensions contributing to DB load. These dimensions include `waits`, `queries`, `hosts`, `databases`, and `applications`.

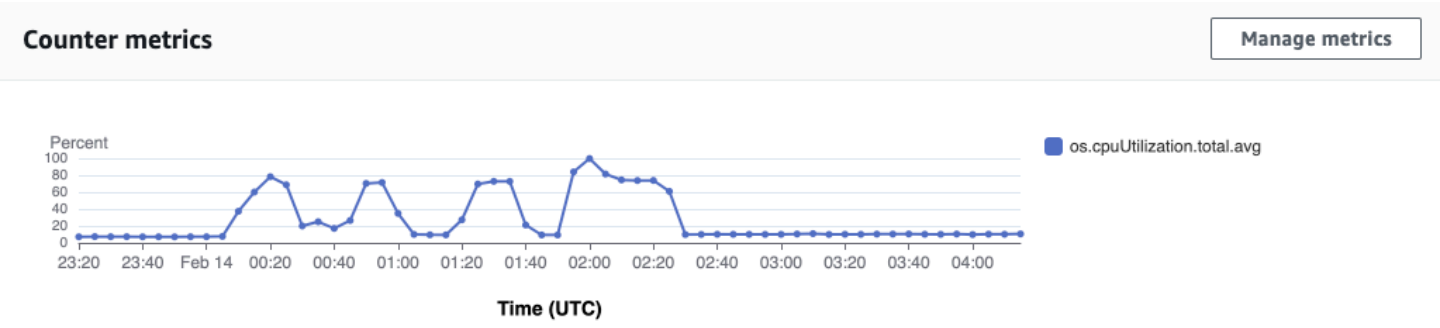
Topics

- Counter metrics chart
- Database load chart
- Top dimensions table

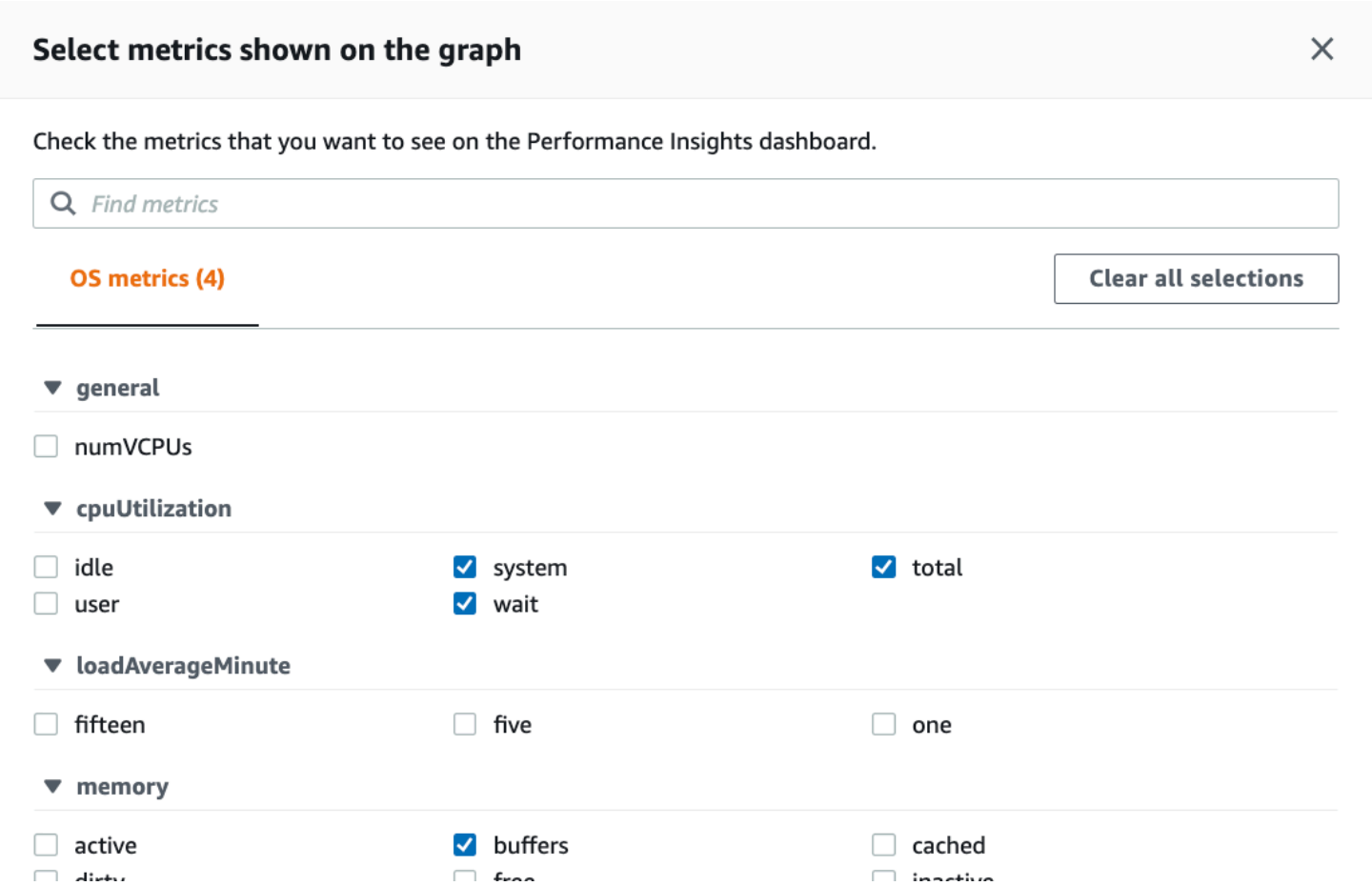
Counter metrics chart

With counter metrics, you can customize the Performance Insights dashboard to include up to 10 additional graphs. These graphs show a selection of dozens of operating system metrics. You can correlate this information with DB load to help identify and analyze performance problems.

The **Counter metrics** chart displays data for performance counters.



To change the performance counters, choose **Manage metrics**. You can select multiple **OS metrics** as shown in the following screenshot. To see details for any metric, hover over the metric name.

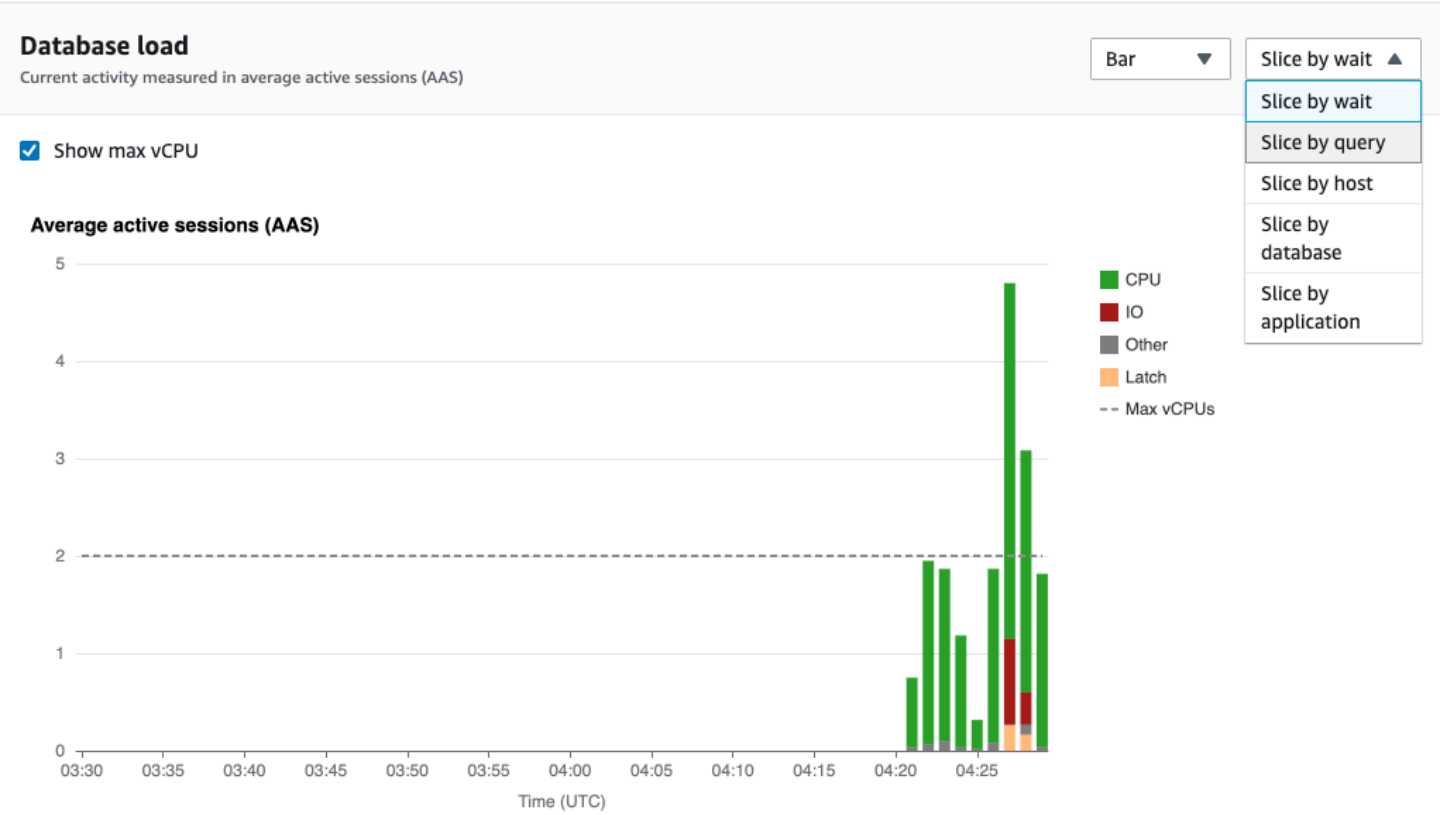


Database load chart

The **Database load** chart shows how the database activity compares to instance capacity as represented by the **Max vCPU** line. By default, the stacked line chart represents DB load as average active sessions per unit of time. The DB load is sliced (grouped) by wait states.

DB load sliced by dimensions

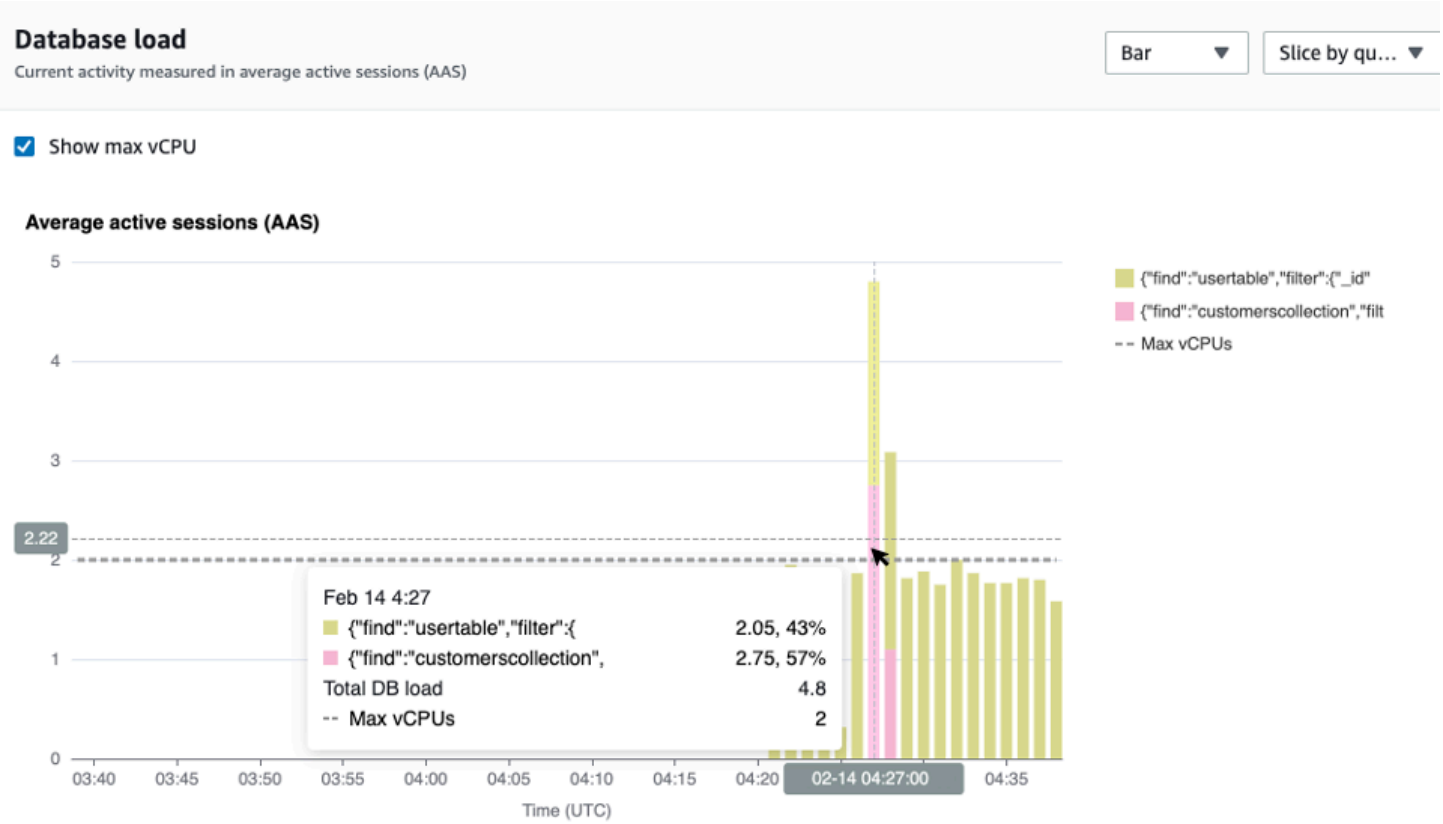
You can choose to display load as active sessions grouped by any supported dimensions. The following image shows the dimensions for the Amazon DocumentDB instance.



DB load details for a dimension item

To see details about a DB load item within a dimension, hover over the item name. The following image shows details for a query statement.

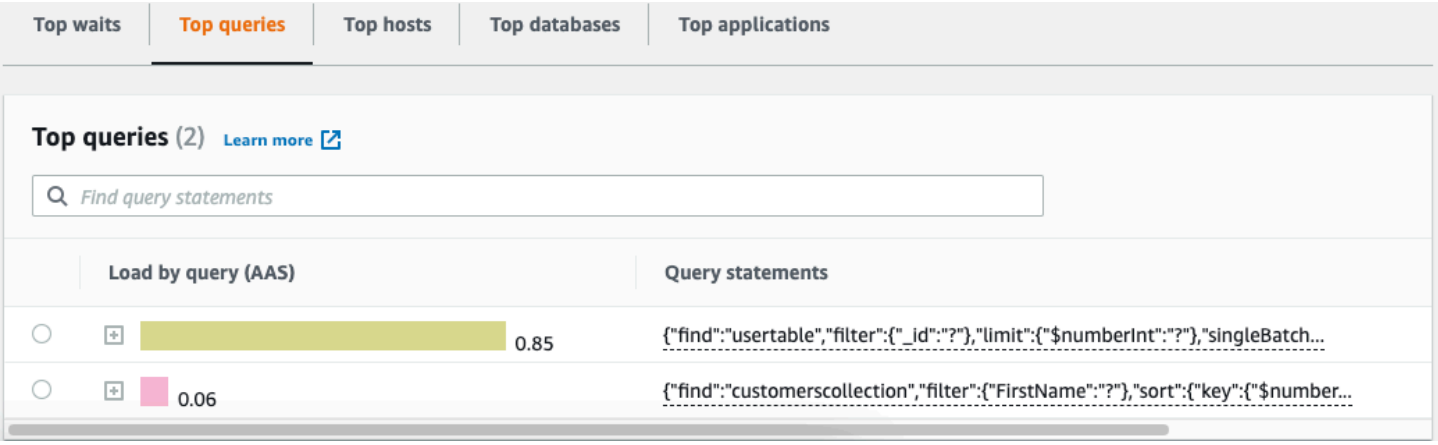
To see details for any item for the selected time period in the legend, hover over that item.



Top dimensions table

The **Top dimensions table** slices DB load by different dimensions. A dimension is a category or "slice by" for different characteristics of DB load. If the dimension is query, **Top queries** shows the query statements that contribute most to the DB load.

Choose any of the following dimension tabs.



The following table provides a brief description of each tab.

Tab	Description
Top waits	The waits for which the database backend is waiting
Top queries	The queries statements that are

Description
currently running
The host IP and port of the connected client
The database of the database to which the client is connected

Tab	Description
Top applications	The application that is connected to the database

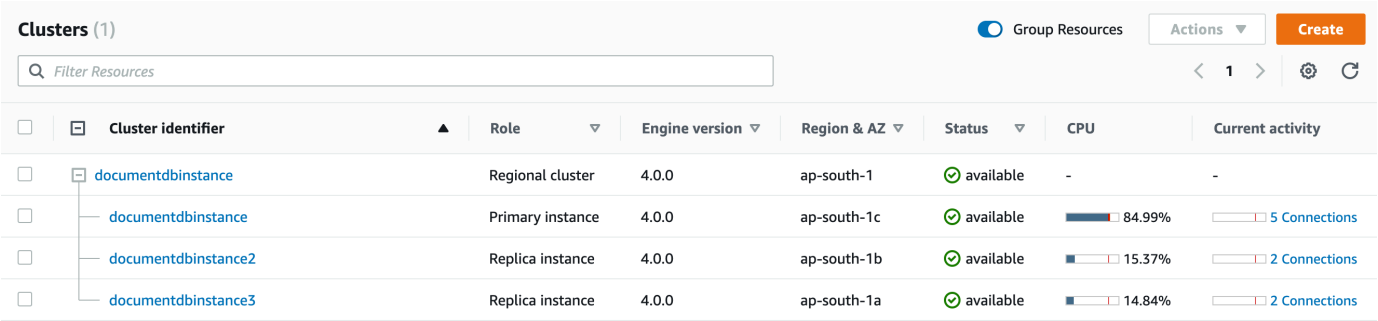
To learn how to analyze queries by using the **Top queries** tab, see Overview of the Top queries tab.

Opening the Performance Insights dashboard

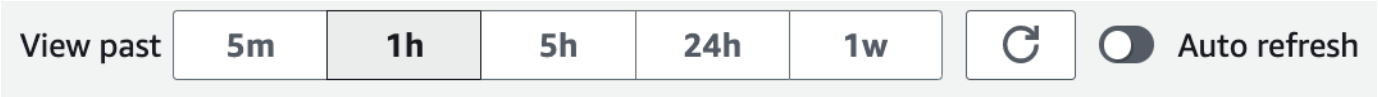
To view the Performance Insights dashboard in the AWS Management Console, use the following steps:

1. Open the Performance Insights console at https://console.aws.amazon.com/docdb/.
2. Choose a DB instance. The Performance Insights dashboard is shown for that Amazon DocumentDB instance.

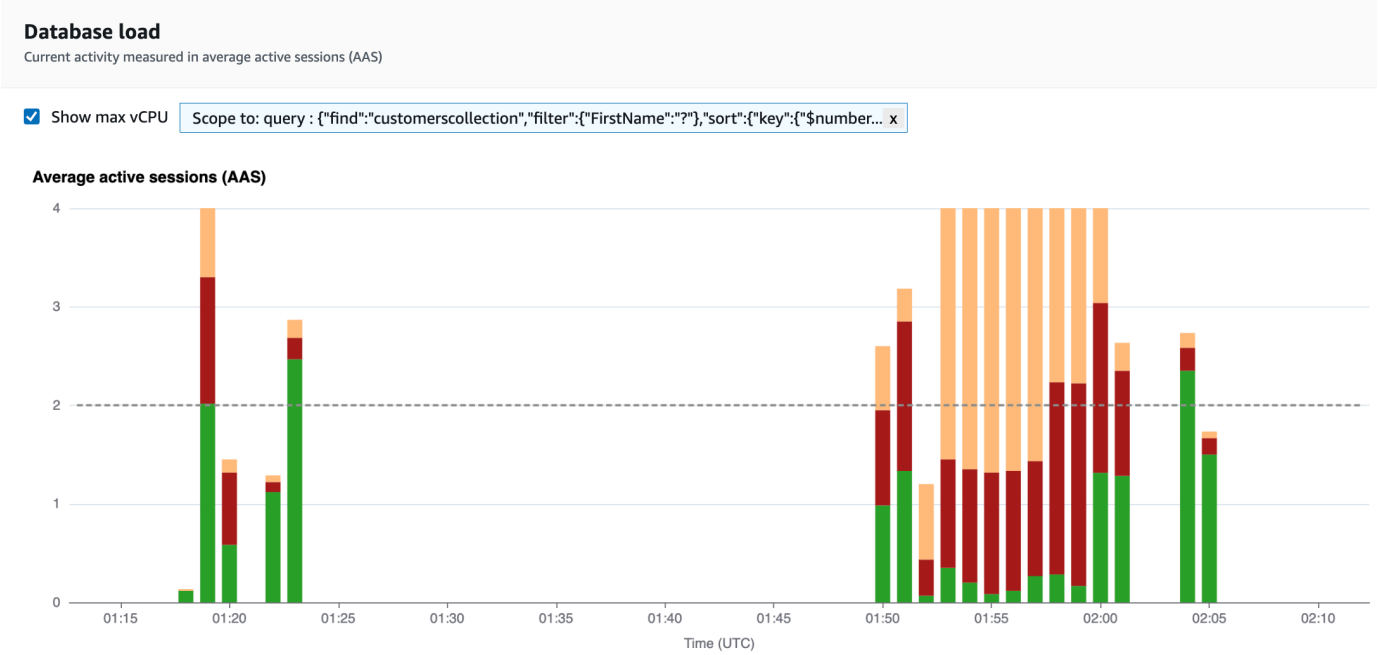
   For Amazon DocumentDB instances with Performance Insights enabled, you can also reach the dashboard by choosing the **Sessions** item in the list of instances. Under **Current activity**, the **Sessions** item shows the database load in average active sessions over the last five minutes. The bar graphically shows the load. When the bar is empty, the instance is idle. As the load increases, the bar fills with blue. When the load passes the number of virtual CPUs (vCPUs) on the instance class, the bar turns red, indicating a potential bottleneck.



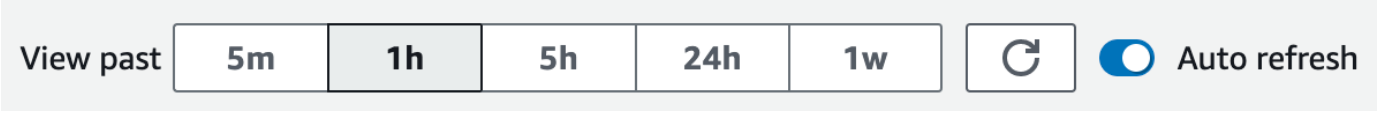
3. (Optional) Choose a different time interval by selecting a button in the upper right. For example, to change the interval to 1 hour, select **1h**.



In the following screenshot, the DB load interval is 1 hour.



4. To refresh your data automatically, enable **Auto refresh**.



The Performance Insight dashboard automatically refreshes with new data. The refresh rate depends on the amount of data displayed:

- 5 minutes refreshes every 5 seconds.

- 1 hour refreshes every minute.
- 5 hours refreshes every minute.
- 24 hours refreshes every 5 minutes.
- 1 week refreshes every hour.

Analyzing database load by wait states

If the **Database load (DB load)** chart shows a bottleneck, you can find out where the load is coming from. To do so, look at the top load items table below the **Database load** chart. Choose a particular item, like a query or an application, to drill down into that item and see details about it.

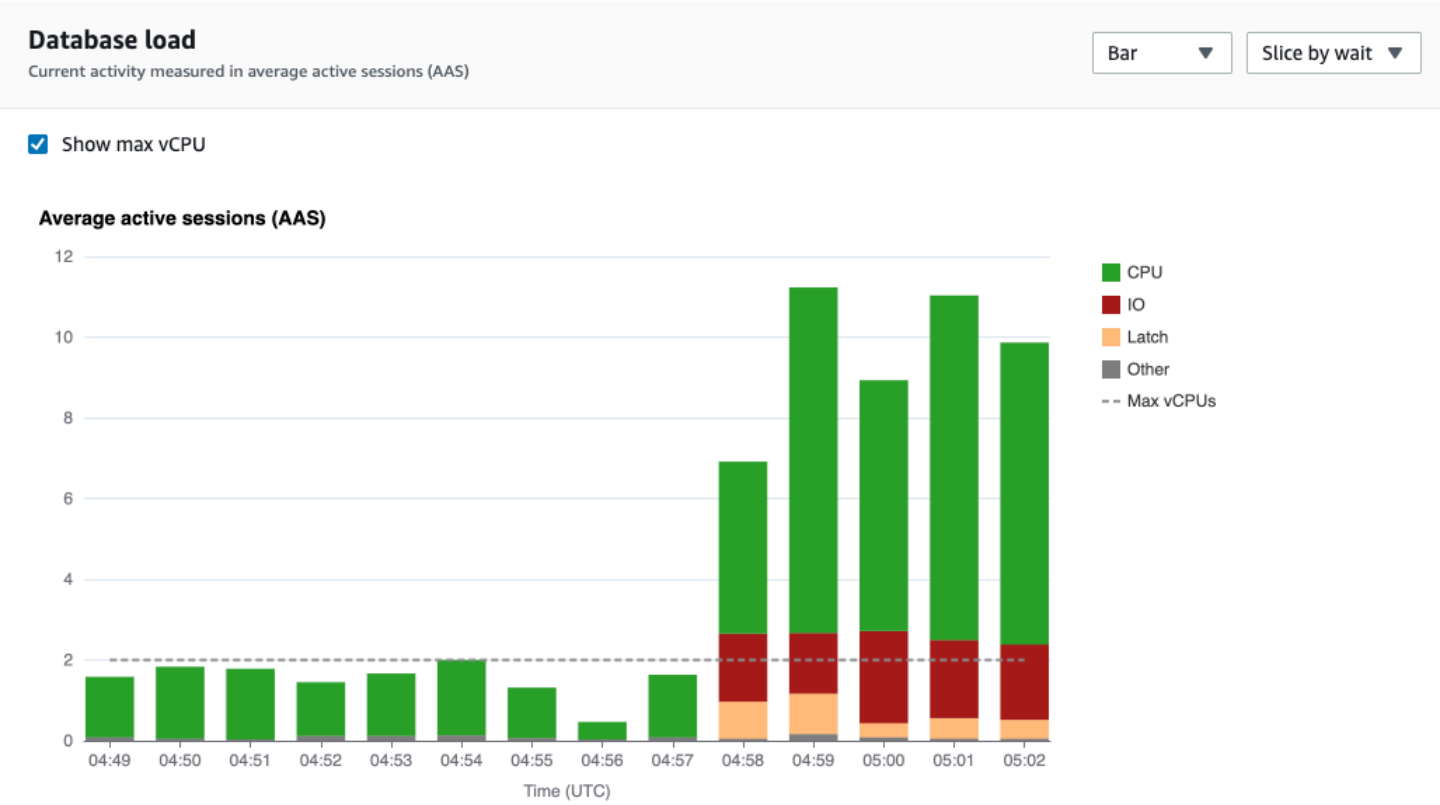
DB load grouped by waits and top queries typically provides the most insight into performance issues. DB load grouped by waits shows if there are any resource or concurrency bottlenecks in the database. In this case, the **Top queries** tab of the top load items table shows which queries are driving that load.

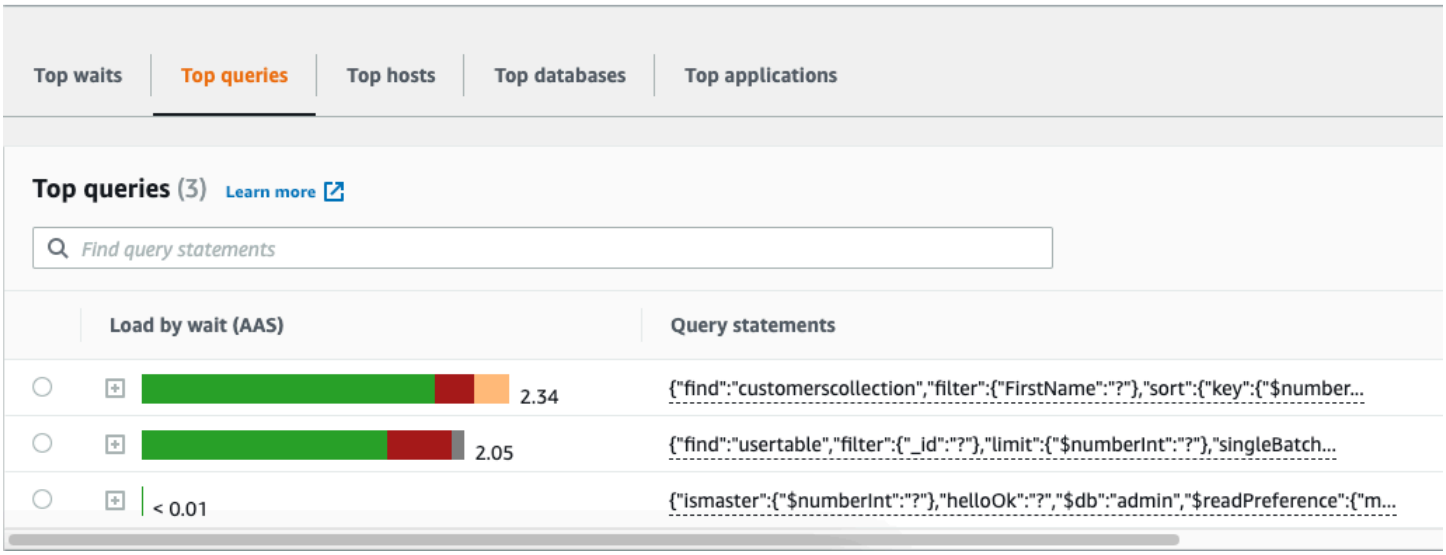
Your typical workflow for diagnosing performance issues is as follows:

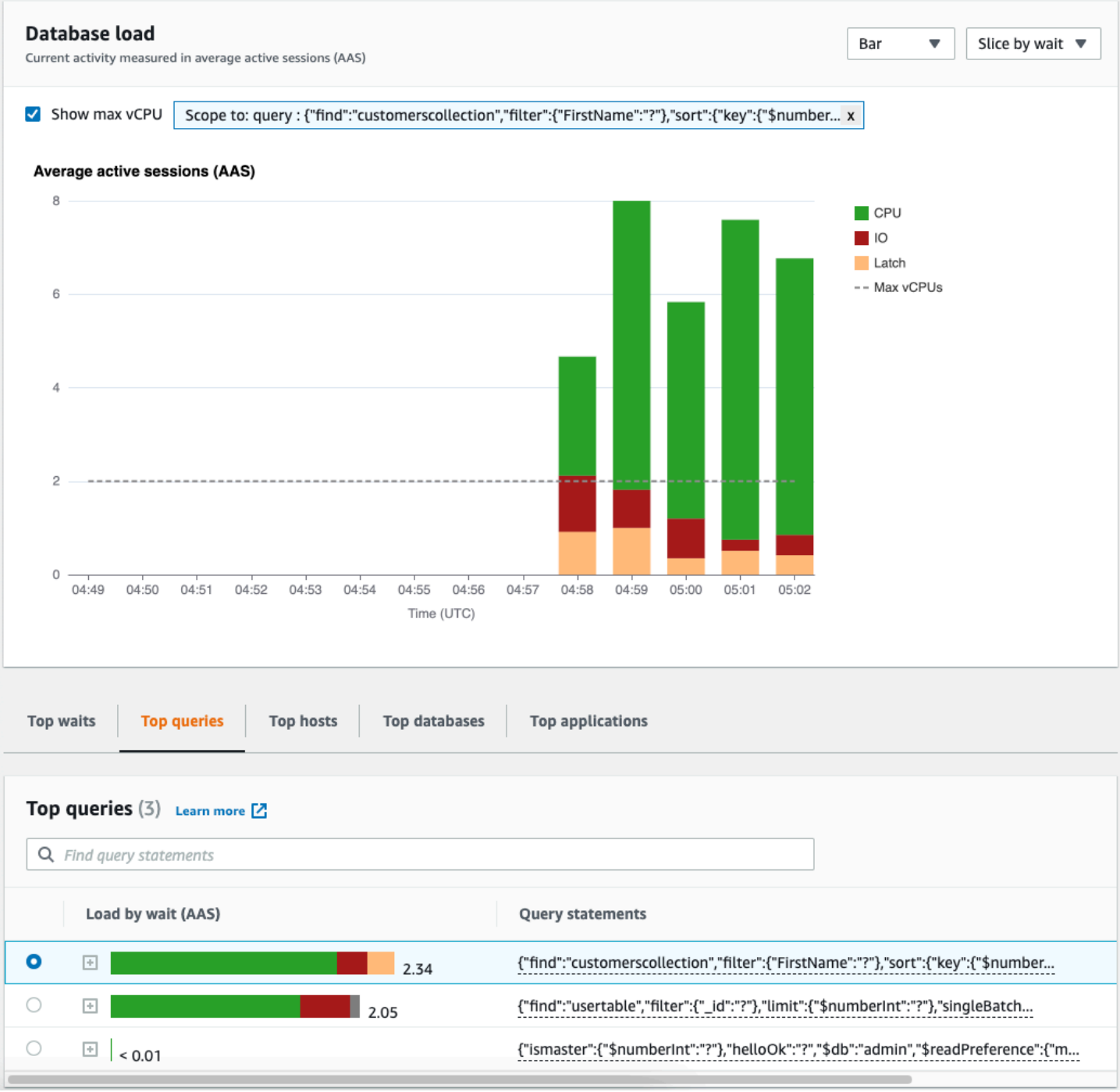
1. Review the **Database load** chart and see if there are any incidents of database load exceeding the **Max CPU** line.
2. If there is, look at the **Database load** chart and identify which wait state or states are primarily responsible.
3. Identify the digest queries causing the load by seeing which of the queries the **Top queries** tab on the top load items table are contributing most to those wait states. You can identify these by the **Load by Wait (AAS)** column.
4. Choose one of these digest queries in the **Top queries** tab to expand it and see the child queries that it is composed of.

You can also see which hosts or applications are contributing the most load by selecting **Top hosts** or **Top applications**, respectively. Application names are specified in the connection string to the Amazon DocumentDB instance. `Unknown` indicates that the application field was not specified.

For example, in the following dashboard, **CPU** waits account for most of the DB load. Selecting the top query under **Top queries** will scope the Database load chart to focus on the most load that is being contributed by the select query.


Database load
Current activity measured in average active sessions (AAS)
Bar
Slice by wait
Show max vCPU
Average active sessions (AAS)
12
10
8
6
4
2
0
04:49
04:50
04:51
04:52
04:53
04:54
04:55
04:56
04:57
04:58
04:59
05:00
05:01
05:02
Time (UTC)
CPU
IO
Latch
Other
Max vCPUs


Top waits
Top queries
Top hosts
Top databases
Top applications
Top queries (3)
Learn more
Find query statements
Load by wait (AAS)
Query statements
2.34
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":{"$number...
2.05
{"find":"usertable","filter":{"_id":"?"},"limit":{"$numberInt":"?"},"singleBatch...
< 0.01
{"ismaster":{"$numberInt":"?"},"helloOk":"?","$db":"admin","$readPreference":{"m...



Overview of the Top queries tab

By default, the **Top query** tab shows the queries that are contributing the most to DB load. You can analyze the query text to help tune your queries.

Topics

- Query digests

- Load by waits (AAS)
- Viewing detailed query information
- Accessing statement query text
- Viewing and downloading statement query text

Query digests

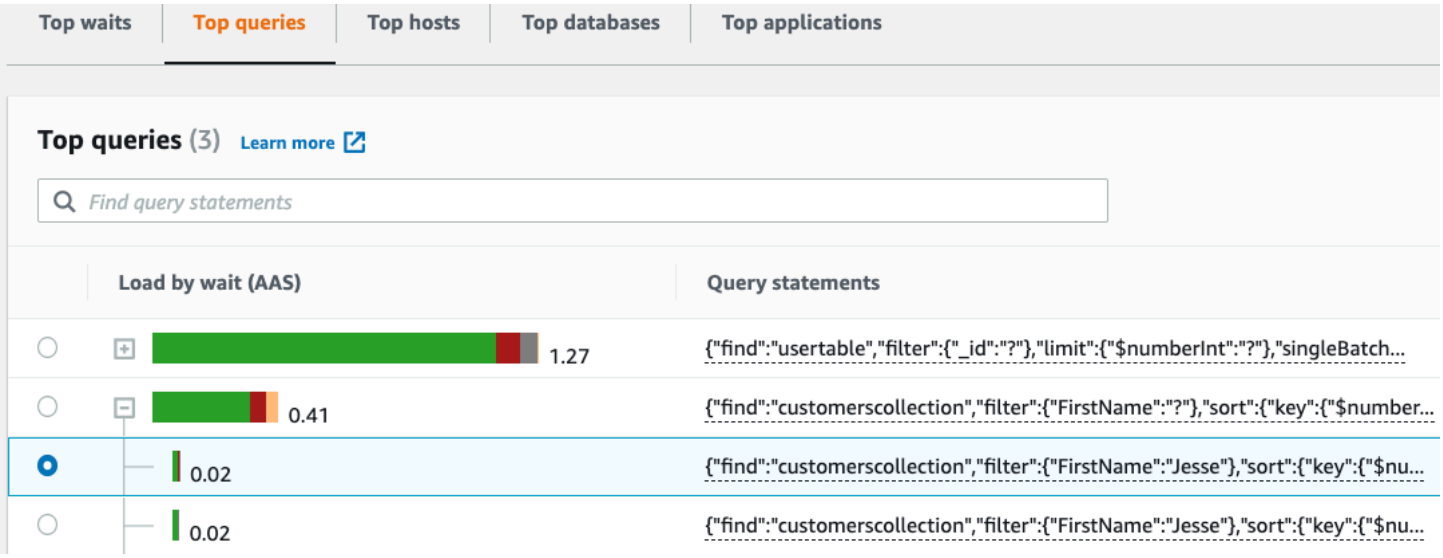
A *query digest* is a composite of multiple actual queries that are structurally similar but might have different literal values. The digest replaces hardcoded values with a question mark. For example, a query digest might look like this:

```
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":
{"$numberInt":"?"}},"limit":{"$numberInt":"?"}}
```

This digest might include the following child queries:

```
{"find":"customerscollection","filter":{"FirstName":"Karrie"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
{"find":"customerscollection","filter":{"FirstName":"Met"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
{"find":"customerscollection","filter":{"FirstName":"Rashin"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
```

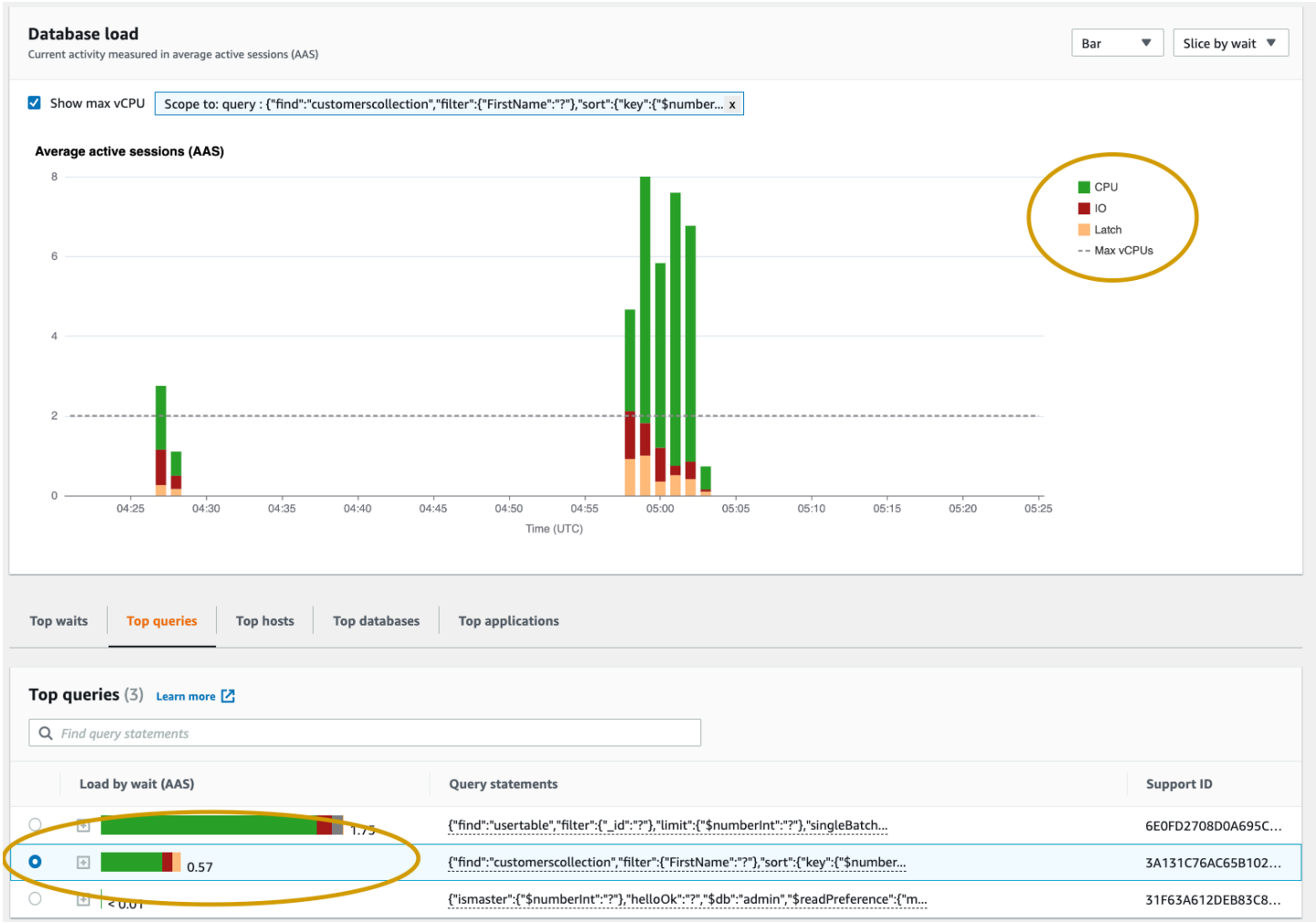
To see the literal query statements in a digest, select the query, and then choose the plus symbol (+). In the following screenshot, the selected query is a digest.



> **Note**
>
> A query digest groups similar query statements, but does not redact sensitive information.

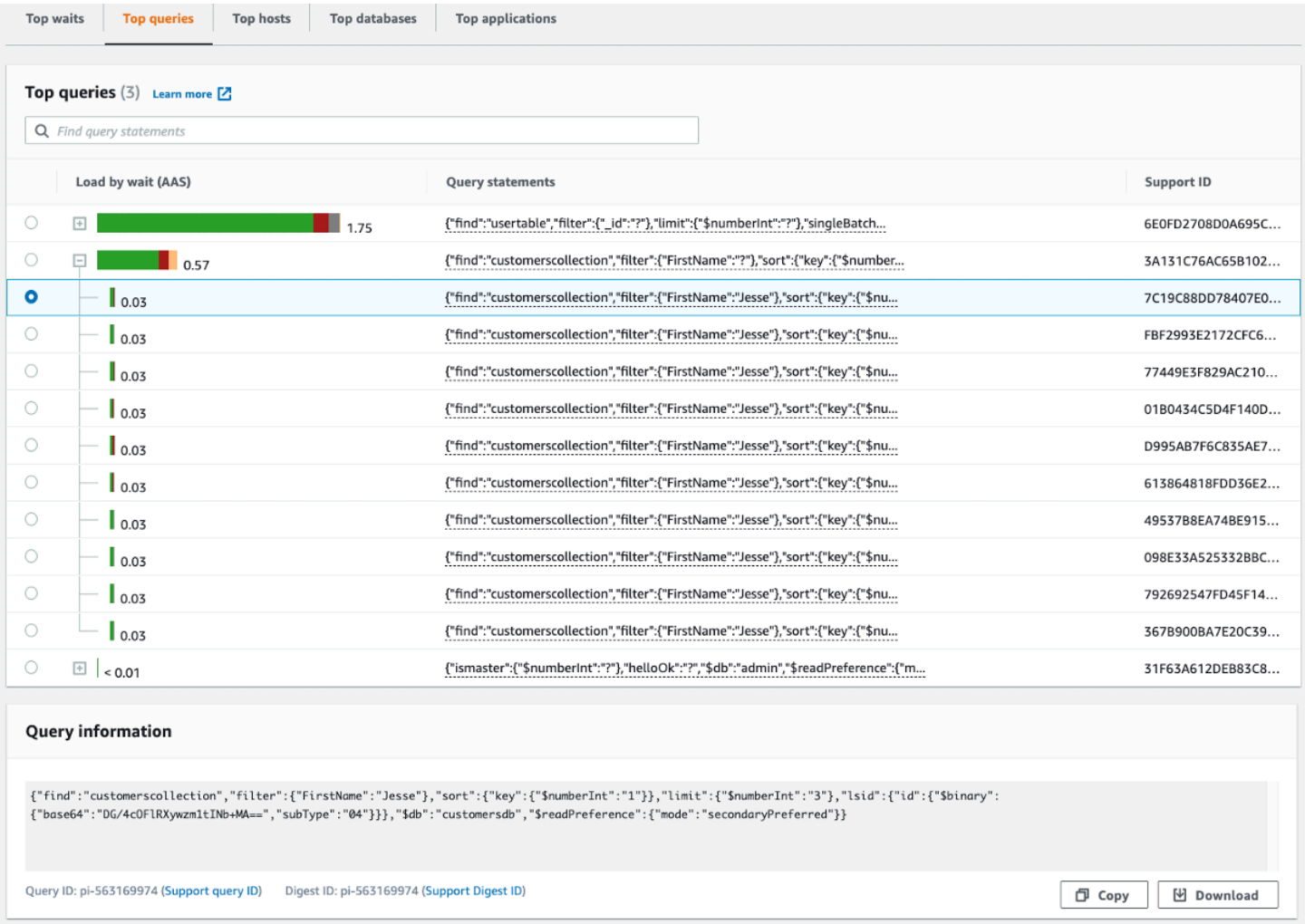
Load by waits (AAS)

In **Top queries**, the **Load by waits (AAS)** column illustrates the percentage of the database load associated with each top load item. This column reflects the load for that item by whatever grouping is currently selected in the **DB load chart**. For example, you might group the **DB load chart** by wait states. In this case, the **DB Load by Waits** bar is sized, segmented, and color-coded to show how much of a given wait state that query is contributing to. It also shows which wait states are affecting the selected query.



Viewing detailed query information

In the **Top query** table, you can open a *digest statement* to view its information. The information appears in the bottom pane.



The following types of identifiers (IDs) are associated with query statements:

1. **Support query ID** – A hash value of the query ID. This value is only for referencing a query ID when you are working with AWS Support. AWS Support doesn't have access to your actual query IDs and query text.
2. **Support digest ID** – A hash value of the digest ID. This value is only for referencing a digest ID when you are working with AWS Support. AWS Support doesn't have access to your actual digest IDs and query text.

Accessing statement query text

By default, each row in the **Top queries** table shows 500 bytes of query text for each query statement. When a digest statement exceeds 500 bytes, you can view more text by opening the statement in the Performance Insights dashboard. In this case, the maximum length for the displayed query is 1 KB. If you view a full query statement, you can also choose **Download**.

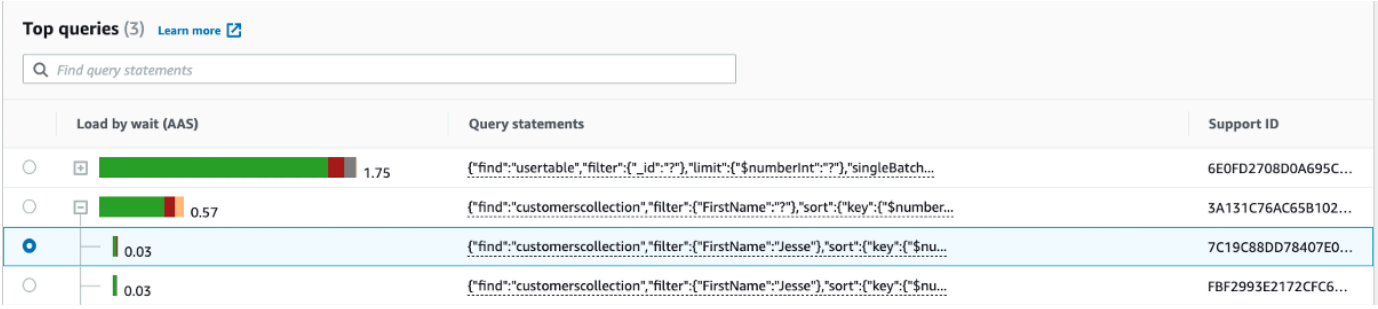
Viewing and downloading statement query text

In the Performance Insights dashboard, you can view or download query text.

To view more query text in the Performance Insights dashboard

1. Open the Amazon DocumentDB console at: https://console.aws.amazon.com/docdb/
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is displayed for that DB instance.

   Query statements with text larger than 500 bytes will look like the following image:



4. Examine the query information section to view more of the query text.

The Performance Insights dashboard can display up to 1 KB for each full query statement.

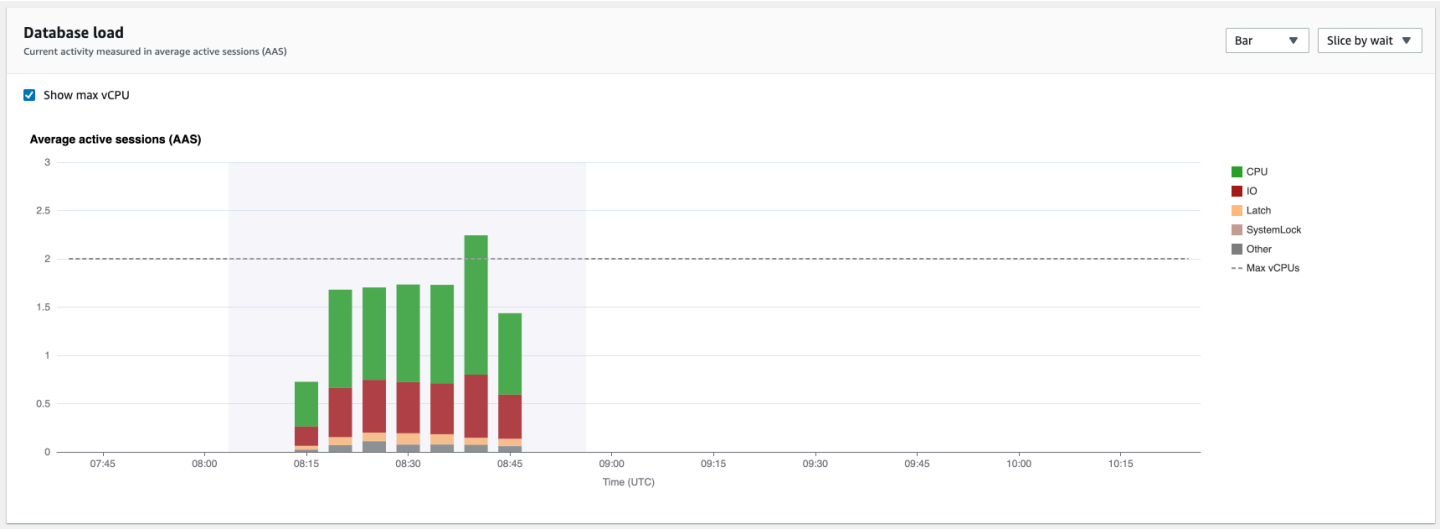
> **Note**
>
> To copy or download the query statement, disable any pop-up blockers.

Zooming in on the database load chart

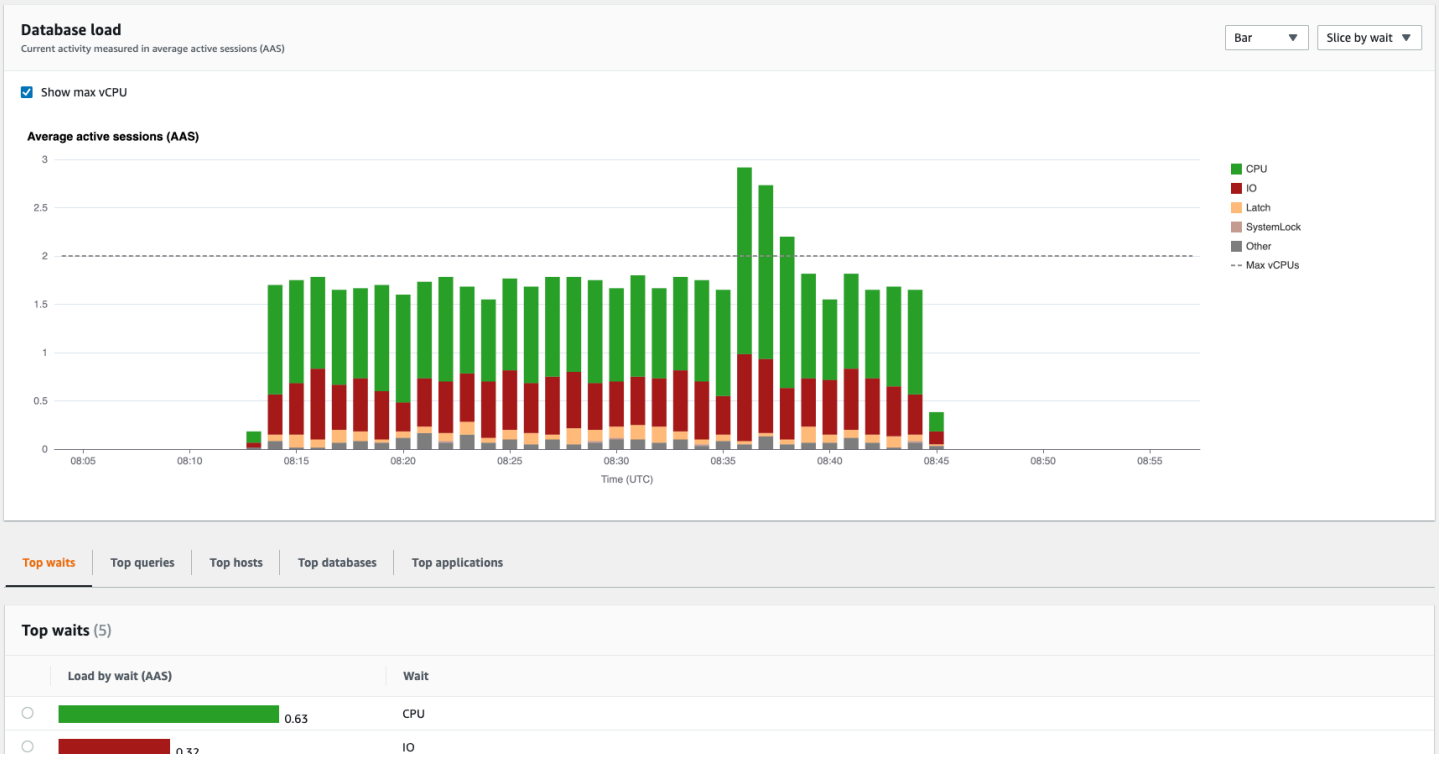
You can use other features of the Performance Insights user interface to help analyze performance data.

Click-and-Drag Zoom In

In the Performance Insights interface, you can choose a small portion of the load chart and zoom in on the detail.



To zoom in on a portion of the load chart, choose the start time and drag to the end of the time period you want. When you do this, the selected area is highlighted. When you release the mouse, the load chart zooms in on the selected area, and the **Top *items*** table is recalculated.



Retrieving metrics with the Performance Insights API

When Performance Insights is enabled, the API provides visibility into instance performance. Amazon CloudWatch Logs provides the authoritative source for vended monitoring metrics for AWS services.

Performance Insights offers a domain-specific view of database load measured as average active sessions (AAS). This metric appears to API consumers as a two-dimensional time-series dataset. The time dimension of the data provides DB load data for each time point in the queried time range. Each time point decomposes overall load in relation to the requested dimensions, such as `Query`, `Wait-state`, `Application`, or `Host`, measured at that time point.

Amazon DocumentDB Performance Insights monitors your Amazon DocumentDB DB instance so that you can analyze and troubleshoot database performance. One way to view Performance Insights data is in the AWS Management Console. Performance Insights also provides a public API so that you can query your own data. You can use the API to do the following:

- Offload data into a database
- Add Performance Insights data to existing monitoring dashboards
- Build monitoring tools

To use the Performance Insights API, enable Performance Insights on one of your Amazon DocumentDB instances. For information about enabling Performance Insights, see Enabling and disabling Performance Insights. For more information about the Performance Insights API, see the Performance Insights API Reference.

The Performance Insights API provides the following operations.

Performance Insights action	AWS CLI command	Description
`DescribeDimensionKeys`	`aws pi describe-dimension-keys`	Retrieves the top N dimension keys for a metric for a specific time period.
`GetDimensionKeyDetails`	`aws pi get-dimension-key-details`	Retrieves the attributes of the specified dimension group for a DB instance or data source. For example, if you specify a query ID, and if the dimension details are available, `GetDimensionKeyDetails` retrieves the full text of the dimension `db.query.statement` associated with this ID. This operation is useful because `GetResourceMetrics` and `DescribeDimensionKeys` don't support retrieval of large query statement text.
`GetResourceMetadata`	`aws pi get-resource-metadata`	Retrieve the metadata for different features. For example, the metadata might indicate that a feature is turned on or off on a specific DB instance.

Performance Insights action	AWS CLI command	Description
GetResourceMetrics	aws pi get-resource-metrics	Retrieves Performance Insights metrics for a set of data sources over a time period. You can provide specific dimension groups and dimensions, and provide aggregation and filtering criteria for each group.
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	Retrieve the dimensions that can be queried for each specified metric type on a specified instance.
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	Retrieve all available metrics of the specified metric types that can be queried for a specified DB instance.

Topics

- AWS CLI for Performance Insights
- Retrieving time-series metrics
- AWS CLI examples for Performance Insights

AWS CLI for Performance Insights

You can view Performance Insights data using the AWS CLI. You can view help for the AWS CLI commands for Performance Insights by entering the following on the command line.

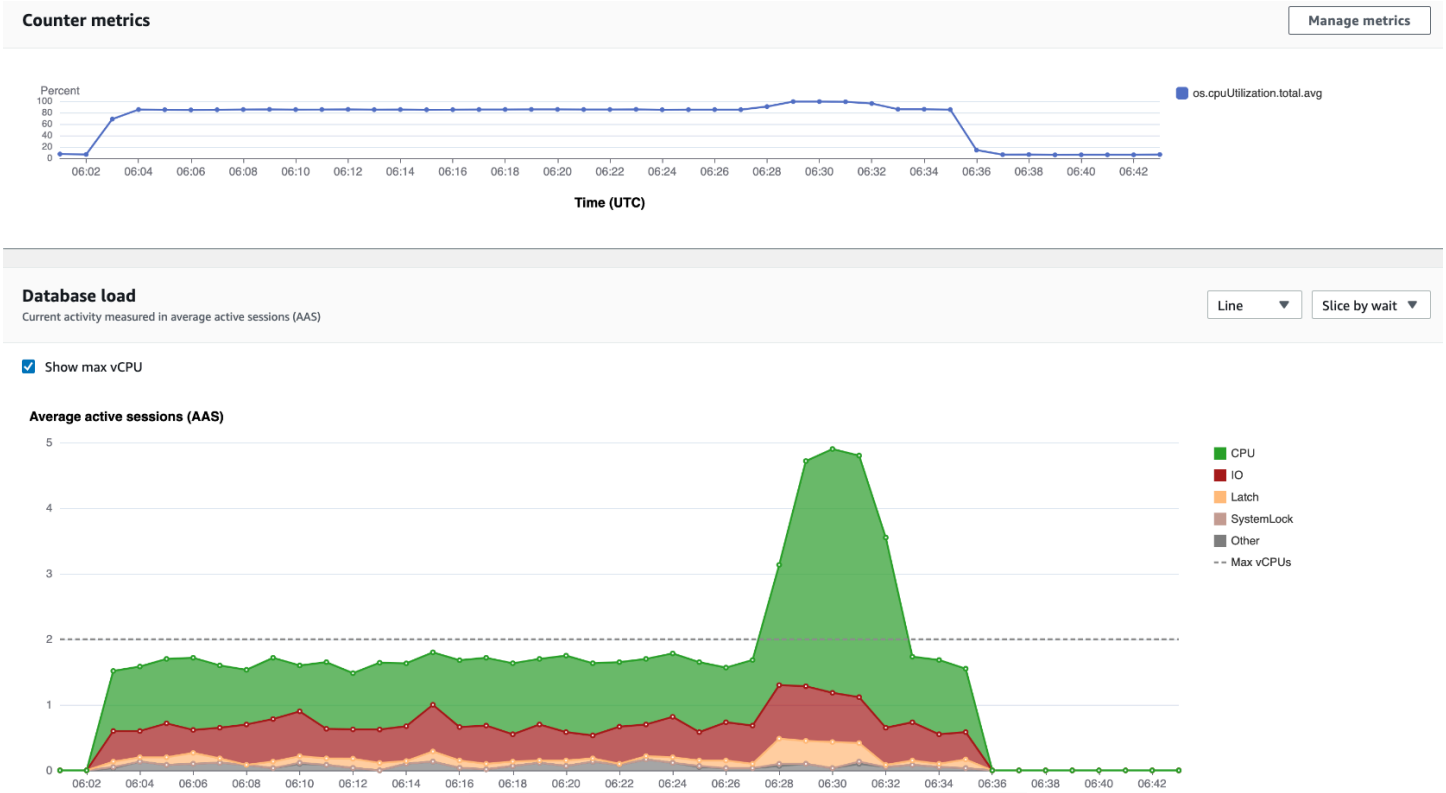
```
aws pi help
```

If you don't have the AWS CLI installed, see Installing the AWS Command Line Interface in the *AWS CLI User Guide* for information about installing it.

Retrieving time-series metrics

The `GetResourceMetrics` operation retrieves one or more time-series metrics from the Performance Insights data. `GetResourceMetrics` requires a metric and time period, and returns a response with a list of data points.

For example, the AWS Management Console uses `GetResourceMetrics` to populate the **Counter Metrics** chart and the **Database Load** chart, as seen in the following image.



All metrics returned by `GetResourceMetrics` are standard time-series metrics, with the exception of `db.load`. This metric is displayed in the **Database Load** chart. The `db.load` metric is different from the other time-series metrics because you can break it into subcomponents called *dimensions*. In the previous image, `db.load` is broken down and grouped by the waits states that make up the `db.load`.

> **Note**
>
> `GetResourceMetrics` can also return the `db.sampleload` metric, but the `db.load` metric is appropriate in most cases.

For information about the counter metrics returned by `GetResourceMetrics`, see Performance Insights for counter metrics.

The following calculations are supported for the metrics:

- Average – The average value for the metric over a period of time. Append `.avg` to the metric name.
- Minimum – The minimum value for the metric over a period of time. Append `.min` to the metric name.
- Maximum – The maximum value for the metric over a period of time. Append `.max` to the metric name.
- Sum – The sum of the metric values over a period of time. Append `.sum` to the metric name.
- Sample count – The number of times the metric was collected over a period of time. Append `.sample_count` to the metric name.

For example, assume that a metric is collected for 300 seconds (5 minutes), and that the metric is collected one time each minute. The values for each minute are 1, 2, 3, 4, and 5. In this case, the following calculations are returned:

- Average – 3
- Minimum – 1
- Maximum – 5
- Sum – 15
- Sample count – 5

For information about using the `get-resource-metrics` AWS CLI command, see `get-resource-metrics`.

For the `--metric-queries` option, specify one or more queries that you want to get results for. Each query consists of a mandatory `Metric` and optional `GroupBy` and `Filter` parameters. The following is an example of a `--metric-queries` option specification.

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
```

```
    "Dimensions": ["string", ...],
    "Limit": integer
  },
  "Filter": {"string": "string"
    ...}
```

AWS CLI examples for Performance Insights

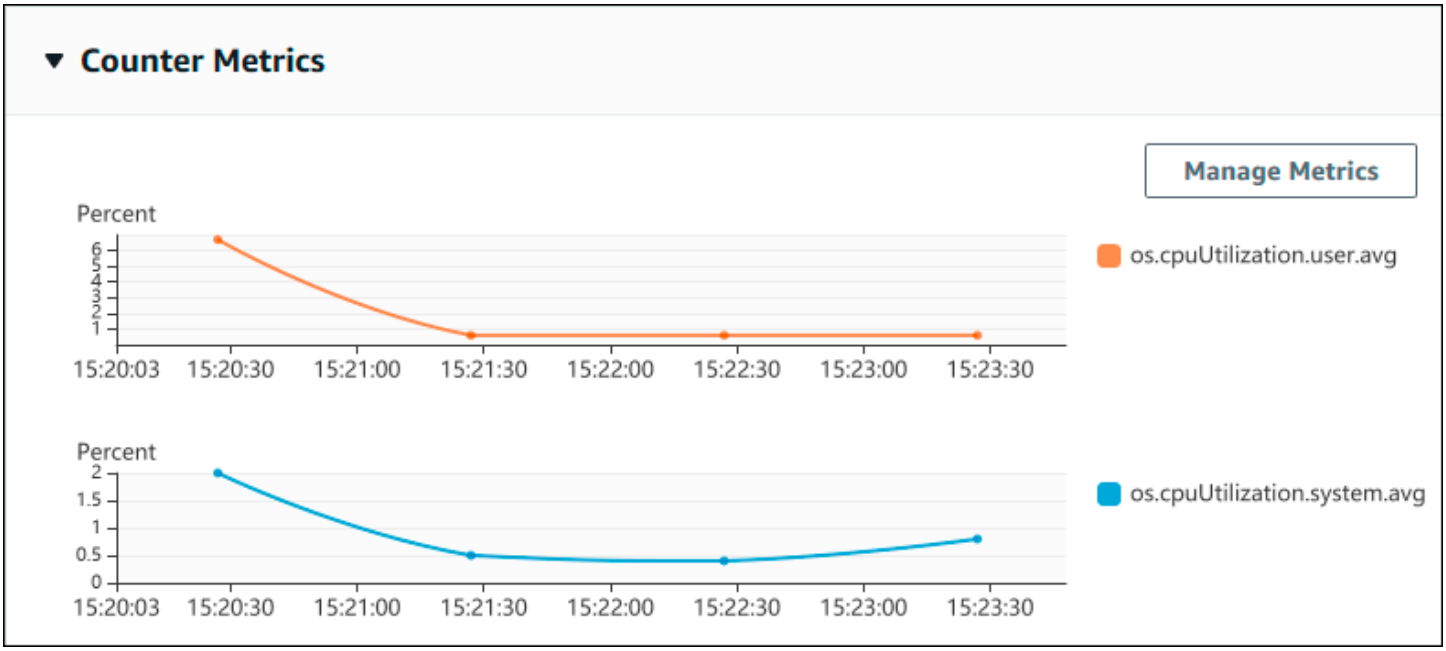
The following examples show how to use the AWS CLI for Performance Insights.

Topics

- Retrieving counter metrics
- Retrieving the DB load average for top wait states
- Retrieving the DB load average for top Query
- Retrieving the DB load average filtered by Query

Retrieving counter metrics

The following screenshot shows two counter metrics charts in the AWS Management Console.



The following example shows how to gather the same data that the AWS Management Console uses to generate the two counter metric charts.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

You can also make a command easier to read by specifying a file for the `--metrics-query` option. The following example uses a file called query.json for the option. The file has the following contents.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

Run the following command to use the file.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
```

```
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

The preceding example specifies the following values for the options:

- `--service-type` – DOCDB for Amazon DocumentDB
- `--identifier` – The resource ID for the DB instance
- `--start-time` and `--end-time` – The ISO 8601 `DateTime` values for the period to query, with multiple supported formats

It queries for a one-hour time range:

- `--period-in-seconds` – 60 for a per-minute query
- `--metric-queries` – An array of two queries, each just for one metric.

  The metric name uses dots to classify the metric in a useful category, with the final element being a function. In the example, the function is `avg` for each query. As with Amazon CloudWatch, the supported functions are `min`, `max`, `total`, and `avg`.

The response looks similar to the following.

```
{
    "AlignedStartTime": "2022-03-13T08:00:00+00:00",
    "AlignedEndTime": "2022-03-13T09:00:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {
            "Key": {
                "Metric": "os.cpuUtilization.user.avg"
```

```
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-03-13T08:01:00+00:00", //Minute1
                    "Value": 3.6
                },
                {
                    "Timestamp": "2022-03-13T08:02:00+00:00", //Minute2
                    "Value": 2.6
                },
                //.... 60 datapoints for the os.cpuUtilization.user.avg metric
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-03-13T08:01:00+00:00",
                    "Value": 92.7
                },
                {
                    "Timestamp": "2022-03-13T08:02:00+00:00",
                    "Value": 93.7
                },
                //.... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

The response has an `Identifier`, `AlignedStartTime`, and `AlignedEndTime`. B the `--period-in-seconds` value was `60`, the start and end times have been aligned to the minute. If the `--period-in-seconds` was `3600`, the start and end times would have been aligned to the hour.

The `MetricList` in the response has a number of entries, each with a `Key` and a `DataPoints` entry. Each `DataPoint` has a `Timestamp` and a `Value`. Each `Datapoints` list has 60 data points because the queries are for per-minute data over an hour, with `Timestamp1/Minute1`, `Timestamp2/Minute2`, and so on, up to `Timestamp60/Minute60`.

Because the query is for two different counter metrics, there are two elements in the response `MetricList`.

Retrieving the DB load average for top wait states

The following example is the same query that the AWS Management Console uses to generate a stacked area line graph. This example retrieves the `db.load.avg` for the last hour with load divided according to the top seven wait states. The command is the same as the command in Retrieving counter metrics. However, the query.json file has the following contents.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 7 }
    }
]
```

Run the following command.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

The example specifies the metric of `db.load.avg` and a `GroupBy` of the top seven wait states. For details about valid values for this example, see DimensionGroup in the *Performance Insights API Reference.*

The response looks similar to the following.

```
{
    "AlignedStartTime": "2022-04-04T06:00:00+00:00",
    "AlignedEndTime": "2022-04-04T06:15:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {//A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": "2022-04-04T06:01:00+00:00",//Minute1
                    "Value": 0.0
                },
                {
                    "Timestamp": "2022-04-04T06:02:00+00:00",//Minute2
                    "Value": 0.0
                },
                //... 60 datapoints for the total db.load.avg key
                ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_state.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-04-04T06:01:00+00:00",//Minute1
                    "Value": 0.0
                },
                {
                    "Timestamp": "2022-04-04T06:02:00+00:00",//Minute2
                    "Value": 0.0
                },
                //... 60 datapoints for the CPU key
            ]
```

```
        },//... In total we have 3 key/datapoints entries, 1) total, 2-3) Top Wait
 States
    ] //end of MetricList
} //end of response
```

In this response, there are three entries in the `MetricList`. There is one entry for the total `db.load.avg`, and three entries each for the `db.load.avg` divided according to one of the top three wait states. Since there was a grouping dimension (unlike the first example), there must be one key for each grouping of the metric. There can't be only one key for each metric, as in the basic counter metric use case.

Retrieving the DB load average for top Query

The following example groups `db.wait_state` by the top 10 query statements. There are two different groups for query statements:

- `db.query` – The full query statement, such as `{"find":"customers","filter":{"FirstName":"Jesse"},"sort":{"key":{"$numberInt":"1"}}}`
- `db.query_tokenized` – The tokenized query statement, such as `{"find":"customers","filter":{"FirstName":"?"},"sort":{"key":{"$numberInt":"?"}},"limit":{"$numberInt":"?"}}`

When analyzing database performance, it can be useful to consider query statements that only differ by their parameters as one logic item. So, you can use `db.query_tokenized` when querying. However, especially when you're interested in `explain()`, sometimes it's more useful to examine full query statements with parameters. There is a parent-child relationship between tokenized and full queries, with multiple full queries (children) grouped under the same tokenized query (parent).

The command in this example is the similar to the command in [Retrieving the DB load average for top wait states](). However, the query.json file has the following contents.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.query_tokenized", "Limit": 10 }
    }
]
```

The following example uses `db.query_tokenized`.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

This example queries over 1 hour, with a one minute period-in-seconds.

The example specifies the metric of `db.load.avg` and a `GroupBy` of the top seven wait states. For details about valid values for this example, see DimensionGroup in the *Performance Insights API Reference.*

The response looks similar to the following.

```
{
    "AlignedStartTime": "2022-04-04T06:00:00+00:00",
    "AlignedEndTime": "2022-04-04T06:15:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {//A list of key/datapoints
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //... 60 datapoints for the total db.load.avg key
                ]
```

```
        },
               {
            "Key": {//Next key are the top tokenized queries
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.query_tokenized.db_id": "pi-1064184600",
                    "db.query_tokenized.id": "77DE8364594EXAMPLE",
                    "db.query_tokenized.statement": "{\"find\":\"customers\",\"filter
\":{\"FirstName\":\"?\"},\"sort\":{\"key\":{\"$numberInt\":\"?\"}},\"limit\"
:{\"$numberInt\":\"?\"},\"$db\":\"myDB\",\"$readPreference\":{\"mode\":\"primary\"}}"
                }
            },
            "DataPoints": [
            //... 60 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized queries, 1 total key
    ] //End of MetricList
} //End of response
```

This response has 11 entries in the `MetricList` (1 total, 10 top tokenized query), with each entry having 24 per-hour `DataPoints`.

For tokenized queries, there are three entries in each dimensions list:

- `db.query_tokenized.statement` – The tokenized query statement.
- `db.query_tokenized.db_id` – The synthetic ID that Performance Insights generates for you. This example returns the `pi-1064184600` synthetic ID.
- `db.query_tokenized.id` – The ID of the query inside Performance Insights.

  In the AWS Management Console, this ID is called the Support ID. It's named this because the ID is data that AWS Support can examine to help you troubleshoot an issue with your database. AWS takes the security and privacy of your data extremely seriously, and almost all data is stored encrypted with your AWS KMS key. Therefore, nobody inside AWS can look at this data. In the example preceding, both the `tokenized.statement` and the `tokenized.db_id` are stored encrypted. If you have an issue with your database, AWS Support can help you by referencing the Support ID.

When querying, it might be convenient to specify a `Group` in `GroupBy`. However, for finer-grained control over the data that's returned, specify the list of dimensions. For example, if all that is

needed is the `db.query_tokenized.statement`, then a `Dimensions` attribute can be added to the query.json file.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.query_tokenized",
            "Dimensions":["db.query_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

Retrieving the DB load average filtered by Query

The corresponding API query in this example is similar to the command in Retrieving the DB load average for top Query. However, the query.json file has the following contents.

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 5  },
        "Filter": { "db.query_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

In this response, all values are filtered according to the contribution of tokenized query AKIAIOSFODNN7EXAMPLE specified in the query.json file. The keys also might follow a different order than a query without a filter, because it's the top five wait states that affected the filtered query.

Amazon CloudWatch metrics for Performance Insights

Performance Insights automatically publishes metrics to Amazon CloudWatch. The same data can be queried from Performance Insights, but having the metrics in CloudWatch makes it easy to add CloudWatch alarms. It also makes it easy to add the metrics to existing CloudWatch Dashboards.

Metric	Description
DBLoad	The number of active sessions for Amazon DocumentDB. Typically, you want the data for the average number of active sessions. In Performance Insights, this data is queried as `db.load.avg` .
DBLoadCPU	The number of active sessions where the wait state type is CPU. In Performance Insights, this data is queried as `db.load.avg` , filtered by the wait state type CPU.
DBLoadNonCPU	The number of active sessions where the wait state type is not CPU.

> **Note**
>
> These metrics are published to CloudWatch only if there is load on the DB instance.

You can examine these metrics using the CloudWatch console, the AWS CLI, or the CloudWatch API.

For example, you can get the statistics for the `DBLoad` metric by running the get-metric-statistics command.

```
aws cloudwatch get-metric-statistics \
    --region ap-south-1 \
    --namespace AWS/DocDB \
    --metric-name DBLoad  \
    --period 360 \
    --statistics Average \
    --start-time 2022-03-14T8:00:00Z \
    --end-time 2022-03-14T9:00:00Z \
    --dimensions Name=DBInstanceIdentifier,Value=documentdbinstance
```

This example generates output similar to the following.

```
{
```

```
    "Datapoints": [
        {
            "Timestamp": "2022-03-14T08:42:00Z",
            "Average": 1.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:24:00Z",
            "Average": 2.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:54:00Z",
            "Average": 6.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:36:00Z",
            "Average": 5.7,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:06:00Z",
            "Average": 4.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:00:00Z",
            "Average": 5.2,
            "Unit": "None"
        }
    ],
    "Label": "DBLoad"
}
```

You can use the `DB_PERF_INSIGHTS` metric math function in the CloudWatch console to query Amazon DocumentDB Performance Insights counter metrics. The `DB_PERF_INSIGHTS` function also includes the `DBLoad` metric at sub-minute intervals. You can set CloudWatch alarms on these metrics. For more details on how to create an alarm, see Create an alarm on Performance Insights counter metrics from an AWS database.

For more information about CloudWatch, see What is Amazon CloudWatch? in the *Amazon CloudWatch User Guide*.

Performance Insights for counter metrics

Counter metrics are operating system metrics in the Performance Insights dashboard. To help identify and analyze performance problems, you can correlate counter metrics with DB load.

Performance Insights operating system counters

The following operating system counters are available with Amazon DocumentDB Performance Insights.

Counter	Type	Metric
active	memory	os.memory.active
buffers	memory	os.memory.buffers
cached	memory	os.memory.cached
dirty	memory	os.memory.dirty
free	memory	os.memory.free
inactive	memory	os.memory.inactive
mapped	memory	os.memory.mapped
pageTables	memory	os.memory.pageTables
slab	memory	os.memory.slab
total	memory	os.memory.total
writeback	memory	os.memory.writeback
idle	cpuUtilization	os.cpuUtilization.idle
system	cpuUtilization	os.cpuUtilization.system
total	cpuUtilization	os.cpuUtilization.total

Counter	Type	Metric
user	cpuUtilization	os.cpuUtilization.user
wait	cpuUtilization	os.cpuUtilization.wait
one	loadAverageMinute	os.loadAverageMinute.one
fifteen	loadAverageMinute	os.loadAverageMinute.fifteen
five	loadAverageMinute	os.loadAverageMinute.five
cached	swap	os.swap.cached
free	swap	os.swap.free
in	swap	os.swap.in
out	swap	os.swap.out
total	swap	os.swap.total
rx	network	os.network.rx
tx	network	os.network.tx
numVCPUs	general	os.general.numVCPUs

Zero-ETL integration with Amazon OpenSearch Service

Topics

- Amazon OpenSearch Service as a destination
- Limitations

Amazon OpenSearch Service as a destination

OpenSearch Service integration with Amazon DocumentDB enables you to stream full load and change data events to OpenSearch domains. The ingestion infrastructure is hosted as OpenSearch ingestion pipelines and provides a high-scale, low latency mechanism to continuously stream data from Amazon DocumentDB collections.

During full load, the zero-ETL integration first extracts historical full load data to OpenSearch using an ingestion pipeline. Once full load data is ingested, the OpenSearch ingestion pipelines will start reading data from Amazon DocumentDB change streams and eventually catch up to maintain near real time data consistency between Amazon DocumentDB and OpenSearch. OpenSearch stores documents in indexes. Incoming data from a Amazon DocumentDB collections can be sent to either one index or can be partitioned into different indices. Ingestion pipelines will sync all create, update and delete events in an Amazon DocumentDB collection as corresponding create, update, and delete of OpenSearch documents to keep both data systems in sync. Ingestion pipelines can be configured to read data from one collection and write to one index or read data from one collection and conditionally route to multiple indexes.

Ingestion pipelines can be configured to stream data from Amazon DocumentDB to Amazon OpenSearch Service using:

- Full load only
- Stream change stream events from Amazon DocumentDB without full load
- Full load followed by change streams from Amazon DocumentDB

To set up your ingestion pipeline, perform the following steps:

Step 1: Create an Amazon OpenSearch Service domain or OpenSearch serverless collection

An Amazon OpenSearch Service collection with appropriate permissions to read data is required. Refer to Getting started with Amazon OpenSearch Service or Getting started with Amazon OpenSearch Serverless in the *Amazon OpenSearch Service* Developer Guide to create a collection. Refer to Amazon OpenSearch Ingestion in the *Amazon OpenSearch Service* Developer Guide to create an AIM role with the correct permissions to access write data to the collection or domain.

Step 2: Enable change streams on the Amazon DocumentDB cluster

Ensure that change streams are enabled on the required collections in the Amazon DocumentDB cluster. Refer to Using change streams with Amazon DocumentDB for more information.

Step 3: Set up the pipeline role with permissions to write to the Amazon S3 bucket and destination domain or collection

After you have your Amazon DocumentDB collection created and change stream enabled, set up the pipeline role that you want to use in your pipeline configuration, and add the following permissions in the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allowReadAndWriteToS3ForExport",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/export/*"
            ]
        }
    ]
}
```

In order for an OpenSearch pipeline to write data to an OpenSearch domain, the domain must have a domain-level access policy that allows the **sts_role_arn** pipeline role to access it. The following sample domain access policy allows the pipeline role named `pipeline-role`, which you created in the previous step, to write data to the domain named `ingestion-domain`:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{your-account-id}:role/{pipeline-role}"
      },
      "Action": ["es:DescribeDomain", "es:ESHttp*"],
      "Resource": "arn:aws:es:{region}:{your-account-id}:domain/{domain-name}/*"
    }
  ]
}
```

Step 4: Add the permissions required on the pipeline role to create X-ENI

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": [
                "arn:aws:ec2:*:420497401461:network-interface/*",
                "arn:aws:ec2:*:420497401461:subnet/*",
                "arn:aws:ec2:*:420497401461:security-group/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [ "ec2:CreateTags" ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
               "StringEquals": { "aws:RequestTag/OSISManaged": "true" }
            }
        }
    ]
}
```

Step 5: Create the pipeline

Configure an OpenSearch ingestion pipeline specifying Amazon DocumentDB as the source. This sample pipeline configuration assumes the use of a change stream fetching mechanism. Refer to Using an OpenSearch Ingestion pipeline with Amazon DocumentDB in the *Amazon OpenSearch Service Developer Guide* for more information.

Limitations

The following limitations apply to the Amazon DocumentDB OpenSearch integration:

- Only one Amazon DocumentDB collection as the source per pipeline is supported.
- Cross-region data ingestion is not supported. Your Amazon DocumentDB cluster and OpenSearch domain must be in the same AWS region.
- Cross-account data ingestion is not supported. Your Amazon DocumentDB cluster and OpenSearch ingestion pipeline must be in the same AWS account.

- Amazon DocumentDB elastic clusters are not supported. Only Amazon DocumentDB instance-based clusters are supported.
- Ensure that the Amazon DocumentDB cluster has authentication enabled using AWS secrets. AWS secrets are the only supported authentication mechanism.
- The existing pipeline configuration can not be updated to ingest data from a different database and/or a different collection. To update the database and/or collection name of a pipeline, you must create a new pipeline.

Developing with Amazon DocumentDB

These sections cover development using Amazon DocumentDB.

Topics

- Connect to Amazon DocumentDB
- Program with Amazon DocumentDB

Connect to Amazon DocumentDB

Amazon DocumentDB provides multiple connection options, including programmatic connections using MongoDB drivers, connecting from outside a VPC through SSH tunneling or AWS VPN, and popular tools like Studio 3T and DataGrip. The service supports replica set connections for high availability. For analytics and reporting, you can connect using JDBC or ODBC drivers. EC2 instances in the same VPC can connect directly to DocumentDB clusters. All connections require TLS and proper AWS credentials/authentication.

Topics

- Connecting programmatically to Amazon DocumentDB
- Connecting to an Amazon DocumentDB cluster from outside an Amazon VPC
- Connecting to Amazon DocumentDB as a replica set
- Connecting to an Amazon DocumentDB cluster from Studio 3T
- Connect to Amazon DocumentDB using DataGrip
- Connect using Amazon EC2
- Connect using the Amazon DocumentDB JDBC driver
- Connect using the Amazon DocumentDB ODBC driver

Connecting programmatically to Amazon DocumentDB

This section contains code examples that demonstrate how to connect to Amazon DocumentDB (with MongoDB compatibility) using several different languages. The examples are separated into two sections based on whether you are connecting to a cluster that has Transport Layer Security (TLS) enabled or disabled. By default, TLS is enabled on Amazon DocumentDB clusters. However, you can turn off TLS if you want. For more information, see Encrypting data in transit.

If you are attempting to connect to your Amazon DocumentDB from outside the VPC in which your cluster resides, please see Connecting to an Amazon DocumentDB cluster from outside an Amazon VPC.

Before you connect to your cluster, you must know whether TLS is enabled on the cluster. The next section shows you how to determine the value of your cluster's `tls` parameter using either the AWS Management Console or the AWS CLI. Following that, you can continue by finding and applying the appropriate code example.

Topics

- Determining the value of your tls parameter
- Connecting with TLS enabled
- Connecting with TLS disabled

Determining the value of your `tls` parameter

Determining whether your cluster has TLS enabled is a two-step process that you can perform using either the AWS Management Console or AWS CLI.

1. **Determine which parameter group is governing your cluster.**

   Using the AWS Management Console

   1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
   2. In the left navigation pane, choose **Clusters**.
   3. In the list of clusters, select the name of your cluster.
   4. The resulting page shows the details of the cluster that you selected. Select the **Configuration** tab. In the **Configurations and status** section, locate the parameter group's name below **Cluster parameter group**.

   Using the AWS CLI

   The following AWS CLI code determines which parameter is governing your cluster. Make sure you replace *`sample-cluster`* with the name of your cluster.

   ```
   aws docdb describe-db-clusters \
   ```

```
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

Output from this operation looks something like the following:

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

2. **Determine the value of the `tls` parameter in your cluster's parameter group.**

Using the AWS Management Console

1. In the navigation pane, choose **Parameter groups**.
2. In the **Cluster parameter groups** window, select your cluster parameter group name from Step 1d.
3. The resulting page shows your cluster parameter group's parameters. You can see the value of the `tls` parameter here. For information on modifying this parameter, see Modifying Amazon DocumentDB cluster parameter groups.

Using the AWS CLI

You can use the `describe-db-cluster-parameters` AWS CLI command to view the details of the parameters in your cluster parameter group.

- **`--describe-db-cluster-parameters`** — To list all the parameters inside a parameter group and their values.
  - **`--db-cluster-parameter-group name`** — Required. The name of your cluster parameter group.

In the following examples, replace each *`user input placeholder`* with your cluster's information.

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

Output from this operation looks something like the following:

```
{
        "Parameters": [
            {
                "ParameterName": "profiler_threshold_ms",
                "ParameterValue": "100",
                "Description": "Operations longer than profiler_threshold_ms
 will be logged",
                "Source": "system",
                "ApplyType": "dynamic",
                "DataType": "integer",
                "AllowedValues": "50-2147483646",
                "IsModifiable": true,
                "ApplyMethod": "pending-reboot"
            },
            {
                "ParameterName": "tls",
                "ParameterValue": "disabled",
                "Description": "Config to enable/disable TLS",
                "Source": "user",
                "ApplyType": "static",
                "DataType": "string",
                "AllowedValues": "disabled,enabled,fips-140-3",
                "IsModifiable": true,
                "ApplyMethod": "pending-reboot"
            }
        ]
}
```

> **Note**
>
> Amazon DocumentDB supports FIPS 140-3 endpoints starting with Amazon DocumentDB 5.0 (engine version 3.0.3727) clusters in these regions: ca-central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1.

After determining the value of your `tls` parameter, continue connecting to your cluster by using one of the code examples in the following sections.

- Connecting with TLS enabled
- Connecting with TLS disabled

Connecting with TLS enabled

To view a code example for programmatically connecting to a TLS-enabled Amazon DocumentDB cluster, choose the appropriate tab for the language that you want to use.

To encrypt data in transit, download the public key for Amazon DocumentDB named `global-bundle.pem` using the following operation.

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

If your application is on Microsoft Windows and requires a PKCS7 file, you can download the PKCS7 certificate bundle. This bundle contains both the intermediate and root certificates at https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b.

Python

The following code demonstrates how to connect to Amazon DocumentDB using Python when TLS is enabled.

In the following example, replace each *`user input placeholder`* with your cluster's information.

```
import pymongo
import sys

##Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set
 and specify the read preference as secondary preferred
client = pymongo.MongoClient('mongodb://sample-user:password@sample-
cluster.node.us-east-1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

##Specify the database to be used
db = client.sample_database

##Specify the collection to be used
col = db.sample_collection

##Insert a single document
```

```
col.insert_one({'hello':'Amazon DocumentDB'})

##Find the document that was previously written
x = col.find_one({'hello':'Amazon DocumentDB'})

##Print the result to the screen
print(x)

##Close the connection
client.close()
```

Node.js

The following code demonstrates how to connect to Amazon DocumentDB using Node.js when TLS is enabled.

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

In the following example, replace each *`user input placeholder`* with your cluster's information.

```
var MongoClient = require('mongodb').MongoClient

//Create a MongoDB client, open a connection to DocDB; as a replica set,
//  and specify the read preference as secondary preferred

var client = MongoClient.connect(
'mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/sample-database?
tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false',
{
  tlsCAFile: `global-bundle.pem` //Specify the DocDB; cert
},
function(err, client) {
    if(err)
        throw err;
```

```
    //Specify the database to be used
    db = client.db('sample-database');

    //Specify the collection to be used
    col = db.collection('sample-collection');

    //Insert a single document
    col.insertOne({'hello':'Amazon DocumentDB'}, function(err, result){
      //Find the document that was previously written
      col.findOne({'hello':'Amazon DocumentDB'}, function(err, result){
        //Print the result to the screen
        console.log(result);

        //Close the connection
        client.close()
      });
   });
});
```

PHP

The following code demonstrates how to connect to Amazon DocumentDB using PHP when TLS is enabled.

In the following example, replace each *`user input placeholder`* with your cluster's information.

```
<?php
//Include Composer's autoloader
require 'vendor/autoload.php';

$TLS_DIR = "/home/ubuntu/global-bundle.pem";

//Create a MongoDB client and open connection to Amazon DocumentDB
$client = new MongoDB\Client("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?retryWrites=false", ["tls" => "true", "tlsCAFile"
 => $TLS_DIR ]);

//Specify the database and collection to be used
$col = $client->sampledatabase->samplecollection;
```

```
//Insert a single document
$result = $col->insertOne( [ 'hello' => 'Amazon DocumentDB'] );

//Find the document that was previously written
$result = $col->findOne(array('hello' => 'Amazon DocumentDB'));

//Print the result to the screen
print_r($result);
?>
```

Go

The following code demonstrates how to connect to Amazon DocumentDB using Go when TLS is enabled.

> **Note**
>
> As of version 1.2.1, the MongoDB Go Driver will only use the first CA server certificate found in `sslcertificateauthorityfile`. The example code below addresses this limitation by manually appending all server certificates found in `sslcertificateauthorityfile` to a custom TLS configuration used during client creation.

In the following examples, replace each *`user input placeholder`* with your cluster's information.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "go.mongodb.org/mongo-driver/bson"
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"

 "io/ioutil"
 "crypto/tls"
 "crypto/x509"
```

```
 "errors"
)

const (
 // Path to the AWS CA file
 caFilePath = "global-bundle.pem"

 // Timeout operations after N seconds
 connectTimeout  = 5
 queryTimeout    = 30
 username        = "sample-user"
 password        = "password"
 clusterEndpoint = "sample-cluster.node.us-east-1.docdb.amazonaws.com:27017"

 // Which instances to read from
 readPreference = "secondaryPreferred"

 connectionStringTemplate = "mongodb://%s:%s@%s/sample-database?
tls=true&replicaSet=rs0&readpreference=%s"
)

func main() {

 connectionURI := fmt.Sprintf(connectionStringTemplate, username, password,
 clusterEndpoint, readPreference)

 tlsConfig, err := getCustomTLSConfig(caFilePath)
 if err != nil {
  log.Fatalf("Failed getting TLS configuration: %v", err)
 }

 client, err :=
 mongo.NewClient(options.Client().ApplyURI(connectionURI).SetTLSConfig(tlsConfig))
 if err != nil {
  log.Fatalf("Failed to create client: %v", err)
 }

 ctx, cancel := context.WithTimeout(context.Background(),
 connectTimeout*time.Second)
 defer cancel()

 err = client.Connect(ctx)
 if err != nil {
  log.Fatalf("Failed to connect to cluster: %v", err)
```

```
}

// Force a connection to verify our connection string
err = client.Ping(ctx, nil)
if err != nil {
 log.Fatalf("Failed to ping cluster: %v", err)
}

fmt.Println("Connected to DocumentDB!")

collection := client.Database("sample-database").Collection("sample-collection")

ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
defer cancel()

res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
if err != nil {
 log.Fatalf("Failed to insert document: %v", err)
}

id := res.InsertedID
log.Printf("Inserted document ID: %s", id)

ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
defer cancel()

cur, err := collection.Find(ctx, bson.D{})

if err != nil {
 log.Fatalf("Failed to run find query: %v", err)
}
defer cur.Close(ctx)

for cur.Next(ctx) {
 var result bson.M
 err := cur.Decode(&result)
 log.Printf("Returned: %v", result)

 if err != nil {
  log.Fatal(err)
 }
}

if err := cur.Err(); err != nil {
```

```
  log.Fatal(err)
 }

}

func getCustomTLSConfig(caFile string) (*tls.Config, error) {
 tlsConfig := new(tls.Config)
 certs, err := ioutil.ReadFile(caFile)

 if err != nil {
  return tlsConfig, err
 }

 tlsConfig.RootCAs = x509.NewCertPool()
 ok := tlsConfig.RootCAs.AppendCertsFromPEM(certs)

 if !ok {
  return tlsConfig, errors.New("Failed parsing pem file")
 }

 return tlsConfig, nil
```

Java

When connecting to a TLS-enabled Amazon DocumentDB cluster from a Java application, your program must use the AWS-provided certificate authority (CA) file to validate the connection. To use the Amazon RDS CA certificate, do the following:

1. Download the Amazon RDS CA file from https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem .
2. Create a trust store with the CA certificate contained in the file by performing the following commands. Be sure to change the *truststore-password* to something else. If you are accessing a trust store that contains both the old CA certificate (`rds-ca-2015-root.pem`) and the new CA certificate (`rds-ca-2019-root.pem`), you can import the certificate bundle into the trust store.

   The following is a sample shell script that imports the certificate bundle into a trust store on a Linux operating system. In the following examples, replace each *user input placeholder* with your own information. Most notably, wherever the example directory "*mydir*" is located in the script, replace it with a directory you created for this task.

```
mydir=/tmp/certs
truststore=${mydir}/rds-truststore.jks
storepassword=truststore-password

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /
Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep
 Alias | cut -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword}
 -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print
 "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

The following is a sample shell script that imports the certificate bundle into a trust store on macOS.

```
mydir=/tmp/certs
truststore=${mydir}/rds-truststore.jks
storepassword=truststore-password

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-
```

```
for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /
Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep
 Alias | cut -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword}
 -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print
 "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

3. Use the `keystore` in your program by setting the following system properties in your application before making a connection to the Amazon DocumentDB cluster.

```
javax.net.ssl.trustStore: truststore
javax.net.ssl.trustStorePassword: truststore-password;
```

4. The following code demonstrates how to connect to Amazon DocumentDB using Java when TLS is enabled.

   In the following example, replace each *`user input placeholder`* with your cluster's information.

```
package com.example.documentdb;

import com.mongodb.client.*;
import org.bson.Document;

public final class Test {
    private Test() {
    }
```

```
    public static void main(String[] args) {

        String template = "mongodb://%s:%s@%s/sample-database?
ssl=true&replicaSet=rs0&readpreference=%s";
        String username = "sample-user";
        String password = "password";
        String clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
        String readPreference = "secondaryPreferred";
        String connectionString = String.format(template, username, password,
 clusterEndpoint, readPreference);

        String truststore = "truststore";
        String truststorePassword = "truststore-password";

        System.setProperty("javax.net.ssl.trustStore", truststore);
        System.setProperty("javax.net.ssl.trustStorePassword",
 truststorePassword);

        MongoClient mongoClient = MongoClients.create(connectionString);

        MongoDatabase testDB = mongoClient.getDatabase("sample-database");
        MongoCollection<Document> numbersCollection =
 testDB.getCollection("sample-collection");

        Document doc = new Document("name", "pi").append("value", 3.14159);
        numbersCollection.insertOne(doc);

        MongoCursor<Document> cursor = numbersCollection.find().iterator();
        try {
            while (cursor.hasNext()) {
                System.out.println(cursor.next().toJson());
            }
        } finally {
            cursor.close();
        }

    }
}
```

C# / .NET

The following code demonstrates how to connect to Amazon DocumentDB using C# / .NET when TLS is enabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
using System;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using MongoDB.Driver;
using MongoDB.Bson;

namespace DocDB
{
    class Program
    {
        static void Main(string[] args)
        {
            string template = "mongodb://{0}:{1}@{2}/sampledatabase?
tls=true&replicaSet=rs0&readpreference={3}";
            string username = "sample-user";
            string password = "password";
            string readPreference = "secondaryPreferred";
            string clusterEndpoint="sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
            string connectionString = String.Format(template, username, password,
 clusterEndpoint, readPreference);

            string pathToCAFile = "<PATH/global-bundle.p7b_file>";

            // ADD CA certificate to local trust store
            // DO this once - Maybe when your service starts
            X509Store localTrustStore = new X509Store(StoreName.Root);
            X509Certificate2Collection certificateCollection = new
 X509Certificate2Collection();
            certificateCollection.Import(pathToCAFile);
            try
```

```
            {
                localTrustStore.Open(OpenFlags.ReadWrite);
                localTrustStore.AddRange(certificateCollection);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Root certificate import failed: " + ex.Message);
                throw;
            }
            finally
            {
                localTrustStore.Close();
            }

            var settings = MongoClientSettings.FromUrl(new
 MongoUrl(connectionString));
            var client = new MongoClient(settings);

            var database = client.GetDatabase("sampledatabase");
            var collection =
 database.GetCollection<BsonDocument>("samplecollection");
            var docToInsert = new BsonDocument { { "pi", 3.14159 } };
            collection.InsertOne(docToInsert);
        }
    }
}
```

MongoDB Shell

The following code demonstrates how to connect to and query Amazon DocumentDB using the newest version, mongosh, or the previous mongo shell version, when TLS is enabled.

Connect to Amazon DocumentDB with mongosh

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

In the following examples, replace each *user input placeholder* with your cluster's information.

```
mongosh --tls --host cluster-end-point:27017 --tlsCAFile  global-bundle.pem --
username sample-user --password password --retryWrites false
```

Connect to Amazon DocumentDB with the previous mongo shell version

If you use IAM, you must use a previous version of mongo shell. Enter one of the following command options:

```
mongo --ssl --host cluster-end-point:27017 --sslCAFile global-bundle.pem --
username sample-user --password password
```

If you are using a version equal to or greater than 4.2, use the following code to connect. Retryable writes are not supported in Amazon DocumentDB. If you are using legacy mongo shell (not mongosh), do not include the `retryWrites=false` command in any code string. By default, retryable writes are disabled. Including `retryWrites=false` might cause a failure in normal read commands.

```
mongo --tls --host cluster-end-point:27017 --tlsCAFile global-bundle.pem --
username sample-user --password password
```

Test the connection

1. Insert a single document.

   ```
   db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
   ```

2. Find the document that was previously inserted.

   ```
   db.myTestCollection.find({'hello':'Amazon DocumentDB'})
   ```

R

The following code demonstrates how to connect to Amazon DocumentDB with R using mongolite (https://jeroen.github.io/mongolite/) when TLS is enabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
#Include the mongolite library.
library(mongolite)

mongourl <- paste("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/test2?ssl=true&",
          "readPreference=secondaryPreferred&replicaSet=rs0", sep="")

#Create a MongoDB client, open a connection to Amazon DocumentDB as a replica
#   set and specify the read preference as secondary preferred
client <-  mongo(url = mongourl, options = ssl_options(weak_cert_validation = F, ca
 ="<PATH/global-bundle.pem>"))

#Insert a single document
str <- c('{"hello" : "Amazon DocumentDB"}')
client$insert(str)

#Find the document that was previously written
client$find()
```

Ruby

The following code demonstrates how to connect to Amazon DocumentDB with Ruby when TLS is enabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
require 'mongo'
require 'neatjson'
require 'json'
client_host = 'mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017'
client_options = {
   database: 'test',
   replica_set: 'rs0',
   read: {:secondary_preferred => 1},
   user: 'sample-user',
   password: 'password',
   ssl: true,
   ssl_verify: true,
   ssl_ca_cert: 'PATH/global-bundle.pem',
```

```
   retry_writes: false
}


begin
   ##Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
   client = Mongo::Client.new(client_host, client_options)

   ##Insert a single document
   x = client[:test].insert_one({"hello":"Amazon DocumentDB"})

   ##Find the document that was previously written
   result = client[:test].find()

   #Print the document
   result.each do |document|
      puts JSON.neat_generate(document)
   end
end

#Close the connection
client.close
```

Connecting with TLS disabled

To view a code example for programmatically connecting to a TLS-disabled Amazon DocumentDB cluster, choose the tab for language that you want to use.

Python

The following code demonstrates how to connect to Amazon DocumentDB using Python when TLS is disabled.

In the following example, replace each *`user input placeholder`* with your cluster's information.

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set
 and specify the read preference as secondary preferred

import pymongo
import sys
```

```
client = pymongo.MongoClient('mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

##Specify the database to be used
db = client.sample_database

##Specify the collection to be used
col = db.sample_collection

##Insert a single document
col.insert_one({'hello':'Amazon DocumentDB'})

##Find the document that was previously written
x = col.find_one({'hello':'Amazon DocumentDB'})

##Print the result to the screen
print(x)

##Close the connection
client.close()
```

Node.js

The following code demonstrates how to connect to Amazon DocumentDB using Node.js when TLS is disabled.

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

In the following example, replace each *`user input placeholder`* with your cluster's information.

```
var MongoClient = require('mongodb').MongoClient;

//Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set,
```

```
//  and specify the read preference as secondary preferred
var client = MongoClient.connect(
'mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/sample-database?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false',
{
  useNewUrlParser: true
},

function(err, client) {
    if(err)
        throw err;
    //Specify the database to be used
    db = client.db('sample-database');

    //Specify the collection to be used
    col = db.collection('sample-collection');

    //Insert a single document
    col.insertOne({'hello':'Amazon DocumentDB'}, function(err, result){
      //Find the document that was previously written
      col.findOne({'hello':'Amazon DocumentDB'}, function(err, result){
        //Print the result to the screen
        console.log(result);

        //Close the connection
        client.close()
      });
   });
});
```

PHP

The following code demonstrates how to connect to Amazon DocumentDB using PHP when TLS is disabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
<?php
//Include Composer's autoloader
require 'vendor/autoload.php';
```

```
//Create a MongoDB client and open connection to Amazon DocumentDB
$client = new MongoDB\Client("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?retryWrites=false");

//Specify the database and collection to be used
$col = $client->sampledatabase->samplecollection;

//Insert a single document
$result = $col->insertOne( [ 'hello' => 'Amazon DocumentDB'] );

//Find the document that was previously written
$result = $col->findOne(array('hello' => 'Amazon DocumentDB'));

//Print the result to the screen
print_r($result);
?>
```

Go

The following code demonstrates how to connect to Amazon DocumentDB using Go when TLS is disabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "go.mongodb.org/mongo-driver/bson"
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"
)

const (
 // Timeout operations after N seconds
 connectTimeout  = 5
 queryTimeout    = 30
 username        = "sample-user"
```

```
 password        = "password"
 clusterEndpoint = "sample-cluster.node.us-east-1.docdb.amazonaws.com:27017"

 // Which instances to read from
 readPreference           = "secondaryPreferred"
 connectionStringTemplate = "mongodb://%s:%s@%s/sample-database?
replicaSet=rs0&readpreference=%s"
)

func main() {

 connectionURI := fmt.Sprintf(connectionStringTemplate, username, password,
 clusterEndpoint, readPreference)

 client, err := mongo.NewClient(options.Client().ApplyURI(connectionURI))
 if err != nil {
  log.Fatalf("Failed to create client: %v", err)
 }

 ctx, cancel := context.WithTimeout(context.Background(),
 connectTimeout*time.Second)
 defer cancel()

 err = client.Connect(ctx)
 if err != nil {
  log.Fatalf("Failed to connect to cluster: %v", err)
 }

 // Force a connection to verify our connection string
 err = client.Ping(ctx, nil)
 if err != nil {
  log.Fatalf("Failed to ping cluster: %v", err)
 }

 fmt.Println("Connected to DocumentDB!")

 collection := client.Database("sample-database").Collection("sample-collection")

 ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
 defer cancel()

 res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
 if err != nil {
  log.Fatalf("Failed to insert document: %v", err)
```

```
 }

 id := res.InsertedID
 log.Printf("Inserted document ID: %s", id)

 ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
 defer cancel()

 cur, err := collection.Find(ctx, bson.D{})

 if err != nil {
  log.Fatalf("Failed to run find query: %v", err)
 }
 defer cur.Close(ctx)

 for cur.Next(ctx) {
  var result bson.M
  err := cur.Decode(&result)
  log.Printf("Returned: %v", result)

  if err != nil {
   log.Fatal(err)
  }
 }

 if err := cur.Err(); err != nil {
  log.Fatal(err)
 }

}
```

Java

The following code demonstrates how to connect to Amazon DocumentDB using Java when TLS is disabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
package com.example.documentdb;

import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;
```

```
import com.mongodb.ServerAddress;
import com.mongodb.MongoException;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import org.bson.Document;


public final class Main {
    private Main() {
    }
    public static void main(String[] args) {

        String template = "mongodb://%s:%s@%s/sample-database?
replicaSet=rs0&readpreference=%s";
        String username = "sample-user";
        String password = "password";
        String clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
        String readPreference = "secondaryPreferred";
        String connectionString = String.format(template, username, password,
 clusterEndpoint, readPreference);

        MongoClientURI clientURI = new MongoClientURI(connectionString);
        MongoClient mongoClient = new MongoClient(clientURI);

        MongoDatabase testDB = mongoClient.getDatabase("sample-database");
        MongoCollection<Document> numbersCollection = testDB.getCollection("sample-
collection");

        Document doc = new Document("name", "pi").append("value", 3.14159);
        numbersCollection.insertOne(doc);

        MongoCursor<Document> cursor = numbersCollection.find().iterator();
        try {
            while (cursor.hasNext()) {
                System.out.println(cursor.next().toJson());
            }
        } finally {
            cursor.close();
        }

    }
```

```
}
```

C# / .NET

The following code demonstrates how to connect to Amazon DocumentDB using C# / .NET when TLS is disabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
using System;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using MongoDB.Driver;
using MongoDB.Bson;

namespace CSharpSample
{
   class Program
    {
       static void Main(string[] args)
        {
           string template = "mongodb://{0}:{1}@{2}/sampledatabase?
replicaSet=rs0&readpreference={3}";
           string username = "sample-user";
           string password = "password";
           string clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
           string readPreference = "secondaryPreferred";
           string connectionString = String.Format(template, username, password,
 clusterEndpoint, readPreference);

           var settings = MongoClientSettings.FromUrl(new
 MongoUrl(connectionString));
           var client = new MongoClient(settings);

           var database = client.GetDatabase("sampledatabase");
           var collection =
 database.GetCollection<BsonDocument>("samplecollection");
```

```
            var docToInsert = new BsonDocument { { "pi", 3.14159 } };
             collection.InsertOne(docToInsert);
        }
    }
}
```

MongoDB Shell

The following code demonstrates how to connect to and query Amazon DocumentDB using the newest version, mongosh, or the previous mongo shell version, when TLS is disabled.

Connect to Amazon DocumentDB with mongosh

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

In the following examples, replace each *`user input placeholder`* with your cluster's information.

```
mongosh --host cluster-end-point:27017 --username sample-user --password password --
retryWrites false
```

Connect to Amazon DocumentDB with the previous mongo shell version

If you use IAM, you must use a previous version of mongo shell. Enter one of the following command options:

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

If you are using a version equal to or greater than 4.2, use the following code to connect. Retryable writes are not supported in Amazon DocumentDB. If you are using legacy mongo shell (not mongosh), do not include the `retryWrites=false` command in any code string. By default, retryable writes are disabled. Including `retryWrites=false` might cause a failure in normal read commands.

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

Test the connection

1. Insert a single document.

```
db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
```

2. Find the document that was previously inserted.

```
db.myTestCollection.find({'hello':'Amazon DocumentDB'})
```

R

The following code demonstrates how to connect to Amazon DocumentDB with R using mongolite (https://jeroen.github.io/mongolite/) when TLS is disabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
#Include the mongolite library.
library(mongolite)

#Create a MongoDB client, open a connection to Amazon DocumentDB as a replica
#   set and specify the read preference as secondary preferred
client <- mongo(url = "mongodb://sample-user:password@sample-
cluster.node.us-east-1.docdb.amazonaws.com:27017/sample-database?
readPreference=secondaryPreferred&replicaSet=rs0")

##Insert a single document
str <- c('{"hello" : "Amazon DocumentDB"}')
client$insert(str)

##Find the document that was previously written
client$find()
```

Ruby

The following code demonstrates how to connect to Amazon DocumentDB with Ruby when TLS is disabled.

In the following example, replace each *user input placeholder* with your cluster's information.

```
require 'mongo'
require 'neatjson'
require 'json'
client_host = 'mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017'
client_options = {
   database: 'test',
   replica_set: 'rs0',
   read: {:secondary_preferred => 1},
   user: 'sample-user',
   password: 'password',
   retry_writes: false
}

begin
   ##Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
   client = Mongo::Client.new(client_host, client_options)

   ##Insert a single document
   x = client[:test].insert_one({"hello":"Amazon DocumentDB"})

   ##Find the document that was previously written
   result = client[:test].find()

   #Print the document
   result.each do |document|
      puts JSON.neat_generate(document)
   end
end

#Close the connection
client.close
```

Connecting to an Amazon DocumentDB cluster from outside an Amazon VPC

Amazon DocumentDB (with MongoDB compatibility) clusters are deployed within an Amazon Virtual Private Cloud (Amazon VPC). They can be accessed directly by Amazon EC2 instances or

other AWS services that are deployed in the same Amazon VPC. Additionally, Amazon DocumentDB can be accessed by EC2 instances or other AWS services in different VPCs in the same AWS Region or other Regions via VPC peering.

However, suppose that your use case requires that you (or your application) access your Amazon DocumentDB resources from outside the cluster's VPC. In that case, you can use SSH tunneling (also known as *port forwarding*) to access your Amazon DocumentDB resources.

It is beyond the scope of this topic to discuss SSH tunneling in depth. For more information about SSH tunneling, see the following:

- SSH Tunnel
- SSH Port Forwarding Example, specifically the Local Forwarding section

To create an SSH tunnel, you need an Amazon EC2 instance running in the same Amazon VPC as your Amazon DocumentDB cluster. You can either use an existing EC2 instance in the same VPC as your cluster or create one. For more information, see the topic that is appropriate for your operating system:

- Getting Started with Amazon EC2 Linux Instances
- Getting Started with Amazon EC2 Windows Instances

You might typically connect to an EC2 instance using the following command.

```
ssh -i "ec2Access.pem" ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com
```

If so, you can set up an SSH tunnel to the Amazon DocumentDB cluster `sample-cluster.node.us-east-1.docdb.amazonaws.com` by running the following command on your local computer. The `-L` flag is used for forwarding a local port. When using an SSH tunnel, we recommend that you connect to your cluster using the cluster endpoint and do not attempt to connect in replica set mode (i.e., specifying `replicaSet=rs0` in your connection string) as it will result in an error.

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-east-1.docdb.amazonaws.com:27017
 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -N
```

After the SSH tunnel is created, any commands that you issue to `localhost:27017` are forwarded to the Amazon DocumentDB cluster `sample-cluster` running in the

Amazon VPC. If Transport Layer Security (TLS) is enabled on your Amazon DocumentDB cluster, you need to download the public key for Amazon DocumentDB from https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem . The following operation downloads this file:

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

> **Note**
>
> TLS is enabled by default for new Amazon DocumentDB clusters. However, you can disable it. For more information, see Managing Amazon DocumentDB cluster TLS settings.

To connect to your Amazon DocumentDB cluster from outside the Amazon VPC, use the following command.

```
mongo --sslAllowInvalidHostnames --ssl --sslCAFile global-bundle.pem --username
 <yourUsername> --password <yourPassword>
```

Connecting to Amazon DocumentDB as a replica set

When you're developing against Amazon DocumentDB (with MongoDB compatibility), we recommend that you connect to your cluster as a replica set and distribute reads to replica instances using the built-in read preference capabilities of your driver. This section goes deeper into what that means and describes how you can connect to your Amazon DocumentDB cluster as a replica set using the SDK for Python as an example.

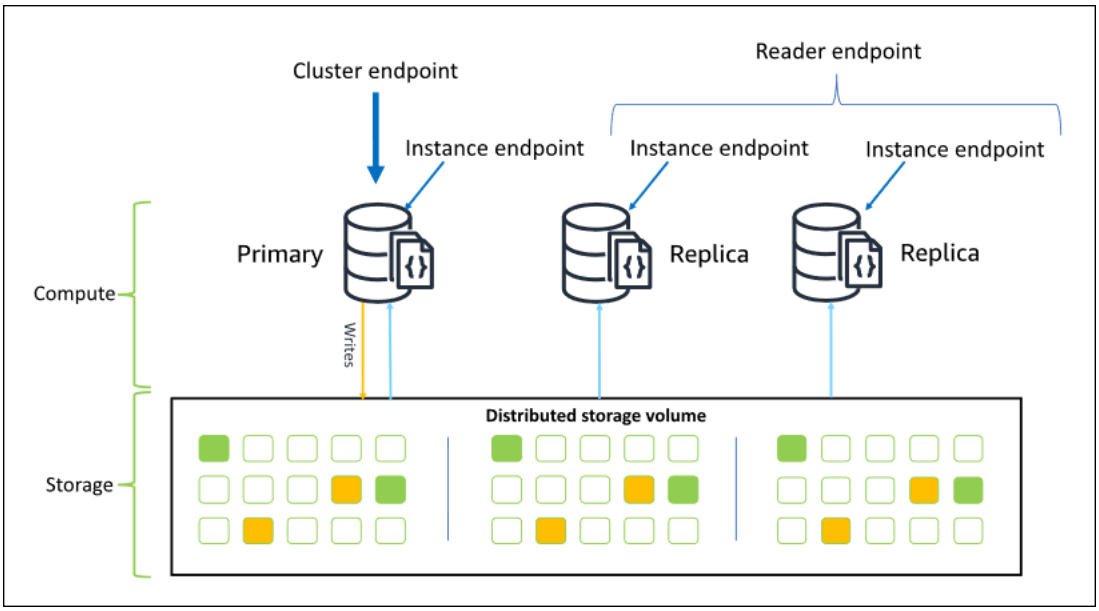
Amazon DocumentDB has three endpoints that you can use to connect to your cluster:

- Cluster endpoint
- Reader endpoint
- Instance endpoints

In most cases when you connect to Amazon DocumentDB, we recommend that you use the cluster endpoint. This is a CNAME that points to the primary instance in your cluster, as shown in the following diagram.

When using an SSH tunnel, we recommend that you connect to your cluster using the cluster endpoint and do not attempt to connect in replica set mode (i.e., specifying `replicaSet=rs0` in your connection string) as it will result in an error.

> **Note**
>
> For more information about Amazon DocumentDB endpoints, see Amazon DocumentDB endpoints.



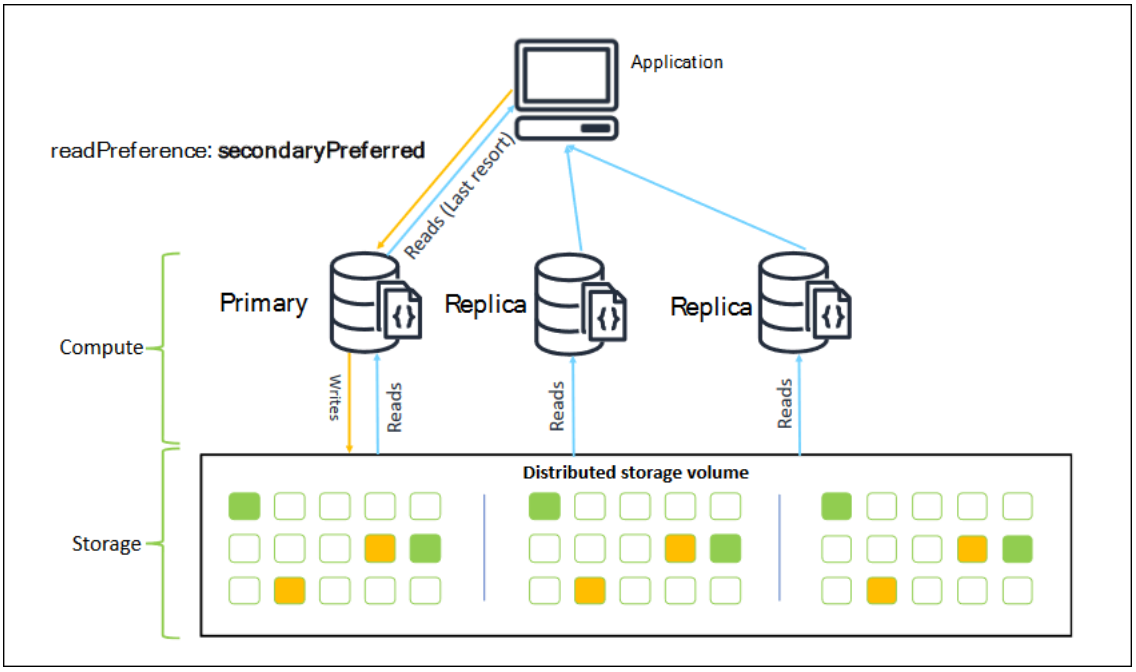
Using the cluster endpoint, you can connect to your cluster in replica set mode. You can then use the built-in read preference driver capabilities. In the following example, specifying `/?replicaSet=rs0` signifies to the SDK that you want to connect as a replica set. If you omit `/?replicaSet=rs0'`, the client routes all requests to the cluster endpoint, that is, your primary instance.

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ## replica set and specify the read preference as secondary preferred
   client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0')
```

The advantage of connecting as a replica set is that it enables your SDK to discover the cluster topography automatically, including when instances are added or removed from the cluster. You can then use your cluster more efficiently by routing read requests to your replica instances.

When you connect as a replica set, you can specify the `readPreference` for the connection. If you specify a read preference of `secondaryPreferred`, the client routes read queries to your replicas and write queries to your primary instance (as in the following diagram). This is a better use of your cluster resources. For more information, see Read preference options.

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0&readPreference=secondaryPreferred')
```



Reads from Amazon DocumentDB replicas are eventually consistent. They return the data in the same order as it was written on the primary, and there is often less than a 50 ms replication lag. You can monitor the replica lag for your cluster using the Amazon CloudWatch metrics `DBInstanceReplicaLag` and `DBClusterReplicaLagMaximum`. For more information, see Monitoring Amazon DocumentDB with CloudWatch.

Unlike traditional monolithic database architecture, Amazon DocumentDB separates storage and compute. Given this modern architecture, we encourage you to read scale on replica instances. Reads on replica instances don't block writes being replicated from the primary instance. You can add up to 15 read replica instances in a cluster and scale out to millions of reads per second.

The key benefit of connecting as a replica set and distributing reads to replicas is that it increases the overall resources in your cluster that are available to do work for your application. We

recommend connecting as a replica set as a best practice. Further, we recommend it most commonly in the following scenarios:

- You're using nearly 100 percent CPU on your primary.
- The buffer cache hit ratio is near zero.
- You reach the connection or cursor limits for an individual instance.

Scaling up a cluster instance size is an option, and in some cases, that can be the best way to scale the cluster. But you should also consider how to better use the replicas that you already have in your cluster. This lets you increase scale without the increased cost of using a larger instance type. We also recommend that you monitor and alert on these limits (that is `CPUUtilization`, `DatabaseConnections`, and `BufferCacheHitRatio`) using CloudWatch alarms so that you know when a resource is being heavily used.

For more information, see the following topics:

- Best practices for Amazon DocumentDB
- Amazon DocumentDB Quotas and limits

Using cluster connections

Consider the scenario of using all the connections in your cluster. For example, an `r5.2xlarge` instance has a limit of 4,500 connections (and 450 open cursors). If you create a three-instance Amazon DocumentDB cluster and connect only to the primary instance using the cluster endpoint, your cluster limits for open connections and cursors are 4,500 and 450 respectively. You might reach these limits if you're building applications that use many workers that get spun up in containers. The containers open up a number of connections all at once and saturate the cluster.

Instead, you could connect to the Amazon DocumentDB cluster as a replica set and distribute your reads to the replica instances. You could then effectively triple the number of available connections and cursors available in the cluster to 13,500 and 1,350 respectively. Adding more instances to the cluster only increases the number of connections and cursors for read workloads. If you need to increase the number of connections for writes to your cluster, we recommend increasing the instance size.

> **Note**
>
> The number of connections for `large`, `xlarge`, and `2xlarge` instances increases with the instance size up to 4,500. The maximum number of connections per instance for `4xlarge` instances or greater is 4,500. For more information on limits by instance types, see Instance limits.

Typically we don't recommend that you connect to your cluster using the read preference of `secondary`. This is because if there are no replica instances in your cluster, the reads fail. For example, suppose that you have a two-instance Amazon DocumentDB cluster with one primary and one replica. If the replica has an issue, read requests from a connection pool that is set as `secondary` fail. The advantage of `secondaryPreferred` is that if the client can't find a suitable replica instance to connect to, it falls back to the primary for reads.

Multiple connection pools

In some scenarios, reads in an application need to have read-after-write consistency, which can be served only from the primary instance in Amazon DocumentDB. In these scenarios, you might create two client connection pools: one for writes and one for reads that need read-after-write consistency. To do that, your code would look something like the following.

```
## Create a MongoDB client,
##    open a connection to Amazon DocumentDB as a replica set and specify the
 readPreference as primary
clientPrimary = pymongo.MongoClient('mongodb://<user-
name>:<password>@mycluster.node.us-east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=primary')

## Create a MongoDB client,
##    open a connection to Amazon DocumentDB as a replica set and specify the
 readPreference as secondaryPreferred
secondaryPreferred = pymongo.MongoClient('mongodb://<user-
name>:<password>@mycluster.node.us-east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=secondaryPreferred')
```

Another option is to create a single connection pool and overwrite the read preference for a given collection.

```
##Specify the collection and set the read preference level for that collection
```

```
col = db.review.with_options(read_preference=ReadPreference.SECONDARY_PREFERRED)
```

Summary

To better use the resources in your cluster, we recommend that you connect to your cluster using the replica set mode. If it's suitable for your application, you can read scale your application by distributing your reads to the replica instances.

Connecting to an Amazon DocumentDB cluster from Studio 3T

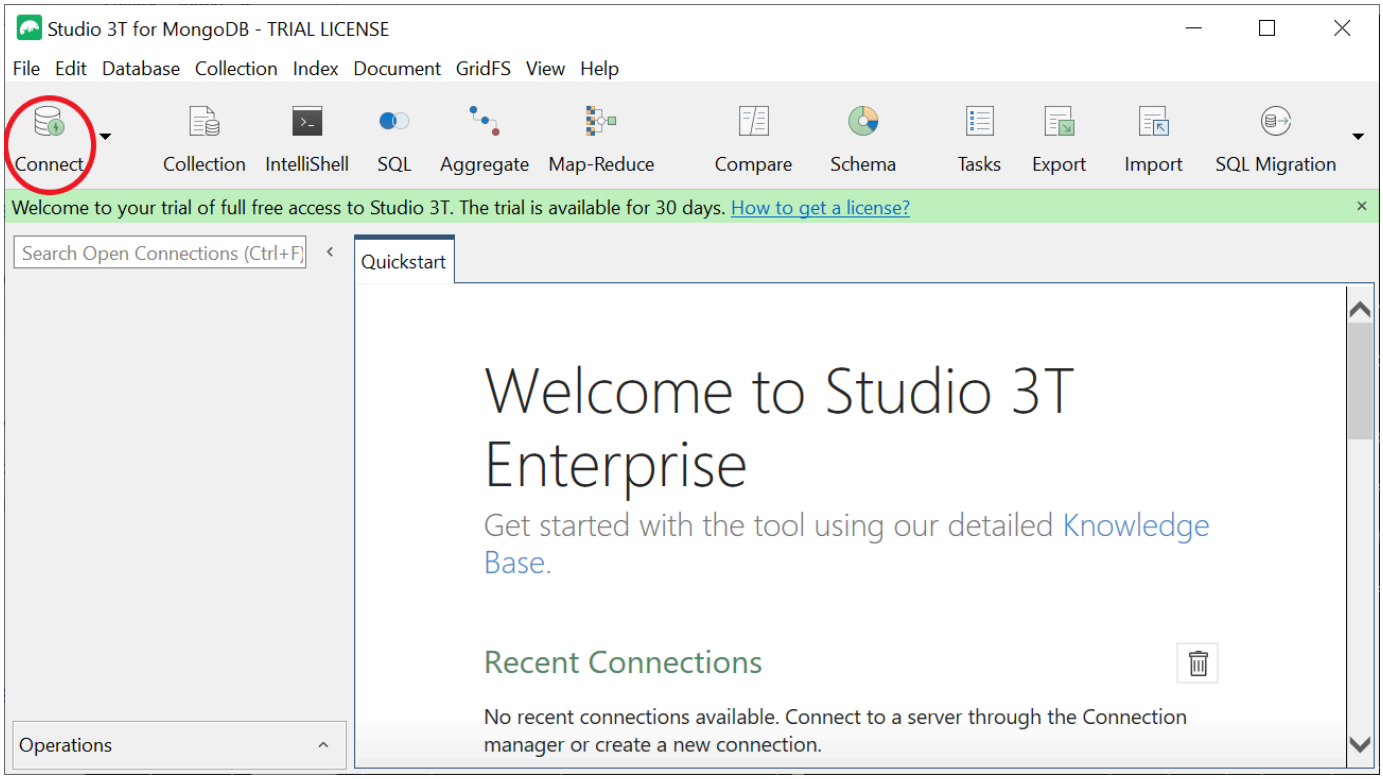
Studio 3T is a popular GUI and IDE for developers and data engineers who work with MongoDB. It offers several powerful capabilities Tree, Table and JSON views of your data, easy import/export in CSV, JSON, SQL and BSON/mongodump, flexible querying option, a visual drag-and-drop UI, a built-in mongo shell with auto-completion, an aggregation pipeline editor, and SQL query support.

Prerequisites

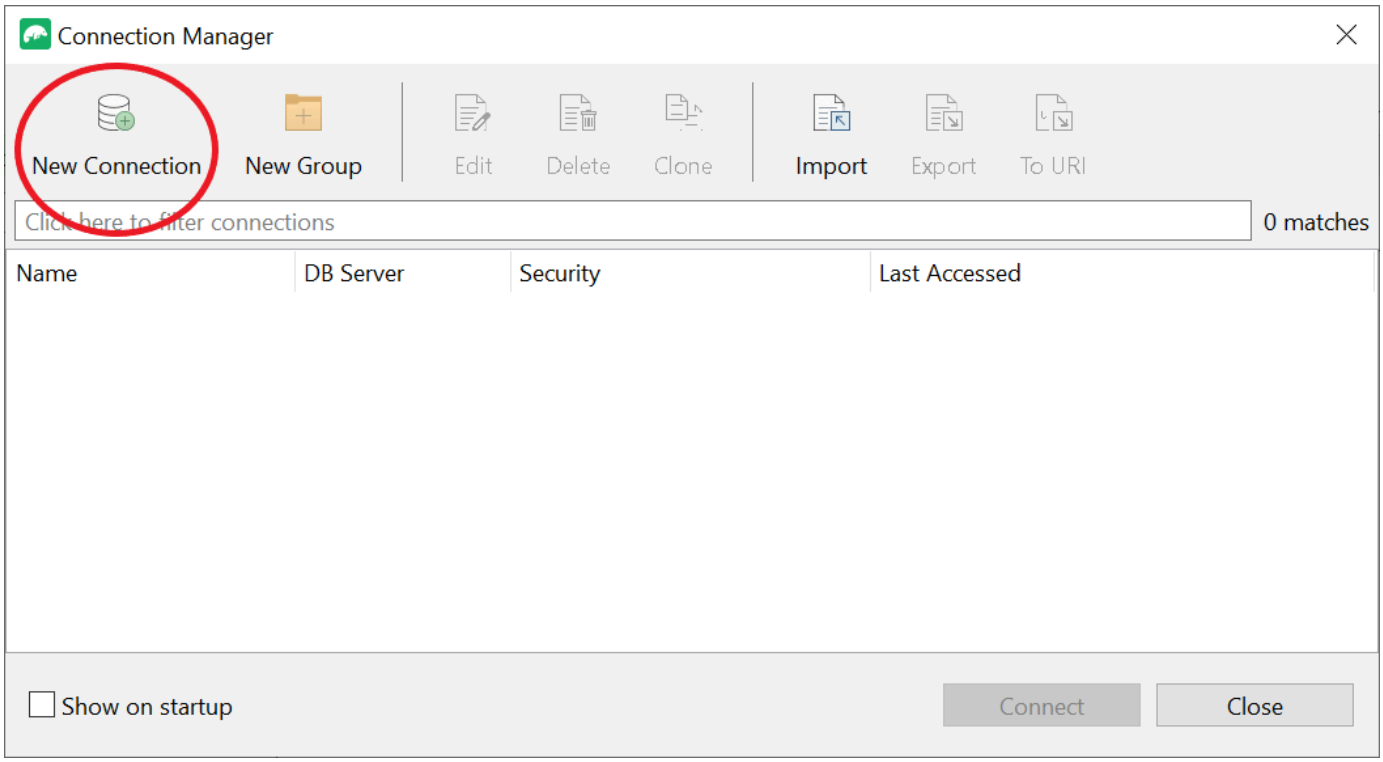
- If you don't already have an Amazon DocumentDB cluster using Amazon EC2 as a bastion/jump host, follow the instructions on how to Connect with Amazon EC2.
- If you don't have Studio 3T, download and install it.

Connect with Studio 3T

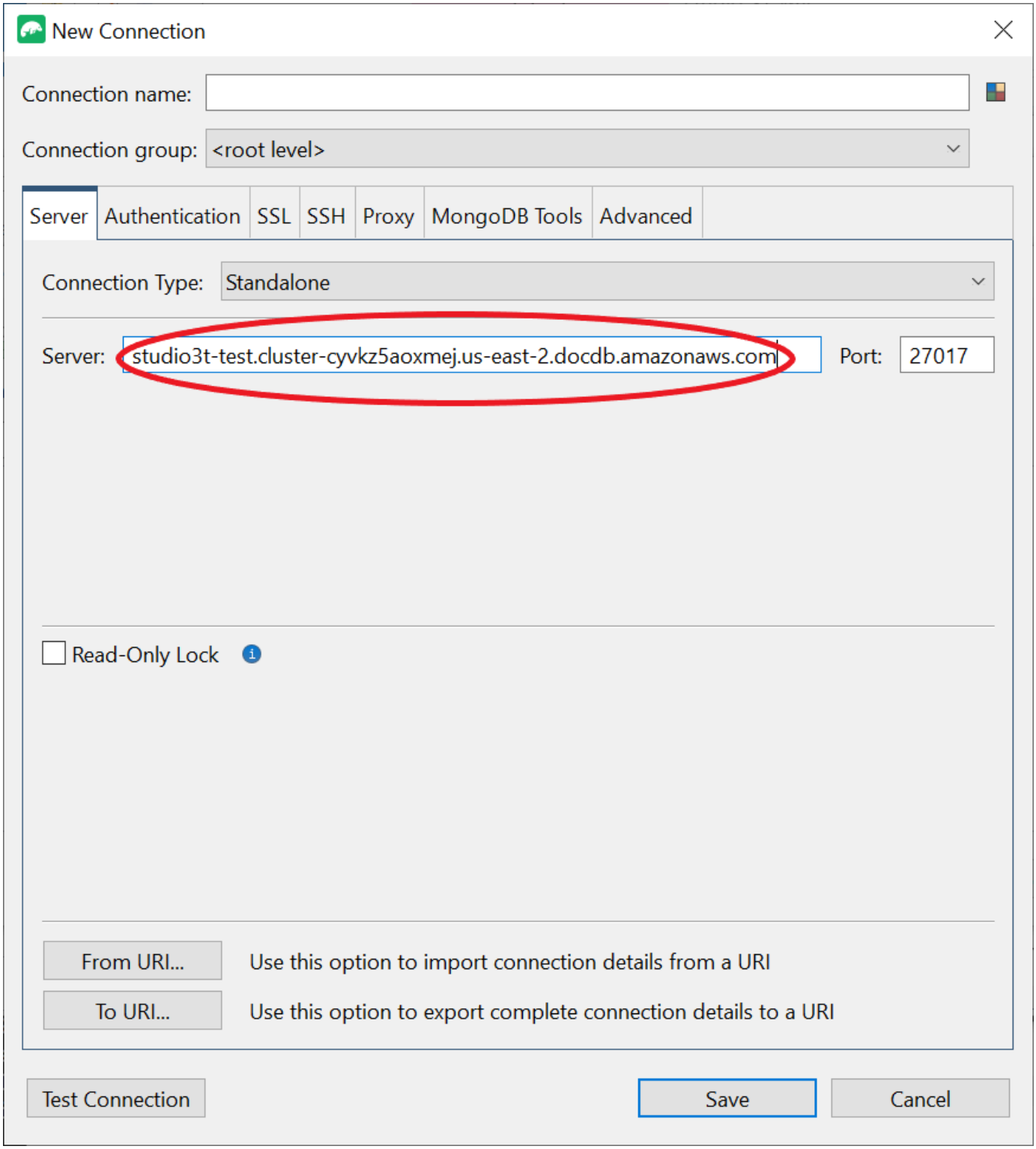
1. Choose **Connect** in the top left corner of the toolbar.



2. Choose **New Connection** in the top left corner of the toolbar.

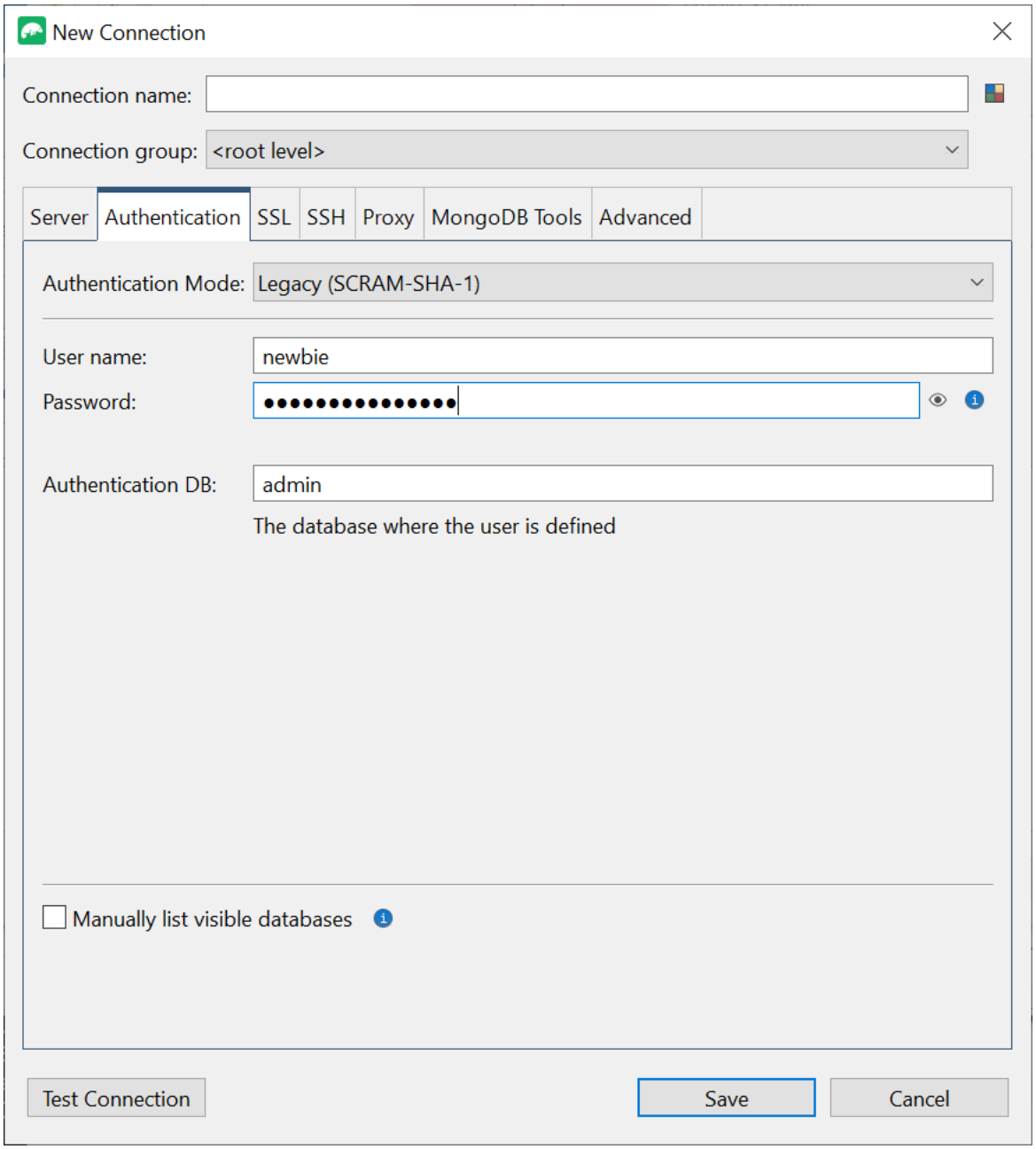


3. On the **Server** tab, in the **Server** field, enter the cluster endpoint information.



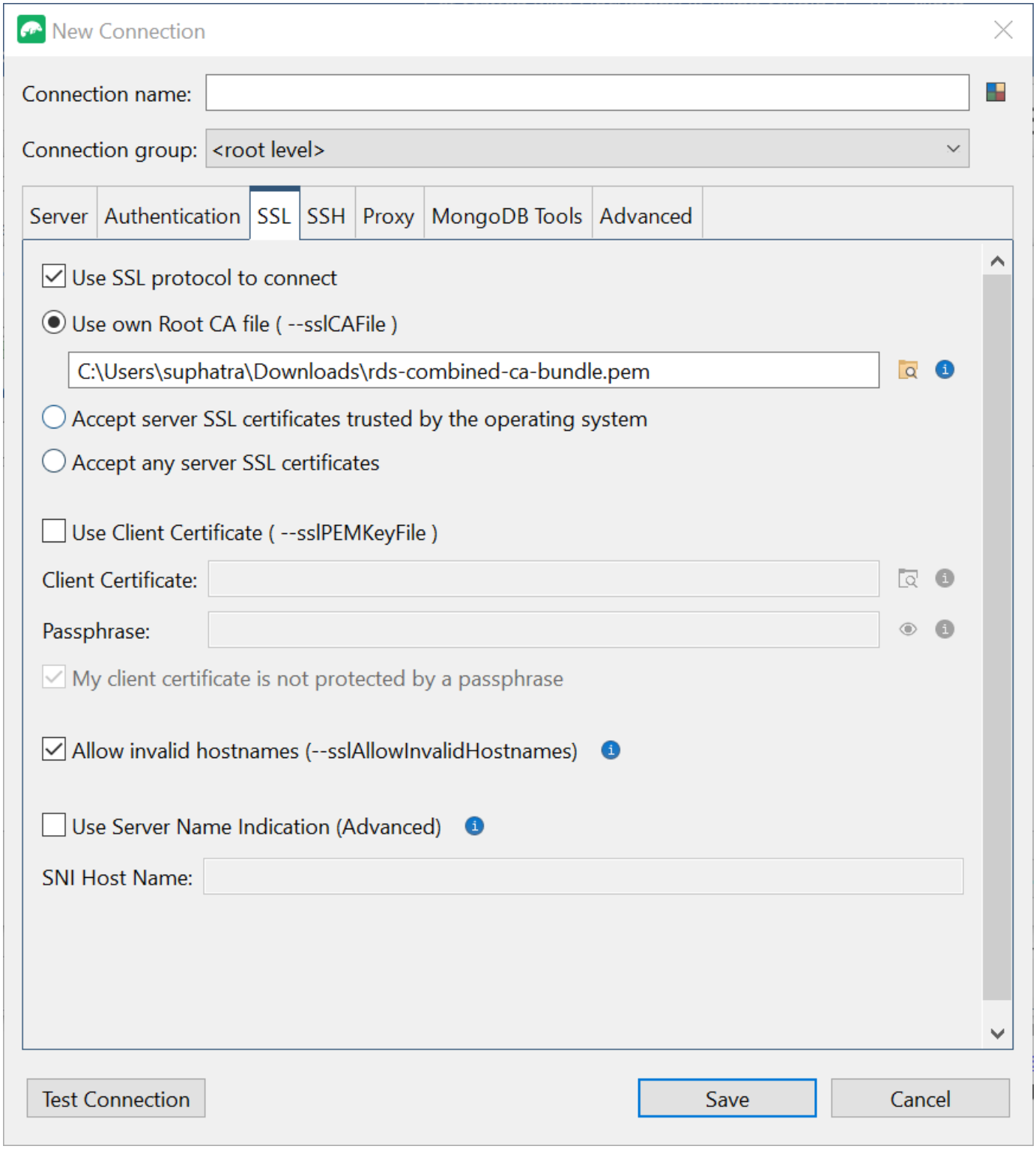
> **Note**
>
> Can't find your cluster endpoint? Just follow the steps here.

4. Choose the **Authentication** tab and select **Legacy** in the drop down menu for **Authentication Mode**.

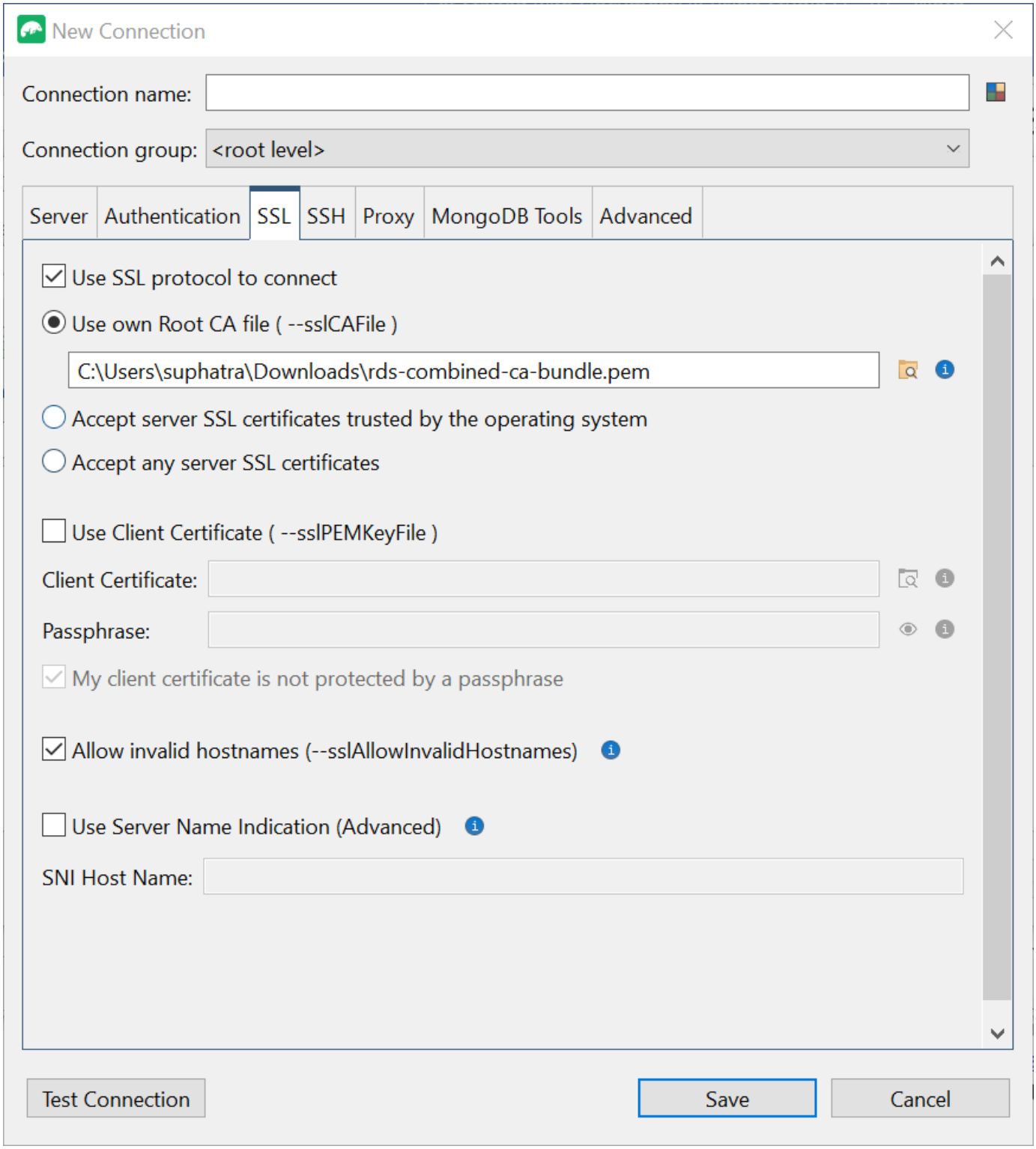


5. Input your username and credentials in the **User name** and **Password** fields.

6. Choose the **SSL** tab and check the box **Use SSL protocol to connect**.



7. Choose **Use own Root CA file**. Then add the Amazon DocumentDB certificate (you can skip this step if SSL is disabled on your DocumentDB cluster). Check the box to allow **invalid hostnames**.



> **Note**
>
> Don't have the certificate? You can download it with the following command:

```
wget https://truststore.pki.rds.amazonaws.com/global/global-
bundle.pem
```

8. If you are connecting from a client machine outside the Amazon VPC, you need to create a SSH tunnel. You will do this in the **SSH** tab.

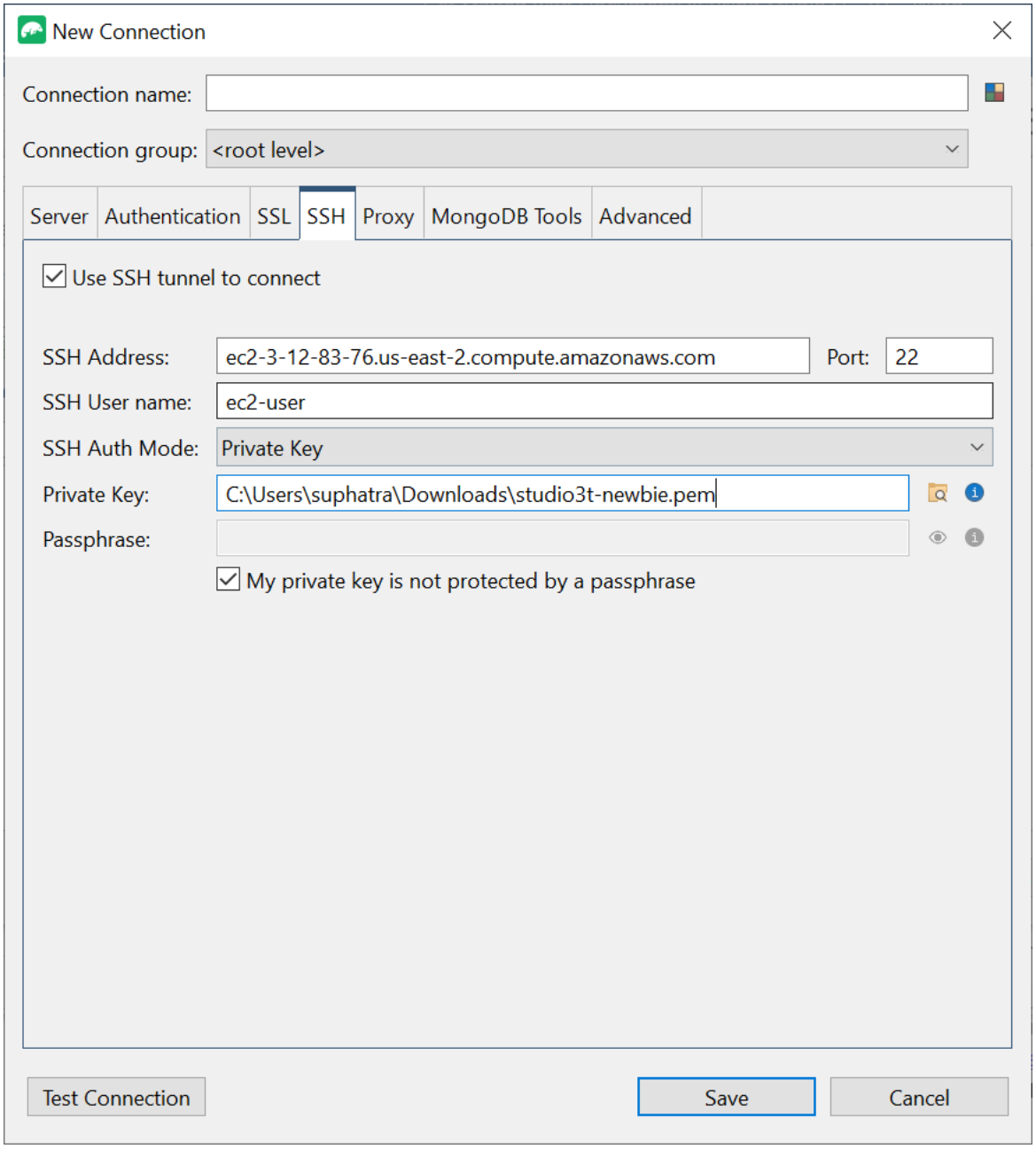
   a. Check the box for **Use SSH tunnel** and input the SSH address in the **SSH Address** field. This is your instance Public DNS (IPV4). You can get this URL from your Amazon EC2 Management Console.

   b. Enter your username. This is the username of your Amazon EC2 instance

   c. For **SSH Auth Mode**, select **Private Key**. In the **Private Key** field, choose the file finder icon to locate and choose the Private key of your Amazon EC2 instance. This is the .pem file (key pair) that you saved while creating your instance in Amazon EC2 Console.

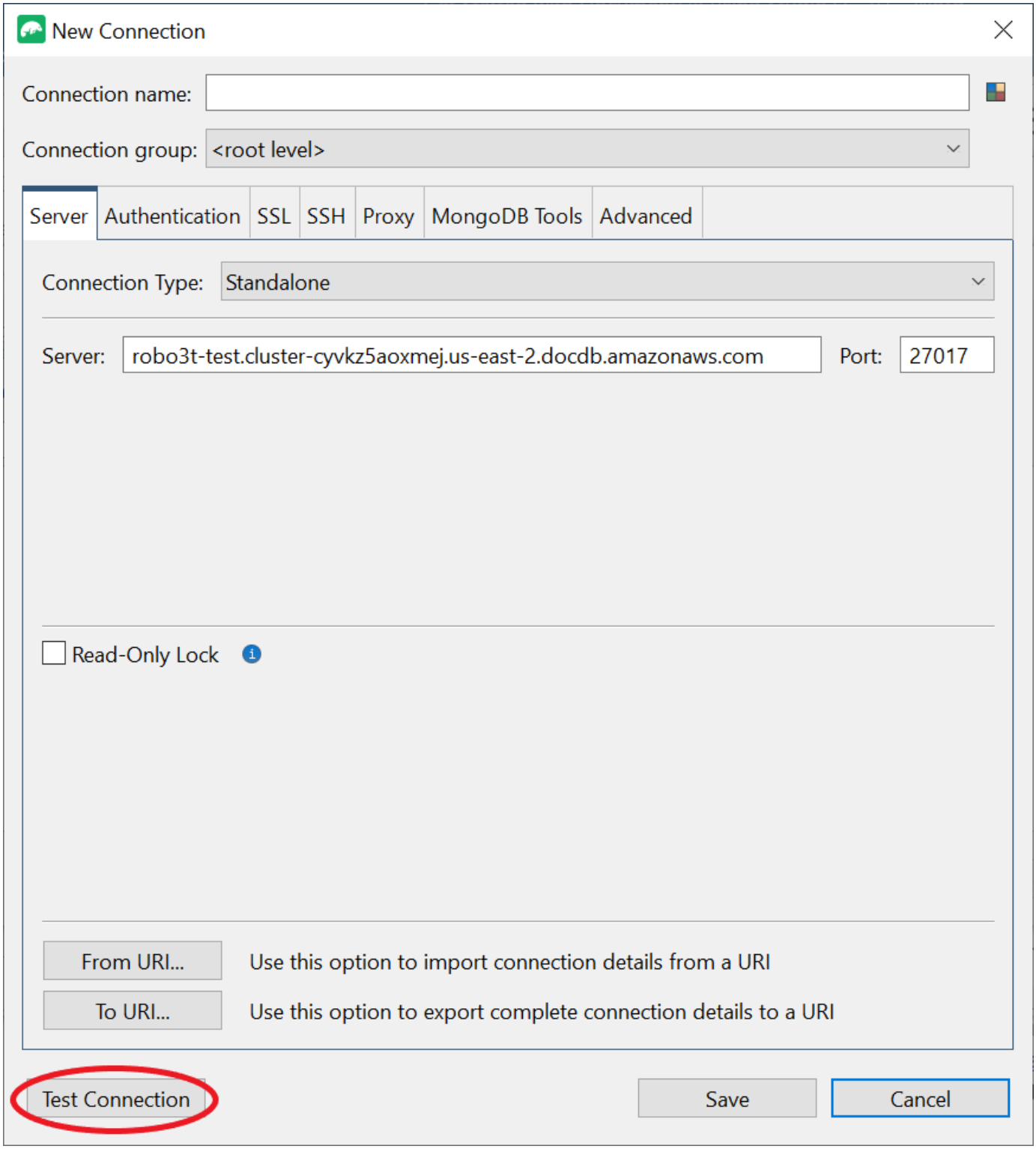
   d. If you are on Linux/macOS client machine, you might have to change the permissions of your private key using the following command:

   ```
   chmod 400 /fullPathToYourPemFile/<yourKey>.pem
   ```

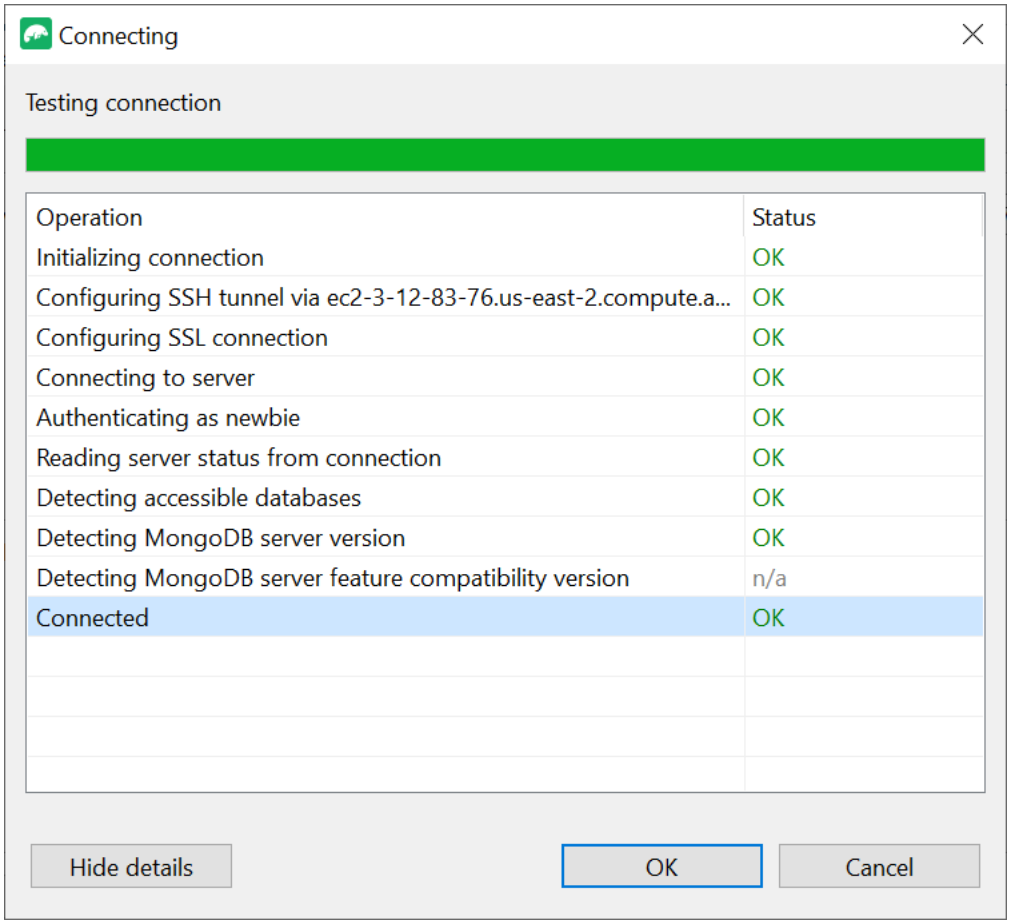


> **Note**
>
> This Amazon EC2 instance should be in the same Amazon VPC and security group as your DocumentDB cluster. You can get the SSH address, username and private key from your Amazon EC2 Management Console.

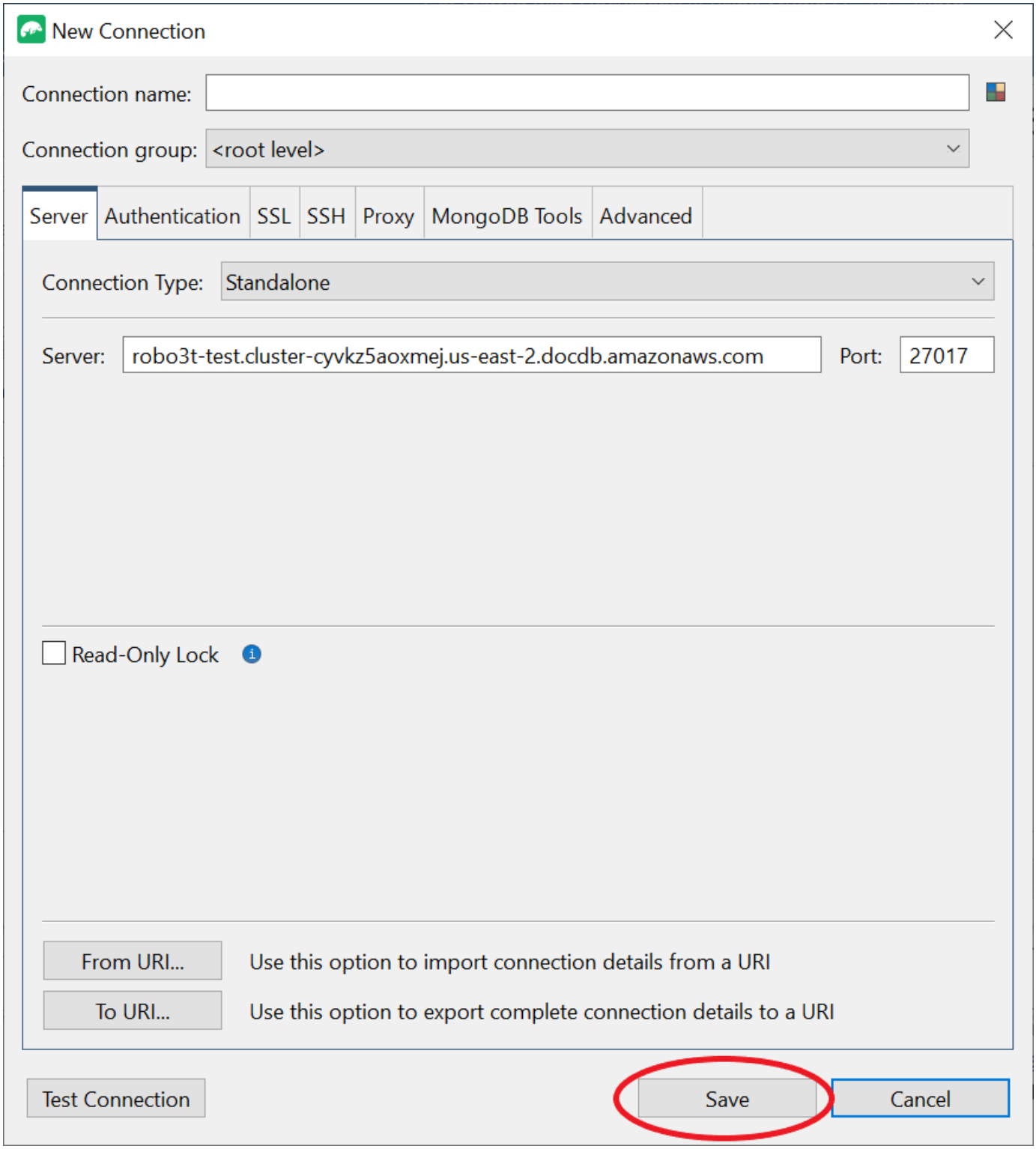
9. Now test your configuration by choosing the **Test connection** button.



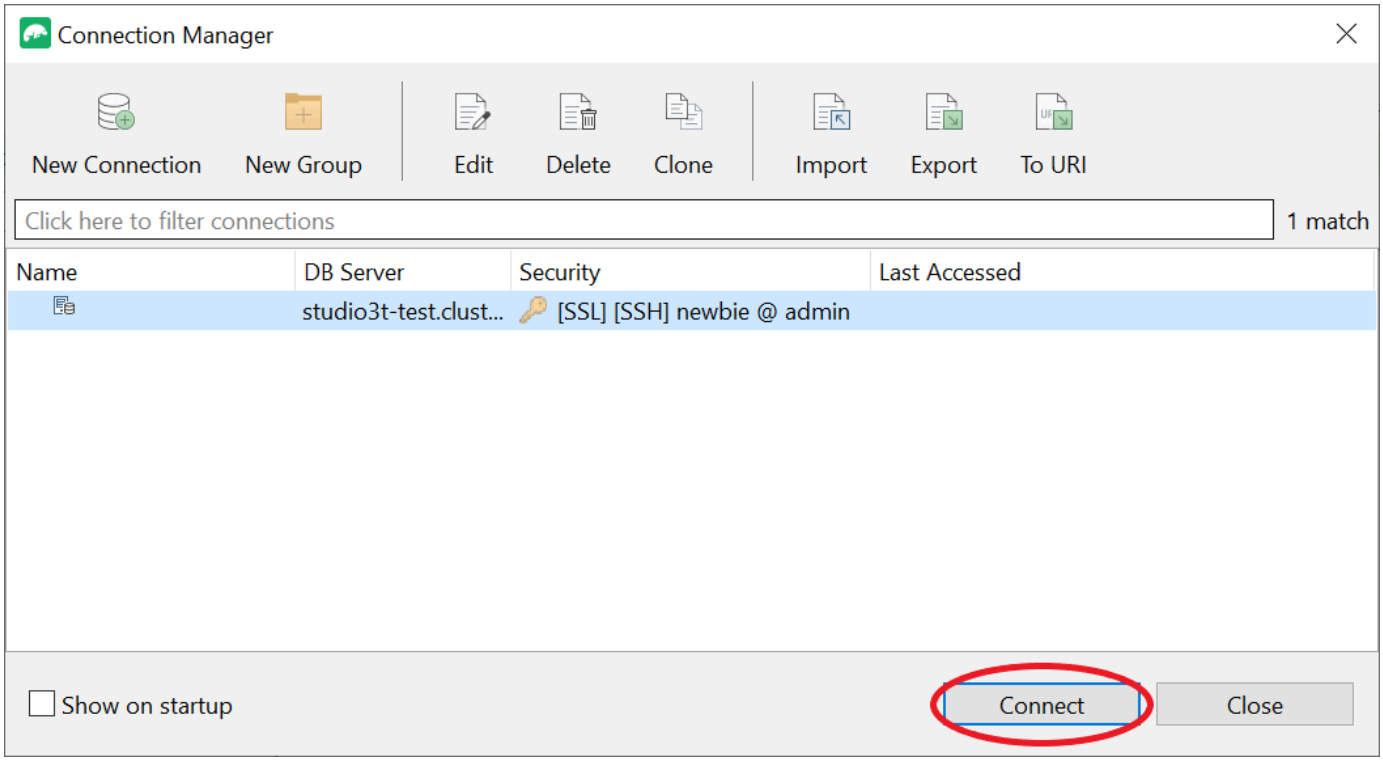
10. A diagnostic window should load a green bar to indicate the test was successful. Now choose **OK** to close out the diagnostic window.



11. Choose **Save** to save your connection for future use.



12. Now select your cluster and choose **Connect**.



Congratulations! You are now successfully connected to your Amazon DocumentDB cluster through Studio 3T.

Connect to Amazon DocumentDB using DataGrip

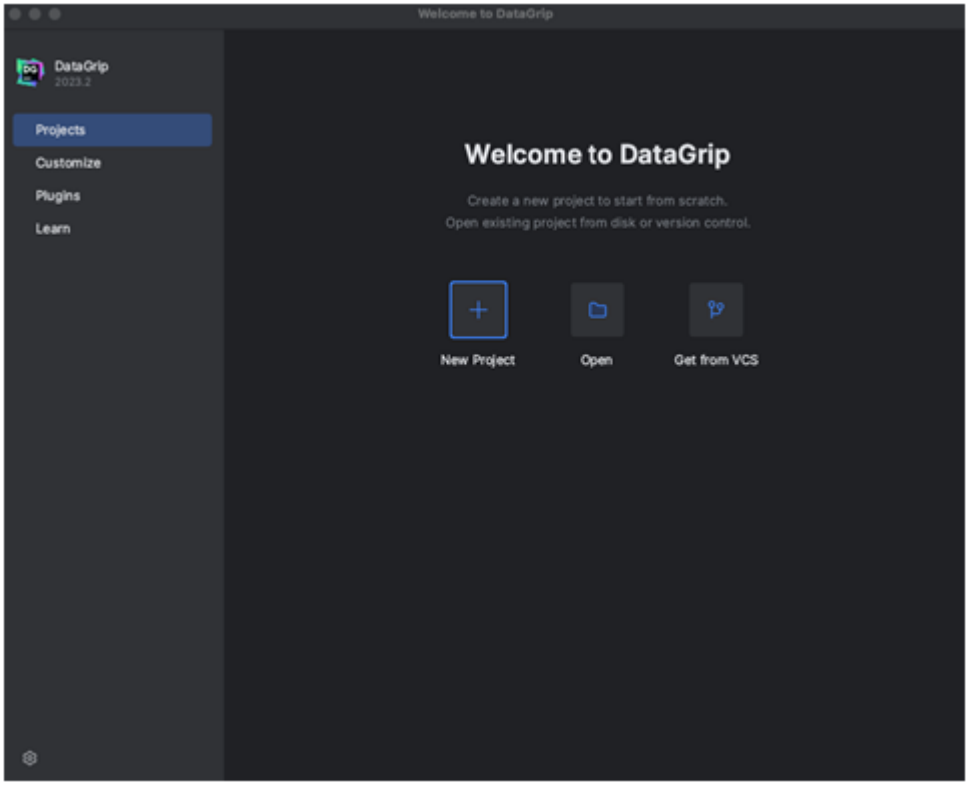
DataGrip is a powerful integrated development environment (IDE) that supports various database systems, including Amazon DocumentDB. This section walks you through the steps to connect to your Amazon DocumentDB cluster using DataGrip, allowing you to easily manage and query your data using a graphical interface.

Prerequisites

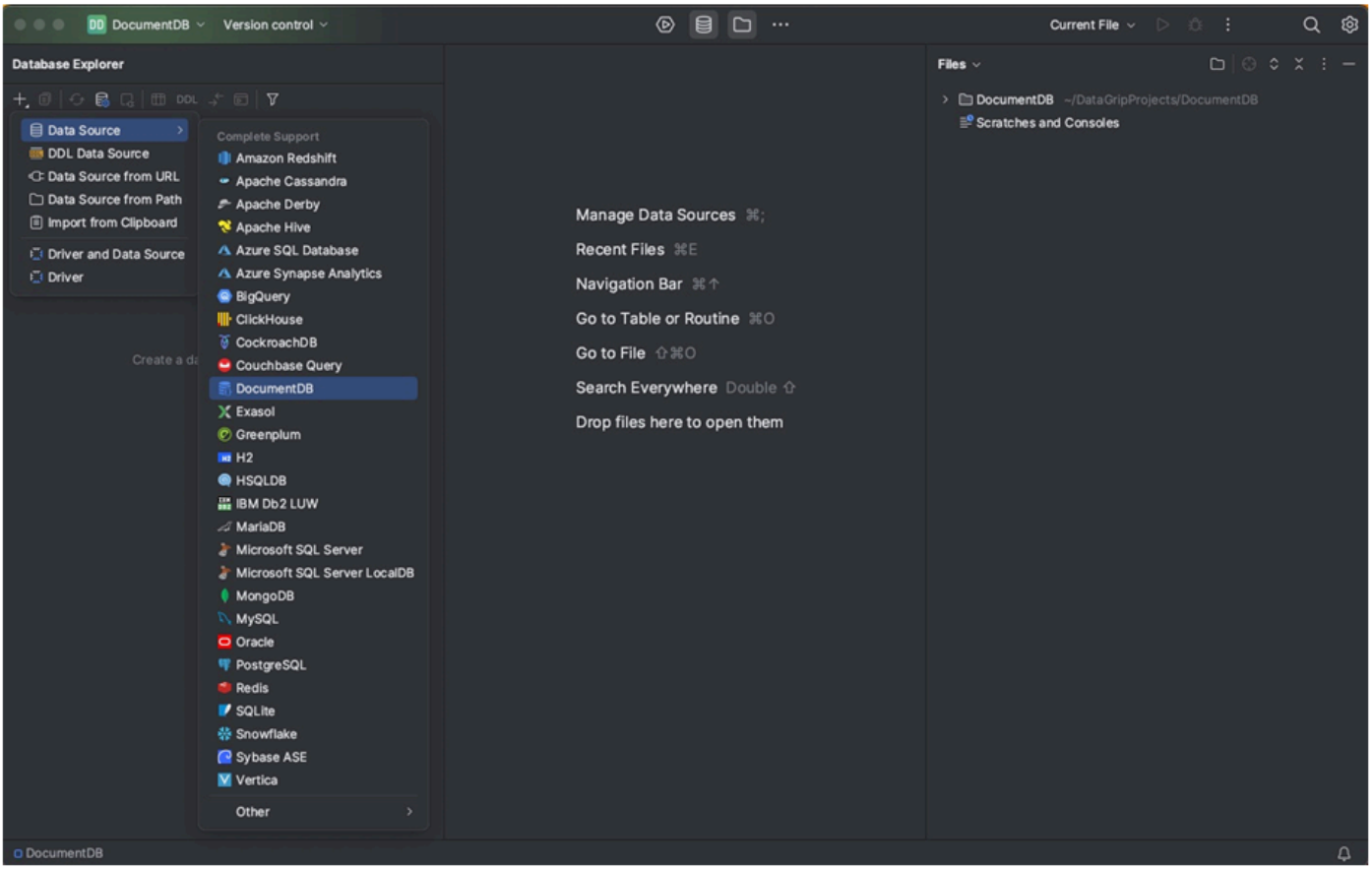
- DataGrip IDE installed on your machine. You can download it from JetBrains.
- An Amazon EC2 instance running in the same VPC as your Amazon DocumentDB cluster. You'll use this instance to establish a secure tunnel from your local machine to the Amazon DocumentDBcluster. Follow the instructions on how to Connect using Amazon EC2.
- Alternative to an Amazon EC2 instance, a VPN connection, or if you are already accessing your AWS infrastructure using a secure VPN. If you prefer this option, follow the instructions to securely access Amazon DocumentDB using AWS Client VPN.

Connect using DataGrip

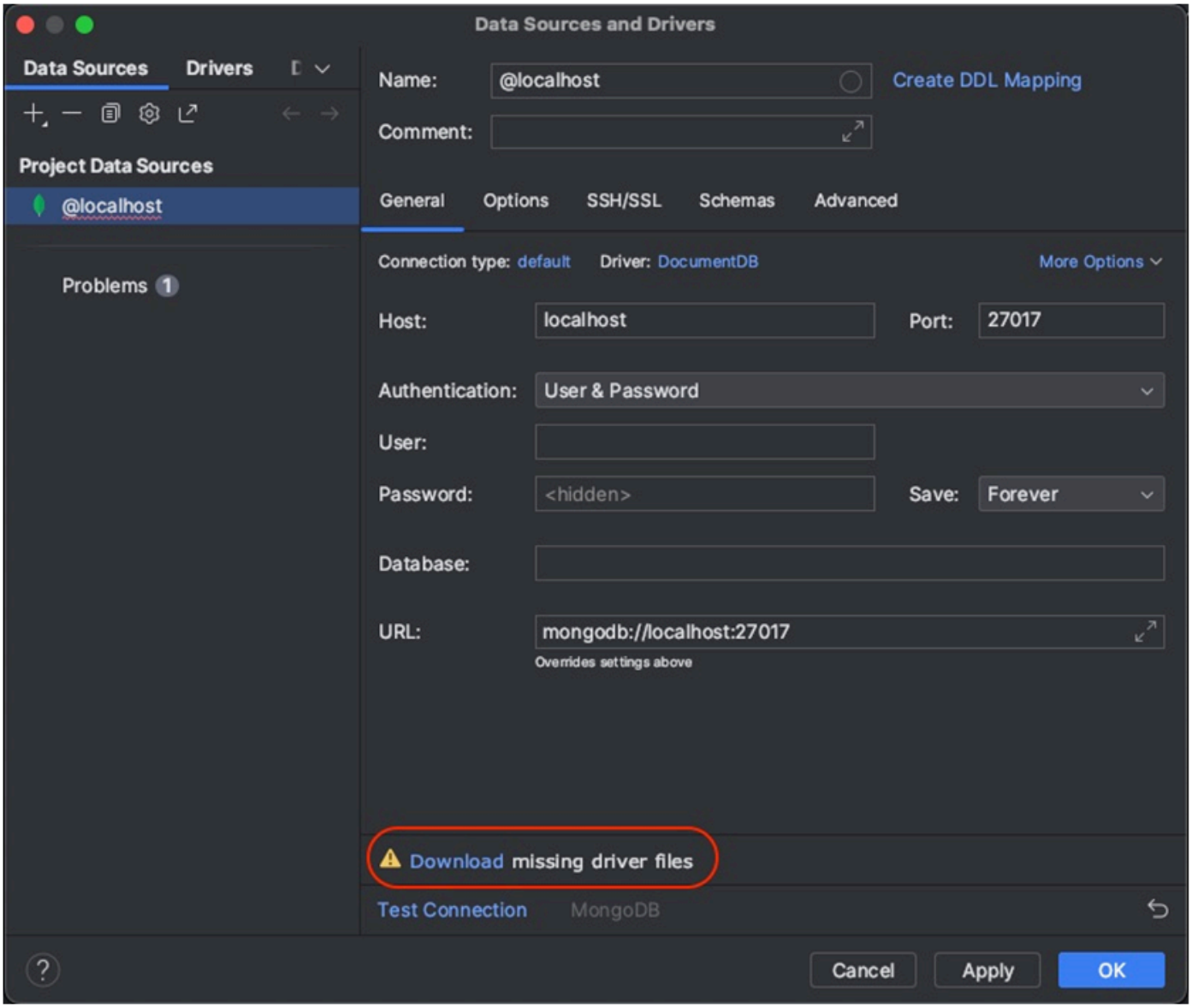
1. Launch DataGrip on your computer and create a **New Project**.



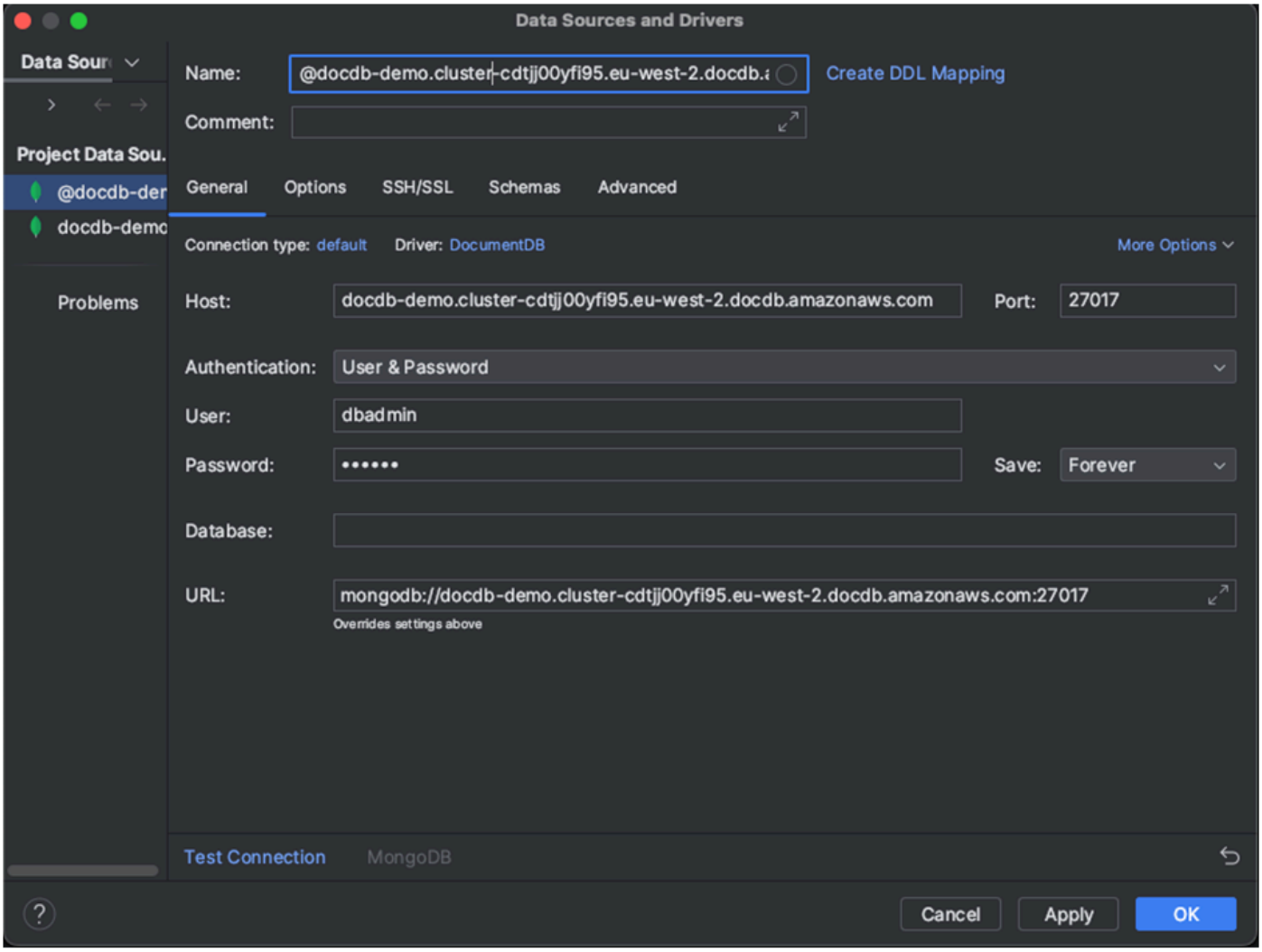
2. Add a new data source using one of the following ways:
   a. From the main menu, navigate to **File – New – Data Source** and select **DocumentDB**
   b. In the **Database Explorer**, click the new icon (**+**) in the toolbar. Navigate to **Data Source** and select **DocumentDB**.



3. On the **Data Sources** page in the **General** tab, check if there is a **Download missing driver files** link at the bottom of the connection settings area. Click this link to download drivers that are required to interact with a database. For a direct download link, refer to JetBrains JDBC drivers.

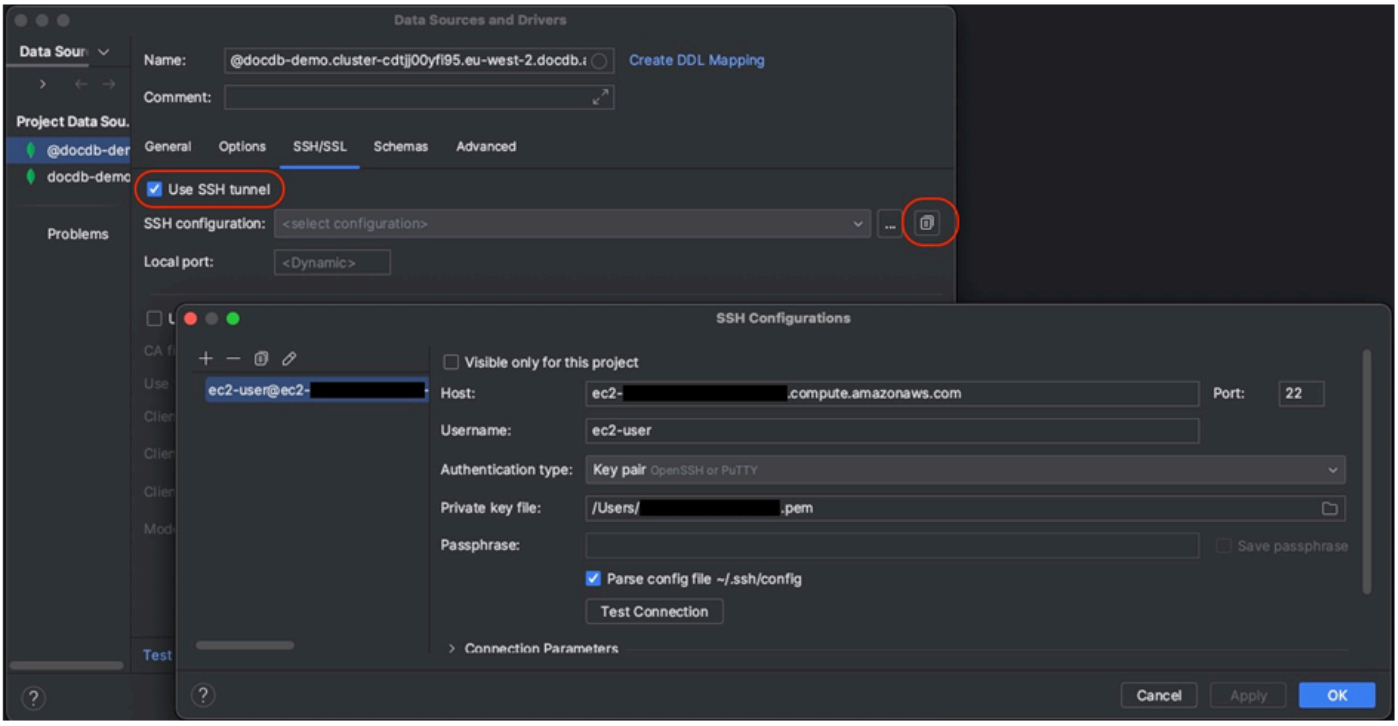


4. In the **General** tab, specify the connection details:
   a. In the **Host** field, specify the Amazon DocumentDB cluster endpoint.
   b. **Port** is already set to 27017. Change it if your cluster was deployed on a different port.
   c. For **Authentication**, choose **User & Password**.
   d. Enter your user name and password information.
   e. The **Database** field is optional. You can specify the database to which you want to connect.
   f. The **URL** field auto-completes as you add the above details.



5. In the **SSH/SSL** tab, enable **Use SSH tunnel**, then click on the icon to open the **SSH Configuration** dialog. Enter the following information:
   a. in the **Host** field, enter the hostname of your Amazon EC2 instance.
   b. Enter the username and password for your Amazon EC2 instance.
   c. For **Authentication type**, choose **Key pair**.
   d. Enter your **Private key file**.

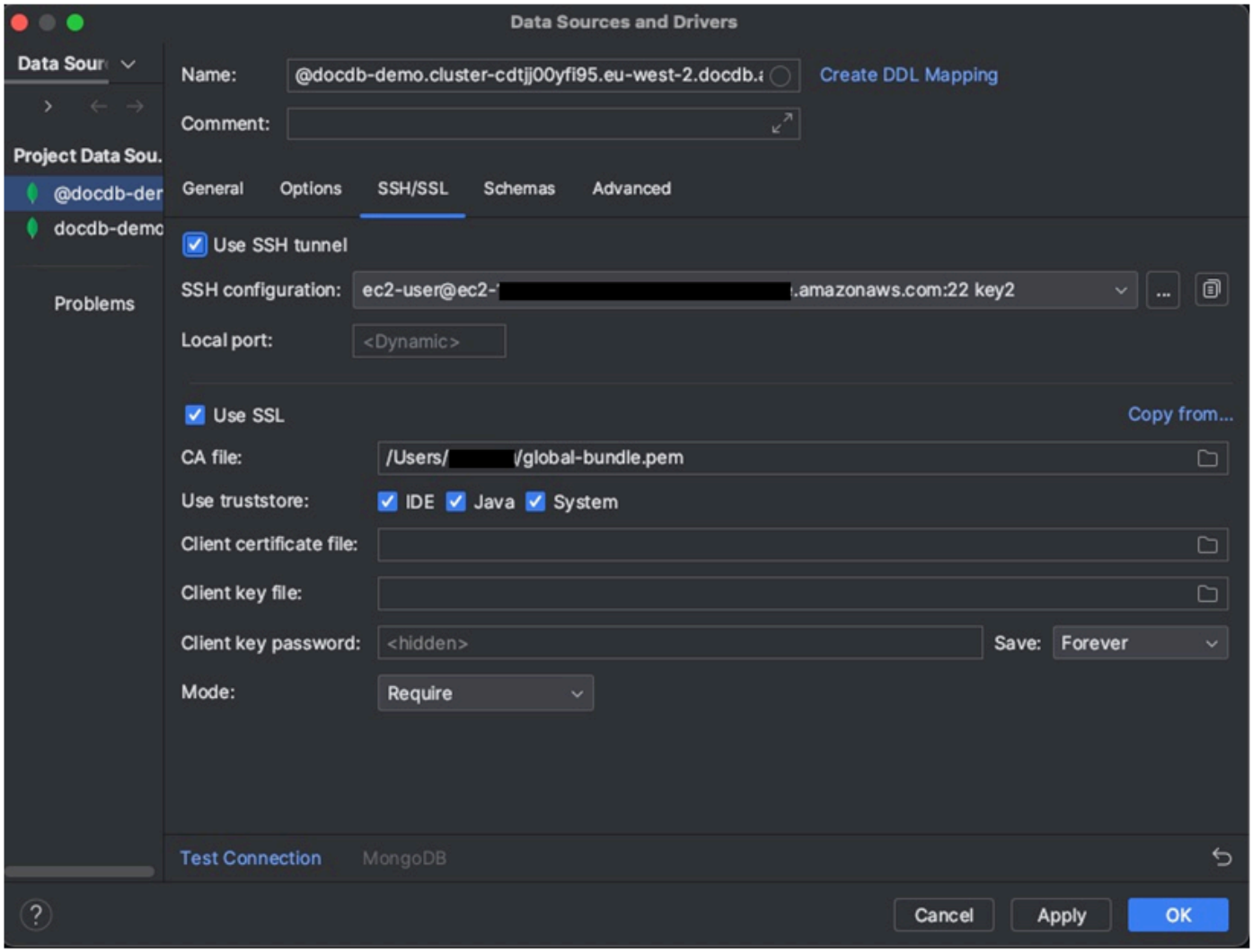
> **Note**
>
> If you're using the VPN option, there is no need to configure the SSH tunnel.



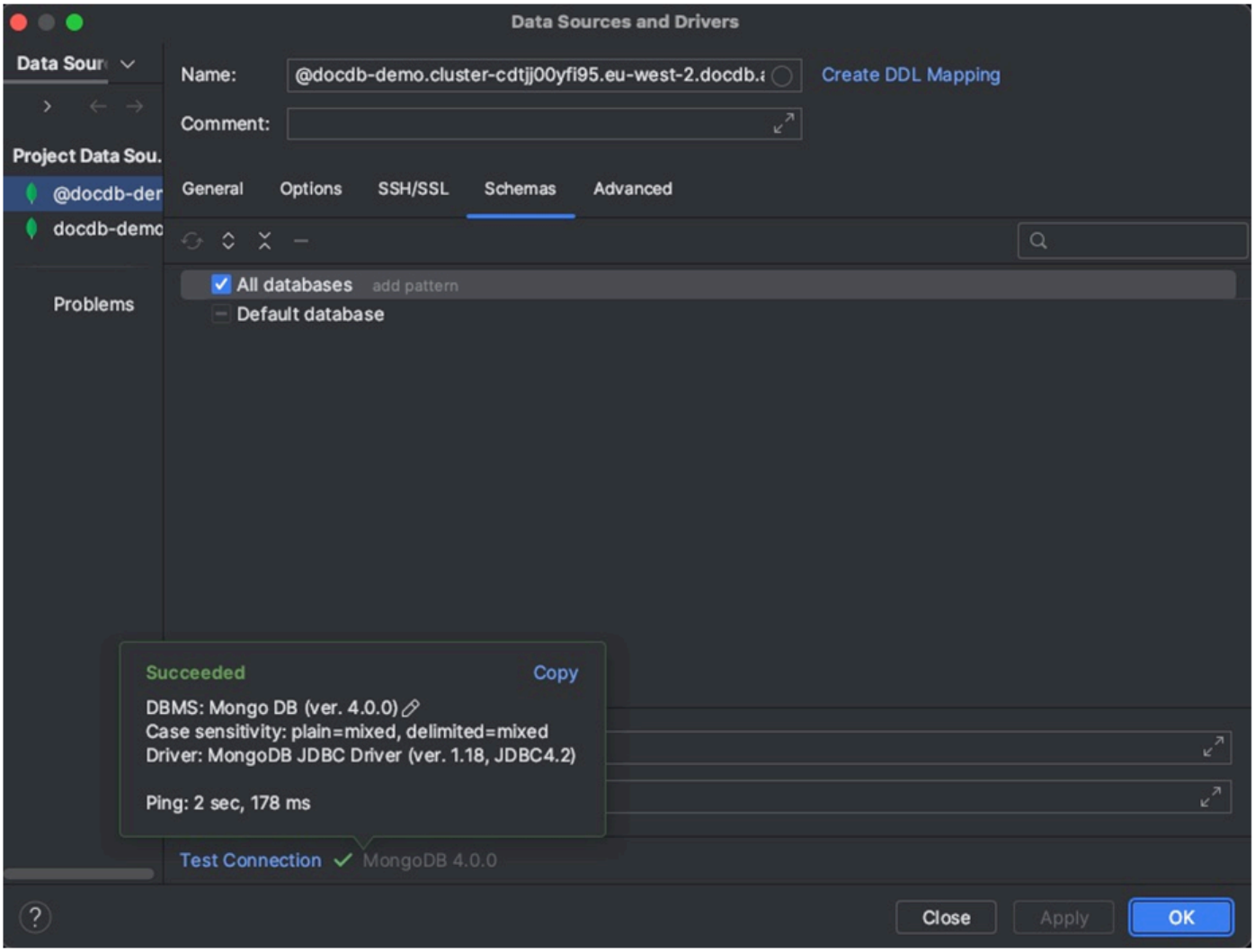
6. In the **SSH/SSL** tab, enable **Use SSL**. In the **CA file** field, enter the location to the `global-bundle.pem` file on your computer. For **Mode**, leave the option **Require**.

> **Note**
>
> You can download the certificate from this location or with this command: wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem

> **Note**
>
> If you are connecting to Amazon DocumentDB elastic cluster, you don't have to specify the CA file. Leave the **Use SSL** option checked and all the other options at their default values.



7. In the **Schemas** tab, choose **All databases** or enter the filter "*:*" in the **Schema pattern** field. Click on the **Test Connection** link to test the connection.



8. Once the connection is successfully tested, click **OK** to save the data source configuration.

DataGrip features

DataGrip provides various features to help you work with Amazon DocumentDB efficiently:

- **SQL Editor** — Write and execute SQL-like queries on your DocumentDB collections using the SQL editor in DataGrip.
- **Visual Query Builder** — Use the visual query builder to create queries graphically without writing SQL code.
- **Schema Management** — Easily manage your database schema, including creating, altering, and dropping collections.

- **Data Visualization** — View and analyze your data using various visualization tools available in DataGrip.
- **Export and Import Data** — Transfer data between Amazon DocumentDB and other databases using DataGrip's export and import features.

Refer to the official DataGrip documentation for more advanced features and tips on working with Amazon DocumentDB and other database systems.

Connect using Amazon EC2

This section describes how to set up connectivity between an Amazon DocumentDB cluster and Amazon EC2 and access the Amazon DocumentDB cluster from the Amazon EC2 instance.

There are two options for configuring the EC2 connection:

- Automatically connect your EC2 instance to an Amazon DocumentDB database — Use the automatic connection feature in the EC2 console to automatically configure the connection between your EC2 instance and a new or existing Amazon DocumentDB database. This connection allows traffic to travel between the EC2 instance and the Amazon DocumentDB database. This option is typically used for testing and creating new security groups.
- Manually connect your EC2 instance to your Amazon DocumentDB database — Configure the connection between your EC2 instance to your Amazon DocumentDB database by manually configuring and assigning the security groups to reproduce the configuration that is created by the automatic connection feature. This option is typically used for changing more advanced settings and using exisitng security groups.

Prerequisites

Regardless of the option, and before you create your first Amazon DocumentDB cluster, you must do the following:

Create an Amazon Web Services (AWS) account

Before you can begin using Amazon DocumentDB, you must have an Amazon Web Services (AWS) account. The AWS account is free. You pay only for the services and resources that you use.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

Optionally, set up the needed AWS Identity and Access Management (IAM) permissions.

Access to manage Amazon DocumentDB resources such as clusters, instances, and cluster parameter groups requires credentials that AWS can use to authenticate your requests. For more information, see Identity and Access Management for Amazon DocumentDB.

1. In the search bar of the AWS Management Console, type in IAM and select **IAM** in the drop down menu that appears.
2. Once you're in the IAM console, select **Users** from the navigation pane.
3. Select your username.
4. Click the button **Add permissions**.
5. Select **Attach existing policies directly**.
6. Type `AmazonDocDBFullAccess` in the search bar and select it once it appears in the search results.
7. Click the blue button at the bottom that says **Next: Review**.
8. Click the blue button at the bottom that says **Add permissions**.

Create an Amazon Virtual Private Cloud (Amazon VPC)

Depending on which AWS Region you are in, you may or may not have a default VPC already created. If you don't have a default VPC, complete step 1 of the Getting Started with Amazon VPC in the *Amazon VPC User Guide*. This will take less than five minutes.

Connect Amazon EC2 automatically

Topics

- Automatically connect an EC2 instance to a new Amazon DocumentDB database
- Automatically connect an EC2 instance to an existing Amazon DocumentDB database
- Overview of automatic connectivity with an EC2 instance
- Viewing connected compute resources

Before setting up a connection between an EC2 instance and a new Amazon DocumentDB database, make sure you meet the requirements described in Overview of automatic connectivity with an EC2 instance. If you make changes to security groups after you configure connectivity, the changes might affect the connection between the EC2 instance and the Amazon DocumentDB database.

> **Note**
>
> You can only set up a connection between an EC2 instance and an Amazon DocumentDB database automatically by using the AWS Management Console. You can't set up a connection automatically with the AWS CLI or Amazon DocumentDB API.

Automatically connect an EC2 instance to a new Amazon DocumentDB database

The following process assume you have completed the steps in the Prerequisites topic.

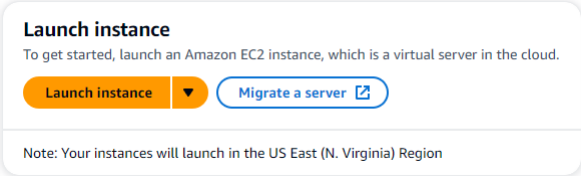
Steps

- Step 1: Create an Amazon EC2 instance
- Step 2: Create an Amazon DocumentDB cluster
- Step 3: Connect to your Amazon EC2 instance
- Step 4: Install the MongoDB Shell
- Step 5: Manage Amazon DocumentDB TLS
- Step 6: Connect to your Amazon DocumentDB cluster
- Step 7: Insert and query data
- Step 8: Explore

Step 1: Create an Amazon EC2 instance

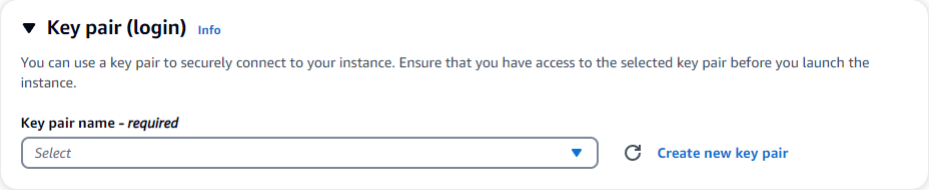
In this step, you will create an Amazon EC2 instance in the same Region and Amazon VPC that you will later use to provision your Amazon DocumentDB cluster.

1. On the Amazon EC2 console, choose **Launch instance**.



2. Enter a name or identifier in the **Name** field located in the **Name and tags** section.
3. In the **Amazon Machine Image (AMI)** drop-down list, locate **Amazon Linux 2 AMI** and choose it.

4. Locate and choose **t3.micro** in the **Instance type** drop-down list.
5. In the **Key pair (login)** section, enter the identifier of an existing key-pair, or choose **Create new key pair**.



   You must provide an Amazon EC2 key pair.

   If you do have an Amazon EC2 key pair:

   a. Select a key pair, choose your key pair from the list.
   b. You must already have the private key file (.pem or .ppk file) available to log in to your Amazon EC2 instance.

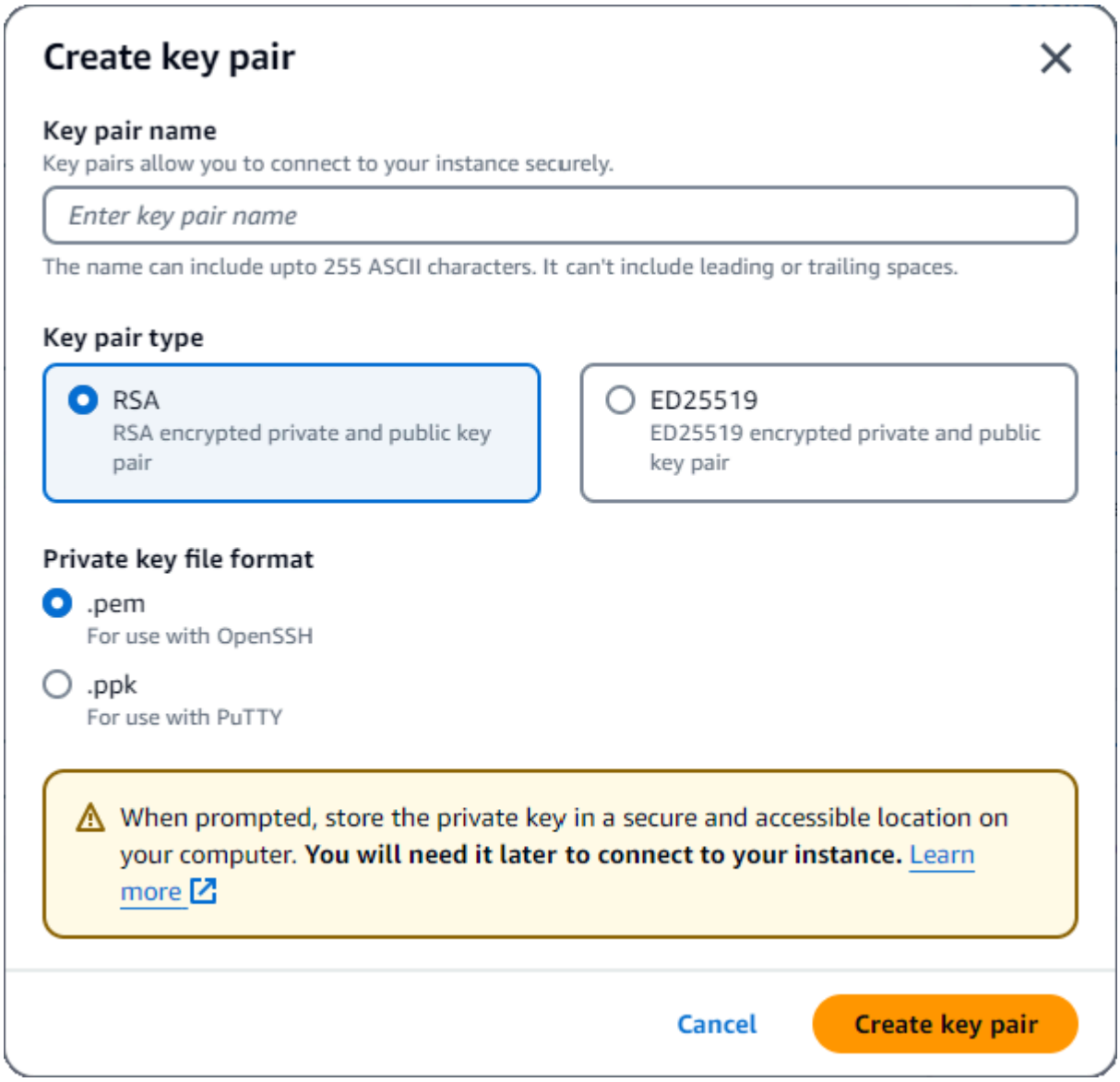
   If you do not have an Amazon EC2 key pair:

   a. Choose **Create new key pair**, the **Create key pair** dialog box appears.

b. Enter a name in the **Key pair name** field.

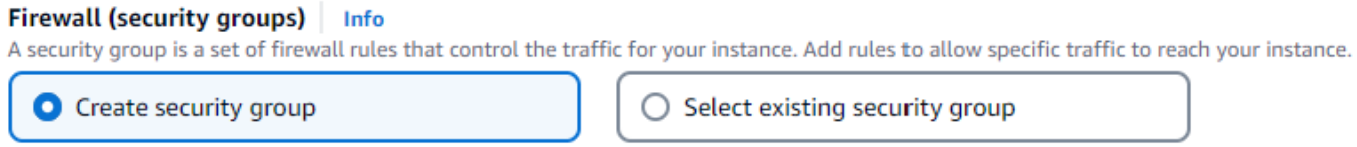
c. Choose the **Key pair type** and **Private key file format**.

d. Choose **Create key pair**.



> **Note**
>
> For security purposes, we highly recommend using a key-pair for both SSH and internet connectivity to your EC2 instance.

6. **Optional:** In the **Network settings section**, under **Firewall (security groups)**, choose **Create security group**.



   Choose **Create security group** (check all the traffic allow rules that apply to your EC2 connectivity).

   > **Note**
   >
   > If you want to use an existing security group, follow the instructions in Connect Amazon EC2 manually.

7. In the **Summary** section, review your EC2 configuration and choose **Launch instance** if correct.

Step 2: Create an Amazon DocumentDB cluster

While the Amazon EC2 instance is being provisioned, create your Amazon DocumentDB cluster.

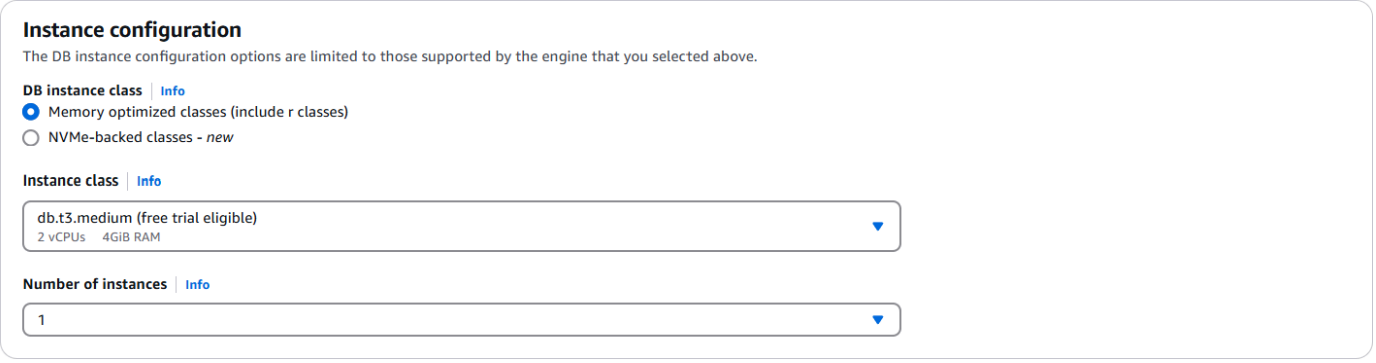
1. Navigate to the Amazon DocumentDB console and choose **Clusters** from the navigation pane.
2. Choose **Create**.
3. Leave the **Cluster type** setting at it's default of **Instance Based Cluster**.
4. In **Cluster configuration**, for **Cluster identifier**, enter a unique name. Note that the console will change all cluster names into lower-case regardless of how they are entered.

   Leave the **Engine version** at it's default value of **5.0.0**.
5. For **Cluster storage configuration**, leave the default setting of **Amazon DocumentDB Standard**.
6. In **Instance configuration**:

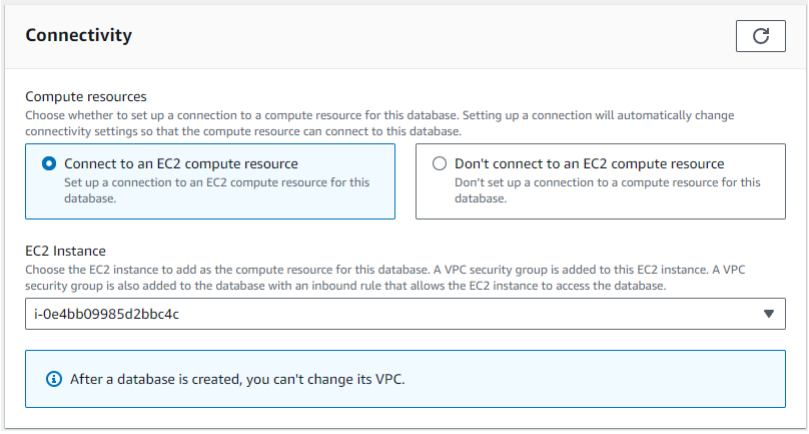
   - For **DB instance class**, choose **Memory optimized classes (include r classes)** (this is default).

     The other instance option is **NVMe-backed classes**. To learn more, see NVMe-backed instances.
   - For **Instance class**, choose the instance type that suits your needs. For a more detailed explanation of instance classes, see Instance class specifications.

- For **number of instances**, choose a number that best reflects your needs. Remember, the lower the number, the lower the cost, and the lower the read/write volume that can be managed by the cluster.



7. For **Connectivity**, choose **Connect to an EC2 compute resource**. This is the EC2 instance you created in Step 1.



> **Note**
>
> Connecting to an EC2 compute resource automatically creates a security group for your EC2 compute resource connection to your Amazon DocumentDB cluster. When you have completed creating your cluster and you want to see the newly created security group, navigate to the cluster list and choose your cluster's identifier. In the **Connectivity & security** tab, go to **Security Groups** and find your group under **Security group name (ID)**. It will contain the name of your cluster and look similar to this: `docdb-ec2-docdb-2023-12-11-21-33-41:i-0e4bb09985d2bbc4c (sg-0238e0b0bf0f73877)`.

8. In the **Authentication** section, enter a username for the primary user, and then choose **Self managed**. Enter a password, then confirm it.

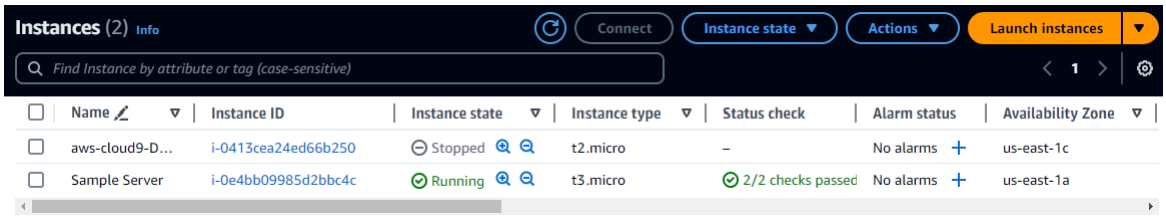
   If you instead chose **Managed in AWS Secrets Manager**, see Password management with Amazon DocumentDB and AWS Secrets Manager for more information.

9. Choose **Create cluster**.

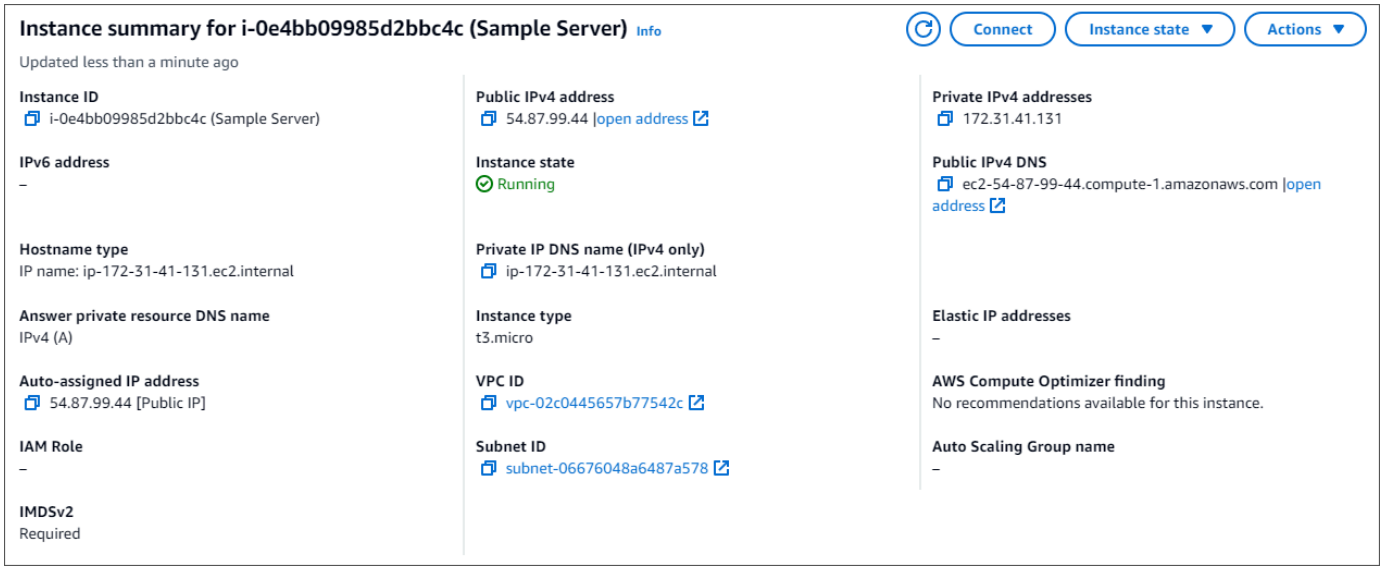
Step 3: Connect to your Amazon EC2 instance

To install the mongo shell, you must first connect to your Amazon EC2 instance. Installing the mongo shell enables you to connect to and query your Amazon DocumentDB cluster. Complete the following steps:

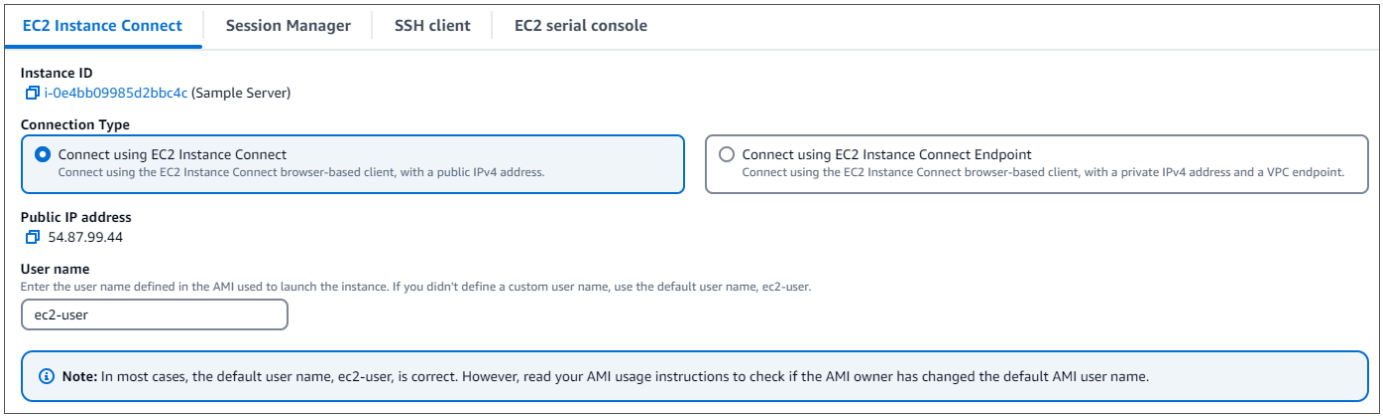
1. On the Amazon EC2 console, navigate to your instances and see if the instance you just created is running. If it is, select the instance by clicking the instance ID.



2. Choose **Connect**.



3. There are four tabbed options for your connection method: Amazon EC2 Instance Connect, Session Manager, SSH client, or EC2 serial console. You must choose one and follow its instructions. When complete, choose **Connect**.



> **Note**
>
> If your IP address changed after you started this walk-through, or you are coming back to your environment at a later time, you must update your demoEC2 security group inbound rule to enable inbound traffic from your new API address.

Step 4: Install the MongoDB Shell

You can now install the MongoDB shell, which is a command-line utility that you use to connect and query your Amazon DocumentDB cluster. There are currently two versions of MongoDB shell: the newest version, mongosh, and the previous version, mongo shell.

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

Follow the instructions below to install the MongoDB shell for your operating system.

On Amazon Linux

To install the MongoDB shell on Amazon Linux

If you are not using IAM authentication and want to use the latest MongoDB shell (mongosh) to connect to your Amazon DocumentDB cluster, follow these steps:

1. Create the repository file. At the command line of your EC2 instance you created, execute the follow command:

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. When it is complete, install mongosh with one of the two following command options at the command prompt:

   Option 1 — If you chose the default Amazon Linux 2023 during the Amazon EC2 configuration, enter this command:

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   Option 2 — If you chose Amazon Linux 2 during the Amazon EC2 configuration, enter this command:

```
sudo yum install -y mongodb-mongosh
```

If you are using IAM authentication, you must use the previous version of the MongoDB shell (5.0) to connect to your Amazon DocumentDB cluster, follow these steps:

1. Create the repository file. At the command line of your EC2 instance you created, execute the follow command:

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. When it is complete, install the mongodb 5.0 shell with the following command option at the command prompt:

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

To install mongosh on Ubuntu

1. Import the public key that will be used by the package management system.

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. Create the list file `mongodb-org-5.0.list` for MongoDB using the command appropriate for your version of Ubuntu.

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-5.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0
 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

3. Import and update the local package database using the following command:

```
sudo apt-get update
```

4. Install mongosh.

```
sudo apt-get install -y mongodb-mongosh
```

For information about installing earlier versions of MongoDB on your Ubuntu system, see Install MongoDB Community Edition on Ubuntu.

On other operating systems

To install the mongo shell on other operating systems, see Install MongoDB Community Edition in the MongoDB documentation.

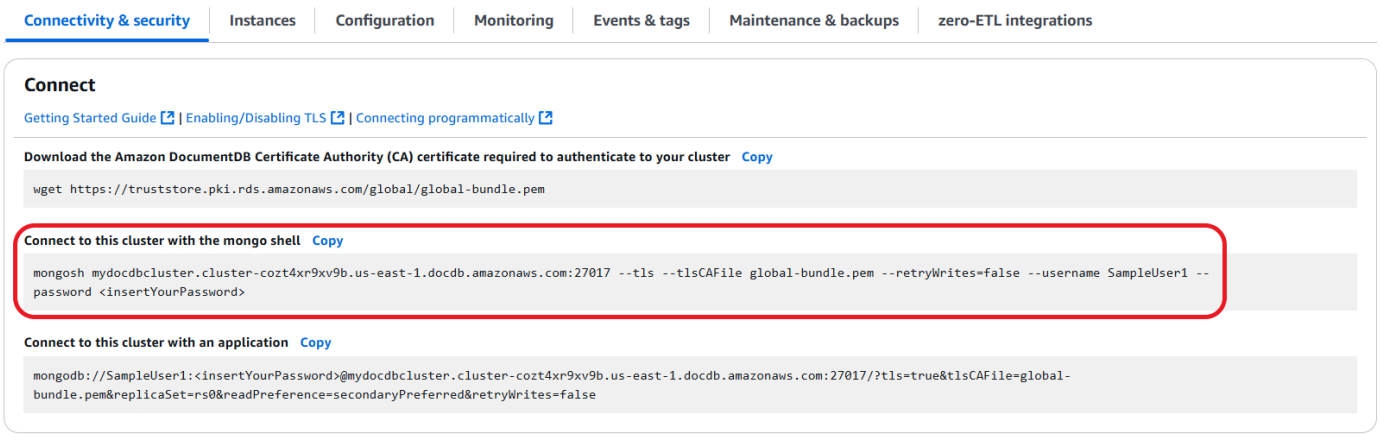
Step 5: Manage Amazon DocumentDB TLS

Download the CA certificate for Amazon DocumentDB with the following code: `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

> **Note**
>
> Transport Layer Security (TLS) is enabled by default for any new Amazon DocumentDB clusters. For more information, see Managing Amazon DocumentDB cluster TLS settings.

Step 6: Connect to your Amazon DocumentDB cluster

1. On the Amazon DocumentDB console, under **Clusters**, locate your cluster. Choose the cluster you created by clicking the **Cluster identifier** for that cluster.
2. In the **Connectivity and security** tab, locate **Connect to this cluster with the mongo shell** in the **Connect** box:



   Copy the connection string provided and paste it into your terminal.

   Make the following changes to it:

   a. Make sure you have the correct username in the string.
   b. Omit `<insertYourPassword>` so that you are prompted for the password by the mongo shell when you connect.
   c. Optional: If you are using IAM authentication, or are using the previous version of the MongoDB shell, modify your connection string as follows:

```
mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-
east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --
username SampleUser1 --password
```

Replace `mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1` with the same information from your cluster.

3. Press enter in your terminal. You are now be prompted for your password. Enter your password.
4. When you enter your password and can see the `rs0 [direct: primary] <env-name>>` prompt, you are successfully connected to your Amazon DocumentDB cluster.

Having problems connecting? See Troubleshooting Amazon DocumentDB.

Step 7: Insert and query data

Now that you are connected to your cluster, you can run a few queries to get familiar with using a document database.

1. To insert a single document, enter the following:

```
db.collection.insertOne({"hello":"DocumentDB"})
```

You get the following output:

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. You can read the document that you wrote with the `findOne()` command (because it only returns a single document). Input the following:

```
db.collection.findOne()
```

You get the following output:

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. To perform a few more queries, consider a gaming profiles use case. First, insert a few entries into a collection titled `profiles`. Input the following:

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

You get the following output:

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. Use the `find()` command to return all the documents in the profiles collection. Input the following:

```
db.profiles.find()
```

You will get an output that will match the data you typed in Step 3.

5. Use a query for a single document using a filter. Input the following:

```
db.profiles.find({name: "Katie"})
```

You get the following output:

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. Now let's try to find a profile and modify it using the `findAndModify` command. We'll give the user Matt an extra 10 points with the following code:

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

You get the following output (note that his score hasn't increased yet):

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. You can verify that his score has changed with the following query:

   ```
   db.profiles.find({name: "Matt"})
   ```

   You get the following output:

   ```
   { "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
   ```

Step 8: Explore

Congratulations! You have successfully completed the Quick Start Guide to Amazon DocumentDB.

What's next? Learn how to fully leverage this powerful database with some of its popular features:

- Managing Amazon DocumentDB
- Scaling
- Backing up and restoring

> **Note**
>
> To save on cost, you can either stop your Amazon DocumentDB cluster to reduce costs or delete the cluster. By default, after 30 minutes of inactivity, your AWS Cloud9 environment will stop the underlying Amazon EC2 instance.

Automatically connect an EC2 instance to an existing Amazon DocumentDB database

The following procedure assumes you have an existing Amazon DocumentDB cluster and an existing Amazon EC2 instance.

Access your Amazon DocumentDB cluster and set up the Amazon EC2 connection

1. Access your Amazon DocumentDB cluster.

   a. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.

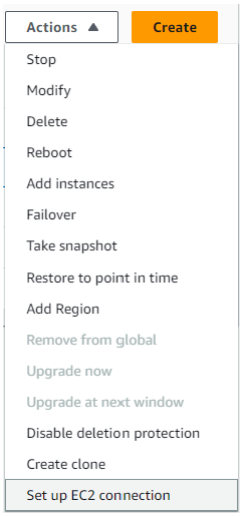
   b. In the navigation pane, choose **Clusters**.

> **Tip**
>
> If you don't see the navigation pane on the left side of your screen, choose the menu icon
> (≡)
> in the upper-left corner of the page.

c. Specify the cluster that you want by choosing the button to the left of the cluster's name.

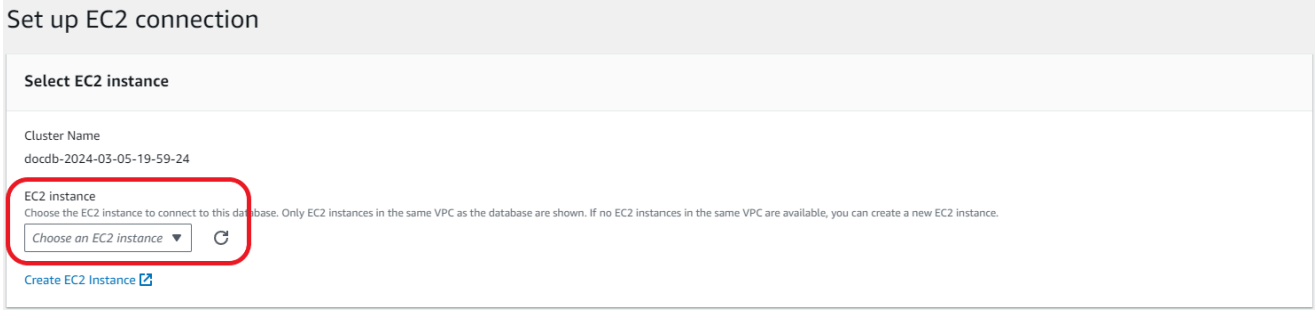
2. Set up the Amazon EC2 connection.

   a. Choose **Actions**, and then choose **Set up EC2 connection**.



   The **Set up EC2 connection** dialog appears.

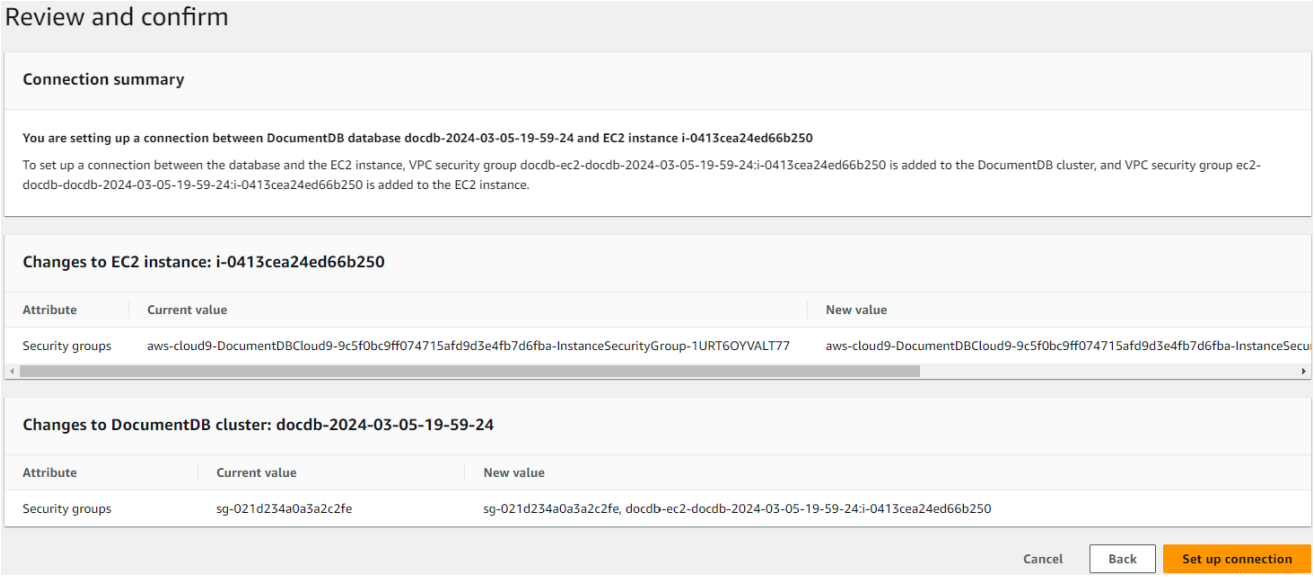
   b. In the **EC2 instance** field, choose the EC2 instance you want connected to your cluster.



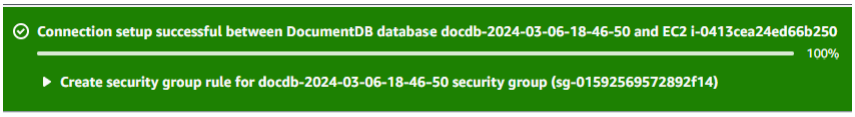
   c. Choose **Continue**.

   The **Review and confirm** dialog appears.

   d. Make sure the changes are correct. Then choose **Set up connection**.



If successful, the following verification appears:



Overview of automatic connectivity with an EC2 instance

When you set up a connection between an EC2 instance and an Amazon DocumentDB database, Amazon DocumentDB automatically configures the VPC security group for your EC2 instance and for your Amazon DocumentDB database.

The following are requirements for connecting an EC2 instance with an Amazon DocumentDB database:

- The EC2 instance must exist in the same VPC as the Amazon DocumentDB database.

  If no EC2 instances exist in the same VPC, then the console provides a link to create one.
- The user who sets up connectivity must have permissions to perform the following Amazon EC2 operations:
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`

- `ec2:DescribeSecurityGroups`
- `ec2:ModifyNetworkInterfaceAttribute`
- `ec2:RevokeSecurityGroupEgress`

If the DB instance and EC2 instance are in different Availability Zones, your account may incur cross-Availability Zone costs.

When you set up a connection to an EC2 instance, Amazon DocumentDB acts according to the current configuration of the security groups associated with the Amazon DocumentDB database and EC2 instance, as described in the following table:

Current Amazon DocumentDB security group configuration	Current EC2 security group configuration	Amazon DocumentDB action
There are one or more security groups associated with the Amazon DocumentDB database with a name that matches the pattern `DocumentDB-ec2-n` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `DocumentDB-ec2-n` (where n is a number). A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the Amazon DocumentDB database as the source.	Amazon DocumentDB takes no action. A connection was already configured automatically between the EC2 instance and the Amazon DocumentDB database. Because a connection already exists between the EC2 instance and the Amazon DocumentDB database, the security groups aren't modified.
Either of the following conditions apply: • There is no security group associated with the Amazon DocumentDB database with a name that matches the	Either of the following conditions apply: • There is no security group associated with the EC2 instance with a name that matches the pattern `ec2-DocumentDB-n` .	Amazon DocumentDB action: create new security groups

Current Amazon DocumentDB security group configuration	Current EC2 security group configuration	Amazon DocumentDB action
pattern `DocumentDB-ec2-n`. • There are one or more security groups associated with the Amazon DocumentDB with a name that matches the pattern `DocumentDB-ec2-n`. However, Amazon DocumentDB can't use any of these security groups for the connection with the EC2 instance. Amazon DocumentDB can't use a security group that doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. Amazon DocumentDB also can't use a security group that has been modified. Examples of modifications include adding a rule or changing the port of an existing rule.	• There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-DocumentDB-n` . However, Amazon DocumentDB can't use any of these security groups for the connection with the Amazon DocumentDB database. Amazon DocumentDB can't use a security group that doesn't have one outbound rule with the VPC security group of the Amazon DocumentDB database as the source. Amazon DocumentDB also can't use a security group that has been modified.	

Current Amazon DocumentDB security group configuration	Current EC2 security group configuration	Amazon DocumentDB action
There are one or more security groups associated with the Amazon DocumentDB database with a name that matches the pattern `DocumentDB-ec2-n` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-DocumentDB-n` . However, Amazon DocumentDB can't use any of these security groups for the connection with the Amazon DocumentDB database. Amazon DocumentDB can't use a security group that doesn't have one outbound rule with the VPC security group of the Amazon DocumentDB database as the source. Amazon DocumentDB also can't use a security group that has been modified.	Amazon DocumentDB action: create new security groups

Current Amazon DocumentDB security group configuration	Current EC2 security group configuration	Amazon DocumentDB action
There are one or more security groups associated with the Amazon DocumentDB database with a name that matches the pattern `DocumentDB-ec2-n` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	A valid EC2 security group for the connection exists, but it is not associated with the EC2 instance. This security group has a name that matches the pattern `DocumentDB-ec2-n`. It hasn't been modified. It has only one outbound rule with the VPC security group of the Amazon DocumentDB database as the source.	Amazon DocumentDB action: associate EC2 security group

Current Amazon DocumentDB security group configuration	Current EC2 security group configuration	Amazon DocumentDB action
Either of the following conditions apply: • There is no security group associated with the Amazon DocumentDB database with a name that matches the pattern `DocumentDB-ec2-n`. • There are one or more security groups associated with the Amazon DocumentDB database with a name that matches the pattern `DocumentDB-ec2-n`. However, Amazon DocumentDB can't use any of these security groups for the connection with the EC2 instance. Amazon DocumentDB can't use a security group that doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. Amazon DocumentDB also can't use security group that has been modified.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `DocumentDB-ec2-n`. A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the Amazon DocumentDB database as the source.	Amazon DocumentDB action: create new security groups

Amazon DocumentDB action: create new security groups

Amazon DocumentDB takes the following actions:

- Creates a new security group that matches the pattern `DocumentDB-ec2-n`. This security group has an inbound rule with the VPC security group of the EC2 instance as the source. This security group is associated with the Amazon DocumentDB database and allows the EC2 instance to access the Amazon DocumentDB database.
- Creates a new security group that matches the pattern `ec2-DocumentDB-n`. This security group has an outbound rule with the VPC security group of the Amazon DocumentDB database as the source. This security group is associated with the EC2 instance and allows the EC2 instance to send traffic to the Amazon DocumentDB database.

Amazon DocumentDB action: associate EC2 security group

Amazon DocumentDB associates the valid, existing EC2 security group with the EC2 instance. This security group allows the EC2 instance to send traffic to the Amazon DocumentDB database.

Viewing connected compute resources

You can use the AWS Management Console to view the compute resources that are connected to an Amazon DocumentDB database. The resources shown include compute resource connections that were set up automatically. You can set up connectivity with compute resources automatically in the following ways:

- You can select the compute resource when you create the database. For more information, see Creating an Amazon DocumentDB cluster and Creating a Multi-AZ DB cluster.
- You can set up connectivity between an existing database and a compute resource. For more information, see Connect Amazon EC2 automatically.

The listed compute resources don't include ones that were connected to the database manually. For example, you can allow a compute resource to access a database manually by adding a rule to the VPC security group associated with the database.

For a compute resource to be listed, the following conditions must apply:

- The name of the security group associated with the compute resource matches the pattern `ec2-DocumentDB-n` (where n is a number).
- The security group associated with the compute resource has an outbound rule with the port range set to the port that the Amazon DocumentDB database uses.

- The security group associated with the compute resource has an outbound rule with the source set to a security group associated with the Amazon DocumentDB database.
- The name of the security group associated with the Amazon DocumentDB database matches the pattern `DocumentDB-ec2-n` (where n is a number).
- The security group associated with the Amazon DocumentDB database has an inbound rule with the port range set to the port that the Amazon DocumentDB database uses.
- The security group associated with the Amazon DocumentDB database has an inbound rule with the source set to a security group associated with the compute resource.

To view compute resources connected to an Amazon DocumentDB database

1. Sign in to the AWS Management Console, and open the Amazon DocumentDB console at https://console.aws.amazon.com/docdb.
2. In the navigation pane, choose **Databases**, and then choose the name of the Amazon DocumentDB database.
3. On the **Connectivity & security** tab, view the compute resources in the **Connected compute resources** section.

Connect Amazon EC2 manually

Topics

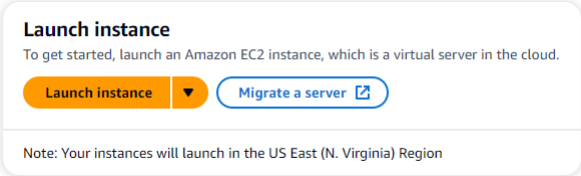
- Step 1: Create an Amazon EC2 instance
- Step 2: Create a security group
- Step 3: Create an Amazon DocumentDB cluster
- Step 4: Connect to your Amazon EC2 instance
- Step 5: Install the MongoDB Shell
- Step 6: Manage Amazon DocumentDB TLS
- Step 7: Connect to your Amazon DocumentDB cluster
- Step 8: Insert and query data
- Step 9: Explore

The following steps assume you have completed the steps in the Prerequisites topic.

Step 1: Create an Amazon EC2 instance

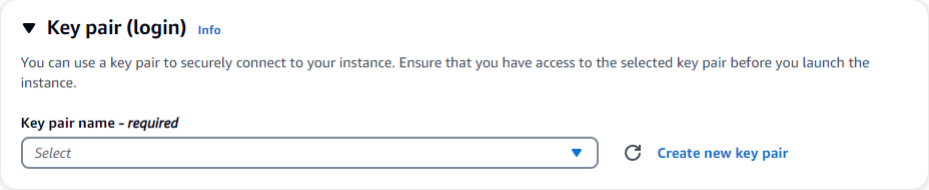
In this step, you will create an Amazon EC2 instance in the same Region and Amazon VPC that you will later use to provision your Amazon DocumentDB cluster.

1. On the Amazon EC2 console, choose **Launch instance**.



2. Enter a name or identifier in the **Name** field located in the **Name and tags** section.
3. In the **Amazon Machine Image (AMI)** drop-down list, locate **Amazon Linux 2 AMI** and choose it.

4. Locate and choose **t3.micro** in the **Instance type** drop-down list.
5. In the **Key pair (login)** section, enter the identifier of an existing key-pair, or choose **Create new key pair**.



   You must provide an Amazon EC2 key pair.

   If you do have an Amazon EC2 key pair:

   a. Select a key pair, choose your key pair from the list.
   b. You must already have the private key file (.pem or .ppk file) available to log in to your Amazon EC2 instance.

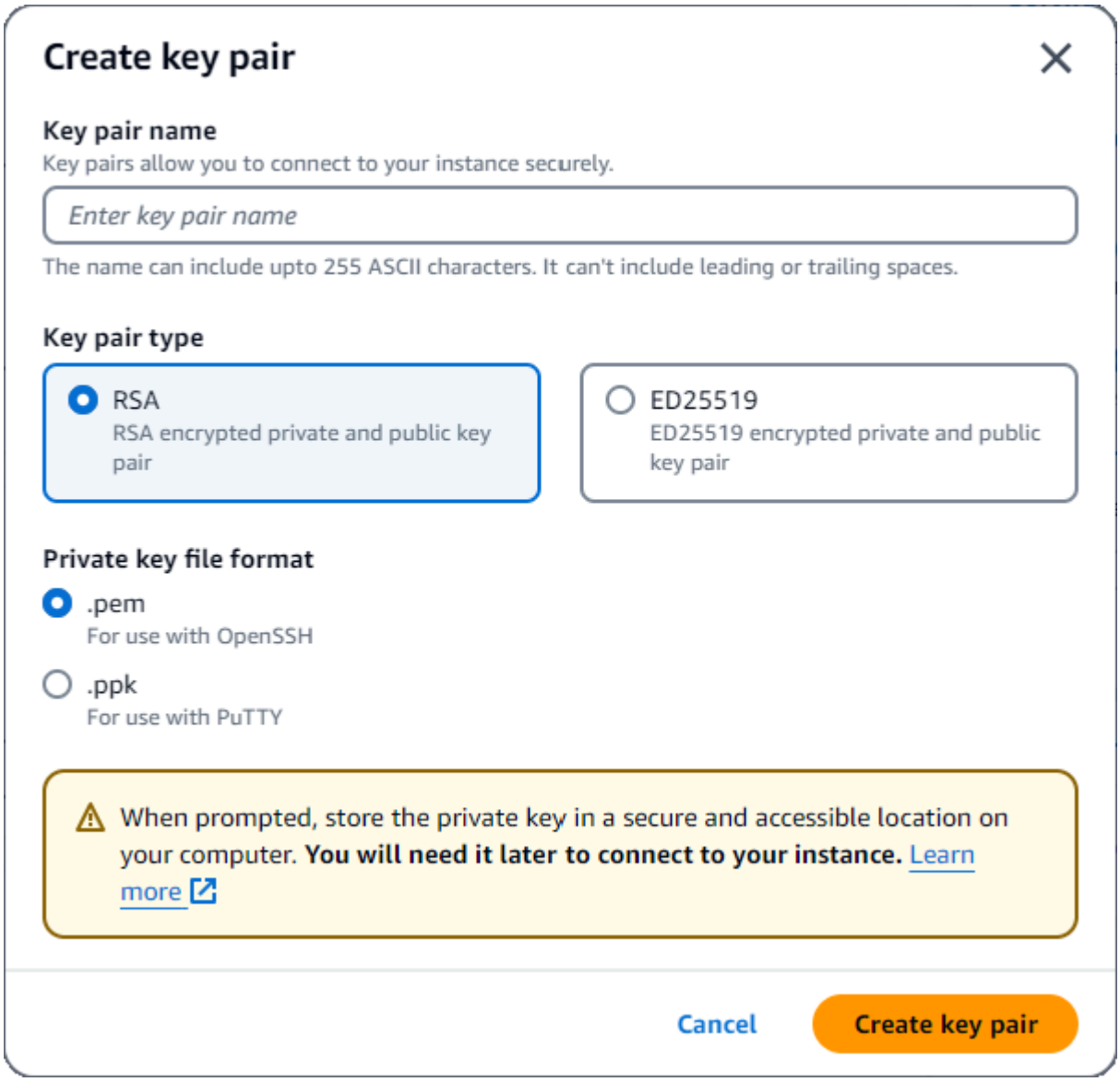
   If you do not have an Amazon EC2 key pair:

   a. Choose **Create new key pair**, the **Create key pair** dialog box appears.

b. Enter a name in the **Key pair name** field.

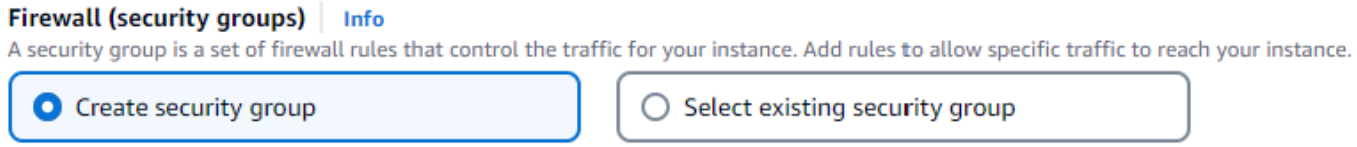
c. Choose the **Key pair type** and **Private key file format**.

d. Choose **Create key pair**.



> **Note**
>
> For security purposes, we highly recommend using a key-pair for both SSH and internet connectivity to your EC2 instance.

6. In the **Network settings section**, under **Firewall (security groups)**, choose either **Create security group** or **Select existing security group**.



   If you chose to select an existing security group, select one from the **Common security groups** drop-down list.

   If you chose to create a new security group, perform the following:

   a. Check all the traffic allow rules that apply to your EC2 connectivity.

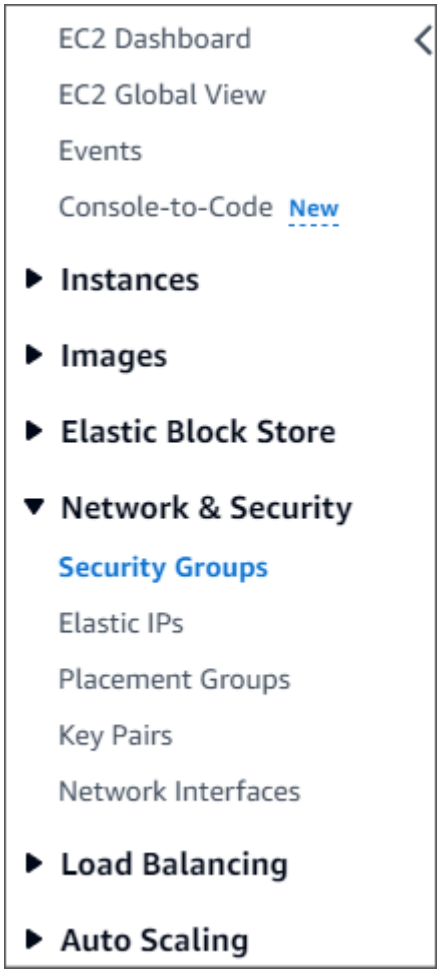
   b. In the IP field, choose **My IP** or select **Custom** to choose from a list of CIDR blocks, prefix lists, or security groups. We do not recommend **Anywhere** as a choice, unless your EC2 instance is on an isolated network, because it allows any IP address access to your EC2 instance.

7. In the **Summary** section, review your EC2 configuration and choose **Launch instance** if correct.

Step 2: Create a security group

You will now create a new security group in your default Amazon VPC. The security group `demoDocDB` enables you to connect to your Amazon DocumentDB cluster on port 27017 (the default port for Amazon DocumentDB) from your Amazon EC2 instance.

1. On the Amazon EC2 Management Console, under **Network and Security**, choose **Security groups**.



2. Choose **Create security group**.

3. In the **Basic details** section:

   a. For **Security group name**, enter `demoDocDB`.

   b. For **Description**, enter a description.

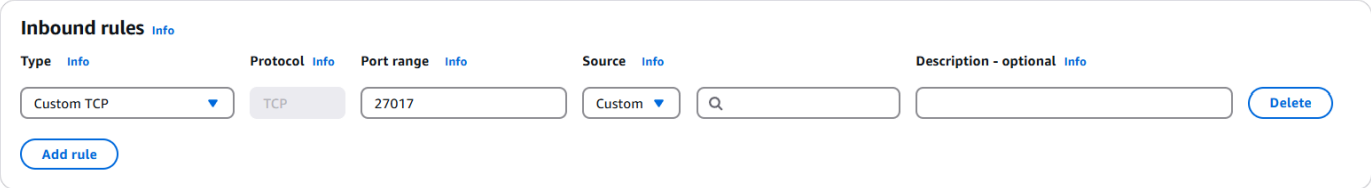
   c. For **VPC**, accept the usage of your default VPC.

4. In the **Inbound rules** section, choose **Add rule**.

   a. For **Type**, choose **Custom TCP Rule** (default).

   b. For **Port range**, enter `27017`.

   c. For **Source**, choose **Custom**. In the field next to it, search for the security group you just created in step 1. You may need to refresh your browser for the Amazon EC2 console to auto-populate the source name.



5. Accept all other defaults and choose **Create security group**.

Step 3: Create an Amazon DocumentDB cluster

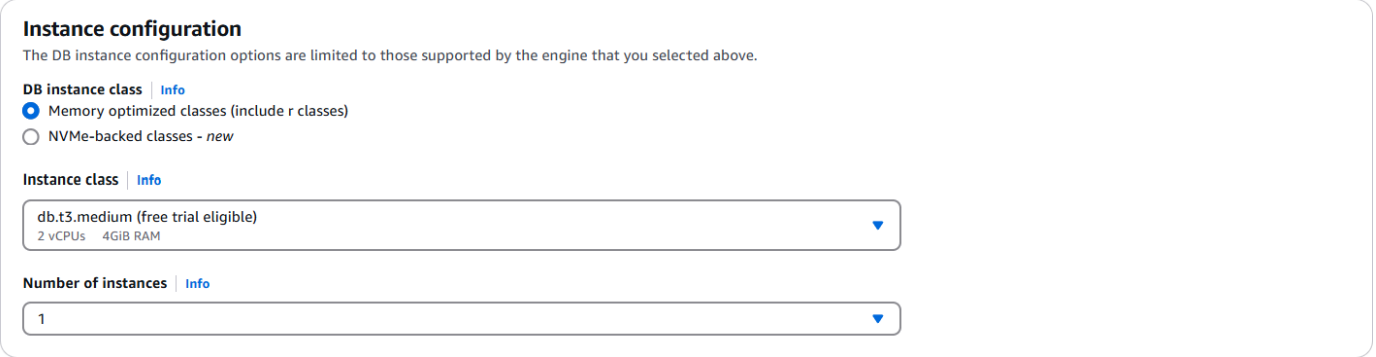
While the Amazon EC2 instance is being provisioned, you will create your Amazon DocumentDB cluster.

1. Navigate to the Amazon DocumentDB console and choose **Clusters** from the navigation pane.
2. Choose **Create**.
3. Leave the **Cluster type** setting at it's default of **Instance Based Cluster**.
4. In **Cluster configuration**, for **Cluster identifier**, enter a unique name. Note that the console will change all cluster names into lower-case regardless of how they are entered.

   Leave the **Engine version** at it's default value of **5.0.0**.
5. For **Cluster storage configuration**, leave the default setting of **Amazon DocumentDB Standard**.
6. In **Instance configuration**:

   - For **DB instance class**, choose **Memory optimized classes (include r classes)** (this is default).

     The other instance option is **NVMe-backed classes**. To learn more, see NVMe-backed instances.
   - For **Instance class**, choose the instance type that suits your needs. For a more detailed explanation of instance classes, see Instance class specifications.
   - For **number of instances**, choose a number that best reflects your needs. Remember, the lower the number, the lower the cost, and the lower the read/write volume that can be managed by the cluster.



7. For **Connectivity**, leave the default setting of **Don't connect to an EC2 compute resource**.

> **Note**
>
> Connecting to an EC2 compute resource automatically creates security groups for your connection to your cluster. Since you manually created these security groups in the previous step, you should select **Don't connect to an EC2 compute resource** so as not to create a second set of security groups.

8. In the **Authentication** section, enter a username for the primary user, and then choose **Self managed**. Enter a password, then confirm it.

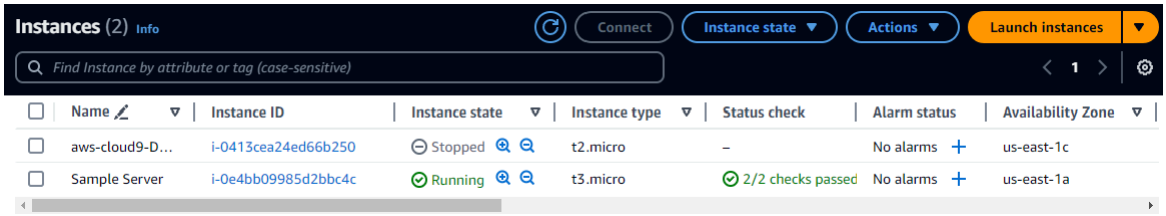
   If you instead chose **Managed in AWS Secrets Manager**, see Password management with Amazon DocumentDB and AWS Secrets Manager for more information.

9. Choose **Create cluster**.

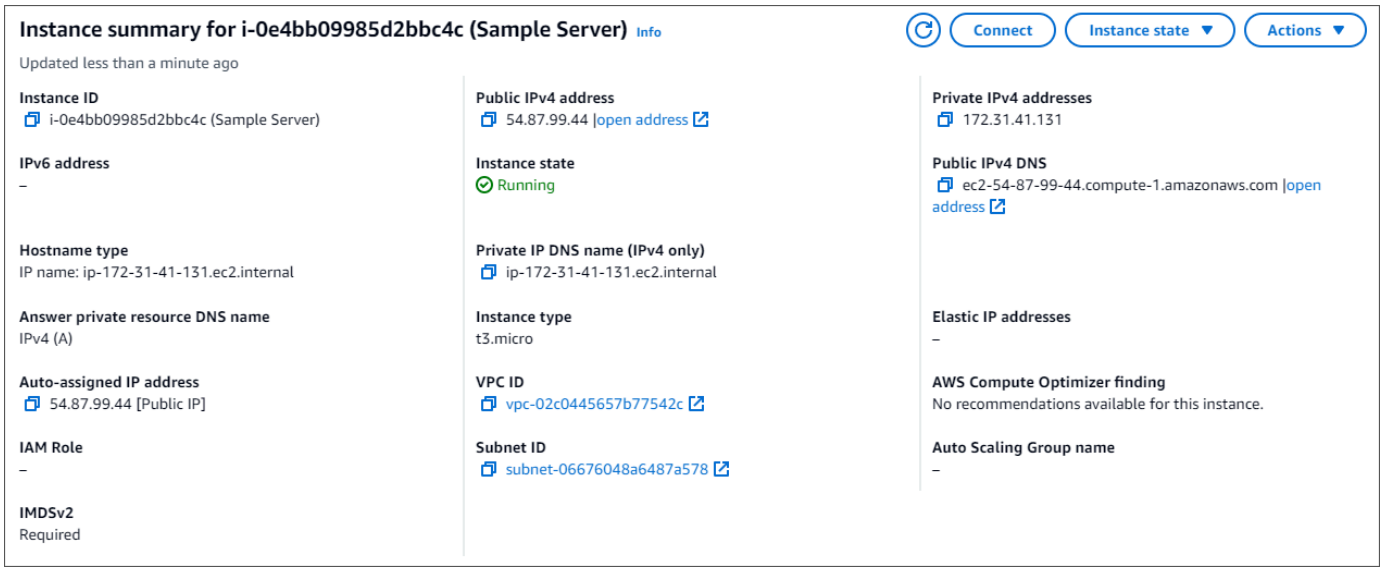
Step 4: Connect to your Amazon EC2 instance

Connecting to your Amazon EC2 instance will allow you to install the MongoDB shell. Installing the mongo shell enables you to connect to and query your Amazon DocumentDB cluster. Complete the following steps:

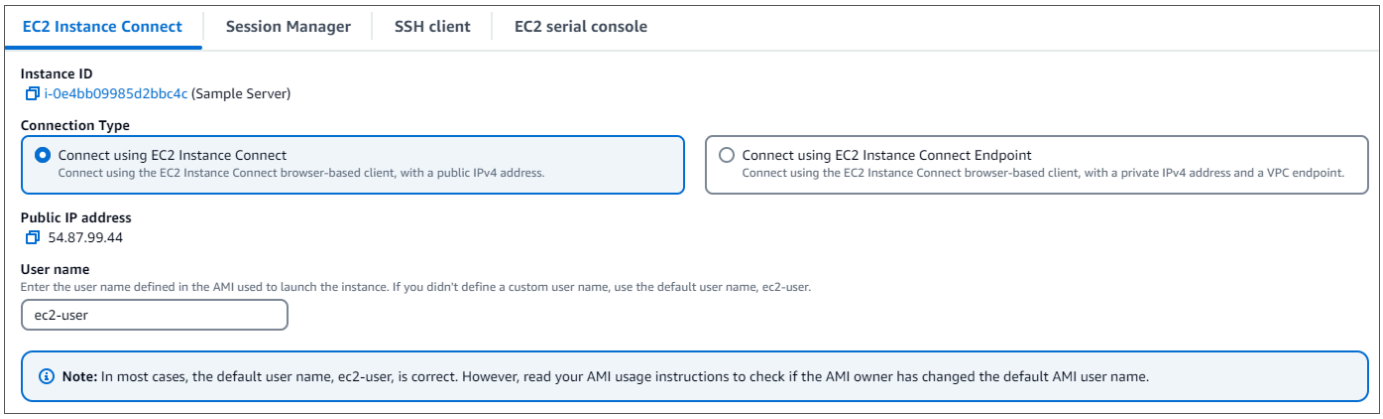
1. On the Amazon EC2 console, navigate to your instances and see if the instance you just created is running. If it is, select the instance by clicking the instance ID.



2. Choose **Connect**.



3. There are four tabbed options for your connection method: Amazon EC2 Instance Connect, Session Manager, SSH client, or EC2 serial console. You must choose one and follow its instructions. When complete, choose **Connect**.



> **Note**
>
> If your IP address changed after you started this walk-through, or you are coming back to your environment at a later time, you must update your demoEC2 security group inbound rule to enable inbound traffic from your new API address.

Step 5: Install the MongoDB Shell

You can now install the MongoDB shell, which is a command-line utility that you use to connect and query your Amazon DocumentDB cluster. There are currently two versions of MongoDB shell: the newest version, mongosh, and the previous version, mongo shell.

> **Important**
>
> There is a known limitation with Node.js drivers older than version 6.13.1, which are currently not supported by IAM identity authentication for Amazon DocumentDB. Node.js drivers and tools that use Node.js driver (for example, mongosh) must be upgraded to use Node.js driver version 6.13.1 or above.

Follow the instructions below to install the MongoDB shell for your operating system.

On Amazon Linux

To install the MongoDB shell on Amazon Linux

If you are not using IAM and want to use the latest MongoDB shell (mongosh) to connect to your Amazon DocumentDB cluster, follow these steps:

1. Create the repository file. At the command line of your EC2 instance you created, execute the follow command:

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. When it is complete, install mongosh with one of the two following command options at the command prompt:

Option 1 — If you chose the default Amazon Linux 2023 during the Amazon EC2 configuration, enter this command:

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

Option 2 — If you chose Amazon Linux 2 during the Amazon EC2 configuration, enter this command:

```
sudo yum install -y mongodb-mongosh
```

If you are using IAM, you must use the previous version of the MongoDB shell (5.0) to connect to your Amazon DocumentDB cluster, follow these steps:

1. Create the repository file. At the command line of your EC2 instance you created, execute the follow command:

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. When it is complete, install the mongodb 5.0 shell with the following command option at the command prompt:

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

To install mongosh on Ubuntu

1. Import the public key that will be used by the package management system.

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. Create the list file `mongodb-org-5.0.list` for MongoDB using the command appropriate for your version of Ubuntu.

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-5.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0
 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

3. Import and update the local package database using the following command:

```
sudo apt-get update
```

4. Install mongosh.

```
sudo apt-get install -y mongodb-mongosh
```

For information about installing earlier versions of MongoDB on your Ubuntu system, see Install MongoDB Community Edition on Ubuntu.

On other operating systems

To install the mongo shell on other operating systems, see Install MongoDB Community Edition in the MongoDB documentation.

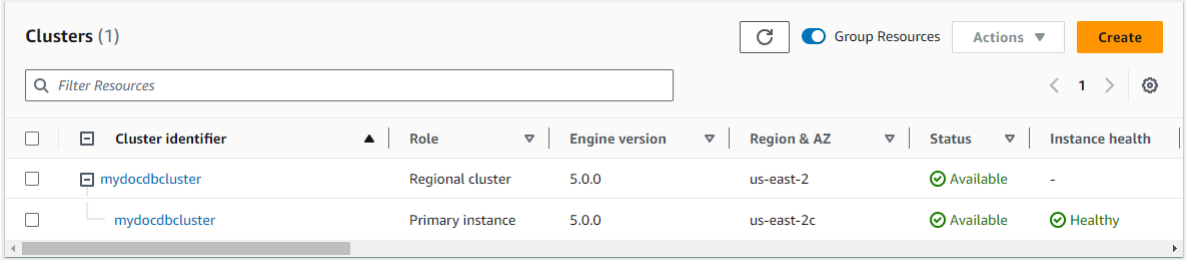
Step 6: Manage Amazon DocumentDB TLS

Download the CA certificate for Amazon DocumentDB with the following code: `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

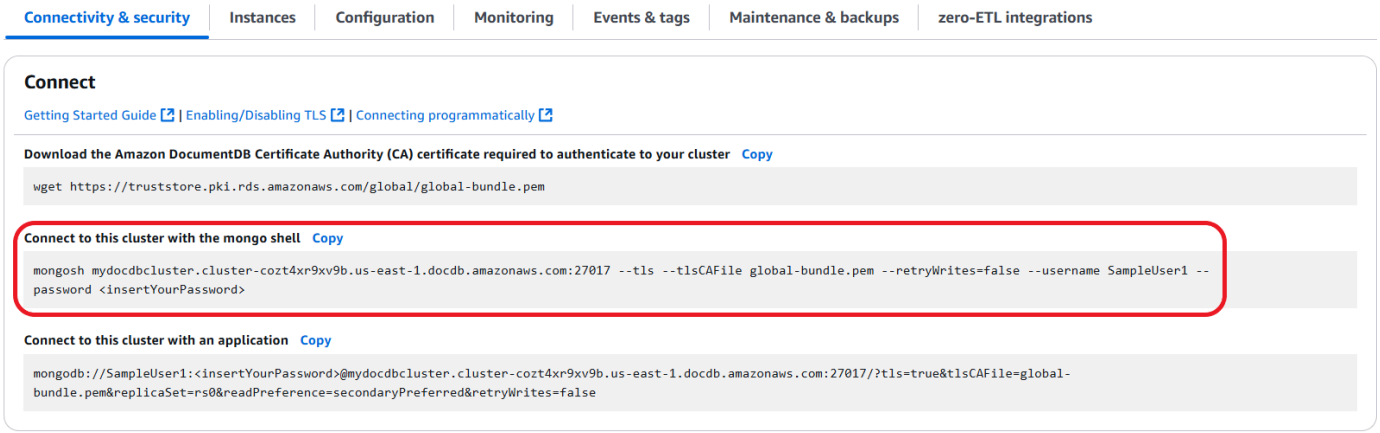
> **Note**
>
> Transport Layer Security (TLS) is enabled by default for any new Amazon DocumentDB clusters. For more information, see Managing Amazon DocumentDB cluster TLS settings.

Step 7: Connect to your Amazon DocumentDB cluster

1. On the Amazon DocumentDB console, under **Clusters**, locate your cluster. Choose the cluster you created by clicking the **Cluster identifier** for that cluster.



2. In the **Connectivity and security** tab, locate **Connect to this cluster with the mongo shell** in the **Connect** box:



Copy the connection string provided and paste it into your terminal.

Make the following changes to it:

a. Make sure you have the correct username in the string.

b. Omit `<insertYourPassword>` so that you are prompted for the password by the mongo shell when you connect.

c. Optional: If you are using IAM authentication, or are using the previous version of the MongoDB shell, modify your connection string as follows:

```
mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-
east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --
username SampleUser1 --password
```

Replace `mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1` with the same information from your cluster.

3. Press enter in your terminal. You are now be prompted for your password. Enter your password.

4. When you enter your password and can see the `rs0 [direct: primary] <env-name>>` prompt, you are successfully connected to your Amazon DocumentDB cluster.

Having problems connecting? See Troubleshooting Amazon DocumentDB.

Step 8: Insert and query data

Now that you are connected to your cluster, you can run a few queries to get familiar with using a document database.

1. To insert a single document, enter the following:

```
db.collection.insertOne({"hello":"DocumentDB"})
```

   You get the following output:

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. You can read the document that you wrote with the `findOne()` command (because it only returns a single document). Input the following:

```
db.collection.findOne()
```

   You get the following output:

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. To perform a few more queries, consider a gaming profiles use case. First, insert a few entries into a collection titled `profiles`. Input the following:

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

You get the following output:

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. Use the `find()` command to return all the documents in the profiles collection. Input the following:

```
db.profiles.find()
```

You will get an output that will match the data you typed in Step 3.

5. Use a query for a single document using a filter. Input the following:

```
db.profiles.find({name: "Katie"})
```

You get the following output:

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. Now let's try to find a profile and modify it using the `findAndModify` command. We'll give the user Matt an extra 10 points with the following code:

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

You get the following output (note that his score hasn't increased yet):

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. You can verify that his score has changed with the following query:

```
db.profiles.find({name: "Matt"})
```

You get the following output:

```
{ "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
```

Step 9: Explore

Congratulations! You have successfully completed the Quick Start Guide to Amazon DocumentDB.

What's next? Learn how to fully leverage this powerful database with some of its popular features:

- Managing Amazon DocumentDB
- Scaling
- Backing up and restoring

> **Note**
>
> To save on cost, you can either stop your Amazon DocumentDB cluster to reduce costs or delete the cluster. By default, after 30 minutes of inactivity, your AWS Cloud9 environment will stop the underlying Amazon EC2 instance.

Connect using the Amazon DocumentDB JDBC driver

The JDBC driver for Amazon DocumentDB provides an SQL-relational interface for developers and enables connectivity from BI tools such as Tableau and DbVisualizer.

For more detailed information, refer to the Amazon DocumentDB JDBC Driver documentation on GitHub.

Topics

- Getting started
- Connect to Amazon DocumentDB from Tableau Desktop
- Connect to Amazon DocumentDB from DbVisualizer
- JDBC automatic schema generation
- SQL support and limitations
- Troubleshooting

Getting started

Step 1. Create Amazon DocumentDB Cluster

If you do not have an Amazon DocumentDB cluster created, then create one using the instructions in the Getting Started section of the Amazon DocumentDB Developer Guide.

> **Note**
>
> Amazon DocumentDB is a Virtual Private Cloud (VPC) only service. If you are connecting from a local machine, outside the cluster's VPC, you will need to create an SSH connection to an Amazon EC2 instance. In this case, launch your cluster using the instructions in Connect with EC2. See Using an SSH Tunnel to Connect to Amazon DocumentDB for more information on SSH tunneling and when you might need it.

Step 2. JRE or JDK Installation

Depending on your BI application, you may need to ensure a 64-bit JRE or JDK installation version 8 or later is installed on your computer. You can download the Java SE Runtime Environment 8 here.

Step 3. Download the DocumentDB JDBC Driver

Download the DocumentDB JDBC driver from here. The driver is packaged as a single JAR file (e.g. documentdb-jdbc-1.0.0-all.jar).

Step 4. Using an SSH Tunnel to Connect to Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) clusters are deployed within an Amazon Virtual Private Cloud (Amazon VPC). They can be accessed directly by Amazon EC2 instances or other AWS services that are deployed in the same Amazon VPC. Additionally, Amazon DocumentDB can be accessed by EC2a instances or other AWS services in different VPCs in the same AWS Region or other Regions via VPC peering.

You can use SSH tunneling (also known as port forwarding) to access your Amazon DocumentDB resources, from outside the cluster's VPC. This will be the case for most users not running their application on a VM in the same VPC as the DocumentDB cluster.

To create an SSH tunnel, you need an Amazon EC2 instance running in the same Amazon VPC as your Amazon DocumentDB cluster. You can either use an existing EC2 instance in the same VPC

as your cluster or create one. You can set up an SSH tunnel to the Amazon DocumentDB cluster `sample-cluster.node.us-east-1.docdb.amazonaws.com` by running the following command on your local computer.

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -
N
```

The -L flag is used for forwarding a local port. This is a prerequisite for connecting to any BI tool running on a client outside your VPC. Once you run the step above you can move on to the next steps for the BI tool of your choice.

For further information on SSH tunneling , please refer to the documentation on Using an SSH tunnel to connect to Amazon DocumentDB.

Connect to Amazon DocumentDB from Tableau Desktop

Topics

- Adding the Amazon DocumentDB JDBC Driver
- Connecting to Amazon DocumentDB using Tableau - SSH Tunnel

Adding the Amazon DocumentDB JDBC Driver

To connect to Amazon DocumentDB from Tableau Desktop you must download and install the Amazon DocumentDB JDBC driver and the DocumentDB Tableau connector.

1. Download the Amazon DocumentDB JDBC driver JAR file from the Amazon DocumentDB JDBC Driver repository and copy it to one of these directories according to your operating system:
   - *Windows* - `C:\Program Files\Tableau\Drivers`
   - *MacOS* - `~/Library/Tableau/Drivers`
2. Download the DocumentDB Tableau connector (a TACO file) from the Tableau Exchange website and copy it to your *My Tableau Repository/Connectors directory*.
   - *Windows* - `C:\Users\[user]\Documents\My Tableau Repository\Connectors`
   - *MacOS* - `/Users/[user]/Documents/My Tableau Repository/Connectors`

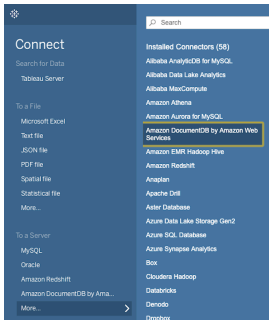
For additional information, refer to the Tableau documentation.

> **Note**
>
> If you are using newer CA certificates, make sure to upgrade your JDBC driver to v1.4.5 (available in this AWS GitHub repository.).

Connecting to Amazon DocumentDB using Tableau - SSH Tunnel

To connect to Tableau from a client machine outside of the VPC of your DocumentDB cluster, you must setup an SSH tunnel before following the steps below:

1. Launch the Tableau Desktop application.
2. Navigate to **Connect** > **To A Server** > **More**.
3. Choose **Amazon DocumentDB by Amazon Web Services** under **Installed Connectors**.

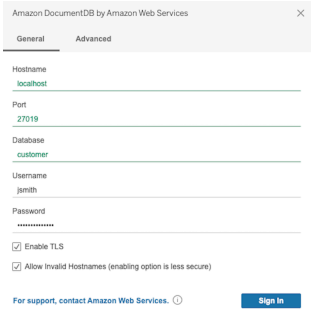


Connecting to Amazon DocumentDB Using Tableau - External SSH Tunnel

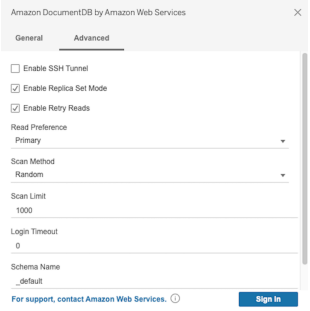
1. Enter the required connection parameters **Hostname**, **Port**, **Database**, **Username** and **Password**. The connection parameters in the example below are equivalent to the JDBC connection string :

   ```
   jdbc:documentdb://localhost:27019/test?
   tls=true&tlsAllowInvalidHostnames=true&scanMethod=random&scanLimit=1000&logir
   ```

   with the username and password parameters passed separately in a properties collection. For more information on connection string parameters, refer to the Amazon DocumentDB JDBC Driver github documentation.



2. (Optional) More advanced options can be found on the **Advanced** tab.

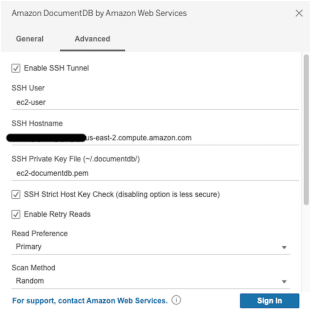


3. Choose **Sign in.**

Connecting to Amazon DocumentDB Using Tableau - Internal SSH Tunnel

> **Note**
>
> If you prefer to not setup the SSH tunnel using a terminal, you can use the Tableau GUI to specify your EC2 instance details which the JDBC driver will inherently use to create a SSH tunnel.

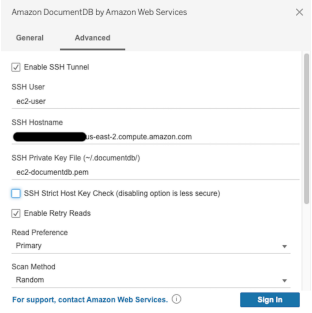
1. On the **Advanced** tab, choose the **Enable SSH Tunnel option** to review further properties.



2. Enter the **SSH User**, **SSH Hostname**, and **SSH Private Key File**.

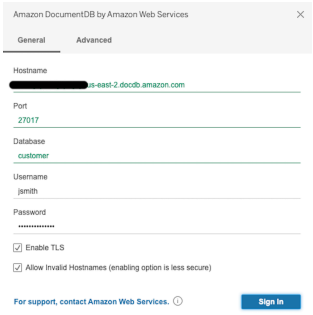
3. (Optional) You can disable the **SSH Strict Host Key Check** option which bypasses the host key check against a known hosts file.

> **Note**
>
> Disabling this option is less secure as it can lead to a man-in-the-middle attack.



4. Enter the required parameters; **Hostname**, **Port**, **Database**, **Username** and **Password**.

> **Note**
>
> Make sure you use the DocumentDB cluster endpoint and not localhost when using the internal SSH tunnel option.



5. Choose **Sign In**.

Connect to Amazon DocumentDB from DbVisualizer

Topics

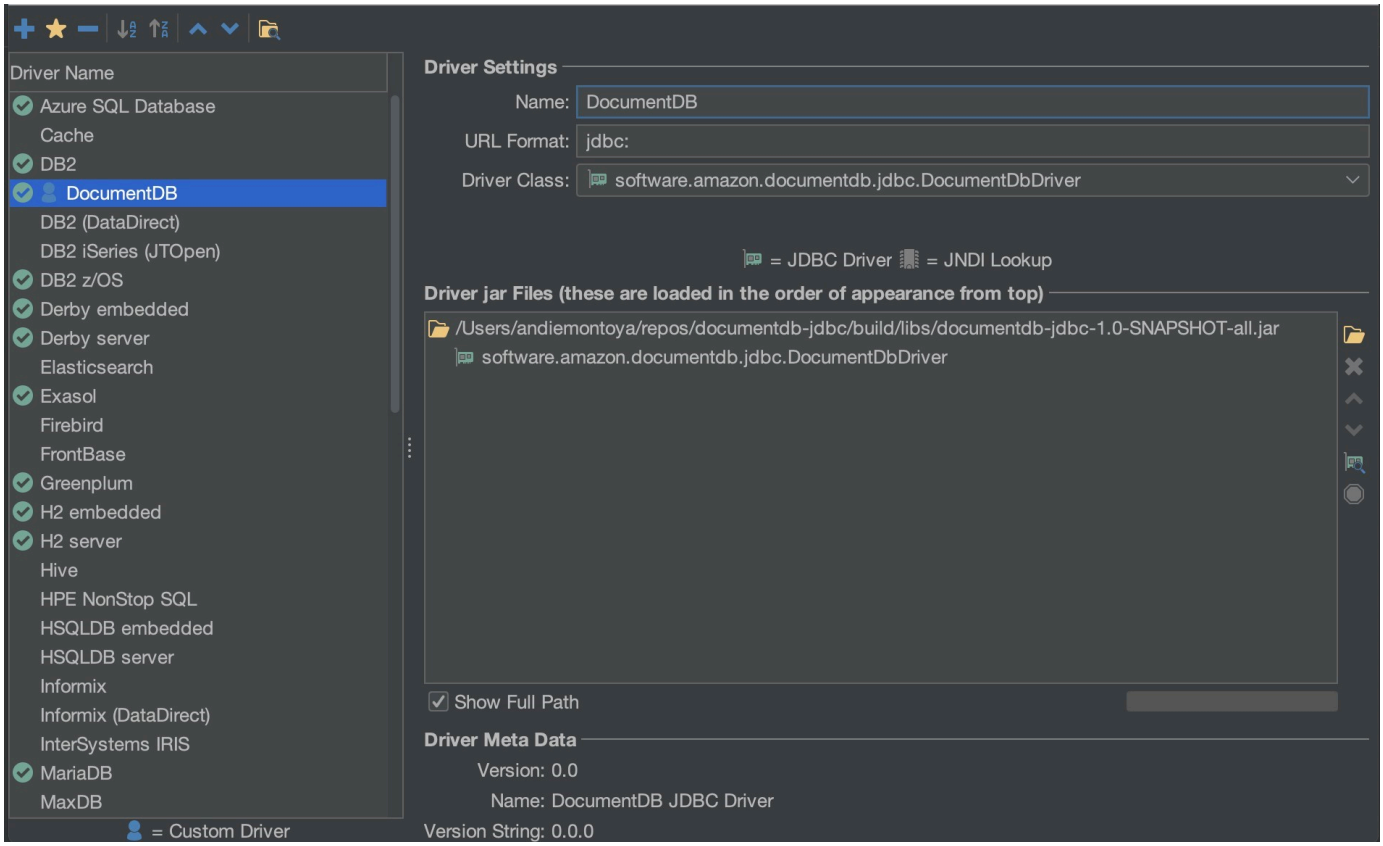
- Adding the Amazon DocumentDB JDBC driver

- Connecting to Amazon DocumentDB using DbVisualizer

Adding the Amazon DocumentDB JDBC driver

To connect to Amazon DocumentDB from DbVisualizer you must first import the Amazon DocumentDB JDBC Driver

1. Start the DbVisualizer application and navigate to the menu path: **Tools > Driver Manager...**
2. Choose **+** (or in the menu, select **Driver > Create Driver**).
3. Set **Name** to `DocumentDB`.
4. Set **URL Format** to `jdbc:documentdb://<host>[:port]/<database>[?option=value[&option=value[...]]]`
5. Choose the **folder** button and then select the Amazon DocumentDB JDBC driver JAR file and choose the **Open** button.
6. Verify that the **Driver Class** field is set to `software.amazon.documentdb.jdbc.DocumentDbDriver`. Your Driver Manager settings for **DocumentDB** should look like the following example.



7. Close the dialog. The Amazon DocumentDB JDBC driver will be setup and ready to use.

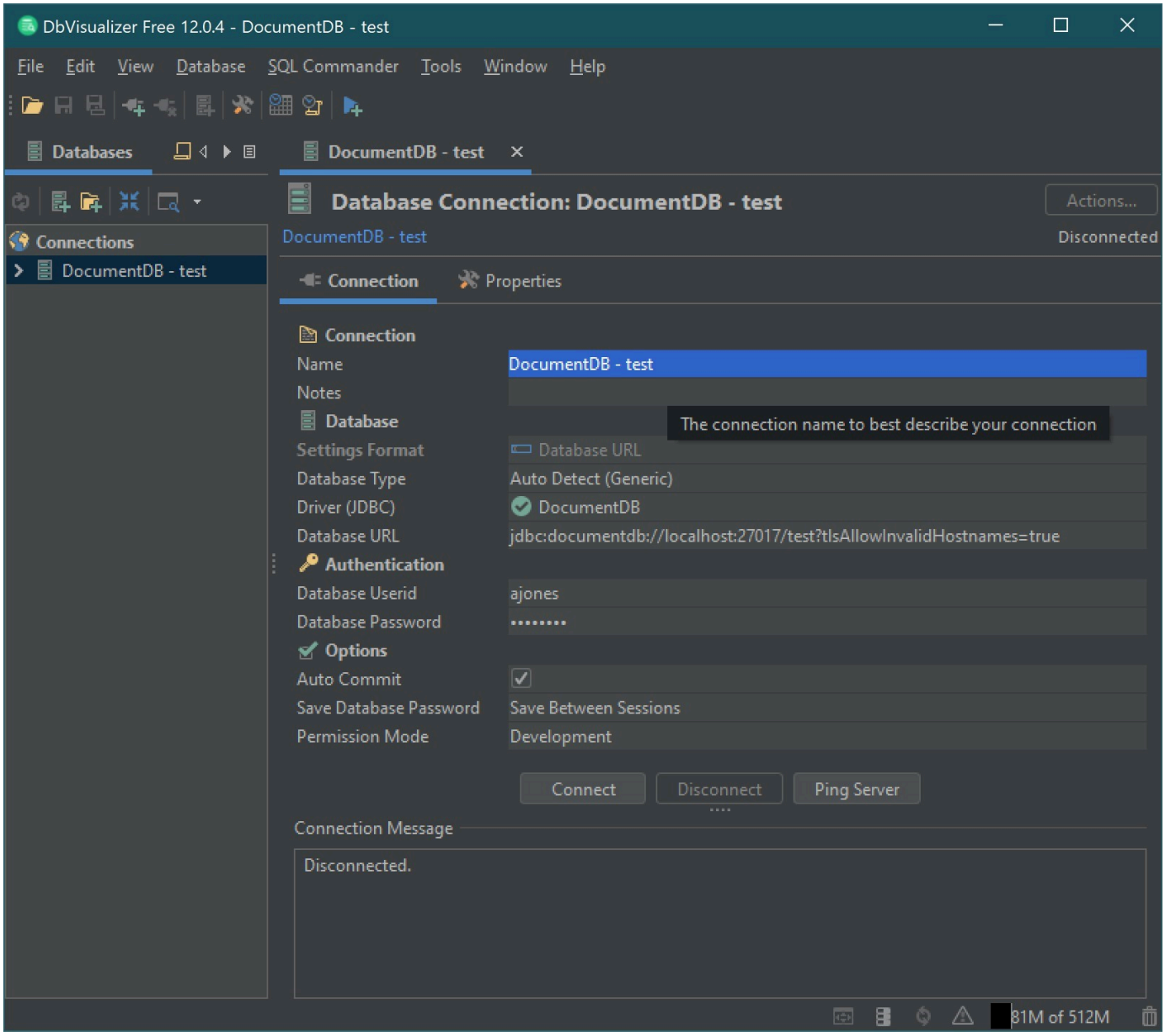
Connecting to Amazon DocumentDB using DbVisualizer

Connect to Amazon DocumentDB Using DbVisualizer

1. If you are connecting from outside the Amazon DocumentDB cluster's VPC, ensure you have setup an SSH tunnel.
2. Choose **Database > Create Database Connection** from the top level menu.
3. Enter a descriptive name for the **Name** field.
4. Set **Driver (JDBC)** to the DocumentDB driver you created in the previous section.
5. Set **Database URL** to your JDBC connection string.

   For example: `jdbc:documentdb://localhost:27017/database?tlsAllowInvalidHostnames=true`
6. Set **Database Userid** to your Amazon DocumentDB user ID.
7. Set **Database Password** to the corresponding password for the user ID.

   Your Database Connection dialog should look like the following dialog:



8. Choose **Connect**.

JDBC automatic schema generation

Amazon DocumentDB is a document database and therefore does not have the concept of tables and schema. However, BI tools such as Tableau will expect the database it connects to present a schema. Specifically, when the JDBC driver connection needs to get the schema for the collection in the database, it will poll for all the collections in the database. The driver will determine if a cached version of the schema for that collection already exists. If a cached version does not exist, it will sample the collection for documents and create a schema based on the following behavior.

Topics

- Schema generation limitations
- Scanning method options
- Amazon DocumentDB data types
- Mapping scalar document fields
- Object and array data type handling

Schema generation limitations

The DocumentDB JDBC driver imposes a limit on the length of identifiers at 128 characters. The schema generator may truncate the length of generated identifiers (table names and column names) to ensure they fit that limit.

Scanning method options

The sampling behavior can be modified using connection string or data source options.

- *scanMethod*=<option>
  - *random - (default) - The sample documents are returned in random order.*
  - *idForward* - The sample documents are returned in order of id.
  - *idReverse* - The sample documents are returned in reverse order of id.
  - *all* - Sample all the documents in the collection.
- *scanLimit*=<n> - The number of documents to sample. The value must be a positive integer. The default value is *1000*. If *scanMethod* is set to *all*, this option is ignored.

Amazon DocumentDB data types

The Amazon DocumentDB server supports a number of MongoDB data types. Listed below are the supported data types, and their associated JDBC data types.

MongoDB Data Type	Supported in DocumentDB	JDBC Data Type
Binary Data	Yes	VARBINARY
Boolean	Yes	BOOLEAN

MongoDB Data Type	Supported in DocumentDB	JDBC Data Type
Double	Yes	DOUBLE
32-bit Integer	Yes	INTEGER
64-bit Integer	Yes	BIGINT
String	Yes	VARCHAR
ObjectId	Yes	VARCHAR
Date	Yes	TIMESTAMP
Null	Yes	VARCHAR
Regular Expression	Yes	VARCHAR
Timestamp	Yes	VARCHAR
MinKey	Yes	VARCHAR
MaxKey	Yes	VARCHAR
Object	Yes	virtual table
Array	Yes	virtual table
Decimal128	No	DECIMAL
JavaScript	No	VARCHAR
JavaScript (with scope)	No	VARCHAR
Undefined	No	VARCHAR
Symbol	No	VARCHAR
DBPointer (4.0+)	No	VARCHAR

Mapping scalar document fields

When scanning a sample of documents from a collection, the JDBC driver will create one or more schema to represent the samples in the collection. In general, a scalar field in the document maps to a column in the table schema. For example, in a collection named team, and a single document `{ "_id" : "112233", "name" : "Alastair", "age": 25 }`, this would map to schema:

Table Name	Column Name	Data Type	Key
team	*team id*	VARCHAR	PK
team	name	VARCHAR	
team	age	INTEGER	

Data type conflict promotion

When scanning the sampled documents, it is possible that the data types for a field are not consistent from document to document. In this case, the JDBC driver will promote the JDBC data type to a common data type that will suit all data types from the sampled documents.

For Example:

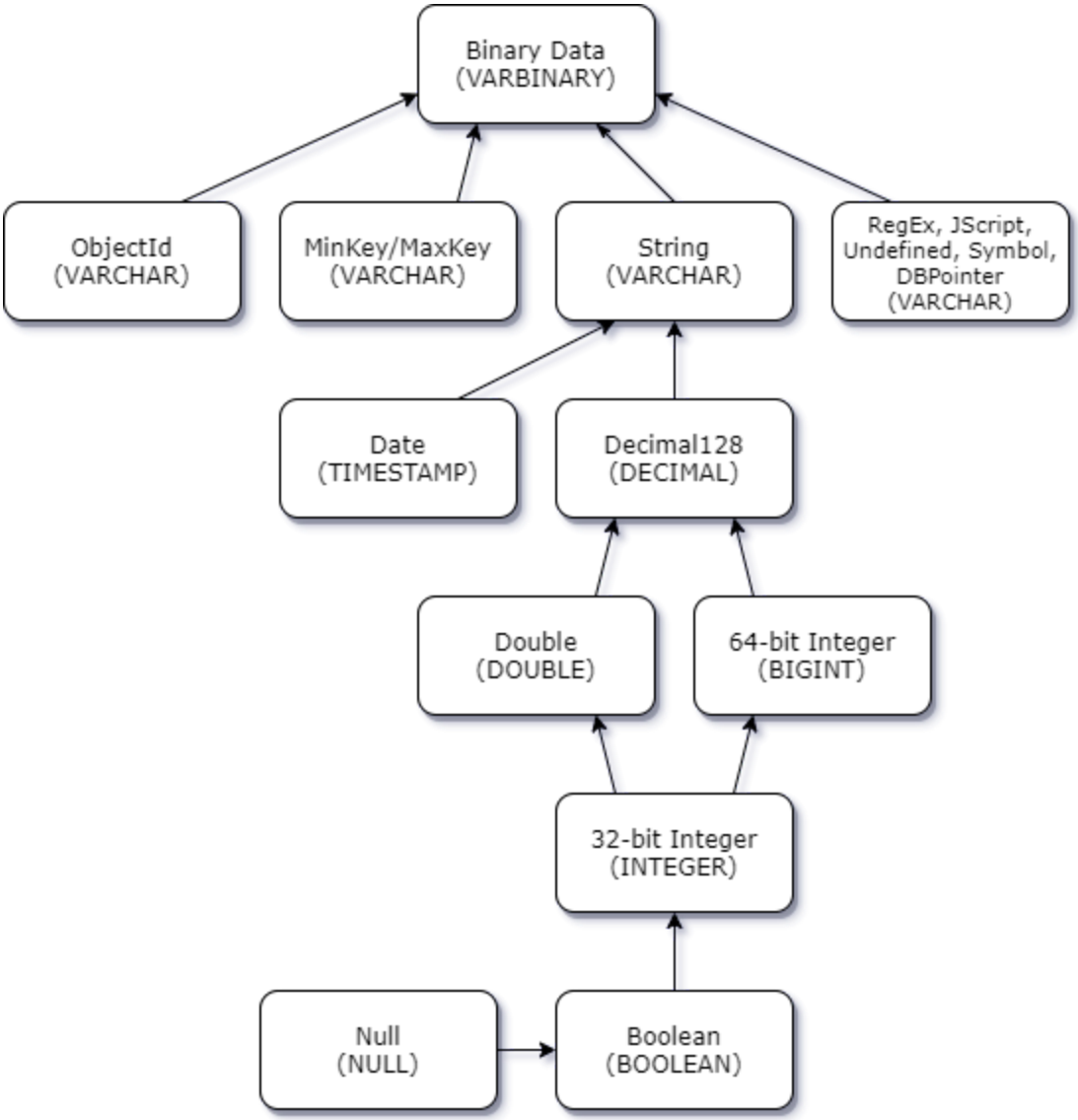
```
{
"_id" : "112233",
"name" : "Alastair", "age" : 25
}

{
"_id" : "112244",
"name" : "Benjamin",
"age" : "32"
}
```

The *age* field is of type 32-bit integer in the first document but string in the second document. Here the JDBC driver will promote the JDBC data type to VARCHAR to handle either data type when encountered.

Table Name	Column Name	Data Type	Key
team	*team id*	VARCHAR	PK
team	name	VARCHAR	
team	age	VARCHAR	

Scalar-scalar conflict promotion

The following diagram shows the way in which scalar-scalar data type conflicts are resolved.



Scalar-complex type conflict promotion

Like the scalar-scalar type conflicts, the same field in different documents can have conflicting data types between complex (array and object) and scalar (integer, boolean, etc.). All of these conflicts are resolved (promoted) to VARCHAR for those fields. In this case, array and object data is returned as the JSON representation.

Embedded Array - String Field Conflict Example:

```
{
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
{
   "_id":"112244",
   "name":"Joan Starr",
   "subscriptions":1
}
```

The above example maps to schema for the customer2 table:

Table Name	Column Name	Data Type	Key
customer2	*customer2 id*	VARCHAR	PK
customer2	name	VARCHAR	
customer2	subscription	VARCHAR	

and the customer1_subscriptions virtual table:

Table Name	Column Name	Data Type	Key
customer1_subscriptions	*customer1 id*	VARCHAR	PK/FK

Table Name	Column Name	Data Type	Key
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	value	VARCHAR	
customer_address	city	VARCHAR	
customer_address	region	VARCHAR	
customer_address	country	VARCHAR	
customer_address	code	VARCHAR	

Object and array data type handling

So far, we've only described how scalar data types are mapped. Object and Array data types are (currently) mapped to virtual tables. The JDBC driver will create a virtual table to represent either object or array fields in a document. The name of the mapped virtual table will concatenate the original collection's name followed by the field's name separated by an underscore character ("_").

The base table's primary key ("_id") takes on a new name in the new virtual table and is provided as a foreign key to the associated base table.

For embedded array type fields, index columns are generated to represent the index into the array at each level of the array.

Embedded object field example

For object fields in a document, a mapping to a virtual table is created by the JDBC driver.

```
{
   "Collection: customer",
   "_id":"112233",
   "name":"George Jackson",
   "address":{
      "address1":"123 Avenue Way",
      "address2":"Apt. 5",
```

```
        "city":"Hollywood",
        "region":"California",
        "country":"USA",
        "code":"90210"
    }
}
```

The above example maps to schema for customer table:

Table Name	Column Name	Data Type	Key
customer	*customer id*	VARCHAR	PK
customer	name	VARCHAR	

and the customer_address virtual table:

Table Name	Column Name	Data Type	Key
customer_address	*customer id*	VARCHAR	PK/FK
customer_address	address1	VARCHAR	
customer_address	address2	VARCHAR	
customer_address	city	VARCHAR	
customer_address	region	VARCHAR	
customer_address	country	VARCHAR	
customer_address	code	VARCHAR	

Embedded array field example

For array fields in a document, a mapping to a virtual table is also created by the JDBC driver.

```
{
   "Collection: customer1",
```

```
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
```

The above example maps to schema for customer1 table:

Table Name	Column Name	Data Type	Key
customer1	*customer1 id*	VARCHAR	PK
customer1	name	VARCHAR	

and the customer1_subscriptions virtual table:

Table Name	Column Name	Data Type	Key
customer1_subscriptions	*customer1 id*	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	value	VARCHAR	
customer_address	city	VARCHAR	
customer_address	region	VARCHAR	
customer_address	country	VARCHAR	
customer_address	code	VARCHAR	

SQL support and limitations

The Amazon DocumentDB JDBC driver is a read-only driver that supports a subset of SQL-92 and some common extensions. Refer to the SQL limitations documentation and JDBC limitations documentation for more information.

Troubleshooting

If you are having problems using the Amazon DocumentDB JDBC driver, refer to the Troubleshooting Guide.

Connect using the Amazon DocumentDB ODBC driver

The ODBC driver for Amazon DocumentDB provides an SQL-relational interface for developers and enables connectivity from BI tools such as Power BI Desktop and Microsoft Excel.

For more detailed information, refer to the Amazon DocumentDB ODBC Driver documentation on GitHub.

Topics

- Getting started
- Setting up the Amazon DocumentDB ODBC driver in Windows
- Connect to Amazon DocumentDB from Microsoft Excel
- Connect to Amazon DocumentDB from Microsoft Power BI Desktop
- Automatic schema generation
- SQL support and limitations
- Troubleshooting

Getting started

Step 1. Create Amazon DocumentDB Clusters

If you don't already have an Amazon DocumentDB cluster, there are a number of ways to get started.

> **Note**
>
> Amazon DocumentDB is a Virtual Private Cloud (VPC)-only service. If you are connecting from a local machine outside the cluster's VPC, you will need to create an SSH connection to an Amazon EC2 instance. In this case, launch your cluster using the instructions in Connect with EC2. See Using an SSH Tunnel to Connect to Amazon DocumentDB for more information on SSH tunneling and when you might need it.

Step 2. JRE or JDK Installation

Depending on your BI application, you may need to ensure a 64-bit JRE or JDK installation version 8 or later installed on your computer. You can download the Java SE Runtime Environment 8 here.

Step 3. Download the Amazon DocumentDB ODBC Driver

Download the Amazon DocumentDB ODBC driver here. Choose the proper installer (for example, documentdb-odbc-1.0.0.msi). Follow the installation guide.

Step 4. Using an SSH Tunnel to Connect to Amazon DocumentDB

Amazon DocumentDB clusters are deployed within an Amazon Virtual Private Cloud (Amazon VPC). They can be accessed directly by Amazon EC2 instances or other AWS services that are deployed in the same Amazon VPC. Additionally, Amazon DocumentDB can be accessed by Amazon EC2 instances or other AWS services in different VPCs in the same AWS region or other regions via VPC peering.

However, suppose that your use case requires that you (or your application) access your Amazon DocumentDB resources from outside the cluster's VPC. This will be the case for most users not running their application on a VM in the same VPC as the Amazon DocumentDB cluster. When connecting from outside the VPC, you can use SSH tunneling (also known as port forwarding) to access your Amazon DocumentDB resources.

To create an SSH tunnel, you need an Amazon EC2 instance running in the same Amazon VPC as your Amazon DocumentDB cluster. You can either use an existing EC2 instance in the same VPC as your cluster or create one. You can set up an SSH tunnel to the Amazon DocumentDB cluster `sample-cluster.node.us-east-1.docdb.amazonaws.com` by running the following command on your local computer:

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -
N
```

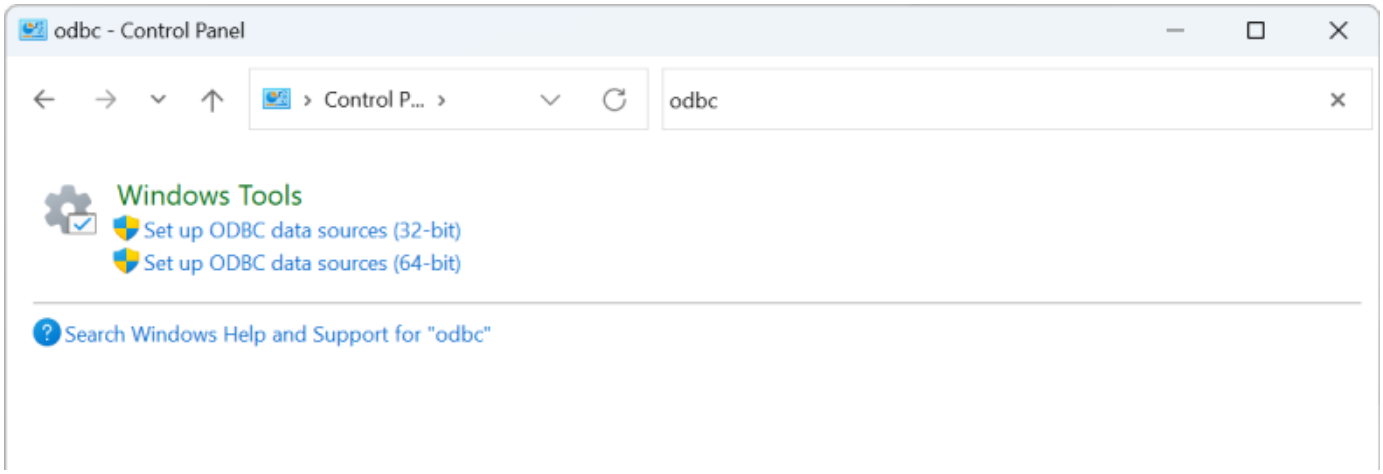
The `-L` flag is used for forwarding a local port. This is a prerequisite for connecting to any BI tool running on a client outside your VPC. Once you run the step above you can move on to the next steps for the BI tool of your choice.

For further information on SSH tunneling, please refer to the documentation on Using an SSH Tunnel to Connect to Amazon DocumentDB.

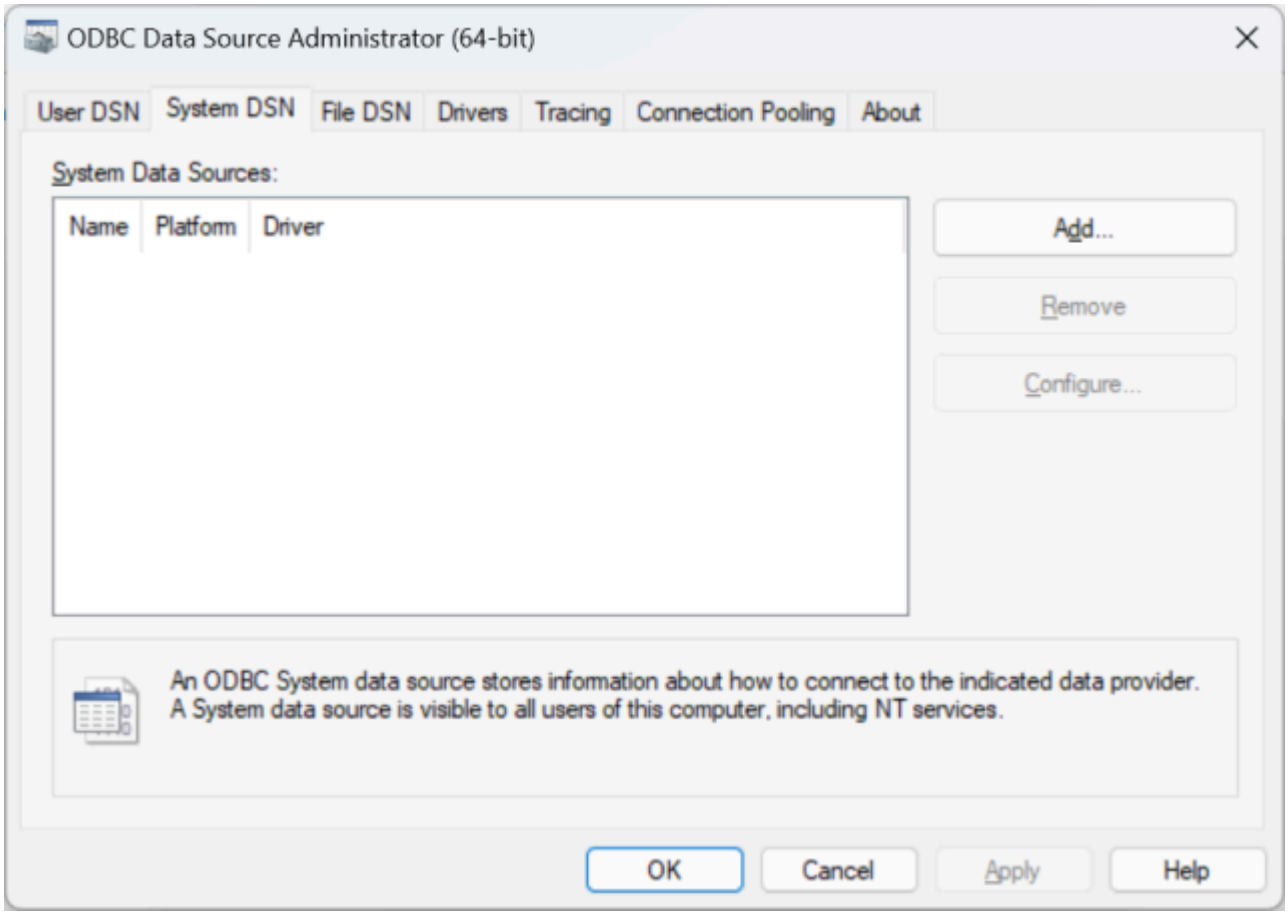
Setting up the Amazon DocumentDB ODBC driver in Windows

Use the following procedure to set up the Amazon DocumentDB ODBC driver in Windows:

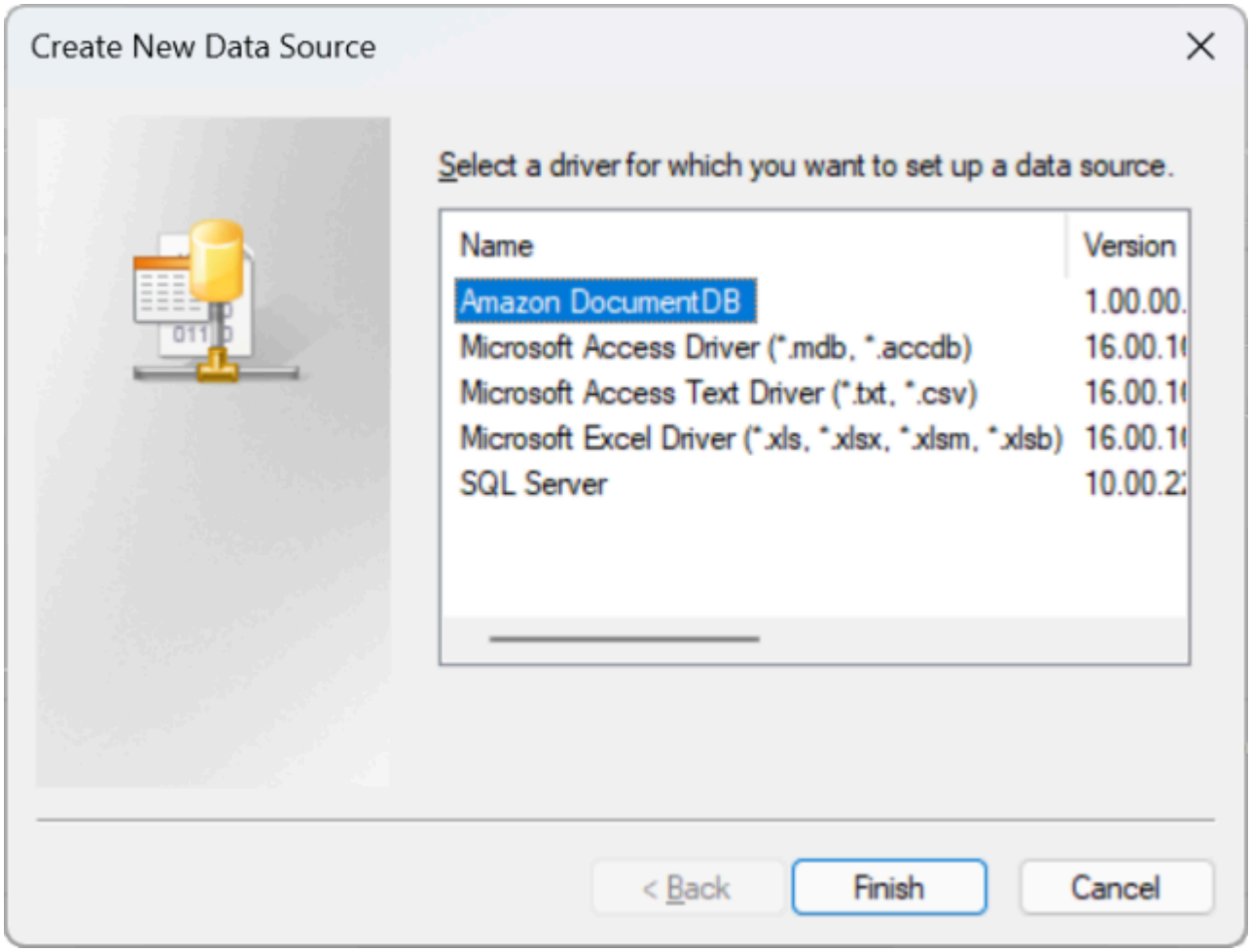
1. Open **Control Panel** in Windows and search for ODBC (or in the menu, select **Windows Tools** > **ODBC Data Sources (32-bit)** or **ODBC Data Sources (64-bit)**):



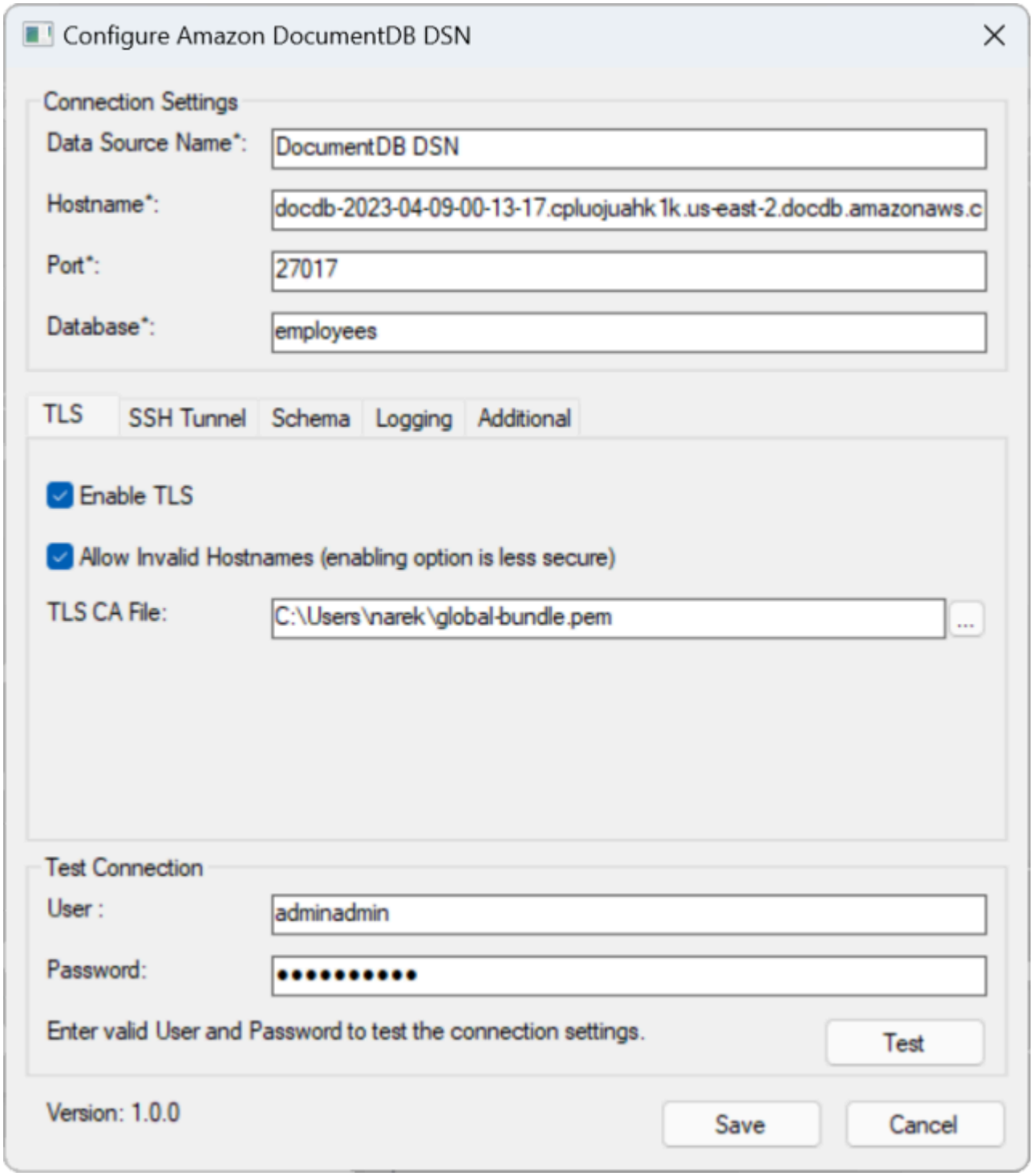
2. Select the appropriate ODBC Driver Data Source Administrator: opt for the 32-bit version if it is installed, otherwise, choose the 64-bit version.
3. Select the Sytem DSN tab and then click **Add...** to add a new DSN:



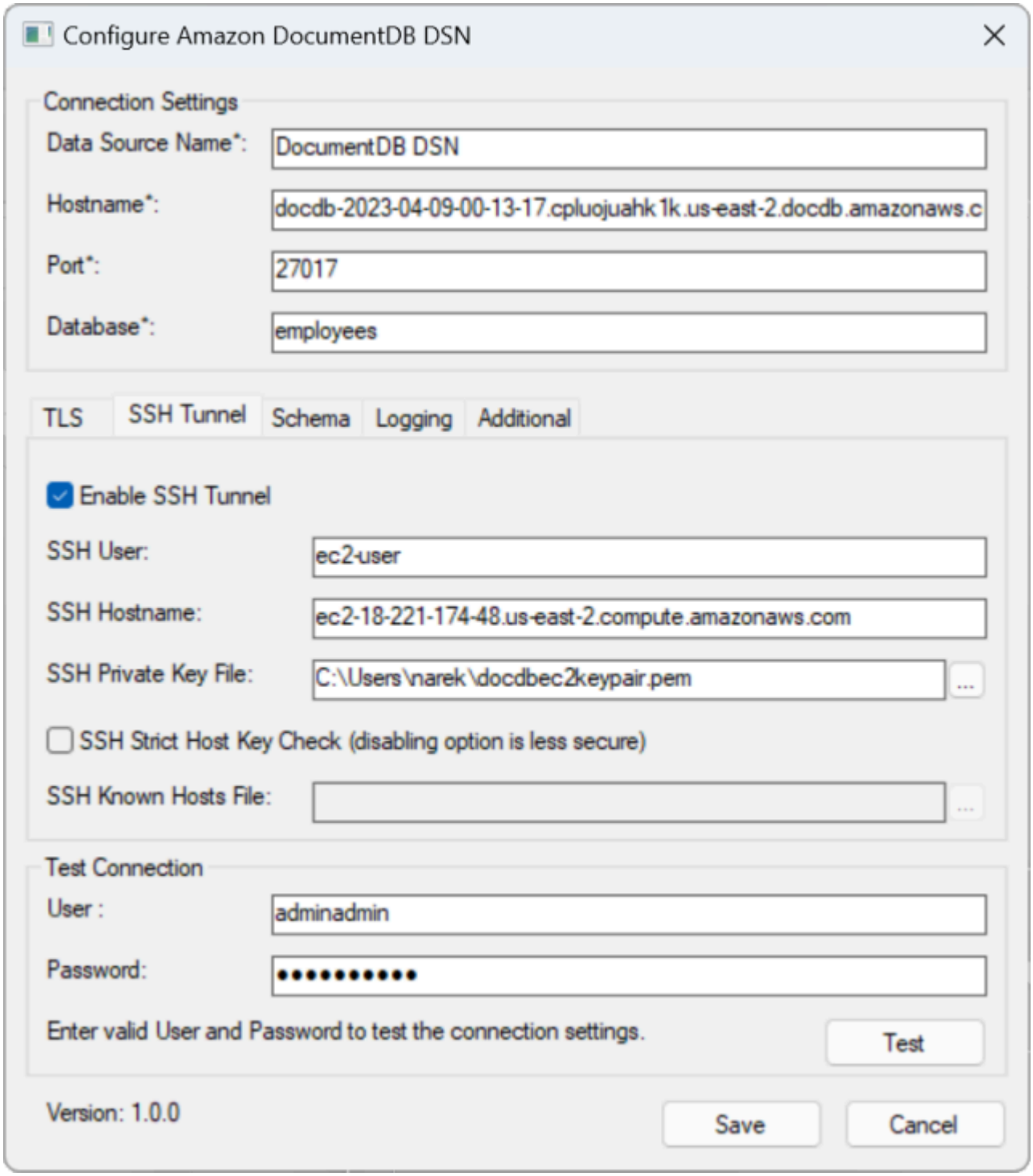
4. Choose **Amazon DocumentDB** from the data source driver list:



5. In the **Configure Amazon DocumentDB DSN** dialog, complete the **Connection Settings**, **TLS** tab, and **Test Connection** fields, then click **Save**:



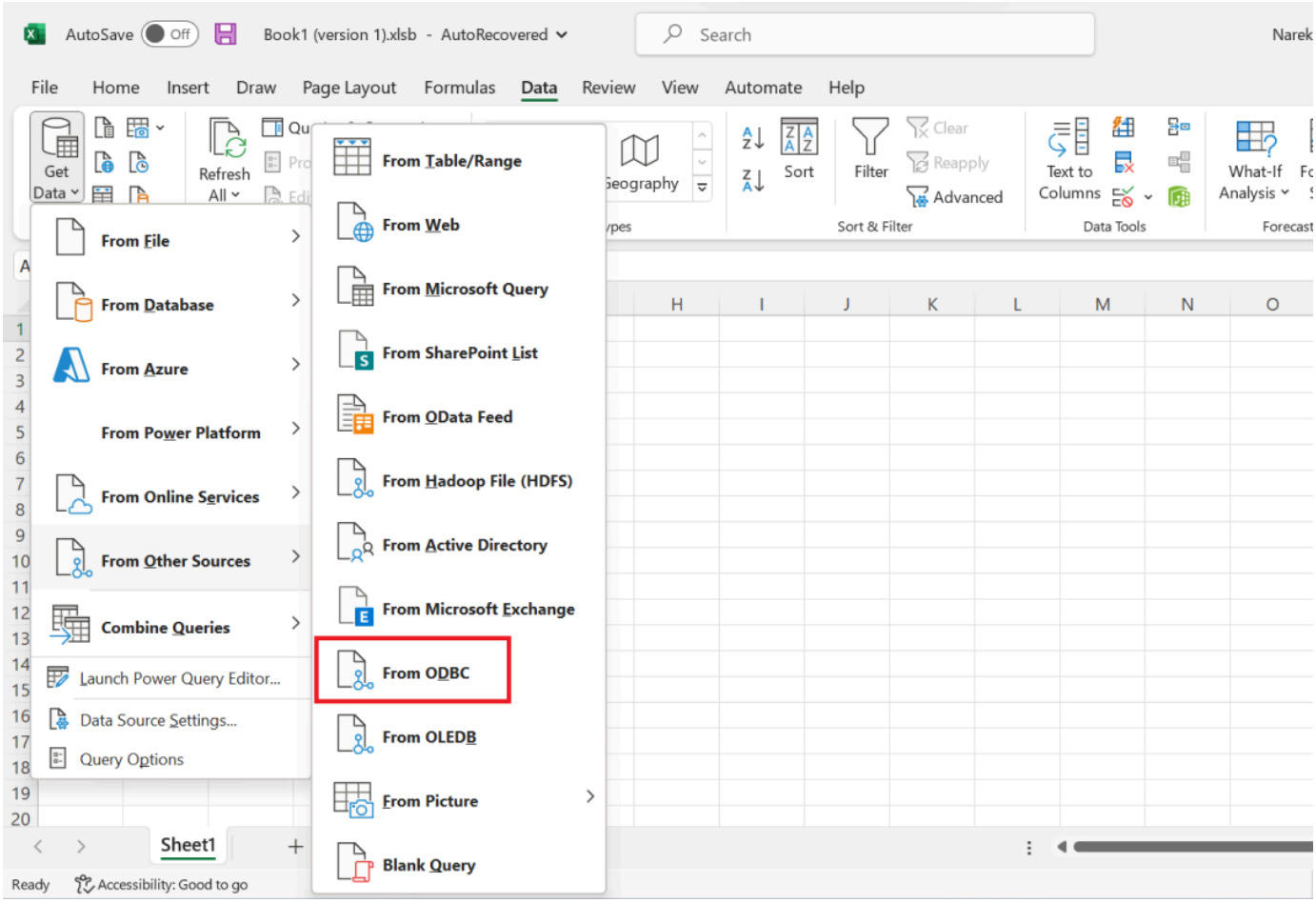
6. Ensure you complete the Windows form accurately, as connection details will differ depending on your chosen SSH tunneling method to the EC2 instance. See SSH tunneling methods here. See Connection String Syntax and Options for more information about each property.



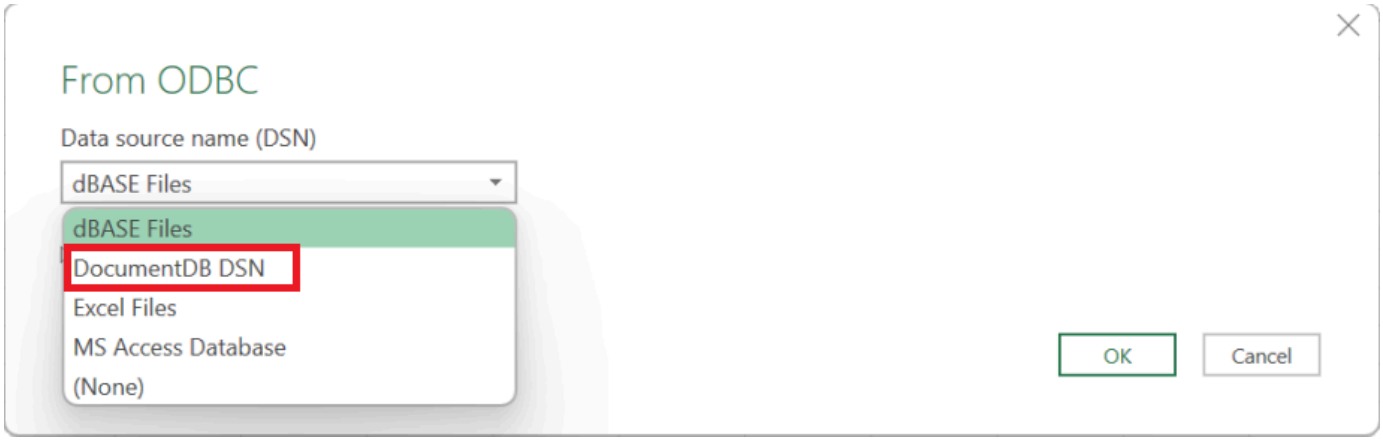
For more information about configuring the Amazon DocumentDB ODBC Driver on Windows, click here.

Connect to Amazon DocumentDB from Microsoft Excel

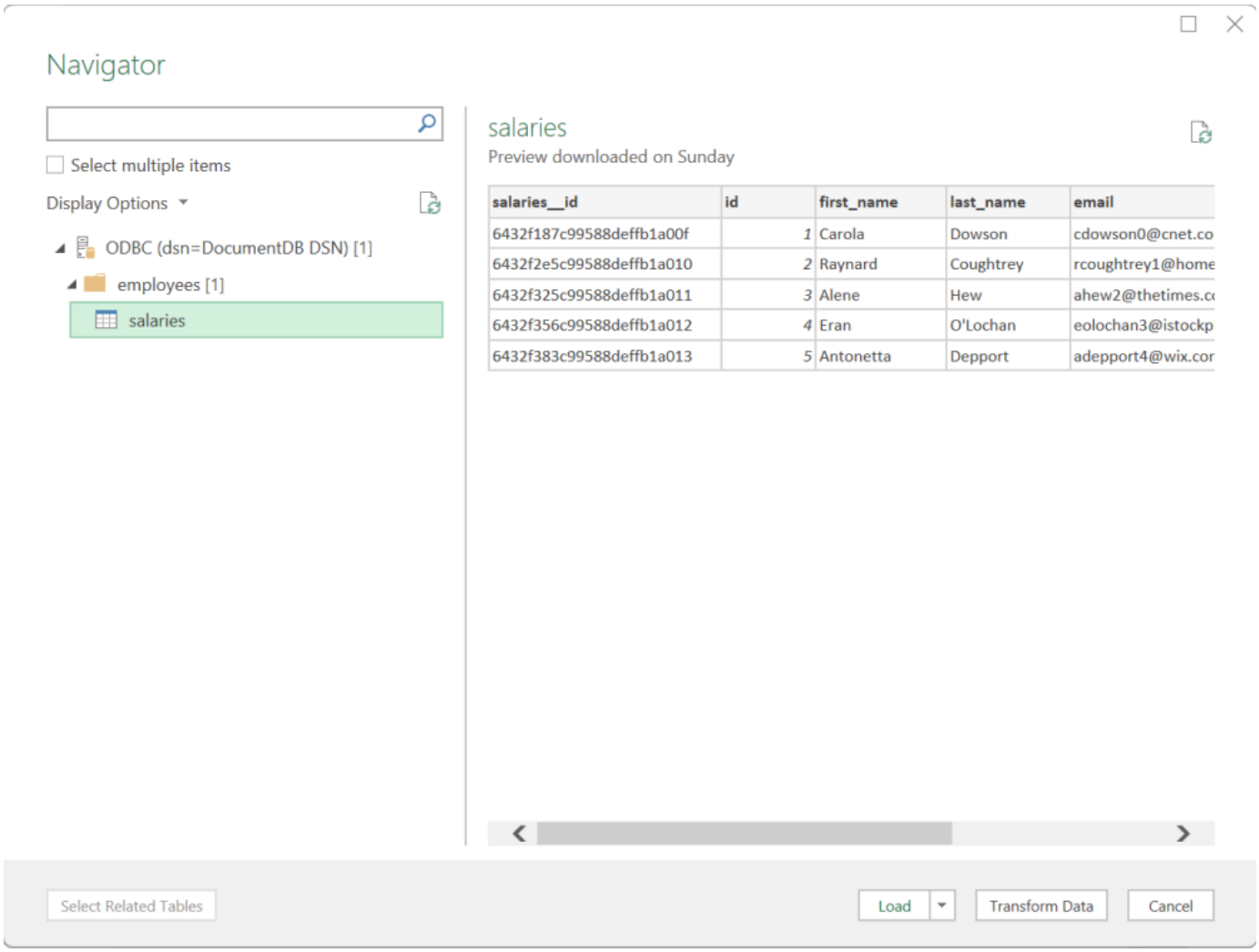
1. Ensure that the Amazon DocumentDB driver has been correctly installed and configured. For additional information, refer to Setting up the ODBC driver in Windows.
2. Launch Microsoft Excel.
3. Navigate to **Data** > **Get Data** > **From Other Sources**.
4. Choose **From ODBC**:



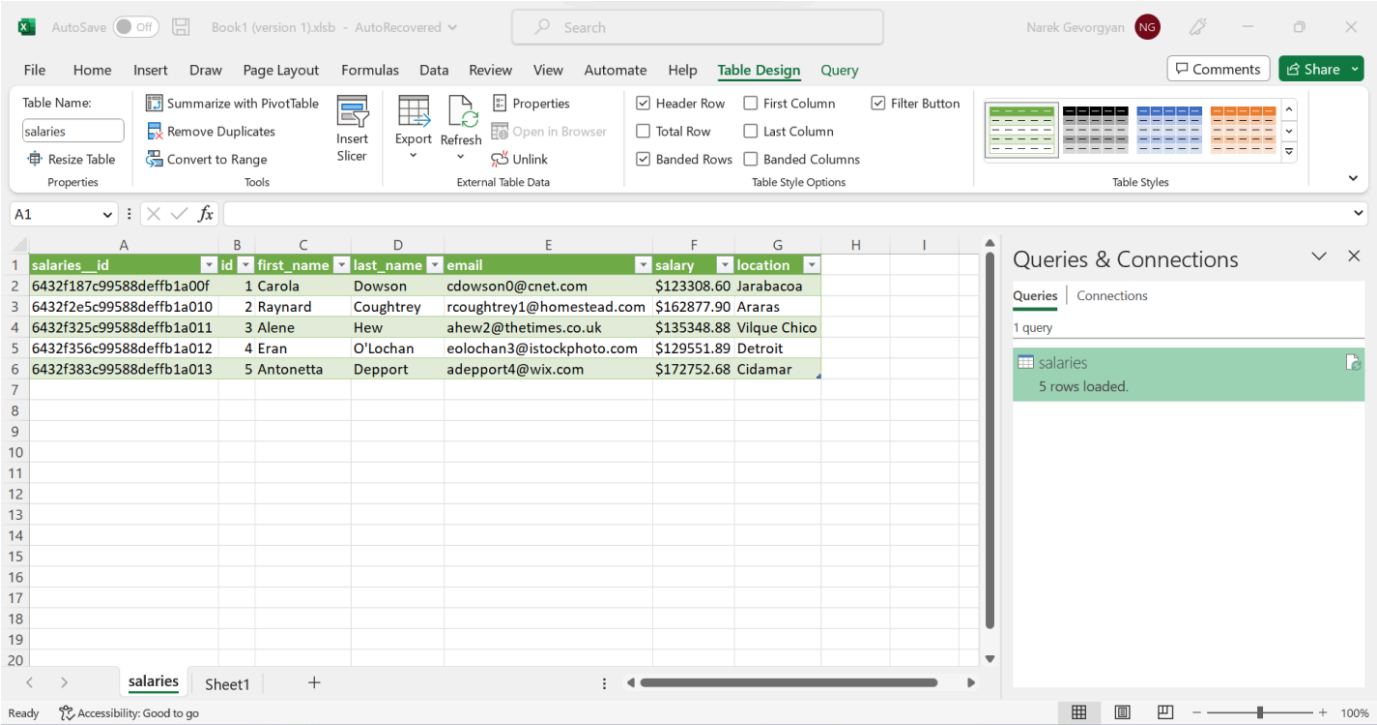
5. Select the data source from the **Data source name (DSN)** drop down menu that is associated with Amazon DocumentDB:



6. Choose the collection from which you want to load data into Excel:



7. Load data into Excel:



Connect to Amazon DocumentDB from Microsoft Power BI Desktop

Topics

- Prerequisites
- Adding Microsoft Power BI Desktop custom connector
- Connecting using the Amazon DocumentDB custom connector
- Configuring Microsoft Power BI Gateway

Prerequisites

Before beginning, ensure that the Amazon DocumentDB ODBC driver is correctly installed.

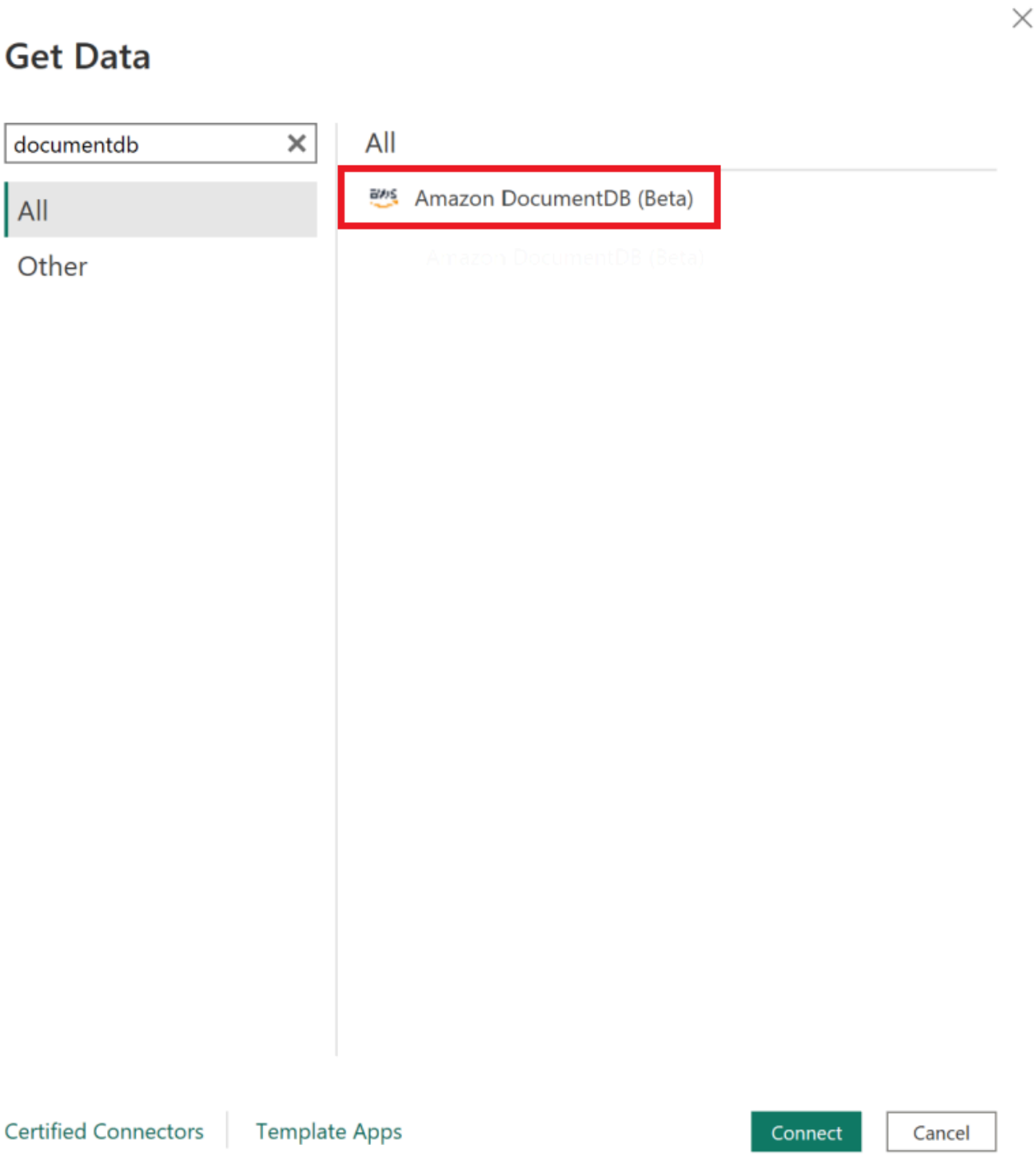
Adding Microsoft Power BI Desktop custom connector

Copy the `AmazonDocumentDBConnector.mez` file to the `<User>\Documents\Power BI Desktop\Custom Connectors\` folder (or to `<User>\OneDrive\Documents\Power BI Desktop\Custom Connectors` if using OneDrive). This will allow Power BI to access custom connector. You can get the connector to Power BI Desktop here. Restart Power BI Desktop to make sure the connector is loaded.

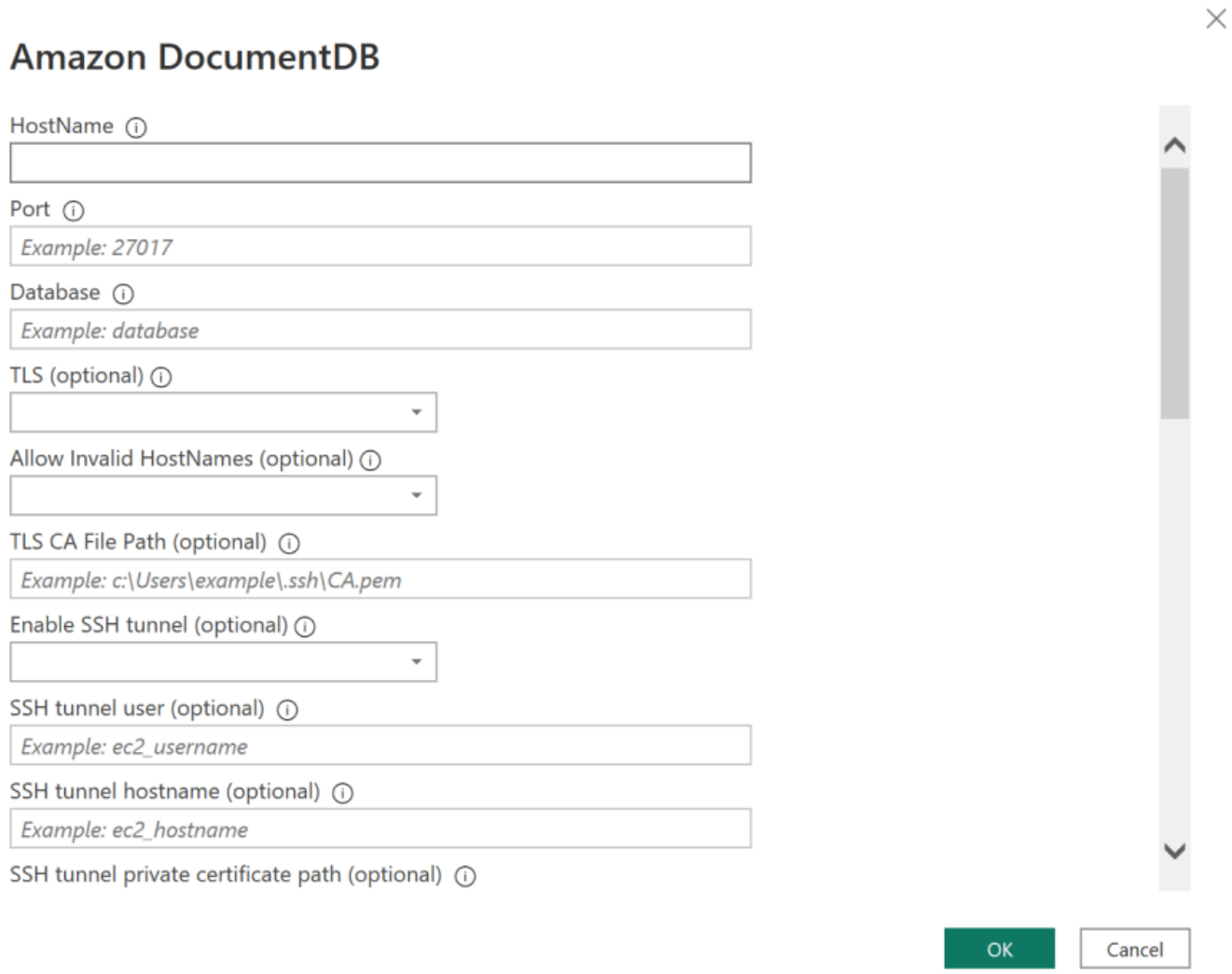
> **Note**
>
> The custom connector only supports Amazon DocumentDB username and password for authentication.

Connecting using the Amazon DocumentDB custom connector

1. Select Amazon DocumentDB (Beta) from **Get Data** and click **Connect**. If you get a warning for using a third-party service, click **Continue**.



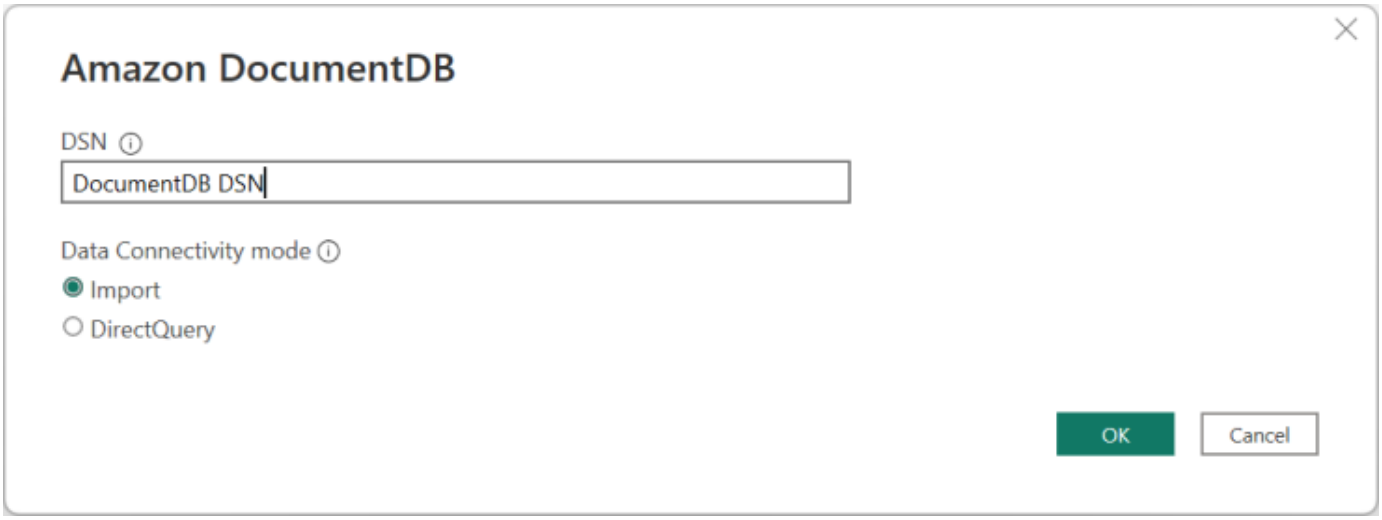
2. Enter all necessary information to connect to your Amazon DocumentDB cluster, then click **OK**:



> **Note**
>
> Depending on the configuration of your ODBC driver's Data Source Name (DSN), the SSH connection details screen may not be displayed if you have already provided the necessary information within the DSN settings.

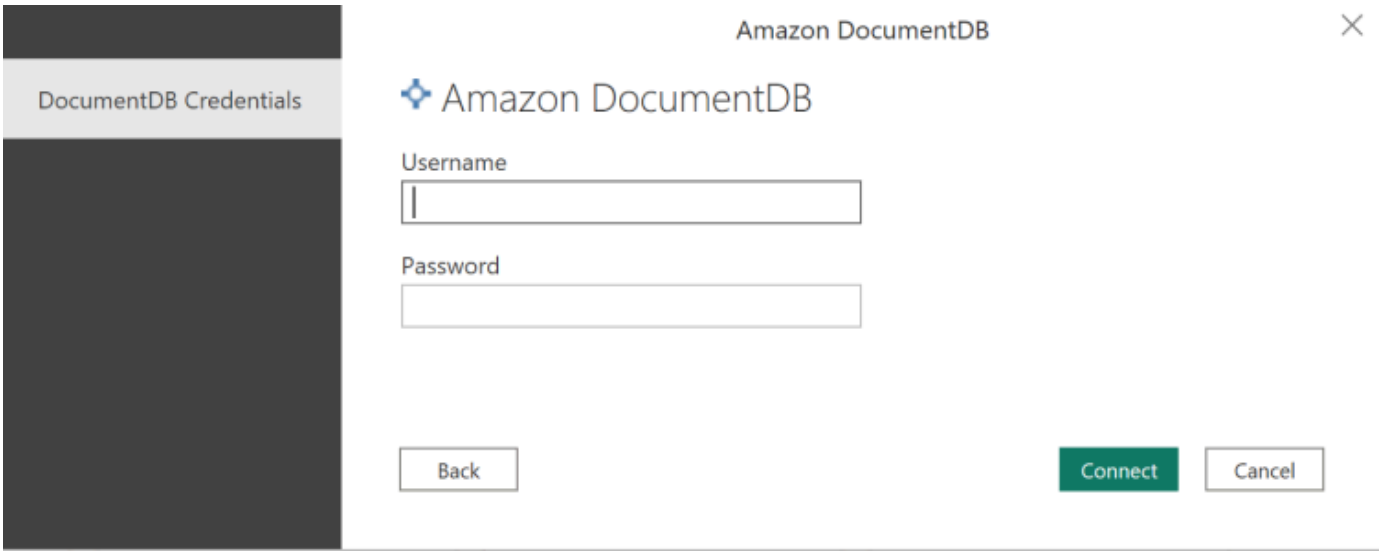
3. Choose the data connectivity mode:

   - **Import** - loads all data and stores the information on disk. The data must be refreshed and reloaded in order to show data updates.
   - **Direct Query** - does not load data, but does live queries on the data. This means that data does not need to be refreshed and reloaded in order to show data updates.

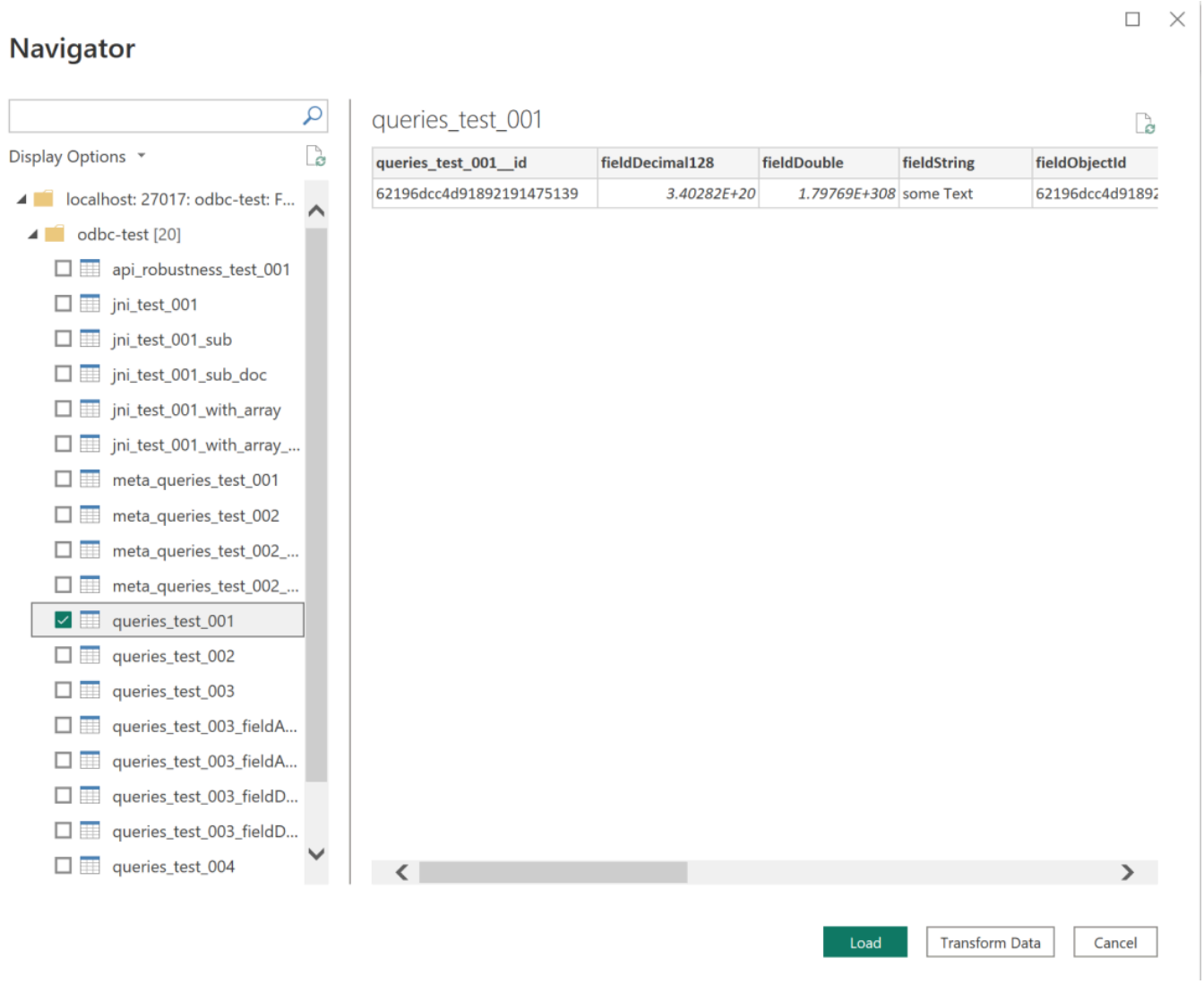


> **Note**
>
> If you are using a very large dataset, importing all of the data may take a longer period of time.

4. If this is the first time connecting to this data source, select the authentication type and input your credentials when prompted. Then click **Connect**:



5. In the **Navigator** dialog, select the database tables you want, then either click **Load** to load the data or **Transform Data** to continue transforming the data.



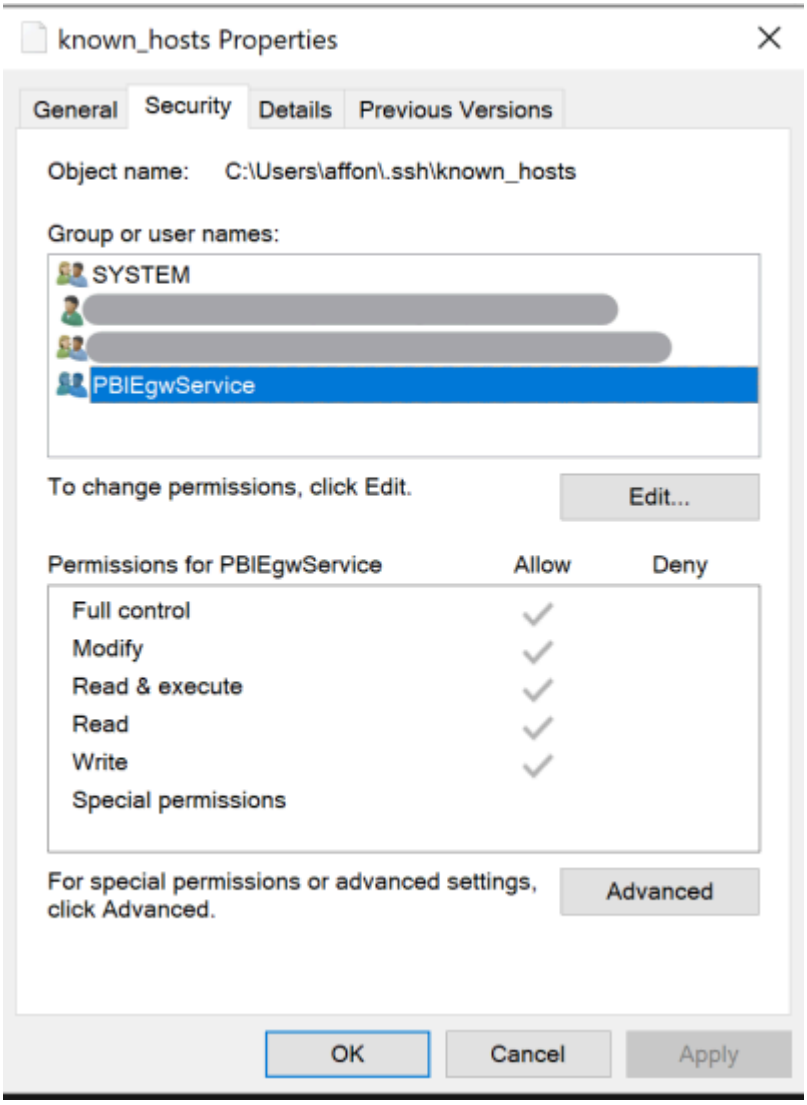
> **Note**
>
> Your data source settings are saved once you connect. To modify them, select **Transform Data** > **Data Source Settings**.

Configuring Microsoft Power BI Gateway

Prerequisites:

- Enure that the custom connector will work with Power BI Gateway.
- Make sure that the ODBC DSN is created in the ODBC data sources in the **System** tab on the machine where Power BI Gateway is installed.

If you are using the internal SSH tunnel feature, the file `known_hosts` needs to be located where the Power BI service account has access to it.



> **Note**
>
> This also applies to any file(s) that you might need to be able to establish a connection to your Amazon DocumentDB cluster, such as a certificate authority (CA) certificate file (pem file).

Automatic schema generation

The ODBC driver is utilizing the Amazon DocumentDB JDBC driver through JNI (Java Native Interface) - making the automatic schema generation feature to work similarly in the JDBC driver.

For more information on automatic schema generation, see JDBC automatic schema generation. Additionally, to learn more about the ODBC driver architecture, click here.

SQL support and limitations

The Amazon DocumentDB ODBC driver is a read-only driver that supports a subset of SQL-92 and some common extensions. Refer to the ODBC support and limitations documentation for more information.

Troubleshooting

If you are having problems using the Amazon DocumentDB ODBC driver, refer to the Troubleshooting Guide.

Program with Amazon DocumentDB

The service supports JSON schema validation.

Topics

- Amazon DocumentDB Java programming guide
- Using JSON schema validation

Amazon DocumentDB Java programming guide

This comprehensive guide provides a detailed walk through for working with Amazon DocumentDB using MongoDB's Java drivers, covering essential aspects of database operations and management.

Topics

- Introduction
- Prerequisites
- Data models for this guide
- Connecting to Amazon DocumentDB with a MongoDB Java driver
- Performing CRUD operations in Amazon DocumentDB with Java
- Index management in Amazon DocumentDB with Java
- Event-driven programming with Amazon DocumentDB and Java

Introduction

The guide begins with connectivity, explaining how to establish secure connections to DocumentDB clusters using the MongoDB Java driver. It details the connection string components, SSL/TLS implementation, and various connection options including IAM authentication and connection pooling, along with robust error handling strategies.

In the CRUD (create, read, update, delete) operations section, the guide thoroughly covers document manipulation, demonstrating how to create, read, update, and delete documents using both single and bulk operations. It explains the usage of filters, queries, and various operation options, while emphasizing best practices for error handling and implementing retry logic for improved reliability. The guide also extensively covers index management, detailing the creation and maintenance of different index types including single-field, compound, sparse, and text indexes. It explains how to optimize query performance through proper index selection and usage of the `explain()` function to analyze query execution plans.

The final section focuses on event-driven programming using Amazon DocumentDB's change streams, demonstrating how to implement real-time data change monitoring in Java applications. It covers the implementation of change stream cursors, handling of resume tokens for continuous operation, and time-based operations for historical data processing. Throughout the guide, practical code examples and best practices are provided, making it an invaluable resource for you when building robust Java applications with Amazon DocumentDB.

Prerequisites

Before you begin, make sure you have the following:

- An AWS account with a configured DocumentDB cluster. See this getting started blog post for DocumentDB cluster setup.
- Java Development Kit (JDK) installed (we will be using Amazon Corretto 21 for this guide).
- Maven for dependency management.

Data models for this guide

All the example code in this guide assumes a connection to a “ProgGuideData” test database that has a “Restaurants” collection. All the sample codes in this guide work on a restaurant listing system and below is an example of what a document in this system looks like:

```
{
```

```
    "_id": "ab6ad8f119b5bca3efa2c7ae",
    "restaurantId": "REST-CRT9BL",
    "name": "Thai Curry Palace",
    "description": "Amazing Restaurant, must visit",
    "cuisine": "Thai",
    "address": {
        "street": "914 Park Street",
        "city": "Bryan",
        "state": "AL",
        "zipCode": "96865",
        "location": {
            "type": "Point",
            "coordinates": [-25.4619, 8.389]
        }
    },
    "contact": {
        "phone": "(669) 915-9056 x6657"
    },
    "rating": {
        "average": 3.4,
        "totalReviews": 275
    },
    "priceRange": "$",
    "menu": [{
        "category": "Appetizers",
        "items": [{
            "name": "Buffalo Chicken Wings",
            "price": 13.42
        }]
    }],
    "features": [
        "Private Dining"
    ],
    "isActive": false“ michelin”: {“
        star”: 3,
        “ranking_years”: 4
    }
}
```

All code samples showing CRUD, index management, and event-driven programming assumes that you have a `MongoClient` object `dbClient`, a `MongoDatabase` object `connectionDB`, and a `MongoCollection` object `collection`.

> **Note**
>
> All code examples in this guide have been tested with MongoDB Java driver version 5.3.0.

Connecting to Amazon DocumentDB with a MongoDB Java driver

This section provides a step-by-step guide for connecting to Amazon DocumentDB using Java drivers. This will get you started with integrating DocumentDB into your Java applications.

Topics

- Step 1: Set up your project
- Step 2: Create the connection string
- Step 3: Write the connection code
- Step 4: Handle connection exceptions
- Step 5: Running the code
- Connection best practices

Step 1: Set up your project

1. Using Maven, create java project:

```
mvn archetype:generate -DgroupId=com.docdb.guide -DartifactId=my-docdb-project -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

2. Add the MongoDB Java driver as a dependency for the project in your 'pom.xml' file:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>5.3.0</version>
</dependency>
```

Step 2: Create the connection string

Amazon DocumentDB connection string is essential for establishing a connection between your application and your DocumentDB cluster. This string encapsulates crucial information such as

the cluster endpoint, port, authentication details, and various connection options. To build a DocumentDB connection string, you typically start with the basic format:

```
"mongodb://username:password@cluster-endpoint:port/?[connection options]"
```

You'll need to replace "username" and "password" with your actual credentials. You can find your cluster's endpoint and port number in the AWS Management Console as well as through the AWS CLI. See Finding a cluster's endpoints to find the cluster endpoint for your cluster. The default port for DocumentDB is 27017.

Connection string examples

- Making a connection to DocumentDB using encryption in transit and making sure read requests go to read replicas and write to primary:

  ```
  "mongodb://username:password@cluster-endpoint:27017/?tls=true&
     tlsCAFile=global-bundle.pem&
     readPreference=secondaryPreferred&
     retryWrites=false"
  ```

- Making a connection to DocumentDB using IAM authentication:

  ```
  "mongodb://cluster-endpoint:27017/?tls=true&
     tlsCAFile=global-bundle.pem&
     readPreference=secondaryPreferred&
     retryWrites=false&
     authSource=%24external&
     authMechanism=MONGODB-AWS"
  ```

The different options available for the connection string are as follows:

- TLS certificate
- Reading from read replicas
- Write concern and journaling
- RetryWrites
- IAM authentication
- Connection pool
- Connection timeout parameters

TLS certificate

`tls=true|false` — This option enables or disables Transport Layer Security (TLS). By default, encryption in transit is enabled on Amazon DocumentDB cluster and therefore, unless TLS is disabled at the cluster level, the value for this option should be `true`.

When using TLS, the code needs to provide an SSL certificate when creating connection to a DocumentDB cluster. Download the certificate that is required to make the secure connection to the cluster: `global-bundle.pem`. There are two ways to use the `global-bundle.pem` file.

- **Option 1** — Extract all the certificates from the `global-bundle.pem` file and use Java's keytool to store them in a `.jks` file that can be later used in the code. Refer to the Java tab in Connecting with TLS enabled for the script that shows how to do this.
- **Option 2** — Dynamically add the `global-bundle.pem` file in the code, build an in-memory keystore and use `SSLContext` to provide the certificate as part of making the connection.

Reading from read replicas

`replicaSet=rs0&readPreference=secondaryPreferred` — Specifying these two options routes all the read requests to the read replicas, and write requests to the primary instance. Using `replicaSet=rs0` in the connection string enables the MongoDB driver to maintain an automatically updated view of the cluster topology, allowing applications to maintain visibility of current node configurations as instances are added or removed. Not providing these options or specifying `readPreference=primary` sends all the reads and writes to primary instance. For more options for `readPreference`, see Read preference options.

Write concern and journaling

Write concern determines the level of acknowledgment requested from the database for write operations. MongoDB drivers provide an option to tune write concern and journal files. Amazon DocumentDB doesn't expect you to set write concern and journal, and ignores the values sent for `w` and `j` (`writeConcern` and `journal`). DocumentDB always writes data with `writeConcern`: `majority` and `journal`: `true` so the writes are durably recorded on a majority of nodes before sending an acknowledgement to the client.

RetryWrites

`retryWrites=false` — DocumentDB does not support retryable writes and therefore this attribute should always be set to `false`.

IAM authentication

`authSource=%24external` and `authMechanism=MONGODB-AWS` — These two parameters are used to authenticate using AWS Identity and Access Management. IAM authentication is currently only available in instance-based cluster version 5.0. For more information, see Authentication using IAM identity.

Connection pool

These options are available for connection pooling:

- **`maxPoolSize`** — Sets the maximum number of connections that can be created in the pool. When all connections are in use and a new request comes in, it waits for a connection to become available. The default for MongoDB Java drivers is 100.
- **`minPoolSize`** — Indicates the minimum number of connections that should be maintained in the pool at all times. The default for MongoDB Java drivers is 0.
- **`maxIdleTimeMS`** — Determines how long a connection can remain idle in the pool before being closed and removed. The default for MongoDB Java drivers is 100 milliseconds.
- **`waitQueueTimeoutMS`** — Configures how long a thread should wait for a connection to become available when the pool is at its maximum size. If a connection doesn't become available within this time, an exception is thrown. The default value for MongoDB Java drivers is 120,000 milliseconds (2 minutes).

Connection timeout parameters

Timeout is a mechanism to limit the amount of time an operation or connection attempt can take before it is considered failed. The following timeout parameters are available for preventing indefinite waits and managing resource allocation:

- **`connectTimeoutMS`** — Configures how long the driver will wait to establish a connection to the cluster. The default is 10,000 milliseconds (10 seconds).
- **`socketTimeoutMS`** — Specifies how long the driver will wait for a response from the server for a non-write operation. The default is 0, (no timeout or infinite).
- **`serverSelectionTimeoutMS`** — Species how long the driver will wait to find an available server in the cluster. The default value for this setting is 30 seconds and is sufficient for a new primary instance to be elected during failover.

Step 3: Write the connection code

The following code example shows how to make a TLS connection to Amazon DocumentDB:

- It creates Java's `KeyStore` and `SSLContext>` objects.
- It also creates the `MongoClientSettings` object by passing it to the `ConnectionString` object. To make TLS connection, you must use the `MongoClientSettings` object to bind the `connectionstring` and `sslcontext`.
- Using `MongoClients` gets a `MongoClient` object.

```
public static MongoClient makeDbConnection(String dbName, String DbUserName, String
 DbPassword,
    String DbClusterEndPoint, String keyStorePass) throws Exception {
    MongoClient connectedClient;
    String connectionOptions = "?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false";
    String connectionUrl = "mongodb://" + DbUserName + ":" + DbPassword + "@" +
 DbClusterEndPoint + ":27017/" +
        dbName + connectionOptions;

    try {
        KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
        try (FileInputStream fis = new FileInputStream("src/main/resources/certs/
truststore.jks")) {
            trustStore.load(fis, keyStorePass.toCharArray());
            TrustManagerFactory tmf =
 TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
            tmf.init(trustStore);

            SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
            sslContext.init(null, tmf.getTrustManagers(), new SecureRandom());
            ConnectionString connectionString = new ConnectionString(connectionUrl);
            MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(connectionString)
                .applyToSslSettings(builder - > {
                    builder.enabled(true);
                    builder.context(sslContext);
                })
                .build();
            connectedClient = MongoClients.create(settings);
        }
```

```
        return connectedClient;
    } catch (MongoException e5) {
        throw new RuntimeException(e5);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Step 4: Handle connection exceptions

When working with DocumentDB in Java applications, handling connection exceptions is crucial for maintaining robust and reliable database operations. Properly managing, these exceptions not only help in diagnosing issues quickly, but also ensure that your application can gracefully handle temporary network disruptions or server unavailability, leading to improved stability and user experience. Some of the critical exceptions related to establishing connection include:

- **`MongoException`** — A general exception and could be issued in various scenarios not covered by more specific exceptions. Make sure this exception is handled after all the other specific exceptions as this is a general catch all MongoDB exception.
- **`MongoTimeoutException`** — Issued when an operation times out. For example, querying a nonexistent cluster endpoint.
- **`MongoSocketException`** — Issued for network-related issues. For example, sudden network disconnection during an operation.
- **`MongoSecurityException`** — Issued when authentication fails. For example, connecting with incorrect credentials.
- **`MongoConfigurationException`** — Issued when there is an error in the client configuration. For example, using an invalid connection string.

Step 5: Running the code

The following code sample creates an Amazon DocumentDB connection and prints all the databases:

```
public static void TestConnection() {
    try (MongoClient mongoClient = makeDbConnection(DATABASE_NAME, DB_USER_NAME,
 DB_PASSWORD, DB_CLUSTER_ENDPOINT, KEYSTORE_PASSWORD)) {
        List < String > databases = mongoClient.listDatabaseNames().into(new ArrayList
 < > ());
        System.out.println("Databases: " + databases);
```

```
    } catch (MongoException e) {
        System.err.println("MongoDB error: " + e.getMessage());
        throw new RuntimeException(e);
    }
}
```

Connection best practices

The following are best practices to consider when connecting to Amazon DocumentDB with a MongoDB Java driver:

- Always close your `MongoClient` when you no longer need the client to release resources.
- Handle exceptions appropriately and implement proper error logging.
- Use environment variables or AWS Secrets Manager to store sensitive information like usernames and passwords.

Performing CRUD operations in Amazon DocumentDB with Java

This section discusses performing CRUD (create, read, update, delete) operation in Amazon DocumentDB using MongoDB Java drivers.

Topics

- Creating and inserting documents in a DocumentDB collection
- Reading and retrieving data from a DocumentDB collection
- Updating existing documents in a DocumentDB collection
- Removing documents from a DocumentDB collection
- Error handling with retry logic

Creating and inserting documents in a DocumentDB collection

Inserting documents into Amazon DocumentDB allows you to add new data to your collections. There are several ways to perform insertions, depending on your needs and the volume of data you're working with. The most basic method for inserting an individual document to the collection is `insertOne()`. For inserting multiple documents at once, you can use the `insertMany()` method, which allows you to add an array of documents in a single operation. Another method for inserting many documents in a DocumentDB collection is `bulkWrite()`. In this guide, we discuss all of these methods for creating documents in a DocumentDB collection.

insertOne()

Let's begin by examining how to insert an individual document into an Amazon DocumentDBB collection. Inserting a single document is accomplished by using the `insertOne()` method. This method takes a BsonDocument for insertion and returns an `InsertOneResult` object that can be used to get the object id of the new inserted document. The example code below shows inserting one restaurant document into the collection:

```
Document article = new Document()
    .append("restaurantId", "REST-21G145")
    .append("name", "Future-proofed Intelligent Bronze Hat")
    .append("cuisine", "International")
    .append("rating", new Document()
        .append("average", 1.8)
        .append("totalReviews", 267))
    .append("features", Arrays.asList("Outdoor Seating", "Live Music"));

try {
    InsertOneResult result = collection.insertOne(article);
    System.out.println("Inserted document with the following id: " +
 result.getInsertedId());
} catch (MongoWriteException e) {
    // Handle duplicate key or other write errors
    System.err.println("Failed to insert document: " + e.getMessage());
    throw e;
} catch (MongoException e) {
    // Handle other MongoDB errors
    System.err.println("MongoDB error: " + e.getMessage());
    throw e;
}
```

When using `insertOne()`, make sure to include appropriate error handling. For instance, in the above code, “`restaurantId`” has a unique index and therefore running this code again will raise the following `MongoWriteException`:

```
Failed to insert document: Write operation error on server
 docdbCluster.docdb.amazonaws.com:27017.
Write error: WriteError{code=11000, message='E11000 duplicate key error collection:
 Restaurants index: restaurantId_1', details={}}.
```

insertMany()

The primary methods used for inserting many document into a collection are insertMany() and `bulkWrite()`.

The `insertMany()` method is the simplest way to insert multiple documents in a single operation. It accepts a list of documents and inserts them into the collection. This method is ideal when you're inserting a batch of new documents that are independent of each other and don't require any special processing or mixed operations. The following code shows reading JSON documents from a file and inserting them into the collection. The `insertMany()` function returns an `InsertManyResult``InsertManyResult` object that can be used to get the IDs of all the inserted documents.

```
// Read JSON file content
String content = new String(Files.readAllBytes(Paths.get(jsonFileName)));
JSONArray jsonArray = new JSONArray(content);

// Convert JSON articles to Documents
List < Document > restaurants = new ArrayList < > ();
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObject = jsonArray.getJSONObject(i);
    Document doc = Document.parse(jsonObject.toString());
    restaurants.add(doc);
}
//insert documents in collection
InsertManyResult result = collection.insertMany(restaurants);

System.out.println("Count of inserted documents: " + result.getInsertedIds().size());
```

`bulkWrite()`

The `bulkWrite()` method allows to perform multiple write operations (insert, update, delete) in a single batch. You can use `bulkWrite()` when you need to perform different types of operations in a single batch, such as inserting some documents while updating others. `bulkWrite()` support two types of batch write, ordered and unordered:

- *Ordered operations* — (default) Amazon DocumentDB processes the write operations sequentially, and stops at the first error it encounters. This is useful when the order of operations matters, such as when later operations depend on earlier ones. However, ordered operations are generally slower then unordered operations. With ordered operations, you must address the case where the batch stops at the first error, potentially leaving some operations unprocessed.

- *Unordered operations* — Allows Amazon DocumentDB to process inserts as a single execution in the database. If an error occurs with one document, the operation continues with the remaining documents. This is particularly useful when you're inserting large amounts of data and can tolerate some failures, such as during data migration or bulk imports where some documents might fail due to duplicate keys. With unordered operations, you must address partial success scenarios where some operations succeed while others fail.

When working with the `bulkWrite()` method, there are some essential classes that are required. First, the `WriteModel` class serves as the base class for all write operations and with specific implementations like `InsertOneModel`, `UpdateOneModel`, `UpdateManyModel`, `DeleteOneModel`, and `DeleteManyModel` handling different types of operations.

The `BulkWriteOptions` class is necessary for configuring the behavior of bulk operations, such as setting ordered/unordered execution or bypassing document validation. The `BulkWriteResult` class provides detailed information about the execution results, including counts of inserted, updated, and deleted documents.

For error handling, the `MongoBulkWriteException` class is crucial as it contains information about any failures during the bulk operation, while the `BulkWriteError` class provides specific details about individual operation failures. The following code shows an example of inserting a list of documents, as well as updating and deleting a single document, all within the execution of single `bulkWrite()` method call. The code also shows how to work with `BulkWriteOptions` and `BulkWriteResult`, as well as proper error handling of the `bulkWrite()` operation.

```
List < WriteModel < Document >> bulkOperations = new ArrayList < > ();
// get list of 10 documents representing 10 restaurants
List < Document > restaurantsToInsert = getSampleData();

for (Document doc: restaurantsToInsert) {
    bulkOperations.add(new InsertOneModel < > (doc));
}
// Update operation
bulkOperations.add(new UpdateOneModel < > (
    new Document("restaurantId", "REST-Y2E9H5"),
    new Document("", new Document("stats.likes", 20))
    .append("", new Document("rating.average", 4.5))));
// Delete operation
bulkOperations.add(new DeleteOneModel < > (new Document("restaurantId", "REST-
D2L431")));
```

```
// Perform bulkWrite operation
try {
    BulkWriteOptions options = new BulkWriteOptions()
        .ordered(false); // Allow unordered inserts

    BulkWriteResult result = collection.bulkWrite(bulkOperations, options);

    System.out.println("Inserted: " + result.getInsertedCount());
    System.out.println("Updated: " + result.getModifiedCount());
    System.out.println("Deleted: " + result.getDeletedCount());
} catch (MongoBulkWriteException e) {
    System.err.println("Bulk write error occurred: " + e.getMessage());
    // Log individual write errors
    for (BulkWriteError error: e.getWriteErrors()) {
        System.err.printf("Error at index %d: %s (Code: %d)%n", error.getIndex(),
 error.getMessage(),
            error.getCode());

        // Log the problematic document
        Document errorDoc = new Document(error.getDetails());
        if (errorDoc != null) {
            System.err.println("Problematic document: " + errorDoc);
        }
    }
} catch (Exception e) {
    System.err.println("Error during bulkWrite: " + e.getMessage());
}
```

Retryable writes

Unlike MongoDB, Amazon DocumentDB doesn't support retryable writes. As a result, you must implement custom retry logic in their applications, particularly for handling network issues or temporary service unavailability. A well-implemented retry strategy, typically, involves increasing the delay between retry attempts and limiting the total number of retries. See Error handling with retry logic below for a code sample of building retry logic with error handling.

Reading and retrieving data from a DocumentDB collection

Querying documents in Amazon DocumentDB revolves around several key components that allow you to precisely retrieve and manipulate data. The `find()` method is the fundamental querying APIs in MongoDB Java drivers. It allows for complex data retrieval with numerous options for filtering, sorting, and projecting results. Besides the `find()` method, `Filters` and

`FindIterable` are two other fundamental components that provide the building blocks for query operations in MongoDB Java drivers.

The `Filters` class is a utility class in the MongoDB Java driver that provides a fluent API for constructing query filters. This class offers static factory methods that create instances of `Bson` objects representing various query conditions. The most commonly used methods include `eq()` for equality comparisons, `gt()`, `lt()`, `gte()`, and `lte()` for numeric comparisons, `and()` and `or()` for combining multiple conditions, `in()` and `nin()` for array membership tests, and `regex()` for pattern matching. The class is designed to be type-safe and provides better compile-time checking compared to raw document-based queries, making it the preferred approach for constructing DocumentDB queries in Java applications. Error handling is robust, with clear exceptions thrown for invalid filter constructions.

`FindIterable` is a specialized interface designed to handle the result of the `find()` method. It provides a rich set of methods for refining and controlling query execution, offering a fluent API for method chaining. The interface includes essential query modification methods such as `limit()` for restricting the number of returned documents, `skip()` for pagination, `sort()` for ordering results, `projection()` for selecting specific fields, and `hint()` for index selection. The batch, skip, and limit operations in `FindIterable` are essential pagination and data management tools that help control how documents are retrieved and processed from the database.

Batching (`batchSize`) controls how many documents DocumentDB returns to the client in a single network round trip. When you set a batch size, DocumentDB doesn't return all matching documents at once but instead returns them in groups of the specified batch size.

Skip allows you to offset the starting point of your results, essentially telling DocumentDB to skip over a specified number of documents before beginning to return matches. For example, `skip(20)` will bypass the first 20 matching documents. This is commonly used in pagination scenarios where you want to retrieve subsequent pages of results.

Limit restricts the total number of documents that can be returned from a query. When you specify `limit(n)`, DocumentDB will stop returning documents after it has returned 'n' documents, even if there are more matches in the database.

`FindIterable` supports both iterator and cursor patterns when retrieving documents from Amazon DocumentDB. The benefit of using `FindIterable` as iterator is that it allows lazy loading of documents and only fetches documents when requested by the application. Another benefit of using iterator is that you are not responsible for maintaining the connection to the cluster and thus no explicit closing of the connection is required.

`FindIterable` also provide support for `MongoCursor` which allows cursor patterns to be used when working with Amazon DocumentDB queries. `MongoCursor` is a MongoDB Java driver-specific implementation that provides control over database operations and resource management. It implements the `AutoCloseable` interface, allowing for explicit resource management through try-with-resources blocks, which is crucial for properly closing database connections and freeing server resources. By default, the cursor times out in 10 minutes and DocumentDB does not give you the option of changing this time out behavior. When working with batched data, make sure to retrieve the next batch of data before the cursor times out. One key consideration when using `MongoCursor` is that it requires explicit closing to prevent resource leaks.

In this section, several examples are presented for `find()`, `Filters` and `FindIterable`.

The following code example shows how to use `find()` to retrieve a single document using its "restaurantId" field:

```
Document filter = new Document("restaurantId", "REST-21G145");
Document result = collection.find(filter).first();
```

Even though using `Filters` allows for better compile time error checking, the java driver also allows you to specify a `Bson` filter directly in the `find()` method. The following example code passes `Bson` document to `find()`:

```
result = collection.find(new Document("$and", Arrays.asList(
    new Document("rating.totalReviews", new Document("$gt", 1000)),
    new Document("priceRange", "$$"))))
```

The next example code shows several examples of using the `Filters` class with `find()`:

```
FindIterable < Document > results;

// Exact match
results = collection.find(Filters.eq("name", "Thai Curry Palace"));

// Not equal
results = collection.find(Filters.ne("cuisine", "Thai"));

// find an element in an array
results = collection.find(Filters.in("features", Arrays.asList("Private Dining")));

// Greater than
```

```
results = collection.find(Filters.gt("rating.average", 3.5));

// Between (inclusive)
results = collection.find(Filters.and(
    Filters.gte("rating.totalReviews", 100),
    Filters.lte("rating.totalReviews", 200)));
// AND
results = collection.find(Filters.and(
    Filters.eq("cuisine", "Thai"),
    Filters.gt("rating.average", 4.5)));

// OR
results = collection.find(Filters.or(
    Filters.eq("cuisine", "Thai"),
    Filters.eq("cuisine", "American")));


// All document where the Field exists
results = collection.find(Filters.exists("michelin"));

// Regex
results = collection.find(Filters.regex("name", Pattern.compile("Curry",
 Pattern.CASE_INSENSITIVE)));

// Find all document where the array contain the list of value regardless of its order
results = collection.find(Filters.all("features", Arrays.asList("Private Dining",
 "Parking")));

// Array size
results = collection.find(Filters.size("features", 4));
```

The following example shows how to chain the operations of `sort()`, `skip()`, `limit()`, and `batchSize()` on a `FindIterable` object. The order of how these operations is provided will influence the performance of your query. As a best practice, the order of these operations should be `sort()`, `projection()`, `skip()`, `limit()` and `batchSize()`.

```
FindIterable < Document > results = collection.find(Filters.gt("rating.totalReviews",
 1000))
    // Sorting
    .sort(Sorts.orderBy(
        Sorts.descending("address.city"),
        Sorts.ascending("cuisine")))
    // Field selection
```

```
    .projection(Projections.fields(
        Projections.include("name", "cuisine", "priceRange"),
        Projections.excludeId()))

    // Pagination
    .skip(20)
    .limit(10)
    .batchSize(2);
```

The following example code shows creating an iterator on `FindIterable`. It uses Java's `forEach` construct to traverse through the result set.

```
collection.find(Filters.eq("cuisine", "American")).forEach(doc ->
 System.out.println(doc.toJson()));
```

In the last `find()` code example, it shows how to use `cursor()` for document retrieval. It creates the cursor in the try block which ensures that the cursor will get closed when the code exits the try block.

```
try (MongoCursor < Document > cursor = collection.find(Filters.eq("cuisine",
 "American"))
    .batchSize(25)
    .cursor()) {
    while (cursor.hasNext()) {
        Document doc = cursor.next();
        System.out.println(doc.toJson());
    }
} // Cursor automatically closed
```

Updating existing documents in a DocumentDB collection

Amazon DocumentDB provides flexible and powerful mechanisms for modifying existing documents and inserting new ones when they don't exist. The MongoDB Java driver offers several methods for updates: `updateOne()` for single document updates, `updateMany()` for multiple document updates, and `replaceOne()` for complete document replacement. Beside these three methods, `Updates`, `UpdateOptions`, and `UpdateResult` are other fundamental components that provide the building blocks for update operations in MongoDB Java drivers.

The `Updates` class in the MongoDB Java driver is a utility class that provides static factory methods for creating update operators. It serves as the primary builder for constructing update operations in a type-safe and readable manner. Basic methods like `set()`, `unset()`, and `inc()`

allows direct modification of the documents. The power of this class becomes evident when combining multiple operations using the `Updates.combine()` method which allows multiple update operations to be executed atomically, ensuring data consistency.

`UpdateOptions` is a powerful configuration class in MongoDB's Java driver that provides essential customization capabilities for document update operations. Two important aspects of this class is providing upsert and array filter support for update operations. The upsert feature, enabled through `upsert(true)`, allows for the creation of new documents when no matching documents are found during an update operation. Through `arrayFilters()`, the update operation can precisely update array elements that meet specific criteria.

`UpdateResult` in MongoDB's Java driver provides the feedback mechanism detailing the outcome of an update operation. This class encapsulates three key metrics: the number of documents matched by the update criteria (`matchedCount`), the number of documents actually modified (`modifiedCount`), and information about any upserted documents (`upsertedId`). Understanding these metrics is essential for proper error handling, verification of update operations, and maintaining data consistency in applications.

Update and replace a single document

In DocumentDB, updating a single document can be accomplished using the updateOne() method. This method take a filter parameter, usually provided by the `Filters` class, to identify the document to be updated, an `Update` parameter that determines which field(s) to be updated, and an optional `UpdateOptions` parameter to set different options for the update. Using the `updateOne()` method will only update the first document that matches the selection criteria. The following example code updates a single field of one document:

```
collection.updateOne(Filters.eq("restaurantId", "REST-Y2E9H5"),
    Updates.set("name", "Amazing Japanese sushi"));
```

To update multiple fields in one document, use `updateOne()` with `Update.combine()` as shown in the following example. This example also shows how to add an item to an array in the document.

```
List<Bson> updates = new ArrayList<>();
// Basic field updates
updates.add(Updates.set("name", "Shanghai Best"));
// Array operations
updates.add(Updates.addEachToSet("features", Arrays.asList("Live Music")));
// Counter updates
```

```
updates.add(Updates.inc("rating.totalReviews", 10));
// Combine all updates
Bson combinedUpdates = Updates.combine(updates);
// Execute automic update with one call
collection.updateOne(Filters.eq("restaurantId","REST-1J83NH"), combinedUpdates);
```

The following code example demonstrates how to update a document in the database. If the specified document doesn't exist, the operation will automatically insert it as a new document instead. This code also shows how to use the metrics available via the `UpdateResult` object.

```
Bson filter = Filters.eq("restaurantId", "REST-0Y9GL0");
Bson update = Updates.set("cuisine", "Indian");
// Upsert operation
UpdateOptions options = new UpdateOptions().upsert(true);
UpdateResult result = collection.updateOne(filter, update, options);

if (result.getUpsertedId() != null) {
    System.out.println("Inserted document with _id: " + result.getUpsertedId());
} else {
     System.out.println("Updated " + result.getModifiedCount() + " document(s)");
}
```

The following code example demonstrates how to completely replace an existing document with a new document using the `replaceOne()` method, rather than updating individual fields. The `replaceOne()` method overwrites the entire document, preserving only the `_id` field of the original. If multiple documents match the filter criteria, only the first encountered document is replaced.

```
Document newDocument = new Document()
                .append("restaurantId", "REST-0Y9GL0")
                .append("name", "Bhiryani Adda")
                .append("cuisine", "Indian")
                .append("rating", new Document()
                        .append("average", 4.8)
                        .append("totalReviews", 267))
                .append("features", Arrays.asList("Outdoor Seating", "Live Music"));

UpdateResult result = collection.replaceOne(
                    Filters.eq("restaurantId", "REST-0Y9GL0"),
                    newDocument);
System.out.printf("Modified %d document%n", result.getModifiedCount());
```

Update multiple documents

There are two ways to update multiple document in a collection simultaneously. You can use the `updateMany()` method or use the `UpdateManyModel` with the `bulkWrite()` method. The `updateMany()` method uses a filter parameter to select documents for update, the `Update` parameter to identify the fields to be updated, and an optional `UpdateOptions` parameter to specify update options.

The following example code demonstrates the usage of the `updateMany()` method:

```
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Thai"));
UpdateResult result1 = collection.updateMany(filter, Updates.set("priceRange", "$$$"));
```

The following example code demonstrates the `bulkWrite()` method using the same update:

```
BulkWriteOptions options = new BulkWriteOptions().ordered(false);
List < WriteModel < Document >> updates = new ArrayList < > ();
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Indian"));
Bson updateField = Updates.set("priceRange", "$$$");
updates.add(new UpdateManyModel < > (filter, updateField));
BulkWriteResult result = collection.bulkWrite(updates, options);
System.out.printf("Modified %d document%n", result.getModifiedCount());
```

Removing documents from a DocumentDB collection

The MongoDB Java driver offers `deleteOne()` for removing a single document and `deleteMany()` for removing multiple documents that match specific criteria. Just like update, delete operation can also be used with the `bulkWrite()` method. Both `deleteOne()` and `deleteMany()` return a `DeleteResult` object that provides information about the operation's outcome, including the count of documents deleted. The following is an example of using `deleteMany()` to remove multiple documents:

```
Bson filter = Filters.and(
    Filters.eq("cuisine", "Thai"),
    Filters.lt("rating.totalReviews", 50));
DeleteResult result = collection.deleteMany(filter);
```

```
System.out.printf("Deleted %d document%n", result.getDeletedCount());
```

Error handling with retry logic

A robust error handling strategy for Amazon DocumentDB should implement categorization of errors into retryable (like network timeouts, connection issues) and non-retryable (like authentication failures, invalid queries). For operation failures due of errors that should be retried, it should implement a time delay between each retry as well as max retry attempts. The CRUD operations should be in a try-catch block that catches `MongoException` and its subclasses. Additionally, it should include monitoring and logging of errors for operational visibility. The following is sample code that shows how to implement retry error handling:

```
int MAX_RETRIES = 3;
int INITIAL_DELAY_MS = 1000;
int retryCount = 0;

while (true) {
    try {
        crud_operation(); //perform crud that will throw MongoException or one of its
 subclass
        break;
    } catch (MongoException e) {
        if (retryCount < MAX_RETRIES) {
            retryCount++;
            long delayMs = INITIAL_DELAY_MS * (long) Math.pow(2, retryCount - 1);
            try {
                TimeUnit.MILLISECONDS.sleep(delayMs);
            } catch (InterruptedException t) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Retry interrupted", t);
            }
            continue;
        } else
            throw new RuntimeException("Crud operation failed", e);
    }
}
```

Index management in Amazon DocumentDB with Java

Indexes allow efficient retrieval of data from an Amazon DocumentDB collection. Without indexes, DocumentDB must scan every document in the collection to return results that satisfy a given query. This topic provides information on how to create, drop, and list indexes using the MongoDB

Java drivers. It also discusses how to determine if a particular index is being used in the query and how to give hints to Amazon DocumentDB to use a specific index.

Topics

- Creating indexes with Java
- Dropping indexes
- Determining index selection and providing index hint

Amazon DocumentDB support many types of indexes. For a comprehensive overview of all the supported indexes refer to this blog post.

Creating indexes with Java

There are two mechanisms for creating indexes in Amazon DocumentDB using MongoDB Java drivers: through `runCommand()`, and by either the `createIndex()` method for a single index or the `createIndexes()` method for multiple indexes. One reason for using the `createIndex()` and `createIndexes()` methods is that you can build better error handling by catching specific errors related to index creation. Another reason for using these method over `runCommand()` is that the MongDB Java driver provides a rich set of supporting classes for index creation and manipulation. Note that these supporting classes can only be used when you are using the `createIndex()` or `createIndexes()` methods. There are three supporting classes:

- **`Indexes`** — This class serves as a utility class offering static factory methods for creating various types of indexes. It simplifies the process of creating complex index definitions and is commonly used in conjunction with other index-related classes.
- **`IndexModel`** — This is a fundamental class that encapsulates both the index keys definition and its options. It represents a complete index specification, combining what to index (the keys) with how to index (the options). This class is particularly useful when creating multiple indexes simultaneously, as it allows you to define a collection of index specifications that can be passed to the `createIndexes()` method.
- **`IndexOptions`** — This is a comprehensive configuration class that provides a rich set of methods for customizing index behavior. It includes settings for unique indexes, sparse indexes, expiration time (TTL), and partial filter expressions. Through method chaining, you can configure multiple options like background index building and unique constraints.

Create single index

This example shows how to create a single index using the `createIndex()` method in the background. To understand background and foreground index creation, please refer Index build types. The following code example uses `IndexOptions` to create a unique index with a name of "unique_restaurantId_idx" in the background. This `IndexOptions` object is then passed to the `createIndex()` method.

```
collection.createIndex(
    Indexes.ascending("restaurantId"),
    new IndexOptions()
        .unique(true)
        .name("unique_restaurantId_idx")
        .background(true));
```

Create multiple indexes

This example creates multiple indexes using the `createIndexes()` method. It first builds the option for each index by using the `IndexModel` object and then passes a list of `IndexModel` objects to the `createIndexes()` method. The the following code example shows how to create a composite index by using the `Indexes` utility class. This class is also used to specify if you want to create an index using ascending or descending sort order. After creating multiple indexes, it verifies index creation by calling the `listIndexes()` method.

```
// Single Field Index on cuisine
IndexModel singleIndex = new IndexModel(
    Indexes.ascending("cuisine"),
    new IndexOptions().name("cuisine_idx"));

// Compound Index
IndexModel compoundIndex = new IndexModel(
    Indexes.compoundIndex(
        Indexes.ascending("address.state"),
        Indexes.ascending("priceRange")),
    new IndexOptions().name("location_price_idx"));

// Build a list of IndexModel for the indexes
List < IndexModel > indexes = Arrays.asList(
    singleIndex,
    compoundIndex
);

collection.createIndexes(indexes);
```

```
// Verify created indexes
collection.listIndexes().forEach(index - > System.out.println("Created index: " +
 index.toJson()));
```

Create sparse and partial indexes

This example shows creates a sparse and a partial index by creating an `IndexModel` for each type of index.

```
// Sparse Index Model, this will identify only those documents that have a
// michelin star rating
IndexModel sparseIndex = new IndexModel(
    Indexes.ascending("michelin.star"),
    new IndexOptions()
    .name("michelin_sparse_idx")
    .sparse(true));

// Partial Index Model where the restaurant is active and has a rating of 4 and above
IndexModel partialIndex = new IndexModel(
    Indexes.ascending("rating.average"),
    new IndexOptions()
    .name("high_rated_active_idx")
    .partialFilterExpression(
        Filters.and(
            Filters.eq("isActive", true),
            Filters.gte("rating.average", 4.0))));
```

Create a text index

This example shows how to create a text index. Only one text index is allowed on a collection but that one text index can be a compound index covering multiple fields. When using multiple fields in the text index, you can also assign weights to each of the fields in the index. Text indexes on array fields are not supported by Amazon DocumentDB and even though you can use up to 30 fields in the compound text index, only three fields can be assigned a weight.

```
IndexModel textIndex = new IndexModel(
    new Document()
    .append("name", "text")
    .append("description", "text")
    .append("cuisine", "text"),
    new IndexOptions()
```

```
    .name("restaurant_text_idx")
    .weights(new Document()
        .append("name", 10) // Restaurant name gets highest weight
        .append("description", 5) // Description get medium weight
        .append("cuisine", 2) // Cuisine type gets low weight
    ));

collection.createIndex(textIndex.getKeys(), textIndex.getOptions());
```

Create an index using `runCommand()`

Amazon DocumentDB supports parallel index creation to decrease the time it takes to create indexes. Parallel indexing uses multiple concurrent workers. The default workers used for index creation is two. This blog post provides an in-depth discussion on parallel indexing. Currently, MongDB Java drivers do not support specifying the worker option when you are using `createIndex()` or `createIndexes()` and therefore the only way to specify workers is through the `runCommand`. The following code example demonstrates how to use `runCommand` to create an index that increase the worker to four:

```
Document command = new Document("createIndexes", "Restaurants")
    .append("indexes", Arrays.asList(
        new Document("key", new Document("name", 1))
        .append("name", "restaurant_name_idx")
        .append("workers", 4) // Specify number of workers
    ));

Document commendResult = connectedDB.runCommand(command);
```

Dropping indexes

The MongoDB Java driver provides multiple methods to drop indexes, catering to different scenarios and your preferences. You can drop indexes by name, by key specification, or drop all indexes at once. The methods `dropIndex()` and `dropIndexes()` can be invoked on a collection object to drop an index. When dropping an index by name, you should ensure they use the correct index name which may not always be intuitive, especially for compound or auto-generated indexes. Attempting to drop a non-existent index will result in a `MongoCommandException`. The `default _id` index cannot be dropped as it ensures document uniqueness within the collection.

The following code example shows demonstrates how to drop an index by providing field name where the index is created or by deleting all the indexes:

```
String indexName = "unique_restaurantId_idx";
Document keys = new Document("cuisine", 1);
// Drop index by name
collection.dropIndex(indexName);

// Drop index by keys
collection.dropIndex(keys);

// Drop all indexes
collection.dropIndexes();
```

When dropping indexes using multiple keys, make sure there is a compound index containing all the keys specified and the order of the keys is correct. The above index creation example code shows a compound key on "cuisine" and features. If you try to drop that compound key but the order is not what was used for creation, then a MongoCommnadException error occurs as follows:

```
Document keys = new Document("features", 1)
    .append("cuisine", 1);
try {
    // Drop index by keys
    collection.dropIndex(keys);
    System.out.println("Successfully dropped index with keys: " + keys.toJson());

} catch (MongoCommandException commErr) {
    System.out.println("Error dropping index: " + commErr.getErrorMessage());
    throw new RuntimeException("MongoCommandException was thrown while dropping index",
 commErr);
}
```

The following error is displayed:

```
Error dropping index: Cannot drop index: index not found.
Tests run: 3, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.819 sec <<< FAILURE!
com.amazon.docdb.guide.DocDBGuideTest.testindexGuide()  Time elapsed: 0.817 sec  <<<
 FAILURE!
org.opentest4j.AssertionFailedError: Unexpected exception thrown:
 java.lang.RuntimeException: MongoCommandException was thrown while dropping index
```

Determining index selection and providing index hint

Working with the explain functionality in Amazon DocumentDB is crucial for you to understand query performance and index usage. When executing a query, you can append the `explain()` method to obtain detailed information about the query plan, including which indexes, if any, are being used. The `explain()` output provides insights into the query execution stages, the number of documents examined, and the time taken for each stage. This information is invaluable for identifying whether a particular index is being used effectively or if the query could benefit from a different index structure.

The `explain()` method can be chained with the `find()` method. The `explain()` method can take an optional `ExplainVerbosity` enum that determines the level of verbosity that is returned by `explain()`. At this time, only the `EXECUTION_STATS` and `QUERY_PLANNER` enumerators are supported by DocumentDB. The following code example shows how to get query planner for a specific query:

```
// Query we want to analyze
Document query = new Document()
    .append("cuisine", "Thai")
    .append("rating.average", new Document("$gte", 4.0));


Document allPlansExplain =
 collection.find(query).explain(ExplainVerbosity.QUERY_PLANNER);
System.out.println("All Plans Explain:\n" + allPlansExplain.toJson());
```

The following JSON document is returned for the query planner verbosity level:

```
{
  "queryPlanner": {
    "plannerVersion": 1,
    "namespace": "ProgGuideData.Restaurants",
    "winningPlan": {
      "stage": "IXSCAN",
      "indexName": "cuisine_idx",
      "direction": "forward"
    }
  },
  "serverInfo": {
    "host": "guidecluster3",
    "port": 27017,
```

```
    "version": "5.0.0"
  },
  "ok": 1,
  "operationTime": {
    "$timestamp": {
      "t": 1739221668,
      "i": 1
    }
  }
}
```

You have several options to influence or force Amazon DocumentDB to use a specific index. The `hint()` and `hintString()` methods allow you to override the query optimizer's default index selection behavior by explicitly specifying which index should be used for a query. While DocumentDB's query optimizer generally makes good choices for index selection, there are scenarios where forcing a specific index through `hint()` or `hintString()` can be beneficial, such as when dealing with skewed data, or testing index performance.

The following code example forces the use of compound index “cuisine_features_idx” for the same query that was run in the above code:

```
// Query we want to analyze
Document query = new Document()
    .append("cuisine", "Thai")
    .append("rating.average", new Document("$gte", 4.0));

List < Document > queryDocs = new ArrayList < > ();
collection.find(query).hintString("cuisine_features_idx").forEach(doc - >
 queryDocs.add(doc));
```

Event-driven programming with Amazon DocumentDB and Java

Event-driven programming in the context of Amazon DocumentDB represents a powerful architectural pattern where database changes serve as the primary event generators that trigger subsequent business logic and processes. When records are inserted, updated, or deleted in a DocumentDB collection, these changes act as events that automatically initiate various downstream processes, notifications, or data synchronization tasks. This pattern is particularly valuable in modern distributed systems where multiple applications or services need to react to data changes in real-time. The primary mechanism of implementing event-driven programming in DocumentDB is by change streams.

> **Note**
>
> This guide assumes you have enabled change streams on a collection that you are working with. See Using change streams with Amazon DocumentDB to learn how to enable change streams on the collection.

Working with change streams from the Java application

The `watch()` method in MongoDB's Java driver is the primary mechanism for monitoring real-time data changes in Amazon DocumentDB. The `watch()` method can be called on by `MongoClient`, `MongoDatabase`, and `MongoCollection` objects.

The `watch()` method returns an instance of `ChangeStreamIterable` that supports various configuration options, including full document lookup for updates, providing resume tokens and timestamp for reliability, and pipeline aggregation stages for filtering changes.

`ChangeStreamIterable` implements the core Java interface `Iterable` and can be used with `forEach()`. To capture events using `forEach()`, pass in a callback function to `forEach()` that processes the changed event. The following code snippet shows how to open a change streams on a collection to start change event monitoring:

```
ChangeStreamIterable < Document > iterator = collection.watch();
iterator.forEach(event - > {
    System.out.println("Received a change: " + event);
});
```

Another way of traversing through all the change events is by opening a cursor that maintains a connection to the cluster and continuously receives new change events as they occur. To obtain a change streams cursor, use the `cursor()` method of `ChangeStreamIterable` object. The following code example shows how to monitor change events using cursor:

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println(cursor.tryNext());
}
```

As a best practice, either create the `MongoChangeStreamCursor` in a try-with-resource statement or manually close the cursor. Calling the `cursor()` method on `ChangeStreamIterable` returns a `MongoChangeStreamCursor` that is created over a `ChangeStreamDocument` object.

The `ChangeStreamDocument` class is a crucial component that represents individual change events in the stream. It contains detailed information about each modification, including the operation type (insert, update, delete, replace), the document key, namespace information, and the full document content when available. The class provides methods to access various aspects of the change event, such as `getOperationType()` to determine the type of change, `getFullDocument()` to access the complete document state, and `getDocumentKey()` to identify the modified document.

The `ChangeStreamDocument` object provides two important pieces of information, a resume token and time of the change event.

Resume tokens and time-based operations in DocumentDB change streams provide crucial mechanisms for maintaining continuity and managing historical data access. A resume token is a unique identifier generated for each change event, serving as a bookmark that allows applications to restart change stream processing from a specific point after disconnections or failures. When a change stream cursor is created, it can use a previously stored resume token through the `resumeAfter()` option, enabling the stream to continue from where it left off rather than starting from the beginning or losing events.

Time-based operations in change streams offer different approaches to manage the starting point of change event monitoring. The `startAtOperationTime()` option allows you to begin watching changes that occurred at or after a specific timestamp. These time-based features are particularly valuable in scenarios requiring historical data processing, point-in-time recovery, or synchronization between systems.

The following code example retrieves the event associated with the insert document, captures it's resume token, and then provides that token to start monitoring for events after the insert event. The event is associated with the update event, then gets the cluster time when the update happened and uses that timestamp as a starting point for further processing.

```
BsonDocument resumeToken;
BsonTimestamp resumeTime;

try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println("****************** Insert Document *******************");
```

```
    ChangeStreamDocument < Document > insertChange = cursor.tryNext();
    resumeToken = insertChange.getResumeToken();
    printJson(cursor.tryNext());
}
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch()
    .resumeAfter(resumeToken)
    .cursor()) {
    System.out.println("****************** Update Document *******************");
    ChangeStreamDocument < Document > insertChange = cursor.tryNext();
    resumeTime = insertChange.getClusterTime();
    printJson(cursor.tryNext());
}
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch()
    .startAtOperationTime(resumeTime)
    .cursor()) {
    System.out.println("****************** Delete Document *******************");
    printJson(cursor.tryNext());
  }
```

By default, the update change event does not include the full document and it only include the changes that were made. If you need to access the complete document that was updated, you can call the `fullDocument()` method on the `ChangeStreamIterable` object. Keep in mind that when you ask for a full document to be returned for an update event, it returns the document that exists at the time the call to change streams is made.

This method takes a `FullDocument` enum as a parameter. Currently, Amazon DocumentDB only support DEFAULT and UPDATE_LOOKUP values. The following code snippet shows how to ask for full document for update events when starting to watch for changes:

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().fullDocument(FullDocument.UPDATE_LOOKUP).cursor())
```

Using JSON schema validation

Using the `$jsonSchema` evaluation query operator, you can validate documents being inserted into your collections.

Topics

- Creating and using JSON schema validation

- Supported keywords
- bypassDocumentValidation
- Limitations

Creating and using JSON schema validation

Creating a collection with schema validation

You can create a collection with `createCollection` operation and validation rules. These validation rules are applied during inserts or updates of Amazon DocumentDB documents. The following code example shows validation rules for a collection of employees:

```
db.createCollection("employees", {
   "validator": {
      "$jsonSchema": {
         "bsonType": "object",
         "title": "employee validation",
         "required": [ "name", "employeeId"],
         "properties": {
            "name": {
                  "bsonType": "object",
                  "properties": {
                     "firstName": {
                        "bsonType": ["string"]
                     },
                     "lastName": {
                        "bsonType": ["string"]
                     }
                  },
                  "additionalProperties" : false
            },
            "employeeId": {
               "bsonType": "string",
               "description": "Unique Identifier for employee"
            },
             "salary": {
               "bsonType": "double"
            },
            "age": {
               "bsonType": "number"
            }
         },
```

```
         "additionalProperties" : true
      }
   },
   "validationLevel": "strict", "validationAction": "error"
} )
```

Inserting a valid document

The following example inserts documents that comply with the above schema validation rules:

```
db.employees.insert({"name" : { "firstName" : "Carol" , "lastName" : "Smith"},
 "employeeId": "c720a" , "salary": 1000.0 })
db.employees.insert({ "name" : { "firstName" : "William", "lastName" : "Taylor" },
 "employeeId" : "c721a", "age" : 24})
```

Inserting an invalid document

The following example inserts documents that do not comply with the above schema validation rules. In this example, the employeeId value is not a string:

```
db.employees.insert({
    "name" : { "firstName" : "Carol" , "lastName" : "Smith"},
    "employeeId": 720 ,
    "salary": 1000.0
})
```

This example shows incorrect syntax within the document.

Modifying a collection

The `collMod` command is used to add or modify validation rules of existing collection. The following example adds a salary field to the required field list:

```
db.runCommand({"collMod" : "employees",
   "validator": {
      "$jsonSchema": {
         "bsonType": "object",
         "title": "employee validation",
         "required": [ "name", "employeeId", "salary"],
         "properties": {
            "name": {
                  "bsonType": "object",
```

```
                  "properties": {
                     "firstName": {
                        "bsonType": ["string"]
                     },
                     "lastName": {
                        "bsonType": ["string"]
                     }
                  },
                  "additionalProperties" : false
            },
            "employeeId": {
               "bsonType": "string",
               "description": "Unique Identifier for employee"
            },
             "salary": {
               "bsonType": "double"
            },
            "age": {
               "bsonType": "number"
            }
         },
         "additionalProperties" : true
      }
   }
} )
```

Addressing documents added before the validation rules were changed

To address documents that were added to you collection before the validation rules were changed, use the following `validationLevel` modifiers:

- **strict**: Applies validation rules on all inserts and updates.
- **moderate**: Applies validation rules to existing valid documents. During updates, existing invalid documents are not checked.

In the following example, after updating the validation rules on the collection named “employees”, the salary field is required. Updating following document will fail:

```
db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
```

```
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB returns the following output:

```
{
"n" : 0,
    "nModified" : 0,
    "writeErrors" : [
        {
"index" : 0,
            "code" : 121,
            "errmsg" : "Document failed validation"
        }
    ],
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

Updating the validation level to `moderate` will allow the above document to be updated successfully:

```
db.runCommand({
    "collMod" : "employees",
    validationLevel : "moderate"
})

db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB returns the following output:

```
{
"n" : 1,
    "nModified" : 1,
    "ok" : 1,
```

```
    "operationTime" : Timestamp(1234567890, 1)
}
```

Retrieving documents with the $jsonSchema

The `$jsonSchema` operator can be used as a filter to query documents that match the JSON schema. This is a top-level operator which can be present in filter documents as a top level field or used with query operators such as `$and`, `$or`, and `$nor`. The following examples show the use of $jsonSchema as an individual filter and with other filter operators:

Document inserted into an "employee" collection:

```
{ "name" : { "firstName" : "Carol", "lastName" : "Smith" }, "employeeId" : "c720a",
 "salary" : 1000 }
{ "name" : { "firstName" : "Emily", "lastName" : "Brown" }, "employeeId" : "c720b",
 "age" : 25, "salary" : 1050.2 }
{ "name" : { "firstName" : "William", "lastName" : "Taylor" }, "employeeId" : "c721a",
 "age" : 24, "salary" : 1400.5 }
{ "name" : { "firstName" : "Jane", "lastName" : "Doe" }, "employeeId" : "c721a",
 "salary" : 1300 }
```

Collection filtered with the `$jsonSchema` operator only:

```
db.employees.find({
       $jsonSchema: { required: ["age"] } })
```

Amazon DocumentDB returns the following output:

```
{ "_id" : ObjectId("64e5f91c6218c620cf0e8f8b"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "c720b", "age" : 25, "salary" : 1050.2 }
{ "_id" : ObjectId("64e5f94e6218c620cf0e8f8c"), "name" : { "firstName" : "William",
 "lastName" : "Taylor" }, "employeeId" : "c721a", "age" : 24, "salary" : 1400.5 }
```

Collection filtered with the `$jsonSchema` operator and another operator:

```
db.employees.find({
       $or: [{ $jsonSchema: { required: ["age", "name"]}},
            { salary: { $lte:1000}}]});
```

Amazon DocumentDB returns the following output:

```
{ "_id" : ObjectId("64e5f8886218c620cf0e8f8a"), "name" : { "firstName" : "Carol",
 "lastName" : "Smith" }, "employeeId" : "c720a", "salary" : 1000 }
{ "_id" : ObjectId("64e5f91c6218c620cf0e8f8b"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "c720b", "age" : 25, "salary" : 1050.2 }
{ "_id" : ObjectId("64e5f94e6218c620cf0e8f8c"), "name" : { "firstName" : "William",
 "lastName" : "Taylor" }, "employeeId" : "c721a", "age" : 24, "salary" : 1400.5 }
```

Collection filtered with the `$jsonSchema` operator and with `$match` in the aggregate filter:

```
db.employees.aggregate(
    [{ $match: {
        $jsonSchema: {
            required: ["name", "employeeId"],
            properties: {"salary" :{"bsonType": "double"}}
        }
       }
    }]
)
```

Amazon DocumentDB returns the following output:

```
{
"_id" : ObjectId("64e5f8886218c620cf0e8f8a"),
 "name" : { "firstName" : "Carol", "lastName" : "Smith" },
"employeeId" : "c720a",
"salary" : 1000
}
{
"_id" : ObjectId("64e5f91c6218c620cf0e8f8b"),
"name" : { "firstName" : "Emily", "lastName" : "Brown" },
"employeeId" : "c720b",
"age" : 25,
"salary" : 1050.2
}
{
"_id" : ObjectId("64e5f94e6218c620cf0e8f8c"),
"name" : { "firstName" : "William", "lastName" : "Taylor" },
"employeeId" : "c721a",
"age" : 24,
"salary" : 1400.5
}
{
```

```
"_id" : ObjectId("64e5f9786218c620cf0e8f8d"),
"name" : { "firstName" : "Jane", "lastName" : "Doe" },
"employeeId" : "c721a",
"salary" : 1300
}
```

Viewing existing validation rules

To see the existing validation rules on a collection, use:

```
db.runCommand({
    listCollections: 1,
    filter: { name: 'employees' }
})
```

Amazon DocumentDB returns the following output:

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "name" : "employees",
                "type" : "collection",
                "options" : {
                    "autoIndexId" : true,
                    "capped" : false,
                    "validator" : {
                        "$jsonSchema" : {
                            "bsonType" : "object",
                            "title" : "employee validation",
                            "required" : [
                                "name",
                                "employeeId",
                                "salary"
                            ],
                            "properties" : {
                                "name" : {
                                    "bsonType" : "object",
                                    "properties" : {
                                        "firstName" : {
                                            "bsonType" : [
                                                "string"
```

```
                                        ]
                                    },
                                    "lastName" : {
                                        "bsonType" : [
                                            "string"
                                        ]
                                    }
                                },
                                "additionalProperties" : false
                            },
                            "employeeId" : {
                                "bsonType" : "string",
                                "description" : "Unique Identifier for employee"
                            },
                            "salary" : {
                                "bsonType" : "double"
                            },
                            "age" : {
                                "bsonType" : "number"
                            }
                        },
                        "additionalProperties" : true
                    }
                },
                "validationLevel" : "moderate",
                "validationAction" : "error"
            },
            "info" : {
                "readOnly" : false
            },
            "idIndex" : {
                "v" : 2,
                "key" : {
                    "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.employees"
            }
        }
    ],
    "id" : NumberLong(0),
    "ns" : "test.$cmd.listCollections"
},
"ok" : 1,
```

```
    "operationTime" : Timestamp(1692788937, 1)
}
```

Amazon DocumentDB also retains validation rules in the `$out` aggregation stage.

Supported keywords

The following fields are supported in the `create` and `collMod` commands:

- **`Validator`** — Supports the `$jsonSchema` operator.
- **`ValidationLevel`** — Supports `off`, `strict`, and `moderate` values.
- **`ValidationAction`** — Supports the `error` value.

The $jsonSchema operator supports the following keywords:

- `additionalItems`
- `additionalProperties`
- `allOf`
- `anyOf`
- `bsonType`
- `dependencies`
- `description`
- `enum`
- `exclusiveMaximum`
- `exclusiveMinimum`
- `items`
- `maximum`
- `minimum`
- `maxItems`
- `minItems`
- `maxLength`
- `minLength`
- `maxProperties`
- `minProperties`

- `multipleOf`
- `not`
- `oneOf`
- `pattern`
- `patternProperties`
- `properties`
- `required`
- `title`
- `type`
- `uniqueItems`

bypassDocumentValidation

Amazon DocumentDB supports `bypassDocumentValidation` for the following commands and methods:

- `insert`
- `update`
- `findAndModify`
- `$out` stage in `aggregate` command and in `db.collection.aggregate()` method

Amazon DocumentDB does not support the following commands for `bypassDocumentValidation`:

- `$merge` in `aggregate` command and in `db.collection.aggregate()` method
- `mapReduce` command and `db.collection.mapReduce()` method
- `applyOps` command

Limitations

The following limitations apply to `$jsonSchema` validation:

- Amazon DocumentDB returns the error "Document failed validation" when an operation fails the validation rule.

- Amazon DocumentDB elastic clusters do not support `$jsonSchema`.

Amazon DocumentDB Quotas and limits

This topic describes the resource quotas, limits, and naming constraints for Amazon DocumentDB (with MongoDB compatibility).

For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS) and Amazon Neptune.

Topics

- Supported instance types
- Supported regions
- Regional quotas
- Aggregation limits
- Cluster limits
- Instance limits
- Naming constraints
- TTL constraints
- Elastic cluster limits
- Elastic cluster shard limits
- Elastic cluster CPU, memory, connection, and cursor limits per shard

Supported instance types

Amazon DocumentDB supports on-demand instances and the following instance types:

- NVMe-backed:
  - **R6GD instance types:** `db.r6gd.xlarge`, `db.r6gd.2xlarge`, `db.r6gd.4xlarge`, `db.r6gd.8xlarge`, `db.r6gd.12xlarge`, `db.r6gd.16xlarge`.
- Memory Optimized:
  - **R6G instance types:** `db.r6g.large`, `db.r6g.2xlarge`, `db.r6g.4xlarge`, `db.r6g.8xlarge`, `db.r6g.12xlarge`, `db.r6g.16xlarge`.
  - **R5 instance types:** `db.r5.large`, `db.r5.2xlarge`, `db.r5.4xlarge`, `db.r5.8xlarge`, `db.r5.12xlarge`, `db.r5.16xlarge` `db.r5.24xlarge`.

- **R4 instance types:** `db.r4.large`, `db.r4.2xlarge`, `db.r4.4xlarge`, `db.r4.8xlarge`, `db.r4.16xlarge`.
- Burstable Performance:
  - **T4G instance types:** `db.t4g.medium`.
  - **T3 instance types:** `db.t3.medium`.

For more information on the supported instance types and their specifications, see Instance class specifications.

Supported regions

Amazon DocumentDB is available in the following AWS regions:

Region Name	Region	Availability Zones (compute)
US East (Ohio)	`us-east-2`	3
US East (N. Virginia)	`us-east-1`	6
US West (Oregon)	`us-west-2`	4
Africa (Cape Town)	`af-south-1`	3
South America (São Paulo)	`sa-east-1`	3
Asia Pacific (Hong Kong)	`ap-east-1`	3
Asia Pacific (Hyderabad)	`ap-south-2`	3
Asia Pacific (Mumbai)	`ap-south-1`	3
Asia Pacific (Seoul)	`ap-northeast-2`	4
Asia Pacific (Singapore)	`ap-southeast-1`	3
Asia Pacific (Sydney)	`ap-southeast-2`	3
Asia Pacific (Tokyo)	`ap-northeast-1`	3
Canada (Central)	`ca-central-1`	3
China (Beijing) Region	`cn-north-1`	3
China (Ningxia)	`cn-northwest-1`	3
Europe (Frankfurt)	`eu-central-1`	3

Region Name	Region	Availability Zones (compute)
Europe (Ireland)	`eu-west-1`	3
Europe (London)	`eu-west-2`	3
Europe (Milan)	`eu-south-1`	3
Europe (Paris)	`eu-west-3`	3
Europe (Spain)	`eu-south-2`	3
Europe (Stockholm)	`eu-north-1`	3
Middle East (UAE)	`me-central-1`	3
Israel (Tel Aviv)	`il-central-1`	3
AWS GovCloud (US-West)	`us-gov-west-1`	3
AWS GovCloud (US-East)	`us-gov-east-1`	3

Regional quotas

For certain management features, Amazon DocumentDB uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS). The following table contains regional limits that are shared among Amazon DocumentDB and Amazon RDS.

> **Note**
>
> The Amazon RDS shared technology described above only applies to Amazon DocumentDB instance-based clusters. Amazon DocumentDB elastic clusters do not share technology with Amazon RDS.

The following limits apply to Amazon DocumentDB instance-based clusters and are per AWS account per region.

Resource	AWS default limit
Clusters	40
Cluster parameter groups	50
Event subscriptions	20
Instances	40
Manual cluster snapshots	100
Read replicas per cluster	15
Subnet groups	50
Subnets per subnet group	20
Tags per resource	50
VPC security groups per instance	5

The following limits apply to Amazon DocumentDB elastic clusters and are per AWS account per region.

Resource	AWS default limit
Elastic clusters	20
Elastic clusters vCPU	1024
Manual elastic cluster snapshot	20

You can use Service Quotas to request an increase for a quota, if the quota is adjustable. Some requests are automatically resolved, while others are submitted to Support. You can track the status of a quota increase request that is submitted to Support. Requests to increase service quotas do not receive priority support. If you have an urgent request, please contact Support. For more information on Service Quotas, see What Is Service Quotas?

To request a quota increase for Amazon DocumentDB:

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas and, if necessary, sign in.
2. In the navigation pane, choose **AWS services**.
3. Select Amazon DocumentDB (with MongoDB compatibility) or Amazon DocumentDB Elastic Cluster from the list, or type either in the search field.
4. If the quota is adjustable, you can select its radio button or its name, and then choose **Request quota increase** from the top right of the page.
5. For **Change quota value**, enter the new value. The new value must be greater than the current value.
6. Choose **Request**. After the request is resolved, the **Applied quota value** for the quota is set to the new value.
7. To view any pending or recently resolved requests, choose **Dashboard** from the navigation pane. For pending requests, choose the status of the request to open the request receipt. The initial status of a request is `Pending`. After the status changes to `Quota requested`, you'll see the case number with Support. Choose the case number to open the ticket for your request.

Aggregation limits

The following table describes aggregation limits in Amazon DocumentDB.

Resource	Limit
Maximum number of supported stages	500

Cluster limits

The following table describes Amazon DocumentDB instance-based cluster limits.

Resource	Limit
Cluster size (sum of all collections and indexes)	128 TiB
Collection size (sum of all collections can't exceed cluster limit) – does not include the index size	32 TiB
Collections per cluster	100,000
Databases per cluster	100,000
Database size (sum of all databases can't exceed cluster limit)	128 TiB
Document nesting depth	200 levels
Document size	16 MiB
Index key size	2,048 bytes
Indexes per collection	64
Keys in a compound index	32
Maximum number of writes in a single batch command	100,000
Number of users per cluster	1000

Instance limits

The following table describes Amazon DocumentDB limits per instance.

Instance Type	Instance Memory (GiB)	Connections (all)	Cursor Limit	Open Transactions	Connections (active)
T3.medium	4	1000	30	50	102
T4G.medium	4	1000	30	50	102
R4.large	15.25	1700	450	N/A	1100
R4.xlarge	30.5	3400	450	N/A	2700
R4.2xlarge	61	6800	450	N/A	4500
R4.4xlarge	122	13600	725	N/A	4500
R4.8xlarge	288	27200	1450	N/A	4500
R4.16xlarge	488	30000	2900	N/A	4500
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R5.24xlarge	768	60000	4560	9600	4500
R6G.large*	16	3400	450	200	1100

Instance Type	Instance Memory (GiB)	Connections (all)	Cursor Limit	Open Transactions	Connections (active)
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500

* including R6GD

You can monitor and alarm on the per instance limits using the following CloudWatch metrics. For more on Amazon DocumentDB CloudWatch metrics, see Monitoring Amazon DocumentDB with CloudWatch.

Limit	CloudWatch Metrics
Instance Memory	FreeableMemory
Connections	DatabaseConnectionsMax
Cursors	DatabaseCursorsMax
Transactions	TransactionsOpenMax

Naming constraints

The following table describes naming constraints in Amazon DocumentDB.

Resource	Default Limit
Cluster identifier	• Length is [1–63] letters, numbers, or hyphens. • First character must be a letter. • Cannot end with a hyphen or contain two consecutive hyphens. • Must be unique for all clusters (across Amazon RDS, Amazon Neptune, and Amazon DocumentDB) per AWS account, per Region.
Collection name: `<col>`	Length is [1–57] characters.
Database name: `<db>`	Length is [1–63] characters.
Fully qualified collection name: `<db>.<col>`	Length is [3–120] characters.
Fully qualified index name: `<db>.<col>$<index>`	Length is [6–377] characters.
Index name: `<col>$<index>`	Length is [3–255] characters.
Instance identifier	• Length is [1–63] letters, numbers, or hyphens • First character must be a letter • Cannot end with a hyphen or contain two consecutive hyphens • Must be unique for all instances (across Amazon RDS, Amazon Neptune, and Amazon

Resource	Default Limit
	DocumentDB) per AWS account, per Region.

Primary password	• Length is [8-100] printable ASCII characters.
	• Can use any printable ASCII characters except for the following:
	  • **/** (forward slash)
	  • **"** (double quotation mark)
	  • **@** (at symbol)

Primary user name	• Length is [1-63] alphanumeric characters. • First character must be a letter. • Cannot be a word reserved by the database engine.
Parameter group name	• Length is [1–255] alphanumeric characters. • First character must be a letter. • Cannot end with a hyphen or contain two consecutive hyphens.

TTL constraints

Deletes from a TTL index are not guaranteed within a specific timeframe and are best effort. Factors like instance resource utilization, document size, and overall throughput can affect the timing of a TTL delete.

Elastic cluster limits

The following table describes maximum limits in Amazon DocumentDB elastic clusters.

Resource	Limit
Elastic clusters per region	20
vCPU summed across all elastic clusters per region	1024
Manual cluster snapshots per region	20
Shards per cluster	32
Storage per cluster (when data is evenly distributed by shard-key)	4 PiB
Connections to cluster	The lower value of either 300,000 or the number of shards x the connection limit associated with vCPU per shard
UnSharded collection size	32 TiB
Sharded collection size (when data is evenly distributed by shard-key)	1PB
Databases per cluster	10,000
UnSharded collections per cluster	100,000
Sharded collections per cluster	1000
Users per cluster	100
Writes in a single batch command	100,000
Indexes per collection	64
Document nesting depth	100 levels

Resource	Limit
Document size	16MB
Index key size	2048 bytes
Keys in a compound index	32

Elastic cluster shard limits

The following table describes maximum shard limits in Amazon DocumentDB elastic clusters.

Resource	Limit
vCPU per shard instance	64
Instances per shard	16
Storage per shard	128 TiB
Storage per collection per shard	32 TiB

Elastic cluster CPU, memory, connection, and cursor limits per shard

The following table describes maximum CPU, memory, connection, and cursor limits in Amazon DocumentDB elastic cluster shards.

vCPUs per shard	Instance memory (GiB)	Connection limit	Cursor limit
2	16	1700	450
4	32	3500	450
8	64	7100	450

vCPUs per shard	Instance memory (GiB)	Connection limit	Cursor limit
16	128	14200	760
32	256	28400	1520
48	384	30000	2280
64	512	30000	3040

Querying in Amazon DocumentDB

This section explains all aspects of querying with Amazon DocumentDB.

Topics

- Querying documents
- Query plan
- Explain results
- Querying geospatial data with Amazon DocumentDB
- Partial index
- Performing text search with Amazon DocumentDB

Querying documents

At times, you might need to look up your online store's inventory so that customers can see and purchase what you're selling. Querying a collection is relatively easy, whether you want all documents in the collection or only those documents that satisfy a particular criterion.

To query for documents, use the `find()` operation. The `find()` command has a single document parameter that defines the criteria to use in choosing the documents to return. The output from `find()` is a document formatted as a single line of text with no line breaks. To format the output document for easier reading, use `find().pretty()`. All the examples in this topic use `.pretty()` to format the output.

The following code samples use the four documents you inserted into the `example` collection in the preceding two exercises — `insertOne()` and `insertMany()` that are located in the Adding Documents section of Working with Documents.

Topics

- Retrieving all documents in a collection
- Retrieving documents that match a field Value
- Retrieving documents that match an embedded document
- Retrieving documents that match a field value in an embedded document

- Retrieving documents that match an array
- Retrieving documents that match a value in an array
- Retrieving documents using operators

Retrieving all documents in a collection

To retrieve all the documents in your collection, use the `find()` operation with an empty query document.

The following query returns all documents in the `example` collection.

```
db.example.find( {} ).pretty()
```

Retrieving documents that match a field Value

To retrieve all documents that match a field and value, use the `find()` operation with a query document that identifies the fields and values to match.

Using the preceding documents, this query returns all documents where the "Item" field equals "Pen".

```
db.example.find( { "Item": "Pen" } ).pretty()
```

Retrieving documents that match an embedded document

To find all the documents that match an embedded document, use the `find()` operation with a query document that specifies the embedded document name and all the fields and values for that embedded document.

When matching an embedded document, the document's embedded document must have the same name as in the query. In addition, the fields and values in the embedded document must match the query.

The following query returns only the "Poster Paint" document. This is because the "Pen" has different values for "`OnHand`" and "`MinOnHand`", and "Spray Paint" has one more field (`OrderQnty`) than the query document.

```
db.example.find({"Inventory": {
    "OnHand": 47,
    "MinOnHand": 50 } } ).pretty()
```

Retrieving documents that match a field value in an embedded document

To find all the documents that match an embedded document, use the `find()` operation with a query document that specifies the embedded document name and all the fields and values for that embedded document.

Given the preceding documents, the following query uses "dot notation" to specify the embedded document and fields of interest. Any document that matches these are returned, regardless of what other fields might be present in the embedded document. The query returns "Poster Paint" and "Spray Paint" because they both match the specified fields and values.

```
db.example.find({"Inventory.OnHand": 47, "Inventory.MinOnHand": 50 }).pretty()
```

Retrieving documents that match an array

To find all documents that match an array, use the `find()` operation with the array name that you are interested in and all the values in that array. The query returns all documents that have an array with that name in which the array values are identical to and in the same order as in the query.

The following query returns only the "Pen" because the "Poster Paint" has an additional color (White), and "Spray Paint" has the colors in a different order.

```
db.example.find( { "Colors": ["Red","Green","Blue","Black"] } ).pretty()
```

Retrieving documents that match a value in an array

To find all the documents that have a particular array value, use the `find()` operation with the array name and the value that you're interested in.

```
db.example.find( { "Colors": "Red" } ).pretty()
```

The preceding operation returns all three documents because each of them has an array named `Colors` and the value "`Red`" somewhere in the array. If you specify the value "`White`," the query would only return "Poster Paint."

Retrieving documents using operators

The following query returns all documents where the "`Inventory.OnHand`" value is less than 50.

```
db.example.find(
        { "Inventory.OnHand": { $lt: 50 } } )
```

For a listing of supported query operators, see Query and projection operators.

Query plan

How Can I See the `executionStats` for a Query Plan?

When determining why a query is executing slower than expected, it can be useful to understand what the `executionStats` are for the query plan. The `executionStats` provide the number of documents returned from a particular stage (`nReturned`), the amount of execution time spent at each stage (`executionTimeMillisEstimate`), and the amount of time it takes to generate a query plan (`planningTimeMillis`). You can determine the most time-intensive stages of your query to help focus your optimization efforts from the output of `executionStats`, as shown in the query examples below. The `executionStats` parameter does not currently support `update` and `delete` commands.

> **Note**
>
> Amazon DocumentDB emulates the MongoDB 3.6 API on a purpose-built database engine that utilizes a distributed, fault-tolerant, self-healing storage system. As a result, query plans and the output of `explain()` may differ between Amazon DocumentDB and MongoDB. Customers who want control over their query plan can use the `$hint` operator to enforce selection of a preferred index.

Run the query that you want to improve under the `explain()` command as follows.

```
db.runCommand({explain: {query document}}).
```

```
explain("executionStats").executionStats;
```

The following is an example operation.

```
db.fish.find({}).limit(2).explain("executionStats");
```

Output from this operation looks something like the following.

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.fish",
        "winningPlan" : {
            "stage" : "SUBSCAN",
            "inputStage" : {
                "stage" : "LIMIT_SKIP",
                "inputStage" : {
                    "stage" : "COLLSCAN"
                }
            }
        }
    },
    "executionStats" : {
        "executionSuccess" : true,
        "executionTimeMillis" : "0.063",
        "planningTimeMillis" : "0.040",
        "executionStages" : {
            "stage" : "SUBSCAN",
            "nReturned" : "2",
            "executionTimeMillisEstimate" : "0.012",
            "inputStage" : {
                "stage" : "LIMIT_SKIP",
                "nReturned" : "2",
                "executionTimeMillisEstimate" : "0.005",
                "inputStage" : {
                    "stage" : "COLLSCAN",
                    "nReturned" : "2",
                    "executionTimeMillisEstimate" : "0.005"
                }
            }
        }
    },
    "serverInfo" : {
```

```
        "host" : "enginedemo",
        "port" : 27017,
        "version" : "3.6.0"
    },
    "ok" : 1
}
```

If you are interested in seeing only the `executionStats` from the query above, you can use the following command. For small collections, the Amazon DocumentDB query processor can choose to not use an index if the performance gains are negligible.

```
db.fish.find({}).limit(2).explain("executionStats").executionStats;
```

Query plan cache

In order to optimize performance and reduce planning duration, Amazon DocumentDB internally caches query plans. This enables queries with the same shape to be executed directly using a cached plan.

However, this caching may sometimes cause a random delay for the same query; for example, a query that typically takes one second to run may occasionally take ten seconds. This is because over time, the reader instance cached various shapes of the query, thus consuming memory. If you experience this random slowness, there is no action needed you need to do to release the memory--the system will manage the memory usage for you and once the memory reaches certain threshold, it will be automatically released.

Explain results

If you want to return information on query plans, Amazon DocumentDB supports verbosity mode `queryPlanner`. The `explain` results return the selected query plan chosen by the optimizer in a format similar to the following:

```
{
   "queryPlanner" : {
      "plannerVersion" : <int>,
      "namespace" : <string>,
      "winningPlan" : {
         "stage" : <STAGE1>,
```

```
            ...
            "inputStage" : {
               "stage" : <STAGE2>,
               ...
               "inputStage" : {
                  ...
               }
            }
         }
      }
}
```

The following sections will define common `explain` results.

Topics

- Scan and filter stage
- Index intersection
- Index union
- Multiple index intersection/union
- Compound index
- Sort stage
- Group stage

Scan and filter stage

The optimizer may choose one of the following scans:

COLLSCAN

This stage is a sequential collection scan.

```
{
    "stage" : "COLLSCAN"
}
```

IXSCAN

This stage scans the index keys. The optimizer may retrieve the document within this stage and this may result in a FETCH stage appended later.

```
db.foo.find({"a": 1})
{
    "stage" : "IXSCAN",
    "direction" : "forward",
    "indexName" : <idx_name>
}
```

FETCH

If the optimizer retrieved documents in a stage other than IXSCAN, the result will include a FETCH stage. For example, the IXSCAN query above may result a combination of FETCH and IXSCAN stages:

```
db.foo.find({"a": 1})
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "indexName" : <idx_name>
    }
}
```

IXONLYSCAN scans only the index key. Create compound indexes won't avoid FETCH.

Index intersection

IXAND

Amazon DocumentDB may include an IXAND stage with an inputStages array of IXSCAN if it can utilize index intersection. For example, we may see output like:

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXAND",
        "inputStages" : [
```

```
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

Index union

IXOR

Similar to index intersection, Amazon DocumentDB may include `IXOR` stage with an `inputStages` array for the `$or` operator.

```
db.foo.find({"$or": [{"a": {"$gt": 2}}, {"b": {"$lt": 2}}]})
```

For the above query, the explain output may look like this:

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

Multiple index intersection/union

Amazon DocumentDB can combine multiple index intersection or union stages together then fetch the result. For example:

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                ...
            },
            {
                "stage" : "IXAND",
                "inputStages" : [
                    {
                        "stage" : "IXSCAN",
                        ...
                    },
                    {
                        "stage" : "IXSCAN",
                        ...
                    }
                ]
            }
        ]
    }
}
```

The usage of index intersection or union stages are not impacted by the index type (sparse, compound, etc).

Compound index

Amazon DocumentDB compound index usage is not limited in the beginning subsets of indexed fields; it can use index with the suffix part but it may not be very efficient.

For example, the compound index of `{ a: 1, b: -1 }` can support all three queries below:

`db.orders.find( { a: 1 } )`

```
db.orders.find( { b: 1 } )

db.orders.find( { a: 1, b: 1 } )
```

Sort stage

If there is an index on the requested sort key(s), Amazon DocumentDB can use the index to obtain the order. In that case, the result will not include a `SORT` stage, but rather an `IXSCAN` stage. If the optimizer favors a plain sort, it will include a stage like this:

```
{
    "stage" : "SORT",
    "sortPattern" : {
        "a" : 1,
        "b" : -1
    }
}
```

Group stage

Amazon DocumentDB supports two different group strategies:

- `SORT_AGGREGATE`: On disk sort aggregate.
- `HASH_AGGREGATE`: In memory hash aggregate.

Querying geospatial data with Amazon DocumentDB

This section covers how you can query Geospatial data with Amazon DocumentDB. After you read this section, you will be able to answer how do store, query and index Geospatial data in Amazon DocumentDB.

Topics

- Overview
- Indexing and storing geospatial data
- Querying geospatial data
- Limitations

Overview

Common use cases for Geospatial involve proximity analysis from your data. For example, “finding all airports within 50 miles of Seattle”, or “find the closest restaurants from a given location”. Amazon DocumentDB uses the GeoJSON specification to represent geospatial data. GeoJSON is an open-source specification for the JSON-formatting of shapes in a coordinate space. GeoJSON coordinates captures both longitude and latitude, representing positions on an earth-like sphere.

Indexing and storing geospatial data

Amazon DocumentDB uses the ‘Point’ GeoJSON type to store geospatial data. Each GeoJSON document (or subdocument) is generally composed of two fields:

- **type** - the shape being represented, which informs Amazon DocumentDB how to interpret the "coordinates" field. At this moment, Amazon DocumentDB only supports points
- **coordinates** – a latitude and longitude pair represented as an object in an array – [longitude, latitude]

Amazon DocumentDB also uses 2dsphere indexes to index Geospatial data. Amazon DocumentDB supports indexing points. Amazon DocumentDB supports proximity querying with 2dsphere indexing.

Let’s consider a scenario where you are building an application for food delivery service. You want to store various restaurant’s latitudes and longitude pair in Amazon DocumentDB. To do so, first we recommend that you create an index on the Geospatial field that holds the latitude and longitude pair.

```
use restaurantsdb
db.usarestaurants.createIndex({location:"2dsphere"})
```

The output of this command would look something like this:

```
{
 "createdCollectionAutomatically" : true,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1
}
```

Once you have created an index, you can start inserting data into your Amazon DocumentDB collection.

```
db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Thai Palace",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
         -122.3264,
         47.6009
      ]
   }
});

db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Noodle House",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
        -122.3517,
         47.6159
      ]
   }
});

db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Curry House",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
         -121.4517,
         47.6229
      ]
   }
```

```
});
```

Querying geospatial data

Amazon DocumentDB supports proximity, inclusion and intersection querying of Geospatial data. A good example of a proximity query is finding all points (all airports) that are less than a certain distance and more than a distance from another point (city). A good example of inclusion querying is to find all points (all airports) that located in a specified area/polygon (state of New York). A good example of intersection query is finding a polygon (state) which intersects with a point (city). You can use the following Geospatial operators to gain insights from your data.

- **`$nearSphere`** - `$nearSphere` is a find operator that supports finding points from nearest to farthest from a GeoJSON point.
- **`$geoNear`** - `$geoNear` is an aggregation operator that supports calculating the distance in meters from a GeoJSON point.
- **`$minDistance`** - `$minDistance` is a find operator that is used in conjunction with `$nearSphere` or `$geoNear` to filter documents that are at least at the specified minimum distance from the center point.
- **`$maxDistance`** - `$maxDistance` is a find operator that is used in conjunction with `$nearSphere` or `$geoNear` to filter documents that are at most at the specified maximum distance from the center point.
- **`$geoWithin`** - `$geoWithin` is a find operator that supports finding documents with geospatial data that exists entirely within a specified shape such as a polygon.
- **`$geoIntersects`** - `$geoIntersects` is a find operator that supports finding documents whose geospatial data intersects with a specified GeoJSON object.

> **Note**
>
> `$geoNear` and `$nearSphere` require a 2dsphere index on the GeoJSON field that you use in your proximity query.

Example 1

In this example, you will learn how to find all restaurants (points) sorted by closest distance from an address (point).

To perform such a query, you can use `$geoNear` to calculate distance of set of points from another point. You can also add the `distanceMultiplier` to measure the distance in kilometers.

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001
      }
   }
])
```

The command above would return restaurants sorted by distance (closest to furthest) from the point specified. The output of this command would look something like this

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "state" : "Washington", "city" :
 "Seattle", "name" : "Noodle House", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -122.3517, 47.6159 ] }, "DistanceKilometers" : 0.03422834547294996 }
{ "_id" : ObjectId("611f3da185009a81ad38e74a"), "state" : "Washington", "city" :
 "Seattle", "name" : "Thai Palace", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -122.3264, 47.6009 ] }, "DistanceKilometers" : 2.5009390081704277 }
{ "_id" : ObjectId("611f3dae85009a81ad38e74c"), "state" : "Washington", "city" :
 "Seattle", "name" : "Curry House", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -121.4517, 47.6229 ] }, "DistanceKilometers" : 67.52845344856914 }
```

To limit the number of results in a query, use the `limit` or `num` option.

`limit`:

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
```

```
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001,
         "limit": 10
      }
   }
])
```

num:

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001,
         "num": 10
      }
   }
])
```

Note

`$geoNear` stage supports the `limit` and `num` options to specify maximum number of documents to return. `$geoNear` returns a maximum of 100 documents by default if the `limit` or `num` options are not specified. This is overridden by the value of the `$limit` stage if present and the value is less than 100.

Example 2

In this example, you will learn how to find all restaurants (points) within 2 kilometers of a specific address (point). To perform such a query, you can use `$nearSphere` within a minimum `$minDistance` and maximum `$maxDistance` from a GeoJSON Point

```
db.usarestaurants.find({
   "location":{
      "$nearSphere":{
         "$geometry":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "$minDistance":1,
         "$maxDistance":2000
      }
   }
},
{
   "name":1
})
```

The command above would return restaurants at a maximum distance of 2 kilometers from the point specified. The output of this command would look something like this

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
```

Limitations

Amazon DocumentDB does not support querying or indexing of Polygons, LineString, MultiPoint, MultiPolygon, MultiLineString, and GeometryCollection.

Partial index

A partial index indexes documents in a collection that meets a specified filter criterion. The partial index feature is supported in Amazon DocumentDB 5.0 instance-based clusters.

Topics

- Create a partial index
- Supported operators
- Query using a partial index
- Partial index functionalities
- Partial index limitations

Create a partial index

To create a partial index, use the `createIndex()` method with the `partialFilterExpression` option. For example, the following operation creates a unique compound index in the orders collection that indexes documents having an `OrderID` and having the `isDelivered` field as true:

```
db.orders.createIndex(
  {"category": 1, "CustomerId": 1, "OrderId": 1},
  {"unique": true, "partialFilterExpression":
    {"$and": [
      {"OrderId": {"$exists": true}},
      {"isDelivered": {"$eq": false}}
    ]}
  }
)
```

Supported operators

- **$eq**
- **$exists**
- **$and** (only at top-level)
- **$gt/$gte/$lt/$lte** (index scan is only used when the filter, predicated in the query, matches the partial filter expression exactly) (See Limitations)

Query using a partial index

The following query patterns are possible using partial indexes:

- The query predicate exactly matches the partial index filter expression:

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}}
  ]}).explain()
```

- The query filter's expected result is a logical subset of the partial filter:

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}},
    {"OrderAmount": {"$eq": "5"}}
  ]}).explain()
```

- A sub-predicate of the query can be used in conjunction with other indexes:

```
db.orders.createIndex({"anotherIndex":1})
db.orders.find({ "$or": [
      {"$and": [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]},
      {"anotherIndex": {"$eq": 5}}
    ]
  }).explain()
```

> **Note**
>
> A query planner may opt to use a collection scan rather than an index scan if it is efficient to do so. This is typically seen for very small collections or queries that would return a large portion of a collection.

Partial index functionalities

List partial indexes

List partial indexes with partialFilterExpression using the `getIndex` operation. For example, the `getIndex` operation issued in lists partial indexes with key, name, and partialfilterExpressions fields:

```
db.orders.getIndexes()
```

This example returns the following output:

```
[
  {
    "v" : 4,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "ecommerceApp.orders"
  },
  {
    "v" : 4,
    "unique" : true,
    "key" : {
      "category" : 1,
      "" : 1,
      "CustomerId" : 1,
      "OrderId" : 1
    },
    "name" : "category_1_CustID_1_OrderId_1",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {
      "$and" : [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]
    }
  }
]
```

Multiple partial filter expression on same key:order

Different partial indexes can be created for the same field combinations (key:order). These indexes must have a different name.

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"firstPartialIndex",
```

```
    partialFilterExpression:{"OrderId":{"$exists": true}}
  }
)
```

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"secondPartialIndex",
    partialFilterExpression:{"OrderId":{"$gt": 1000}}
  }
)
```

Run `getIndexes` operation to list all indexes in the collection:

```
db.orders.getIndexes()
```

These examples returns the following output:

```
[
  {
    "v" : 4,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "ecommerceApp.orders"
  },
  {
    "v" : 4,
    "key" : {
      "OrderId" : 1
    },
    "name" : "firstPartialIndex",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {"OrderId":{"$exists": true}}
  },
  {
    "v" : 4,
    "key" : {
      "OrderId" : 1
    },
    "name" : "secondPartialIndex",
```

```
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {"OrderId":{"$gt": 1000}}
  }
]
```

> **Important**
>
> Index names must be different and must be deleted by name only.

Indexes with partial and TTL properties

You can also create indexes having partial and TTL properties by specifying both `partialFilterExpression` and `expireAfterSeconds` options during index creation. This allows you to provide more control over which documents are now removed from a collection.

For example, you may have a TTL index that identifies documents to be deleted after a certain time period. You can now provide extra conditions on when to delete documents using the partial index option:

```
db.orders.createIndex(
    { "OrderTimestamp": 1 },
    {
        expireAfterSeconds: 3600 ,
        partialFilterExpression: { "isDelivered": { $eq: true } }
    }
)
```

This example returns the following output:

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1,
        "operationTime" : Timestamp(1234567890, 1)
}
```

Run the `getIndexes` operation to list indexes present in the collection:

```
db.orders.getIndexes()
```

```
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.orders"
    }
```

This example returns the following output:

```
[
    {
        "v": 4,
        "key": {
            "_id": 1
        },
        "name": "_id_",
        "ns": "ecommerceApp.orders"
    },
    {
        "v": 4,
        "key": {
            "OrderTimestamp": 1
        },
        "name": "OrderTimestamp_1",
        "ns": "ecommerceApp.orders",
        "partialFilterExpression": {
            "isDelivered": {
                "$eq": true
            }
        },
        "expireAfterSeconds": 3600
    }
]
```

Partial index limitations

The following limitations apply to the partial index feature:

- Inequality queries in Amazon DocumentDB will only use a partial index when the query filter predicate exactly matches the `partialFilterExpression` and is of the same datatype.

> **Note**
>
> Even `$hint` cannot be used to force IXSCAN for the above case.

In the following example, the `partialFilterExpression` is only applied to `field1` but not `field2`:

```
db.orders.createIndex(
  {"OrderAmount": 1},
  {"partialFilterExpression": { OrderAmount : {"$gt" : 5}}}
)

db.orders.find({OrderAmount : {"$gt" : 5}}) // Will use partial index
db.orders.find({OrderAmount : {"$gt" : 6}}) // Will not use partial index
db.orders.find({OrderAmount : {"$gt" : Decimal128(5.00)}}) // Will not use partial
 index
```

- A `partialFilterExpression` with array operators are not supported. The following operation will generate an error:

```
db.orders.createIndex(
  {"CustomerId":1},
  {'partialFilterExpression': {'OrderId': {'$eq': [1000, 1001, 1002]}}}
)
```

- The following operators are not supported in partialFilterExpression field:
  - `$all` (array operator)
  - $mod (array operator)
  - `$or`
  - `$xor`
  - `$not`
  - `$nor`
- The data type of the filter expression and the filter should be the same.

Performing text search with Amazon DocumentDB

Amazon DocumentDB's native full text search feature allows you to perform text search on large textual data sets using special purpose text indexes. This section describes the functionalities of the text index feature and provides steps on how to create and use text indexes in Amazon DocumentDB. Text search limitations are also listed.

Topics

- Supported functionalities
- Using Amazon DocumentDB text index
- Differences with MongoDB
- Best practices and guidelines
- Limitations

Supported functionalities

Amazon DocumentDB text search supports the following MongoDB API compatible functionalities:

- Create text indexes on a single field.
- Create compound text indexes that include more than one text field.
- Perform single word or multi-word searches.
- Control search results using weights.
- Sort search results by score.
- Use text index in aggregation pipeline.
- Search for exact phrase.

Using Amazon DocumentDB text index

To create a text index on a field containing string data, specify the string “text” as shown below:

Single field index:

```
db.test.createIndex({"comments": "text"})
```

This index supports text search queries in the "comments" string field in the specified collection.

Create a compound text index on more than one string field:

```
db.test.createIndex({"comments": "text", "title":"text"})
```

This index supports text search queries in the "comments" and "title" string fields in the specified collection. You can specify up to 30 fields when creating a compound text index. Once created, your text search queries will query all the indexed fields.

> **Note**
>
> Only one text index is allowed on each collection.

Listing a text index on an Amazon DocumentDB collection

You can use `getIndexes()` on your collection to identify and describe indexes, including text indexes, as shown in the example below:

```
rs0:PRIMARY> db.test.getIndexes()
[
   {
      "v" : 4,
      "key" : {
         "_id" : 1
      },
      "name" : "_id_",
      "ns" : "test.test"
   },
   {
      "v" : 1,
      "key" : {
         "_fts" : "text",
         "_ftsx" : 1
      },
      "name" : "contents_text",
      "ns" : "test.test",
      "default_language" : "english",
      "weights" : {
         "comments" : 1
      },
      "textIndexVersion" : 1
```

```
    }
]
```

Once you have created an index, start inserting data into your Amazon DocumentDB collection.

```
db.test.insertMany([{"_id": 1, "star_rating": 4, "comments": "apple is red"},
                    {"_id": 2, "star_rating": 5, "comments": "pie is delicious"},
                    {"_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy
 fruit"},
                    {"_id": 4, "star_rating": 2, "comments": "bake the apple pie in the
 oven"},
                    {"_id": 5, "star_rating": 5, "comments": "interesting couch"},
                    {"_id": 6, "star_rating": 5, "comments": "interested in couch for
 sale, year 2022"}])
```

Running text search queries

Run a single-word text search query

You will need to use `$text` and `$search` operators to perform text searches. The following example returns all documents where your text indexed field contain the string “apple” or “apple” in other formats such as “apples”:

```
db.test.find({$text: {$search: "apple"}})
```

Output:

The output of this command looks something like this:

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

Run a multi-word text search

You can also perform multi-word text searches on your Amazon DocumentDB data. The command below returns documents with a text indexed field containing “apple” or “pie”:

```
db.test.find({$text: {$search: "apple pie"}})
```

Output:

The output of this command looks something like this:

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 2, "star_rating" : 5, "comments" : "pie is delicious" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

Run a multi-word phrase text search

For a multi-word phrase search, use this example:

```
db.test.find({$text: {$search: "\"apple pie\""}})
```

Output:

The command above returns documents with text indexed field containing the exact phrase “apple pie”. The output of this command looks something like this:

```
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

Run a text search with filters

You can also combine text search with other query operators to filter results based on additional criteria:

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

Output:

The command above returns documents with a text indexed field containing any form of “interest” and a “star_rating” equal to 5. The output of this command looks something like this:

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

Limit the number of documents returned in a text search

You can choose to restrict the number of documents returned by using `limit`:

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "couch"}}]}).limit(1)
```

Output:

The command above returns one result that satisfies the filter:

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
```

Sort results by text score

The following example sorts the text search results by text score:

```
db.test.find({$text: {$search: "apple"}}, {score: {$meta: "textScore"}}).sort({score:
 {$meta: "textScore"}})
```

Output:

The command above returns documents with a text indexed field containing “apple”, or “apple” in it's other formats like “apples”, and sorts the result based on how relevant the document is related to the search term. The output of this command looks something like this:

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red", "score" :
 0.6079270860936958 }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit",
 "score" : 0.6079270860936958 }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven",
 "score" : 0.6079270860936958 }
```

`$text` and `$search` are also supported for `aggregate`, `count`, `findAndModify`, `update`, and `delete` commands.

Aggregation operators

Aggregation pipeline using `$match`

```
db.test.aggregate(
   [{ $match: { $text: { $search: "apple pie" } } }]
)
```

Output:

The command above returns the following results:

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apple - a healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
{ "_id" : 2, "star_rating" : 5, "comments" : "pie is delicious" }
```

A combination of other aggregation operators

```
db.test.aggregate(
   [
      { $match: { $text: { $search: "apple pie" } } },
      { $sort: { score: { $meta: "textScore" } } },
      { $project: { score: { $meta: "textScore" } } }
   ]
)
```

Output:

The command above returns the following results:

```
{ "_id" : 4, "score" : 0.6079270860936958 }
{ "_id" : 1, "score" : 0.3039635430468479 }
{ "_id" : 2, "score" : 0.3039635430468479 }
{ "_id" : 3, "score" : 0.3039635430468479 }
```

Specify multiple fields when creating a text index

You can assign weights to up to three fields in your compound text index. The default weight assigned to a field in a text index is one (1). Weight is an optional parameter and must be in the range from 1 to 100000.

```
db.test.createIndex(
   {
     "firstname": "text",
     "lastname": "text",
     ...
   },
   {
```

```
    weights: {
      "firstname": 5,
      "lastname":10,
      ...
    },
    name: "name_text_index"
  }
)
```

Differences with MongoDB

Amazon DocumentDB's text index feature uses inverted index with a term-frequency algorithm. Text indexes are sparse by default. Due to differences in parsing logic, tokenization delimiters, and others, the same result set as MongoDB may not be returned for the same dataset or query shape.

The following additional differences between Amazon DocumentDB text index and MongoDB exist:

- Compound indexes using non-text indexes are not supported.
- Amazon DocumentDB text indexes are case insensitive.
- Only English language is supported with text index.
- Text indexing of array (or multi-key) fields is not supported. For example, creating a text index on "a" with the document {"a":["apple", "pie"]} will fail.
- Wildcard text indexing is not supported.
- Unique text indexes are not supported.
- Excluding a term is not supported.

Best practices and guidelines

- For optimal performance on text search queries involving sorting by text scores, we recommended that you create the text index before loading data.
- Text indexes require additional storage for an optimized internal copy of the indexed data. This has additional cost implications.

Limitations

Text search has the following limitations in Amazon DocumentDB:

- Text search is supported on Amazon DocumentDB 5.0 instance-based clusters only.
- Text indexes store lexemes and their position information. The combined size of all lexemes and their position information, within a single document, is limited to 1MB.

Troubleshooting Amazon DocumentDB

The following sections provide information about how to troubleshoot problems that you might encounter when using Amazon DocumentDB (with MongoDB compatibility).

Topics

- Troubleshooting connection issues
- Troubleshooting index creation
- Troubleshooting performance and resource utilization

Troubleshooting connection issues

Having trouble connecting? Here are some common scenarios and how to resolve them.

Topics

- Cannot connect to an Amazon DocumentDB endpoint
- Testing a connection to an Amazon DocumentDB instance
- Connecting to an invalid endpoint
- Driver configuration impacting number of connections

Cannot connect to an Amazon DocumentDB endpoint

When you try to connect to Amazon DocumentDB, the following is one of the most common error messages that you might receive.

```
connecting to: mongodb://docdb-2018-11-08-21-47-27.cluster-ccuszbx3pn5e.us-east-
1.docdb.amazonaws.com:27017/
2018-11-14T14:33:46.451-0800 W NETWORK [thread1] Failed to connect to
172.31.91.193:27017 after 5000ms milliseconds, giving up.
2018-11-14T14:33:46.452-0800 E QUERY [thread1] Error: couldn't connect to server
docdb-2018-11-08-21-47-27.cluster-ccuszbx3pn5e.us-east-1.docdb.amazonaws.com:27017,
connection attempt failed :
connect@src/mongo/shell/mongo.js:237:13
@(connect):1:6
exception: connect failed
```

What this error message typically means is that your client (the mongo shell in this example) cannot access the Amazon DocumentDB endpoint. This might be the case for several reasons:

Topics

- Connecting from public endpoints
- Cross region connections
- Connecting from different Amazon VPCs
- Security group blocks inbound connections
- Java Mongo driver read preference issue

Connecting from public endpoints

You are trying to connect to an Amazon DocumentDB cluster directly from your laptop or local development machine.

Trying to connect to an Amazon DocumentDB cluster directly from a public endpoint, such as your laptop or local development machine, will fail. Amazon DocumentDB is virtual private cloud (VPC)-only and does not currently support public endpoints. Thus, you can't connect directly to your Amazon DocumentDB cluster from your laptop or local development environment outside of your VPC.

To connect to an Amazon DocumentDB cluster from outside an Amazon VPC, you can use an SSH tunnel. For more information, see Connecting to an Amazon DocumentDB cluster from outside an Amazon VPC. Additionally, if your development environment is in a different Amazon VPC, you can also use VPC Peering and connect to your Amazon DocumentDB cluster from another Amazon VPC in the same region or a different region.

Cross region connections

You are trying to connect to an Amazon DocumentDB cluster in another region.

If you try to connect to an Amazon DocumentDB cluster from an Amazon EC2 instance in a Region other than the cluster's Region—for example, trying to connect to a cluster in US East (N. Virginia) Region (us-east-1) from US West (Oregon) Region (us-west-2)—the connection will fail.

To verify the Region of your Amazon DocumentDB cluster, run the following command. The Region is in the endpoint.

```
aws docdb describe-db-clusters \
```

```
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].Endpoint'
```

Output from this operation looks something like the following.

```
[
    "sample-cluster.node.us-east-1.docdb.amazonaws.com"
]
```

To verify the Region of your EC2 instance, run the following command.

```
 aws ec2 describe-instances \
     --query 'Reservations[*].Instances[*].Placement.AvailabilityZone'
```

Output from this operation looks something like the following.

```
[
    [
        "us-east-1a"
    ]
]
```

Connecting from different Amazon VPCs

You are trying to connect to an Amazon DocumentDB cluster from a VPC that is different than the Amazon VPC your cluster is deployed to.

If both your Amazon DocumentDB cluster and Amazon EC2 instance are in the same AWS Region, but not in the same Amazon VPC, you cannot connect directly to your Amazon DocumentDB cluster unless VPC Peering is enabled between the two Amazon VPCs.

To verify the Amazon VPC of your Amazon DocumentDB instance, run the following command.

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].DBSubnetGroup.VpcId'
```

To verify the Amazon VPC of your Amazon EC2 instance, run the following command.

```
aws ec2 describe-instances \
```

```
   --query 'Reservations[*].Instances[*].VpcId'
```

Security group blocks inbound connections

You are trying to connect to an Amazon DocumentDB cluster, and the cluster's security group does not allow inbound connections on the cluster's port (default port: 27017).

Suppose that your Amazon DocumentDB cluster and Amazon EC2 instance are both in the same Region and Amazon VPC and use the same Amazon VPC security group. If you can't connect to your Amazon DocumentDB cluster, the likely cause is that your security group (that is, firewall) for your cluster doesn't allow inbound connections on the port you chose for your Amazon DocumentDB cluster (default port is 27017).

To verify the port for your Amazon DocumentDB cluster, run the following command.

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port]'
```

To get your Amazon DocumentDB security group for your cluster, run the following command.

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[VpcSecurityGroups[*],VpcSecurityGroupId]'
```

To check the inbound rules for your security group, see the following topics in the Amazon EC2 documentation:

- Authorizing Inbound Traffic for Your Linux Instances
- Authorizing Inbound Traffic for Your Windows Instances

Java Mongo driver read preference issue

Client read preferences are not honored and some clients cannot write to Amazon DocumentDB after failover unless they reboot.

This issue, first discovered in Java Mongo Driver 3.7.x, occurs when a client establishes a connection to Amazon DocumentDB using `MongoClientSettings` and, specifically, when chaining the `applyToClusterSettings` method. The MongoClient Cluster Settings can be defined using a few different methods, such as `hosts()`, `requiredReplicaSetName()`, and `mode()`.

When the client specifies only one host in the `hosts()` method, the mode is set to `ClusterConnectionMode.SINGLE` instead of `ClusterConnectionMode.MULTIPLE` This causes the client to disregard the read preference and only connect to the server configured in `hosts()`. So even if the client settings are initialized like below, all reads would still go to the primary instead of the secondary.

```
final ServerAddress serverAddress0 = new ServerAddress("cluster-endpoint", 27317));
    final MongoCredential credential = MongoCredential.createCredential("xxx",
            "admin", "xxxx".toCharArray());
    final MongoClientSettings settings = MongoClientSettings.builder()
            .credential(credential)
            .readPreference(ReadPreference.secondaryPreferred())
            .retryWrites(false)
            .applyToSslSettings(builder -> builder
                    .enabled(false))
            .applyToClusterSettings(builder -> builder.hosts(
                            Arrays.asList(serverAddress0
                            ))
                    .requiredReplicaSetName("rs0"))
            .build();
    MongoClient mongoClient = MongoClients.create(settings);
```

Failover case

Using the above client connection settings, if there is a failover and a delayed DNS record update for the cluster writer endpoint, the client would still try to issue writes to the old writer (now reader after failover). This results in a server-side error (not primary) which is not handled appropriately by the Java driver (this is still under investigation). Thus, the client can be left in a bad state until the application server is rebooted, for example.

There are two workarounds for this:

- Clients that connect to Amazon DocumentDB via a connection string will not have this issue, since `ClusterConnectionMode` will be set to `MULTIPLE` when setting read preference.

  ```
  MongoClientURI mongoClientURI = new MongoClientURI("mongodb://usr:pass:cluster-
  endpoint:27317/test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred");
  MongoClient mongoClient = MongoClients.create(mongoClientURI.getURI());
  ```

  Or using `MongoClientSettings` builder with the `applyConnectionString` method.

```
final MongoClientSettings settings = MongoClientSettings.builder()
        .credential(credential)
        .applyConnectionString(new ConnectionString("usr:pass:cluster-endpoint:27317/
test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred"))
        .retryWrites(false)
        .applyToSslSettings(builder # builder
                .enabled(false))
        .build();
MongoClient mongoClient = MongoClients.create(settings);
```

- Explicitly set `ClusterConnectionMode` to `MULTIPLE`. This is only needed when using `applyToClusterSettings` and `hosts().size() == 1`.

```
final ServerAddress serverAddress0 = new ServerAddress("cluster-endpoint", 27317));
final MongoCredential credential = MongoCredential.createCredential("xxx","admin",
 "xxxx".toCharArray());
final MongoClientSettings settings = MongoClientSettings.builder()
    .credential(credential)
    .readPreference(ReadPreference.secondaryPreferred())
    .retryWrites(false)
    .applyToSslSettings(builder # builder
    .enabled(false))
    .applyToClusterSettings(builder # builder
                .hosts(Arrays.asList(serverAddress0))
                .requiredReplicaSetName("rs0"))
                .mode(ClusterConnectionMode.MULTIPLE))
    .build();
MongoClient mongoClient = MongoClients.create(settings);
```

Testing a connection to an Amazon DocumentDB instance

You can test your connection to a cluster using common Linux or Windows tools.

From a Linux or Unix terminal, test the connection by entering the following (replace `cluster-endpoint` with the endpoint, and replace `port` with the port of your instance):

```
nc -zv cluster-endpoint port
```

The following is an example of a sample operation and the return value:

```
nc -zv docdbTest.d4c7nm7stsfc0.us-west-2.docdb.amazonaws.com 27017

Connection to docdbTest.d4c7nm7stsfc0.us-west-2.docdb.amazonaws.com 27017 port [tcp/*]
 succeeded!
```

Connecting to an invalid endpoint

When connecting to an Amazon DocumentDB cluster and you use a cluster endpoint that is not valid, an error similar to the following appears.

```
mongo --ssl \
   --host sample-cluster.node.us-east-1.docdb.amazonaws.com:27017 \
   --sslCAFile global-bundle.pem \
   --username <user-name> \
   --password <password>
```

The output looks like this:

```
MongoDB shell version v3.6
connecting to: mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
2018-11-14T17:21:18.516-0800 I NETWORK [thread1] getaddrinfo("sample-cluster.node.us-
east-1.docdb.amazonaws.com") failed:
nodename nor servname provided, or not known 2018-11-14T17:21:18.537-0800 E QUERY
 [thread1] Error: couldn't initialize
connection to host sample-cluster.node.us-east-1.docdb.amazonaws.com, address is
 invalid :
connect@src/mongo/shell/mongo.js:237:13@(connect):1:6
exception: connect failed
```

To get the valid endpoint for a cluster, run the following command:

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[Endpoint,Port]'
```

To get the valid endpoint for an instance, run the following command:

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].[Endpoint.Address,Endpoint.Port]'
```

For more information, see Understanding Amazon DocumentDB endpoints.

Driver configuration impacting number of connections

When using the client driver to connect to an Amazon DocumentDB cluster, it's important to consider the `maxPoolSize` configuration parameter. The `maxPoolSize` setting determines the maximum number of connections that the client driver will maintain in its connection pool.

Troubleshooting index creation

The following topics address what to do if your index or background index build fails.

Topics

- Index build fails
- Background index build latency issues and fails

Index build fails

Amazon DocumentDB utilizes local storage on an instance as part of the index creation process. You can monitor this disk usage using the **FreeLocalStorage** CloudWatch metric (`CloudWatch -> Metrics -> DocDB -> Instance Metrics`). When an index build consumes all of the local disk and fails, you will receive an error. When migrating data to Amazon DocumentDB, we encourage you to create indexes first and then insert the data. For more information on migration strategies and creating indexes, see Migrating to Amazon DocumentDB in the Amazon DocumentDB documentation and the blog: Migrate from MongoDB to Amazon DocumentDB using the offline method.

When creating indexes on an existing cluster, if the index build is taking longer than expected or is failing, we recommend that you scale up the instance to create the index then, after the index is created, scale back down. Amazon DocumentDB enables you to quickly scale instance sizes in minutes using the AWS Management Console or the AWS CLI. For more information, see Managing instance classes. With per-second instance pricing, you only pay for the resource you use up to the second.

Background index build latency issues and fails

Background index builds in Amazon DocumentDB do not start until all queries on the primary instance that started before the index build was initiated complete executing. If there is a long

running query, background index builds will block until the query finishes and thus can take longer than expected to complete. This is true even if collections are empty.

Foreground index builds do not exhibit the same blocking behavior. Instead, foreground index builds take an exclusive lock on the collection until the index build is completed. Thus, to create indexes on empty collection and to avoid blocking on any long running queries, we suggest using foreground index builds.

> **Note**
>
> Amazon DocumentDB allows only one background index build to occur on a collection at any given time. If DDL (Data Definition Language) operations such as `createIndex()` or`dropIndex()` occur on the same collection during a background index build, the background index build fails.

Troubleshooting performance and resource utilization

This section provides questions and solutions for common diagnostics issues in Amazon DocumentDB deployments. The examples provided use the *mongo shell* and are scoped to an individual instance. To find an instance endpoint, see Understanding Amazon DocumentDB endpoints.

Topics

- How do I determine the number of insert, update, and delete operations performed on my collection through the Mongo API?
- How do I analyze cache performance?
- How do I find and terminate long running or blocked queries?
- How can I see a query plan and optimize a query?
- How can I see a query plan in elastic clusters?
- How do I list all running operations on an instance?
- How do I know when a query is making progress?
- How do I determine why a system suddenly runs slowly?
- How do I determine the cause of high CPU utilization on one or more cluster instances?
- How do I determine the open cursors on an instance?
- How do I determine the current Amazon DocumentDB engine version?

- How do I analyze index usage and identify unused indexes?
- How do I identify missing indexes?
- Summary of useful queries

How do I determine the number of insert, update, and delete operations performed on my collection through the Mongo API?

To view the number of insert, update, and delete operations performed on a certain collection, run the following command on that collection:

```
db.collection.stats()
```

The output from this command describes the following under its `opCounters` field:

- **numDocsIns** - The number of documents inserted into this collection. This includes documents inserted using the `insert` and `insertMany` commands, as well as documents inserted by an upsert.
- **numDocsUpd** - The number of documents updates in this collection. This includes documents updated using the `update` and `findAndModify` commands.
- **numDocsDel** - The number of documents deleted from this collection. This includes documents deleted using the `deleteOne`, `deleteMany`, `remove`, and `findAndModify` commands.
- **lastReset** - The time these counters have been last reset. The statistics provided by this command are reset when starting/stopping the cluster or scaling up/down the instance.

An example output from running `db.collection.stats()` is shown below.

```
{
    "ns" : "db.test",
    "count" : ...,
    "size" : ...,
    "avgObjSize" : ...,
    "storageSize" : ...,
    "capped" : false,
    "nindexes" : ...,
    "totalIndexSize" : ...,
    "indexSizes" : {
        "_id_" : ...,
```

```
        "x_1" : ...
    },
    "collScans" : ...,
    "idxScans" : ...,
    "opCounter" : {
        "numDocsIns" : ...,
        "numDocsUpd" : ...,
        "numDocsDel" : ...
    },
    "cacheStats" : {
        "collBlksHit" : ...,
        "collBlksRead" : ..,
        "collHitRatio" : ...,
        "idxBlksHit" : ...,
        "idxBlksRead" : ...,
        "idxHitRatio" : ...
    },
    "lastReset" : "2022-09-02 19:41:40.471473+00",
    "ok" : 1,
    "operationTime" : Timestamp(1662159707, 1)
}
```

This stats command should be used when viewing collection-specific counters for insert, update, and delete operation through the Mongo API. Another way to view collection-specific operation counters is by enabling DML auditing. The number of insert, update, and delete operations over all collections during one minute time intervals can be viewed in Monitoring Amazon DocumentDB with CloudWatch.

How do I analyze cache performance?

Analyzing cache performance can give insights into the efficiency of data retrieval and system performance, and is based on how much data is read from the disk versus the cache. We provide cache statistics about the number of cache hits (data read from the cache) and cache misses (data that is not found in the cache and read from the disk) in order give insight into the cache performance. The cache statistics for a specific collection can by found by running the following command on that collection:

```
db.collection.stats()
```

The values in the `cacheStats` field in the output of this command provide cache statistics for the collection as well as the total cache statistics for the indexes created on the collection. These statistics are listed below:

- **`collBlksHit`** - The number of blocks read from the cache during operations on this collection.
- **`collBlksRead`** - The number of blocks read from the disk (cache misses) during operations on this collection.
- **`collHitRatio`** - The cache hit ratio for this collection (`100 * [collBlksHit / (collBlksHit + collBlksRead)]`).
- **`idxBlksHit`** - The number of blocks read from the cache for any index created on this collection.
- **`idxBlksRead`** - The number of blocks read from the disk (cache misses) for any index created on this collection.
- **`idxHitRatio`** - The cache hit ratio for the indexes created on this collection (`100 * [idxBlksHit / (idxBlksHit + idxBlksRead)]`).
- **`lastReset`** - The time these statistics have been last reset. The statistics provided by `db.collection.stats()` are reset when starting/stopping the cluster or scaling up/down the instance.

A breakdown of the `idxBlksHit` and `idxBlksRead` fields for each index can also be found using the `indexStats` command. Index specific cache statistics can be found by running the following command:

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

For each index, the following cache statistics can be found under the `cacheStats` field:

- **`blksHit`** - The number of blocks read from cache for this index.
- **`blksRead`** - The number of blocks read from the disk for this index.
- **`blksHitRatio`** - The cache hit ratio rounded to four decimal places, calculated by `100 * [blksHit / (blksHit + blksRead)]`.

How do I find and terminate long running or blocked queries?

User queries can run slowly because of a suboptimal query plan or can be blocked due to resource contention.

To find long running queries that slow down due to a suboptimal query plan, or queries that are blocked due to resource contention, use the `currentOp` command. You can filter the command to help narrow down the list of relevant queries to terminate. You must have `opid` associated with the long running query to be able to terminate the query.

The following query uses the `currentOp` command to list all queries that are either blocked or running for more than 10 seconds.

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or: [
                {secs_running: {$gt: 10}},
                {WaitState: {$exists: true}}]}},
        {$project: {_id:0, opid: 1, secs_running: 1}}],
    cursor: {}
});
```

Next, you can narrow down the query to find the `opid` of a query running for more than 10 seconds and terminate it.

To find and terminate a query running for more than 10 seconds

1. Find the `opid` of the query.

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or:
                [{secs_running: {$gt: 10}},
                 {WaitState: {$exists: true}}]}}],
    cursor: {}
```

```
});
```

Output from this operation looks something like the following (JSON format).

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 24646,
                "secs_running" : 12
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

2. Terminate the query using the `killOp` operation.

```
db.adminCommand({killOp: 1, op: 24646});
```

How can I see a query plan and optimize a query?

If a query runs slow, it could be because the query execution requires a full scan of the collection to choose the relevant documents. Sometimes creating appropriate indexes enables the query to run faster. To detect this scenario and decide the fields on which to create the indexes, use the `explain` command.

> **Note**
>
> Amazon DocumentDB emulates the MongoDB 3.6 API on a purpose-built database engine that utilizes a distributed, fault-tolerant, self-healing storage system. As a result, query plans and the output of `explain()` may differ between Amazon DocumentDB and MongoDB. Customers who want control over their query plan can use the `$hint` operator to enforce selection of a preferred index.

Run the query that you want to improve under the `explain` command as follows.

```
db.runCommand({explain: {<query document>}})
```

The following is an example operation.

```
db.runCommand({explain:{
    aggregate: "sample-document",
    pipeline: [{$match: {x: {$eq: 1}}}],
    cursor: {batchSize: 1}}
});
```

Output from this operation looks something like the following (JSON format).

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "db.test",
        "winningPlan" : {
            "stage" : "COLLSCAN"
        }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

The preceding output indicates that the `$match` stage requires scanning the whole collection and checking if the field `"x"` in each document is equal to 1. If there are many documents in the collection, the collection scan (and therefore the overall query performance) is very slow. Thus the presence of the `"COLLSCAN"` in the output of the `explain` command indicates that the query performance can be improved by creating appropriate indexes.

In this example, the query checks whether the field `"x"` equals 1 in all documents. So creating an index on field `"x"` enables the query to avoid the complete collection scan and use the index to return the relevant documents sooner.

After creating an index on field `"x"`, the `explain` output is as follows.

```
{
    "queryPlanner" : {
         "plannerVersion" : 1,
         "namespace" : "db.test",
         "winningPlan" : {
             "stage" : "IXSCAN",
             "indexName" : "x_1",
             "direction" : "forward"
         }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

Creating an index on field `"x"` enables the `$match` stage to use an index scan to reduce the number of documents on which the predicate `"x = 1"` must be evaluated.

For small collections, the Amazon DocumentDB query processor can choose not to use an index if the performance gains are negligible.

How can I see a query plan in elastic clusters?

To examine a query plan in elastic clusters, use the `explain` command. The following is an example `explain` operation on a find query targeting a sharded collection:

```
db.runCommand(
   {
     explain: { find: "cities", filter: {"name": "Seoul"}}
   }
)
```

> **Note**
>
> Amazon DocumentDB emulates MongoDB on a purpose-built database engine. As a result, query plans and the output of `explain()` may differ between Amazon DocumentDB

and MongoDB. You can control query plan with the use of the `$hint` operator to enforce selection of a preferred index.

Output from this operation may look something like the following (JSON format):

```
{
  "queryPlanner" : {
    "elasticPlannerVersion" : 1,
    "winningPlan" : {
      "stage" : "SINGLE_SHARD",
      "shards" : [
        {
          "plannerVersion" : 1,
          "namespace" : "population.cities",
          "winningPlan" : {
            "stage" : "SHARD_MERGE",
            "shards" : [
              {
                "shardName" : "f2cf5cfd-fe9c-40ca-b4e5-298ca0d11111",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 21
                    }
                  ]
                }
              },
              {
                "shardName" : "8f3f80e2-f96c-446e-8e9d-aab8c7f22222",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 21
                    }
```

```
                  ]
                }
              },
              {
                "shardName" : "32c5a06f-1b2b-4af1-8849-d7c4a033333",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 22
                    }
                  ]
                }
              }
            ]
          },
          "shardName" : "32c5a06f-1b2b-4af1-8849-d7c4a0f3fb58"
        }
      ]
    }
  },
  "serverInfo" : {
    "host" : "example-4788267630.us-east-1.docdb-elastic.amazonaws.com:27017",
    "version" : "5.0.0"
  },
  "ok" : 1,
  "operationTime" : Timestamp(1695097923, 1)
}
```

The preceding output shows the query plan for the `find` query on a three-shard cluster. Each shard has multiple data partitions which can have different input stages. In this example, a “COLLSCAN“ (a collection scan) is run on all partitions before the results are merged at the ”PARTITION_MERGE“ stage within each shard. The results across the shards are then merged together at the ”SHARD_MERGE“ stage before being sent back to the client.

How do I list all running operations on an instance?

As a user or primary user, you often want to list all the current operations running on an instance for diagnostics and troubleshooting purposes. (For information about managing users, see Managing Amazon DocumentDB users.)

With the mongo shell, you can use the following query to list all the running operations on an Amazon DocumentDB instance.

```
db.adminCommand({currentOp: 1, $all: 1});
```

The query returns the complete list of all user queries and internal system tasks currently operating on the instance.

Output from this operation looks something like the following (JSON format).

```
{
    "inprog" : [
        {
            "desc" : "INTERNAL"
        },
        {
            "desc" : "TTLMonitor",
            "active" : false
        },
        {
            "client" : ...,
            "desc" : "Conn",
            "active" : true,
            "killPending" : false,
            "opid" : 195,
            "ns" : "admin.$cmd",
            "command" : {
                "currentOp" : 1,
                "$all" : 1
            },
            "op" : "command",
            "$db" : "admin",
            "secs_running" : 0,
            "microsecs_running" : NumberLong(68),
            "clientMetaData" : {
            "application" : {
```

```
                "name" : "MongoDB Shell"
            },
            "driver" : {
                ...
            },
            "os" : {
                ...
            }
        }
    },
    {
        "desc": "GARBAGE_COLLECTION",
        "garbageCollection": {
            "databaseName": "testdb",
            "collectionName": "testCollectionA"
        },
        "secs_running": 3,
        "microsecs_running": NumberLong(3123456)
    },
    {
        "desc": "GARBAGE_COLLECTION",
        "garbageCollection": {
            "databaseName": "testdb",
            "collectionName": "testCollectionB"
        },
        "secs_running": 4,
        "microsecs_running": NumberLong(4123456)
    }
  ],
  "ok" : 1
}
```

The following are valid values for the `"desc"` field:

- **`INTERNAL`** — Internal system tasks like the cursor cleanup or stale user cleanup tasks.
- **`TTLMonitor`** — The Time to Live (TTL) monitor thread. Its running status is reflected in the `"active"` field.
- **`GARBAGE_COLLECTION`** — The internal garbage collector thread.
- **`CONN`** — The user query.

- **CURSOR** — The operation is an idle cursor waiting on the user to call the "getMore" command to get the next batch of results. In this state, the cursor is consuming memory, but is not consuming any compute.

The preceding output also lists all user queries running in the system. Each user query runs in the context of a database and collection, and the union of these two is called a *namespace*. The namespace of each user query is available in the `"ns"` field.

Sometimes you need to list all user queries that are running in a particular namespace. So the previous output must be filtered on the `"ns"` field. The following is an example query to achieve the output to filter. The query lists all user queries that are currently running in the database `"db"` and collection `"test"` (that is, the `"db.test"` namespace).

```
db.adminCommand({aggregate: 1,
    pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
               {$match: {ns: {$eq: "db.test"}}}],
    cursor: {}
});
```

As the primary user of the system, you can see queries of all users and also all internal system tasks. All other users can see only their respective queries.

If the total number of queries and internal system tasks exceeds the default batch cursor size, the mongo shell automatically generates an iterator object `'it'` to view the rest of the results. Keep executing the `'it'` command until all results have been exhausted.

How do I know when a query is making progress?

User queries can run slowly due to a suboptimal query plan, or they can be blocked due to resource contention. Debugging such queries is a multi-step process that can require executing the same step multiple times.

The first step of debugging is to list all queries that are long running or blocked. The following query lists all user queries that have been running for more than 10 seconds or that are waiting for resources.

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {}},
                            {$match: {$or: [{secs_running: {$gt: 10}},
                                            {WaitState: {$exists: true}}]}},
```

```
                            {$project: {_id:0,
                                        opid: 1,
                                        secs_running: 1,
                                        WaitState: 1,
                                        blockedOn: 1,
                                        command: 1}}],
                 cursor: {}
                });
```

Repeat the preceding query periodically to determine whether the list of queries changes and to identify the long running or blocked queries.

If the output document for the query of interest has a `WaitState` field, it indicates that resource contention is why the query is running slow or is blocked. The resource contention could either be due to I/O, internal system tasks, or other user queries.

Output from this operation looks something like the following (JSON format).

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 201,
                "command" : {
                    "aggregate" : ...
                },
                "secs_running" : 208,
                "WaitState" : "IO"
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

I/O can be a bottleneck if many queries across different collections are running concurrently on the same instance, or if the instance size is too small for the dataset that the query is running on. If the queries are read-only queries, you can mitigate the former situation by separating the queries for each collection across separate replicas. For concurrent updates across different collections, or when the instance size is too small for the dataset, you can mitigate by scaling up the instance.

If the resource contention is due to other user queries, the `"blockedOn"` field in the output document will have the `"opid"` of the query that is affecting this query. Using the `"opid"` follows the chain of `"WaitState"` and `"blockedOn"` fields of all the queries to find the query at the head of the chain.

If the task at the head of the chain is an internal task, the only mitigation in this case would be to terminate the query and rerun it later.

The following is sample output in which the find query is blocked on a collection lock that is owned by another task.

```
{
    "inprog" : [
        {
            "client" : "...",
            "desc" : "Conn",
            "active" : true,
            "killPending" : false,
            "opid" : 75,
            "ns" : "...",
            "command" : {
                "find" : "...",
                "filter" : {

                }
            },
            "op" : "query",
            "$db" : "test",
            "secs_running" : 9,
            "microsecs_running" : NumberLong(9449440),
            "threadId" : 24773,
            "clientMetaData" : {
                "application" : {
                   "name" : "MongoDB Shell"
                },
                "driver" : {
                    ...
                },
                "os" : {
                    ...
                }
            },
            "WaitState" : "CollectionLock",
```

```
            "blockedOn" : "INTERNAL"
        },
        {
            "desc" : "INTERNAL"
        },
        {
            "client" : "...",
            ...
            "command" : {
                "currentOp" : 1
            },
            ...
        }
    ],
    "ok" : 1
}
```

If the `"WaitState"` has values `"Latch"`, `"SystemLock"`, `"BufferLock"`, `"BackgroundActivity"`, or `"Other"`, the source of resource contention is internal system tasks. If the situation continues for a long time, the only mitigation would be to terminate the query and rerun it later.

How do I determine why a system suddenly runs slowly?

The following are some common reasons for a system slowing down:

- Excessive resource contention between concurrent queries
- The number of active concurrent queries increasing over time
- Internal system tasks such as `"GARBAGE_COLLECTION"`

To monitor the system usage over time, run the following `"currentOp"` query periodically and output the results to an external store. The query counts the number of queries and operations in each namespace in the system. You can then analyze the system usage results to understand the load on the system and take appropriate action.

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
```

```
                });
```

This query returns an aggregate of all queries running in each namespace, all the internal system tasks, and the unique number of wait states (if any) per namespace.

Output from this operation looks something like the following (JSON format).

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "_id" : {
                    "desc" : "Conn",
                    "ns" : "db.test",
                    "WaitState" : "CollectionLock"
                },
               "count" : 2
            },
            {
                "_id" : {
                    "desc" : "Conn",
                    "ns" : "admin.$cmd"
                },
                "count" : 1
            },
            {
                "_id" : {
                    "desc" : "TTLMonitor"
                },
                "count" : 1
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

In the preceding output, two user queries in namespace `"db.test"` are blocked on collection lock: one query in the namespace `"admin.$cmd"`, and one internal `"TTLMonitor"` task.

If the output indicates many queries with blocking wait states, see How do I find and terminate long running or blocked queries?

How do I determine the cause of high CPU utilization on one or more cluster instances?

The following sections might help you identify the cause of high instance CPU utilization. Your results can vary depending on the workload.

- To determine why an instance is suddenly running slowly, see How do I determine why a system suddenly runs slowly?
- To identify and terminate long running queries on a particular instance, see How do I find and terminate long running or blocked queries?
- To understand whether a query is progressing, see How do I know when a query is making progress?
- To determine why a query takes a long time to run, see How can I see a query plan and optimize a query?
- To track long-running queries over time, see Profiling Amazon DocumentDB operations.

Depending on the reason for your high instance CPU utilization, doing one or more of the following can help.

- If the primary instance exhibits high CPU utilization, but the replica instances don't, consider distributing read traffic across replicas via client read preference settings (for example, `secondaryPreferred`). For more information, see Connecting to Amazon DocumentDB as a replica set.

  Using replicas for reads can make better use of the cluster's resources by allowing the primary instance to process more write traffic. Reads from replicas are eventually consistent.
- If the high CPU utilization is a result of your write workload, changing the size of the cluster's instances to a larger instance type increases the number of CPU cores available to service the workload. For more information, see Instances and Instance class specifications.
- If all cluster instances exhibit high CPU utilization, and the workload is using replicas for reads, adding more replicas to the cluster increases the resources available for read traffic. For more information, see Adding an Amazon DocumentDB instance to a cluster.

How do I determine the open cursors on an instance?

When connected to a Amazon DocumentDB instance, you can use the command `db.runCommand("listCursors")` to list the open cursors on that instance. There is a limit of up to 4,560 active cursors open at any given time on a given Amazon DocumentDB instance, depending on the instance type. It is generally advised to close cursors that are no longer in use because cursors utilize resources on an instance and have an upper limit. See Amazon DocumentDB Quotas and limits for specific limits.

```
db.runCommand("listCursors")
```

How do I determine the current Amazon DocumentDB engine version?

To determine your current Amazon DocumentDB engine version, run the following command.

```
db.runCommand({getEngineVersion: 1})
```

Output from this operation looks something like the following (JSON format).

```
{ "engineVersion" : "2.x.x", "ok" : 1 }
```

> **Note**
>
> The engine version for Amazon DocumentDB 3.6 is 1.x.x and the engine version for Amazon DocumentDB 4.0 is 2.x.x.

How do I analyze index usage and identify unused indexes?

To identify the indexes for a given collection, run the following command:

```
db.collection.getIndexes()
```

To analyze how much indexes are being used during operations performed on the collections, the `collStats` and `indexStats` commands can be used. In order to view the total number of scans performed using indexes (index scans) compared to the number of scans performed without an index (collection scans), run the following command:

```
db.collection.stats()
```

The output for this command includes the following values:

- **idxScans** - The number of scans performed on this collection using an index.
- **collScans** - The number of scans performed on this collection without using an index. These scans would have involved looking over the documents in the collection one at a time.
- **lastReset** - The time these counters have been last reset. The statistics provided by this command are reset when starting/stopping the cluster or scaling up/down the instance.

A breakdown of how much each index is used can be found in the output of the following command. It is a best practice to regularly identify and remove unused indexes in order to improve performance and reduce cost, as it eliminates unnecessary compute, storage, and I/Os used to maintain the indexes.

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

The output from this command gives the following values for each index created on the collection:

- **ops** - The number of operations that used the index. If your workload has been running for a sufficiently long time and you are confident that your workload is in a steady state, an `ops` value of zero would indicate that the index is not used at all.
- **numDocsRead** - The number of documents read during operations using this index.
- **since** - The time since Amazon DocumentDB started collecting stats on index usage, which is typically the value since the last database restart or maintenance action.
- **size** - The size of this index in bytes.

The following example is a sample output from running the above command:

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "example-host.com:12345",
    "size" : NumberLong(...),
    "accesses" : {
```

```
        "ops" : NumberLong(...),
        "docsRead" : NumberLong(...),
        "since" : ISODate("...")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(...),
        "blksHit" : NumberLong(...),
        "hitRatio" : ...
    }
}
{
    "name" : "x_1",
    "key" : {
        "x" : 1
    },
    "host" : "example-host.com:12345",
    "size" : NumberLong(...),
    "accesses" : {
        "ops" : NumberLong(...),
        "docsRead" : NumberLong(...),
        "since" : ISODate("...")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(...),
        "blksHit" : NumberLong(...),
        "hitRatio" : ...
    }
}
```

To determine the overall index size for a collection, run the following command:

```
db.collection.stats()
```

To drop an unused index, run the following command:

```
db.collection.dropIndex("indexName")
```

How do I identify missing indexes?

You can use the Amazon DocumentDB profiler to log slow queries. A query that appears repeatedly in the slow query log may indicate that an additional index is required to improve that query's performance.

You can identify opportunities for helpful indexes by looking for long running queries that have one or more stages that perform at least one `COLLSCAN` stage, meaning that the query stage has to read every document in the collection in order to provide a response to the query.

The following example shows a query on a collection of taxi rides that ran on a large collection.

```
db.rides.count({"fare.totalAmount":{$gt:10.0}}))
```

In order to execute this example, the query had to perform a collection scan (i.e. read every single document in the collection) since there is no index on the `fare.totalAmount` field. Output from the Amazon DocumentDB profiler for this query looks something like the following:

```
{
    ...
    "cursorExhausted": true,
    "nreturned": 0,
    "responseLength": 0,
    "protocol": "op_query",
    "millis": 300679,
    "planSummary": "COLLSCAN",
    "execStats": {
        "stage": "COLLSCAN",
        "nReturned": "0",
        "executionTimeMillisEstimate": "300678.042"
    },
    "client": "172.31.5.63:53878",
    "appName": "MongoDB Shell",
    "user": "example"
}
```

To speed up the query in this example, you want to create an index on `fare.totalAmount`, as shown below.

```
db.rides.createIndex( {"fare.totalAmount": 1}, {background: true} )
```

> **Note**
>
> Indexes created in the foreground (meaning if the `{background:true}` option was not supplied when creating the index) take an exclusive write lock, which prevents applications from writing data to the collection until the index build completes. Be aware of this

potential impact when creating indexes on production clusters. When creating indexes, we recommend setting `{background:true}`.

In general, you want to create indexes on fields that have high cardinality (for example, a large number of unique values). Creating an index on a field with low cardinality can result in a large index that is not used. The Amazon DocumentDB query optimizer considers the overall size of the collection and selectivity of the indexes when creating a query plan. There are times where you will see the query processor select a `COLLSCAN` even when an index is present. This happens when the query processor estimates that utilizing the index will not yield a performance advantage over scanning the entire collection. If you want to force the query processor to utilize a particular index, you can use the `hint()` operator as shown below.

```
db.collection.find().hint("indexName")
```

Summary of useful queries

The following queries can be useful for monitoring performance and resource utilization in Amazon DocumentDB.

- Use the following command to view statistics about a specific collection, including operation counters, cache statistics, accesses statistics, and size statistics:

  ```
  db.collection.stats()
  ```

- Use the following command to view statistics about each index created on a collection including the size of the index, index-specific cache statistics, and index usage statistics:

  ```
  db.collection.aggregate([{$indexStats:{}}]).pretty()
  ```

- Use the following query to list all activity.

  ```
  db.adminCommand({currentOp: 1, $all: 1});
  ```

- The following code lists all long running or blocked queries.

  ```
  db.adminCommand({aggregate: 1,
                   pipeline: [{$currentOp: {}},
                              {$match: {$or: [{secs_running: {$gt: 10}},
  ```

```
                                                {WaitState: {$exists: true}}]}},
                              {$project: {_id:0,
                                          opid: 1,
                                          secs_running: 1,
                                          WaitState: 1,
                                          blockedOn: 1,
                                          command: 1}}],
                   cursor: {}
                  });
```

- The following code terminates a query.

```
db.adminCommand({killOp: 1, op: <opid of running or blocked query>});
```

- Use the following code to get an aggregated view of the system state.

```
db.adminCommand({aggregate: 1,
                   pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                              {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                   cursor: {}
                  });
```

Amazon DocumentDB cluster, instance, and resource management API reference

This section describes the cluster, instance, and resource management operations for Amazon DocumentDB (with MongoDB compatibility) that are accessible via HTTP, the AWS Command Line Interface (AWS CLI), or the AWS SDK. You can use these APIs to create, delete, and modify clusters and instances.

> **Important**
>
> These APIs are used only for managing clusters, instances, and related resources. For information about how to connect to a running Amazon DocumentDB cluster, see Get started guide.

Topics

- Actions
- Data Types
- Common Errors
- Common Parameters

Actions

The following actions are supported by Amazon DocumentDB (with MongoDB compatibility):

- AddSourceIdentifierToSubscription
- AddTagsToResource
- ApplyPendingMaintenanceAction
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CreateDBCluster
- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot

- CreateDBInstance
- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBClusterSnapshot
- DeleteDBInstance
- DeleteDBSubnetGroup
- DeleteEventSubscription
- DeleteGlobalCluster
- DescribeCertificates
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusters
- DescribeDBClusterSnapshotAttributes
- DescribeDBClusterSnapshots
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeDBSubnetGroups
- DescribeEngineDefaultClusterParameters
- DescribeEventCategories
- DescribeEvents
- DescribeEventSubscriptions
- DescribeGlobalClusters
- DescribeOrderableDBInstanceOptions
- DescribePendingMaintenanceActions
- FailoverDBCluster
- FailoverGlobalCluster
- ListTagsForResource

- ModifyDBCluster
- ModifyDBClusterParameterGroup
- ModifyDBClusterSnapshotAttribute
- ModifyDBInstance
- ModifyDBSubnetGroup
- ModifyEventSubscription
- ModifyGlobalCluster
- RebootDBInstance
- RemoveFromGlobalCluster
- RemoveSourceIdentifierFromSubscription
- RemoveTagsFromResource
- ResetDBClusterParameterGroup
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- StartDBCluster
- StopDBCluster
- SwitchoverGlobalCluster

The following actions are supported by Amazon DocumentDB Elastic Clusters:

- ApplyPendingMaintenanceAction
- CopyClusterSnapshot
- CreateCluster
- CreateClusterSnapshot
- DeleteCluster
- DeleteClusterSnapshot
- GetCluster
- GetClusterSnapshot
- GetPendingMaintenanceAction
- ListClusters
- ListClusterSnapshots

- ListPendingMaintenanceActions
- ListTagsForResource
- RestoreClusterFromSnapshot
- StartCluster
- StopCluster
- TagResource
- UntagResource
- UpdateCluster

Amazon DocumentDB (with MongoDB compatibility)

The following actions are supported by Amazon DocumentDB (with MongoDB compatibility):

- AddSourceIdentifierToSubscription
- AddTagsToResource
- ApplyPendingMaintenanceAction
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CreateDBCluster
- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBClusterSnapshot
- DeleteDBInstance
- DeleteDBSubnetGroup
- DeleteEventSubscription
- DeleteGlobalCluster

- DescribeCertificates
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusters
- DescribeDBClusterSnapshotAttributes
- DescribeDBClusterSnapshots
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeDBSubnetGroups
- DescribeEngineDefaultClusterParameters
- DescribeEventCategories
- DescribeEvents
- DescribeEventSubscriptions
- DescribeGlobalClusters
- DescribeOrderableDBInstanceOptions
- DescribePendingMaintenanceActions
- FailoverDBCluster
- FailoverGlobalCluster
- ListTagsForResource
- ModifyDBCluster
- ModifyDBClusterParameterGroup
- ModifyDBClusterSnapshotAttribute
- ModifyDBInstance
- ModifyDBSubnetGroup
- ModifyEventSubscription
- ModifyGlobalCluster
- RebootDBInstance
- RemoveFromGlobalCluster
- RemoveSourceIdentifierFromSubscription
- RemoveTagsFromResource

- ResetDBClusterParameterGroup
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- StartDBCluster
- StopDBCluster
- SwitchoverGlobalCluster

AddSourceIdentifierToSubscription

Service: Amazon DocumentDB (with MongoDB compatibility)

Adds a source identifier to an existing event notification subscription.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SourceIdentifier

The identifier of the event source to be added:

- If the source type is an instance, a `DBInstanceIdentifier` must be provided.
- If the source type is a security group, a `DBSecurityGroupName` must be provided.
- If the source type is a parameter group, a `DBParameterGroupName` must be provided.
- If the source type is a snapshot, a `DBSnapshotIdentifier` must be provided.

Type: String

Required: Yes

SubscriptionName

The name of the Amazon DocumentDB event notification subscription that you want to add a source identifier to.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

EventSubscription

Detailed information about an event to which you have subscribed.

Type: EventSubscription object

Errors

For information about the errors that are common to all actions, see Common Errors.

SourceNotFound

The requested source could not be found.

HTTP Status Code: 404

SubscriptionNotFound

The subscription name does not exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddTagsToResource

Service: Amazon DocumentDB (with MongoDB compatibility)

Adds metadata tags to an Amazon DocumentDB resource. You can use these tags with cost allocation reporting to track costs that are associated with Amazon DocumentDB resources or in a `Condition` statement in an AWS Identity and Access Management (IAM) policy for Amazon DocumentDB.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

ResourceName

The Amazon DocumentDB resource that the tags are added to. This value is an Amazon Resource Name .

Type: String

Required: Yes

Tags.Tag.N

The tags to be assigned to the Amazon DocumentDB resource.

Type: Array of Tag objects

Required: Yes

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

DBSnapshotNotFound

`DBSnapshotIdentifier` doesn't refer to an existing snapshot.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ApplyPendingMaintenanceAction

Service: Amazon DocumentDB (with MongoDB compatibility)

Applies a pending maintenance action to a resource (for example, to an Amazon DocumentDB instance).

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

ApplyAction

The pending maintenance action to apply to this resource.

Valid values: `system-update`, `db-upgrade`

Type: String

Required: Yes

OptInType

A value that specifies the type of opt-in request or undoes an opt-in request. An opt-in request of type `immediate` can't be undone.

Valid values:

- `immediate` - Apply the maintenance action immediately.
- `next-maintenance` - Apply the maintenance action during the next maintenance window for the resource.
- `undo-opt-in` - Cancel any existing `next-maintenance` opt-in requests.

Type: String

Required: Yes

ResourceIdentifier

The Amazon Resource Name (ARN) of the resource that the pending maintenance action applies to.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

ResourcePendingMaintenanceActions

Represents the output of ApplyPendingMaintenanceAction.

Type: ResourcePendingMaintenanceActions object

Errors

For information about the errors that are common to all actions, see Common Errors.

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

ResourceNotFoundFault

The specified resource ID was not found.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CopyDBClusterParameterGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Copies the specified cluster parameter group.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SourceDBClusterParameterGroupIdentifier

The identifier or Amazon Resource Name (ARN) for the source cluster parameter group.

Constraints:

- Must specify a valid cluster parameter group.
- If the source cluster parameter group is in the same AWS Region as the copy, specify a valid parameter group identifier; for example, `my-db-cluster-param-group`, or a valid ARN.
- If the source parameter group is in a different AWS Region than the copy, specify a valid cluster parameter group ARN; for example, `arn:aws:rds:us-east-1:123456789012:sample-cluster:sample-parameter-group`.

Type: String

Required: Yes

TargetDBClusterParameterGroupDescription

A description for the copied cluster parameter group.

Type: String

Required: Yes

TargetDBClusterParameterGroupIdentifier

The identifier for the copied cluster parameter group.

Constraints:

- Cannot be null, empty, or blank.
- Must contain from 1 to 255 letters, numbers, or hyphens.
- The first character must be a letter.

- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster-param-group1`

Type: String

Required: Yes

Tags.Tag.N

The tags that are to be assigned to the parameter group.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

DBClusterParameterGroup

Detailed information about a cluster parameter group.

Type: DBClusterParameterGroup object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupAlreadyExists

A parameter group with the same name already exists.

HTTP Status Code: 400

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

DBParameterGroupQuotaExceeded

This request would cause you to exceed the allowed number of parameter groups.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CopyDBClusterSnapshot

Service: Amazon DocumentDB (with MongoDB compatibility)

Copies a snapshot of a cluster.

To copy a cluster snapshot from a shared manual cluster snapshot, `SourceDBClusterSnapshotIdentifier` must be the Amazon Resource Name (ARN) of the shared cluster snapshot. You can only copy a shared DB cluster snapshot, whether encrypted or not, in the same AWS Region.

To cancel the copy operation after it is in progress, delete the target cluster snapshot identified by `TargetDBClusterSnapshotIdentifier` while that cluster snapshot is in the *copying* status.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SourceDBClusterSnapshotIdentifier

The identifier of the cluster snapshot to copy. This parameter is not case sensitive.

Constraints:

- Must specify a valid system snapshot in the *available* state.
- If the source snapshot is in the same AWS Region as the copy, specify a valid snapshot identifier.
- If the source snapshot is in a different AWS Region than the copy, specify a valid cluster snapshot ARN.

Example: `my-cluster-snapshot1`

Type: String

Required: Yes

TargetDBClusterSnapshotIdentifier

The identifier of the new cluster snapshot to create from the source cluster snapshot. This parameter is not case sensitive.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.

- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster-snapshot2`

Type: String

Required: Yes

CopyTags

Set to `true` to copy all tags from the source cluster snapshot to the target cluster snapshot, and otherwise `false`. The default is `false`.

Type: Boolean

Required: No

KmsKeyId

The AWS KMS key ID for an encrypted cluster snapshot. The AWS KMS key ID is the Amazon Resource Name (ARN), AWS KMS key identifier, or the AWS KMS key alias for the AWS KMS encryption key.

If you copy an encrypted cluster snapshot from your AWS account, you can specify a value for `KmsKeyId` to encrypt the copy with a new AWS KMS encryption key. If you don't specify a value for `KmsKeyId`, then the copy of the cluster snapshot is encrypted with the same AWS KMS key as the source cluster snapshot.

If you copy an encrypted cluster snapshot that is shared from another AWS account, then you must specify a value for `KmsKeyId`.

To copy an encrypted cluster snapshot to another AWS Region, set `KmsKeyId` to the AWS KMS key ID that you want to use to encrypt the copy of the cluster snapshot in the destination Region. AWS KMS encryption keys are specific to the AWS Region that they are created in, and you can't use encryption keys from one AWS Region in another AWS Region.

If you copy an unencrypted cluster snapshot and specify a value for the `KmsKeyId` parameter, an error is returned.

Type: String

Required: No

PreSignedUrl

The URL that contains a Signature Version 4 signed request for the`CopyDBClusterSnapshot` API action in the AWS Region that contains the source cluster snapshot to copy. You must use the `PreSignedUrl` parameter when copying a cluster snapshot from another AWS Region.

If you are using an AWS SDK tool or the AWS CLI, you can specify `SourceRegion` (or `--source-region` for the AWS CLI) instead of specifying `PreSignedUrl` manually. Specifying `SourceRegion` autogenerates a pre-signed URL that is a valid request for the operation that can be executed in the source AWS Region.

The presigned URL must be a valid request for the `CopyDBClusterSnapshot` API action that can be executed in the source AWS Region that contains the cluster snapshot to be copied. The presigned URL request must contain the following parameter values:

- `SourceRegion` - The ID of the region that contains the snapshot to be copied.
- `SourceDBClusterSnapshotIdentifier` - The identifier for the the encrypted cluster snapshot to be copied. This identifier must be in the Amazon Resource Name (ARN) format for the source AWS Region. For example, if you are copying an encrypted cluster snapshot from the us-east-1 AWS Region, then your `SourceDBClusterSnapshotIdentifier` looks something like the following: `arn:aws:rds:us-east-1:12345678012:sample-cluster:sample-cluster-snapshot`.
- `TargetDBClusterSnapshotIdentifier` - The identifier for the new cluster snapshot to be created. This parameter isn't case sensitive.

Type: String

Required: No

Tags.Tag.N

The tags to be assigned to the cluster snapshot.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

DBClusterSnapshot

Detailed information about a cluster snapshot.

Type: DBClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterSnapshotAlreadyExistsFault

You already have a cluster snapshot with the given identifier.

HTTP Status Code: 400

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

KMSKeyNotAccessibleFault

An error occurred when accessing an AWS KMS key.

HTTP Status Code: 400

SnapshotQuotaExceeded

The request would cause you to exceed the allowed number of snapshots.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateDBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates a new Amazon DocumentDB cluster.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The cluster identifier. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster`

Type: String

Required: Yes

Engine

The name of the database engine to be used for this cluster.

Valid values: `docdb`

Type: String

Required: Yes

AvailabilityZones.AvailabilityZone.N

A list of Amazon EC2 Availability Zones that instances in the cluster can be created in.

Type: Array of strings

Required: No

BackupRetentionPeriod

The number of days for which automated backups are retained. You must specify a minimum value of 1.

Default: 1

Constraints:

- Must be a value from 1 to 35.

Type: Integer

Required: No

DBClusterParameterGroupName

The name of the cluster parameter group to associate with this cluster.

Type: String

Required: No

DBSubnetGroupName

A subnet group to associate with this cluster.

Constraints: Must match the name of an existing `DBSubnetGroup`. Must not be default.

Example: `mySubnetgroup`

Type: String

Required: No

DeletionProtection

Specifies whether this cluster can be deleted. If `DeletionProtection` is enabled, the cluster cannot be deleted unless it is modified and `DeletionProtection` is disabled. `DeletionProtection` protects clusters from being accidentally deleted.

Type: Boolean

Required: No

EnableCloudwatchLogsExports.member.N

A list of log types that need to be enabled for exporting to Amazon CloudWatch Logs. You can enable audit logs or profiler logs. For more information, see Auditing Amazon DocumentDB Events and Profiling Amazon DocumentDB Operations.

Type: Array of strings

Required: No

EngineVersion

The version number of the database engine to use. The `--engine-version` will default to the latest major engine version. For production workloads, we recommend explicitly declaring this parameter with the intended major engine version.

Type: String

Required: No

GlobalClusterIdentifier

The cluster identifier of the new global cluster.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: No

KmsKeyId

The AWS KMS key identifier for an encrypted cluster.

The AWS KMS key identifier is the Amazon Resource Name (ARN) for the AWS KMS encryption key. If you are creating a cluster using the same AWS account that owns the AWS KMS encryption key that is used to encrypt the new cluster, you can use the AWS KMS key alias instead of the ARN for the AWS KMS encryption key.

If an encryption key is not specified in `KmsKeyId`:

- If the `StorageEncrypted` parameter is `true`, Amazon DocumentDB uses your default encryption key.

AWS KMS creates the default encryption key for your AWS account. Your AWS account has a different default encryption key for each AWS Regions.

Type: String

Required: No

ManageMasterUserPassword

Specifies whether to manage the master user password with Amazon Web Services Secrets Manager.

Constraint: You can't manage the master user password with Amazon Web Services Secrets Manager if `MasterUserPassword` is specified.

Type: Boolean

Required: No

MasterUsername

The name of the master user for the cluster.

Constraints:

- Must be from 1 to 63 letters or numbers.
- The first character must be a letter.
- Cannot be a reserved word for the chosen database engine.

Type: String

Required: No

MasterUserPassword

The password for the master database user. This password can contain any printable ASCII character except forward slash (/), double quote ("), or the "at" symbol (@).

Constraints: Must contain from 8 to 100 characters.

Type: String

Required: No

MasterUserSecretKmsKeyId

The Amazon Web Services KMS key identifier to encrypt a secret that is automatically generated and managed in Amazon Web Services Secrets Manager. This setting is valid only if the master user password is managed by Amazon DocumentDB in Amazon Web Services Secrets Manager for the DB cluster.

The Amazon Web Services KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different Amazon Web Services account, specify the key ARN or alias ARN.

If you don't specify `MasterUserSecretKmsKeyId`, then the `aws/secretsmanager` KMS key is used to encrypt the secret. If the secret is in a different Amazon Web Services account, then you can't use the `aws/secretsmanager` KMS key to encrypt the secret, and you must use a customer managed KMS key.

There is a default KMS key for your Amazon Web Services account. Your Amazon Web Services account has a different default KMS key for each Amazon Web Services Region.

Type: String

Required: No

Port

The port number on which the instances in the cluster accept connections.

Type: Integer

Required: No

PreferredBackupWindow

The daily time range during which automated backups are created if automated backups are enabled using the `BackupRetentionPeriod` parameter.

The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region.

Constraints:

- Must be in the format `hh24:mi-hh24:mi`.
- Must be in Universal Coordinated Time (UTC).

- Must not conflict with the preferred maintenance window.
- Must be at least 30 minutes.

Type: String

Required: No

PreferredMaintenanceWindow

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region, occurring on a random day of the week.

Valid days: Mon, Tue, Wed, Thu, Fri, Sat, Sun

Constraints: Minimum 30-minute window.

Type: String

Required: No

PreSignedUrl

Not currently supported.

Type: String

Required: No

StorageEncrypted

Specifies whether the cluster is encrypted.

Type: Boolean

Required: No

StorageType

The storage type to associate with the DB cluster.

For information on storage types for Amazon DocumentDB clusters, see Cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Valid values for storage type - `standard | iopt1`

Default value is `standard`

> **Note**
>
> When you create a DocumentDB DB cluster with the storage type set to `iopt1`, the storage type is returned in the response. The storage type isn't returned when you set it to `standard`.

Type: String

Required: No

Tags.Tag.N

The tags to be assigned to the cluster.

Type: Array of Tag objects

Required: No

VpcSecurityGroupIds.VpcSecurityGroupId.N

A list of EC2 VPC security groups to associate with this cluster.

Type: Array of strings

Required: No

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterAlreadyExistsFault

You already have a cluster with the given identifier.

HTTP Status Code: 400

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBClusterParameterGroupNotFound

`DBClusterParameterGroupName` doesn't refer to an existing cluster parameter group.

HTTP Status Code: 404

DBClusterQuotaExceededFault

The cluster can't be created because you have reached the maximum allowed quota of clusters.

HTTP Status Code: 403

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

DBSubnetGroupDoesNotCoverEnoughAZs

Subnets in the subnet group should cover at least two Availability Zones unless there is only one Availability Zone.

HTTP Status Code: 400

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

InsufficientStorageClusterCapacity

There is not enough storage available for the current action. You might be able to resolve this error by updating your subnet group to use different Availability Zones that have more storage available.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

InvalidDBSubnetGroupStateFault

The subnet group can't be deleted because it's in use.

HTTP Status Code: 400

InvalidGlobalClusterStateFault

The requested operation can't be performed while the cluster is in this state.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

InvalidVPCNetworkStateFault

The subnet group doesn't cover all Availability Zones after it is created because of changes that were made.

HTTP Status Code: 400

KMSKeyNotAccessibleFault

An error occurred when accessing an AWS KMS key.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateDBClusterParameterGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates a new cluster parameter group.

Parameters in a cluster parameter group apply to all of the instances in a cluster.

A cluster parameter group is initially created with the default parameters for the database engine used by instances in the cluster. In Amazon DocumentDB, you cannot make modifications directly to the `default.docdb3.6` cluster parameter group. If your Amazon DocumentDB cluster is using the default cluster parameter group and you want to modify a value in it, you must first create a new parameter group or copy an existing parameter group, modify it, and then apply the modified parameter group to your cluster. For the new cluster parameter group and associated settings to take effect, you must then reboot the instances in the cluster without failover. For more information, see Modifying Amazon DocumentDB Cluster Parameter Groups.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterParameterGroupName

The name of the cluster parameter group.

Constraints:

- Must not match the name of an existing `DBClusterParameterGroup`.

> **Note**
>
> This value is stored as a lowercase string.

Type: String

Required: Yes

DBParameterGroupFamily

The cluster parameter group family name.

Type: String

Required: Yes

Description

The description for the cluster parameter group.

Type: String

Required: Yes

Tags.Tag.N

The tags to be assigned to the cluster parameter group.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

DBClusterParameterGroup

Detailed information about a cluster parameter group.

Type: DBClusterParameterGroup object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupAlreadyExists

A parameter group with the same name already exists.

HTTP Status Code: 400

DBParameterGroupQuotaExceeded

This request would cause you to exceed the allowed number of parameter groups.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateDBClusterSnapshot

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates a snapshot of a cluster.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The identifier of the cluster to create a snapshot for. This parameter is not case sensitive.

Constraints:

- Must match the identifier of an existing `DBCluster`.

Example: `my-cluster`

Type: String

Required: Yes

DBClusterSnapshotIdentifier

The identifier of the cluster snapshot. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster-snapshot1`

Type: String

Required: Yes

Tags.Tag.N

The tags to be assigned to the cluster snapshot.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

DBClusterSnapshot

Detailed information about a cluster snapshot.

Type: DBClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBClusterSnapshotAlreadyExistsFault

You already have a cluster snapshot with the given identifier.

HTTP Status Code: 400

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

SnapshotQuotaExceeded

The request would cause you to exceed the allowed number of snapshots.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateDBInstance

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates a new instance.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The identifier of the cluster that the instance will belong to.

Type: String

Required: Yes

DBInstanceClass

The compute and memory capacity of the instance; for example, `db.r5.large`.

Type: String

Required: Yes

DBInstanceIdentifier

The instance identifier. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `mydbinstance`

Type: String

Required: Yes

Engine

The name of the database engine to be used for this instance.

Valid value: `docdb`

Type: String

Required: Yes

AutoMinorVersionUpgrade

This parameter does not apply to Amazon DocumentDB. Amazon DocumentDB does not perform minor version upgrades regardless of the value set.

Default: `false`

Type: Boolean

Required: No

AvailabilityZone

The Amazon EC2 Availability Zone that the instance is created in.

Default: A random, system-chosen Availability Zone in the endpoint's AWS Region.

Example: `us-east-1d`

Type: String

Required: No

CACertificateIdentifier

The CA certificate identifier to use for the DB instance's server certificate.

For more information, see Updating Your Amazon DocumentDB TLS Certificates and Encrypting Data in Transit in the *Amazon DocumentDB Developer Guide*.

Type: String

Required: No

CopyTagsToSnapshot

A value that indicates whether to copy tags from the DB instance to snapshots of the DB instance. By default, tags are not copied.

Type: Boolean

Required: No

EnablePerformanceInsights

A value that indicates whether to enable Performance Insights for the DB Instance. For more information, see Using Amazon Performance Insights.

Type: Boolean

Required: No

PerformanceInsightsKMSKeyId

The AWS KMS key identifier for encryption of Performance Insights data.

The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key.

If you do not specify a value for PerformanceInsightsKMSKeyId, then Amazon DocumentDB uses your default KMS key. There is a default KMS key for your Amazon Web Services account. Your Amazon Web Services account has a different default KMS key for each Amazon Web Services region.

Type: String

Required: No

PreferredMaintenanceWindow

The time range each week during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region, occurring on a random day of the week.

Valid days: Mon, Tue, Wed, Thu, Fri, Sat, Sun

Constraints: Minimum 30-minute window.

Type: String

Required: No

PromotionTier

A value that specifies the order in which an Amazon DocumentDB replica is promoted to the primary instance after a failure of the existing primary instance.

Default: 1

Valid values: 0-15

Type: Integer

Required: No

Tags.Tag.N

The tags to be assigned to the instance. You can assign up to 10 tags to an instance.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

DBInstance

Detailed information about an instance.

Type: DBInstance object

Errors

For information about the errors that are common to all actions, see Common Errors.

AuthorizationNotFound

The specified CIDR IP or Amazon EC2 security group isn't authorized for the specified security group.

Amazon DocumentDB also might not be authorized to perform necessary actions on your behalf using IAM.

HTTP Status Code: 404

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBInstanceAlreadyExists

You already have a instance with the given identifier.

HTTP Status Code: 400

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

DBSecurityGroupNotFound

`DBSecurityGroupName` doesn't refer to an existing security group.

HTTP Status Code: 404

DBSubnetGroupDoesNotCoverEnoughAZs

Subnets in the subnet group should cover at least two Availability Zones unless there is only one Availability Zone.

HTTP Status Code: 400

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

InstanceQuotaExceeded

The request would cause you to exceed the allowed number of instances.

HTTP Status Code: 400

InsufficientDBInstanceCapacity

The specified instance class isn't available in the specified Availability Zone.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

InvalidVPCNetworkStateFault

The subnet group doesn't cover all Availability Zones after it is created because of changes that were made.

HTTP Status Code: 400

KMSKeyNotAccessibleFault

An error occurred when accessing an AWS KMS key.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

StorageTypeNotSupported

Storage of the specified `StorageType` can't be associated with the DB instance.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateDBSubnetGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates a new subnet group. subnet groups must contain at least one subnet in at least two Availability Zones in the AWS Region.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBSubnetGroupDescription

The description for the subnet group.

Type: String

Required: Yes

DBSubnetGroupName

The name for the subnet group. This value is stored as a lowercase string.

Constraints: Must contain no more than 255 letters, numbers, periods, underscores, spaces, or hyphens. Must not be default.

Example: `mySubnetgroup`

Type: String

Required: Yes

SubnetIds.SubnetIdentifier.N

The Amazon EC2 subnet IDs for the subnet group.

Type: Array of strings

Required: Yes

Tags.Tag.N

The tags to be assigned to the subnet group.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

DBSubnetGroup

Detailed information about a subnet group.

Type: DBSubnetGroup object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBSubnetGroupAlreadyExists

`DBSubnetGroupName` is already being used by an existing subnet group.

HTTP Status Code: 400

DBSubnetGroupDoesNotCoverEnoughAZs

Subnets in the subnet group should cover at least two Availability Zones unless there is only one Availability Zone.

HTTP Status Code: 400

DBSubnetGroupQuotaExceeded

The request would cause you to exceed the allowed number of subnet groups.

HTTP Status Code: 400

DBSubnetQuotaExceededFault

The request would cause you to exceed the allowed number of subnets in a subnet group.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateEventSubscription

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates an Amazon DocumentDB event notification subscription. This action requires a topic Amazon Resource Name (ARN) created by using the Amazon DocumentDB console, the Amazon SNS console, or the Amazon SNS API. To obtain an ARN with Amazon SNS, you must create a topic in Amazon SNS and subscribe to the topic. The ARN is displayed in the Amazon SNS console.

You can specify the type of source (`SourceType`) that you want to be notified of. You can also provide a list of Amazon DocumentDB sources (`SourceIds`) that trigger the events, and you can provide a list of event categories (`EventCategories`) for events that you want to be notified of. For example, you can specify `SourceType = db-instance`, `SourceIds = mydbinstance1, mydbinstance2` and `EventCategories = Availability, Backup`.

If you specify both the `SourceType` and `SourceIds` (such as `SourceType = db-instance` and `SourceIdentifier = myDBInstance1`), you are notified of all the `db-instance` events for the specified source. If you specify a `SourceType` but do not specify a `SourceIdentifier`, you receive notice of the events for that source type for all your Amazon DocumentDB sources. If you do not specify either the `SourceType` or the `SourceIdentifier`, you are notified of events generated from all Amazon DocumentDB sources belonging to your customer account.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SnsTopicArn

The Amazon Resource Name (ARN) of the SNS topic created for event notification. Amazon SNS creates the ARN when you create a topic and subscribe to it.

Type: String

Required: Yes

SubscriptionName

The name of the subscription.

Constraints: The name must be fewer than 255 characters.

Type: String

Required: Yes

Enabled

A Boolean value; set to `true` to activate the subscription, set to `false` to create the subscription but not active it.

Type: Boolean

Required: No

EventCategories.EventCategory.N

A list of event categories for a `SourceType` that you want to subscribe to.

Type: Array of strings

Required: No

SourceIds.SourceId.N

The list of identifiers of the event sources for which events are returned. If not specified, then all sources are included in the response. An identifier must begin with a letter and must contain only ASCII letters, digits, and hyphens; it can't end with a hyphen or contain two consecutive hyphens.

Constraints:

- If `SourceIds` are provided, `SourceType` must also be provided.
- If the source type is an instance, a `DBInstanceIdentifier` must be provided.
- If the source type is a security group, a `DBSecurityGroupName` must be provided.
- If the source type is a parameter group, a `DBParameterGroupName` must be provided.
- If the source type is a snapshot, a `DBSnapshotIdentifier` must be provided.

Type: Array of strings

Required: No

SourceType

The type of source that is generating the events. For example, if you want to be notified of events generated by an instance, you would set this parameter to `db-instance`. If this value is not specified, all events are returned.

Valid values: `db-instance`, `db-cluster`, `db-parameter-group`, `db-security-group`, `db-cluster-snapshot`

Type: String

Required: No

Tags.Tag.N

The tags to be assigned to the event subscription.

Type: Array of Tag objects

Required: No

Response Elements

The following element is returned by the service.

EventSubscription

Detailed information about an event to which you have subscribed.

Type: EventSubscription object

Errors

For information about the errors that are common to all actions, see Common Errors.

EventSubscriptionQuotaExceeded

You have reached the maximum number of event subscriptions.

HTTP Status Code: 400

SNSInvalidTopic

Amazon SNS has responded that there is a problem with the specified topic.

HTTP Status Code: 400

SNSNoAuthorization

You do not have permission to publish to the SNS topic Amazon Resource Name (ARN).

HTTP Status Code: 400

SNSTopicArnNotFound

The SNS topic Amazon Resource Name (ARN) does not exist.

HTTP Status Code: 404

SourceNotFound

The requested source could not be found.

HTTP Status Code: 404

SubscriptionAlreadyExist

The provided subscription name already exists.

HTTP Status Code: 400

SubscriptionCategoryNotFound

The provided category does not exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateGlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates an Amazon DocumentDB global cluster that can span multiple multiple AWS Regions. The global cluster contains one primary cluster with read-write capability, and up-to give read-only secondary clusters. Global clusters uses storage-based fast replication across regions with latencies less than one second, using dedicated infrastructure with no impact to your workload's performance.

You can create a global cluster that is initially empty, and then add a primary and a secondary to it. Or you can specify an existing cluster during the create operation, and this cluster becomes the primary of the global cluster.

> **Note**
>
> This action only applies to Amazon DocumentDB clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

GlobalClusterIdentifier

The cluster identifier of the new global cluster.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

DatabaseName

The name for your database of up to 64 alpha-numeric characters. If you do not provide a name, Amazon DocumentDB will not create a database in the global cluster you are creating.

Type: String

Required: No

DeletionProtection

The deletion protection setting for the new global cluster. The global cluster can't be deleted when deletion protection is enabled.

Type: Boolean

Required: No

Engine

The name of the database engine to be used for this cluster.

Type: String

Required: No

EngineVersion

The engine version of the global cluster.

Type: String

Required: No

SourceDBClusterIdentifier

The Amazon Resource Name (ARN) to use as the primary cluster of the global cluster. This parameter is optional.

Type: String

Required: No

StorageEncrypted

The storage encryption setting for the new global cluster.

Type: Boolean

Required: No

Response Elements

The following element is returned by the service.

GlobalCluster

A data type representing an Amazon DocumentDB global cluster.

Type: GlobalCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

GlobalClusterAlreadyExistsFault

The `GlobalClusterIdentifier` already exists. Choose a new global cluster identifier (unique name) to create a new global cluster.

HTTP Status Code: 400

GlobalClusterQuotaExceededFault

The number of global clusters for this account is already at the maximum allowed.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteDBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes a previously provisioned cluster. When you delete a cluster, all automated backups for that cluster are deleted and can't be recovered. Manual DB cluster snapshots of the specified cluster are not deleted.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The cluster identifier for the cluster to be deleted. This parameter isn't case sensitive.

Constraints:

- Must match an existing `DBClusterIdentifier`.

Type: String

Required: Yes

FinalDBSnapshotIdentifier

The cluster snapshot identifier of the new cluster snapshot created when `SkipFinalSnapshot` is set to `false`.

> **Note**
>
> Specifying this parameter and also setting the `SkipFinalShapshot` parameter to `true` results in an error.

Constraints:

- Must be from 1 to 255 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Type: String

Required: No

SkipFinalSnapshot

Determines whether a final cluster snapshot is created before the cluster is deleted. If `true` is specified, no cluster snapshot is created. If `false` is specified, a cluster snapshot is created before the DB cluster is deleted.

> **Note**
>
> If `SkipFinalSnapshot` is `false`, you must specify a `FinalDBSnapshotIdentifier` parameter.

Default: `false`

Type: Boolean

Required: No

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBClusterSnapshotAlreadyExistsFault

You already have a cluster snapshot with the given identifier.

HTTP Status Code: 400

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

SnapshotQuotaExceeded

The request would cause you to exceed the allowed number of snapshots.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteDBClusterParameterGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes a specified cluster parameter group. The cluster parameter group to be deleted can't be associated with any clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterParameterGroupName

The name of the cluster parameter group.

Constraints:

- Must be the name of an existing cluster parameter group.
- You can't delete a default cluster parameter group.
- Cannot be associated with any clusters.

Type: String

Required: Yes

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

InvalidDBParameterGroupState

The parameter group is in use, or it is in a state that is not valid. If you are trying to delete the parameter group, you can't delete it when the parameter group is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteDBClusterSnapshot

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes a cluster snapshot. If the snapshot is being copied, the copy operation is terminated.

> **Note**
>
> The cluster snapshot must be in the `available` state to be deleted.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterSnapshotIdentifier

The identifier of the cluster snapshot to delete.

Constraints: Must be the name of an existing cluster snapshot in the `available` state.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

DBClusterSnapshot

Detailed information about a cluster snapshot.

Type: DBClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteDBInstance

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes a previously provisioned instance.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBInstanceIdentifier

The instance identifier for the instance to be deleted. This parameter isn't case sensitive.

Constraints:

- Must match the name of an existing instance.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

DBInstance

Detailed information about an instance.

Type: DBInstance object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

DBSnapshotAlreadyExists

`DBSnapshotIdentifier` is already being used by an existing snapshot.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

SnapshotQuotaExceeded

The request would cause you to exceed the allowed number of snapshots.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteDBSubnetGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes a subnet group.

> **Note**
>
> The specified database subnet group must not be associated with any DB instances.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBSubnetGroupName

The name of the database subnet group to delete.

> **Note**
>
> You can't delete the default subnet group.

Constraints:

Must match the name of an existing `DBSubnetGroup`. Must not be default.

Example: `mySubnetgroup`

Type: String

Required: Yes

Errors

For information about the errors that are common to all actions, see Common Errors.

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

InvalidDBSubnetGroupStateFault

The subnet group can't be deleted because it's in use.

HTTP Status Code: 400

InvalidDBSubnetStateFault

The subnet isn't in the *available* state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteEventSubscription

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes an Amazon DocumentDB event notification subscription.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SubscriptionName

The name of the Amazon DocumentDB event notification subscription that you want to delete.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

EventSubscription

Detailed information about an event to which you have subscribed.

Type: EventSubscription object

Errors

For information about the errors that are common to all actions, see Common Errors.

InvalidEventSubscriptionState

Someone else might be modifying a subscription. Wait a few seconds, and try again.

HTTP Status Code: 400

SubscriptionNotFound

The subscription name does not exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteGlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Deletes a global cluster. The primary and secondary clusters must already be detached or deleted before attempting to delete a global cluster.

> **Note**
>
> This action only applies to Amazon DocumentDB clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

GlobalClusterIdentifier

The cluster identifier of the global cluster being deleted.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

Response Elements

The following element is returned by the service.

GlobalCluster

A data type representing an Amazon DocumentDB global cluster.

Type: GlobalCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

InvalidGlobalClusterStateFault

The requested operation can't be performed while the cluster is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeCertificates

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of certificate authority (CA) certificates provided by Amazon DocumentDB for this AWS account.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

CertificateIdentifier

The user-supplied certificate identifier. If this parameter is specified, information for only the specified certificate is returned. If this parameter is omitted, a list of up to `MaxRecords` certificates is returned. This parameter is not case sensitive.

Constraints

- Must match an existing `CertificateIdentifier`.

Type: String

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous `DescribeCertificates` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints:

- Minimum: 20
- Maximum: 100

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

Certificates.Certificate.N

A list of certificates for this AWS account.

Type: Array of Certificate objects

Marker

An optional pagination token provided if the number of records retrieved is greater than `MaxRecords`. If this parameter is specified, the marker specifies the next record in the list. Including the value of `Marker` in the next call to `DescribeCertificates` results in the next page of certificates.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

CertificateNotFound

`CertificateIdentifier` doesn't refer to an existing certificate.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBClusterParameterGroups

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of `DBClusterParameterGroup` descriptions. If a `DBClusterParameterGroupName` parameter is specified, the list contains only the description of the specified cluster parameter group.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterParameterGroupName

The name of a specific cluster parameter group to return details for.

Constraints:

- If provided, must match the name of an existing `DBClusterParameterGroup`.

Type: String

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

DBClusterParameterGroups.DBClusterParameterGroup.N

A list of cluster parameter groups.

Type: Array of DBClusterParameterGroup objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBClusterParameters

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns the detailed parameter list for a particular cluster parameter group.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterParameterGroupName

The name of a specific cluster parameter group to return parameter details for.

Constraints:

- If provided, must match the name of an existing `DBClusterParameterGroup`.

Type: String

Required: Yes

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Source

A value that indicates to return only parameters for a specific source. Parameter sources can be `engine`, `service`, or `customer`.

Type: String

Required: No

Response Elements

The following elements are returned by the service.

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Parameters.Parameter.N

Provides a list of parameters for the cluster parameter group.

Type: Array of Parameter objects

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBClusters

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns information about provisioned Amazon DocumentDB clusters. This API operation supports pagination. For certain management features such as cluster and instance lifecycle management, Amazon DocumentDB leverages operational technology that is shared with Amazon RDS and Amazon Neptune. Use the `filterName=engine,Values=docdb` filter parameter to return only Amazon DocumentDB clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The user-provided cluster identifier. If this parameter is specified, information from only the specific cluster is returned. This parameter isn't case sensitive.

Constraints:

- If provided, must match an existing `DBClusterIdentifier`.

Type: String

Required: No

Filters.Filter.N

A filter that specifies one or more clusters to describe.

Supported filters:

- `db-cluster-id` - Accepts cluster identifiers and cluster Amazon Resource Names (ARNs). The results list only includes information about the clusters identified by these ARNs.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

DBClusters.DBCluster.N

A list of clusters.

Type: Array of DBCluster objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBClusterSnapshotAttributes

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of cluster snapshot attribute names and values for a manual DB cluster snapshot.

When you share snapshots with other AWS accounts, `DescribeDBClusterSnapshotAttributes` returns the `restore` attribute and a list of IDs for the AWS accounts that are authorized to copy or restore the manual cluster snapshot. If `all` is included in the list of values for the `restore` attribute, then the manual cluster snapshot is public and can be copied or restored by all AWS accounts.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterSnapshotIdentifier

The identifier for the cluster snapshot to describe the attributes for.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

DBClusterSnapshotAttributesResult

Detailed information about the attributes that are associated with a cluster snapshot.

Type: DBClusterSnapshotAttributesResult object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBClusterSnapshots

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns information about cluster snapshots. This API operation supports pagination.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The ID of the cluster to retrieve the list of cluster snapshots for. This parameter can't be used with the `DBClusterSnapshotIdentifier` parameter. This parameter is not case sensitive.

Constraints:

- If provided, must match the identifier of an existing `DBCluster`.

Type: String

Required: No

DBClusterSnapshotIdentifier

A specific cluster snapshot identifier to describe. This parameter can't be used with the `DBClusterIdentifier` parameter. This value is stored as a lowercase string.

Constraints:

- If provided, must match the identifier of an existing `DBClusterSnapshot`.
- If this identifier is for an automated snapshot, the `SnapshotType` parameter must also be specified.

Type: String

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

IncludePublic

Set to `true` to include manual cluster snapshots that are public and can be copied or restored by any AWS account, and otherwise `false`. The default is `false`.

Type: Boolean

Required: No

IncludeShared

Set to `true` to include shared manual cluster snapshots from other AWS accounts that this AWS account has been given permission to copy or restore, and otherwise `false`. The default is `false`.

Type: Boolean

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

SnapshotType

The type of cluster snapshots to be returned. You can specify one of the following values:

- `automated` - Return all cluster snapshots that Amazon DocumentDB has automatically created for your AWS account.
- `manual` - Return all cluster snapshots that you have manually created for your AWS account.
- `shared` - Return all manual cluster snapshots that have been shared to your AWS account.
- `public` - Return all cluster snapshots that have been marked as public.

If you don't specify a `SnapshotType` value, then both automated and manual cluster snapshots are returned. You can include shared cluster snapshots with these results by setting the `IncludeShared` parameter to `true`. You can include public cluster snapshots with these results by setting the`IncludePublic` parameter to `true`.

The `IncludeShared` and `IncludePublic` parameters don't apply for `SnapshotType` values of `manual` or `automated`. The `IncludePublic` parameter doesn't apply when `SnapshotType` is set to `shared`. The `IncludeShared` parameter doesn't apply when `SnapshotType` is set to `public`.

Type: String

Required: No

Response Elements

The following elements are returned by the service.

DBClusterSnapshots.DBClusterSnapshot.N

Provides a list of cluster snapshots.

Type: Array of DBClusterSnapshot objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBEngineVersions

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of the available engines.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBParameterGroupFamily

The name of a specific parameter group family to return details for.

Constraints:

- If provided, must match an existing `DBParameterGroupFamily`.

Type: String

Required: No

DefaultOnly

Indicates that only the default version of the specified engine or engine and major version combination is returned.

Type: Boolean

Required: No

Engine

The database engine to return.

Type: String

Required: No

EngineVersion

The database engine version to return.

Example: `3.6.0`

Type: String

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

ListSupportedCharacterSets

If this parameter is specified and the requested engine supports the `CharacterSetName` parameter for `CreateDBInstance`, the response includes a list of supported character sets for each engine version.

Type: Boolean

Required: No

ListSupportedTimezones

If this parameter is specified and the requested engine supports the `TimeZone` parameter for `CreateDBInstance`, the response includes a list of supported time zones for each engine version.

Type: Boolean

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

DBEngineVersions.DBEngineVersion.N

Detailed information about one or more engine versions.

Type: Array of DBEngineVersion objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBInstances

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns information about provisioned Amazon DocumentDB instances. This API supports pagination.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBInstanceIdentifier

The user-provided instance identifier. If this parameter is specified, information from only the specific instance is returned. This parameter isn't case sensitive.

Constraints:

- If provided, must match the identifier of an existing `DBInstance`.

Type: String

Required: No

Filters.Filter.N

A filter that specifies one or more instances to describe.

Supported filters:

- `db-cluster-id` - Accepts cluster identifiers and cluster Amazon Resource Names (ARNs). The results list includes only the information about the instances that are associated with the clusters that are identified by these ARNs.
- `db-instance-id` - Accepts instance identifiers and instance ARNs. The results list includes only the information about the instances that are identified by these ARNs.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

DBInstances.DBInstance.N

Detailed information about one or more instances.

Type: Array of DBInstance objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDBSubnetGroups

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of `DBSubnetGroup` descriptions. If a `DBSubnetGroupName` is specified, the list will contain only the descriptions of the specified `DBSubnetGroup`.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBSubnetGroupName

The name of the subnet group to return details for.

Type: String

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

DBSubnetGroups.DBSubnetGroup.N

Detailed information about one or more subnet groups.

Type: Array of DBSubnetGroup objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2

- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeEngineDefaultClusterParameters

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns the default engine and system parameter information for the cluster database engine.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBParameterGroupFamily

The name of the cluster parameter group family to return the engine parameter information for.

Type: String

Required: Yes

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Response Elements

The following element is returned by the service.

EngineDefaults

Contains the result of a successful invocation of the `DescribeEngineDefaultClusterParameters` operation.

Type: EngineDefaults object

Errors

For information about the errors that are common to all actions, see Common Errors.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeEventCategories

Service: Amazon DocumentDB (with MongoDB compatibility)

Displays a list of categories for all event source types, or, if specified, for a specified source type.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

SourceType

The type of source that is generating the events.

Valid values: `db-instance`, `db-parameter-group`, `db-security-group`

Type: String

Required: No

Response Elements

The following element is returned by the service.

EventCategoriesMapList.EventCategoriesMap.N

A list of event category maps.

Type: Array of EventCategoriesMap objects

Errors

For information about the errors that are common to all actions, see Common Errors.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeEvents

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns events related to instances, security groups, snapshots, and DB parameter groups for the past 14 days. You can obtain events specific to a particular DB instance, security group, snapshot, or parameter group by providing the name as a parameter. By default, the events of the past hour are returned.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

Duration

The number of minutes to retrieve events for.

Default: 60

Type: Integer

Required: No

EndTime

The end of the time interval for which to retrieve events, specified in ISO 8601 format.

Example: 2009-07-08T18:00Z

Type: Timestamp

Required: No

EventCategories.EventCategory.N

A list of event categories that trigger notifications for an event notification subscription.

Type: Array of strings

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

SourceIdentifier

The identifier of the event source for which events are returned. If not specified, then all sources are included in the response.

Constraints:

- If `SourceIdentifier` is provided, `SourceType` must also be provided.
- If the source type is `DBInstance`, a `DBInstanceIdentifier` must be provided.
- If the source type is `DBSecurityGroup`, a `DBSecurityGroupName` must be provided.
- If the source type is `DBParameterGroup`, a `DBParameterGroupName` must be provided.
- If the source type is `DBSnapshot`, a `DBSnapshotIdentifier` must be provided.
- Cannot end with a hyphen or contain two consecutive hyphens.

Type: String

Required: No

SourceType

The event source to retrieve events for. If no value is specified, all events are returned.

Type: String

Valid Values: `db-instance | db-parameter-group | db-security-group | db-snapshot | db-cluster | db-cluster-snapshot`

Required: No

StartTime

The beginning of the time interval to retrieve events for, specified in ISO 8601 format.

Example: 2009-07-08T18:00Z

Type: Timestamp

Required: No

Response Elements

The following elements are returned by the service.

Events.Event.N

Detailed information about one or more events.

Type: Array of Event objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeEventSubscriptions

Service: Amazon DocumentDB (with MongoDB compatibility)

Lists all the subscription descriptions for a customer account. The description for a subscription includes `SubscriptionName`, `SNSTopicARN`, `CustomerID`, `SourceType`, `SourceID`, `CreationTime`, and `Status`.

If you specify a `SubscriptionName`, lists the description for that subscription.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

SubscriptionName

The name of the Amazon DocumentDB event notification subscription that you want to describe.

Type: String

Required: No

Response Elements

The following elements are returned by the service.

EventSubscriptionsList.EventSubscription.N

A list of event subscriptions.

Type: Array of EventSubscription objects

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

SubscriptionNotFound

The subscription name does not exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeGlobalClusters

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns information about Amazon DocumentDB global clusters. This API supports pagination.

> **Note**
>
> This action only applies to Amazon DocumentDB clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

Filters.Filter.N

A filter that specifies one or more global DB clusters to describe.

Supported filters: `db-cluster-id` accepts cluster identifiers and cluster Amazon Resource Names (ARNs). The results list will only include information about the clusters identified by these ARNs.

Type: Array of Filter objects

Required: No

GlobalClusterIdentifier

The user-supplied cluster identifier. If this parameter is specified, information from only the specific cluster is returned. This parameter isn't case-sensitive.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: No

Marker

An optional pagination token provided by a previous `DescribeGlobalClusters` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that you can retrieve the remaining results.

Type: Integer

Required: No

Response Elements

The following elements are returned by the service.

GlobalClusters.GlobalClusterMember.N

Type: Array of GlobalCluster objects

Marker

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeOrderableDBInstanceOptions

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of orderable instance options for the specified engine.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

Engine

The name of the engine to retrieve instance options for.

Type: String

Required: Yes

DBInstanceClass

The instance class filter value. Specify this parameter to show only the available offerings that match the specified instance class.

Type: String

Required: No

EngineVersion

The engine version filter value. Specify this parameter to show only the available offerings that match the specified engine version.

Type: String

Required: No

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

LicenseModel

The license model filter value. Specify this parameter to show only the available offerings that match the specified license model.

Type: String

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

Vpc

The virtual private cloud (VPC) filter value. Specify this parameter to show only the available VPC or non-VPC offerings.

Type: Boolean

Required: No

Response Elements

The following elements are returned by the service.

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

OrderableDBInstanceOptions.OrderableDBInstanceOption.N

The options that are available for a particular orderable instance.

Type: Array of OrderableDBInstanceOption objects

Errors

For information about the errors that are common to all actions, see Common Errors.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribePendingMaintenanceActions

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns a list of resources (for example, instances) that have at least one pending maintenance action.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

Filters.Filter.N

A filter that specifies one or more resources to return pending maintenance actions for.

Supported filters:

- `db-cluster-id` - Accepts cluster identifiers and cluster Amazon Resource Names (ARNs). The results list includes only pending maintenance actions for the clusters identified by these ARNs.
- `db-instance-id` - Accepts instance identifiers and instance ARNs. The results list includes only pending maintenance actions for the DB instances identified by these ARNs.

Type: Array of Filter objects

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

MaxRecords

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Type: Integer

Required: No

ResourceIdentifier

The ARN of a resource to return pending maintenance actions for.

Type: String

Required: No

Response Elements

The following elements are returned by the service.

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

PendingMaintenanceActions.ResourcePendingMaintenanceActions.N

The maintenance actions to be applied.

Type: Array of ResourcePendingMaintenanceActions objects

Errors

For information about the errors that are common to all actions, see Common Errors.

ResourceNotFoundFault

The specified resource ID was not found.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

FailoverDBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Forces a failover for a cluster.

A failover for a cluster promotes one of the Amazon DocumentDB replicas (read-only instances) in the cluster to be the primary instance (the cluster writer).

If the primary instance fails, Amazon DocumentDB automatically fails over to an Amazon DocumentDB replica, if one exists. You can force a failover when you want to simulate a failure of a primary instance for testing.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

A cluster identifier to force a failover for. This parameter is not case sensitive.

Constraints:

- Must match the identifier of an existing `DBCluster`.

Type: String

Required: No

TargetDBInstanceIdentifier

The name of the instance to promote to the primary instance.

You must specify the instance identifier for an Amazon DocumentDB replica in the cluster. For example, `mydbcluster-replica1`.

Type: String

Required: No

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

FailoverGlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Promotes the specified secondary DB cluster to be the primary DB cluster in the global cluster when failing over a global cluster occurs.

Use this operation to respond to an unplanned event, such as a regional disaster in the primary region. Failing over can result in a loss of write transaction data that wasn't replicated to the chosen secondary before the failover event occurred. However, the recovery process that promotes a DB instance on the chosen seconday DB cluster to be the primary writer DB instance guarantees that the data is in a transactionally consistent state.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

GlobalClusterIdentifier

The identifier of the Amazon DocumentDB global cluster to apply this operation. The identifier is the unique key assigned by the user when the cluster is created. In other words, it's the name of the global cluster.

Constraints:

- Must match the identifier of an existing global cluster.
- Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

TargetDbClusterIdentifier

The identifier of the secondary Amazon DocumentDB cluster that you want to promote to the primary for the global cluster. Use the Amazon Resource Name (ARN) for the identifier so that Amazon DocumentDB can locate the cluster in its AWS region.

Constraints:

- Must match the identifier of an existing secondary cluster.
- Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

AllowDataLoss

Specifies whether to allow data loss for this global cluster operation. Allowing data loss triggers a global failover operation.

If you don't specify `AllowDataLoss`, the global cluster operation defaults to a switchover.

Constraints:

- Can't be specified together with the `Switchover` parameter.

Type: Boolean

Required: No

Switchover

Specifies whether to switch over this global database cluster.

Constraints:

- Can't be specified together with the `AllowDataLoss` parameter.

Type: Boolean

Required: No

Response Elements

The following element is returned by the service.

GlobalCluster

A data type representing an Amazon DocumentDB global cluster.

Type: GlobalCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidGlobalClusterStateFault

The requested operation can't be performed while the cluster is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2

- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

Service: Amazon DocumentDB (with MongoDB compatibility)

Lists all tags on an Amazon DocumentDB resource.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

ResourceName

The Amazon DocumentDB resource with tags to be listed. This value is an Amazon Resource Name (ARN).

Type: String

Required: Yes

Filters.Filter.N

This parameter is not currently supported.

Type: Array of Filter objects

Required: No

Response Elements

The following element is returned by the service.

TagList.Tag.N

A list of one or more tags.

Type: Array of Tag objects

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

DBSnapshotNotFound

`DBSnapshotIdentifier` doesn't refer to an existing snapshot.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyDBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Modifies a setting for an Amazon DocumentDB cluster. You can change one or more database configuration parameters by specifying these parameters and the new values in the request.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The cluster identifier for the cluster that is being modified. This parameter is not case sensitive.

Constraints:

- Must match the identifier of an existing `DBCluster`.

Type: String

Required: Yes

AllowMajorVersionUpgrade

A value that indicates whether major version upgrades are allowed.

Constraints: You must allow major version upgrades when specifying a value for the `EngineVersion` parameter that is a different major version than the DB cluster's current version.

Type: Boolean

Required: No

ApplyImmediately

A value that specifies whether the changes in this request and any pending changes are asynchronously applied as soon as possible, regardless of the `PreferredMaintenanceWindow` setting for the cluster. If this parameter is set to `false`, changes to the cluster are applied during the next maintenance window.

The `ApplyImmediately` parameter affects only the `NewDBClusterIdentifier` and `MasterUserPassword` values. If you set this parameter value to `false`, the changes to the `NewDBClusterIdentifier` and `MasterUserPassword` values are applied during the next

maintenance window. All other changes are applied immediately, regardless of the value of the `ApplyImmediately` parameter.

Default: `false`

Type: Boolean

Required: No

BackupRetentionPeriod

The number of days for which automated backups are retained. You must specify a minimum value of 1.

Default: 1

Constraints:

- Must be a value from 1 to 35.

Type: Integer

Required: No

CloudwatchLogsExportConfiguration

The configuration setting for the log types to be enabled for export to Amazon CloudWatch Logs for a specific instance or cluster. The `EnableLogTypes` and `DisableLogTypes` arrays determine which logs are exported (or not exported) to CloudWatch Logs.

Type: CloudwatchLogsExportConfiguration object

Required: No

DBClusterParameterGroupName

The name of the cluster parameter group to use for the cluster.

Type: String

Required: No

DeletionProtection

Specifies whether this cluster can be deleted. If `DeletionProtection` is enabled, the cluster cannot be deleted unless it is modified and `DeletionProtection` is disabled. `DeletionProtection` protects clusters from being accidentally deleted.

Type: Boolean

Required: No

EngineVersion

The version number of the database engine to which you want to upgrade. Changing this parameter results in an outage. The change is applied during the next maintenance window unless `ApplyImmediately` is enabled.

To list all of the available engine versions for Amazon DocumentDB use the following command:

```
aws docdb describe-db-engine-versions --engine docdb --query
"DBEngineVersions[].EngineVersion"
```

Type: String

Required: No

ManageMasterUserPassword

Specifies whether to manage the master user password with Amazon Web Services Secrets Manager. If the cluster doesn't manage the master user password with Amazon Web Services Secrets Manager, you can turn on this management. In this case, you can't specify `MasterUserPassword`. If the cluster already manages the master user password with Amazon Web Services Secrets Manager, and you specify that the master user password is not managed with Amazon Web Services Secrets Manager, then you must specify `MasterUserPassword`. In this case, Amazon DocumentDB deletes the secret and uses the new password for the master user specified by `MasterUserPassword`.

Type: Boolean

Required: No

MasterUserPassword

The password for the master database user. This password can contain any printable ASCII character except forward slash (/), double quote ("), or the "at" symbol (@).

Constraints: Must contain from 8 to 100 characters.

Type: String

Required: No

MasterUserSecretKmsKeyId

The Amazon Web Services KMS key identifier to encrypt a secret that is automatically generated and managed in Amazon Web Services Secrets Manager.

This setting is valid only if both of the following conditions are met:

- The cluster doesn't manage the master user password in Amazon Web Services Secrets Manager. If the cluster already manages the master user password in Amazon Web Services Secrets Manager, you can't change the KMS key that is used to encrypt the secret.
- You are enabling `ManageMasterUserPassword` to manage the master user password in Amazon Web Services Secrets Manager. If you are turning on `ManageMasterUserPassword` and don't specify `MasterUserSecretKmsKeyId`, then the `aws/secretsmanager` KMS key is used to encrypt the secret. If the secret is in a different Amazon Web Services account, then you can't use the `aws/secretsmanager` KMS key to encrypt the secret, and you must use a customer managed KMS key.

The Amazon Web Services KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different Amazon Web Services account, specify the key ARN or alias ARN.

There is a default KMS key for your Amazon Web Services account. Your Amazon Web Services account has a different default KMS key for each Amazon Web Services Region.

Type: String

Required: No

NewDBClusterIdentifier

The new cluster identifier for the cluster when renaming a cluster. This value is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster2`

Type: String

Required: No

Port

The port number on which the cluster accepts connections.

Constraints: Must be a value from `1150` to `65535`.

Default: The same port as the original cluster.

Type: Integer

Required: No

PreferredBackupWindow

The daily time range during which automated backups are created if automated backups are enabled, using the `BackupRetentionPeriod` parameter.

The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region.

Constraints:

- Must be in the format `hh24:mi-hh24:mi`.
- Must be in Universal Coordinated Time (UTC).
- Must not conflict with the preferred maintenance window.
- Must be at least 30 minutes.

Type: String

Required: No

PreferredMaintenanceWindow

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region, occurring on a random day of the week.

Valid days: Mon, Tue, Wed, Thu, Fri, Sat, Sun

Constraints: Minimum 30-minute window.

Type: String

Required: No

RotateMasterUserPassword

Specifies whether to rotate the secret managed by Amazon Web Services Secrets Manager for the master user password.

This setting is valid only if the master user password is managed by Amazon DocumentDB in Amazon Web Services Secrets Manager for the cluster. The secret value contains the updated password.

Constraint: You must apply the change immediately when rotating the master user password.

Type: Boolean

Required: No

StorageType

The storage type to associate with the DB cluster.

For information on storage types for Amazon DocumentDB clusters, see Cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Valid values for storage type - `standard | iopt1`

Default value is `standard`

Type: String

Required: No

VpcSecurityGroupIds.VpcSecurityGroupId.N

A list of virtual private cloud (VPC) security groups that the cluster will belong to.

Type: Array of strings

Required: No

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterAlreadyExistsFault

You already have a cluster with the given identifier.

HTTP Status Code: 400

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBClusterParameterGroupNotFound

`DBClusterParameterGroupName` doesn't refer to an existing cluster parameter group.

HTTP Status Code: 404

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

InvalidDBSecurityGroupState

The state of the security group doesn't allow deletion.

HTTP Status Code: 400

InvalidDBSubnetGroupStateFault

The subnet group can't be deleted because it's in use.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

InvalidVPCNetworkStateFault

The subnet group doesn't cover all Availability Zones after it is created because of changes that were made.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2

- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyDBClusterParameterGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Modifies the parameters of a cluster parameter group. To modify more than one parameter, submit a list of the following: `ParameterName`, `ParameterValue`, and `ApplyMethod`. A maximum of 20 parameters can be modified in a single request.

> **Note**
>
> Changes to dynamic parameters are applied immediately. Changes to static parameters require a reboot or maintenance window before the change can take effect.

> **Important**
>
> After you create a cluster parameter group, you should wait at least 5 minutes before creating your first cluster that uses that cluster parameter group as the default parameter group. This allows Amazon DocumentDB to fully complete the create action before the parameter group is used as the default for a new cluster. This step is especially important for parameters that are critical when creating the default database for a cluster, such as the character set for the default database defined by the `character_set_database` parameter.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterParameterGroupName

The name of the cluster parameter group to modify.

Type: String

Required: Yes

Parameters.Parameter.N

A list of parameters in the cluster parameter group to modify.

Type: Array of Parameter objects

Required: Yes

Response Elements

The following element is returned by the service.

DBClusterParameterGroupName

The name of a cluster parameter group.

Constraints:

- Must be from 1 to 255 letters or numbers.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

> **Note**
> This value is stored as a lowercase string.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

InvalidDBParameterGroupState

The parameter group is in use, or it is in a state that is not valid. If you are trying to delete the parameter group, you can't delete it when the parameter group is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyDBClusterSnapshotAttribute

Service: Amazon DocumentDB (with MongoDB compatibility)

Adds an attribute and values to, or removes an attribute and values from, a manual cluster snapshot.

To share a manual cluster snapshot with other AWS accounts, specify `restore` as the `AttributeName`, and use the `ValuesToAdd` parameter to add a list of IDs of the AWS accounts that are authorized to restore the manual cluster snapshot. Use the value `all` to make the manual cluster snapshot public, which means that it can be copied or restored by all AWS accounts. Do not add the `all` value for any manual cluster snapshots that contain private information that you don't want available to all AWS accounts. If a manual cluster snapshot is encrypted, it can be shared, but only by specifying a list of authorized AWS account IDs for the `ValuesToAdd` parameter. You can't use `all` as a value for that parameter in this case.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

AttributeName

The name of the cluster snapshot attribute to modify.

To manage authorization for other AWS accounts to copy or restore a manual cluster snapshot, set this value to `restore`.

Type: String

Required: Yes

DBClusterSnapshotIdentifier

The identifier for the cluster snapshot to modify the attributes for.

Type: String

Required: Yes

ValuesToAdd.AttributeValue.N

A list of cluster snapshot attributes to add to the attribute specified by `AttributeName`.

To authorize other AWS accounts to copy or restore a manual cluster snapshot, set this list to include one or more AWS account IDs. To make the manual cluster snapshot restorable by any

AWS account, set it to `all`. Do not add the `all` value for any manual cluster snapshots that contain private information that you don't want to be available to all AWS accounts.

Type: Array of strings

Required: No

ValuesToRemove.AttributeValue.N

A list of cluster snapshot attributes to remove from the attribute specified by `AttributeName`.

To remove authorization for other AWS accounts to copy or restore a manual cluster snapshot, set this list to include one or more AWS account identifiers. To remove authorization for any AWS account to copy or restore the cluster snapshot, set it to `all` . If you specify `all`, an AWS account whose account ID is explicitly added to the `restore` attribute can still copy or restore a manual cluster snapshot.

Type: Array of strings

Required: No

Response Elements

The following element is returned by the service.

DBClusterSnapshotAttributesResult

Detailed information about the attributes that are associated with a cluster snapshot.

Type: DBClusterSnapshotAttributesResult object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

SharedSnapshotQuotaExceeded

You have exceeded the maximum number of accounts that you can share a manual DB snapshot with.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyDBInstance

Service: Amazon DocumentDB (with MongoDB compatibility)

Modifies settings for an instance. You can change one or more database configuration parameters by specifying these parameters and the new values in the request.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBInstanceIdentifier

The instance identifier. This value is stored as a lowercase string.

Constraints:

- Must match the identifier of an existing `DBInstance`.

Type: String

Required: Yes

ApplyImmediately

Specifies whether the modifications in this request and any pending modifications are asynchronously applied as soon as possible, regardless of the `PreferredMaintenanceWindow` setting for the instance.

If this parameter is set to `false`, changes to the instance are applied during the next maintenance window. Some parameter changes can cause an outage and are applied on the next reboot.

Default: `false`

Type: Boolean

Required: No

AutoMinorVersionUpgrade

This parameter does not apply to Amazon DocumentDB. Amazon DocumentDB does not perform minor version upgrades regardless of the value set.

Type: Boolean

Required: No

CACertificateIdentifier

Indicates the certificate that needs to be associated with the instance.

Type: String

Required: No

CertificateRotationRestart

Specifies whether the DB instance is restarted when you rotate your SSL/TLS certificate.

By default, the DB instance is restarted when you rotate your SSL/TLS certificate. The certificate is not updated until the DB instance is restarted.

> **Important**
>
> Set this parameter only if you are *not* using SSL/TLS to connect to the DB instance.

If you are using SSL/TLS to connect to the DB instance, see Updating Your Amazon DocumentDB TLS Certificates and Encrypting Data in Transit in the *Amazon DocumentDB Developer Guide*.

Type: Boolean

Required: No

CopyTagsToSnapshot

A value that indicates whether to copy all tags from the DB instance to snapshots of the DB instance. By default, tags are not copied.

Type: Boolean

Required: No

DBInstanceClass

The new compute and memory capacity of the instance; for example, `db.r5.large`. Not all instance classes are available in all AWS Regions.

If you modify the instance class, an outage occurs during the change. The change is applied during the next maintenance window, unless `ApplyImmediately` is specified as `true` for this request.

Default: Uses existing setting.

Type: String

Required: No

EnablePerformanceInsights

A value that indicates whether to enable Performance Insights for the DB Instance. For more information, see Using Amazon Performance Insights.

Type: Boolean

Required: No

NewDBInstanceIdentifier

The new instance identifier for the instance when renaming an instance. When you change the instance identifier, an instance reboot occurs immediately if you set `Apply Immediately` to `true`. It occurs during the next maintenance window if you set `Apply Immediately` to `false`. This value is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `mydbinstance`

Type: String

Required: No

PerformanceInsightsKMSKeyId

The AWS KMS key identifier for encryption of Performance Insights data.

The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key.

If you do not specify a value for PerformanceInsightsKMSKeyId, then Amazon DocumentDB uses your default KMS key. There is a default KMS key for your Amazon Web Services account. Your Amazon Web Services account has a different default KMS key for each Amazon Web Services region.

Type: String

Required: No

PreferredMaintenanceWindow

The weekly time range (in UTC) during which system maintenance can occur, which might result in an outage. Changing this parameter doesn't result in an outage except in the following situation, and the change is asynchronously applied as soon as possible. If there are pending actions that cause a reboot, and the maintenance window is changed to include the current time, changing this parameter causes a reboot of the instance. If you are moving this window to the current time, there must be at least 30 minutes between the current time and end of the window to ensure that pending changes are applied.

Default: Uses existing setting.

Format: `ddd:hh24:mi-ddd:hh24:mi`

Valid days: Mon, Tue, Wed, Thu, Fri, Sat, Sun

Constraints: Must be at least 30 minutes.

Type: String

Required: No

PromotionTier

A value that specifies the order in which an Amazon DocumentDB replica is promoted to the primary instance after a failure of the existing primary instance.

Default: 1

Valid values: 0-15

Type: Integer

Required: No

Response Elements

The following element is returned by the service.

DBInstance

Detailed information about an instance.

Type: DBInstance object

Errors

For information about the errors that are common to all actions, see Common Errors.

AuthorizationNotFound

The specified CIDR IP or Amazon EC2 security group isn't authorized for the specified security group.

Amazon DocumentDB also might not be authorized to perform necessary actions on your behalf using IAM.

HTTP Status Code: 404

CertificateNotFound

`CertificateIdentifier` doesn't refer to an existing certificate.

HTTP Status Code: 404

DBInstanceAlreadyExists

You already have a instance with the given identifier.

HTTP Status Code: 400

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

DBSecurityGroupNotFound

`DBSecurityGroupName` doesn't refer to an existing security group.

HTTP Status Code: 404

DBUpgradeDependencyFailure

The upgrade failed because a resource that the depends on can't be modified.

HTTP Status Code: 400

InsufficientDBInstanceCapacity

The specified instance class isn't available in the specified Availability Zone.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

InvalidDBSecurityGroupState

The state of the security group doesn't allow deletion.

HTTP Status Code: 400

InvalidVPCNetworkStateFault

The subnet group doesn't cover all Availability Zones after it is created because of changes that were made.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

StorageTypeNotSupported

Storage of the specified `StorageType` can't be associated with the DB instance.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyDBSubnetGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Modifies an existing subnet group. subnet groups must contain at least one subnet in at least two Availability Zones in the AWS Region.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBSubnetGroupName

The name for the subnet group. This value is stored as a lowercase string. You can't modify the default subnet group.

Constraints: Must match the name of an existing `DBSubnetGroup`. Must not be default.

Example: `mySubnetgroup`

Type: String

Required: Yes

SubnetIds.SubnetIdentifier.N

The Amazon EC2 subnet IDs for the subnet group.

Type: Array of strings

Required: Yes

DBSubnetGroupDescription

The description for the subnet group.

Type: String

Required: No

Response Elements

The following element is returned by the service.

DBSubnetGroup

Detailed information about a subnet group.

Type: DBSubnetGroup object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBSubnetGroupDoesNotCoverEnoughAZs

Subnets in the subnet group should cover at least two Availability Zones unless there is only one Availability Zone.

HTTP Status Code: 400

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

DBSubnetQuotaExceededFault

The request would cause you to exceed the allowed number of subnets in a subnet group.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

SubnetAlreadyInUse

The subnet is already in use in the Availability Zone.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyEventSubscription

Service: Amazon DocumentDB (with MongoDB compatibility)

Modifies an existing Amazon DocumentDB event notification subscription.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SubscriptionName

The name of the Amazon DocumentDB event notification subscription.

Type: String

Required: Yes

Enabled

A Boolean value; set to `true` to activate the subscription.

Type: Boolean

Required: No

EventCategories.EventCategory.N

A list of event categories for a `SourceType` that you want to subscribe to.

Type: Array of strings

Required: No

SnsTopicArn

The Amazon Resource Name (ARN) of the SNS topic created for event notification. The ARN is created by Amazon SNS when you create a topic and subscribe to it.

Type: String

Required: No

SourceType

The type of source that is generating the events. For example, if you want to be notified of events generated by an instance, set this parameter to `db-instance`. If this value is not specified, all events are returned.

Valid values: `db-instance`, `db-parameter-group`, `db-security-group`

Type: String

Required: No

Response Elements

The following element is returned by the service.

EventSubscription

Detailed information about an event to which you have subscribed.

Type: EventSubscription object

Errors

For information about the errors that are common to all actions, see Common Errors.

EventSubscriptionQuotaExceeded

You have reached the maximum number of event subscriptions.

HTTP Status Code: 400

SNSInvalidTopic

Amazon SNS has responded that there is a problem with the specified topic.

HTTP Status Code: 400

SNSNoAuthorization

You do not have permission to publish to the SNS topic Amazon Resource Name (ARN).

HTTP Status Code: 400

SNSTopicArnNotFound

The SNS topic Amazon Resource Name (ARN) does not exist.

HTTP Status Code: 404

SubscriptionCategoryNotFound

The provided category does not exist.

HTTP Status Code: 404

SubscriptionNotFound

The subscription name does not exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ModifyGlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Modify a setting for an Amazon DocumentDB global cluster. You can change one or more configuration parameters (for example: deletion protection), or the global cluster identifier by specifying these parameters and the new values in the request.

> **Note**
>
> This action only applies to Amazon DocumentDB clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

GlobalClusterIdentifier

The identifier for the global cluster being modified. This parameter isn't case-sensitive.

Constraints:

- Must match the identifier of an existing global cluster.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

DeletionProtection

Indicates if the global cluster has deletion protection enabled. The global cluster can't be deleted when deletion protection is enabled.

Type: Boolean

Required: No

NewGlobalClusterIdentifier

The new identifier for a global cluster when you modify a global cluster. This value is stored as a lowercase string.

- Must contain from 1 to 63 letters, numbers, or hyphens

  The first character must be a letter

  Can't end with a hyphen or contain two consecutive hyphens

Example: `my-cluster2`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: No

Response Elements

The following element is returned by the service.

GlobalCluster

A data type representing an Amazon DocumentDB global cluster.

Type: GlobalCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

InvalidGlobalClusterStateFault

The requested operation can't be performed while the cluster is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

RebootDBInstance

Service: Amazon DocumentDB (with MongoDB compatibility)

You might need to reboot your instance, usually for maintenance reasons. For example, if you make certain changes, or if you change the cluster parameter group that is associated with the instance, you must reboot the instance for the changes to take effect.

Rebooting an instance restarts the database engine service. Rebooting an instance results in a momentary outage, during which the instance status is set to *rebooting*.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBInstanceIdentifier

The instance identifier. This parameter is stored as a lowercase string.

Constraints:

- Must match the identifier of an existing `DBInstance`.

Type: String

Required: Yes

ForceFailover

When `true`, the reboot is conducted through a Multi-AZ failover.

Constraint: You can't specify `true` if the instance is not configured for Multi-AZ.

Type: Boolean

Required: No

Response Elements

The following element is returned by the service.

DBInstance

Detailed information about an instance.

Type: DBInstance object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

RemoveFromGlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Detaches an Amazon DocumentDB secondary cluster from a global cluster. The cluster becomes a standalone cluster with read-write capability instead of being read-only and receiving data from a primary in a different region.

> **Note**
>
> This action only applies to Amazon DocumentDB clusters.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DbClusterIdentifier

The Amazon Resource Name (ARN) identifying the cluster that was detached from the Amazon DocumentDB global cluster.

Type: String

Required: Yes

GlobalClusterIdentifier

The cluster identifier to detach from the Amazon DocumentDB global cluster.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

Response Elements

The following element is returned by the service.

GlobalCluster

A data type representing an Amazon DocumentDB global cluster.

Type: GlobalCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

InvalidGlobalClusterStateFault

The requested operation can't be performed while the cluster is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

RemoveSourceIdentifierFromSubscription

Service: Amazon DocumentDB (with MongoDB compatibility)

Removes a source identifier from an existing Amazon DocumentDB event notification subscription.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

SourceIdentifier

The source identifier to be removed from the subscription, such as the instance identifier for an instance, or the name of a security group.

Type: String

Required: Yes

SubscriptionName

The name of the Amazon DocumentDB event notification subscription that you want to remove a source identifier from.

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

EventSubscription

Detailed information about an event to which you have subscribed.

Type: EventSubscription object

Errors

For information about the errors that are common to all actions, see Common Errors.

SourceNotFound

The requested source could not be found.

HTTP Status Code: 404

SubscriptionNotFound

The subscription name does not exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

RemoveTagsFromResource

Service: Amazon DocumentDB (with MongoDB compatibility)

Removes metadata tags from an Amazon DocumentDB resource.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

ResourceName

The Amazon DocumentDB resource that the tags are removed from. This value is an Amazon Resource Name (ARN).

Type: String

Required: Yes

TagKeys.member.N

The tag key (name) of the tag to be removed.

Type: Array of strings

Required: Yes

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBInstanceNotFound

`DBInstanceIdentifier` doesn't refer to an existing instance.

HTTP Status Code: 404

DBSnapshotNotFound

`DBSnapshotIdentifier` doesn't refer to an existing snapshot.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ResetDBClusterParameterGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Modifies the parameters of a cluster parameter group to the default value. To reset specific parameters, submit a list of the following: `ParameterName` and `ApplyMethod`. To reset the entire cluster parameter group, specify the `DBClusterParameterGroupName` and `ResetAllParameters` parameters.

When you reset the entire group, dynamic parameters are updated immediately and static parameters are set to `pending-reboot` to take effect on the next DB instance reboot.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterParameterGroupName

The name of the cluster parameter group to reset.

Type: String

Required: Yes

Parameters.Parameter.N

A list of parameter names in the cluster parameter group to reset to the default values. You can't use this parameter if the `ResetAllParameters` parameter is set to `true`.

Type: Array of Parameter objects

Required: No

ResetAllParameters

A value that is set to `true` to reset all parameters in the cluster parameter group to their default values, and `false` otherwise. You can't use this parameter if there is a list of parameter names specified for the `Parameters` parameter.

Type: Boolean

Required: No

Response Elements

The following element is returned by the service.

DBClusterParameterGroupName

The name of a cluster parameter group.

Constraints:

- Must be from 1 to 255 letters or numbers.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

> **Note**
>
> This value is stored as a lowercase string.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

DBParameterGroupNotFound

`DBParameterGroupName` doesn't refer to an existing parameter group.

HTTP Status Code: 404

InvalidDBParameterGroupState

The parameter group is in use, or it is in a state that is not valid. If you are trying to delete the parameter group, you can't delete it when the parameter group is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

RestoreDBClusterFromSnapshot

Service: Amazon DocumentDB (with MongoDB compatibility)

Creates a new cluster from a snapshot or cluster snapshot.

If a snapshot is specified, the target cluster is created from the source DB snapshot with a default configuration and default security group.

If a cluster snapshot is specified, the target cluster is created from the source cluster restore point with the same configuration as the original source DB cluster, except that the new cluster is created with the default security group.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The name of the cluster to create from the snapshot or cluster snapshot. This parameter isn't case sensitive.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-snapshot-id`

Type: String

Required: Yes

Engine

The database engine to use for the new cluster.

Default: The same as source.

Constraint: Must be compatible with the engine of the source.

Type: String

Required: Yes

SnapshotIdentifier

The identifier for the snapshot or cluster snapshot to restore from.

You can use either the name or the Amazon Resource Name (ARN) to specify a cluster snapshot. However, you can use only the ARN to specify a snapshot.

Constraints:

- Must match the identifier of an existing snapshot.

Type: String

Required: Yes

AvailabilityZones.AvailabilityZone.N

Provides the list of Amazon EC2 Availability Zones that instances in the restored DB cluster can be created in.

Type: Array of strings

Required: No

DBClusterParameterGroupName

The name of the DB cluster parameter group to associate with this DB cluster.

Type: String. *Required:* No.

If this argument is omitted, the default DB cluster parameter group is used. If supplied, must match the name of an existing default DB cluster parameter group. The string must consist of from 1 to 255 letters, numbers or hyphens. Its first character must be a letter, and it cannot end with a hyphen or contain two consecutive hyphens.

Type: String

Required: No

DBSubnetGroupName

The name of the subnet group to use for the new cluster.

Constraints: If provided, must match the name of an existing `DBSubnetGroup`.

Example: `mySubnetgroup`

Type: String

Required: No

DeletionProtection

Specifies whether this cluster can be deleted. If `DeletionProtection` is enabled, the cluster cannot be deleted unless it is modified and `DeletionProtection` is disabled. `DeletionProtection` protects clusters from being accidentally deleted.

Type: Boolean

Required: No

EnableCloudwatchLogsExports.member.N

A list of log types that must be enabled for exporting to Amazon CloudWatch Logs.

Type: Array of strings

Required: No

EngineVersion

The version of the database engine to use for the new cluster.

Type: String

Required: No

KmsKeyId

The AWS KMS key identifier to use when restoring an encrypted cluster from a DB snapshot or cluster snapshot.

The AWS KMS key identifier is the Amazon Resource Name (ARN) for the AWS KMS encryption key. If you are restoring a cluster with the same AWS account that owns the AWS KMS encryption key used to encrypt the new cluster, then you can use the AWS KMS key alias instead of the ARN for the AWS KMS encryption key.

If you do not specify a value for the `KmsKeyId` parameter, then the following occurs:

- If the snapshot or cluster snapshot in `SnapshotIdentifier` is encrypted, then the restored cluster is encrypted using the AWS KMS key that was used to encrypt the snapshot or the cluster snapshot.

- If the snapshot or the cluster snapshot in `SnapshotIdentifier` is not encrypted, then the restored DB cluster is not encrypted.

Type: String

Required: No

Port

The port number on which the new cluster accepts connections.

Constraints: Must be a value from `1150` to `65535`.

Default: The same port as the original cluster.

Type: Integer

Required: No

StorageType

The storage type to associate with the DB cluster.

For information on storage types for Amazon DocumentDB clusters, see Cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Valid values for storage type - `standard | iopt1`

Default value is `standard`

Type: String

Required: No

Tags.Tag.N

The tags to be assigned to the restored cluster.

Type: Array of Tag objects

Required: No

VpcSecurityGroupIds.VpcSecurityGroupId.N

A list of virtual private cloud (VPC) security groups that the new cluster will belong to.

Type: Array of strings

Required: No

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterAlreadyExistsFault

You already have a cluster with the given identifier.

HTTP Status Code: 400

DBClusterQuotaExceededFault

The cluster can't be created because you have reached the maximum allowed quota of clusters.

HTTP Status Code: 403

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

DBSnapshotNotFound

`DBSnapshotIdentifier` doesn't refer to an existing snapshot.

HTTP Status Code: 404

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

InsufficientDBClusterCapacityFault

The cluster doesn't have enough capacity for the current operation.

HTTP Status Code: 403

InsufficientStorageClusterCapacity

There is not enough storage available for the current action. You might be able to resolve this error by updating your subnet group to use different Availability Zones that have more storage available.

HTTP Status Code: 400

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

InvalidDBSnapshotState

The state of the snapshot doesn't allow deletion.

HTTP Status Code: 400

InvalidRestoreFault

You cannot restore from a virtual private cloud (VPC) backup to a non-VPC DB instance.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

InvalidVPCNetworkStateFault

The subnet group doesn't cover all Availability Zones after it is created because of changes that were made.

HTTP Status Code: 400

KMSKeyNotAccessibleFault

An error occurred when accessing an AWS KMS key.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V3

RestoreDBClusterToPointInTime

Service: Amazon DocumentDB (with MongoDB compatibility)

Restores a cluster to an arbitrary point in time. Users can restore to any point in time before `LatestRestorableTime` for up to `BackupRetentionPeriod` days. The target cluster is created from the source cluster with the same configuration as the original cluster, except that the new cluster is created with the default security group.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The name of the new cluster to be created.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Type: String

Required: Yes

SourceDBClusterIdentifier

The identifier of the source cluster from which to restore.

Constraints:

- Must match the identifier of an existing `DBCluster`.

Type: String

Required: Yes

DBSubnetGroupName

The subnet group name to use for the new cluster.

Constraints: If provided, must match the name of an existing `DBSubnetGroup`.

Example: `mySubnetgroup`

Type: String

Required: No

DeletionProtection

Specifies whether this cluster can be deleted. If `DeletionProtection` is enabled, the cluster cannot be deleted unless it is modified and `DeletionProtection` is disabled. `DeletionProtection` protects clusters from being accidentally deleted.

Type: Boolean

Required: No

EnableCloudwatchLogsExports.member.N

A list of log types that must be enabled for exporting to Amazon CloudWatch Logs.

Type: Array of strings

Required: No

KmsKeyId

The AWS KMS key identifier to use when restoring an encrypted cluster from an encrypted cluster.

The AWS KMS key identifier is the Amazon Resource Name (ARN) for the AWS KMS encryption key. If you are restoring a cluster with the same AWS account that owns the AWS KMS encryption key used to encrypt the new cluster, then you can use the AWS KMS key alias instead of the ARN for the AWS KMS encryption key.

You can restore to a new cluster and encrypt the new cluster with an AWS KMS key that is different from the AWS KMS key used to encrypt the source cluster. The new DB cluster is encrypted with the AWS KMS key identified by the `KmsKeyId` parameter.

If you do not specify a value for the `KmsKeyId` parameter, then the following occurs:

- If the cluster is encrypted, then the restored cluster is encrypted using the AWS KMS key that was used to encrypt the source cluster.
- If the cluster is not encrypted, then the restored cluster is not encrypted.

If `DBClusterIdentifier` refers to a cluster that is not encrypted, then the restore request is rejected.

Type: String

Required: No

Port

The port number on which the new cluster accepts connections.

Constraints: Must be a value from `1150` to `65535`.

Default: The default port for the engine.

Type: Integer

Required: No

RestoreToTime

The date and time to restore the cluster to.

Valid values: A time in Universal Coordinated Time (UTC) format.

Constraints:

- Must be before the latest restorable time for the instance.
- Must be specified if the `UseLatestRestorableTime` parameter is not provided.
- Cannot be specified if the `UseLatestRestorableTime` parameter is `true`.
- Cannot be specified if the `RestoreType` parameter is `copy-on-write`.

Example: `2015-03-07T23:45:00Z`

Type: Timestamp

Required: No

RestoreType

The type of restore to be performed. You can specify one of the following values:

- `full-copy` - The new DB cluster is restored as a full copy of the source DB cluster.
- `copy-on-write` - The new DB cluster is restored as a clone of the source DB cluster.

Constraints: You can't specify `copy-on-write` if the engine version of the source DB cluster is earlier than 1.11.

If you don't specify a `RestoreType` value, then the new DB cluster is restored as a full copy of the source DB cluster.

Type: String

Required: No

StorageType

The storage type to associate with the DB cluster.

For information on storage types for Amazon DocumentDB clusters, see Cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Valid values for storage type - `standard | iopt1`

Default value is `standard`

Type: String

Required: No

Tags.Tag.N

The tags to be assigned to the restored cluster.

Type: Array of Tag objects

Required: No

UseLatestRestorableTime

A value that is set to `true` to restore the cluster to the latest restorable backup time, and `false` otherwise.

Default: `false`

Constraints: Cannot be specified if the `RestoreToTime` parameter is provided.

Type: Boolean

Required: No

VpcSecurityGroupIds.VpcSecurityGroupId.N

A list of VPC security groups that the new cluster belongs to.

Type: Array of strings

Required: No

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterAlreadyExistsFault

You already have a cluster with the given identifier.

HTTP Status Code: 400

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

DBClusterQuotaExceededFault

The cluster can't be created because you have reached the maximum allowed quota of clusters.

HTTP Status Code: 403

DBClusterSnapshotNotFoundFault

`DBClusterSnapshotIdentifier` doesn't refer to an existing cluster snapshot.

HTTP Status Code: 404

DBSubnetGroupNotFoundFault

`DBSubnetGroupName` doesn't refer to an existing subnet group.

HTTP Status Code: 404

InsufficientDBClusterCapacityFault

The cluster doesn't have enough capacity for the current operation.

HTTP Status Code: 403

InsufficientStorageClusterCapacity

There is not enough storage available for the current action. You might be able to resolve this error by updating your subnet group to use different Availability Zones that have more storage available.

HTTP Status Code: 400

InvalidDBClusterSnapshotStateFault

The provided value isn't a valid cluster snapshot state.

HTTP Status Code: 400

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBSnapshotState

The state of the snapshot doesn't allow deletion.

HTTP Status Code: 400

InvalidRestoreFault

You cannot restore from a virtual private cloud (VPC) backup to a non-VPC DB instance.

HTTP Status Code: 400

InvalidSubnet

The requested subnet is not valid, or multiple subnets were requested that are not all in a common virtual private cloud (VPC).

HTTP Status Code: 400

InvalidVPCNetworkStateFault

The subnet group doesn't cover all Availability Zones after it is created because of changes that were made.

HTTP Status Code: 400

KMSKeyNotAccessibleFault

An error occurred when accessing an AWS KMS key.

HTTP Status Code: 400

StorageQuotaExceeded

The request would cause you to exceed the allowed amount of storage available across all instances.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartDBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Restarts the stopped cluster that is specified by `DBClusterIdentifier`. For more information, see Stopping and Starting an Amazon DocumentDB Cluster.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The identifier of the cluster to restart. Example: `docdb-2019-05-28-15-24-52`

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StopDBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Stops the running cluster that is specified by `DBClusterIdentifier`. The cluster must be in the *available* state. For more information, see Stopping and Starting an Amazon DocumentDB Cluster.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

DBClusterIdentifier

The identifier of the cluster to stop. Example: `docdb-2019-05-28-15-24-52`

Type: String

Required: Yes

Response Elements

The following element is returned by the service.

DBCluster

Detailed information about a cluster.

Type: DBCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidDBInstanceState

The specified instance isn't in the *available* state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

SwitchoverGlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Switches over the specified secondary Amazon DocumentDB cluster to be the new primary Amazon DocumentDB cluster in the global database cluster.

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters.

GlobalClusterIdentifier

The identifier of the Amazon DocumentDB global database cluster to switch over. The identifier is the unique key assigned by the user when the cluster is created. In other words, it's the name of the global cluster. This parameter isn't case-sensitive.

Constraints:

- Must match the identifier of an existing global cluster (Amazon DocumentDB global database).
- Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

TargetDbClusterIdentifier

The identifier of the secondary Amazon DocumentDB cluster to promote to the new primary for the global database cluster. Use the Amazon Resource Name (ARN) for the identifier so that Amazon DocumentDB can locate the cluster in its AWS region.

Constraints:

- Must match the identifier of an existing secondary cluster.
- Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: Yes

Response Elements

The following element is returned by the service.

GlobalCluster

A data type representing an Amazon DocumentDB global cluster.

Type: GlobalCluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

DBClusterNotFoundFault

`DBClusterIdentifier` doesn't refer to an existing cluster.

HTTP Status Code: 404

GlobalClusterNotFoundFault

The `GlobalClusterIdentifier` doesn't refer to an existing global cluster.

HTTP Status Code: 404

InvalidDBClusterStateFault

The cluster isn't in a valid state.

HTTP Status Code: 400

InvalidGlobalClusterStateFault

The requested operation can't be performed while the cluster is in this state.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Amazon DocumentDB Elastic Clusters

The following actions are supported by Amazon DocumentDB Elastic Clusters:

- ApplyPendingMaintenanceAction
- CopyClusterSnapshot
- CreateCluster
- CreateClusterSnapshot
- DeleteCluster
- DeleteClusterSnapshot
- GetCluster
- GetClusterSnapshot
- GetPendingMaintenanceAction
- ListClusters
- ListClusterSnapshots
- ListPendingMaintenanceActions
- ListTagsForResource

- RestoreClusterFromSnapshot
- StartCluster
- StopCluster
- TagResource
- UntagResource
- UpdateCluster

ApplyPendingMaintenanceAction

Service: Amazon DocumentDB Elastic Clusters

The type of pending maintenance action to be applied to the resource.

Request Syntax

```
POST /pending-action HTTP/1.1
Content-type: application/json

{
   "applyAction": "string",
   "applyOn": "string",
   "optInType": "string",
   "resourceArn": "string"
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

applyAction

The pending maintenance action to apply to the resource.

Valid actions are:

- ENGINE_UPDATE
- ENGINE_UPGRADE
- SECURITY_UPDATE
- OS_UPDATE
- MASTER_USER_PASSWORD_UPDATE

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: Yes

optInType

A value that specifies the type of opt-in request, or undoes an opt-in request. An opt-in request of type `IMMEDIATE` can't be undone.

Type: String

Valid Values: `IMMEDIATE | NEXT_MAINTENANCE | APPLY_ON | UNDO_OPT_IN`

Required: Yes

resourceArn

The Amazon DocumentDB Amazon Resource Name (ARN) of the resource to which the pending maintenance action applies.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: Yes

applyOn

A specific date to apply the pending maintenance action. Required if opt-in-type is `APPLY_ON`. Format: `yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "resourcePendingMaintenanceAction": {
      "pendingMaintenanceActionDetails": [
         {
            "action": "string",
```

```
            "autoAppliedAfterDate": "string",
            "currentApplyDate": "string",
            "description": "string",
            "forcedApplyDate": "string",
            "optInStatus": "string"
        }
    ],
    "resourceArn": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

resourcePendingMaintenanceAction

The output of the pending maintenance action being applied.

Type: ResourcePendingMaintenanceAction object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CopyClusterSnapshot

Service: Amazon DocumentDB Elastic Clusters

Copies a snapshot of an elastic cluster.

Request Syntax

```
POST /cluster-snapshot/snapshotArn/copy HTTP/1.1
Content-type: application/json

{
   "copyTags": boolean,
   "kmsKeyId": "string",
   "tags": {
      "string" : "string"
   },
   "targetSnapshotName": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

snapshotArn

The Amazon Resource Name (ARN) identifier of the elastic cluster snapshot.

Required: Yes

Request Body

The request accepts the following data in JSON format.

targetSnapshotName

The identifier of the new elastic cluster snapshot to create from the source cluster snapshot. This parameter is not case sensitive.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.

- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `elastic-cluster-snapshot-5`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Required: Yes

copyTags

Set to `true` to copy all tags from the source cluster snapshot to the target elastic cluster snapshot. The default is `false`.

Type: Boolean

Required: No

kmsKeyId

The AWS KMS key ID for an encrypted elastic cluster snapshot. The AWS KMS key ID is the Amazon Resource Name (ARN), AWS KMS key identifier, or the AWS KMS key alias for the AWS KMS encryption key.

If you copy an encrypted elastic cluster snapshot from your AWS account, you can specify a value for `KmsKeyId` to encrypt the copy with a new AWSS KMS encryption key. If you don't specify a value for `KmsKeyId`, then the copy of the elastic cluster snapshot is encrypted with the same `AWS` KMS key as the source elastic cluster snapshot.

If you copy an unencrypted elastic cluster snapshot and specify a value for the `KmsKeyId` parameter, an error is returned.

Type: String

Required: No

tags

The tags to be assigned to the elastic cluster snapshot.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Key Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "snapshot": {
      "adminUserName": "string",
      "clusterArn": "string",
      "clusterCreationTime": "string",
      "kmsKeyId": "string",
      "snapshotArn": "string",
      "snapshotCreationTime": "string",
      "snapshotName": "string",
      "snapshotType": "string",
      "status": "string",
      "subnetIds": [ "string" ],
      "vpcSecurityGroupIds": [ "string" ]
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

snapshot

Returns information about a specific elastic cluster snapshot.

Type: ClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ServiceQuotaExceededException

The service quota for the action was exceeded.

HTTP Status Code: 402

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateCluster

Service: Amazon DocumentDB Elastic Clusters

Creates a new Amazon DocumentDB elastic cluster and returns its cluster structure.

Request Syntax

```
POST /cluster HTTP/1.1
Content-type: application/json

{
   "adminUserName": "string",
   "adminUserPassword": "string",
   "authType": "string",
   "backupRetentionPeriod": number,
   "clientToken": "string",
   "clusterName": "string",
   "kmsKeyId": "string",
   "preferredBackupWindow": "string",
   "preferredMaintenanceWindow": "string",
   "shardCapacity": number,
   "shardCount": number,
   "shardInstanceCount": number,
   "subnetIds": [ "string" ],
   "tags": {
      "string" : "string"
   },
   "vpcSecurityGroupIds": [ "string" ]
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

adminUserName

The name of the Amazon DocumentDB elastic clusters administrator.

Constraints:

- Must be from 1 to 63 letters or numbers.
- The first character must be a letter.
- Cannot be a reserved word.

Type: String

Required: Yes

adminUserPassword

The password for the Amazon DocumentDB elastic clusters administrator. The password can contain any printable ASCII characters.

Constraints:

- Must contain from 8 to 100 characters.
- Cannot contain a forward slash (/), double quote ("), or the "at" symbol (@).

Type: String

Required: Yes

authType

The authentication type used to determine where to fetch the password used for accessing the elastic cluster. Valid types are `PLAIN_TEXT` or `SECRET_ARN`.

Type: String

Valid Values: `PLAIN_TEXT | SECRET_ARN`

Required: Yes

clusterName

The name of the new elastic cluster. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster`

Type: String

Required: Yes

shardCapacity

The number of vCPUs assigned to each elastic cluster shard. Maximum is 64. Allowed values are 2, 4, 8, 16, 32, 64.

Type: Integer

Required: Yes

shardCount

The number of shards assigned to the elastic cluster. Maximum is 32.

Type: Integer

Required: Yes

backupRetentionPeriod

The number of days for which automatic snapshots are retained.

Type: Integer

Required: No

clientToken

The client token for the elastic cluster.

Type: String

Required: No

kmsKeyId

The KMS key identifier to use to encrypt the new elastic cluster.

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are creating a cluster using the same Amazon account that owns this KMS encryption key, you can use the KMS key alias instead of the ARN as the KMS encryption key.

If an encryption key is not specified, Amazon DocumentDB uses the default encryption key that KMS creates for your account. Your account has a different default encryption key for each Amazon Region.

Type: String

Required: No

preferredBackupWindow

The daily time range during which automated backups are created if automated backups are enabled, as determined by the `backupRetentionPeriod`.

Type: String

Required: No

preferredMaintenanceWindow

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

Default: a 30-minute window selected at random from an 8-hour block of time for each AWS Region, occurring on a random day of the week.

Valid days: Mon, Tue, Wed, Thu, Fri, Sat, Sun

Constraints: Minimum 30-minute window.

Type: String

Required: No

shardInstanceCount

The number of replica instances applying to all shards in the elastic cluster. A `shardInstanceCount` value of 1 means there is one writer instance, and any additional instances are replicas that can be used for reads and to improve availability.

Type: Integer

Required: No

subnetIds

The Amazon EC2 subnet IDs for the new elastic cluster.

Type: Array of strings

Required: No

tags

The tags to be assigned to the new elastic cluster.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Key Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

vpcSecurityGroupIds

A list of EC2 VPC security groups to associate with the new elastic cluster.

Type: Array of strings

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
```

```
        "kmsKeyId": "string",
        "preferredBackupWindow": "string",
        "preferredMaintenanceWindow": "string",
        "shardCapacity": number,
        "shardCount": number,
        "shardInstanceCount": number,
        "shards": [
           {
              "createTime": "string",
              "shardId": "string",
              "status": "string"
           }
        ],
        "status": "string",
        "subnetIds": [ "string" ],
        "vpcSecurityGroupIds": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

The new elastic cluster that has been created.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ServiceQuotaExceededException

The service quota for the action was exceeded.

HTTP Status Code: 402

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateClusterSnapshot

Service: Amazon DocumentDB Elastic Clusters

Creates a snapshot of an elastic cluster.

Request Syntax

```
POST /cluster-snapshot HTTP/1.1
Content-type: application/json

{
   "clusterArn": "string",
   "snapshotName": "string",
   "tags": {
      "string" : "string"
   }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

clusterArn

The ARN identifier of the elastic cluster of which you want to create a snapshot.

Type: String

Required: Yes

snapshotName

The name of the new elastic cluster snapshot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Required: Yes

tags

The tags to be assigned to the new elastic cluster snapshot.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Key Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "snapshot": {
      "adminUserName": "string",
      "clusterArn": "string",
      "clusterCreationTime": "string",
      "kmsKeyId": "string",
      "snapshotArn": "string",
      "snapshotCreationTime": "string",
      "snapshotName": "string",
      "snapshotType": "string",
      "status": "string",
      "subnetIds": [ "string" ],
      "vpcSecurityGroupIds": [ "string" ]
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

snapshot

Returns information about the new elastic cluster snapshot.

Type: ClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ServiceQuotaExceededException

The service quota for the action was exceeded.

HTTP Status Code: 402

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteCluster

Service: Amazon DocumentDB Elastic Clusters

Delete an elastic cluster.

Request Syntax

```
DELETE /cluster/clusterArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

clusterArn

The ARN identifier of the elastic cluster that is to be deleted.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
      "kmsKeyId": "string",
      "preferredBackupWindow": "string",
      "preferredMaintenanceWindow": "string",
      "shardCapacity": number,
      "shardCount": number,
```

```
        "shardInstanceCount": number,
        "shards": [
           {
              "createTime": "string",
              "shardId": "string",
              "status": "string"
           }
        ],
        "status": "string",
        "subnetIds": [ "string" ],
        "vpcSecurityGroupIds": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

Returns information about the newly deleted elastic cluster.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteClusterSnapshot

Service: Amazon DocumentDB Elastic Clusters

Delete an elastic cluster snapshot.

Request Syntax

```
DELETE /cluster-snapshot/snapshotArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

snapshotArn

The ARN identifier of the elastic cluster snapshot that is to be deleted.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "snapshot": {
      "adminUserName": "string",
      "clusterArn": "string",
      "clusterCreationTime": "string",
      "kmsKeyId": "string",
      "snapshotArn": "string",
      "snapshotCreationTime": "string",
      "snapshotName": "string",
      "snapshotType": "string",
      "status": "string",
      "subnetIds": [ "string" ],
      "vpcSecurityGroupIds": [ "string" ]
   }
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

snapshot

Returns information about the newly deleted elastic cluster snapshot.

Type: ClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetCluster

Service: Amazon DocumentDB Elastic Clusters

Returns information about a specific elastic cluster.

Request Syntax

```
GET /cluster/clusterArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

clusterArn

The ARN identifier of the elastic cluster.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
      "kmsKeyId": "string",
      "preferredBackupWindow": "string",
      "preferredMaintenanceWindow": "string",
      "shardCapacity": number,
      "shardCount": number,
```

```
        "shardInstanceCount": number,
        "shards": [
           {
              "createTime": "string",
              "shardId": "string",
              "status": "string"
           }
        ],
        "status": "string",
        "subnetIds": [ "string" ],
        "vpcSecurityGroupIds": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

Returns information about a specific elastic cluster.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetClusterSnapshot

Service: Amazon DocumentDB Elastic Clusters

Returns information about a specific elastic cluster snapshot

Request Syntax

```
GET /cluster-snapshot/snapshotArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

snapshotArn

The ARN identifier of the elastic cluster snapshot.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "snapshot": {
      "adminUserName": "string",
      "clusterArn": "string",
      "clusterCreationTime": "string",
      "kmsKeyId": "string",
      "snapshotArn": "string",
      "snapshotCreationTime": "string",
      "snapshotName": "string",
      "snapshotType": "string",
      "status": "string",
      "subnetIds": [ "string" ],
      "vpcSecurityGroupIds": [ "string" ]
   }
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

snapshot

Returns information about a specific elastic cluster snapshot.

Type: ClusterSnapshot object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetPendingMaintenanceAction

Service: Amazon DocumentDB Elastic Clusters

Retrieves all maintenance actions that are pending.

Request Syntax

```
GET /pending-action/resourceArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

Retrieves pending maintenance actions for a specific Amazon Resource Name (ARN).

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "resourcePendingMaintenanceAction": {
      "pendingMaintenanceActionDetails": [
         {
            "action": "string",
            "autoAppliedAfterDate": "string",
            "currentApplyDate": "string",
            "description": "string",
            "forcedApplyDate": "string",
            "optInStatus": "string"
         }
```

```
        ],
        "resourceArn": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

resourcePendingMaintenanceAction

Provides information about a pending maintenance action for a resource.

Type: ResourcePendingMaintenanceAction object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListClusters

Service: Amazon DocumentDB Elastic Clusters

Returns information about provisioned Amazon DocumentDB elastic clusters.

Request Syntax

```
GET /clusters?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

maxResults

The maximum number of elastic cluster snapshot results to receive in the response.

Valid Range: Minimum value of 1. Maximum value of 100.

nextToken

A pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond this token, up to the value specified by `max-results`.

If there is no more data in the responce, the `nextToken` will not be returned.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "clusters": [
      {
         "clusterArn": "string",
         "clusterName": "string",
         "status": "string"
      }
```

```
    ],
    "nextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

clusters

A list of Amazon DocumentDB elastic clusters.

Type: Array of ClusterInList objects

nextToken

A pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond this token, up to the value specified by `max-results`.

If there is no more data in the responce, the `nextToken` will not be returned.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListClusterSnapshots

Service: Amazon DocumentDB Elastic Clusters

Returns information about snapshots for a specified elastic cluster.

Request Syntax

```
GET /cluster-snapshots?
clusterArn=clusterArn&maxResults=maxResults&nextToken=nextToken&snapshotType=snapshotType
 HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

clusterArn

The ARN identifier of the elastic cluster.

maxResults

The maximum number of elastic cluster snapshot results to receive in the response.

Valid Range: Minimum value of 20. Maximum value of 100.

nextToken

A pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond this token, up to the value specified by `max-results`.

If there is no more data in the responce, the `nextToken` will not be returned.

snapshotType

The type of cluster snapshots to be returned. You can specify one of the following values:

- `automated` - Return all cluster snapshots that Amazon DocumentDB has automatically created for your AWS account.
- `manual` - Return all cluster snapshots that you have manually created for your AWS account.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "snapshots": [
      {
         "clusterArn": "string",
         "snapshotArn": "string",
         "snapshotCreationTime": "string",
         "snapshotName": "string",
         "status": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

nextToken

A pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond this token, up to the value specified by `max-results`.

If there is no more data in the responce, the `nextToken` will not be returned.

Type: String

snapshots

A list of snapshots for a specified elastic cluster.

Type: Array of ClusterSnapshotInList objects

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListPendingMaintenanceActions

Service: Amazon DocumentDB Elastic Clusters

Retrieves a list of all maintenance actions that are pending.

Request Syntax

```
GET /pending-actions?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

maxResults

The maximum number of results to include in the response. If more records exist than the specified `maxResults` value, a pagination token (marker) is included in the response so that the remaining results can be retrieved.

Valid Range: Minimum value of 1. Maximum value of 100.

nextToken

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `maxResults`.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "resourcePendingMaintenanceActions": [
      {
         "pendingMaintenanceActionDetails": [
            {
               "action": "string",
               "autoAppliedAfterDate": "string",
```

```
                "currentApplyDate": "string",
                "description": "string",
                "forcedApplyDate": "string",
                "optInStatus": "string"
            }
        ],
        "resourceArn": "string"
    }
  ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

resourcePendingMaintenanceActions

Provides information about a pending maintenance action for a resource.

Type: Array of ResourcePendingMaintenanceAction objects

nextToken

An optional pagination token provided by a previous request. If this parameter is displayed, the responses will include only records beyond the marker, up to the value specified by `maxResults`.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

Service: Amazon DocumentDB Elastic Clusters

Lists all tags on a elastic cluster resource

Request Syntax

```
GET /tags/resourceArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

The ARN identifier of the elastic cluster resource.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "tags": {
      "string" : "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

tags

The list of tags for the specified elastic cluster resource.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Key Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Errors

For information about the errors that are common to all actions, see Common Errors.

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

RestoreClusterFromSnapshot

Service: Amazon DocumentDB Elastic Clusters

Restores an elastic cluster from a snapshot.

Request Syntax

```
POST /cluster-snapshot/snapshotArn/restore HTTP/1.1
Content-type: application/json

{
   "clusterName": "string",
   "kmsKeyId": "string",
   "shardCapacity": number,
   "shardInstanceCount": number,
   "subnetIds": [ "string" ],
   "tags": {
      "string" : "string"
   },
   "vpcSecurityGroupIds": [ "string" ]
}
```

URI Request Parameters

The request uses the following URI parameters.

snapshotArn

The ARN identifier of the elastic cluster snapshot.

Required: Yes

Request Body

The request accepts the following data in JSON format.

clusterName

The name of the elastic cluster.

Type: String

Required: Yes

kmsKeyId

The KMS key identifier to use to encrypt the new Amazon DocumentDB elastic clusters cluster.

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are creating a cluster using the same Amazon account that owns this KMS encryption key, you can use the KMS key alias instead of the ARN as the KMS encryption key.

If an encryption key is not specified here, Amazon DocumentDB uses the default encryption key that KMS creates for your account. Your account has a different default encryption key for each Amazon Region.

Type: String

Required: No

shardCapacity

The capacity of each shard in the new restored elastic cluster.

Type: Integer

Required: No

shardInstanceCount

The number of replica instances applying to all shards in the elastic cluster. A `shardInstanceCount` value of 1 means there is one writer instance, and any additional instances are replicas that can be used for reads and to improve availability.

Type: Integer

Required: No

subnetIds

The Amazon EC2 subnet IDs for the elastic cluster.

Type: Array of strings

Required: No

tags

A list of the tag names to be assigned to the restored elastic cluster, in the form of an array of key-value pairs in which the key is the tag name and the value is the key value.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Key Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

vpcSecurityGroupIds

A list of EC2 VPC security groups to associate with the elastic cluster.

Type: Array of strings

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
      "kmsKeyId": "string",
      "preferredBackupWindow": "string",
      "preferredMaintenanceWindow": "string",
      "shardCapacity": number,
      "shardCount": number,
      "shardInstanceCount": number,
      "shards": [
         {
            "createTime": "string",
            "shardId": "string",
            "status": "string"
         }
```

```
        ],
        "status": "string",
        "subnetIds": [ "string" ],
        "vpcSecurityGroupIds": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

Returns information about a the restored elastic cluster.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ServiceQuotaExceededException

The service quota for the action was exceeded.

HTTP Status Code: 402

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartCluster

Service: Amazon DocumentDB Elastic Clusters

Restarts the stopped elastic cluster that is specified by `clusterARN`.

Request Syntax

```
POST /cluster/clusterArn/start HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

clusterArn

The ARN identifier of the elastic cluster.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
      "kmsKeyId": "string",
      "preferredBackupWindow": "string",
      "preferredMaintenanceWindow": "string",
      "shardCapacity": number,
      "shardCount": number,
```

```
        "shardInstanceCount": number,
        "shards": [
           {
              "createTime": "string",
              "shardId": "string",
              "status": "string"
           }
        ],
        "status": "string",
        "subnetIds": [ "string" ],
        "vpcSecurityGroupIds": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

Returns information about a specific elastic cluster.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StopCluster

Service: Amazon DocumentDB Elastic Clusters

Stops the running elastic cluster that is specified by `clusterArn`. The elastic cluster must be in the *available* state.

Request Syntax

```
POST /cluster/clusterArn/stop HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

clusterArn

The ARN identifier of the elastic cluster.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
      "kmsKeyId": "string",
      "preferredBackupWindow": "string",
      "preferredMaintenanceWindow": "string",
```

```
        "shardCapacity": number,
        "shardCount": number,
        "shardInstanceCount": number,
        "shards": [
           {
              "createTime": "string",
              "shardId": "string",
              "status": "string"
           }
        ],
        "status": "string",
        "subnetIds": [ "string" ],
        "vpcSecurityGroupIds": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

Returns information about a specific elastic cluster.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

TagResource

Service: Amazon DocumentDB Elastic Clusters

Adds metadata tags to an elastic cluster resource

Request Syntax

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "tags": {
      "string" : "string"
   }
}
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

The ARN identifier of the elastic cluster resource.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Request Body

The request accepts the following data in JSON format.

tags

The tags that are assigned to the elastic cluster resource.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Key Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: Yes

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors.

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UntagResource

Service: Amazon DocumentDB Elastic Clusters

Removes metadata tags from an elastic cluster resource

Request Syntax

```
DELETE /tags/resourceArn?tagKeys=tagKeys HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

The ARN identifier of the elastic cluster resource.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

tagKeys

The tag keys to be removed from the elastic cluster resource.

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!aws:)[a-zA-Z+-=._:/]+`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors.

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateCluster

Service: Amazon DocumentDB Elastic Clusters

Modifies an elastic cluster. This includes updating admin-username/password, upgrading the API version, and setting up a backup window and maintenance window

Request Syntax

```
PUT /cluster/clusterArn HTTP/1.1
Content-type: application/json

{
   "adminUserPassword": "string",
   "authType": "string",
   "backupRetentionPeriod": number,
   "clientToken": "string",
   "preferredBackupWindow": "string",
   "preferredMaintenanceWindow": "string",
   "shardCapacity": number,
   "shardCount": number,
   "shardInstanceCount": number,
   "subnetIds": [ "string" ],
   "vpcSecurityGroupIds": [ "string" ]
}
```

URI Request Parameters

The request uses the following URI parameters.

clusterArn

The ARN identifier of the elastic cluster.

Required: Yes

Request Body

The request accepts the following data in JSON format.

adminUserPassword

The password associated with the elastic cluster administrator. This password can contain any printable ASCII character except forward slash (/), double quote ("), or the "at" symbol (@).

Constraints: Must contain from 8 to 100 characters.

Type: String

Required: No

authType

The authentication type used to determine where to fetch the password used for accessing the elastic cluster. Valid types are `PLAIN_TEXT` or `SECRET_ARN`.

Type: String

Valid Values: `PLAIN_TEXT | SECRET_ARN`

Required: No

backupRetentionPeriod

The number of days for which automatic snapshots are retained.

Type: Integer

Required: No

clientToken

The client token for the elastic cluster.

Type: String

Required: No

preferredBackupWindow

The daily time range during which automated backups are created if automated backups are enabled, as determined by the `backupRetentionPeriod`.

Type: String

Required: No

preferredMaintenanceWindow

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

Default: a 30-minute window selected at random from an 8-hour block of time for each AWS Region, occurring on a random day of the week.

Valid days: Mon, Tue, Wed, Thu, Fri, Sat, Sun

Constraints: Minimum 30-minute window.

Type: String

Required: No

shardCapacity

The number of vCPUs assigned to each elastic cluster shard. Maximum is 64. Allowed values are 2, 4, 8, 16, 32, 64.

Type: Integer

Required: No

shardCount

The number of shards assigned to the elastic cluster. Maximum is 32.

Type: Integer

Required: No

shardInstanceCount

The number of replica instances applying to all shards in the elastic cluster. A `shardInstanceCount` value of 1 means there is one writer instance, and any additional instances are replicas that can be used for reads and to improve availability.

Type: Integer

Required: No

subnetIds

The Amazon EC2 subnet IDs for the elastic cluster.

Type: Array of strings

Required: No

vpcSecurityGroupIds

A list of EC2 VPC security groups to associate with the elastic cluster.

Type: Array of strings

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "cluster": {
      "adminUserName": "string",
      "authType": "string",
      "backupRetentionPeriod": number,
      "clusterArn": "string",
      "clusterEndpoint": "string",
      "clusterName": "string",
      "createTime": "string",
      "kmsKeyId": "string",
      "preferredBackupWindow": "string",
      "preferredMaintenanceWindow": "string",
      "shardCapacity": number,
      "shardCount": number,
      "shardInstanceCount": number,
      "shards": [
         {
            "createTime": "string",
            "shardId": "string",
            "status": "string"
         }
      ],
      "status": "string",
      "subnetIds": [ "string" ],
      "vpcSecurityGroupIds": [ "string" ]
   }
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

cluster

Returns information about the updated elastic cluster.

Type: Cluster object

Errors

For information about the errors that are common to all actions, see Common Errors.

AccessDeniedException

An exception that occurs when there are not sufficient permissions to perform an action.

HTTP Status Code: 403

ConflictException

There was an access conflict.

HTTP Status Code: 409

InternalServerException

There was an internal server error.

HTTP Status Code: 500

ResourceNotFoundException

The specified resource could not be located.

HTTP Status Code: 404

ThrottlingException

ThrottlingException will be thrown when request was denied due to request throttling.

HTTP Status Code: 429

ValidationException

A structure defining a validation exception.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types

The following data types are supported by Amazon DocumentDB (with MongoDB compatibility):

- AvailabilityZone
- Certificate
- CertificateDetails
- CloudwatchLogsExportConfiguration
- ClusterMasterUserSecret
- DBCluster
- DBClusterMember

- DBClusterParameterGroup
- DBClusterRole
- DBClusterSnapshot
- DBClusterSnapshotAttribute
- DBClusterSnapshotAttributesResult
- DBEngineVersion
- DBInstance
- DBInstanceStatusInfo
- DBSubnetGroup
- Endpoint
- EngineDefaults
- Event
- EventCategoriesMap
- EventSubscription
- Filter
- GlobalCluster
- GlobalClusterMember
- OrderableDBInstanceOption
- Parameter
- PendingCloudwatchLogsExports
- PendingMaintenanceAction
- PendingModifiedValues
- ResourcePendingMaintenanceActions
- Subnet
- Tag
- UpgradeTarget
- VpcSecurityGroupMembership

The following data types are supported by Amazon DocumentDB Elastic Clusters:

- Cluster

- ClusterInList
- ClusterSnapshot
- ClusterSnapshotInList
- PendingMaintenanceActionDetails
- ResourcePendingMaintenanceAction
- Shard
- ValidationExceptionField

Amazon DocumentDB (with MongoDB compatibility)

The following data types are supported by Amazon DocumentDB (with MongoDB compatibility):

- AvailabilityZone
- Certificate
- CertificateDetails
- CloudwatchLogsExportConfiguration
- ClusterMasterUserSecret
- DBCluster
- DBClusterMember
- DBClusterParameterGroup
- DBClusterRole
- DBClusterSnapshot
- DBClusterSnapshotAttribute
- DBClusterSnapshotAttributesResult
- DBEngineVersion
- DBInstance
- DBInstanceStatusInfo
- DBSubnetGroup
- Endpoint
- EngineDefaults
- Event
- EventCategoriesMap

- EventSubscription
- Filter
- GlobalCluster
- GlobalClusterMember
- OrderableDBInstanceOption
- Parameter
- PendingCloudwatchLogsExports
- PendingMaintenanceAction
- PendingModifiedValues
- ResourcePendingMaintenanceActions
- Subnet
- Tag
- UpgradeTarget
- VpcSecurityGroupMembership

AvailabilityZone

Service: Amazon DocumentDB (with MongoDB compatibility)

Information about an Availability Zone.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Name

The name of the Availability Zone.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Certificate

Service: Amazon DocumentDB (with MongoDB compatibility)

A certificate authority (CA) certificate for an AWS account.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

CertificateArn

The Amazon Resource Name (ARN) for the certificate.

Example: `arn:aws:rds:us-east-1::cert:rds-ca-2019`

Type: String

Required: No

CertificateIdentifier

The unique key that identifies a certificate.

Example: `rds-ca-2019`

Type: String

Required: No

CertificateType

The type of the certificate.

Example: CA

Type: String

Required: No

Thumbprint

The thumbprint of the certificate.

Type: String

Required: No

ValidFrom

The starting date-time from which the certificate is valid.

Example: `2019-07-31T17:57:09Z`

Type: Timestamp

Required: No

ValidTill

The date-time after which the certificate is no longer valid.

Example: `2024-07-31T17:57:09Z`

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CertificateDetails

Service: Amazon DocumentDB (with MongoDB compatibility)

Returns the details of the DB instance's server certificate.

For more information, see Updating Your Amazon DocumentDB TLS Certificates and Encrypting Data in Transit in the *Amazon DocumentDB Developer Guide*.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

CAIdentifier

The CA identifier of the CA certificate used for the DB instance's server certificate.

Type: String

Required: No

ValidTill

The expiration date of the DB instance's server certificate.

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CloudwatchLogsExportConfiguration

Service: Amazon DocumentDB (with MongoDB compatibility)

The configuration setting for the log types to be enabled for export to Amazon CloudWatch Logs for a specific instance or cluster.

The `EnableLogTypes` and `DisableLogTypes` arrays determine which logs are exported (or not exported) to CloudWatch Logs. The values within these arrays depend on the engine that is being used.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DisableLogTypes.member.N

The list of log types to disable.

Type: Array of strings

Required: No

EnableLogTypes.member.N

The list of log types to enable.

Type: Array of strings

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ClusterMasterUserSecret

Service: Amazon DocumentDB (with MongoDB compatibility)

Contains the secret managed by Amazon DocumentDB in AWS Secrets Manager for the master user password.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

KmsKeyId

The AWS KMS key identifier that is used to encrypt the secret.

Type: String

Required: No

SecretArn

The Amazon Resource Name (ARN) of the secret.

Type: String

Required: No

SecretStatus

The status of the secret.

The possible status values include the following:

- creating - The secret is being created.
- active - The secret is available for normal use and rotation.
- rotating - The secret is being rotated.
- impaired - The secret can be used to access database credentials, but it can't be rotated. A secret might have this status if, for example, permissions are changed so that Amazon DocumentDB can no longer access either the secret or the KMS key for the secret.

When a secret has this status, you can correct the condition that caused the status. Alternatively, modify the instance to turn off automatic management of database credentials, and then modify the instance again to turn on automatic management of database credentials.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about a cluster.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AssociatedRoles.DBClusterRole.N

Provides a list of the AWS Identity and Access Management (IAM) roles that are associated with the cluster. (IAM) roles that are associated with a cluster grant permission for the cluster to access other AWS services on your behalf.

Type: Array of DBClusterRole objects

Required: No

AvailabilityZones.AvailabilityZone.N

Provides the list of Amazon EC2 Availability Zones that instances in the cluster can be created in.

Type: Array of strings

Required: No

BackupRetentionPeriod

Specifies the number of days for which automatic snapshots are retained.

Type: Integer

Required: No

CloneGroupId

Identifies the clone group to which the DB cluster is associated.

Type: String

Required: No

ClusterCreateTime

Specifies the time when the cluster was created, in Universal Coordinated Time (UTC).

Type: Timestamp

Required: No

DBClusterArn

The Amazon Resource Name (ARN) for the cluster.

Type: String

Required: No

DBClusterIdentifier

Contains a user-supplied cluster identifier. This identifier is the unique key that identifies a cluster.

Type: String

Required: No

DBClusterMembers.DBClusterMember.N

Provides the list of instances that make up the cluster.

Type: Array of DBClusterMember objects

Required: No

DBClusterParameterGroup

Specifies the name of the cluster parameter group for the cluster.

Type: String

Required: No

DbClusterResourceId

The AWS Region-unique, immutable identifier for the cluster. This identifier is found in AWS CloudTrail log entries whenever the AWS KMS key for the cluster is accessed.

Type: String

Required: No

DBSubnetGroup

Specifies information on the subnet group that is associated with the cluster, including the name, description, and subnets in the subnet group.

Type: String

Required: No

DeletionProtection

Specifies whether this cluster can be deleted. If `DeletionProtection` is enabled, the cluster cannot be deleted unless it is modified and `DeletionProtection` is disabled. `DeletionProtection` protects clusters from being accidentally deleted.

Type: Boolean

Required: No

EarliestRestorableTime

The earliest time to which a database can be restored with point-in-time restore.

Type: Timestamp

Required: No

EnabledCloudwatchLogsExports.member.N

A list of log types that this cluster is configured to export to Amazon CloudWatch Logs.

Type: Array of strings

Required: No

Endpoint

Specifies the connection endpoint for the primary instance of the cluster.

Type: String

Required: No

Engine

Provides the name of the database engine to be used for this cluster.

Type: String

Required: No

EngineVersion

Indicates the database engine version.

Type: String

Required: No

HostedZoneId

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

Type: String

Required: No

KmsKeyId

If `StorageEncrypted` is `true`, the AWS KMS key identifier for the encrypted cluster.

Type: String

Required: No

LatestRestorableTime

Specifies the latest time to which a database can be restored with point-in-time restore.

Type: Timestamp

Required: No

MasterUsername

Contains the master user name for the cluster.

Type: String

Required: No

MasterUserSecret

The secret managed by Amazon DocumentDB in AWS Secrets Manager for the master user password.

Type: ClusterMasterUserSecret object

Required: No

MultiAZ

Specifies whether the cluster has instances in multiple Availability Zones.

Type: Boolean

Required: No

PercentProgress

Specifies the progress of the operation as a percentage.

Type: String

Required: No

Port

Specifies the port that the database engine is listening on.

Type: Integer

Required: No

PreferredBackupWindow

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

Type: String

Required: No

PreferredMaintenanceWindow

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Type: String

Required: No

ReaderEndpoint

The reader endpoint for the cluster. The reader endpoint for a cluster load balances connections across the Amazon DocumentDB replicas that are available in a cluster. As clients request new connections to the reader endpoint, Amazon DocumentDB distributes the connection requests among the Amazon DocumentDB replicas in the cluster. This functionality can help balance your read workload across multiple Amazon DocumentDB replicas in your cluster.

If a failover occurs, and the Amazon DocumentDB replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Amazon DocumentDB replicas in the cluster, you can then reconnect to the reader endpoint.

Type: String

Required: No

ReadReplicaIdentifiers.ReadReplicaIdentifier.N

Contains one or more identifiers of the secondary clusters that are associated with this cluster.

Type: Array of strings

Required: No

ReplicationSourceIdentifier

Contains the identifier of the source cluster if this cluster is a secondary cluster.

Type: String

Required: No

Status

Specifies the current state of this cluster.

Type: String

Required: No

StorageEncrypted

Specifies whether the cluster is encrypted.

Type: Boolean

Required: No

StorageType

Storage type associated with your cluster

For information on storage types for Amazon DocumentDB clusters, see Cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Valid values for storage type - `standard | iopt1`

Default value is `standard`

Type: String

Required: No

VpcSecurityGroups.VpcSecurityGroupMembership.N

Provides a list of virtual private cloud (VPC) security groups that the cluster belongs to.

Type: Array of VpcSecurityGroupMembership objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBClusterMember

Service: Amazon DocumentDB (with MongoDB compatibility)

Contains information about an instance that is part of a cluster.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBClusterParameterGroupStatus

Specifies the status of the cluster parameter group for this member of the DB cluster.

Type: String

Required: No

DBInstanceIdentifier

Specifies the instance identifier for this member of the cluster.

Type: String

Required: No

IsClusterWriter

A value that is `true` if the cluster member is the primary instance for the cluster and `false` otherwise.

Type: Boolean

Required: No

PromotionTier

A value that specifies the order in which an Amazon DocumentDB replica is promoted to the primary instance after a failure of the existing primary instance.

Type: Integer

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBClusterParameterGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about a cluster parameter group.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBClusterParameterGroupArn

The Amazon Resource Name (ARN) for the cluster parameter group.

Type: String

Required: No

DBClusterParameterGroupName

Provides the name of the cluster parameter group.

Type: String

Required: No

DBParameterGroupFamily

Provides the name of the parameter group family that this cluster parameter group is compatible with.

Type: String

Required: No

Description

Provides the customer-specified description for this cluster parameter group.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBClusterRole

Service: Amazon DocumentDB (with MongoDB compatibility)

Describes an AWS Identity and Access Management (IAM) role that is associated with a cluster.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

RoleArn

The Amazon Resource Name (ARN) of the IAMrole that is associated with the DB cluster.

Type: String

Required: No

Status

Describes the state of association between the IAMrole and the cluster. The `Status` property returns one of the following values:

- `ACTIVE` - The IAMrole ARN is associated with the cluster and can be used to access other AWS services on your behalf.
- `PENDING` - The IAMrole ARN is being associated with the cluster.
- `INVALID` - The IAMrole ARN is associated with the cluster, but the cluster cannot assume the IAMrole to access other AWS services on your behalf.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBClusterSnapshot

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about a cluster snapshot.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AvailabilityZones.AvailabilityZone.N

Provides the list of Amazon EC2 Availability Zones that instances in the cluster snapshot can be restored in.

Type: Array of strings

Required: No

ClusterCreateTime

Specifies the time when the cluster was created, in Universal Coordinated Time (UTC).

Type: Timestamp

Required: No

DBClusterIdentifier

Specifies the cluster identifier of the cluster that this cluster snapshot was created from.

Type: String

Required: No

DBClusterSnapshotArn

The Amazon Resource Name (ARN) for the cluster snapshot.

Type: String

Required: No

DBClusterSnapshotIdentifier

Specifies the identifier for the cluster snapshot.

Type: String

Required: No

Engine

Specifies the name of the database engine.

Type: String

Required: No

EngineVersion

Provides the version of the database engine for this cluster snapshot.

Type: String

Required: No

KmsKeyId

If `StorageEncrypted` is `true`, the AWS KMS key identifier for the encrypted cluster snapshot.

Type: String

Required: No

MasterUsername

Provides the master user name for the cluster snapshot.

Type: String

Required: No

PercentProgress

Specifies the percentage of the estimated data that has been transferred.

Type: Integer

Required: No

Port

Specifies the port that the cluster was listening on at the time of the snapshot.

Type: Integer

Required: No

SnapshotCreateTime

Provides the time when the snapshot was taken, in UTC.

Type: Timestamp

Required: No

SnapshotType

Provides the type of the cluster snapshot.

Type: String

Required: No

SourceDBClusterSnapshotArn

If the cluster snapshot was copied from a source cluster snapshot, the ARN for the source cluster snapshot; otherwise, a null value.

Type: String

Required: No

Status

Specifies the status of this cluster snapshot.

Type: String

Required: No

StorageEncrypted

Specifies whether the cluster snapshot is encrypted.

Type: Boolean

Required: No

StorageType

Storage type associated with your cluster snapshot

For information on storage types for Amazon DocumentDB clusters, see Cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Valid values for storage type - `standard | iopt1`

Default value is `standard`

Type: String

Required: No

VpcId

Provides the virtual private cloud (VPC) ID that is associated with the cluster snapshot.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBClusterSnapshotAttribute

Service: Amazon DocumentDB (with MongoDB compatibility)

Contains the name and values of a manual cluster snapshot attribute.

Manual cluster snapshot attributes are used to authorize other AWS accounts to restore a manual cluster snapshot.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AttributeName

The name of the manual cluster snapshot attribute.

The attribute named `restore` refers to the list of AWS accounts that have permission to copy or restore the manual cluster snapshot.

Type: String

Required: No

AttributeValues.AttributeValue.N

The values for the manual cluster snapshot attribute.

If the `AttributeName` field is set to `restore`, then this element returns a list of IDs of the AWS accounts that are authorized to copy or restore the manual cluster snapshot. If a value of `all` is in the list, then the manual cluster snapshot is public and available for any AWS account to copy or restore.

Type: Array of strings

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBClusterSnapshotAttributesResult

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about the attributes that are associated with a cluster snapshot.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBClusterSnapshotAttributes.DBClusterSnapshotAttribute.N

The list of attributes and values for the cluster snapshot.

Type: Array of DBClusterSnapshotAttribute objects

Required: No

DBClusterSnapshotIdentifier

The identifier of the cluster snapshot that the attributes apply to.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBEngineVersion

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about an engine version.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBEngineDescription

The description of the database engine.

Type: String

Required: No

DBEngineVersionDescription

The description of the database engine version.

Type: String

Required: No

DBParameterGroupFamily

The name of the parameter group family for the database engine.

Type: String

Required: No

Engine

The name of the database engine.

Type: String

Required: No

EngineVersion

The version number of the database engine.

Type: String

Required: No

ExportableLogTypes.member.N

The types of logs that the database engine has available for export to Amazon CloudWatch Logs.

Type: Array of strings

Required: No

SupportedCACertificateIdentifiers.member.N

A list of the supported CA certificate identifiers.

For more information, see Updating Your Amazon DocumentDB TLS Certificates and Encrypting Data in Transit in the *Amazon DocumentDB Developer Guide*.

Type: Array of strings

Required: No

SupportsCertificateRotationWithoutRestart

Indicates whether the engine version supports rotating the server certificate without rebooting the DB instance.

Type: Boolean

Required: No

SupportsLogExportsToCloudwatchLogs

A value that indicates whether the engine version supports exporting the log types specified by `ExportableLogTypes` to CloudWatch Logs.

Type: Boolean

Required: No

ValidUpgradeTarget.UpgradeTarget.N

A list of engine versions that this database engine version can be upgraded to.

Type: Array of UpgradeTarget objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBInstance

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about an instance.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AutoMinorVersionUpgrade

Does not apply. This parameter does not apply to Amazon DocumentDB. Amazon DocumentDB does not perform minor version upgrades regardless of the value set.

Type: Boolean

Required: No

AvailabilityZone

Specifies the name of the Availability Zone that the instance is located in.

Type: String

Required: No

BackupRetentionPeriod

Specifies the number of days for which automatic snapshots are retained.

Type: Integer

Required: No

CACertificateIdentifier

The identifier of the CA certificate for this DB instance.

Type: String

Required: No

CertificateDetails

The details of the DB instance's server certificate.

Type: CertificateDetails object

Required: No

CopyTagsToSnapshot

A value that indicates whether to copy tags from the DB instance to snapshots of the DB instance. By default, tags are not copied.

Type: Boolean

Required: No

DBClusterIdentifier

Contains the name of the cluster that the instance is a member of if the instance is a member of a cluster.

Type: String

Required: No

DBInstanceArn

The Amazon Resource Name (ARN) for the instance.

Type: String

Required: No

DBInstanceClass

Contains the name of the compute and memory capacity class of the instance.

Type: String

Required: No

DBInstanceIdentifier

Contains a user-provided database identifier. This identifier is the unique key that identifies an instance.

Type: String

Required: No

DBInstanceStatus

Specifies the current state of this database.

Type: String

Required: No

DbiResourceId

The AWS Region-unique, immutable identifier for the instance. This identifier is found in AWS CloudTrail log entries whenever the AWS KMS key for the instance is accessed.

Type: String

Required: No

DBSubnetGroup

Specifies information on the subnet group that is associated with the instance, including the name, description, and subnets in the subnet group.

Type: DBSubnetGroup object

Required: No

EnabledCloudwatchLogsExports.member.N

A list of log types that this instance is configured to export to CloudWatch Logs.

Type: Array of strings

Required: No

Endpoint

Specifies the connection endpoint.

Type: Endpoint object

Required: No

Engine

Provides the name of the database engine to be used for this instance.

Type: String

Required: No

EngineVersion

Indicates the database engine version.

Type: String

Required: No

InstanceCreateTime

Provides the date and time that the instance was created.

Type: Timestamp

Required: No

KmsKeyId

If `StorageEncrypted` is `true`, the AWS KMS key identifier for the encrypted instance.

Type: String

Required: No

LatestRestorableTime

Specifies the latest time to which a database can be restored with point-in-time restore.

Type: Timestamp

Required: No

PendingModifiedValues

Specifies that changes to the instance are pending. This element is included only when changes are pending. Specific changes are identified by subelements.

Type: PendingModifiedValues object

Required: No

PerformanceInsightsEnabled

Set to `true` if Amazon RDS Performance Insights is enabled for the DB instance, and otherwise `false`.

Type: Boolean

Required: No

PerformanceInsightsKMSKeyId

The AWS KMS key identifier for encryption of Performance Insights data. The AWS KMS key ID is the Amazon Resource Name (ARN), AWS KMS key identifier, or the AWS KMS key alias for the AWS KMS encryption key.

Type: String

Required: No

PreferredBackupWindow

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

Type: String

Required: No

PreferredMaintenanceWindow

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Type: String

Required: No

PromotionTier

A value that specifies the order in which an Amazon DocumentDB replica is promoted to the primary instance after a failure of the existing primary instance.

Type: Integer

Required: No

PubliclyAccessible

Not supported. Amazon DocumentDB does not currently support public endpoints. The value of `PubliclyAccessible` is always `false`.

Type: Boolean

Required: No

StatusInfos.DBInstanceStatusInfo.N

The status of a read replica. If the instance is not a read replica, this is blank.

Type: Array of DBInstanceStatusInfo objects

Required: No

StorageEncrypted

Specifies whether or not the instance is encrypted.

Type: Boolean

Required: No

VpcSecurityGroups.VpcSecurityGroupMembership.N

Provides a list of VPC security group elements that the instance belongs to.

Type: Array of VpcSecurityGroupMembership objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBInstanceStatusInfo

Service: Amazon DocumentDB (with MongoDB compatibility)

Provides a list of status information for an instance.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Message

Details of the error if there is an error for the instance. If the instance is not in an error state, this value is blank.

Type: String

Required: No

Normal

A Boolean value that is `true` if the instance is operating normally, or `false` if the instance is in an error state.

Type: Boolean

Required: No

Status

Status of the instance. For a `StatusType` of read replica, the values can be `replicating`, error, `stopped`, or `terminated`.

Type: String

Required: No

StatusType

This value is currently "`read replication`."

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DBSubnetGroup

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about a subnet group.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBSubnetGroupArn

The Amazon Resource Name (ARN) for the DB subnet group.

Type: String

Required: No

DBSubnetGroupDescription

Provides the description of the subnet group.

Type: String

Required: No

DBSubnetGroupName

The name of the subnet group.

Type: String

Required: No

SubnetGroupStatus

Provides the status of the subnet group.

Type: String

Required: No

Subnets.Subnet.N

Detailed information about one or more subnets within a subnet group.

Type: Array of Subnet objects

Required: No

VpcId

Provides the virtual private cloud (VPC) ID of the subnet group.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Endpoint

Service: Amazon DocumentDB (with MongoDB compatibility)

Network information for accessing a cluster or instance. Client programs must specify a valid endpoint to access these Amazon DocumentDB resources.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Address

Specifies the DNS address of the instance.

Type: String

Required: No

HostedZoneId

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

Type: String

Required: No

Port

Specifies the port that the database engine is listening on.

Type: Integer

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Java V2
- AWS SDK for Ruby V3

EngineDefaults

Service: Amazon DocumentDB (with MongoDB compatibility)

Contains the result of a successful invocation of the `DescribeEngineDefaultClusterParameters` operation.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBParameterGroupFamily

The name of the cluster parameter group family to return the engine parameter information for.

Type: String

Required: No

Marker

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Type: String

Required: No

Parameters.Parameter.N

The parameters of a particular cluster parameter group family.

Type: Array of Parameter objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Event

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about an event.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Date

Specifies the date and time of the event.

Type: Timestamp

Required: No

EventCategories.EventCategory.N

Specifies the category for the event.

Type: Array of strings

Required: No

Message

Provides the text of this event.

Type: String

Required: No

SourceArn

The Amazon Resource Name (ARN) for the event.

Type: String

Required: No

SourceIdentifier

Provides the identifier for the source of the event.

Type: String

Required: No

SourceType

Specifies the source type for this event.

Type: String

Valid Values: `db-instance | db-parameter-group | db-security-group | db-snapshot | db-cluster | db-cluster-snapshot`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

EventCategoriesMap

Service: Amazon DocumentDB (with MongoDB compatibility)

An event source type, accompanied by one or more event category names.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

EventCategories.EventCategory.N

The event categories for the specified source type.

Type: Array of strings

Required: No

SourceType

The source type that the returned categories belong to.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

EventSubscription

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about an event to which you have subscribed.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

CustomerAwsId

The AWS customer account that is associated with the Amazon DocumentDB event notification subscription.

Type: String

Required: No

CustSubscriptionId

The Amazon DocumentDB event notification subscription ID.

Type: String

Required: No

Enabled

A Boolean value indicating whether the subscription is enabled. A value of `true` indicates that the subscription is enabled.

Type: Boolean

Required: No

EventCategoriesList.EventCategory.N

A list of event categories for the Amazon DocumentDB event notification subscription.

Type: Array of strings

Required: No

EventSubscriptionArn

The Amazon Resource Name (ARN) for the event subscription.

Type: String

Required: No

SnsTopicArn

The topic ARN of the Amazon DocumentDB event notification subscription.

Type: String

Required: No

SourceIdsList.SourceId.N

A list of source IDs for the Amazon DocumentDB event notification subscription.

Type: Array of strings

Required: No

SourceType

The source type for the Amazon DocumentDB event notification subscription.

Type: String

Required: No

Status

The status of the Amazon DocumentDB event notification subscription.

Constraints:

Can be one of the following: `creating`, `modifying`, `deleting`, `active`, `no-permission`, `topic-not-exist`

The `no-permission` status indicates that Amazon DocumentDB no longer has permission to post to the SNS topic. The `topic-not-exist` status indicates that the topic was deleted after the subscription was created.

Type: String

Required: No

SubscriptionCreationTime

The time at which the Amazon DocumentDB event notification subscription was created.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Filter

Service: Amazon DocumentDB (with MongoDB compatibility)

A named set of filter values, used to return a more specific list of results. You can use a filter to match a set of resources by specific criteria, such as IDs.

Wildcards are not supported in filters.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Name

The name of the filter. Filter names are case sensitive.

Type: String

Required: Yes

Values.Value.N

One or more filter values. Filter values are case sensitive.

Type: Array of strings

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

GlobalCluster

Service: Amazon DocumentDB (with MongoDB compatibility)

A data type representing an Amazon DocumentDB global cluster.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DatabaseName

The default database name within the new global cluster.

Type: String

Required: No

DeletionProtection

The deletion protection setting for the new global cluster.

Type: Boolean

Required: No

Engine

The Amazon DocumentDB database engine used by the global cluster.

Type: String

Required: No

EngineVersion

Indicates the database engine version.

Type: String

Required: No

GlobalClusterArn

The Amazon Resource Name (ARN) for the global cluster.

Type: String

Required: No

GlobalClusterIdentifier

Contains a user-supplied global cluster identifier. This identifier is the unique key that identifies a global cluster.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[A-Za-z][0-9A-Za-z-:._]*`

Required: No

GlobalClusterMembers.GlobalClusterMember.N

The list of cluster IDs for secondary clusters within the global cluster. Currently limited to one item.

Type: Array of GlobalClusterMember objects

Required: No

GlobalClusterResourceId

The AWS RegionRegion-unique, immutable identifier for the global database cluster. This identifier is found in AWS CloudTrail log entries whenever the AWS KMS customer master key (CMK) for the cluster is accessed.

Type: String

Required: No

Status

Specifies the current state of this global cluster.

Type: String

Required: No

StorageEncrypted

The storage encryption setting for the global cluster.

Type: Boolean

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

GlobalClusterMember

Service: Amazon DocumentDB (with MongoDB compatibility)

A data structure with information about any primary and secondary clusters associated with an Amazon DocumentDB global clusters.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

DBClusterArn

The Amazon Resource Name (ARN) for each Amazon DocumentDB cluster.

Type: String

Required: No

IsWriter

Specifies whether the Amazon DocumentDB cluster is the primary cluster (that is, has read-write capability) for the Amazon DocumentDB global cluster with which it is associated.

Type: Boolean

Required: No

Readers.member.N

The Amazon Resource Name (ARN) for each read-only secondary cluster associated with the Amazon DocumentDB global cluster.

Type: Array of strings

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

OrderableDBInstanceOption

Service: Amazon DocumentDB (with MongoDB compatibility)

The options that are available for an instance.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AvailabilityZones.AvailabilityZone.N

A list of Availability Zones for an instance.

Type: Array of AvailabilityZone objects

Required: No

DBInstanceClass

The instance class for an instance.

Type: String

Required: No

Engine

The engine type of an instance.

Type: String

Required: No

EngineVersion

The engine version of an instance.

Type: String

Required: No

LicenseModel

The license model for an instance.

Type: String

Required: No

StorageType

The storage type to associate with the DB cluster

Type: String

Required: No

Vpc

Indicates whether an instance is in a virtual private cloud (VPC).

Type: Boolean

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Parameter

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about an individual parameter.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AllowedValues

Specifies the valid range of values for the parameter.

Type: String

Required: No

ApplyMethod

Indicates when to apply parameter updates.

Type: String

Valid Values: `immediate | pending-reboot`

Required: No

ApplyType

Specifies the engine-specific parameters type.

Type: String

Required: No

DataType

Specifies the valid data type for the parameter.

Type: String

Required: No

Description

Provides a description of the parameter.

Type: String

Required: No

IsModifiable

Indicates whether (`true`) or not (`false`) the parameter can be modified. Some parameters have security or operational implications that prevent them from being changed.

Type: Boolean

Required: No

MinimumEngineVersion

The earliest engine version to which the parameter can apply.

Type: String

Required: No

ParameterName

Specifies the name of the parameter.

Type: String

Required: No

ParameterValue

Specifies the value of the parameter.

Type: String

Required: No

Source

Indicates the source of the parameter value.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

PendingCloudwatchLogsExports

Service: Amazon DocumentDB (with MongoDB compatibility)

A list of the log types whose configuration is still pending. These log types are in the process of being activated or deactivated.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

LogTypesToDisable.member.N

Log types that are in the process of being enabled. After they are enabled, these log types are exported to Amazon CloudWatch Logs.

Type: Array of strings

Required: No

LogTypesToEnable.member.N

Log types that are in the process of being deactivated. After they are deactivated, these log types aren't exported to CloudWatch Logs.

Type: Array of strings

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

PendingMaintenanceAction

Service: Amazon DocumentDB (with MongoDB compatibility)

Provides information about a pending maintenance action for a resource.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Action

The type of pending maintenance action that is available for the resource.

Type: String

Required: No

AutoAppliedAfterDate

The date of the maintenance window when the action is applied. The maintenance action is applied to the resource during its first maintenance window after this date. If this date is specified, any `next-maintenance` opt-in requests are ignored.

Type: Timestamp

Required: No

CurrentApplyDate

The effective date when the pending maintenance action is applied to the resource.

Type: Timestamp

Required: No

Description

A description providing more detail about the maintenance action.

Type: String

Required: No

ForcedApplyDate

The date when the maintenance action is automatically applied. The maintenance action is applied to the resource on this date regardless of the maintenance window for the resource. If this date is specified, any `immediate` opt-in requests are ignored.

Type: Timestamp

Required: No

OptInStatus

Indicates the type of opt-in request that has been received for the resource.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

PendingModifiedValues

Service: Amazon DocumentDB (with MongoDB compatibility)

One or more modified settings for an instance. These modified settings have been requested, but haven't been applied yet.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AllocatedStorage

Contains the new `AllocatedStorage` size for then instance that will be applied or is currently being applied.

Type: Integer

Required: No

BackupRetentionPeriod

Specifies the pending number of days for which automated backups are retained.

Type: Integer

Required: No

CACertificateIdentifier

Specifies the identifier of the certificate authority (CA) certificate for the DB instance.

Type: String

Required: No

DBInstanceClass

Contains the new `DBInstanceClass` for the instance that will be applied or is currently being applied.

Type: String

Required: No

DBInstanceIdentifier

Contains the new `DBInstanceIdentifier` for the instance that will be applied or is currently being applied.

Type: String

Required: No

DBSubnetGroupName

The new subnet group for the instance.

Type: String

Required: No

EngineVersion

Indicates the database engine version.

Type: String

Required: No

Iops

Specifies the new Provisioned IOPS value for the instance that will be applied or is currently being applied.

Type: Integer

Required: No

LicenseModel

The license model for the instance.

Valid values: `license-included`, `bring-your-own-license`, `general-public-license`

Type: String

Required: No

MasterUserPassword

Contains the pending or currently in-progress change of the master credentials for the instance.

Type: String

Required: No

MultiAZ

Indicates that the Single-AZ instance is to change to a Multi-AZ deployment.

Type: Boolean

Required: No

PendingCloudwatchLogsExports

A list of the log types whose configuration is still pending. These log types are in the process of being activated or deactivated.

Type: PendingCloudwatchLogsExports object

Required: No

Port

Specifies the pending port for the instance.

Type: Integer

Required: No

StorageType

Specifies the storage type to be associated with the instance.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ResourcePendingMaintenanceActions

Service: Amazon DocumentDB (with MongoDB compatibility)

Represents the output of ApplyPendingMaintenanceAction.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

PendingMaintenanceActionDetails.PendingMaintenanceAction.N

A list that provides details about the pending maintenance actions for the resource.

Type: Array of PendingMaintenanceAction objects

Required: No

ResourceIdentifier

The Amazon Resource Name (ARN) of the resource that has pending maintenance actions.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Subnet

Service: Amazon DocumentDB (with MongoDB compatibility)

Detailed information about a subnet.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

SubnetAvailabilityZone

Specifies the Availability Zone for the subnet.

Type: AvailabilityZone object

Required: No

SubnetIdentifier

Specifies the identifier of the subnet.

Type: String

Required: No

SubnetStatus

Specifies the status of the subnet.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

Tag

Service: Amazon DocumentDB (with MongoDB compatibility)

Metadata assigned to an Amazon DocumentDB resource consisting of a key-value pair.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Key

The required name of the tag. The string value can be from 1 to 128 Unicode characters in length and can't be prefixed with "`aws:`" or "`rds:`". The string can contain only the set of Unicode letters, digits, white space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$").

Type: String

Required: No

Value

The optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and can't be prefixed with "`aws:`" or "`rds:`". The string can contain only the set of Unicode letters, digits, white space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$").

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

UpgradeTarget

Service: Amazon DocumentDB (with MongoDB compatibility)

The version of the database engine that an instance can be upgraded to.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

AutoUpgrade

A value that indicates whether the target version is applied to any source DB instances that have `AutoMinorVersionUpgrade` set to `true`.

Type: Boolean

Required: No

Description

The version of the database engine that an instance can be upgraded to.

Type: String

Required: No

Engine

The name of the upgrade target database engine.

Type: String

Required: No

EngineVersion

The version number of the upgrade target database engine.

Type: String

Required: No

IsMajorVersionUpgrade

A value that indicates whether a database engine is upgraded to a major version.

Type: Boolean

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

VpcSecurityGroupMembership

Service: Amazon DocumentDB (with MongoDB compatibility)

Used as a response element for queries on virtual private cloud (VPC) security group membership.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

Status

The status of the VPC security group.

Type: String

Required: No

VpcSecurityGroupId

The name of the VPC security group.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Amazon DocumentDB Elastic Clusters

The following data types are supported by Amazon DocumentDB Elastic Clusters:

- Cluster
- ClusterInList
- ClusterSnapshot
- ClusterSnapshotInList
- PendingMaintenanceActionDetails
- ResourcePendingMaintenanceAction
- Shard
- ValidationExceptionField

Cluster

Service: Amazon DocumentDB Elastic Clusters

Returns information about a specific elastic cluster.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

adminUserName

The name of the elastic cluster administrator.

Type: String

Required: Yes

authType

The authentication type for the elastic cluster.

Type: String

Valid Values: `PLAIN_TEXT | SECRET_ARN`

Required: Yes

clusterArn

The ARN identifier of the elastic cluster.

Type: String

Required: Yes

clusterEndpoint

The URL used to connect to the elastic cluster.

Type: String

Required: Yes

clusterName

The name of the elastic cluster.

Type: String

Required: Yes

createTime

The time when the elastic cluster was created in Universal Coordinated Time (UTC).

Type: String

Required: Yes

kmsKeyId

The KMS key identifier to use to encrypt the elastic cluster.

Type: String

Required: Yes

preferredMaintenanceWindow

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

Type: String

Required: Yes

shardCapacity

The number of vCPUs assigned to each elastic cluster shard. Maximum is 64. Allowed values are 2, 4, 8, 16, 32, 64.

Type: Integer

Required: Yes

shardCount

The number of shards assigned to the elastic cluster. Maximum is 32.

Type: Integer

Required: Yes

status

The status of the elastic cluster.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | UPDATING | VPC_ENDPOINT_LIMIT_EXCEEDED | IP_ADDRESS_LIMIT_EXCEEDED | INVALID_SECURITY_GROUP_ID | INVALID_SUBNET_ID | INACCESSIBLE_ENCRYPTION_CREDS | INACCESSIBLE_SECRET_ARN | INACCESSIBLE_VPC_ENDPOINT | INCOMPATIBLE_NETWORK | MERGING | MODIFYING | SPLITTING | COPYING | STARTING | STOPPING | STOPPED | MAINTENANCE | INACCESSIBLE_ENCRYPTION_CREDENTIALS_RECOVERABLE`

Required: Yes

subnetIds

The Amazon EC2 subnet IDs for the elastic cluster.

Type: Array of strings

Required: Yes

vpcSecurityGroupIds

A list of EC2 VPC security groups associated with thie elastic cluster.

Type: Array of strings

Required: Yes

backupRetentionPeriod

The number of days for which automatic snapshots are retained.

Type: Integer

Required: No

preferredBackupWindow

The daily time range during which automated backups are created if automated backups are enabled, as determined by `backupRetentionPeriod`.

Type: String

Required: No

shardInstanceCount

The number of replica instances applying to all shards in the cluster. A `shardInstanceCount` value of 1 means there is one writer instance, and any additional instances are replicas that can be used for reads and to improve availability.

Type: Integer

Required: No

shards

The total number of shards in the cluster.

Type: Array of Shard objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ClusterInList

Service: Amazon DocumentDB Elastic Clusters

A list of Amazon DocumentDB elastic clusters.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

clusterArn

The ARN identifier of the elastic cluster.

Type: String

Required: Yes

clusterName

The name of the elastic cluster.

Type: String

Required: Yes

status

The status of the elastic cluster.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | UPDATING | VPC_ENDPOINT_LIMIT_EXCEEDED | IP_ADDRESS_LIMIT_EXCEEDED | INVALID_SECURITY_GROUP_ID | INVALID_SUBNET_ID | INACCESSIBLE_ENCRYPTION_CREDS | INACCESSIBLE_SECRET_ARN | INACCESSIBLE_VPC_ENDPOINT | INCOMPATIBLE_NETWORK | MERGING | MODIFYING | SPLITTING | COPYING | STARTING | STOPPING | STOPPED | MAINTENANCE | INACCESSIBLE_ENCRYPTION_CREDENTIALS_RECOVERABLE`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ClusterSnapshot

Service: Amazon DocumentDB Elastic Clusters

Returns information about a specific elastic cluster snapshot.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

adminUserName

The name of the elastic cluster administrator.

Type: String

Required: Yes

clusterArn

The ARN identifier of the elastic cluster.

Type: String

Required: Yes

clusterCreationTime

The time when the elastic cluster was created in Universal Coordinated Time (UTC).

Type: String

Required: Yes

kmsKeyId

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are creating a cluster using the same Amazon account that owns this KMS encryption key, you can use the KMS key alias instead of the ARN as the KMS encryption key. If an encryption key is not specified here, Amazon DocumentDB uses the default encryption key that KMS creates for your account. Your account has a different default encryption key for each Amazon Region.

Type: String

Required: Yes

snapshotArn

The ARN identifier of the elastic cluster snapshot.

Type: String

Required: Yes

snapshotCreationTime

The time when the elastic cluster snapshot was created in Universal Coordinated Time (UTC).

Type: String

Required: Yes

snapshotName

The name of the elastic cluster snapshot.

Type: String

Required: Yes

status

The status of the elastic cluster snapshot.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | UPDATING | VPC_ENDPOINT_LIMIT_EXCEEDED | IP_ADDRESS_LIMIT_EXCEEDED | INVALID_SECURITY_GROUP_ID | INVALID_SUBNET_ID | INACCESSIBLE_ENCRYPTION_CREDS | INACCESSIBLE_SECRET_ARN | INACCESSIBLE_VPC_ENDPOINT | INCOMPATIBLE_NETWORK | MERGING | MODIFYING | SPLITTING | COPYING | STARTING | STOPPING | STOPPED | MAINTENANCE | INACCESSIBLE_ENCRYPTION_CREDENTIALS_RECOVERABLE`

Required: Yes

subnetIds

The Amazon EC2 subnet IDs for the elastic cluster.

Type: Array of strings

Required: Yes

vpcSecurityGroupIds

A list of EC2 VPC security groups to associate with the elastic cluster.

Type: Array of strings

Required: Yes

snapshotType

The type of cluster snapshots to be returned. You can specify one of the following values:

- `automated` - Return all cluster snapshots that Amazon DocumentDB has automatically created for your AWS account.
- `manual` - Return all cluster snapshots that you have manually created for your AWS account.

Type: String

Valid Values: `MANUAL | AUTOMATED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ClusterSnapshotInList

Service: Amazon DocumentDB Elastic Clusters

A list of elastic cluster snapshots.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

clusterArn

The ARN identifier of the elastic cluster.

Type: String

Required: Yes

snapshotArn

The ARN identifier of the elastic cluster snapshot.

Type: String

Required: Yes

snapshotCreationTime

The time when the elastic cluster snapshot was created in Universal Coordinated Time (UTC).

Type: String

Required: Yes

snapshotName

The name of the elastic cluster snapshot.

Type: String

Required: Yes

status

The status of the elastic cluster snapshot.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | UPDATING | VPC_ENDPOINT_LIMIT_EXCEEDED | IP_ADDRESS_LIMIT_EXCEEDED | INVALID_SECURITY_GROUP_ID | INVALID_SUBNET_ID | INACCESSIBLE_ENCRYPTION_CREDS | INACCESSIBLE_SECRET_ARN | INACCESSIBLE_VPC_ENDPOINT | INCOMPATIBLE_NETWORK | MERGING | MODIFYING | SPLITTING | COPYING | STARTING | STOPPING | STOPPED | MAINTENANCE | INACCESSIBLE_ENCRYPTION_CREDENTIALS_RECOVERABLE`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

PendingMaintenanceActionDetails

Service: Amazon DocumentDB Elastic Clusters

Retrieves the details of maintenance actions that are pending.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

action

Displays the specific action of a pending maintenance action.

Type: String

Required: Yes

autoAppliedAfterDate

Displays the date of the maintenance window when the action is applied. The maintenance action is applied to the resource during its first maintenance window after this date. If this date is specified, any `NEXT_MAINTENANCE` `optInType` requests are ignored.

Type: String

Required: No

currentApplyDate

Displays the effective date when the pending maintenance action is applied to the resource.

Type: String

Required: No

description

Displays a description providing more detail about the maintenance action.

Type: String

Required: No

forcedApplyDate

Displays the date when the maintenance action is automatically applied. The maintenance action is applied to the resource on this date regardless of the maintenance window for the resource. If this date is specified, any `IMMEDIATE` `optInType` requests are ignored.

Type: String

Required: No

optInStatus

Displays the type of `optInType` request that has been received for the resource.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ResourcePendingMaintenanceAction

Service: Amazon DocumentDB Elastic Clusters

Provides information about a pending maintenance action for a resource.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

pendingMaintenanceActionDetails

Provides information about a pending maintenance action for a resource.

Type: Array of PendingMaintenanceActionDetails objects

Required: No

resourceArn

The Amazon DocumentDB Amazon Resource Name (ARN) of the resource to which the pending maintenance action applies.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Shard

Service: Amazon DocumentDB Elastic Clusters

The name of the shard.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

createTime

The time when the shard was created in Universal Coordinated Time (UTC).

Type: String

Required: Yes

shardId

The ID of the shard.

Type: String

Required: Yes

status

The current status of the shard.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | UPDATING | VPC_ENDPOINT_LIMIT_EXCEEDED | IP_ADDRESS_LIMIT_EXCEEDED | INVALID_SECURITY_GROUP_ID | INVALID_SUBNET_ID | INACCESSIBLE_ENCRYPTION_CREDS | INACCESSIBLE_SECRET_ARN | INACCESSIBLE_VPC_ENDPOINT | INCOMPATIBLE_NETWORK | MERGING | MODIFYING | SPLITTING | COPYING | STARTING | STOPPING | STOPPED | MAINTENANCE | INACCESSIBLE_ENCRYPTION_CREDENTIALS_RECOVERABLE`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ValidationExceptionField

Service: Amazon DocumentDB Elastic Clusters

A specific field in which a given validation exception occurred.

Contents

> **Note**
>
> In the following list, the required parameters are described first.

message

An error message describing the validation exception in this field.

Type: String

Required: Yes

name

The name of the field where the validation exception occurred.

Type: String

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

NotAuthorized

You do not have permission to perform this action.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signing AWS API requests in the *IAM User Guide*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: `AWS4-HMAC-SHA256`

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key/YYYYMMDD/region/service/* aws4_request.

For more information, see Create a signed AWS API request in the *IAM User Guide*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: `20120325T120000Z`.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Elements of an AWS API request signature in the *IAM User Guide*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS STS, see AWS services that work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from AWS STS, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Create a signed AWS API request in the *IAM User Guide*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

Release notes

These release notes describe the Amazon DocumentDB features, improvements, and bug fixes by release date. The release notes include updates for all Amazon DocumentDB engine versions as they occur.

You can determine the current Amazon DocumentDB engine patch version by running the following command:

```
db.runCommand({getEngineVersion: 1})
```

If your cluster is not on the latest version of the engine, it is likely that you have pending maintenance available that will upgrade your engine. For more information, see Maintaining Amazon DocumentDB in the Developer Guide.

You can filter new Amazon DocumentDB features on the What's New with Database? page. For **Products**, choose **Amazon DocumentDB**. Then search using keywords such as **`elastic clusters`** or **`vector search`**.

Topics

- June 18, 2025
- May 8, 2025
- April 2, 2025
- March 24, 2025
- February 6, 2025
- January 28, 2025
- January 15, 2025
- December 18, 2024
- November 12, 2024
- November 6, 2024
- November 1, 2024
- October 22, 2024
- September 18, 2024

- September 17, 2024
- August 22, 2024
- August 20, 2024
- August 8, 2024
- July 23, 2024
- July 22, 2024
- July 9, 2024
- July 8, 2024
- June 25, 2024
- May 29, 2024
- April 3, 2024
- February 22, 2024
- January 30, 2024
- January 10, 2024
- December 20, 2023
- December 13, 2023
- November 29, 2023
- November 21, 2023
- November 17, 2023
- November 6, 2023
- September 25, 2023
- September 20, 2023
- September 15, 2023
- September 11, 2023
- August 3, 2023
- July 13, 2023
- June 7, 2023
- May 10, 2023
- April 4, 2023

- March 22, 2023
- March 1, 2023
- February 27, 2023
- February 2, 2023
- November 30, 2022
- August 9, 2022
- July 25, 2022
- June 27, 2022
- April 29, 2022
- April 7, 2022
- March 16, 2022
- February 8, 2022
- January 24, 2022
- January 21, 2022
- October 25, 2021
- June 24, 2021
- May 4, 2021
- January 15, 2021
- November 9, 2020
- October 30, 2020
- September 22, 2020
- July 10, 2020
- June 30, 2020

June 18, 2025

New feature

Amazon DocumentDB is now available in the Israel (Tel Aviv) region.

May 8, 2025

New feature

Amazon DocumentDB is now available in the Europe (Stockholm) region. For more information, see this blog post.

April 2, 2025

> **Note**
>
> The following Amazon DocumentDB engine patch is in the process of being delivered to all Amazon DocumentDB regions in the next few weeks. When this engine patch becomes available in your region, you will receive a service patch notification via the AWS Health Dashboard (AHD) in the AWS Management Console and through e-mail to your AWS account's root user email address.
> This engine patch includes the following new features and bug fixes. Please note that the below list, along with the relevant supporting documentation, may be updated to include additional feature announcements once the engine patch is available in all regions.

New feature

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.14228)

Added support for unused storage size and percentage within the `collStats` diagnostic command.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.11357)

Added support for unused storage size and percentage within the `collStats` diagnostic command.

Bug fixes and other changes

Fixed a bug in vector index creation with background indexes.

March 24, 2025

New feature

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.11800)

Added support for `postBatchResumeToken` in change streams.

For more information, see Resuming a change stream with `postBatchResumeToken`.

February 6, 2025

New feature

Amazon DocumentDB is now integrated with AWS Toolkit for Visual Studio Code. For more information, see this What's New blog post, and see Working with Amazon DocumentDB in the Toolkit in the *AWS Toolkit for Visual Studio Code User Guide*.

January 28, 2025

New feature

For one-click connectivity, Amazon DocumentDB is now integrated with AWS CloudShell for instance-based and elastic clusters.

For more information see any or all of the following:

- Get started with Amazon DocumentDB
- Get started with Amazon DocumentDB elastic clusters
- What's new blog post
- Technical how-to blog post

January 15, 2025

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.11800)

Amazon DocumentDB now maintains read availability through writer instance restarts. Reader instances will now continue to serve read requests during writer instance restarts.

Bug fixes and other changes

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.11800)

- Fixed `killOp` command to handle a special case of bulk inserts.
- Improved network I/O-related memory usage on Amazon DocumentDB instances.
- Fixed `count` command for `$text` filter queries.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.11096)

- Fixed `killOp` command to handle a special case of bulk inserts.
- Improved network I/O-related memory usage on Amazon DocumentDB instances.

December 18, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.5721)

Amazon DocumentDB instance-based clusters now support NVMe-backed R6GD instances.

For more information, see NVMe-backed instances and this what's new blog post.

November 12, 2024

New features

Amazon DocumentDB elastic clusters now support Background Indexes.

November 6, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.11051)

- Added support for minimum TLS versions `tls1.2+` and `tls1.3+` to `tls` cluster parameter.
- Enabled support for characters dollar($) and dot(.) in field names. For functional differences, see Dollar($) and dot(.) in field names.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.10980)

- Added support for minimum TLS versions `tls1.2+` and `tls1.3+` to `tls` cluster parameter.

November 1, 2024

New feature

Amazon DocumentDB now supports elastic cluster maintenance actions. For more information, see Maintaining Amazon DocumentDB elastic clusters.

October 22, 2024

New feature

Amazon DocumentDB now supports storage network throughput metrics `StorageNetworkReceiveThroughput`, `StorageNetworkTransmitThroughput`, and `StorageNetworkThroughput`. For more information, see Evaluating Amazon DocumentDB instance usage with CloudWatch metrics.

September 18, 2024

New feature

Amazon DocumentDB is now available in the Africa (Cape Town) region. For more information, see this blog post.

Amazon DocumentDB is now available in the Europe (Spain) region. For more information, see this blog post.

September 17, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.10696)

Amazon DocumentDB now supports rebuilding indexes with `reIndex` in the `runCommand`. For more information, see Index maintenance using `reIndex`.

> **Note**
>
> `reIndex` is only supported on Amazon DocumentDB 5.0 (Engine Patch Version 3.0.10696 and higher).

Bug fixes and other changes

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.10696) and **Amazon DocumentDB 4.0 (Engine Patch Version 2.0.10898)**

- `$setOnInsert` now supports the `_id` field during inserts resulting from update operations.
- Fixed issue that prevents reclaiming storage space after a major version upgrade to Amazon DocumentDB 5.0.

August 22, 2024

New feature

Amazon DocumentDB 5.0 (All engine patch versions) and **Amazon DocumentDB 4.0 (Engine Patch Version 2.0.5704)**

Amazon DocumentDB global clusters now support cluster switchovers and managed cluster failovers. For more information, see Performing a switchover for an Amazon DocumentDB global cluster and Performing a managed failover for an Amazon DocumentDB global cluster.

> **Note**
>
> Global cluster switchovers and failovers are supported on Amazon DocumentDB 4.0 and 5.0 only.

August 20, 2024

New feature

For Amazon DocumentDB 3.6 (minimum engine patch version 1.0.208662), TLS CA certificate updates no longer require a system reboot. For more information, see Updating your Amazon DocumentDB TLS certificates.

August 8, 2024

New feature

Amazon DocumentDB elastic clusters are now available in the Asia Pacific (Hong Kong), Canada (Central), and Europe (Paris) regions. For more information, see Region availability for elastic clusters.

July 23, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.8126) and **Amazon DocumentDB 4.0 (Engine Patch Version 2.0.10709)**

- Added support for longer index names (up to 255 characters). For more information, see Naming constraints
- Increased the maximum connection limit by two-fold:

Instance type	Original limit	New limit
t3.medium	500	1000

Instance type	Original limit	New limit
t4g.medium	500	1000
r5.large	1700	3400
r5.xlarge	3500	7000
r5.2xlarge	7100	14200
r5.4xlarge	14200	28400
r5.8xlarge	28400	60000
r5.12xlarge	30000	60000
r5.16xlarge	30000	60000
r5.24xlarge	30000	60000
r6g.large	1700	3400
r6g.xlarge	3500	7000
r6g.2xlarge	7100	14200
r6g.4xlarge	14200	28400
r6g.8xlarge	28400	60000
r6g.12xlarge	30000	60000
r6g.16xlarge	30000	60000

For more information, see Instance limits.

Bug fixes and other changes

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.8126)

Enhanced the logic to synchronize `CurrentTime` and `ResumeToken` for the change stream on readers.

July 22, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.6742)

- Added support DML Audit filtering. You can now setup filter conditions to filter out DML Audit logs based on their specific requirements instead of logging every DML query. For more information, see Filtering DML audit events.
- Added support in document compression for the following:
  - Setting a minimum compression threshold
  - Enabling compression for existing collections (applicable to new documents)
  - Allow default compression setting at the cluster level

  For more information, see Managing collection-level document compression.
- Added support for consuming change streams on reader instances. For more information, see Using change streams on secondary instances.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.10593)

- Added support DML Audit filtering. You can now setup filter conditions to filter out DML Audit logs based on their specific requirements instead of logging every DML query. For more information, see Filtering DML audit events.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.208662)

Removed index limitations on db.r5.* and db.r6.* instances in Amazon DocumentDB MVU. For more information, see MVU prerequisites and limitations.

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.208662)

Amazon DocumentDB now recognizes -NaN as a valid JSON token.

July 9, 2024

New feature

For Amazon DocumentDB 4.0 (minimum engine patch version 2.0.10179) and 5.0 (minimum engine patch version 3.0.4780), TLS CA certificate updates no longer require a system reboot. For more information, see Updating your Amazon DocumentDB TLS certificates.

July 8, 2024

New feature

Amazon DocumentDB elastic clusters are now available in the Europe (Milan) region. For more information, see Region availability for elastic clusters.

June 25, 2024

New feature

Authentication with AWS IAM ARNs is available in Amazon DocumentDB instance-based 5.0 clusters across all supported regions. For more information, see Authentication using IAM identity.

May 29, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.6742)

- Added support for `$regexMatch` and `$regexFind` operators.
- Added support to ensure full precision in audit logs when addressing large integers. Audit logs now maintain the exact numeric representation for all numbers, preventing any loss of precision.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.10593)

- Added support to ensure full precision in audit logs when addressing large integers. Audit logs now maintain the exact numeric representation for all numbers, preventing any loss of precision.

April 3, 2024

Amazon DocumentDB is now available in the Middle East (UAE) region. For more information, see this blog post.

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.5721)

- Added support for `bypassDocumentValidation` and granular error message for `$jsonSchema`. For more information about `bypassDocumentValidation`, see bypassDocumentValidation.
- Added support of `$expr`.
- Added support for Uncorrelated Joins in `$lookup`.
- Added support to retain validation rules in `$out` aggregation stage.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.10392)

- Added support for `bypassDocumentValidation` for `$jsonSchema`. For more information about `bypassDocumentValidation`, see bypassDocumentValidation.
- Added support of `$expr`.
- Added support for Uncorrelated Joins in `$lookup`.
- Added support to retain validation rules in `$out` aggregation stage.

Bug fixes and other changes

- Fixed an error when invoking `db.coll.stats()` on mongo shell version 1.7 and later.
- Fixed a memory leak issue for change stream queries that contain `$regex` as a part of the same aggregation pipeline.

February 22, 2024

New features

Amazon DocumentDB elastic clusters

Amazon DocumentDB elastic clusters now support the following features:

- Readable secondary shard instance replicas - for more information, see step 5b of Step 1: Create an elastic cluster.
- Start/stop cluster - for more information, see Stopping and starting an Amazon DocumentDB elastic cluster.
- Configurable shard instances - for more information, see step 5b of Step 1: Create an elastic cluster.
- Automatic backups for snapshots - for more information, see Managing an elastic cluster snapshot automatic backup.
- Copy snapshot - for more information, see Copying an elastic cluster snapshot.

January 30, 2024

New features

Amazon DocumentDB elastic clusters

Amazon DocumentDB elastic clusters are now available in the following regions:

- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- South America (São Paulo)
- Europe (London)

For more information, see Elastic cluster region and version availability.

Amazon DocumentDB global clusters

Global clusters are now available in both AWS GovCloud (US) regions: AWS GovCloud (US-East) and AWS GovCloud (US-West).

January 10, 2024

New features

Amazon DocumentDB 5.0 (Engine Patch Versions 3.0.4574, 3.0.4780, 3.0.4960)

- Added support for HNSW vector indexes. For more information, see Vector search for Amazon DocumentDB.
- Added a support for partial indexes. For more information, see Partial index.
- Added a support for GC runtime on a collection within `currentOp` command.
- Added text index support for native text search on Amazon DocumentDB. For more information, see Performing text search with Amazon DocumentDB.
- Added support for `$jsonSchema` schema keywords `type`, `allOf`, `oneOf`, `anyOf`, `not`, `maxItems`, `minItems`, `maxProperties`, `minProperties`, `pattern`, `patternProperties`, `multipleOf`, `dependencies`, and `uniqueItems`.

  For more information see Using JSON schema validation.
- Added support for arithmetic operators `$ceil`, `$floor`, `$ln`, `$log`, `$log10`, `$sqrt`, and `$exp`.

  For more information see Arithmetic operators.
- Added support for the conditional expression operator `$switch`.
- Added support for parallel `IVFFLAT` vector index builds. Documentation was updated by removing the parallel `IVFFLAT` vector index builds limitation from the developer guide.

Amazon DocumentDB 4.0 (Engine Patch Versions 2.0.10124, 2.0.10179, 2.0.10221)

- Added a support for GC runtime on a collection within `currentOp` command.
- Added support for `$jsonSchema` schema keywords `type`, `allOf`, `oneOf`, `anyOf`, `not`, `maxItems`, `minItems`, `maxProperties`, `minProperties`, `pattern`, `patternProperties`, `multipleOf`, `dependencies`, and `uniqueItems`.

  For more information see Using JSON schema validation.
- Added support for arithmetic operators `$ceil`, `$floor`, `$ln`, `$log`, `$log10`, `$sqrt`, and `$exp`.

  For more information see Arithmetic operators.
- Added support for the conditional expression operator `$switch`.

Bug fixes and other changes

- Added back case-insensitive functionality for invoking `db.runCommand("dbstats")`. Amazon DocumentDB 5.0 and 4.0 customers on engine patch versions prior to 3.0.4960 or 2.0.10221 should apply these latest engine patches.
- Fixed an error when invoking `db.coll.stats()` on mongo shell version 1.7 and later. Documentation was updated by removing the mongo shell `db.coll.stats()` troubleshooting tip from the developer guide.

December 20, 2023

Other changes

Enabled support for in-place major version upgrade in Amazon DocumentDB 3.6 and 4.0. For more information, see Amazon DocumentDB in-place major version upgrade.

December 13, 2023

New features

Added support for 1-click EC2 connectivity. For more information, see Connect using Amazon EC2.

November 29, 2023

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.3727)

New features

Added support for vector search. For more information, see this blog post and visit Vector search for Amazon DocumentDB in the *Amazon DocumentDB Developer Guide*.

November 21, 2023

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.3727)

New features

Added support for I/O-optimized storage. For more information, see Amazon DocumentDB cluster storage configurations in the *Amazon DocumentDB Developer Guide*.

Added integration for no-code machine learning with SageMaker Canvas. For more information, see No-code machine learning with Amazon SageMaker AI Canvas in the *Amazon DocumentDB Developer Guide*.

November 17, 2023

New features

Amazon DocumentDB is now available in the AWS GovCloud (US-East) region. For more information, see this blog post.

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.208570)

User defined local variable names now support "_" (underscore) for projection operators such as `$let` and `$filter`.

November 6, 2023

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.3727) and 4.0 (Engine Patch Version 2.0.9876)

New features

- Added support for `$jsonSchema` schema keywords `maxLength`, `minLength`, `maximum`, `minimum`, `exclusiveMaximum`, `exclusiveMinimum`, `items`, and `additionalItems`.

  Please note that JSON schema validation is supported in instance-based clusters only.
- Added support for `$convert` aggregation pipeline operator and its shorthand derived operators `$toBool`, `$toInt`, `$toLong`, `$toDouble`, `$toString`, `$toDecimal`, `$toObjectId`, and `$toDate`.

- Added support for set expression operators `$setDifference`, `$anyElementTrue`, and `$allElementTrue`.

Bug fixes and other changes

Fixed issue where a change stream update from `-NaN` to `NaN` was not being displayed.

September 25, 2023

New features

Amazon DocumentDB is now available in the Asia Pacific (Hong Kong) region. For more information, see this blog post.

September 20, 2023

New features

Added support for in-place major version upgrades in Amazon DocumentDB 3.6 and 4.0. For more information see Amazon DocumentDB in-place major version upgrade.

September 15, 2023

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.3140) and 4.0 (Engine Patch Version 2.0.9686)

Added support for $jsonSchema schema validator in instance-based clusters only. For more information see Using JSON schema validation.

September 11, 2023

New features

Amazon DocumentDB is now available in the Asia Pacific (Hyderabad) region. For more information, see this blog post.

August 3, 2023

New features

Amazon DocumentDB Elastic clusters

- Amazon DocumentDB Elastic clusters now support the following operations:
  - `top`
  - `collStats`
  - `hint`
  - `dataSize`

  See Supported MongoDB APIs, operations, and data types in Amazon DocumentDB for the complete list of supported commands and operations.
- Time to Live (TTL) indexes are now supported.
- Index `hints` are now supported with index expressions.

July 13, 2023

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.1948)

- Added support for document compression.
- Added support for parallel index builds.
- Added support for index build status.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.9259)

- Added support for parallel index builds.

Bug fixes and other changes

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.1948)

- Fixed authentication issue with `createCollection` for Amazon DocumentDB elastic clusters when users don't have access to system collections.
- Fixed issue where secondary region instances couldn't use the same primary region instance names.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.9259)

- Stopped adding internal monitoring queries to auditing logs.

June 7, 2023

Bug fixes and other changes

Amazon DocumentDB 5.0

- r5 and t3.medium instances are now supported in Amazon DocumentDB 5.0.
- `engineVersion` option default is `5.0.0` in AWS SDK, AWS CLI, and AWS CloudFormation.

May 10, 2023

Bug fixes and other changes

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.1361)

- Added support for `ignoreunknownindexoptions` in the **createIndex** command.
- Stopped adding internal monitoring queries to auditing logs.
- User defined local variable names now support "_" (underscore) for projection operators such as `$let` and `$filter`.

April 4, 2023

Bug fixes and other changes

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.8934)

- Fixed issue with DML auditing when it is enabled during an ongoing workload.

- Fixed issue with DML auditing when aggregate commands with hint is passed a string value.
- Fixed issue with `listCollections` command not working when users with readwriteanydatabase role having both authorizedCollections and nameOnly options set to true.
- Fixed issue to properly parse numeric string in a field name.
- Cancel long running cursors when they are impacting garbage collection.
- User defined local variable names now support "_" (underscore) for projection operators such as `$let` and `$filter`.

March 22, 2023

New features

Amazon DocumentDB elastic clusters are now available in the Asia Pacific (Singapore), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions. For more information, see Elastic cluster region and version availability.

March 1, 2023

New features

Amazon DocumentDB 5.0 (Engine Patch Version 3.0.775)

- Introduced Amazon DocumentDB 5.0
  - MongoDB 5.0 compatibility (support for MongoDB 5.0 API drivers)
  - Support for Client-side Field Level Encryption (FLE). You can now encrypt fields at the client-side before writing the data to Amazon DocumentDB cluster. For more information, see Client-side field level encryption
  - New aggregation operators: `$dateAdd`, `$dateSubtract`
- Increased storage limit to 128 TiB for all instance-based Amazon DocumentDB clusters and shard-based elastic clusters.
- Amazon DocumentDB 5.0 now supports index scan with the `$elemMatch` operator in the first nesting level. Index scans are supported when query only have one level of the `$elemMatch` filter and the nested `$elemMatch` query does not support index scan.

  Query shape that supports index scan:

```
db.foo.find( { "a": {$elemMatch: { "b": "xyz", "c": "abc"} } })
```

Query shape that does not support index scan:

```
db.foo.find( { "a": {$elemMatch: { "b": {$elemMatch: { "d": "xyz", "e":
 "abc"} }} } })
```

February 27, 2023

Bug fixes and other changes

Amazon DocumentDB 4.0

Added support for AWS Lambda. For more information, see Using AWS Lambda with Change Streams.

February 2, 2023

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.208432)

- Fixed issue with DML auditing when it is enabled during an ongoing workload.
- Fixed issue with DML auditing when aggregate commands with hint is passed a string value.
- Fixed issue with `listCollections` command not working when users with readwriteanydatabase role having both authorizedCollections and nameOnly options set to true.
- Fixed issue to properly parse numeric string in a field name.
- Cancel long running cursors when they are impacting garbage collection.

November 30, 2022

New features

Amazon DocumentDB Elastic clusters

Amazon DocumentDB elastic clusters is a new type of Amazon DocumentDB cluster that enables users to leverage the MongoDB sharding APIs to scale out their cluster. Elastic clusters handle virtually any number of reads and writes with petabytes of storage capacity by distributing the data and compute across multiple underlying compute instances and volumes. To learn more, see Using Amazon DocumentDB elastic clusters.

August 9, 2022

New features

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.208152) and 4.0

- Added support for Decimal128 data type. The Decimal128 is a BSON data type supported in all regions where DocumentDB is available.

  For more information, see Data Types.
- Added support for DML query auditing with Amazon CloudWatch Logs. Now **Amazon DocumentDB** can record Data Manipulation Language (DML) events and Data Definition Language (DDL) events to Amazon CloudWatch Logs.

  For more information, see this blog post.

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.208152) and 4.0

- You can now change your own passwoprd with own password with `changeOwnPassword` privilege.

July 25, 2022

New features

Amazon DocumentDB 4.0

You can now create clusters faster with the ability to create clones that use the same DocumentDB cluster volume and have the same data as the original cluster. For details, see Managing Amazon DocumentDB Clusters.

June 27, 2022

New features

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.7509)

Amazon DocumentDB dynamically resizes your database based on usage patterns. Adding more data increases the space up to 64 Tebibytes (TiB) and deleting data decreases allotted space.

April 29, 2022

New features

Amazon DocumentDB is now available in China (Beijing) region. For more information, see this blog post.

April 7, 2022

New features

Amazon DocumentDB 3.6 (Engine Patch Versions 1.0.207836 and 1.0.208015) and 4.0 (Engine Patch Versions 2.0.6142 and 2.0.6948)

Amazon DocumentDB Performance Insights is now in preview. You can now store seven days of performance history in a rolling window at no additional cost. For more information, see Monitoring with Performance Insights.

March 16, 2022

New features

Amazon DocumentDB is now available in Europe (Milan) region. For more information, see this blog post.

February 8, 2022

New features

Amazon DocumentDB R6g and T4g instances are now available in Asia Pacific, South America, and Europe. For more information, see this blog post.

January 24, 2022

New features

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207684) and 4.0 (Engine Patch Version 2.0.5170)

- Amazon DocumentDB now offers a free trial. For details, see Amazon DocumentDB free trial page.
- You can now use enhanced features with Geospatial query, including the following APIs:
  - `$geoWithin`
  - `$geoIntersects`
- Added support for the following MongoDB operators:
  - `$mergeObjects`
  - `$reduce`

  For more information, see the Querying Geospatial data with Amazon DocumentDB.

January 21, 2022

New features

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.5706)

- Amazon DocumentDB Graviton2 (r6g.large, r6g.2xlarge, r6g.4xlarge, r6g.8xlarge, r6g.12xlarge, r6g.16xlarge and t4g.medium) instances are now supported.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207781) and 4.0 (Engine Patch Version 2.0.5706)

- Added support for the following MongoDB APIs:
  - `$reduce`
  - `$mergeObjects`
  - `$geoWithin`
  - `$geoIntersects`

October 25, 2021

New features

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207780) and 4.0 (Engine Patch Version 2.0.5704)

- Added support for the following MongoDB APIs
  - `$literal`
  - `$map`
  - `$$ROOT`
- Support for GeoSpatial Query capabilities. See this blog post for more details
- Support for access control with user-defined roles. See this blog post for more details
- Amazon DocumentDB JDBC Driver to enable connectivity from BI tools such as Tableau and query tools such as SQL Workbench

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207780) and 4.0 (Engine Patch Version 2.0.5704)

- Bug fix for `$natural` to sort correctly when an explicit `.sort()` is present along with `$natural`
- Bug fix for change stream to work with `$redact`
- Bug fix for `$ifNull` to work with empty array
- Bug fix for excessive resource consumption/server crash when a currently logged-in user is deleted or that user's privilege for an ongoing activity is revoked

- Bug fix in `listDatabase` and `listCollection` privilege check
- Bug Fix dedupe logic for multi-key elements

June 24, 2021

New features

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207117) and 4.0 (Engine Patch Version 2.0.3371)

- r5.8xlarge and r5.16xlarge instances are now supported. Learn more at the blog post Amazon DocumentDB Now Supports r5.8xlarge and r5.16xlarge Instances.
- Global clusters are now supported to provide disaster recovery from region-wide outages and enable low-latency global reads by allowing reads from the nearest Amazon DocumentDB cluster. Note that global clusters are not currently supported in the South America (São Paulo), Europe (Milan), China (Beijing), and China (Ningxia) Regions.

May 4, 2021

New features

See all the new features in this blog post.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207117) and 4.0 (Engine Patch Version 2.0.3371)

- `renameCollection`
- `$zip`
- `$indexOfArray`
- `$reverseArray`
- `$natural`
- `$hint` support for update
- Index scan for `distinct`

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.207117) and 4.0 (Engine Patch Version 2.0.3371)

- Reduced memory usage for `$in` queries
- Fixed a memory leak in multikey indexes
- Fixed the explain plan and profiler output for `$out`
- Added a timeout for operations from internal monitoring system to improve reliability
- Fixed a defect impacting the query predicates passed to multikey indexes

January 15, 2021

New features

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.722)

- None

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- Ability to use an index with the `$lookup` aggregation stage
- `find()` queries with projections can be served direction from an index (covered query)
- Ability to use `hint()` with the `findAndModify`
- Performance optimizations for `$addToSet` operator
- Improvements to reduce overall index sizes
- New aggregation operators: `$ifNull`, `$replaceRoot`, `$setIsSubset`, `$setInstersection`, `$setUnion`, and `$setEquals`
- Users can also end their own cursors without requiring the `KillCursor` role

November 9, 2020

New features

See all the new features in this blog post.

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.722)

- MongoDB 4.0 compatibility
- ACID transactions
- Support for `cluster(client.watch()` or `mongo.watch())` and the database level `(db.watch())` change streams
- Ability to start or resume a change streams using `startAtOperationTime`
- Extend your change stream retention period to 7 days (previously 24 hours)
- AWS DMS target for Amazon DocumentDB 4.0
- CloudWatch metrics: `TransactionsOpen`, `TransactionsOpenMax`, `TransactionsAborted`, `TransactionsStarted`, and `TransactionsCommitted`
- New fields for transactions in `currentOp`, `ServerStatus`, and `profiler`.
- Ability to use an index with the `$lookup` aggregation stage
- `find()` queries with projections can be served direction from an index (covered query)
- Ability to use `hint()` with the `findAndModify`
- Performance optimizations for `$addToSet` operator
- Improvements to reduce overall index sizes.
- New aggregation operators: `$ifNull`, `$replaceRoot`, `$setIsSubset`, `$setInstersection`, `$setUnion`, and `$setEquals`
- With the `ListCollection` and `ListDatabase` commands, you can now optionally use the `authorizedCollections` and `authorizedDatabases` parameters to allow users to list the collections and databases that they have permission to access without requiring the `listCollections` and `listDatabase` roles, respectively
- Users can also end their own cursors without requiring the `KillCursor` role
- Comparing numeric types of subdocuments is now consistent with comparing numeric types of first-level documents. The behavior in Amazon DocumentDB 4.0 is now compatible with MongoDB.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- None

Bug fixes and other changes

Amazon DocumentDB 4.0 (Engine Patch Version 2.0.722)

- `$setOnInsert` no longer allow updates when using the positional operator `$`. The behavior in Amazon DocumentDB 4.0 is now compatible with MongoDB.
- Fixed issue with `$createCollection` and set `autoIndexId`
- Projection for nested documents
- Changed default setting for working memory to scale with instance memory size
- Garbage collection improvements
- Lookup with empty key in path, behavior difference with mongo
- Fixed `dateToString` bug in timezone behavior
- Fixed `$push` (aggregation) to respect sort order
- Fixed bug in `$currentOp` with aggregate
- Fixed issue with `readPreference` on secondary
- Fixed issue with validating `$createIndex` is the same database as the command was issued
- Fixed inconsistent behavior for `minKey`, `maxKey` lookup fails
- Fixed issue with `$size` operator not working with composite array
- Fixed issue with the negation of `$in` with regex
- Fixed issue with `$distinct` command run against a view
- Fixed issue with aggregations and find commands sorting missing fields differently
- Fixed `$eq` to regular expression not checking type
- Fixed `$currentDate` bug in timestamp ordinal position behavior
- Fixed millisecond granularity for `$currentDate`

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- None

October 30, 2020

New features

See all the new features in this blog post.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- Added the ability to open a change stream cursor at the cluster level (`client.watch()` or `mongo.watch()`) and the database (`db.watch()`)
- Ability to increase the change stream retention period to 7 days(previously 24 hours)

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- Various general case performance improvements
- A targeted security improvement
- Fixed an issue with skip sort on second field of a compound index
- Enable regular index for equality on single field of a multi-key index (not compound)
- Fixed authentication race condition
- Fixed issue that caused an infrequent garbage collection crash
- RBAC security improvement
- Added `databaseConnectionsMax` metric
- Performance improvements for certain workloads on `r5.24xlarge` instances

September 22, 2020

New features

See all the new features in this blog post.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- `$out` aggregation stage

- Increased the maximum number of connections and cursor per instance by as much as 10x

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- None

July 10, 2020

New features

See all the new features in this blog post.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- Cross Region Snapshot Copy

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- None

June 30, 2020

New features

See all the new features in this blog post.

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- T3 medium instances

Bug fixes and other changes

Amazon DocumentDB 3.6 (Engine Patch Version 1.0.206295)

- Idle memory reclamation for t3 instances
- Authentication improvements
- Improved SASL authentication performance
- Fixed `currentOp` issue when exceeding maximum possible ops
- Fixed `killOps` issue for bulk update and delete
- Improvements to `$sample` performance with `$match`
- Fixed support for `$$` in cond case in redact stage
- Fixed various recurring crash root causes
- Improvements to TTL sweeping to reduce IOs and latency
- Optimized memory utilization for `$unwind`
- Fixed collection stats race condition with drop index
- Fixed race condition during concurrent index build
- Fixed infrequent crash in `hash_search` in index

Document history for the Amazon DocumentDB Developer Guide

- **API version:** 2014-10-31
- **Latest documentation update:** July 23, 2024

The following table describes the documentation for this release of the *Amazon DocumentDB Developer Guide*.

Change	Description	Date
AWS managed policy update - policy change	Amazon DocumentDB updates the full-access policies for elastic clusters.	February 11, 2025
AWS managed policy update - policy change	Amazon DocumentDB updates the full-access policies for elastic clusters.	February 21, 2024
AWS managed policy update - policy change	Amazon DocumentDB updates the read-only and full-access policies for elastic clusters.	June 21, 2023
AWS managed policy update - new policy	Amazon DocumentDB introduces a new read-only policy for elastic clusters.	June 8, 2023
AWS managed policy update - new policy	Amazon DocumentDB introduces a new full-access policy for elastic clusters.	June 5, 2023
MongoDB 5.0 Compatibility	Amazon DocumentDB is now compatible with version 5.0 of MongoDB.	March 1, 2023

Policy Update	To support the Amazon DocumentDB elastic cluster feature, AmazonDoc DBConsoleFullAccess policy is updated, and the AmazonDoc DB-ElasticServiceRolePolicy is introduced.	November 30, 2022
Elastic Clusters	Added new Elastic Cluster feature supporting hash-based partitioning (sharding) of data across Amazon DocumentDB's distributed storage system.	November 30, 2022
Global Clusters	Added documentation on how to use Global Clusters.	June 2, 2021
Event Subscriptions	Added event subscription documentation.	March 26, 2021
Version 3.6 Upgrades	Documented improvements to version 3.6 in role-based access controls, aggregation operators, and performance.	January 15, 2021
MongoDB 4.0 Compatibility	Amazon DocumentDB is now compatible with version 4.0 of MongoDB.	November 9, 2020
Get Started Guides	New Get Started Guides for getting started with Amazon DocumentDB using AWS Cloud9, Amazon EC2, Robo3T or Studio3T.	August 15, 2020

Additional Availability Zones supported	Amazon DocumentDB added support for an additiona l Availability Zone in Asia Pacific (Seoul) (ap-north east-2).	July 14, 2020
Added support for copying snapshots across Regions.	Amazon DocumentDB added support for copying cluster snapshots across AWS Regions. For more informati on, see Copying Snapshots Across Regions.	July 10, 2020
Added support for T3 instance class.	Added support for T3 instance types in all Regions supporting Amazon DocumentDB. For more information, see Supported Instance Classes by Region and Instance Class Specifica tions.	June 30, 2020
Added support for AWS GovCloud (US).	Amazon DocumentDB is now available in the AWS GovCloud (US) Region (us-gov-west-1).	June 29, 2020
Added 16 new CloudWatch metrics.	Amazon DocumentDB added support for 16 new Amazon CloudWatch metrics. For more information, see Monitorin g Amazon DocumentDB with CloudWatch.	June 23, 2020

Added support for null characters and $regex operator.	Amazon DocumentDB added support for null characters in strings and the ability to use an index for $regex. To view the supported MongoDB APIs and aggregation pipeline capabilities for Amazon DocumentDB, see Functional Differences with MongoDB.	June 22, 2020
Added support for improved multi-key indexing capabilities.	Amazon DocumentDB added support for improved multi-key indexing capabilities that include indexing of arrays larger than 2,048 bytes and the ability to create a compound multi-key index with multiple keys in the same array. For more information, see Functional Differences with MongoDB.	April 23, 2020
Added support for deletion protection for an Amazon DocumentDB AWS CloudFormation stack.	Amazon DocumentDB added support for enabling deletion protection when creating an Amazon DocumentDB AWS CloudFormation stack.	April 20, 2020
Added support for role-based access control.	Amazon DocumentDB added support for role-based access control using built-in roles.	March 26, 2020
Added support for an additional Availability Zone in Canada (Central) (ca-central-1).	Amazon DocumentDB is now available in the Canada (Central) Region (ca-central-1) with R5 class instances and 3 Availability Zones.	March 26, 2020

Added support for two additional MongoDB APIs.	Amazon DocumentDB added support for `$dateFromString` and `executionStats` MongoDB APIs.	March 23, 2020
Added support for five additional MongoDB APIs.	Amazon DocumentDB added support for `$objectToArray`, `$arrayToObject`, `$slice`, `$mod`, and `$range` MongoDB APIs.	February 6, 2020
Added support for Canada (Central).	Amazon DocumentDB is now available in the Canada (Central) Region (ca-central-1) with R5 class instances.	December 11, 2019
Added support for ChangeStreamLogSize.	Amazon DocumentDB added support for `ChangeStreamLogSize` for Cloudwatch metrics.	November 22, 2019
Added support for Europe (Paris) region	Amazon DocumentDB is now available in the Europe (Paris) region (eu-west-3) with R5 class instances.	October 30, 2019
Added support for Asia Pacific (Mumbai) region	Amazon DocumentDB is now available in the Asia Pacific (Mumbai) region (ap-south-1) with R5 class instances.	October 17, 2019
Added support for three additional MongoDB APIs	Amazon DocumentDB added support for the `$addFields`, `$concatArrays`, and `$lookup` MongoDB APIs.	October 16, 2019

Added support for Asia Pacific (Singapore) region	Amazon DocumentDB is now available in the Asia Pacific (Singapore) region (ap-south east-1) with R5 class instances .	October 14, 2019
Added new document for updating TLS certificates	Added instructions for updating CA certificates to use the new CA certificate to create TLS connections.	October 2, 2019
Added API support for certificates	Amazon DocumentDB a new Certificate data type for instances. For more informati on, see DBInstance.	October 1, 2019
Support for query profiling	Amazon DocumentDB added the ability to profile supported operations on your cluster's instances and databases.	August 19, 2019
Added third AZ in Asia Pacific (Tokyo)	Amazon DocumentDB added a third Availability Zone (AZ) for your compute instances in Asia Pacific (Tokyo).	August 9, 2019

Support for additional Mongo APIs	Added support for additional aggregation pipeline capabilities that include the `$in`, `$isoWeek`, `$isoWeekYear` , `$isoDayOfWeek` , and `$dateToString` aggregation operators and the `$addToSet` aggregation stage. Amazon DocumentDB also added support for the `top()` command for collection level diagnostics and the ability to modify the `expireAfterSeconds` parameter for TTL indexes using the `collMod()` command.	July 31, 2019
Added support for Europe (London)	Amazon DocumentDB is now available in Europe (London) (eu-west-2) with R5 class instances.	July 18, 2019
Added code samples	Added code examples in R and Ruby for programmatically connecting to Amazon DocumentDB.	July 17, 2019
Added best practice	Added a Best Practice to help you manage your Amazon DocumentDB costs.	July 17, 2019
Support for stopping and starting a cluster	Amazon DocumentDB added support for stopping and starting clusters to manage costs for development and test environments.	July 1, 2019

Support for cluster deletion protection	To protect your clusters from accidental deletion, Amazon DocumentDB added deletion protection. For more information, see the following topics: Creating an Amazon DocumentDB Cluster, Modifying an Amazon DocumentDB Cluster, Deleting an Amazon DocumentDB Cluster, and `DeletionProtection` in the API topic DBCluster.	July 1, 2019
Functional differences update	Added Implicit Transactions to Functional Differences.	June 26, 2019
Functional differences addition	Added note regarding storage and index compression in Amazon DocumentDB.	June 13, 2019
Additional region supported	Amazon DocumentDB is now available in Asia Pacific (Sydney) (ap-southeast-2) with R5 class instances.	June 5, 2019
R5 instance class supported in additional regions	Added R5 instance class support for 4 additional regions: US East (Ohio), US East (N. Virginia), US West (Oregon), and EU (Ireland). With this change, R5 instances are supported in all regions supporting Amazon DocumentDB.	May 17, 2019

Additional regions supported	Added support for 2 additional regions, Asia Pacific (Tokyo) (ap-northeast-1) and Asia Pacific (Seoul) (ap-northeast-2) with R5 instance classes. For more information, see Supported Instance Classes by Region and Instance Class Specifications.	May 8, 2019
Added more connection code examples	Added code examples in Java and C# for connecting to Amazon DocumentDB.	April 24, 2019
Additional Mongo API support	Added support for seven aggregation string operators (`$indexOfBytes`, `$indexOfCP`, `$strLenBytes`, `$strLenCP`, `$toLower`, `$toUpper`, and `$split`), nine date-time operators (`$dayOfYear`, `$dayOfMonth`, `$dayOfWeek`, `$year`, `$month`, `$hour`, `$minute`, `$second`, and `$millisecond`), and the `$sample` aggregation pipeline stage.	April 4, 2019
Added connection code examples	Added code examples in Python, Node.js, PHP, and Go for connecting to Amazon DocumentDB.	March 21, 2019

Support for Frankfurt Region and R5 instances	Added support for Europe (Frankfurt) Region (eu-central-1) with R5 instance classes. For more information, see Supported Instance Classes by Region and Instance Class Specifications.	March 13, 2019
Aggregation pipeline operators support	Added support for new aggregation string operators (`$concat`, `$substr`, `$substrBytes` , `$substrCP` , `$strcasecmp`), an array aggregation operator (`$size`), an aggregation group accumulator operator (`$push`), and aggregation stages (`$redact` and `$indexStats`). We also added support for positional array operators (`$[]` and `$[<identifier>]`) and `hint()`.	February 28, 2019
Engine upgrades	Added documentation for determining pending cluster modifications and upgrading your cluster's engine version.	February 15, 2019
Auditing events	Added support for auditing database events with Amazon CloudWatch Logs.	February 12, 2019
Quick Start	Added a Quick Start topic to help you easily start with Amazon DocumentDB using AWS CloudFormation.	January 11, 2019

Public Release	This is the initial public release of Amazon DocumentDB (with MongoDB compatibility). This release includes the Developer Guide and the integrated Resource Management API Reference.	January 9, 2019