aws

User Guide

Application Auto Scaling

Application Auto Scaling: User Guide

Table of Contents

What is Application Auto Scaling? 1
Features of Application Auto Scaling 2
Work with Application Auto Scaling 2
Concepts 3
Learn more 5
Services that integrate 6
Amazon WorkSpaces Applications 8
Service-linked role 8
Service principal 9
Registering WorkSpaces Applications fleets as scalable targets with Application Auto Scaling 9
Related resources 10
Amazon Aurora 10
Service-linked role 10
Service principal 10
Registering Aurora DB clusters as scalable targets with Application Auto Scaling 11
Related resources 12
Amazon Comprehend 12
Service-linked role 12
Service principal 12
Registering Amazon Comprehend resources as scalable targets with Application Auto Scaling 12
Related resources 14
Amazon DynamoDB 14
Service-linked role 14
Service principal 14
Registering DynamoDB resources as scalable targets with Application Auto Scaling 15
Related resources 17
Amazon ECS 17
Service-linked role 18
Service principal 18
Registering ECS services as scalable targets with Application Auto Scaling 18
Related resources 19
Amazon ElastiCache 20

Service-linked role ... 20
Service principal ... 20
Registering ElastiCache resources as scalable targets with Application Auto Scaling ... 20
Related resources ... 22
Amazon Keyspaces (for Apache Cassandra) ... 22
Service-linked role ... 22
Service principal ... 23
Registering Amazon Keyspaces tables as scalable targets with Application Auto Scaling ... 23
Related resources ... 24
AWS Lambda ... 24
Service-linked role ... 25
Service principal ... 25
Registering Lambda functions as scalable targets with Application Auto Scaling ... 25
Related resources ... 26
Amazon Managed Streaming for Apache Kafka (MSK) ... 26
Service-linked role ... 26
Service principal ... 27
Registering Amazon MSK cluster storage as scalable targets with Application Auto Scaling ... 27
Related resources ... 28
Amazon Neptune ... 28
Service-linked role ... 28
Service principal ... 29
Registering Neptune clusters as scalable targets with Application Auto Scaling ... 29
Related resources ... 30
Amazon SageMaker AI ... 30
Service-linked role ... 30
Service principal ... 30
Registering SageMaker AI endpoint variants as scalable targets with Application Auto Scaling ... 31
Registering the provisioned concurrency of serverless endpoints as scalable targets with Application Auto Scaling ... 32
Registering inference components as scalable targets with Application Auto Scaling ... 33
Related resources ... 33
Spot Fleet (Amazon EC2) ... 34
Service-linked role ... 34

Service principal ... 34
Registering Spot Fleets as scalable targets with Application Auto Scaling ... 35
Related resources ... 36
Amazon WorkSpaces ... 36
Service-linked role ... 36
Service principal ... 36
Registering WorkSpaces pools as scalable targets with Application Auto Scaling ... 36
Related resources ... 37
Custom resources ... 37
Service-linked role ... 38
Service principal ... 38
Registering custom resources as scalable targets with Application Auto Scaling ... 38
Related resources ... 39
Configure scaling using CloudFormation ... 40
Application Auto Scaling and CloudFormation templates ... 40
Example template snippets ... 41
Learn more about CloudFormation ... 41
Scheduled scaling ... 42
How scheduled scaling works ... 43
How it works ... 43
Considerations ... 43
Commonly used commands ... 44
Related resources ... 45
Limitations ... 45
Create scheduled actions ... 45
Create a scheduled action that occurs only once ... 46
Create a scheduled action that runs on a recurring interval ... 47
Create a scheduled action that runs on a recurring schedule ... 48
Create a one-time scheduled action that specifies a time zone ... 49
Create a recurring scheduled action that specifies a time zone ... 50
Describe scheduled scaling ... 50
Describe scaling activities for a service ... 51
Describe the scheduled actions for a service ... 53
Describe the scheduled actions for a scalable target ... 55
Schedule recurring scaling actions ... 56
Turn off scheduled scaling ... 59

Delete a scheduled action ... 60
Target tracking scaling policies ... 62
How target tracking works ... 63
How it works ... 63
Choose metrics ... 64
Define target value ... 65
Define cooldown periods ... 66
Considerations ... 67
Multiple scaling policies ... 68
Commonly used commands ... 69
Related resources ... 69
Limitations ... 69
Create a target tracking scaling policy ... 70
Step 1: Register a scalable target ... 70
Step 2: Create a target tracking scaling policy ... 71
Step 3: Describe target tracking scaling policies ... 73
Delete a target tracking scaling policy ... 75
Use metric math ... 76
Example: Amazon SQS queue backlog per task ... 76
Limitations ... 81
Step scaling policies ... 82
How step scaling works ... 83
How it works ... 83
Step adjustments ... 84
Scaling adjustment types ... 86
Cooldown period ... 87
Commonly used commands ... 88
Considerations ... 88
Related resources ... 45
Console access ... 89
Create a step scaling policy ... 89
Step 1: Register a scalable target ... 89
Step 2: Create a step scaling policy ... 90
Step 3: Create an alarm that invokes a scaling policy ... 94
Describe step scaling policies ... 95
Delete a step scaling policy ... 97

Predictive scaling 98
How it works 98
Maximum capacity limit 99
Commonly used commands for scaling policy creation, management, and deletion 100
Considerations 100
Create a predictive scaling policy 101
Override the forecast 102
Step 1: (Optional) Analyze time series data 103
Step 2: Create two scheduled actions 103
Use custom metrics 105
Best practices 105
Prerequisites 106
Constructing the JSON for custom metrics 106
Considerations for custom metrics 115
Tutorial: Configure auto scaling to handle a heavy workload 116
Prerequisites 116
Step 1: Register your scalable target 117
Step 2: Set up scheduled actions according to your requirements 118
Step 3: Add a target tracking scaling policy 121
Step 4: Next steps 124
Step 5: Clean up 124
Suspend scaling 127
Scaling activities 127
Suspend and resume scaling activities 128
View suspended scaling activities 131
Resume scaling activities 131
Scaling activities 133
Look up scaling activities by scalable target 133
Include not scaled activities 134
Reason codes 136
Monitoring 139
Monitor using CloudWatch 140
CloudWatch metrics for monitoring resource usage 141
Predefined metrics for target tracking scaling policies 154
Predictive scaling metrics and dimensions 157
Log API calls using CloudTrail 159

Application Auto Scaling management events in CloudTrail 160
Application Auto Scaling event examples 160
Application Auto Scaling RemoveAction calls on CloudWatch 161
Amazon EventBridge 161
Application Auto Scaling events 162
Working with AWS SDKs 167
Code examples 169
Basics 169
Actions 170
Tagging support 208
Tagging example 208
Tags for security 209
Control access to tags 210
Security 212
Data protection 212
Identity and Access Management 213
Access control 214
How Application Auto Scaling works with IAM 214
AWS managed policies 220
Service-linked roles 230
Identity-based policy examples 236
Troubleshooting 249
Permissions validation 251
AWS PrivateLink 253
Create an interface VPC endpoint 253
Create a VPC endpoint policy 253
Resilience 254
Infrastructure security 255
Compliance validation 255
Quotas 256
Document history 257

What is Application Auto Scaling?

Application Auto Scaling is a web service for developers and system administrators who need a solution for automatically scaling their scalable resources for individual AWS services beyond Amazon EC2 Auto Scaling. With Application Auto Scaling, you can configure automatic scaling for the following resources:

- WorkSpaces Applications fleets
- Aurora replicas
- Amazon Comprehend document classification and entity recognizer endpoints
- DynamoDB tables and global secondary indexes
- Amazon ECS services
- ElastiCache replication groups (Redis OSS and Valkey) and Memcached clusters
- Amazon EMR clusters
- Amazon Keyspaces (for Apache Cassandra) tables
- Lambda function provisioned concurrency
- Amazon Managed Streaming for Apache Kafka (MSK) broker storage
- Amazon Neptune clusters
- SageMaker AI endpoint variants
- SageMaker AI inference components
- SageMaker AI Serverless provisioned concurrency
- Spot Fleet requests
- Pool of Amazon WorkSpaces
- Custom resources provided by your own applications or services. For more information, see the GitHub repository.

To see the regional availability for any of the AWS services listed above, see the Region table.

For information about scaling your fleet of Amazon EC2 instances using Auto Scaling groups, see the Amazon EC2 Auto Scaling User Guide.

Features of Application Auto Scaling

Application Auto Scaling allows you to automatically scale your scalable resources according to conditions that you define.

- **Target tracking scaling** – Scale a resource based on a target value for a specific CloudWatch metric.
- **Step scaling** – Scale a resource based on a set of scaling adjustments that vary based on the size of the alarm breach.
- **Scheduled scaling** – Scale a resource one time only or on a recurring schedule.
- **Predictive scaling** – Scale a resource proactively to match anticipated load based on historical data.

Work with Application Auto Scaling

You can configure scaling using the following interfaces depending on the resource that you are scaling:

- **AWS Management Console** – Provides a web interface that you can use to configure scaling. Sign up for an AWS account and sign into the AWS Management Console. Then, open the service console for one of the resources listed in the introduction. For example, to scale a Lambda function, open the AWS Lambda console. Ensure that you open the console in the same AWS Region as the resource that you want to work with.

  > **Note**
  >
  > Console access is not available for all resources. For more information, see AWS services that you can use with Application Auto Scaling.

- **AWS Command Line Interface (AWS CLI)** – Provides commands for a broad set of AWS services, and is supported on Windows, macOS, and Linux. To get started, see AWS Command Line Interface. For a list of commands, see application-autoscaling in the *AWS CLI Command Reference*.
- **AWS Tools for Windows PowerShell** – Provides commands for a broad set of AWS products for those who script in the PowerShell environment. To get started, see the AWS Tools for

PowerShell User Guide. For more information, see the AWS Tools for PowerShell Cmdlet Reference.

- **AWS SDKs** – Provides language-specific API operations and takes care of many of the connection details, such as calculating signatures, handling request retries, and handling errors. For more information, see Tools to Build on AWS.
- **HTTPS API** – Provides low-level API actions that you call using HTTPS requests. For more information, see the Application Auto Scaling API Reference.
- **CloudFormation** – Supports configuring scaling using a CloudFormation template. For more information, see Configure Application Auto Scaling resources using AWS CloudFormation.

To connect programmatically to an AWS service, you use an endpoint. For information about endpoints for calls to Application Auto Scaling, see Application Auto Scaling endpoints and quotas in the *AWS General Reference*.

Application Auto Scaling concepts

This topic explains key concepts to help you learn about Application Auto Scaling and start using it.

Scalable target

An entity that you create to specify the resource that you want to scale. Each scalable target is uniquely identified by a service namespace, resource ID, and scalable dimension, which represents some capacity dimension of the underlying service. For example, an Amazon ECS service supports auto scaling of its task count, a DynamoDB table supports auto scaling of the read and write capacity of the table and its global secondary indexes, and an Aurora cluster supports scaling of its replica count.

> **Tip**
>
> Each scalable target also has a minimum and maximum capacity. Scaling policies will never go higher or lower than the minimum-maximum range. You can make out-of-band changes directly to the underlying resource that are outside of this range, which Application Auto Scaling doesn't know about. However, anytime a scaling policy is invoked or the `RegisterScalableTarget` API is called, Application Auto Scaling retrieves the current capacity and compares it to the minimum and maximum capacity.

If it falls outside of the minimum-maximum range, then the capacity is updated to comply with the set minimum and maximum.

Scale in

When Application Auto Scaling automatically decreases capacity for a scalable target, the scalable target *scales in*. When scaling policies are set, they cannot scale in the scalable target lower than its minimum capacity.

Scale out

When Application Auto Scaling automatically increases capacity for a scalable target, the scalable target *scales out*. When scaling policies are set, they cannot scale out the scalable target higher than its maximum capacity.

Scaling policy

A scaling policy instructs Application Auto Scaling to track a specific CloudWatch metric. Then, it determines what scaling action to take when the metric is higher or lower than a certain threshold value. For example, you might want to scale out if the CPU usage across your cluster starts to rise, and scale in when it drops again.

The metrics that are used for auto scaling are published by the target service, but you can also publish your own metric to CloudWatch and then use it with a scaling policy.

A cooldown period between scaling activities lets the resource stabilize before another scaling activity starts. Application Auto Scaling continues to evaluate metrics during the cooldown period. When the cooldown period ends, the scaling policy initiates another scaling activity if needed. While a cooldown period is in effect, if a larger scale out is necessary based on the current metric value, the scaling policy scales out immediately.

Scheduled action

Scheduled actions automatically scale resources at a specific date and time. They work by modifying the minimum and maximum capacity for a scalable target, and therefore can be used to scale in and out on a schedule by setting the minimum capacity high or the maximum capacity low. For example, you can use scheduled actions to scale an application that doesn't consume resources on weekends by decreasing capacity on Friday and increasing capacity on the following Monday.

You can also use scheduled actions to optimize the minimum and maximum values over time to adapt to situations where higher than normal traffic is expected, for example, marketing campaigns or seasonal fluctuations. Doing this can help you improve performance for times when you need to scale out higher for the increasing usage, and reduce costs at times when you use fewer resources.

Learn more

AWS services that you can use with Application Auto Scaling — This section introduces you to the services that you can scale and helps you set up auto scaling by registering a scalable target. It also describes each of the IAM service-linked roles that Application Auto Scaling creates to access resources in the target service.

Target tracking scaling policies for Application Auto Scaling — One of the primary features of Application Auto Scaling is target tracking scaling policies. Learn how target tracking policies automatically adjust desired capacity to keep utilization at a constant level based on your configured metric and target values. For example, you can configure target tracking to keep the average CPU utilization for your Spot Fleet at 50 percent. Application Auto Scaling then launches or terminates EC2 instances as required to keep the aggregated CPU utilization across all servers at 50 percent.

AWS services that you can use with Application Auto Scaling

Application Auto Scaling integrates with other AWS services so that you can add scaling capabilities to meet your application's demand. Auto scaling is an optional feature of the service that is disabled by default in almost all cases.

The following table lists the AWS services that you can use with Application Auto Scaling, including information about supported methods for configuring auto scaling. You can also use Application Auto Scaling with custom resources.

- **Console access** – You can configure a compatible AWS service to start auto scaling by configuring a scaling policy in the console of the target service.
- **CLI access** – You can configure a compatible AWS service to start auto scaling using the AWS CLI.
- **SDK access** – You can configure a compatible AWS service to start auto scaling using the AWS SDKs.
- **CloudFormation access** – You can configure a compatible AWS service to start auto scaling using an CloudFormation stack template. For more information, see Configure Application Auto Scaling resources using AWS CloudFormation.

AWS service	Console access[1]	CLI access	SDK access	CloudFormation access
WorkSpaces Applications	Yes	Yes	Yes	Yes
Aurora	Yes	Yes	Yes	Yes
Amazon Comprehend	No	Yes	Yes	Yes

AWS service	Console access[1]	CLI access	SDK access	CloudFormation access
Amazon DynamoDB	Yes	Yes	Yes	Yes
Amazon ECS	Yes	Yes	Yes	Yes
Amazon ElastiCache	Yes	Yes	Yes	Yes
Amazon EMR	Yes	Yes	Yes	Yes
Amazon Keyspaces	Yes	Yes	Yes	Yes
Lambda	No	Yes	Yes	Yes
Amazon MSK	Yes	Yes	Yes	Yes
Amazon Neptune	No	Yes	Yes	Yes

AWS service	Console access[1]	CLI access	SDK access	CloudFormation access
SageMaker AI	Yes	Yes	Yes	Yes
Spot Fleet	Yes	Yes	Yes	Yes
WorkSpaces	Yes	Yes	Yes	Yes
Custom resources	No	Yes	Yes	Yes

[1] Console access for configuring scaling policies. Most services don't support configuring scheduled scaling from the console. Currently, only Amazon WorkSpaces Applications, ElastiCache, and Spot Fleet provide console access for scheduled scaling.

Amazon WorkSpaces Applications and Application Auto Scaling

You can scale WorkSpaces Applications fleets using target tracking scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate WorkSpaces Applications with Application Auto Scaling.

Service-linked role created for WorkSpaces Applications

The following service-linked role is automatically created in your AWS account when registering WorkSpaces Applications resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_AppStreamFleet`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `appstream.application-autoscaling.amazonaws.com`

Registering WorkSpaces Applications fleets as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for an WorkSpaces Applications fleet. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the WorkSpaces Applications console, then WorkSpaces Applications automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for an WorkSpaces Applications fleet. The following example registers the desired capacity of a fleet called `sample-fleet`, with a minimum capacity of one fleet instance and a maximum capacity of five fleet instances.

```
aws application-autoscaling register-scalable-target \
   --service-namespace appstream \
   --scalable-dimension appstream:fleet:DesiredCapacity \
   --resource-id fleet/sample-fleet \
   --min-capacity 1 \
   --max-capacity 5
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Fleet Auto Scaling for Amazon WorkSpaces Applications in the *Amazon WorkSpaces Applications Administration Guide*.

Amazon Aurora and Application Auto Scaling

You can scale Aurora DB clusters using target tracking scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate Aurora with Application Auto Scaling.

Service-linked role created for Aurora

The following service-linked role is automatically created in your AWS account when registering Aurora resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_RDSCluster`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `rds.application-autoscaling.amazonaws.com`

Registering Aurora DB clusters as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for an Aurora cluster. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the Aurora console, then Aurora automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for an Aurora cluster. The following example registers the count of Aurora Replicas in a cluster called `my-db-cluster`, with a minimum capacity of one Aurora Replica and a maximum capacity of eight Aurora Replicas.

```
aws application-autoscaling register-scalable-target \
  --service-namespace rds \
  --scalable-dimension rds:cluster:ReadReplicaCount \
  --resource-id cluster:my-db-cluster \
  --min-capacity 1 \
  --max-capacity 8
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Amazon Aurora Auto Scaling with Aurora Replicas in the *Amazon RDS User Guide for Aurora*.

Amazon Comprehend and Application Auto Scaling

You can scale Amazon Comprehend document classification and entity recognizer endpoints using target tracking scaling policies and scheduled scaling.

Use the following information to help you integrate Amazon Comprehend with Application Auto Scaling.

Service-linked role created for Amazon Comprehend

The following service-linked role is automatically created in your AWS account when registering Amazon Comprehend resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_ComprehendEndpoint`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `comprehend.application-autoscaling.amazonaws.com`

Registering Amazon Comprehend resources as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for an Amazon Comprehend document classification or entity recognizer endpoint. A scalable target is a resource that Application Auto Scaling can scale out and scale in.

Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

To configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a document classification endpoint. The following example registers the desired number of inference units to be used by the model for a document classifier endpoint using the endpoint's ARN, with a minimum capacity of one inference unit and a maximum capacity of three inference units.

```
aws application-autoscaling register-scalable-target \
  --service-namespace comprehend \
  --scalable-dimension comprehend:document-classifier-endpoint:DesiredInferenceUnits
 \
  --resource-id arn:aws:comprehend:us-west-2:123456789012:document-classifier-
endpoint/EXAMPLE \
  --min-capacity 1 \
  --max-capacity 3
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

  Call the register-scalable-target command for an entity recognizer endpoint. The following example registers the desired number of inference units to be used by the model for an entity recognizer using the endpoint's ARN, with a minimum capacity of one inference unit and a maximum capacity of three inference units.

```
aws application-autoscaling register-scalable-target \
  --service-namespace comprehend \
  --scalable-dimension comprehend:entity-recognizer-endpoint:DesiredInferenceUnits \
  --resource-id arn:aws:comprehend:us-west-2:123456789012:entity-recognizer-
endpoint/EXAMPLE \
  --min-capacity 1 \
```

```
  --max-capacity 3
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Auto scaling with endpoints in the *Amazon Comprehend Developer Guide*.

Amazon DynamoDB and Application Auto Scaling

You can scale DynamoDB tables and global secondary indexes using target tracking scaling policies and scheduled scaling.

Use the following information to help you integrate DynamoDB with Application Auto Scaling.

Service-linked role created for DynamoDB

The following service-linked role is automatically created in your AWS account when registering DynamoDB resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `dynamodb.application-autoscaling.amazonaws.com`

Registering DynamoDB resources as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for a DynamoDB table or global secondary index. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the DynamoDB console, then DynamoDB automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a table's write capacity. The following example registers the provisioned write capacity of a table called `my-table`, with a minimum capacity of five write capacity units and a maximum capacity of 10 write capacity units:

```
aws application-autoscaling register-scalable-target \
  --service-namespace dynamodb \
  --scalable-dimension dynamodb:table:WriteCapacityUnits \
  --resource-id table/my-table \
  --min-capacity 5 \
  --max-capacity 10
```

  If successful, this command returns the ARN of the scalable target:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

  Call the register-scalable-target command for a table's read capacity. The following example registers the provisioned read capacity of a table called `my-table`, with a minimum capacity of five read capacity units and a maximum capacity of 10 read units:

```
aws application-autoscaling register-scalable-target \
  --service-namespace dynamodb \
  --scalable-dimension dynamodb:table:ReadCapacityUnits \
  --resource-id table/my-table \
  --min-capacity 5 \
  --max-capacity 10
```

If successful, this command returns the ARN of the scalable target:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Call the register-scalable-target command for the write capacity of a global secondary index. The following example registers the provisioned write capacity of a global secondary index called `my-table-index`, with a minimum capacity of five write capacity units and a maximum capacity of 10 write capacity units:

```
aws application-autoscaling register-scalable-target \
  --service-namespace dynamodb \
  --scalable-dimension dynamodb:index:WriteCapacityUnits \
  --resource-id table/my-table/index/my-table-index \
  --min-capacity 5 \
  --max-capacity 10
```

If successful, this command returns the ARN of the scalable target:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Call the register-scalable-target command for the read capacity of a global secondary index. The following example registers the provisioned read capacity of a global secondary index called `my-table-index`, with a minimum capacity of five read capacity units and a maximum capacity of 10 read capacity units:

```
aws application-autoscaling register-scalable-target \
  --service-namespace dynamodb \
  --scalable-dimension dynamodb:index:ReadCapacityUnits \
  --resource-id table/my-table/index/my-table-index \
  --min-capacity 5 \
  --max-capacity 10
```

If successful, this command returns the ARN of the scalable target:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

If you are just getting started with Application Auto Scaling, you can find additional useful information about scaling your DynamoDB resources in the following documentation:

- Managing throughput capacity with DynamoDB Auto Scaling in the *Amazon DynamoDB Developer Guide*
- Evaluate your table's auto scaling settings in the *Amazon DynamoDB Developer Guide*
- How to use CloudFormation to configure auto scaling for DynamoDB tables and indexes on the AWS Blog

Amazon ECS and Application Auto Scaling

You can scale ECS services using target tracking scaling policies, predictive scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate Amazon ECS with Application Auto Scaling.

Service-linked role created for Amazon ECS

The following service-linked role is automatically created in your AWS account when registering Amazon ECS resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_ECSService`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `ecs.application-autoscaling.amazonaws.com`

Registering ECS services as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for an Amazon ECS service. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the Amazon ECS console, then Amazon ECS automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for an Amazon ECS service. The following example registers a scalable target for a service called `sample-app-service`, running on the `default` cluster, with a minimum task count of one task and a maximum task count of 10 tasks.

```
aws application-autoscaling register-scalable-target \
```

```
  --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/default/sample-app-service \
  --min-capacity 1 \
  --max-capacity 10
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

If you are just getting started with Application Auto Scaling, you can find additional useful information about scaling your Amazon ECS resources in the following documentation:

- Service auto scaling in the *Amazon Elastic Container Service Developer Guide*
- Optimize Amazon ECS service auto scaling in the *Amazon Elastic Container Service Developer Guide*

> **Note**
>
> For instructions for suspending scale-out processes while Amazon ECS deployments are in progress, see the following documentation:
> Service auto scaling and deployments in the *Amazon Elastic Container Service Developer Guide*

ElastiCache and Application Auto Scaling

You can horizontally scale Amazon ElastiCache replication groups (Redis OSS and Valkey) and Memcached self-designed clusters using target tracking scaling policies and scheduled scaling.

To integrate ElastiCache with Application Auto Scaling, use the following information.

Service-linked role created for ElastiCache

The following service-linked role is automatically created in your AWS account when registering ElastiCache resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `elasticache.application-autoscaling.amazonaws.com`

Registering ElastiCache resources as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for an ElastiCache replication group, cluster, or node. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the ElastiCache console, then ElastiCache automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

Call the register-scalable-target command for an ElastiCache replication group. The following example registers the desired number of node groups for a replication group called `mycluster1`, with a minimum capacity of one and a maximum capacity of five.

```
aws application-autoscaling register-scalable-target \
   --service-namespace elasticache \
   --scalable-dimension elasticache:replication-group:NodeGroups \
   --resource-id replication-group/mycluster1 \
   --min-capacity 1 \
   --max-capacity 5
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

The following example registers the desired number of replicas per node group for a replication group called `mycluster2`, with a minimum capacity of one and a maximum capacity of five.

```
aws application-autoscaling register-scalable-target \
   --service-namespace elasticache \
   --scalable-dimension elasticache:replication-group:Replicas \
   --resource-id replication-group/mycluster2 \
   --min-capacity 1 \
   --max-capacity 5
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/234abcd56ab78cd901ef1234567890ab1234"
}
```

The following example registers the desired number of nodes for a cluster called `mynode1`, with a minimum capacity of 20 and a maximum capacity of 50.

```
aws application-autoscaling register-scalable-target \
```

```
    --service-namespace elasticache \
    --scalable-dimension elasticache:cache-cluster:Nodes \
    --resource-id cache-cluster/mynode1 \
    --min-capacity 20 \
    --max-capacity 50
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/01234abcd56ab78cd901ef1234567890ab12"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Auto Scaling Valkey and Redis OSS clusters and Scaling clusters for Memcached in the *Amazon ElastiCache User Guide*.

Amazon Keyspaces (for Apache Cassandra) and Application Auto Scaling

You can scale Amazon Keyspaces tables using target tracking scaling policies and scheduled scaling.

Use the following information to help you integrate Amazon Keyspaces with Application Auto Scaling.

Service-linked role created for Amazon Keyspaces

The following service-linked role is automatically created in your AWS account when registering Amazon Keyspaces resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_CassandraTable`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `cassandra.application-autoscaling.amazonaws.com`

Registering Amazon Keyspaces tables as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for an Amazon Keyspaces table. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the Amazon Keyspaces console, then Amazon Keyspaces automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for an Amazon Keyspaces table. The following example registers the provisioned write capacity of a table called `mytable`, with a minimum capacity of five write capacity units and a maximum capacity of 10 write capacity units.

```
aws application-autoscaling register-scalable-target \
  --service-namespace cassandra \
  --scalable-dimension cassandra:table:WriteCapacityUnits \
  --resource-id keyspace/mykeyspace/table/mytable \
  --min-capacity 5 \
  --max-capacity 10
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

The following example registers the provisioned read capacity of a table called `mytable`, with a minimum capacity of five read capacity units and a maximum capacity of 10 read capacity units.

```
aws application-autoscaling register-scalable-target \
  --service-namespace cassandra \
  --scalable-dimension cassandra:table:ReadCapacityUnits \
  --resource-id keyspace/mykeyspace/table/mytable \
  --min-capacity 5 \
  --max-capacity 10
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Manage throughput capacity automatically with Amazon Keyspaces auto scaling in the *Amazon Keyspaces Developer Guide*.

AWS Lambda and Application Auto Scaling

You can scale AWS Lambda provisioned concurrency using target tracking scaling policies and scheduled scaling.

Use the following information to help you integrate Lambda with Application Auto Scaling.

Service-linked role created for Lambda

The following service-linked role is automatically created in your AWS account when registering Lambda resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_LambdaConcurrency`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `lambda.application-autoscaling.amazonaws.com`

Registering Lambda functions as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for a Lambda function. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

To configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a Lambda function. The following example registers the provisioned concurrency for an alias called BLUE for a function called `my-function`, with a minimum capacity of 0 and a maximum capacity of 100.

```
aws application-autoscaling register-scalable-target \
  --service-namespace lambda \
  --scalable-dimension lambda:function:ProvisionedConcurrency \
  --resource-id function:my-function:BLUE \
```

```
  --min-capacity 0 \
  --max-capacity 100
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

If you are just getting started with Application Auto Scaling, you can find additional useful information about scaling your Lambda functions in the following documentation:

- Configuring provisioned concurrency in the *AWS Lambda Developer Guide*
- Scheduling Lambda Provisioned Concurrency for recurring peak usage on the AWS Blog

Amazon Managed Streaming for Apache Kafka (MSK) and Application Auto Scaling

You can scale out Amazon MSK cluster storage using target tracking scaling policies. Scale in by the target tracking policy is disabled.

Use the following information to help you integrate Amazon MSK with Application Auto Scaling.

Service-linked role created for Amazon MSK

The following service-linked role is automatically created in your AWS account when registering Amazon MSK resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_KafkaCluster`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `kafka.application-autoscaling.amazonaws.com`

Registering Amazon MSK cluster storage as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create a scaling policy for the storage volume size per broker of an Amazon MSK cluster. A scalable target is a resource that Application Auto Scaling can scale. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the Amazon MSK console, then Amazon MSK automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for an Amazon MSK cluster. The following example registers the storage volume size per broker of an Amazon MSK cluster, with a minimum capacity of 100 GiB and a maximum capacity of 800 GiB.

```
aws application-autoscaling register-scalable-target \
   --service-namespace kafka \
   --scalable-dimension kafka:broker-storage:VolumeSize \
   --resource-id arn:aws:kafka:us-east-1:123456789012:cluster/demo-
cluster-1/6357e0b2-0e6a-4b86-a0b4-70df934c2e31-5 \
   --min-capacity 100 \
   --max-capacity 800
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

> **Note**
>
> When an Amazon MSK cluster is the scalable target, scale in is disabled and cannot be enabled.

Related resources

For more information, see Automatic scaling for Amazon MSK clusters in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

Amazon Neptune and Application Auto Scaling

You can scale Neptune clusters using target tracking scaling policies and scheduled scaling.

Use the following information to help you integrate Neptune with Application Auto Scaling.

Service-linked role created for Neptune

The following service-linked role is automatically created in your AWS account when registering Neptune resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_NeptuneCluster`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `neptune.application-autoscaling.amazonaws.com`

Registering Neptune clusters as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for a Neptune cluster. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

To configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a Neptune cluster. The following example registers the desired capacity of a cluster called `mycluster`, with a minimum capacity of one and a maximum capacity of eight.

```
aws application-autoscaling register-scalable-target \
  --service-namespace neptune \
  --scalable-dimension neptune:cluster:ReadReplicaCount \
  --resource-id cluster:mycluster \
  --min-capacity 1 \
  --max-capacity 8
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Auto scaling the number of replicas in an Amazon Neptune DB cluster in the *Neptune User Guide*.

Amazon SageMaker AI and Application Auto Scaling

You can scale SageMaker AI endpoint variants, provisioned concurrency for serverless endpoints, and inference components using target tracking scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate SageMaker AI with Application Auto Scaling.

Service-linked role created for SageMaker AI

The following service-linked role is automatically created in your AWS account when registering SageMaker AI resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `sagemaker.application-autoscaling.amazonaws.com`

Registering SageMaker AI endpoint variants as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for a SageMaker AI model (variant). A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the SageMaker AI console, then SageMaker AI automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a product variant. The following example registers the desired instance count for a product variant called `my-variant`, running on the `my-endpoint` endpoint, with a minimum capacity of one instance and a maximum capacity of eight instances.

  ```
  aws application-autoscaling register-scalable-target \
    --service-namespace sagemaker \
    --scalable-dimension sagemaker:variant:DesiredInstanceCount \
    --resource-id endpoint/my-endpoint/variant/my-variant \
    --min-capacity 1 \
    --max-capacity 8
  ```

  If successful, this command returns the ARN of the scalable target.

  ```
  {
      "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
  target/1234abcd56ab78cd901ef1234567890ab123"
  }
  ```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Registering the provisioned concurrency of serverless endpoints as scalable targets with Application Auto Scaling

Application Auto Scaling also requires a scalable target before you can create scaling policies or scheduled actions for the provisioned concurrency of serverless endpoints.

If you configure auto scaling using the SageMaker AI console, then SageMaker AI automatically registers a scalable target for you.

Otherwise, use one of the following methods to register the scalable target:

- AWS CLI:

  Call the register-scalable-target command for a product variant. The following example registers the provisioned concurrency for a product variant called `my-variant`, running on the `my-endpoint` endpoint, with a minimum capacity of one and a maximum capacity of ten.

  ```
  aws application-autoscaling register-scalable-target \
    --service-namespace sagemaker \
    --scalable-dimension sagemaker:variant:DesiredProvisionedConcurrency \
    --resource-id endpoint/my-endpoint/variant/my-variant \
    --min-capacity 1 \
    --max-capacity 10
  ```

  If successful, this command returns the ARN of the scalable target.

  ```
  {
      "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
  target/1234abcd56ab78cd901ef1234567890ab123"
  }
  ```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Registering inference components as scalable targets with Application Auto Scaling

Application Auto Scaling also requires a scalable target before you can create scaling policies or scheduled actions for inference components.

- AWS CLI:

  Call the register-scalable-target command for an inference component. The following example registers the desired copy count for an inference component called `my-inference-component`, with a minimum capacity of zero copies and a maximum capacity of three copies.

```
aws application-autoscaling register-scalable-target \
  --service-namespace sagemaker \
  --scalable-dimension sagemaker:inference-component:DesiredCopyCount \
  --resource-id inference-component/my-inference-component \
  --min-capacity 0 \
  --max-capacity 3
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

If you are just getting started with Application Auto Scaling, you can find additional useful information about scaling your SageMaker AI resources in the *Amazon SageMaker AI Developer Guide*:

- Automatically scale Amazon SageMaker AI models
- Automatically scale Provisioned Concurrency for a serverless endpoint

- Set auto scaling policies for multi-model endpoint deployments
- Autoscale an asynchronous endpoint

> **Note**
>
> In 2023, SageMaker AI introduced new inference capabilities built on real-time inference endpoints. You create a SageMaker AI endpoint with an endpoint configuration that defines the instance type and initial instance count for the endpoint. Then, create an inference component, which is a SageMaker AI hosting object that you can use to deploy a model to an endpoint. For information about scaling inference components, see Amazon SageMaker AI adds new inference capabilities to help reduce foundation model deployment costs and latency and Reduce model deployment costs by 50% on average using the latest features of Amazon SageMaker AI on the AWS Blog.

Amazon EC2 Spot Fleet and Application Auto Scaling

You can scale Spot Fleets using target tracking scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate Spot Fleet with Application Auto Scaling.

Service-linked role created for Spot Fleet

The following service-linked role is automatically created in your AWS account when registering Spot Fleet resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_EC2SpotFleetRequest`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `ec2.application-autoscaling.amazonaws.com`

Registering Spot Fleets as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for a Spot Fleet. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the Spot Fleet console, then Spot Fleet automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a Spot Fleet. The following example registers the target capacity of a Spot Fleet using its request ID, with a minimum capacity of two instances and a maximum capacity of 10 instances.

```
aws application-autoscaling register-scalable-target \
  --service-namespace ec2 \
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity \
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE \
  --min-capacity 2 \
  --max-capacity 10
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Understand automatic scaling for Spot Fleet in the *Amazon EC2 User Guide*.

Amazon WorkSpaces and Application Auto Scaling

You can scale a pool of WorkSpaces using target tracking scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate WorkSpaces with Application Auto Scaling.

Service-linked role created for WorkSpaces

Application Auto Scaling automatically creates the service-linked role named AWSServiceRoleForApplicationAutoScaling_WorkSpacesPool in your AWS account when you register WorkSpaces resources as scalable targets with Application Auto Scaling. For more information, see Service-linked roles for Application Auto Scaling.

This service-linked role uses the managed policy AWSApplicationAutoscalingWorkSpacesPoolPolicy. This policy grants Application Auto Scaling permissions to call Amazon WorkSpaces on your behalf. For more information, see AWSApplicationAutoscalingWorkSpacesPoolPolicy in the *AWS Managed Policy Reference*.

Service principal used by the service-linked role

The service-linked role trusts the following service principal to assume the role:

- `workspaces.application-autoscaling.amazonaws.com`

Registering WorkSpaces pools as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for WorkSpaces. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

If you configure auto scaling using the WorkSpaces console, then WorkSpaces automatically registers a scalable target for you.

If you want to configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a pool of WorkSpaces. The following example registers the target capacity of a pool of WorkSpaces using its request ID, with a minimum capacity of two virtual desktops and a maximum capacity of ten virtual desktops.

```
aws application-autoscaling register-scalable-target \
  --service-namespace workspaces \
  --resource-id workspacespool/wspool-abcdef012 \
  --scalable-dimension workspaces:workspacespool:DesiredUserSessions \
  --min-capacity 2 \
  --max-capacity 10
```

  If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

For more information, see Auto Scaling for WorkSpaces Pools in the *Amazon WorkSpaces Administration Guide*.

Custom resources and Application Auto Scaling

You can scale custom resources using target tracking scaling policies, step scaling policies, and scheduled scaling.

Use the following information to help you integrate custom resources with Application Auto Scaling.

Service-linked role created for custom resources

The following service-linked role is automatically created in your AWS account when registering custom resources as scalable targets with Application Auto Scaling. This role allows Application Auto Scaling to perform supported operations within your account. For more information, see Service-linked roles for Application Auto Scaling.

- `AWSServiceRoleForApplicationAutoScaling_CustomResource`

Service principal used by the service-linked role

The service-linked role in the previous section can be assumed only by the service principal authorized by the trust relationships defined for the role. The service-linked role used by Application Auto Scaling grants access to the following service principal:

- `custom-resource.application-autoscaling.amazonaws.com`

Registering custom resources as scalable targets with Application Auto Scaling

Application Auto Scaling requires a scalable target before you can create scaling policies or scheduled actions for a custom resource. A scalable target is a resource that Application Auto Scaling can scale out and scale in. Scalable targets are uniquely identified by the combination of resource ID, scalable dimension, and namespace.

To configure auto scaling using the AWS CLI or one of the AWS SDKs, you can use the following options:

- AWS CLI:

  Call the register-scalable-target command for a custom resource. The following example registers a custom resource as a scalable target, with a minimum desired count of one capacity unit and a maximum desired count of 10 capacity units. The `custom-resource-id.txt` file contains a string that identifies the resource ID, which represents the path to the custom resource through your Amazon API Gateway endpoint.

```
aws application-autoscaling register-scalable-target \
  --service-namespace custom-resource \
  --scalable-dimension custom-resource:ResourceType:Property \
  --resource-id file://~/custom-resource-id.txt \
  --min-capacity 1 \
  --max-capacity 10
```

Contents of `custom-resource-id.txt`:

```
https://example.execute-api.us-west-2.amazonaws.com/prod/
scalableTargetDimensions/1-23456789
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

- AWS SDK:

  Call the RegisterScalableTarget operation and provide `ResourceId`, `ScalableDimension`, `ServiceNamespace`, `MinCapacity`, and `MaxCapacity` as parameters.

Related resources

If you are just getting started with Application Auto Scaling, you can find additional useful information about scaling your custom resources in the following documentation:

GitHub repository

Configure Application Auto Scaling resources using AWS CloudFormation

Application Auto Scaling is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, and CloudFormation provisions and configures those resources for you.

When you use CloudFormation, you can reuse your template to set up your Application Auto Scaling resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Application Auto Scaling and CloudFormation templates

To provision and configure resources for Application Auto Scaling and related services, you must understand CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use CloudFormation Designer to help you get started with CloudFormation templates. For more information, see What is CloudFormation Designer? in the *AWS CloudFormation User Guide*.

When you create a stack template for Application Auto Scaling resources, you must provide the following:

- A namespace for the target service (for example, **`appstream`**). See the AWS::ApplicationAutoScaling::ScalableTarget reference to obtain service namespaces.
- A scalable dimension associated with the target resource (for example, **`appstream:fleet:DesiredCapacity`**). See the AWS::ApplicationAutoScaling::ScalableTarget reference to obtain scalable dimensions.
- A resource ID for the target resource (for example, **`fleet/sample-fleet`**). See the AWS::ApplicationAutoScaling::ScalableTarget reference for information about the syntax and examples of specific resource IDs.
- A service-linked role for the target resource (for example, **`arn:aws:iam::012345678910:role/aws-service-role/appstream.application-autoscaling.amazonaws.com/`**

`AWSServiceRoleForApplicationAutoScaling_AppStreamFleet`). See the Service-linked role ARN reference table to obtain role ARNs.

To learn more about Application Auto Scaling resources, see the Application Auto Scaling reference in the *AWS CloudFormation User Guide*.

Example template snippets

You can find example snippets to include in CloudFormation templates in the following sections of the *AWS CloudFormation User Guide*:

- For examples of scaling policies and scheduled actions, see Configure Application Auto Scaling resources with AWS CloudFormation.
- For more examples of scaling policies, see AWS::ApplicationAutoScaling::ScalingPolicy.

Learn more about CloudFormation

To learn more about CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- CloudFormation API Reference
- AWS CloudFormation Command Line Interface User Guide

Scheduled scaling for Application Auto Scaling

With scheduled scaling, you can set up automatic scaling for your application based on predictable load changes by creating scheduled actions that increase or decrease capacity at specific times. This allows you to scale your application proactively to match predictable load changes.

For example, let's say you experience a regular weekly traffic pattern where load increases mid-week and declines toward the end of the week. You can configure a scaling schedule in Application Auto Scaling that aligns with this pattern:

- On Wednesday morning, one scheduled action increases capacity by increasing the previously set minimum capacity of the scalable target.
- On Friday evening, another scheduled action decreases capacity by decreasing the previously set maximum capacity of the scalable target.

These scheduled scaling actions allow you to optimize costs and performance. Your application has sufficient capacity to handle the mid-week traffic peak, but does not over-provision unneeded capacity at other times.

You can use scheduled scaling and scaling policies together to get the benefits of proactive and reactive approaches to scaling. After a scheduled scaling action runs, the scaling policy can continue to make decisions about whether to further scale capacity. This helps you ensure that you have sufficient capacity to handle the load for your application. While your application scales to match demand, current capacity must fall within the minimum and maximum capacity that was set by your scheduled action.

Contents

- How scheduled scaling for Application Auto Scaling works
- Create scheduled actions for Application Auto Scaling using the AWS CLI
- Describe scheduled scaling for Application Auto Scaling using the AWS CLI
- Schedule recurring scaling actions using Application Auto Scaling
- Turn off scheduled scaling for a scalable target
- Delete a scheduled action for Application Auto Scaling using the AWS CLI

How scheduled scaling for Application Auto Scaling works

This topic describes how scheduled scaling works and introduces the key considerations you need to understand to use it effectively.

Contents

- How it works
- Considerations
- Commonly used commands for scheduled action creation, management, and deletion
- Related resources
- Limitations

How it works

To use scheduled scaling, create *scheduled actions*, which tell Application Auto Scaling to perform scaling activities at specific times. When you create a scheduled action, you specify the scalable target, when the scaling activity should occur, a minimum capacity, and a maximum capacity. You can create scheduled actions that scale one time only or that scale on a recurring schedule.

At the specified time, Application Auto Scaling scales based on the new capacity values, by comparing current capacity to the specified minimum and maximum capacity.

- If current capacity is less than the specified minimum capacity, Application Auto Scaling scales out (increases capacity) to the specified minimum capacity.
- If current capacity is greater than the specified maximum capacity, Application Auto Scaling scales in (decreases capacity) to the specified maximum capacity.

Considerations

When you create a scheduled action, keep the following in mind:

- A scheduled action sets the `MinCapacity` and `MaxCapacity` to what is specified by the scheduled action at the date and time specified. The request can optionally include only one of these sizes. For example, you can create a scheduled action with only the minimum capacity specified. In some cases, however, you must include both sizes to ensure that the new minimum

capacity is not greater than the maximum capacity, or the new maximum capacity is not less than the minimum capacity.

- By default, the recurring schedules that you set are in Coordinated Universal Time (UTC). You can change the time zone to correspond to your local time zone or a time zone for another part of your network. When you specify a time zone that observes daylight saving time, the action automatically adjusts for Daylight Saving Time (DST). For more information, see Schedule recurring scaling actions using Application Auto Scaling.
- You can temporarily turn off scheduled scaling for a scalable target. This helps you prevent scheduled actions from being active without having to delete them. You can then resume scheduled scaling when you want to use it again. For more information, see Suspend and resume scaling for Application Auto Scaling.
- The order in which scheduled actions run is guaranteed for the same scalable target, but not for scheduled actions across scalable targets.
- To complete a scheduled action successfully, the specified resource must be in a scalable state in the target service. If it isn't, the request fails and returns an error message, for example, `Resource Id [ActualResourceId] is not scalable. Reason: The status of all DB instances must be 'available' or 'incompatible-parameters'.`
- Due to the distributed nature of Application Auto Scaling and the target services, the delay between the time the scheduled action is triggered and the time the target service honors the scaling action might be a few seconds. Because scheduled actions are run in the order that they are specified, scheduled actions with start times close to each other can take longer to run.

Commonly used commands for scheduled action creation, management, and deletion

The commonly used commands for working with schedule scaling include:

- register-scalable-target to register AWS or custom resources as scalable targets (a resource that Application Auto Scaling can scale), and to suspend and resume scaling.
- put-scheduled-action to add or modify scheduled actions for an existing scalable target.
- describe-scaling-activities to return information about scaling activities in an AWS Region.
- describe-scheduled-actions to return information about scheduled actions in an AWS Region.
- delete-scheduled-action to delete a scheduled action.

Related resources

For a detailed example of using scheduled scaling, see the blog post Scheduling AWS Lambda Provisioned Concurrency for recurring peak usage on the AWS Compute Blog.

For information about creating scheduled actions for Auto Scaling groups, see Scheduled scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Limitations

The following are limitations when using scheduled scaling:

- The names of scheduled actions must be unique per scalable target.
- Application Auto Scaling doesn't provide second-level precision in schedule expressions. The finest resolution using a cron expression is 1 minute.
- The scalable target can't be an Amazon MSK cluster. Scheduled scaling is not supported for Amazon MSK.
- Console access to view, add, update, or remove scheduled actions on scalable resources depends on the resource that you use. For more information, see AWS services that you can use with Application Auto Scaling.

Create scheduled actions for Application Auto Scaling using the AWS CLI

The following examples show how to create scheduled actions using the AWS CLI put-scheduled-action command. When you specify the new capacity, you can specify a minimum capacity, a maximum capacity, or both.

These examples use scalable targets for a few of the services that integrate with Application Auto Scaling. To use a different scalable target, specify its namespace in `--service-namespace`, its scalable dimension in `--scalable-dimension`, and its resource ID in `--resource-id`.

When using the AWS CLI, remember that your commands run in the AWS Region configured for your profile. If you want to run the commands in a different Region, either change the default Region for your profile, or use the `--region` parameter with the command.

Examples

- Create a scheduled action that occurs only once

- Create a scheduled action that runs on a recurring interval
- Create a scheduled action that runs on a recurring schedule
- Create a one-time scheduled action that specifies a time zone
- Create a recurring scheduled action that specifies a time zone

Create a scheduled action that occurs only once

To automatically scale your scalable target one time only, at a specified date and time, use the `--schedule "at(yyyy-mm-ddThh:mm:ss)"` option.

Example: To scale out one time only

The following is an example of creating a scheduled action to scale out capacity at a specific date and time.

At the date and time specified for `--schedule` (10:00 PM UTC on March 31, 2021), if the value specified for `MinCapacity` is above the current capacity, Application Auto Scaling scales out to `MinCapacity`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action --service-namespace custom-resource \
  --scalable-dimension custom-resource:ResourceType:Property \
  --resource-id file://~/custom-resource-id.txt \
  --scheduled-action-name scale-out \
  --schedule "at(2021-03-31T22:00:00)" \
  --scalable-target-action MinCapacity=3
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace custom-resource ^
  --scalable-dimension custom-resource:ResourceType:Property ^
  --resource-id file://~/custom-resource-id.txt ^
  --scheduled-action-name scale-out ^
  --schedule "at(2021-03-31T22:00:00)" ^
  --scalable-target-action MinCapacity=3
```

When this scheduled action runs, if the maximum capacity is less than the value specified for minimum capacity, you must specify a new minimum and maximum capacity, and not just the minimum capacity.

Example: To scale in one time only

The following is an example of creating a scheduled action to scale in capacity at a specific date and time.

At the date and time specified for `--schedule` (10:30 PM UTC on March 31, 2021), if the value specified for `MaxCapacity` is below the current capacity, Application Auto Scaling scales in to `MaxCapacity`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action --service-namespace custom-resource \
  --scalable-dimension custom-resource:ResourceType:Property \
  --resource-id file://~/custom-resource-id.txt \
  --scheduled-action-name scale-in \
  --schedule "at(2021-03-31T22:30:00)" \
  --scalable-target-action MinCapacity=0,MaxCapacity=0
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace custom-resource ^
  --scalable-dimension custom-resource:ResourceType:Property ^
  --resource-id file://~/custom-resource-id.txt ^
  --scheduled-action-name scale-in ^
  --schedule "at(2021-03-31T22:30:00)" ^
  --scalable-target-action MinCapacity=0,MaxCapacity=0
```

Create a scheduled action that runs on a recurring interval

To schedule scaling at a recurring interval, use the `--schedule "rate(value unit)"` option. The value must be a positive integer. The unit can be `minute`, `minutes`, `hour`, `hours`, `day`, or `days`. For more information, see Rate expressions in the *Amazon EventBridge User Guide*.

The following is an example of a scheduled action that uses a rate expression.

On the specified schedule (every 5 hours starting on January 30, 2021 at 12:00 PM UTC and ending on January 31, 2021 at 10:00 PM UTC), if the value specified for `MinCapacity` is above the current capacity, Application Auto Scaling scales out to `MinCapacity`. If the value specified for `MaxCapacity` is below the current capacity, Application Auto Scaling scales in to `MaxCapacity`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service \
  --scheduled-action-name my-recurring-action \
  --schedule "rate(5 hours)" \
  --start-time 2021-01-30T12:00:00 \
  --end-time 2021-01-31T22:00:00 \
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace ecs ^
  --scalable-dimension ecs:service:DesiredCount ^
  --resource-id service/my-cluster/my-service ^
  --scheduled-action-name my-recurring-action ^
  --schedule "rate(5 hours)" ^
  --start-time 2021-01-30T12:00:00 ^
  --end-time 2021-01-31T22:00:00 ^
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

Create a scheduled action that runs on a recurring schedule

To schedule scaling on a recurring schedule, use the `--schedule "cron(fields)"` option. For more information, see Schedule recurring scaling actions using Application Auto Scaling.

The following is an example of a scheduled action that uses a cron expression.

On the specified schedule (every day at 9:00 AM UTC), if the value specified for `MinCapacity` is above the current capacity, Application Auto Scaling scales out to `MinCapacity`. If the value specified for `MaxCapacity` is below the current capacity, Application Auto Scaling scales in to `MaxCapacity`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action --service-namespace appstream \
  --scalable-dimension appstream:fleet:DesiredCapacity \
  --resource-id fleet/sample-fleet \
  --scheduled-action-name my-recurring-action \
  --schedule "cron(0 9 * * ? *)" \
  --scalable-target-action MinCapacity=10,MaxCapacity=50
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace appstream ^
  --scalable-dimension appstream:fleet:DesiredCapacity ^
  --resource-id fleet/sample-fleet ^
  --scheduled-action-name my-recurring-action ^
  --schedule "cron(0 9 * * ? *)" ^
  --scalable-target-action MinCapacity=10,MaxCapacity=50
```

Create a one-time scheduled action that specifies a time zone

Scheduled actions are set to the UTC time zone by default. To specify a different time zone, include the `--timezone` option and specify the canonical name for the time zone (`America/New_York`, for example). For more information, see https://www.joda.org/joda-time/timezones.html, which provides information about the IANA time zones that are supported when calling put-scheduled-action.

The following is an example that uses the `--timezone` option when creating a scheduled action to scale capacity at a specific date and time.

At the date and time specified for `--schedule` (5:00 PM local time on January 31, 2021), if the value specified for `MinCapacity` is above the current capacity, Application Auto Scaling scales out to `MinCapacity`. If the value specified for `MaxCapacity` is below the current capacity, Application Auto Scaling scales in to `MaxCapacity`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action --service-namespace comprehend \
  --scalable-dimension comprehend:document-classifier-endpoint:DesiredInferenceUnits \
  --resource-id arn:aws:comprehend:us-west-2:123456789012:document-classifier-endpoint/
EXAMPLE \
  --scheduled-action-name  my-one-time-action \
  --schedule "at(2021-01-31T17:00:00)" --timezone "America/New_York" \
  --scalable-target-action MinCapacity=1,MaxCapacity=3
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace comprehend ^
  --scalable-dimension comprehend:document-classifier-endpoint:DesiredInferenceUnits ^
  --resource-id arn:aws:comprehend:us-west-2:123456789012:document-classifier-endpoint/
EXAMPLE ^
```

```
  --scheduled-action-name  my-one-time-action ^
  --schedule "at(2021-01-31T17:00:00)" --timezone "America/New_York" ^
  --scalable-target-action MinCapacity=1,MaxCapacity=3
```

Create a recurring scheduled action that specifies a time zone

The following is an example that uses the `--timezone` option when creating a recurring scheduled action to scale capacity. For more information, see Schedule recurring scaling actions using Application Auto Scaling.

On the specified schedule (every Monday through Friday at 6:00 PM local time), if the value specified for `MinCapacity` is above the current capacity, Application Auto Scaling scales out to `MinCapacity`. If the value specified for `MaxCapacity` is below the current capacity, Application Auto Scaling scales in to `MaxCapacity`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action --service-namespace lambda \
  --scalable-dimension lambda:function:ProvisionedConcurrency \
  --resource-id function:my-function:BLUE \
  --scheduled-action-name my-recurring-action \
  --schedule "cron(0 18 ? * MON-FRI *)" --timezone "Etc/GMT+9" \
  --scalable-target-action MinCapacity=10,MaxCapacity=50
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace lambda ^
  --scalable-dimension lambda:function:ProvisionedConcurrency ^
  --resource-id function:my-function:BLUE ^
  --scheduled-action-name my-recurring-action ^
  --schedule "cron(0 18 ? * MON-FRI *)" --timezone "Etc/GMT+9" ^
  --scalable-target-action MinCapacity=10,MaxCapacity=50
```

Describe scheduled scaling for Application Auto Scaling using the AWS CLI

These example AWS CLI commands describe scaling activities and scheduled actions using resources from services that integrate with Application Auto Scaling. For a different scalable

target, specify its namespace in `--service-namespace`, its scalable dimension in `--scalable-dimension`, and its resource ID in `--resource-id`.

When using the AWS CLI, remember that your commands run in the AWS Region configured for your profile. If you want to run the commands in a different Region, either change the default Region for your profile, or use the `--region` parameter with the command.

Examples

- Describe scaling activities for a service
- Describe the scheduled actions for a service
- Describe the scheduled actions for a scalable target

Describe scaling activities for a service

To view the scaling activities for all of the scalable targets in a specified service namespace, use the describe-scaling-activities command.

The following example retrieves the scaling activities associated with the dynamodb service namespace.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-activities --service-namespace dynamodb
```

Windows

```
aws application-autoscaling describe-scaling-activities --service-namespace dynamodb
```

Output

If the command succeeds, it returns output similar to the following.

```
{
    "ScalingActivities": [
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting write capacity units to 10.",
            "ResourceId": "table/my-table",
            "ActivityId": "4d1308c0-bbcf-4514-a673-b0220ae38547",
            "StartTime": 1561574415.086,
```

```
            "ServiceNamespace": "dynamodb",
            "EndTime": 1561574449.51,
            "Cause": "maximum capacity was set to 10",
            "StatusMessage": "Successfully set write capacity units to 10. Change
successfully fulfilled by dynamodb.",
            "StatusCode": "Successful"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting min capacity to 5 and max capacity to 10",
            "ResourceId": "table/my-table",
            "ActivityId": "f2b7847b-721d-4e01-8ef0-0c8d3bacc1c7",
            "StartTime": 1561574414.644,
            "ServiceNamespace": "dynamodb",
            "Cause": "scheduled action name my-second-scheduled-action was triggered",
            "StatusMessage": "Successfully set min capacity to 5 and max capacity to
10",
            "StatusCode": "Successful"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting write capacity units to 15.",
            "ResourceId": "table/my-table",
            "ActivityId": "d8ea4de6-9eaa-499f-b466-2cc5e681ba8b",
            "StartTime": 1561574108.904,
            "ServiceNamespace": "dynamodb",
            "EndTime": 1561574140.255,
            "Cause": "minimum capacity was set to 15",
            "StatusMessage": "Successfully set write capacity units to 15. Change
successfully fulfilled by dynamodb.",
            "StatusCode": "Successful"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting min capacity to 15 and max capacity to 20",
            "ResourceId": "table/my-table",
            "ActivityId": "3250fd06-6940-4e8e-bb1f-d494db7554d2",
            "StartTime": 1561574108.512,
            "ServiceNamespace": "dynamodb",
            "Cause": "scheduled action name my-first-scheduled-action was triggered",
            "StatusMessage": "Successfully set min capacity to 15 and max capacity to
20",
            "StatusCode": "Successful"
        }
```

```
    ]
}
```

To change this command so that it retrieves the scaling activities for only one of your scalable targets, add the `--resource-id` option.

Describe the scheduled actions for a service

To describe the scheduled actions for all of the scalable targets in a specified service namespace, use the describe-scheduled-actions command.

The following example retrieves the scheduled actions associated with the `ec2` service namespace.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scheduled-actions --service-namespace ec2
```

Windows

```
aws application-autoscaling describe-scheduled-actions --service-namespace ec2
```

Output

If the command succeeds, it returns output similar to the following.

```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-one-time-action",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledAction:493a6261-fbb9-432d-855d-3c302c14bdb9:resource/ec2/
spot-fleet-request/sfr-107dc873-0802-4402-a901-37294EXAMPLE:scheduledActionName/my-one-
time-action",
            "ServiceNamespace": "ec2",
            "Schedule": "at(2021-01-31T17:00:00)",
            "Timezone": "America/New_York",
            "ResourceId": "spot-fleet-request/sfr-107dc873-0802-4402-
a901-37294EXAMPLE",
            "ScalableDimension": "ec2:spot-fleet-request:TargetCapacity",
            "ScalableTargetAction": {
                "MaxCapacity": 1
            },
```

```
            "CreationTime": 1607454792.331
        },
        {
            "ScheduledActionName": "my-recurring-action",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledAction:493a6261-fbb9-432d-855d-3c302c14bdb9:resource/ec2/
spot-fleet-request/sfr-107dc873-0802-4402-a901-37294EXAMPLE:scheduledActionName/my-
recurring-action",
            "ServiceNamespace": "ec2",
            "Schedule": "rate(5 minutes)",
            "ResourceId": "spot-fleet-request/sfr-107dc873-0802-4402-
a901-37294EXAMPLE",
            "ScalableDimension": "ec2:spot-fleet-request:TargetCapacity",
            "StartTime": 1604059200.0,
            "EndTime": 1612130400.0,
            "ScalableTargetAction": {
                "MinCapacity": 3,
                "MaxCapacity": 10
            },
            "CreationTime": 1607454949.719
        },
        {
            "ScheduledActionName": "my-one-time-action",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledAction:4bce34c7-bb81-4ecf-b776-5c726efb1567:resource/ec2/
spot-fleet-request/sfr-40edeb7b-9ae7-44be-bef2-5c4c8EXAMPLE:scheduledActionName/my-one-
time-action",
            "ServiceNamespace": "ec2",
            "Schedule": "at(2020-12-08T9:36:00)",
            "Timezone": "America/New_York",
            "ResourceId": "spot-fleet-request/sfr-40edeb7b-9ae7-44be-
bef2-5c4c8EXAMPLE",
            "ScalableDimension": "ec2:spot-fleet-request:TargetCapacity",
            "ScalableTargetAction": {
                "MinCapacity": 1,
                "MaxCapacity": 3
            },
            "CreationTime": 1607456031.391
        }
    ]
}
```

Describe the scheduled actions for a scalable target

To retrieve information about the scheduled actions for a specified scalable target, add the `--resource-id` option when describing scheduled actions using the describe-scheduled-actions command.

If you include the `--scheduled-action-names` option and specify the name of a scheduled action as its value, the command returns only the scheduled action whose name is a match, as shown in the following example.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scheduled-actions --service-namespace ec2 \
  --resource-id spot-fleet-request/sfr-40edeb7b-9ae7-44be-bef2-5c4c8EXAMPLE \
  --scheduled-action-names my-one-time-action
```

Windows

```
aws application-autoscaling describe-scheduled-actions --service-namespace ec2 ^
  --resource-id spot-fleet-request/sfr-40edeb7b-9ae7-44be-bef2-5c4c8EXAMPLE ^
  --scheduled-action-names my-one-time-action
```

Output

If the command succeeds, it returns output similar to the following. If you provided more than one value for `--scheduled-action-names`, the output includes all scheduled actions whose names are a match.

```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-one-time-action",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledAction:4bce34c7-bb81-4ecf-b776-5c726efb1567:resource/ec2/
spot-fleet-request/sfr-40edeb7b-9ae7-44be-bef2-5c4c8EXAMPLE:scheduledActionName/my-one-
time-action",
            "ServiceNamespace": "ec2",
            "Schedule": "at(2020-12-08T9:36:00)",
            "Timezone": "America/New_York",
            "ResourceId": "spot-fleet-request/sfr-40edeb7b-9ae7-44be-
bef2-5c4c8EXAMPLE",
```

```
            "ScalableDimension": "ec2:spot-fleet-request:TargetCapacity",
            "ScalableTargetAction": {
                "MinCapacity": 1,
                "MaxCapacity": 3
            },
            "CreationTime": 1607456031.391
        }
    ]
}
```

Schedule recurring scaling actions using Application Auto Scaling

> **Important**
>
> For help with cron expressions for Amazon EC2 Auto Scaling, see the Recurring schedules topic in the *Amazon EC2 Auto Scaling User Guide*. With Amazon EC2 Auto Scaling, you use traditional cron syntax instead of the custom cron syntax that Application Auto Scaling uses.

You can create scheduled actions that run on a recurring schedule using a cron expression.

To create a recurring schedule, specify a cron expression and a time zone to describe when that scheduled action is to recur. The supported time zone values are the canonical names of the IANA time zones supported by Joda-Time (such as `Etc/GMT+9` or `Pacific/Tahiti`). You can optionally specify a date and time for the start time, the end time, or both. For an example command that uses the AWS CLI to create a scheduled action, see Create a recurring scheduled action that specifies a time zone.

The supported cron expression format consists of six fields separated by white spaces: [Minutes] [Hours] [Day_of_Month] [Month] [Day_of_Week] [Year]. For example, the cron expression `30 6 ? * MON *` configures a scheduled action that recurs every Monday at 6:30 AM. The asterisk is used as a wildcard to match all values for a field.

For more information about the cron syntax for Application Auto Scaling scheduled actions, see Cron expressions reference in the *Amazon EventBridge User Guide*.

When you create a recurring schedule, choose your start and end times carefully. Keep the following in mind:

- If you specify a start time, Application Auto Scaling performs the action at this time, and then performs the action based on the specified recurrence.
- If you specify an end time, the action stops repeating after this time. Application Auto Scaling does not keep track of previous values and revert back to those previous values after the end time.
- The start time and end time must be set in UTC when you use the AWS CLI or the AWS SDKs to create or update a scheduled action.

Examples

You can refer to the following table when you create a recurring schedule for an Application Auto Scaling scalable target. The following examples are the correct syntax for using Application Auto Scaling to create or update a scheduled action.

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0	10	*	*	?	*	Run at 10:00 am (UTC) every day
15	12	*	*	?	*	Run at 12:15 pm (UTC) every day
0	18	?	*	MON-FRI	*	Run at 6:00 pm (UTC) every Monday through Friday

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0	8	1	*	?	*	Run at 8:00 am (UTC) the 1st day of every month
0/15	*	*	*	?	*	Run every 15 minutes
0/10	*	?	*	MON-FRI	*	Run every 10 minutes Monday through Friday
0/5	8-17	?	*	MON-FRI	*	Run every 5 minutes Monday through Friday between 8:00 am and 5:55 pm (UTC)

Exception

You can also create a cron expression with a string value that contains seven fields. In this case, you can use the first three fields to specify the time for when a scheduled action should be run, including the seconds. The full cron expression has the following space-separated fields: [Seconds] [Minutes] [Hours] [Day_of_Month] [Month] [Day_of_Week] [Year]. However, this approach doesn't guarantee that the scheduled action will run on the precise second that you specify. Also, some service consoles may not support the seconds field in a cron expression.

Turn off scheduled scaling for a scalable target

You can temporarily turn off scheduled scaling without deleting your scheduled actions. For more information, see Suspend and resume scaling for Application Auto Scaling.

To suspend scheduled scaling

Suspend scheduled scaling on a scalable target by using the register-scalable-target command with the `--suspended-state` option, and specifying `true` as the value of the `ScheduledScalingSuspended` attribute, as shown in the following example.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target --service-namespace rds \
  --scalable-dimension rds:cluster:ReadReplicaCount --resource-id cluster:my-db-cluster
 \
  --suspended-state '{"ScheduledScalingSuspended": true}'
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace rds ^
  --scalable-dimension rds:cluster:ReadReplicaCount --resource-id cluster:my-db-cluster
 ^
  --suspended-state "{\"ScheduledScalingSuspended\": true}"
```

Output

If the command succeeds, it returns the ARN of the scalable target. The following is example output.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

To resume scheduled scaling

To resume scheduled scaling, run the **register-scalable-target** command again, specifying `false` as the value for `ScheduledScalingSuspended`.

Delete a scheduled action for Application Auto Scaling using the AWS CLI

When you are finished with a scheduled action, you can delete it.

To delete your scheduled action

Use the delete-scheduled-action command. If successful, this command does not return any output.

Linux, macOS, or Unix

```
aws application-autoscaling delete-scheduled-action \
  --service-namespace ec2 \
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity \
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-37294EXAMPLE \
  --scheduled-action-name my-recurring-action
```

Windows

```
aws application-autoscaling delete-scheduled-action ^
  --service-namespace ec2 ^
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity ^
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-37294EXAMPLE ^
  --scheduled-action-name my-recurring-action
```

To deregister the scalable target

If you are also finished with the scalable target, you can deregister it. Use the following deregister-scalable-target command. If there are any scaling policies or scheduled actions that have not yet been deleted, they are deleted by this command. If successful, this command does not return any output.

Linux, macOS, or Unix

```
aws application-autoscaling deregister-scalable-target \
  --service-namespace ec2 \
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity \
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-37294EXAMPLE
```

Windows

```
aws application-autoscaling deregister-scalable-target ^
  --service-namespace ec2 ^
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity ^
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-37294EXAMPLE
```

Target tracking scaling policies for Application Auto Scaling

A target tracking scaling policy automatically scales your application based on a target metric value. This allows your application to maintain optimal performance and cost efficiency without manual intervention.

With target tracking, you select a metric and a target value to represent the ideal average utilization or throughput level for your application. Application Auto Scaling creates and manages the CloudWatch alarms that trigger scaling events when the metric deviates from the target. This is similar to how a thermostat maintains a target temperature.

For example, let's say that you currently have an application that runs on Spot Fleet, and you want the CPU utilization of the fleet to stay at around 50 percent when the load on the application changes. This gives you extra capacity to handle traffic spikes without maintaining an excessive number of idle resources.

You can meet this need by creating a target tracking scaling policy that targets an average CPU utilization of 50 percent. Then, Application Auto Scaling will scale out (increase capacity) when CPU exceeds 50 percent to handle increased load. It will scale in (decrease capacity) when CPU drops below 50 percent to optimize costs during periods of low utilization.

Target tracking policies remove the need to manually define CloudWatch alarms and scaling adjustments. Application Auto Scaling handles this automatically based on the target you set.

You can base target tracking policies on either predefined or custom metrics:

- **Predefined metrics**—Metrics provided by Application Auto Scaling like average CPU utilization or average request count per target.
- **Custom metrics**—You can use metric math to combine metrics, leverage existing metrics, or use your own custom metrics published to CloudWatch.

Choose a metric that changes inversely proportional to a change in the capacity of your scalable target. So if you double capacity, the metric decreases by 50 percent. This allows the metric data to accurately trigger proportional scaling events.

Contents

- How target tracking scaling for Application Auto Scaling works
- Create a target tracking scaling policy for Application Auto Scaling using the AWS CLI
- Delete a target tracking scaling policy for Application Auto Scaling using the AWS CLI
- Create a target tracking scaling policy for Application Auto Scaling using metric math

How target tracking scaling for Application Auto Scaling works

This topic describes how target tracking scaling works and introduces the key elements of a target tracking scaling policy.

Contents

- How it works
- Choose metrics
- Define target value
- Define cooldown periods
- Considerations
- Multiple scaling policies
- Commonly used commands for scaling policy creation, management, and deletion
- Related resources
- Limitations

How it works

To use target tracking scaling, you create a target tracking scaling policy and specify the following:

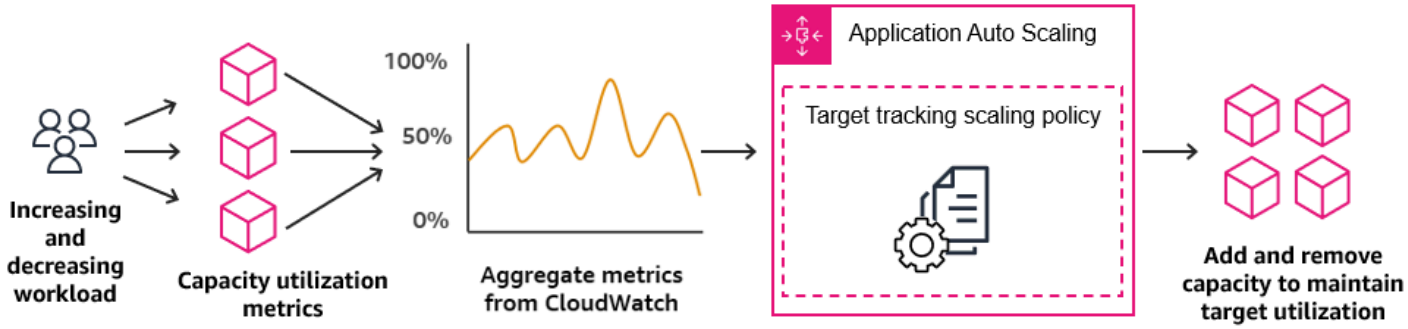
- **Metric**—A CloudWatch metric to track, such as average CPU utilization or average request count per target.
- **Target value**—The target value for the metric, such as 50 percent CPU utilization or 1000 requests per target per minute.

Application Auto Scaling creates and manages the CloudWatch alarms that invoke the scaling policy and calculates the scaling adjustment based on the metric and the target value. It adds and removes capacity as required to keep the metric at, or close to, the specified target value.

When the metric is above the target value, Application Auto Scaling scales out by adding capacity to reduce the difference between the metric value and the target value. When the metric is below the target value, Application Auto Scaling scales in by removing capacity.

Scaling activities are performed with cooldown periods between them to prevent rapid fluctuations in capacity. You can optionally configure the cooldown periods for your scaling policy.

The following diagram shows an overview of how a target tracking scaling policy works when the set up is complete.



Note that a target tracking scaling policy is more aggressive in adding capacity when utilization increases than it is in removing capacity when utilization decreases. For example, if the policy's specified metric reaches its target value, the policy assumes that your application is already heavily loaded. So it responds by adding capacity proportional to the metric value as fast as it can. The higher the metric, the more capacity is added.

When the metric falls below the target value, the policy won't scale-in if it calculates that removing a minimum unit of capacity would likely bring the metric back above the target value. In this case, it slows down scaling by removing capacity only when utilization passes a threshold that is far enough below the target value (usually more than 10% lower) for utilization to be considered to have slowed. The intention of this more conservative behavior is to ensure that removing capacity only happens when the application is no longer experiencing demand at the same high level that it was previously.

Choose metrics

You can create target tracking scaling policies with either predefined metrics or custom metrics.

When you create a target tracking scaling policy with a predefined metric type, you choose one metric from the list of predefined metrics in Predefined metrics for target tracking scaling policies.

Keep the following in mind when choosing a metric:

- Not all custom metrics work for target tracking. The metric must be a valid utilization metric and describe how busy a scalable target is. The metric value must increase or decrease proportionally to the capacity of the scalable target so that the metric data can be used to proportionally scale the scalable target.
- To use the `ALBRequestCountPerTarget` metric, you must specify the `ResourceLabel` parameter to identify the target group that is associated with the metric.
- When a metric emits real 0 values to CloudWatch (for example, `ALBRequestCountPerTarget`), Application Auto Scaling can scale in to 0 when there is no traffic to your application for a sustained period of time. To have your scalable target scale in to 0 when no requests are routed it, the scalable target's minimum capacity must be set to 0.
- Instead of publishing new metrics to use in your scaling policy, you can use metric math to combine existing metrics. For more information, see Create a target tracking scaling policy for Application Auto Scaling using metric math.
- To see whether the service you are using supports specifying a custom metric in the service's console, consult the documentation for that service.
- We recommend that you use metrics that are available at one-minute intervals to help you scale faster in response to utilization changes. Target tracking will evaluate metrics aggregated at a one-minute granularity for all predefined metrics and custom metrics, but the underlying metric might publish data less frequently. For example, all Amazon EC2 metrics are sent in five-minute intervals by default, but they are configurable to one minute (known as detailed monitoring). This choice is up to the individual services. Most try to use the smallest interval possible.

Define target value

When you create a target tracking scaling policy, you must specify a target value. The target value represents the optimal average utilization or throughput for your application. To use resources cost efficiently, set the target value as high as possible with a reasonable buffer for unexpected traffic increases. When your application is optimally scaled out for a normal traffic flow, the actual metric value should be at or just below the target value.

When a scaling policy is based on throughput, such as the request count per target for an Application Load Balancer, network I/O, or other count metrics, the target value represents the optimal average throughput from a single entity (such as a single target of your Application Load Balancer target group), for a one-minute period.

Define cooldown periods

You can optionally define cooldown periods in your target tracking scaling policy.

A cooldown period specifies the amount of time the scaling policy waits for a previous scaling activity to take effect.

There are two types of cooldown periods:

- With the *scale-out cooldown period*, the intention is to continuously (but not excessively) scale out. After Application Auto Scaling successfully scales out using a scaling policy, it starts to calculate the cooldown time. A scaling policy won't increase the desired capacity again unless either a larger scale out is triggered or the cooldown period ends. While the scale-out cooldown period is in effect, the capacity added by the initiating scale-out activity is calculated as part of the desired capacity for the next scale-out activity.
- With the *scale-in cooldown period*, the intention is to scale in conservatively to protect your application's availability, so scale-in activities are blocked until the scale-in cooldown period has expired. However, if another alarm triggers a scale-out activity during the scale-in cooldown period, Application Auto Scaling scales out the target immediately. In this case, the scale-in cooldown period stops and doesn't complete.

Each cooldown period is measured in seconds and applies only to scaling policy-related scaling activities. During a cooldown period, when a scheduled action starts at the scheduled time, it can trigger a scaling activity immediately without waiting for the cooldown period to expire.

You can start with the default values, which can be later fine-tuned. For example, you might need to increase a cooldown period to prevent your target tracking scaling policy from being too aggressive about changes that occur over short periods of time.

Default values

Application Auto Scaling provides a default value of 600 for ElastiCache and a default value of 300 for the following scalable targets:

- WorkSpaces Applications fleets
- Aurora DB clusters
- ECS services
- Neptune clusters

- SageMaker AI endpoint variants
- SageMaker AI inference components
- SageMaker AI Serverless provisioned concurrency
- Spot Fleets
- Pool of WorkSpaces
- Custom resources

For all other scalable targets, the default value is 0 or null:

- Amazon Comprehend document classification and entity recognizer endpoints
- DynamoDB tables and global secondary indexes
- Amazon Keyspaces tables
- Lambda provisioned concurrency
- Amazon MSK broker storage

Null values are treated the same as zero values when Application Auto Scaling evaluates the cooldown period.

You can update any of the default values, including null values, to set your own cooldown periods.

Considerations

The following considerations apply when working with target tracking scaling policies:

- Do not create, edit, or delete the CloudWatch alarms that are used with a target tracking scaling policy. Application Auto Scaling creates and manages the CloudWatch alarms that are associated with your target tracking scaling policies and deletes them when no longer needed.
- If the metric is missing data points, this causes the CloudWatch alarm state to change to `INSUFFICIENT_DATA`. When this happens, Application Auto Scaling cannot scale your scalable target until new data points are found. For more information, see Configuring how CloudWatch alarms treat missing data in the *Amazon CloudWatch User Guide*.
- If the metric is sparsely reported by design, metric math can be helpful. For example, to use the most recent values, then use the `FILL(m1,REPEAT)` function where `m1` is the metric.
- You may see gaps between the target value and the actual metric data points. This is because Application Auto Scaling always acts conservatively by rounding up or down when it determines

how much capacity to add or remove. This prevents it from adding insufficient capacity or removing too much capacity. However, for a scalable target with a small capacity, the actual metric data points might seem far from the target value.

For example, suppose that you set a target value of 50 percent for CPU utilization and your Auto Scaling group then exceeds the target. We might determine that adding 1.5 instances will decrease the CPU utilization to close to 50 percent. Because it is not possible to add 1.5 instances, we round up and add two instances. This might decrease the CPU utilization to a value below 50 percent, but it ensures that your application has enough resources to support it. Similarly, if we determine that removing 0.5 instances increases your CPU utilization to above 50 percent, we will choose not to scale-in until the metric lowers enough that we think scaling in won't cause oscillation.

For a scalable target with a larger capacity, adding or removing capacity causes less of a gap between the target value and the actual metric data points.

- A target tracking scaling policy assumes that it should perform scale out when the specified metric is above the target value. You cannot use a target tracking scaling policy to scale out when the specified metric is below the target value.

Multiple scaling policies

You can have multiple target tracking scaling policies for a scalable target, provided that each of them uses a different metric. The intention of Application Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the scalable target if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion enabled) are ready to scale in.

If multiple scaling policies instruct the scalable target to scale out or in at the same time, Application Auto Scaling scales based on the policy that provides the largest capacity for both scale in and scale out. This provides greater flexibility to cover multiple scenarios and ensures that there is always enough capacity to process your workloads.

You can disable the scale-in portion of a target tracking scaling policy to use a different method for scale in than you use for scale out. For example, you can use a step scaling policy for scale in while using a target tracking scaling policy for scale out.

We recommend caution, however, when using target tracking scaling policies with step scaling policies because conflicts between these policies can cause undesirable behavior. For example, if the step scaling policy initiates a scale-in activity before the target tracking policy is ready to scale in, the scale-in activity will not be blocked. After the scale-in activity completes, the target tracking policy could instruct the scalable target to scale out again.

For workloads that are cyclical in nature, you also have the option to automate capacity changes on a schedule using scheduled scaling. For each scheduled action, a new minimum capacity value and a new maximum capacity value can be defined. These values form the boundaries of the scaling policy. The combination of scheduled scaling and target tracking scaling can help reduce the impact of a sharp increase in utilization levels, when capacity is needed immediately.

Commonly used commands for scaling policy creation, management, and deletion

The commonly used commands for working with scaling policies include:

- register-scalable-target to register AWS or custom resources as scalable targets (a resource that Application Auto Scaling can scale), and to suspend and resume scaling.
- put-scaling-policy to add or modify scaling policies for an existing scalable target.
- describe-scaling-activities to return information about scaling activities in an AWS Region.
- describe-scaling-policies to return information about scaling policies in an AWS Region.
- delete-scaling-policy to delete a scaling-policy.

Related resources

For information about creating target tracking scaling policies for Auto Scaling groups, see Target tracking scaling policies for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Limitations

The following are limitations when using target tracking scaling policies:

- The scalable target can't be an Amazon EMR cluster. Target tracking scaling policies are not supported for Amazon EMR.
- When an Amazon MSK cluster is the scalable target, scale in is disabled and cannot be enabled.

- You cannot use the `RegisterScalableTarget` or `PutScalingPolicy` API operations to update an AWS Auto Scaling scaling plan.
- Console access to view, add, update, or remove target tracking scaling policies on scalable resources depends on the resource that you use. For more information, see AWS services that you can use with Application Auto Scaling.

Create a target tracking scaling policy for Application Auto Scaling using the AWS CLI

This example uses AWS CLI commands to create a target racking policy for an Amazon EC2 Spot Fleet. For a different scalable target, specify its namespace in `--service-namespace`, its scalable dimension in `--scalable-dimension`, and its resource ID in `--resource-id`.

When using the AWS CLI, remember that your commands run in the AWS Region configured for your profile. If you want to run the commands in a different Region, either change the default Region for your profile, or use the `--region` parameter with the command.

Tasks

- Step 1: Register a scalable target
- Step 2: Create a target tracking scaling policy
- Step 3: Describe target tracking scaling policies

Step 1: Register a scalable target

If you haven't already done so, register the scalable target. Use the register-scalable-target command to register a specific resource in the target service as a scalable target. The following example registers a Spot Fleet request with Application Auto Scaling. Application Auto Scaling can scale the number of instances in the Spot Fleet at a minimum of 2 instances and a maximum of 10. Replace each *`user input placeholder`* with your own information.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target --service-namespace ec2 \
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity \
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE \
  --min-capacity 2 --max-capacity 10
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace ec2 ^
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity ^
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE ^
  --min-capacity 2 --max-capacity 10
```

Output

If successful, this command returns the ARN of the scalable target. The following is example output.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Step 2: Create a target tracking scaling policy

To create a target tracking scaling policy, you can use the following examples to help you get started.

To create a target tracking scaling policy

1. Use the following `cat` command to store a target value for your scaling policy and a predefined metric specification in a JSON file named `config.json` in your home directory. The following is an example target tracking configuration that keeps the average CPU utilization at 50 percent.

   ```
   $ cat ~/config.json
   {
     "TargetValue": 50.0,
     "PredefinedMetricSpecification":
       {
         "PredefinedMetricType": "EC2SpotFleetRequestAverageCPUUtilization"
       }
   }
   ```

   For more information, see PredefinedMetricSpecification in the *Application Auto Scaling API Reference*.

Alternatively, you can use a custom metric for scaling by creating a customized metric specification and adding values for each parameter from CloudWatch. The following is an example target tracking configuration that keeps the average utilization of the specified metric at 100.

```
$ cat ~/config.json
{
   "TargetValue": 100.0,
   "CustomizedMetricSpecification":{
      "MetricName": "MyUtilizationMetric",
      "Namespace": "MyNamespace",
      "Dimensions": [
         {
            "Name": "MyOptionalMetricDimensionName",
            "Value": "MyOptionalMetricDimensionValue"
         }
      ],
      "Statistic": "Average",
      "Unit": "Percent"
   }
}
```

For more information, see CustomizedMetricSpecification in the *Application Auto Scaling API Reference*.

2. Use the following put-scaling-policy command, along with the `config.json` file you created, to create a scaling policy named `cpu50-target-tracking-scaling-policy`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scaling-policy --service-namespace ec2 \
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity \
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE \
  --policy-name cpu50-target-tracking-scaling-policy --policy-type
 TargetTrackingScaling \
  --target-tracking-scaling-policy-configuration file://config.json
```

Windows

```
aws application-autoscaling put-scaling-policy --service-namespace ec2 ^
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity ^
```

```
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE ^
  --policy-name cpu50-target-tracking-scaling-policy --policy-type
 TargetTrackingScaling ^
  --target-tracking-scaling-policy-configuration file://config.json
```

Output

If successful, this command returns the ARNs and names of the two CloudWatch alarms created on your behalf. The following is example output.

```
{
    "PolicyARN": "arn:aws:autoscaling:region:account-
id:scalingPolicy:policy-id:resource/ec2/spot-fleet-request/sfr-73fbd2ce-
aa30-494c-8788-1cee4EXAMPLE:policyName/cpu50-target-tracking-scaling-policy",
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:region:account-id:alarm:TargetTracking-
spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE-AlarmHigh-d4f0770c-
b46e-434a-a60f-3b36d653feca",
            "AlarmName": "TargetTracking-spot-fleet-request/sfr-73fbd2ce-
aa30-494c-8788-1cee4EXAMPLE-AlarmHigh-d4f0770c-b46e-434a-a60f-3b36d653feca"
        },
        {
            "AlarmARN": "arn:aws:cloudwatch:region:account-id:alarm:TargetTracking-
spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE-AlarmLow-1b437334-
d19b-4a63-a812-6c67aaf2910d",
            "AlarmName": "TargetTracking-spot-fleet-request/sfr-73fbd2ce-
aa30-494c-8788-1cee4EXAMPLE-AlarmLow-1b437334-d19b-4a63-a812-6c67aaf2910d"
        }
    ]
}
```

Step 3: Describe target tracking scaling policies

You can describe all scaling policies for the specified service namespace using the following describe-scaling-policies command.

```
aws application-autoscaling describe-scaling-policies --service-namespace ec2
```

You can filter the results to just the target tracking scaling policies using the `--query` parameter. For more information about the syntax for `query`, see Controlling command output from the AWS CLI in the *AWS Command Line Interface User Guide*.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-policies --service-namespace ec2 \
  --query 'ScalingPolicies[?PolicyType==`TargetTrackingScaling`]'
```

Windows

```
aws application-autoscaling describe-scaling-policies --service-namespace ec2 ^
  --query "ScalingPolicies[?PolicyType==`TargetTrackingScaling`]"
```

Output

The following is example output.

```
[
    {
        "PolicyARN": "PolicyARN",
        "TargetTrackingScalingPolicyConfiguration": {
            "PredefinedMetricSpecification": {
                "PredefinedMetricType": "EC2SpotFleetRequestAverageCPUUtilization"
            },
            "TargetValue": 50.0
        },
        "PolicyName": "cpu50-target-tracking-scaling-policy",
        "ScalableDimension": "ec2:spot-fleet-request:TargetCapacity",
        "ServiceNamespace": "ec2",
        "PolicyType": "TargetTrackingScaling",
        "ResourceId": "spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE",
        "Alarms": [
            {
                "AlarmARN": "arn:aws:cloudwatch:region:account-id:alarm:TargetTracking-
spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE-AlarmHigh-d4f0770c-
b46e-434a-a60f-3b36d653feca",
                "AlarmName": "TargetTracking-spot-fleet-request/sfr-73fbd2ce-
aa30-494c-8788-1cee4EXAMPLE-AlarmHigh-d4f0770c-b46e-434a-a60f-3b36d653feca"
            },
            {
```

```
                "AlarmARN": "arn:aws:cloudwatch:region:account-id:alarm:TargetTracking-
spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE-AlarmLow-1b437334-
d19b-4a63-a812-6c67aaf2910d",
                "AlarmName": "TargetTracking-spot-fleet-request/sfr-73fbd2ce-
aa30-494c-8788-1cee4EXAMPLE-AlarmLow-1b437334-d19b-4a63-a812-6c67aaf2910d"
            }
        ],
        "CreationTime": 1515021724.807
    }
]
```

Delete a target tracking scaling policy for Application Auto Scaling using the AWS CLI

When you are finished with a target tracking scaling policy, you can delete it using the delete-scaling-policy command.

The following command deletes the specified target tracking scaling policy for the specified Spot Fleet request. It also deletes the CloudWatch alarms that Application Auto Scaling created on your behalf.

Linux, macOS, or Unix

```
aws application-autoscaling delete-scaling-policy --service-namespace ec2 \
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity \
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE \
  --policy-name cpu50-target-tracking-scaling-policy
```

Windows

```
aws application-autoscaling delete-scaling-policy --service-namespace ec2 ^
  --scalable-dimension ec2:spot-fleet-request:TargetCapacity ^
  --resource-id spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE ^
  --policy-name cpu50-target-tracking-scaling-policy
```

Create a target tracking scaling policy for Application Auto Scaling using metric math

Using metric math, you can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series in the CloudWatch console and add them to dashboards. For more information about metric math, see Using metric math in the *Amazon CloudWatch User Guide*.

The following considerations apply to metric math expressions:

- You can query any available CloudWatch metric. Each metric is a unique combination of metric name, namespace, and zero or more dimensions.
- You can use any arithmetic operator (+ - * / ^), statistical function (such as AVG or SUM), or other function that CloudWatch supports.
- You can use both metrics and the results of other math expressions in the formulas of the math expression.
- Any expressions used in a metric specification must eventually return a single time series.
- You can verify that a metric math expression is valid by using the CloudWatch console or the CloudWatch GetMetricData API.

Topics

- Example: Amazon SQS queue backlog per task
- Limitations

Example: Amazon SQS queue backlog per task

To calculate the Amazon SQS queue backlog per task, take the approximate number of messages available for retrieval from the queue and divide that number by the number of Amazon ECS tasks running in the service. For more information, see Amazon Elastic Container Service (ECS) Auto Scaling using custom metrics on the AWS Compute Blog.

The logic for the expression is this:

```
sum of (number of messages in the queue)/(number of tasks that are
currently in the RUNNING state)
```

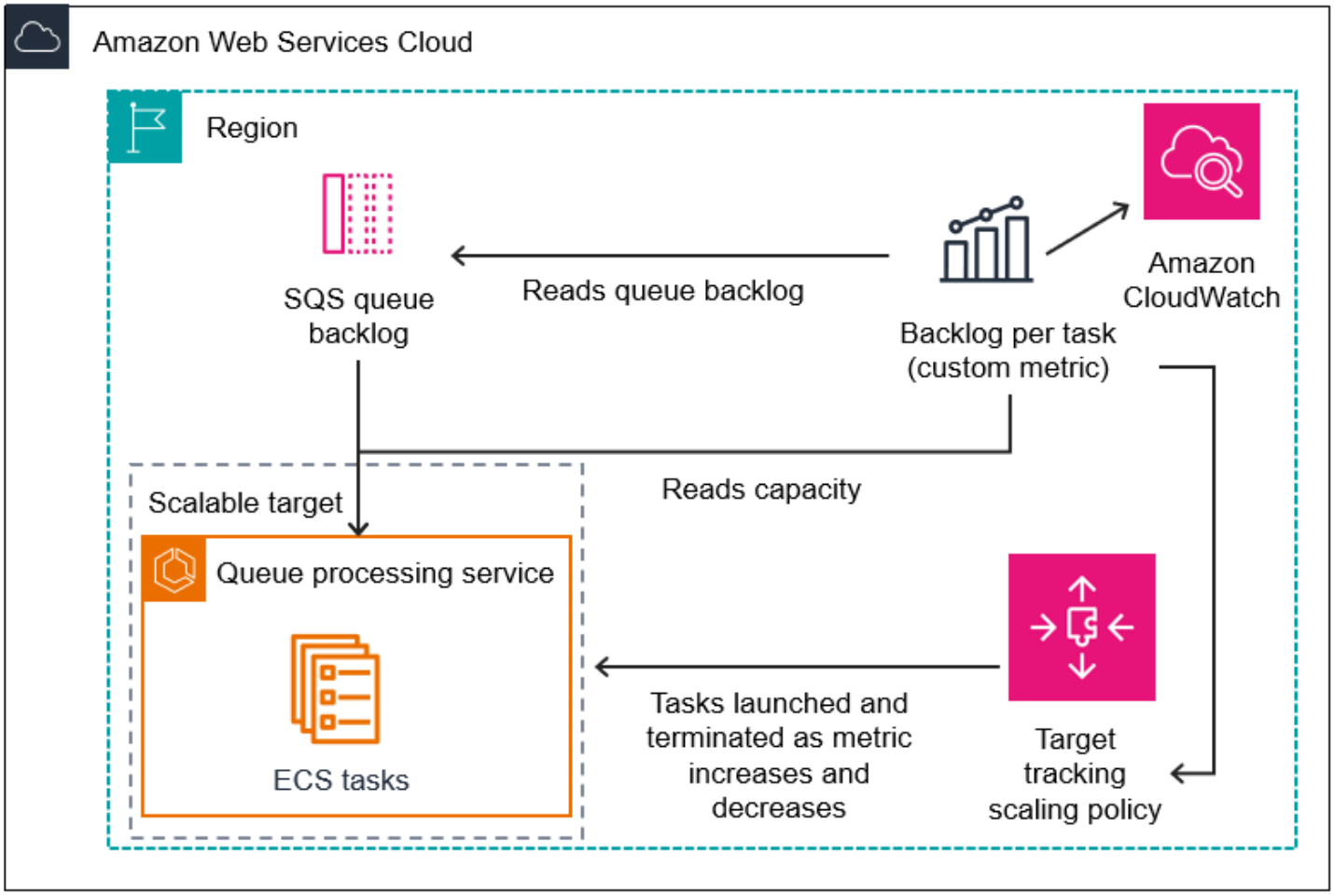
Then your CloudWatch metric information is the following.

ID	CloudWatch metric	Statistic	Period
m1	ApproximateNumberOfMessagesVisible	Sum	1 minute
m2	RunningTaskCount	Average	1 minute

Your metric math ID and expression are the following.

ID	Expression
e1	`(m1)/(m2)`

The following diagram illustrates the architecture for this metric:



To use this metric math to create a target tracking scaling policy (AWS CLI)

1. Store the metric math expression as part of a customized metric specification in a JSON file named `config.json`.

   Use the following example to help you get started. Replace each *`user input placeholder`* with your own information.

```
{
    "CustomizedMetricSpecification": {
        "Metrics": [
            {
                "Label": "Get the queue size (the number of messages waiting to be
 processed)",
                "Id": "m1",
                "MetricStat": {
                    "Metric": {
```

```
                        "MetricName": "ApproximateNumberOfMessagesVisible",
                        "Namespace": "AWS/SQS",
                        "Dimensions": [
                            {
                                "Name": "QueueName",
                                "Value": "my-queue"
                            }
                        ]
                    },
                    "Stat": "Sum"
                },
                "ReturnData": false
            },
            {
                "Label": "Get the ECS running task count (the number of currently
running tasks)",
                "Id": "m2",
                "MetricStat": {
                    "Metric": {
                        "MetricName": "RunningTaskCount",
                        "Namespace": "ECS/ContainerInsights",
                        "Dimensions": [
                            {
                                "Name": "ClusterName",
                                "Value": "my-cluster"
                            },
                            {
                                "Name": "ServiceName",
                                "Value": "my-service"
                            }
                        ]
                    },
                    "Stat": "Average"
                },
                "ReturnData": false
            },
            {
                "Label": "Calculate the backlog per instance",
                "Id": "e1",
                "Expression": "m1 / m2",
                "ReturnData": true
            }
        ]
    },
```

```
    "TargetValue": 100
}
```

For more information, see TargetTrackingScalingPolicyConfiguration in the *Application Auto Scaling API Reference*.

> **Note**
>
> Following are some additional resources that can help you find metric names, namespaces, dimensions, and statistics for CloudWatch metrics:
>
> - For information about the available metrics for AWS services, see AWS services that publish CloudWatch metrics in the *Amazon CloudWatch User Guide*.
> - To get the exact metric name, namespace, and dimensions (if applicable) for a CloudWatch metric with the AWS CLI, see list-metrics.

2. To create this policy, run the put-scaling-policy command using the JSON file as input, as demonstrated in the following example.

```
aws application-autoscaling put-scaling-policy --policy-name sqs-backlog-target-tracking-scaling-policy \
  --service-namespace ecs --scalable-dimension ecs:service:DesiredCount --resource-id service/my-cluster/my-service \
  --policy-type TargetTrackingScaling --target-tracking-scaling-policy-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN) and the ARNs of the two CloudWatch alarms created on your behalf.

```
{
    "PolicyARN": "arn:aws:autoscaling:us-west-2:012345678910:scalingPolicy:
 8784a896-b2ba-47a1-b08c-27301cc499a1:resource/ecs/service/my-cluster/my-service:policyName/sqs-backlog-target-tracking-scaling-policy",
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-service/my-cluster/my-service-AlarmHigh-9bc77b56-0571-4276-ba0f-d4178882e0a0",
            "AlarmName": "TargetTracking-service/my-cluster/my-service-AlarmHigh-9bc77b56-0571-4276-ba0f-d4178882e0a0"
```

```
        },
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-service/my-cluster/my-service-
AlarmLow-9b6ad934-6d37-438e-9e05-02836ddcbdc4",
            "AlarmName": "TargetTracking-service/my-cluster/my-service-
AlarmLow-9b6ad934-6d37-438e-9e05-02836ddcbdc4"
        }
    ]
}
```

> **Note**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Limitations

- The maximum request size is 50 KB. This is the total payload size for the PutScalingPolicy API request when you use metric math in the policy definition. If you exceed this limit, Application Auto Scaling rejects the request.
- The following services are not supported when using metric math with target tracking scaling policies:
  - Amazon Keyspaces (for Apache Cassandra)
  - DynamoDB
  - Amazon EMR
  - Amazon MSK
  - Amazon Neptune

Step scaling policies for Application Auto Scaling

A step scaling policy scales your application's capacity in predefined increments based on CloudWatch alarms. You can define separate scaling policies to handle scaling out (increasing capacity) and scaling in (decreasing capacity) when an alarm threshold is breached.

With step scaling policies, you create and manage the CloudWatch alarms that invoke the scaling process. When an alarm is breached, Application Auto Scaling initiates the scaling policy associated with that alarm.

The step scaling policy scales capacity using a set of adjustments, known as *step adjustments*. The size of the adjustment varies based on the magnitude of the alarm breach.

- If the breach exceeds the first threshold, Application Auto Scaling will apply the first step adjustment.
- If the breach exceeds the second threshold, Application Auto Scaling will apply the second step adjustment, and so on.

This allows the scaling policy to respond appropriately to both minor and major changes in the alarm metric.

The policy will continue to respond to additional alarm breaches, even while a scaling activity is in progress. This means Application Auto Scaling will evaluate all alarm breaches as they occur. A cooldown period is used to protect against over-scaling due to multiple alarm breaches occurring in rapid succession.

Like target tracking, step scaling can help automatically scale your application's capacity as traffic changes occur. However, target tracking policies tend to be easier to implement and manage for steady scaling needs.

Supported scalable targets

You can use step scaling policies with the following scalable targets:

- WorkSpaces Applications fleets
- Aurora DB clusters
- ECS services

- EMR clusters
- SageMaker AI endpoint variants
- SageMaker AI inference components
- SageMaker AI Serverless provisioned concurrency
- Spot Fleets
- Custom resources

Contents

- How step scaling for Application Auto Scaling works
- Create a step scaling policy for Application Auto Scaling using the AWS CLI
- Describe step scaling policies for Application Auto Scaling using the AWS CLI
- Delete a step scaling policy for Application Auto Scaling using the AWS CLI

How step scaling for Application Auto Scaling works

This topic describes how step scaling works and introduces the key elements of a step scaling policy.

Contents

- How it works
- Step adjustments
- Scaling adjustment types
- Cooldown period
- Commonly used commands for scaling policy creation, management, and deletion
- Considerations
- Related resources
- Console access

How it works

To use step scaling, you create a CloudWatch alarm that monitors a metric for your scalable target. Define the metric, threshold value, and number of evaluation periods that determine an alarm

breach. You also create a step scaling policy that defines how to scale capacity when the alarm threshold is breached and associate it with your scalable target.

Add the step adjustments in the policy. You can define different step adjustments based on the breach size of the alarm. For example:

- Scale out by 10 capacity units if the alarm metric reaches 60 percent
- Scale out by 30 capacity units if the alarm metric reaches 75 percent
- Scale out by 40 capacity units if the alarm metric reaches 85 percent

When the alarm threshold is breached for the specified number of evaluation periods, Application Auto Scaling will apply the step adjustments defined in the policy. The adjustments can continue for additional alarm breaches until the alarm state returns to OK.

Scaling activities are performed with cooldown periods between them to prevent rapid fluctuations in capacity. You can optionally configure the cooldown periods for your scaling policy.

Step adjustments

When you create a step scaling policy, you specify one or more step adjustments that automatically scale the capacity of the target dynamically based on the size of the alarm breach. Each step adjustment specifies the following:

- A lower bound for the metric value
- An upper bound for the metric value
- The amount by which to scale, based on the scaling adjustment type

CloudWatch aggregates metric data points based on the statistic for the metric associated with your CloudWatch alarm. When the alarm is breached, the appropriate scaling policy is invoked. Application Auto Scaling applies your specified aggregation type to the most recent metric data points from CloudWatch (as opposed to the raw metric data). It compares this aggregated metric value against the upper and lower bounds defined by the step adjustments to determine which step adjustment to perform.

You specify the upper and lower bounds relative to the breach threshold. For example, let's say you made a CloudWatch alarm and a scale-out policy for when the metric is above 50 percent. You then made a second alarm and a scale-in policy for when the metric is below 50 percent. You made a set of step adjustments with an adjustment type of `PercentChangeInCapacity` for each policy:

Example: Step adjustments for scale-out policy

Lower bound	Upper bound	Adjustment
0	10	0
10	20	10
20	null	30

Example: Step adjustments for scale-in policy

Lower bound	Upper bound	Adjustment
-10	0	0
-20	-10	-10
null	-20	-30

This creates the following scaling configuration.

```
Metric value

-infinity          30%    40%          60%     70%             infinity
-------------------------------------------------------------------------
          -30%      | -10% | Unchanged   | +10%  |       +30%
-------------------------------------------------------------------------
```

Now, let's say that you use this scaling configuration on a scalable target that has a capacity of 10. The following points summarize the behavior of the scaling configuration in relation to the capacity of the scalable target:

- The original capacity is maintained while the aggregated metric value is greater than 40 and less than 60.
- If the metric value gets to 60, Application Auto Scaling increases the capacity of the scalable target by 1, to 11. That's based on the second step adjustment of the scale-out policy (add 10 percent of 10). After the new capacity is added, Application Auto Scaling increases the current capacity to 11. If the metric value rises to 70 even after this increase in capacity, Application Auto

Scaling increases the target capacity by 3, to 14. That's based on the third step adjustment of the scale-out policy (add 30 percent of 11, 3.3, rounded down to 3).

- If the metric value gets to 40, Application Auto Scaling decreases the capacity of the scalable target by 1, to 13, based on the second step adjustment of the scale-in policy (remove 10 percent of 14, 1.4, rounded down to 1). If the metric value falls to 30 even after this decrease in capacity, Application Auto Scaling decreases the target capacity by 3, to 10, based on the third step adjustment of the scale-in policy (remove 30 percent of 13, 3.9, rounded down to 3).

When you specify the step adjustments for your scaling policy, note the following:

- The ranges of your step adjustments can't overlap or have a gap.
- Only one step adjustment can have a null lower bound (negative infinity). If one step adjustment has a negative lower bound, then there must be a step adjustment with a null lower bound.
- Only one step adjustment can have a null upper bound (positive infinity). If one step adjustment has a positive upper bound, then there must be a step adjustment with a null upper bound.
- The upper and lower bound can't be null in the same step adjustment.
- If the metric value is above the breach threshold, the lower bound is inclusive and the upper bound is exclusive. If the metric value is below the breach threshold, the lower bound is exclusive and the upper bound is inclusive.

Scaling adjustment types

You can define a scaling policy that performs the optimal scaling action, based on the scaling adjustment type that you choose. You can specify the adjustment type as a percentage of the current capacity of your scalable target or in absolute numbers.

Application Auto Scaling supports the following adjustment types for step scaling policies:

- **ChangeInCapacity**—Increase or decrease the current capacity of the scalable target by the specified value. A positive value increases the capacity and a negative value decreases the capacity. For example: If the current capacity is 3 and the adjustment is 5, then Application Auto Scaling adds 5 to the capacity for a total of 8.
- **ExactCapacity**—Change the current capacity of the scalable target to the specified value. Specify a non-negative value with this adjustment type. For example: If the current capacity is 3 and the adjustment is 5, then Application Auto Scaling changes the capacity to 5.

- **PercentChangeInCapacity**—Increase or decrease the current capacity of the scalable target by the specified percentage. A positive value increases the capacity and a negative value decreases the capacity. For example: If the current capacity is 10 and the adjustment is 10 percent, then Application Auto Scaling adds 1 to the capacity for a total of 11.

  If the resulting value is not an integer, Application Auto Scaling rounds it as follows:

  - Values greater than 1 are rounded down. For example, `12.7` is rounded to `12`.
  - Values between 0 and 1 are rounded to 1. For example, `.67` is rounded to `1`.
  - Values between 0 and -1 are rounded to -1. For example, `-.58` is rounded to `-1`.
  - Values less than -1 are rounded up. For example, `-6.67` is rounded to `-6`.

  With **PercentChangeInCapacity**, you can also specify the minimum amount to scale using the `MinAdjustmentMagnitude` parameter. For example, suppose that you create a policy that adds 25 percent and you specify a minimum amount of 2. If the scalable target has a capacity of 4 and the scaling policy is performed, 25 percent of 4 is 1. However, because you specified a minimum increment of 2, Application Auto Scaling adds 2.

Cooldown period

You can optionally define a cooldown period in your step scaling policy.

A cooldown period specifies the amount of time the scaling policy waits for a previous scaling activity to take effect.

There are two ways to plan for the use of cooldown periods for a step scaling configuration:

- With the cooldown period for scale-out policies, the intention is to continuously (but not excessively) scale out. After Application Auto Scaling successfully scales out using a scaling policy, it starts to calculate the cooldown time. A scaling policy won't increase the desired capacity again unless either a larger scale out is triggered or the cooldown period ends. While the scale-out cooldown period is in effect, the capacity added by the initiating scale-out activity is calculated as part of the desired capacity for the next scale-out activity.
- With the cooldown period for scale-in policies, the intention is to scale in conservatively to protect your application's availability, so scale-in activities are blocked until the scale-in cooldown period has expired. However, if another alarm triggers a scale-out activity during the scale-in cooldown period, Application Auto Scaling scales out the target immediately. In this case, the scale-in cooldown period stops and doesn't complete.

For example, when a traffic peak occurs, an alarm is triggered and Application Auto Scaling automatically adds capacity to help handle the increased load. If you set a cooldown period for your scale-out policy, when the alarm triggers the policy to increase the capacity by 2, the scaling activity completes successfully, and the scale-out cooldown period starts. If an alarm triggers again during the cooldown period but at a more aggressive step adjustment of 3, the previous increase of 2 is considered part of the current capacity. Therefore, only 1 is added to the capacity. This allows faster scaling than waiting for the cooldown to expire but without adding more capacity than you need.

The cooldown period is measured in seconds and applies only to scaling policy-related scaling activities. During a cooldown period, when a scheduled action starts at the scheduled time, it can trigger a scaling activity immediately without waiting for the cooldown period to expire.

The default value is 300 if no value is specified.

Commonly used commands for scaling policy creation, management, and deletion

The commonly used commands for working with scaling policies include:

- register-scalable-target to register AWS or custom resources as scalable targets (a resource that Application Auto Scaling can scale), and to suspend and resume scaling.
- put-scaling-policy to add or modify scaling policies for an existing scalable target.
- describe-scaling-activities to return information about scaling activities in an AWS Region.
- describe-scaling-policies to return information about scaling policies in an AWS Region.
- delete-scaling-policy to delete a scaling-policy.

Considerations

The following considerations apply when working with step scaling policies:

- Consider whether you can predict the step adjustments on the application accurately enough to use step scaling. If your scaling metric increases or decreases proportionally to the capacity of the scalable target, we recommend that you use a target tracking scaling policy instead. You still have the option to use step scaling as an additional policy for a more advanced configuration. For example, you can configure a more aggressive response when utilization reaches a certain level.

- Make sure to choose an adequate margin between the scale-out and scale-in thresholds to prevent flapping. Flapping is an infinite loop of scaling in and scaling out. That is, if a scaling action is taken, the metric value would change and start another scaling action in the reverse direction.

Related resources

For information about creating step scaling policies for Auto Scaling groups, see Step and simple scaling policies for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Console access

Console access to view, add, update, or remove step scaling policies on scalable resources depends on the resource that you use. For more information, see AWS services that you can use with Application Auto Scaling.

Create a step scaling policy for Application Auto Scaling using the AWS CLI

This example uses AWS CLI commands to create a step scaling policy for an Amazon ECS service. For a different scalable target, specify its namespace in `--service-namespace`, its scalable dimension in `--scalable-dimension`, and its resource ID in `--resource-id`.

When using the AWS CLI, remember that your commands run in the AWS Region configured for your profile. If you want to run the commands in a different Region, either change the default Region for your profile, or use the `--region` parameter with the command.

Tasks

- Step 1: Register a scalable target
- Step 2: Create a step scaling policy
- Step 3: Create an alarm that invokes a scaling policy

Step 1: Register a scalable target

If you haven't already done so, register the scalable target. Use the register-scalable-target command to register a specific resource in the target service as a scalable target. The following

example registers an Amazon ECS service with Application Auto Scaling. Application Auto Scaling can scale the number of tasks at a minimum of 2 tasks and a maximum of 10. Replace each *user input placeholder* with your own information.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service \
  --min-capacity 2 --max-capacity 10
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace ecs ^
  --scalable-dimension ecs:service:DesiredCount ^
  --resource-id service/my-cluster/my-service ^
  --min-capacity 2 --max-capacity 10
```

Output

If successful, this command returns the ARN of the scalable target. The following is example output.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Step 2: Create a step scaling policy

To create a step scaling policy for your scalable target, you can use the following examples to help you get started.

Scale out

To create a step scaling policy for scale out (increase capacity)

1. Use the following `cat` command to store a step scaling policy configuration in a JSON file named `config.json` in your home directory. The following is an example configuration with an adjustment type of `PercentChangeInCapacity` that increases the capacity of

the scalable target based on the following step adjustments (assuming a CloudWatch alarm threshold of 70):

- Increase capacity by 10 percent when the value of the metric is greater than or equal to 70 but less than 85
- Increase capacity by 20 percent when the value of the metric is greater than or equal to 85 but less than 95
- Increase capacity by 30 percent when the value of the metric is greater than or equal to 95

```
$ cat ~/config.json
{
  "AdjustmentType": "PercentChangeInCapacity",
  "MetricAggregationType": "Average",
  "Cooldown": 60,
  "MinAdjustmentMagnitude": 1,
  "StepAdjustments": [
    {
      "MetricIntervalLowerBound": 0.0,
      "MetricIntervalUpperBound": 15.0,
      "ScalingAdjustment": 10
    },
    {
      "MetricIntervalLowerBound": 15.0,
      "MetricIntervalUpperBound": 25.0,
      "ScalingAdjustment": 20
    },
    {
      "MetricIntervalLowerBound": 25.0,
      "ScalingAdjustment": 30
    }
  ]
}
```

For more information, see StepScalingPolicyConfiguration in the *Application Auto Scaling API Reference*.

2. Use the following put-scaling-policy command, along with the `config.json` file that you created, to create a scaling policy named `my-step-scaling-policy`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scaling-policy --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service \
  --policy-name my-step-scaling-policy --policy-type StepScaling \
  --step-scaling-policy-configuration file://config.json
```

Windows

```
aws application-autoscaling put-scaling-policy --service-namespace ecs ^
  --scalable-dimension ecs:service:DesiredCount ^
  --resource-id service/my-cluster/my-service ^
  --policy-name my-step-scaling-policy --policy-type StepScaling ^
  --step-scaling-policy-configuration file://config.json
```

Output

The output includes the ARN that serves as a unique name for the policy. You need it to create a CloudWatch alarm for your policy. The following is example output.

```
{
    "PolicyARN":
 "arn:aws:autoscaling:region:123456789012:scalingPolicy:ac542982-cbeb-4294-891c-a5a941dfa787:resource/ecs/service/my-cluster/my-service:policyName/my-step-scaling-policy"
}
```

Scale in

To create a step scaling policy for scale in (decrease capacity)

1. Use the following `cat` command to store a step scaling policy configuration in a JSON file named `config.json` in your home directory. The following is an example configuration with an adjustment type of `ChangeInCapacity` that decreases the capacity of the scalable target based on the following step adjustments (assuming a CloudWatch alarm threshold of 50):

- Decrease capacity by 1 when the value of the metric is less than or equal to 50 but greater than 40
- Decrease capacity by 2 when the value of the metric is less than or equal to 40 but greater than 30
- Decrease capacity by 3 when the value of the metric is less than or equal to 30

```
$ cat ~/config.json
{
  "AdjustmentType": "ChangeInCapacity",
  "MetricAggregationType": "Average",
  "Cooldown": 60,
  "StepAdjustments": [
    {
      "MetricIntervalUpperBound": 0.0,
      "MetricIntervalLowerBound": -10.0,
      "ScalingAdjustment": -1
    },
    {
      "MetricIntervalUpperBound": -10.0,
      "MetricIntervalLowerBound": -20.0,
      "ScalingAdjustment": -2
    },
    {
      "MetricIntervalUpperBound": -20.0,
      "ScalingAdjustment": -3
    }
  ]
}
```

For more information, see StepScalingPolicyConfiguration in the *Application Auto Scaling API Reference*.

2. Use the following put-scaling-policy command, along with the `config.json` file that you created, to create a scaling policy named `my-step-scaling-policy`.

Linux, macOS, or Unix

```
aws application-autoscaling put-scaling-policy --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service \
```

```
  --policy-name my-step-scaling-policy --policy-type StepScaling \
  --step-scaling-policy-configuration file://config.json
```

Windows

```
aws application-autoscaling put-scaling-policy --service-namespace ecs ^
  --scalable-dimension ecs:service:DesiredCount ^
  --resource-id service/my-cluster/my-service ^
  --policy-name my-step-scaling-policy --policy-type StepScaling ^
  --step-scaling-policy-configuration file://config.json
```

Output

The output includes the ARN that serves as a unique name for the policy. You need this ARN to create a CloudWatch alarm for your policy. The following is example output.

```
{
    "PolicyARN":
 "arn:aws:autoscaling:region:123456789012:scalingPolicy:ac542982-cbeb-4294-891c-a5a941dfa787:resource/ecs/service/my-cluster/my-service:policyName/my-step-scaling-policy"
}
```

Step 3: Create an alarm that invokes a scaling policy

Finally, use the following CloudWatch put-metric-alarm command to create an alarm to use with your step scaling policy. In this example, you have an alarm based on average CPU utilization. The alarm is configured to be in an ALARM state if it reaches a threshold of 70 percent for at least two consecutive evaluation periods of 60 seconds. To specify a different CloudWatch metric or use your own custom metric, specify its name in `--metric-name` and its namespace in `--namespace`.

Linux, macOS, or Unix

```
aws cloudwatch put-metric-alarm --alarm-name Step-Scaling-AlarmHigh-ECS:service/my-cluster/my-service \
  --metric-name CPUUtilization --namespace AWS/ECS --statistic Average \
  --period 60 --evaluation-periods 2 --threshold 70 \
  --comparison-operator GreaterThanOrEqualToThreshold \
  --dimensions Name=ClusterName,Value=default Name=ServiceName,Value=sample-app-service
 \
```

```
  --alarm-actions PolicyARN
```

Windows

```
aws cloudwatch put-metric-alarm --alarm-name Step-Scaling-AlarmHigh-ECS:service/my-
cluster/my-service ^
  --metric-name CPUUtilization --namespace AWS/ECS --statistic Average ^
  --period 60 --evaluation-periods 2 --threshold 70 ^
  --comparison-operator GreaterThanOrEqualToThreshold ^
  --dimensions Name=ClusterName,Value=default Name=ServiceName,Value=sample-app-service
 ^
  --alarm-actions PolicyARN
```

Describe step scaling policies for Application Auto Scaling using the AWS CLI

You can describe all scaling policies for a service namespace using the describe-scaling-policies command. The following example describes all scaling policies for all Amazon ECS services. To list them for a specific Amazon ECS service only add the `--resource-id` option.

```
aws application-autoscaling describe-scaling-policies --service-namespace ecs
```

You can filter the results to just the step scaling policies using the `--query` parameter. For more information about the syntax for `query`, see Controlling command output from the AWS CLI in the *AWS Command Line Interface User Guide*.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-policies --service-namespace ecs \
  --query 'ScalingPolicies[?PolicyType==`StepScaling`]'
```

Windows

```
aws application-autoscaling describe-scaling-policies --service-namespace ecs ^
  --query "ScalingPolicies[?PolicyType==`StepScaling`]"
```

Output

The following is example output.

```
[
    {
        "PolicyARN": "PolicyARN",
        "StepScalingPolicyConfiguration": {
            "MetricAggregationType": "Average",
            "Cooldown": 60,
            "StepAdjustments": [
                {
                    "MetricIntervalLowerBound": 0.0,
                    "MetricIntervalUpperBound": 15.0,
                    "ScalingAdjustment": 1
                },
                {
                    "MetricIntervalLowerBound": 15.0,
                    "MetricIntervalUpperBound": 25.0,
                    "ScalingAdjustment": 2
                },
                {
                    "MetricIntervalLowerBound": 25.0,
                    "ScalingAdjustment": 3
                }
            ],
            "AdjustmentType": "ChangeInCapacity"
        },
        "PolicyType": "StepScaling",
        "ResourceId": "service/my-cluster/my-service",
        "ServiceNamespace": "ecs",
        "Alarms": [
            {
                "AlarmName": "Step-Scaling-AlarmHigh-ECS:service/my-cluster/my-
service",
                "AlarmARN": "arn:aws:cloudwatch:region:012345678910:alarm:Step-Scaling-
AlarmHigh-ECS:service/my-cluster/my-service"
            }
        ],
        "PolicyName": "my-step-scaling-policy",
        "ScalableDimension": "ecs:service:DesiredCount",
        "CreationTime": 1515024099.901
    }
]
```

Delete a step scaling policy for Application Auto Scaling using the AWS CLI

When you no longer need a step scaling policy, you can delete it. To delete both the scaling policy and the associated CloudWatch alarm, complete the following tasks.

To delete your scaling policy

Use the delete-scaling-policy command.

Linux, macOS, or Unix

```
aws application-autoscaling delete-scaling-policy --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service \
  --policy-name my-step-scaling-policy
```

Windows

```
aws application-autoscaling delete-scaling-policy --service-namespace ecs ^
  --scalable-dimension ecs:service:DesiredCount ^
  --resource-id service/my-cluster/my-service ^
  --policy-name my-step-scaling-policy
```

To delete the CloudWatch alarm

Use the delete-alarms command. You can delete one or more alarms at a time. For example, use the following command to delete the `Step-Scaling-AlarmHigh-ECS:service/my-cluster/my-service` and `Step-Scaling-AlarmLow-ECS:service/my-cluster/my-service` alarms.

```
aws cloudwatch delete-alarms --alarm-name Step-Scaling-AlarmHigh-ECS:service/my-
cluster/my-service Step-Scaling-AlarmLow-ECS:service/my-cluster/my-service
```

Predictive scaling for Application Auto Scaling

Predictive scaling proactively scales your application. Predictive scaling analyzes historical load data to detect daily or weekly patterns in traffic flows. It uses this information to forecast future capacity needs to proactively increase the capacity of your application to match the anticipated load.

Predictive scaling is well suited for situations where you have:

- Cyclical traffic, such as high use of resources during regular business hours and low use of resources during evenings and weekends
- Recurring on-and-off workload patterns, such as batch processing, testing, or periodic data analysis.
- Applications that take a long time to initialize, causing a noticeable latency impact on application performance during scale-out events

Contents

- How Application Auto Scaling predictive scaling works
- Create a predictive scaling policy for Application Auto Scaling
- Override forecast values using scheduled actions
- Advanced predictive scaling policy using custom metrics

How Application Auto Scaling predictive scaling works

To use predictive scaling, create a predictive scaling policy that specifies the CloudWatch metric to monitor and analyze. You can use a predefined metric or a custom metric. For predictive scaling to start forecasting future values, this metric must have at least 24 hours of data.

After you create the policy, predictive scaling starts analyzing metric data from up to the past 14 days to identify patterns. It uses this analysis to generate an hourly forecast of capacity requirements for the next 48 hours. The forecast is updated every 6 hours using the latest CloudWatch data. As new data comes in, predictive scaling is able to continuously improve the accuracy of future forecasts.

You can first enable predictive scaling in *forecast only* mode. In this mode, it generates capacity forecasts but does not actually scale your capacity based on those forecasts. This allows you to evaluate the accuracy and suitability of the forecast.

After you review the forecast data and decide to start scaling based on that data, switch the scaling policy to forecast and scale mode. In this mode:

- If the forecast expects an increase in load, predictive scaling will increase the capacity.
- If the forecast expects a decrease in load, predictive scaling will not scale in to remove capacity. This ensures that you scale-in only when the demand actually drops, and not just on predictions. To remove capacity that is no longer needed, you must create a Target Tracking or Step Scaling policy because they respond to real time metric data.

By default, predictive scaling scales your scalable targets at the start of each hour based on the forecast for that hour. You can optionally specify an earlier start time by using the `SchedulingBufferTime` property in the `PutScalingPolicy` API operation. This allows you to launch predicted capacity ahead of the forecasted demand, which gives the new capacity adequate time to become ready to handle traffic.

Maximum capacity limit

By default, when scaling policies are set, they cannot increase capacity higher than its maximum capacity.

Alternatively, you can allow the scalable target's maximum capacity to be automatically increased if the forecast capacity approaches or exceeds the maximum capacity of the scalable target. To enable this behavior, use the `MaxCapacityBreachBehavior` and `MaxCapacityBuffer` properties in the `PutScalingPolicy` API operation or the **Max capacity behavior** setting in the AWS Management Console.

> **Warning**
>
> Use caution when allowing the maximum capacity to be automatically increased. The maximum capacity does not automatically decrease back to the original maximum.

Commonly used commands for scaling policy creation, management, and deletion

The commonly used commands for working with predictive scaling policies include:

- `register-scalable-target` to register AWS or custom resources as scalable targets, to suspend scaling, and to resume scaling.
- `put-scaling-policy` to create a predictive scaling policy.
- `get-predictive-scaling-forecast` to retrieve the forecast data for a predictive scaling policy.
- `describe-scaling-activities` to return information about scaling activities in an AWS Region.
- `describe-scaling-policies` to return information about scaling policies in an AWS Region.
- `delete-scaling-policy` to delete a scaling policy.

Custom metrics

Custom metrics can be used to predict the capacity needed for an application. Custom metrics are useful when predefined metrics are not enough to capture the load on your application.

Considerations

The following considerations apply when working with predictive scaling.

- Confirm whether predictive scaling is suitable for your application. An application is a good fit for predictive scaling if it exhibits recurring load patterns that are specific to the day of the week or the time of day. Evaluate the forecast before letting predictive scaling actively scale your application.
- Predictive scaling needs at least 24 hours of historical data to start forecasting. However, forecasts are more effective if historical data spans two full weeks.
- Choose a load metric that accurately represents the full load on your application and is the aspect of your application that's most important to scale on.

Create a predictive scaling policy for Application Auto Scaling

The following example policy uses the AWS CLI to configure a predictive scaling policy for Amazon ECS service. Replace each *user input placeholder* with your own information.

For more information about the CloudWatch metrics you can specify, see PredictiveScalingMetricSpecification in the *Amazon EC2 Auto Scaling API Reference*.

The following is an example policy with a predefined memory configuration.

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 40,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceMemoryUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

The following example illustrates creating the policy by running the put-scaling-policy command with the configuration file specified.

```
aws aas put-scaling-policy \
--service-namespace ecs \
--region us-east-1 \
--policy-name predictive-scaling-policy-example \
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

If successful, this command returns the policy's ARN.

```
{
```

```
"PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
"Alarms": []
}
```

Override forecast values using scheduled actions

Sometimes, you might have additional information about your future application requirements that the forecast calculation is unable to take into account. For example, forecast calculations might underestimate the capacity needed for an upcoming marketing event. You can use scheduled actions to temporarily override the forecast during future time periods. The scheduled actions can run on a recurring basis, or at a specific date and time when there are one-time demand fluctuations.

For example, you can create a scheduled action with a higher minimum capacity than what is forecasted. At runtime, Application Auto Scaling updates the minimum capacity of your scalable target. Because predictive scaling optimizes for capacity, a scheduled action with a minimum capacity that is higher than the forecast values is honored. This prevents capacity from being less than expected. To stop overriding the forecast, use a second scheduled action to return the minimum capacity to its original setting.

The following procedure outlines the steps for overriding the forecast during future time periods.

Topics

- Step 1: (Optional) Analyze time series data
- Step 2: Create two scheduled actions

> **Important**
>
> This topic assumes that you are trying to override the forecast to scale to a higher capacity than what is forecasted. If you need to temporarily decrease capacity without interference from a predictive scaling policy, use *forecast only* mode instead. While in forecast only mode, predictive scaling will continue to generate forecasts, but it will not automatically increase capacity. You can then monitor resource utilization and manually decrease the size of your group as needed.

Step 1: (Optional) Analyze time series data

Start by analyzing the forecast time series data. This is an optional step, but it is helpful if you want to understand the details of the forecast.

1. **Retrieve the forecast**

   After the forecast is created, you can query for a specific time period in the forecast. The goal of the query is to get a complete view of the time series data for a specific time period.

   Your query can include up to two days of future forecast data. If you have been using predictive scaling for a while, you can also access your past forecast data. However, the maximum time duration between the start and end time is 30 days.

   To retrieve the forecast, use the get-predictive-scaling-forecast command. The following example gets the predictive scaling forecast for the Amazon ECS service.

   ```
   aws application-autoscaling get-predictive-scaling-forecast --service-namespace ecs
    \
       --scalable-dimension ecs:service:DesiredCount \
       --resource-id 1234567890abcdef0
       --policy-name predictive-scaling-policy \
       --start-time "2021-05-19T17:00:00Z" \
       --end-time "2021-05-19T23:00:00Z"
   ```

   The response includes two forecasts: `LoadForecast` and `CapacityForecast`. `LoadForecast` shows the hourly load forecast. `CapacityForecast` shows forecast values for the capacity that is needed on an hourly basis to handle the forecasted load while maintaining a specified `TargetValue`.

2. **Identify the target time period**

   Identify the hour or hours when the one-time demand fluctuation should take place. Remember that dates and times shown in the forecast are in UTC.

Step 2: Create two scheduled actions

Next, create two scheduled actions for a specific time period when your application will have a higher than forecasted load. For example, if you have a marketing event that will drive traffic to your site for a limited period of time, you can schedule a one-time action to update the minimum

capacity when it starts. Then, schedule another action to return the minimum capacity to the original setting when the event ends.

To create two scheduled actions for one-time events (AWS CLI)

To create the scheduled actions, use the put-scheduled-action command.

The following example defines an schedule for Amazon EC2 Auto Scaling that maintains a minimum capacity of three instances on May 19 at 5:00 PM for eight hours. The following commands show how to implement this scenario.

The first put-scheduled-update-group-action command instructs Amazon EC2 Auto Scaling to update the minimum capacity of the specified Auto Scaling group at 5:00 PM UTC on May 19, 2021.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-
start \
  --auto-scaling-group-name my-asg --start-time "2021-05-19T17:00:00Z" --minimum-
capacity 3
```

The second command instructs Amazon EC2 Auto Scaling to set the group's minimum capacity to one at 1:00 AM UTC on May 20, 2021.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-end
 \
  --auto-scaling-group-name my-asg --start-time "2021-05-20T01:00:00Z" --minimum-
capacity 1
```

After you add these scheduled actions to the Auto Scaling group, Amazon EC2 Auto Scaling does the following:

- At 5:00 PM UTC on May 19, 2021, the first scheduled action runs. If the group currently has fewer than three instances, the group scales out to three instances. During this time and for the next eight hours, Amazon EC2 Auto Scaling can continue to scale out if the predicted capacity is higher than the actual capacity or if there is a dynamic scaling policy in effect.
- At 1:00 AM UTC on May 20, 2021, the second scheduled action runs. This returns the minimum capacity to its original setting at the end of the event.

Scaling based on recurring schedules

To override the forecast for the same time period every week, create two scheduled actions and provide the time and date logic using a cron expression.

The cron expression format consists of five fields separated by spaces: [Minute] [Hour] [Day_of_Month] [Month_of_Year] [Day_of_Week]. Fields can contain any allowed values, including special characters.

For example, the following cron expression runs the action every Tuesday at 6:30 AM. The asterisk is used as a wildcard to match all values for a field.

```
30 6 * * 2
```

Advanced predictive scaling policy using custom metrics

In a predictive scaling policy, you can use predefined or custom metrics. Custom metrics are useful when the predefined metrics do not sufficiently describe your application load.

When creating a predictive scaling policy with custom metrics, you can specify other CloudWatch metrics provided by AWS, or you can specify metrics that you define and publish yourself. You can also use metric math to aggregate and transform existing metrics into a new time series that AWS doesn't automatically track. When you combine values in your data, for example, by calculating new sums or averages, it's called *aggregating*. The resulting data is called an *aggregate*.

The following section contains best practices and examples of how to construct the JSON structure for the policy.

Topics

- Best practices
- Prerequisites
- Constructing the JSON for custom metrics
- Considerations for custom metrics in a predictive scaling policy

Best practices

The following best practices can help you use custom metrics more effectively:

- For the load metric specification, the most useful metric is a metric that represents the load on your application.
- The scaling metric must be inversely proportional to capacity. That is, if the scalable target increases, the scaling metric should decrease by roughly the same proportion. To ensure that predictive scaling behaves as expected, the load metric and the scaling metric must also correlate strongly with each other.
- The target utilization must match the type of scaling metric. For a policy configuration that uses CPU utilization, this is a target percentage. For a policy configuration that uses throughput, such as the number of requests or messages, this is the target number of requests or messages per instance during any one-minute interval.
- If these recommendations are not followed, the forecasted future values of the time series will probably be incorrect. To validate that the data is correct, you can view the forecasted values. Alternatively, after you create your predictive scaling policy, inspect the `LoadForecast` and `CapacityForecast` objects returned by a call to the GetPredictiveScalingForecast API.
- We strongly recommend that you configure predictive scaling in forecast only mode so that you can evaluate the forecast before predictive scaling starts actively scaling capacity.

Prerequisites

To add custom metrics to your predictive scaling policy, you must have `cloudwatch:GetMetricData` permissions.

To specify your own metrics instead of the metrics that AWS provides, you must first publish your metrics to CloudWatch. For more information, see Publishing custom metrics in the *Amazon CloudWatch User Guide*.

If you publish your own metrics, make sure to publish the data points at a minimum frequency of five minutes. Application Auto Scaling retrieves the data points from CloudWatch based on the length of the period that it needs. For example, the load metric specification uses hourly metrics to measure the load on your application. CloudWatch uses your published metric data to provide a single data value for any one-hour period by aggregating all data points with timestamps that fall within each one-hour period.

Constructing the JSON for custom metrics

The following section contains examples for how to configure predictive scaling to query data from CloudWatch for Amazon EC2 Auto Scaling. There are two different methods to configure this

option, and the method that you choose affects which format you use to construct the JSON for your predictive scaling policy. When you use metric math, the format of the JSON varies further based on the metric math being performed.

1. To create a policy that gets data directly from other CloudWatch metrics provided by AWS or metrics that you publish to CloudWatch, see Example predictive scaling policy with custom load and scaling metrics (AWS CLI).
2. To create a policy that can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics, see Use metric math expressions.

Example predictive scaling policy with custom load and scaling metrics (AWS CLI)

To create a predictive scaling policy with custom load and scaling metrics with the AWS CLI, store the arguments for `--predictive-scaling-configuration` in a JSON file named `config.json`.

You start adding custom metrics by replacing the replaceable values in the following example with those of your metrics and your target utilization.

```
{
  "MetricSpecifications": [
    {
      "TargetValue": 50,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "scaling_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyUtilizationMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
              },
              "Stat": "Average"
            }
          }
```

```
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyLoadMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
              },
              "Stat": "Sum"
            }
          }
        ]
      }
    }
  ]
}
```

For more information, see MetricDataQuery in the *Amazon EC2 Auto Scaling API Reference*.

> **Note**
>
> Following are some additional resources that can help you find metric names, namespaces, dimensions, and statistics for CloudWatch metrics:
>
> - For information about the available metrics for AWS services, see AWS services that publish CloudWatch metrics in the *Amazon CloudWatch User Guide*.
> - To get the exact metric name, namespace, and dimensions (if applicable) for a CloudWatch metric with the AWS CLI, see list-metrics.

To create this policy, run the put-scaling-policy command using the JSON file as input, as demonstrated in the following example.

```
aws autoscaling put-scaling-policy --policy-name my-predictive-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN).

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-predictive-scaling-policy",
  "Alarms": []
}
```

Use metric math expressions

The following section provides information and examples of predictive scaling policies that show how you can use metric math in your policy.

Topics

- Understand metric math
- Example predictive scaling policy for Amazon EC2 Auto Scaling that combines metrics using metric math (AWS CLI)
- Example predictive scaling policy to use in a blue/green deployment scenario (AWS CLI)

Understand metric math

If all you want to do is aggregate existing metric data, CloudWatch metric math saves you the effort and cost of publishing another metric to CloudWatch. You can use any metric that AWS provides, and you can also use metrics that you define as part of your applications.

For more information, see Using metric math in the *Amazon CloudWatch User Guide*.

If you choose to use a metric math expression in your predictive scaling policy, consider the following points:

- Metric math operations use the data points of the unique combination of metric name, namespace, and dimension keys/value pairs of metrics.
- You can use any arithmetic operator (+ - * / ^), statistical function (such as AVG or SUM), or other function that CloudWatch supports.

- You can use both metrics and the results of other math expressions in the formulas of the math expression.
- Your metric math expressions can be made up of different aggregations. However, it's a best practice for the final aggregation result to use `Average` for the scaling metric and `Sum` for the load metric.
- Any expressions used in a metric specification must eventually return a single time series.

To use metric math, do the following:

- Choose one or more CloudWatch metrics. Then, create the expression. For more information, see Using metric math in the *Amazon CloudWatch User Guide*.
- Verify that the metric math expression is valid by using the CloudWatch console or the CloudWatch GetMetricData API.

Example predictive scaling policy for Amazon EC2 Auto Scaling that combines metrics using metric math (AWS CLI)

Sometimes, instead of specifying the metric directly, you might need to first process its data in some way. For example, you might have an application that pulls work from an Amazon SQS queue, and you might want to use the number of items in the queue as criteria for predictive scaling. The number of messages in the queue does not solely define the number of instances that you need. Therefore, more work is needed to create a metric that can be used to calculate the backlog per instance.

The following is an example predictive scaling policy for this scenario. It specifies scaling and load metrics that are based on the Amazon SQS `ApproximateNumberOfMessagesVisible` metric, which is the number of messages available for retrieval from the queue. It also uses the Amazon EC2 Auto Scaling `GroupInServiceInstances` metric and a math expression to calculate the backlog per instance for the scaling metric.

```
aws autoscaling put-scaling-policy --policy-name my-sqs-custom-metrics-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
{
  "MetricSpecifications": [
    {
      "TargetValue": 100,
      "CustomizedScalingMetricSpecification": {
```

```
        "MetricDataQueries": [
          {
            "Label": "Get the queue size (the number of messages waiting to be
processed)",
            "Id": "queue_size",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Get the group size (the number of running instances)",
            "Id": "running_capacity",
            "MetricStat": {
              "Metric": {
                "MetricName": "GroupInServiceInstances",
                "Namespace": "AWS/AutoScaling",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "my-asg"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Calculate the backlog per instance",
            "Id": "scaling_metric",
            "Expression": "queue_size / running_capacity",
            "ReturnData": true
          }
```

```
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ],
              },
              "Stat": "Sum"
            },
            "ReturnData": true
          }
        ]
      }
    }
  ]
}
```

The example returns the policy's ARN.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-sqs-custom-metrics-policy",
  "Alarms": []
}
```

Example predictive scaling policy to use in a blue/green deployment scenario (AWS CLI)

A search expression provides an advanced option in which you can query for a metric from multiple Auto Scaling groups and perform math expressions on them. This is especially useful for blue/green deployments.

Note

A *blue/green deployment* is a deployment method in which you create two separate but identical Auto Scaling groups. Only one of the groups receives production traffic. User traffic is initially directed to the earlier ("blue") Auto Scaling group, while a new group ("green") is used for testing and evaluation of a new version of an application or service. User traffic is shifted to the green Auto Scaling group after a new deployment is tested and accepted. You can then delete the blue group after the deployment is successful.

When new Auto Scaling groups get created as part of a blue/green deployment, the metric history of each group can be automatically included in the predictive scaling policy without you having to change its metric specifications. For more information, see Using EC2 Auto Scaling predictive scaling policies with Blue/Green deployments on the AWS Compute Blog.

The following example policy shows how this can be done. In this example, the policy uses the `CPUUtilization` metric emitted by Amazon EC2. It uses the Amazon EC2 Auto Scaling `GroupInServiceInstances` metric and a math expression to calculate the value of the scaling metric per instance. It also specifies a capacity metric specification to get the `GroupInServiceInstances` metric.

The search expression finds the `CPUUtilization` of instances in multiple Auto Scaling groups based on the specified search criteria. If you later create a new Auto Scaling group that matches the same search criteria, the `CPUUtilization` of the instances in the new Auto Scaling group is automatically included.

```
aws autoscaling put-scaling-policy --policy-name my-blue-green-predictive-scaling-
policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
{
  "MetricSpecifications": [
    {
      "TargetValue": 25,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_sum",
            "Expression": "SUM(SEARCH('{AWS/EC2,AutoScalingGroupName} MetricName=
\"CPUUtilization\" ASG-myapp', 'Sum', 300))",
```

```
            "ReturnData": false
          },
          {
            "Id": "capacity_sum",
            "Expression": "SUM(SEARCH('{AWS/AutoScaling,AutoScalingGroupName}
 MetricName=\"GroupInServiceInstances\" ASG-myapp', 'Average', 300))",
            "ReturnData": false
          },
          {
            "Id": "weighted_average",
            "Expression": "load_sum / capacity_sum",
            "ReturnData": true
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_sum",
            "Expression": "SUM(SEARCH('{AWS/EC2,AutoScalingGroupName} MetricName=
\"CPUUtilization\" ASG-myapp', 'Sum', 3600))"
          }
        ]
      },
      "CustomizedCapacityMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "capacity_sum",
            "Expression": "SUM(SEARCH('{AWS/AutoScaling,AutoScalingGroupName}
 MetricName=\"GroupInServiceInstances\" ASG-myapp', 'Average', 300))"
          }
        ]
      }
    }
  ]
}
```

The example returns the policy's ARN.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-blue-green-predictive-
scaling-policy",
```

```
  "Alarms": []
}
```

Considerations for custom metrics in a predictive scaling policy

If an issue occurs while using custom metrics, we recommend that you do the following:

- If an error message is provided, read the message and resolve the issue it reports, if possible.
- If you did not validate an expression in advance, the put-scaling-policy command validates it when you create your scaling policy. However, there is a possibility that this command might fail to identify the exact cause of the detected errors. To fix the issues, troubleshoot the errors that you receive in a response from a request to the get-metric-data command. You can also troubleshoot the expression from the CloudWatch console.
- You must specify `false` for `ReturnData` if `MetricDataQueries` specifies the SEARCH() function on its own without a math function like SUM(). This is because search expressions might return multiple time series, and a metric specification based on an expression can return only one time series.
- All metrics involved in a search expression should be of the same resolution.

Limitations

The following limitations apply.

- You can query data points of up to 10 metrics in one metric specification.
- For the purposes of this limit, one expression counts as one metric.

Tutorial: Configure auto scaling to handle a heavy workload

In this tutorial, you learn how to scale out and in based on time windows when your application will have a heavier than normal workload. This is helpful when you have an application that can suddenly have a large number of visitors on a regular schedule or on a seasonal basis.

You can use a target tracking scaling policy together with scheduled scaling to handle the extra load. Scheduled scaling automatically initiates changes to your `MinCapacity` and `MaxCapacity` on your behalf, based on a schedule that you specify. When a target tracking scaling policy is active on the resource, it can scale dynamically based on current resource utilization, within the new minimum and maximum capacity range.

After completing this tutorial, you'll know how to:

- Use scheduled scaling to add extra capacity to meet a heavy load before it arrives, and then remove the extra capacity when it's no longer required.
- Use a target tracking scaling policy to scale your application based on current resource utilization.

Contents

- Prerequisites
- Step 1: Register your scalable target
- Step 2: Set up scheduled actions according to your requirements
- Step 3: Add a target tracking scaling policy
- Step 4: Next steps
- Step 5: Clean up

Prerequisites

This tutorial assumes that you have already done the following:

- Created an AWS account.

- Installed and configured the AWS CLI.
- Granted the necessary permissions for registering and deregistering resources as scalable targets with Application Auto Scaling. In addition, granted the necessary permissions for creating scaling policies and scheduled actions. For more information, see Identity and Access Management for Application Auto Scaling.
- Created a supported resource in a non-production environment available to use for this tutorial. If you don't already have one, create one now. For information about the AWS services and resources that work with Application Auto Scaling, see the AWS services that you can use with Application Auto Scaling section.

> **Note**
>
> While completing this tutorial, there are two steps in which you set your resource's minimum and maximum capacity values to 0 to reset the current capacity to 0. Depending on which resource you're using with Application Auto Scaling, you might be unable to reset the current capacity to 0 during these steps. To help you address the issue, a message in the output will indicate that minimum capacity cannot be less than the value specified and will provide the minimum capacity value that the AWS resource can accept.

Step 1: Register your scalable target

Start by registering your resource as a scalable target with Application Auto Scaling. A scalable target is a resource that Application Auto Scaling can scale out and scale in.

To register your scalable target with Application Auto Scaling

- Use the following register-scalable-target command to register a new scalable target. Set the `--min-capacity` and `--max-capacity` values to 0 to reset the current capacity to 0.

  Replace the sample text for `--service-namespace` with the namespace of the AWS service you're using with Application Auto Scaling, `--scalable-dimension` with the scalable dimension associated with the resource you're registering, and `--resource-id` with an identifier for the resource. These values vary based on which resource is used and how the resource ID is constructed. See the topics in the AWS services that you can use with Application Auto Scaling section for more information. These topics include example commands that show you how to register scalable targets with Application Auto Scaling.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier \
  --min-capacity 0 --max-capacity 0
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace namespace
 --scalable-dimension dimension --resource-id identifier --min-capacity 0 --max-
capacity 0
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-
id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Step 2: Set up scheduled actions according to your requirements

You can use the put-scheduled-action command to create scheduled actions that are configured to meet your business needs. In this tutorial, we focus on a configuration that stops consuming resources outside of working hours by reducing capacity to 0.

To create a scheduled action that scales out in the morning

1. To scale out the scalable target, use the following put-scheduled-action command. Include the `--schedule` parameter with a recurring schedule, in UTC, using a cron expression.

   On the specified schedule (every day at 9:00 AM UTC), Application Auto Scaling updates the `MinCapacity` and `MaxCapacity` values to the desired range of 1-5 capacity units.

   Linux, macOS, or Unix

```
aws application-autoscaling put-scheduled-action \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier \
  --scheduled-action-name my-first-scheduled-action \
  --schedule "cron(0 9 * * ? *)" \
  --scalable-target-action MinCapacity=1,MaxCapacity=5
```

Windows

```
aws application-autoscaling put-scheduled-action --service-namespace namespace --
scalable-dimension dimension --resource-id identifier --scheduled-action-name my-
first-scheduled-action --schedule "cron(0 9 * * ? *)" --scalable-target-action
 MinCapacity=1,MaxCapacity=5
```

This command does not return any output if it is successful.

2. To confirm that your scheduled action exists, use the following describe-scheduled-actions command.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scheduled-actions \
  --service-namespace namespace \
  --query 'ScheduledActions[?ResourceId==`identifier`]'
```

Windows

```
aws application-autoscaling describe-scheduled-actions --service-
namespace namespace --query "ScheduledActions[?ResourceId==`identifier`]"
```

The following is example output.

```
[
    {
        "ScheduledActionName": "my-first-scheduled-action",
        "ScheduledActionARN": "arn",
        "Schedule": "cron(0 9 * * ? *)",
        "ScalableTargetAction": {
            "MinCapacity": 1,
```

```
            "MaxCapacity": 5
        },
        ...
    }
]
```

To create a scheduled action that scales in at night

1. Repeat the preceding procedure to create another scheduled action that Application Auto Scaling uses to scale in at the end of the day.

   On the specified schedule (every day at 8:00 PM UTC), Application Auto Scaling updates the target's `MinCapacity` and `MaxCapacity` to 0, as instructed by the following put-scheduled-action command.

   Linux, macOS, or Unix

   ```
   aws application-autoscaling put-scheduled-action \
     --service-namespace namespace \
     --scalable-dimension dimension \
     --resource-id identifier \
     --scheduled-action-name my-second-scheduled-action \
     --schedule "cron(0 20 * * ? *)" \
     --scalable-target-action MinCapacity=0,MaxCapacity=0
   ```

   Windows

   ```
   aws application-autoscaling put-scheduled-action --service-namespace namespace --
   scalable-dimension dimension --resource-id identifier --scheduled-action-name my-
   second-scheduled-action --schedule "cron(0 20 * * ? *)" --scalable-target-action
    MinCapacity=0,MaxCapacity=0
   ```

2. To confirm that your scheduled action exists, use the following describe-scheduled-actions command.

   Linux, macOS, or Unix

   ```
   aws application-autoscaling describe-scheduled-actions \
     --service-namespace namespace \
     --query 'ScheduledActions[?ResourceId==`identifier`]'
   ```

Windows

```
aws application-autoscaling describe-scheduled-actions --service-
namespace namespace --query "ScheduledActions[?ResourceId==`identifier`]"
```

The following is example output.

```
[
    {
        "ScheduledActionName": "my-first-scheduled-action",
        "ScheduledActionARN": "arn",
        "Schedule": "cron(0 9 * * ? *)",
        "ScalableTargetAction": {
            "MinCapacity": 1,
            "MaxCapacity": 5
        },
        ...
    },
    {
        "ScheduledActionName": "my-second-scheduled-action",
        "ScheduledActionARN": "arn",
        "Schedule": "cron(0 20 * * ? *)",
        "ScalableTargetAction": {
            "MinCapacity": 0,
            "MaxCapacity": 0
        },
        ...
    }
]
```

Step 3: Add a target tracking scaling policy

Now that you have the basic schedule in place, add a target tracking scaling policy to scale based on current resource utilization.

With target tracking, Application Auto Scaling compares the target value in the policy to the current value of the specified metric. When they are unequal for a period of time, Application Auto Scaling adds or removes capacity to maintain steady performance. As the load on your application and the metric value increases, Application Auto Scaling adds capacity as fast as it can without

going above `MaxCapacity`. When Application Auto Scaling removes capacity because the load is minimal, it does so without going below `MinCapacity`. By adjusting the capacity based on usage, you only pay for what your application needs.

If the metric has insufficient data because your application does not have any load, Application Auto Scaling does not add or remove capacity. In other words, Application Auto Scaling prioritizes availability in situations where not enough information is available.

You can add multiple scaling policies, but make sure you do not add conflicting step scaling policies, which might cause undesirable behavior. For example, if the step scaling policy initiates a scale-in activity before the target tracking policy is ready to scale in, the scale-in activity will not be blocked. After the scale-in activity completes, the target tracking policy could instruct Application Auto Scaling to scale out again.

To create a target tracking scaling policy

1. Use the following put-scaling-policy command to create the policy.

   The metrics that are most frequently used for target tracking are predefined, and you can use them without supplying the full metric specification from CloudWatch. For more information about the available predefined metrics, see Target tracking scaling policies for Application Auto Scaling.

   Before you run this command, make sure that your predefined metric expects the target value. For example, to scale out when CPU reaches 50% utilization, specify a target value of 50.0. Or, to scale out Lambda provisioned concurrency when usage reaches 70% utilization, specify a target value of 0.7. For information about target values for a particular resource, refer to the documentation that is provided by the service about how to configure target tracking. For more information, see AWS services that you can use with Application Auto Scaling.

   Linux, macOS, or Unix

```
aws application-autoscaling put-scaling-policy \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier \
  --policy-name my-scaling-policy --policy-type TargetTrackingScaling \
  --target-tracking-scaling-policy-configuration '{ "TargetValue": 50.0,
 "PredefinedMetricSpecification": { "PredefinedMetricType": "predefinedmetric" }}'
```

Windows

```
aws application-autoscaling put-scaling-policy --service-namespace namespace --
scalable-dimension dimension --resource-id identifier --policy-name my-scaling-
policy --policy-type TargetTrackingScaling --target-tracking-scaling-policy-
configuration "{ \"TargetValue\": 50.0, \"PredefinedMetricSpecification\":
 { \"PredefinedMetricType\": \"predefinedmetric\" }}"
```

If successful, this command returns the ARNs and names of the two CloudWatch alarms that were created on your behalf.

2. To confirm that your scheduled action exists, use the following describe-scaling-policies command.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-policies --service-namespace namespace
 \
  --query 'ScalingPolicies[?ResourceId==`identifier`]'
```

Windows

```
aws application-autoscaling describe-scaling-policies --service-namespace namespace
 --query "ScalingPolicies[?ResourceId==`identifier`]"
```

The following is example output.

```
[
    {
        "PolicyARN": "arn",
        "TargetTrackingScalingPolicyConfiguration": {
            "PredefinedMetricSpecification": {
                "PredefinedMetricType": "predefinedmetric"
            },
            "TargetValue": 50.0
        },
        "PolicyName": "my-scaling-policy",
        "PolicyType": "TargetTrackingScaling",
        "Alarms": [],
        ...
    }
```

```
]
```

Step 4: Next steps

When a scaling activity occurs, you see a record of it in the output of the scaling activities for the scalable target, for example:

```
Successfully set desired count to 1. Change successfully fulfilled by ecs.
```

To monitor your scaling activities with Application Auto Scaling, you can use the following describe-scaling-activities command.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-activities
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier
```

Windows

```
aws application-autoscaling describe-scaling-activities --service-namespace namespace
 --scalable-dimension dimension --resource-id identifier
```

Step 5: Clean up

To prevent your account from accruing charges for resources created while actively scaling, you can clean up the associated scaling configuration as follows.

Deleting the scaling configuration does not delete the underlying AWS resource. It also does not return it to its original capacity. You can use the console of the service where you created the resource to delete it or adjust its capacity.

To delete the scheduled actions

The following delete-scheduled-action command deletes a specified scheduled action. You can skip this step if you want to keep the scheduled actions that you created.

Linux, macOS, or Unix

```
aws application-autoscaling delete-scheduled-action \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier \
  --scheduled-action-name my-second-scheduled-action
```

Windows

```
aws application-autoscaling delete-scheduled-action --service-namespace namespace
 --scalable-dimension dimension --resource-id identifier --scheduled-action-name my-
second-scheduled-action
```

To delete the scaling policy

The following delete-scaling-policy command deletes a specified target tracking scaling policy. You can skip this step if you want to keep the scaling policy that you created.

Linux, macOS, or Unix

```
aws application-autoscaling delete-scaling-policy \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier \
  --policy-name my-scaling-policy
```

Windows

```
aws application-autoscaling delete-scaling-policy --service-namespace namespace --
scalable-dimension dimension --resource-id identifier --policy-name my-scaling-policy
```

To deregister the scalable target

Use the following deregister-scalable-target command to deregister the scalable target. If you have any scaling policies that you created or any scheduled actions that have not yet been deleted, they are deleted by this command. You can skip this step if you want to keep the scalable target registered for future use.

Linux, macOS, or Unix

```
aws application-autoscaling deregister-scalable-target \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier
```

Windows

```
aws application-autoscaling deregister-scalable-target --service-namespace namespace --
scalable-dimension dimension --resource-id identifier
```

Suspend and resume scaling for Application Auto Scaling

This topic explains how to suspend and then resume one or more of the scaling activities for the scalable targets in your application. The suspend-resume feature is used to temporarily pause scaling activities triggered by your scaling policies and scheduled actions. This can be useful, for example, when you don't want automatic scaling to potentially interfere while you are making a change or investigating a configuration issue. Your scaling policies and scheduled actions can be retained, and when you are ready, scaling activities can be resumed.

In the example CLI commands that follow, you pass the JSON-formatted parameters in a config.json file. You can also pass these parameters on the command line by using quotation marks to enclose the JSON data structure. For more information, see Using quotation marks with strings in the AWS CLI in the *AWS Command Line Interface User Guide*.

Contents

- Scaling activities
- Suspend and resume scaling activities

> **Note**
>
> For instructions for suspending scale-out processes while Amazon ECS deployments are in progress, see the following documentation:
> Service auto scaling and deployments in the *Amazon Elastic Container Service Developer Guide*

Scaling activities

Application Auto Scaling supports putting the following scaling activities in a suspended state:

- All scale-in activities that are triggered by a scaling policy.
- All scale-out activities that are triggered by a scaling policy.
- All scaling activities that involve scheduled actions.

The following descriptions explain what happens when individual scaling activities are suspended. Each one can be suspended and resumed independently. Depending on the reason for suspending a scaling activity, you might need to suspend multiple scaling activities together.

`DynamicScalingInSuspended`

- Application Auto Scaling does not remove capacity when a target tracking scaling policy or a step scaling policy is triggered. This allows you to temporarily disable scale-in activities associated with scaling policies without deleting the scaling policies or their associated CloudWatch alarms. When you resume scale in, Application Auto Scaling evaluates policies with alarm thresholds that are currently in breach.

`DynamicScalingOutSuspended`

- Application Auto Scaling does not add capacity when a target tracking scaling policy or a step scaling policy is triggered. This allows you to temporarily disable scale-out activities associated with scaling policies without deleting the scaling policies or their associated CloudWatch alarms. When you resume scale out, Application Auto Scaling evaluates policies with alarm thresholds that are currently in breach.

`ScheduledScalingSuspended`

- Application Auto Scaling does not initiate the scaling actions that are scheduled to run during the suspension period. When you resume scheduled scaling, Application Auto Scaling only evaluates scheduled actions whose execution time has not yet passed.

Suspend and resume scaling activities

You can suspend and resume individual scaling activities or all scaling activities for your Application Auto Scaling scalable target.

> **Note**
>
> For brevity, these examples illustrate how to suspend and resume scaling for a DynamoDB table. To specify a different scalable target, specify its namespace in `--service-namespace`, its scalable dimension in `--scalable-dimension`, and its resource ID in `--`

`resource-id`. For more information and examples for each service, see the topics in AWS services that you can use with Application Auto Scaling.

To suspend a scaling activity

Open a command-line window and use the register-scalable-target command with the `--suspended-state` option as follows.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target --service-namespace dynamodb \
  --scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table \
  --suspended-state file://config.json
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace dynamodb --
scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table --
suspended-state file://config.json
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

To only suspend scale-in activities that are triggered by a scaling policy, specify the following in config.json.

```
{
    "DynamicScalingInSuspended":true
}
```

To only suspend scale-out activities that are triggered by a scaling policy, specify the following in config.json.

```
{
```

```
    "DynamicScalingOutSuspended":true
}
```

To only suspend scaling activities that involve scheduled actions, specify the following in config.json.

```
{
    "ScheduledScalingSuspended":true
}
```

To suspend all scaling activities

Use the register-scalable-target command with the `--suspended-state` option as follows.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target --service-namespace dynamodb \
  --scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table \
  --suspended-state file://config.json
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace dynamodb --
scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table --
suspended-state file://config.json
```

This example assumes that the file config.json contains the following JSON-formatted parameters.

```
{
    "DynamicScalingInSuspended":true,
    "DynamicScalingOutSuspended":true,
    "ScheduledScalingSuspended":true
}
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

View suspended scaling activities

Use the describe-scalable-targets command to determine which scaling activities are in a suspended state for a scalable target.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scalable-targets --service-namespace dynamodb \
  --scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table
```

Windows

```
aws application-autoscaling describe-scalable-targets --service-namespace dynamodb --
scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table
```

The following is example output.

```
{
    "ScalableTargets": [
        {
            "ServiceNamespace": "dynamodb",
            "ScalableDimension": "dynamodb:table:ReadCapacityUnits",
            "ResourceId": "table/my-table",
            "MinCapacity": 1,
            "MaxCapacity": 20,
            "SuspendedState": {
                "DynamicScalingOutSuspended": true,
                "DynamicScalingInSuspended": true,
                "ScheduledScalingSuspended": true
            },
            "CreationTime": 1558125758.957,
            "RoleARN": "arn:aws:iam::123456789012:role/aws-
service-role/dynamodb.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
        }
    ]
}
```

Resume scaling activities

When you are ready to resume the scaling activity, you can resume it using the register-scalable-target command.

The following example command resumes all scaling activities for the specified scalable target.

Linux, macOS, or Unix

```
aws application-autoscaling register-scalable-target --service-namespace dynamodb \
  --scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table \
  --suspended-state file://config.json
```

Windows

```
aws application-autoscaling register-scalable-target --service-namespace dynamodb --
scalable-dimension dynamodb:table:ReadCapacityUnits --resource-id table/my-table --
suspended-state file://config.json
```

This example assumes that the file config.json contains the following JSON-formatted parameters.

```
{
    "DynamicScalingInSuspended":false,
    "DynamicScalingOutSuspended":false,
    "ScheduledScalingSuspended":false
}
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Scaling activities for Application Auto Scaling

Application Auto Scaling monitors your scaling policy's CloudWatch metrics and initiates a scaling activity when thresholds are exceeded. It also initiates scaling activities when you modify the maximum or minimum size of the scalable target, either manually or following a schedule.

When a scaling activity occurs, Application Auto Scaling does one of the following:

- Increases the capacity of the scalable target (referred to as *scaling out*)
- Decreases the capacity of the scalable target (referred to as *scaling in*)

You can look up scaling activities from the last six weeks.

Look up scaling activities by scalable target

To see the scaling activities for a specific scalable target, use the following describe-scaling-activities command.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-activities --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount --resource-id service/my-cluster/my-service
```

Windows

```
aws application-autoscaling describe-scaling-activities --service-namespace ecs --scalable-dimension ecs:service:DesiredCount --resource-id service/my-cluster/my-service
```

The following is an example response, where `StatusCode` contains the current status of the activity and `StatusMessage` contains information about the status of the scaling activity.

```
{
    "ScalingActivities": [
        {
            "ScalableDimension": "ecs:service:DesiredCount",
            "Description": "Setting desired count to 1.",
            "ResourceId": "service/my-cluster/my-service",
```

```
            "ActivityId": "e6c5f7d1-dbbb-4a3f-89b2-51f33e766399",
            "StartTime": 1462575838.171,
            "ServiceNamespace": "ecs",
            "EndTime": 1462575872.111,
            "Cause": "monitor alarm web-app-cpu-lt-25 in state ALARM triggered policy
 web-app-cpu-lt-25",
            "StatusMessage": "Successfully set desired count to 1. Change successfully
 fulfilled by ecs.",
            "StatusCode": "Successful"
        }
    ]
}
```

For a description of the fields in the response, see ScalingActivity in the *Application Auto Scaling API Reference*.

The following status codes indicate when the scaling event that leads to the scaling activity reaches a completed state:

- `Successful` – Scaling was completed successfully
- `Overridden` – The desired capacity was updated by a newer scaling event
- `Unfulfilled` – Scaling timed out or the target service cannot fulfill the request
- `Failed` – Scaling failed with an exception

> **Note**
>
> The scaling activity might also have a status of `Pending` or `InProgress`. All scaling activities have a `Pending` status before the target service responds. After the target responds, the status of the scaling activity changes to `InProgress`.

Include not scaled activities

By default, the scaling activities do not reflect times when Application Auto Scaling makes a decision about whether to not scale.

For example, suppose that an Amazon ECS service exceeds the maximum threshold of a given metric, but the number of tasks is already at the maximum number of allowed tasks. In this case, Application Auto Scaling does not scale out the desired number of tasks.

To include activities that aren't scaled (*not scaled activities*) in the response, add the `--include-not-scaled-activities` option to the describe-scaling-activities command.

Linux, macOS, or Unix

```
aws application-autoscaling describe-scaling-activities --include-not-scaled-activities
 \
  --service-namespace ecs --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service
```

Windows

```
aws application-autoscaling describe-scaling-activities --include-not-scaled-activities
 --service-namespace ecs --scalable-dimension ecs:service:DesiredCount --resource-
id service/my-cluster/my-service
```

> **Note**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

To confirm that the response includes the not scaled activities, the `NotScaledReasons` element is shown in the output for some, if not all, failed scaling activities.

```
{
    "ScalingActivities": [
        {
            "ScalableDimension": "ecs:service:DesiredCount",
            "Description": "Attempting to scale due to alarm triggered",
            "ResourceId": "service/my-cluster/my-service",
            "ActivityId": "4d759079-a31f-4d0c-8468-504c56e2eecf",
            "StartTime": 1664928867.915,
            "ServiceNamespace": "ecs",
            "Cause": "monitor alarm web-app-cpu-gt-75 in state ALARM triggered policy
 web-app-cpu-gt-75",
            "StatusCode": "Failed",
            "NotScaledReasons": [
                {
                    "Code": "AlreadyAtMaxCapacity",
                    "MaxCapacity": 4
```

```
                }
            ]
        }
    ]
}
```

For a description of the fields in the response, see ScalingActivity in the *Application Auto Scaling API Reference*.

If a not scaled activity is returned, depending on the reason code listed in `Code`, attributes like `CurrentCapacity`, `MaxCapacity`, and `MinCapacity` might be present in the response.

To prevent large quantities of duplicate entries, only the first not scaled activity will be recorded in the scaling activity history. Any subsequent not scaled activities will not generate new entries unless the reason for not scaling changes.

Reason codes

The following are the reason codes for a not scaled activity.

Reason code	Definition			
AutoScalingAnticipatedFlapping	Auto scaling algorithm decided not to take a scaling action because it would lead to *flapping*. Flapping is an infinite loop of scaling in and scaling out. That is, if a scaling action is taken, the metric value would change to start another			

Reason code	Definition			
	scaling action in the reverse direction.			
TargetServicePutResourceAsUnscalable	The target service has temporarily put the resource in an unscalable state. Application Auto Scaling will attempt to scale again when the automatic scaling conditions specified in the scaling policy are met.			
AlreadyAtMaxCapacity	Scaling is blocked by the maximum capacity that you specified. If you want Application Auto Scaling to scale out, you need to increase the maximum capacity.			

Reason code	Definition			
AlreadyAtMinCapacity	Scaling is blocked by the minimum capacity that you specified. If you want Application Auto Scaling to scale in, you need to decrease the minimum capacity.			
AlreadyAtDesiredCapacity	Auto scaling algorithm calculated the revised capacity to be equal to the current capacity.			

Monitoring Application Auto Scaling

Monitoring is an important part of maintaining the reliability, availability, and performance of Application Auto Scaling and your other AWS solutions. You should collect monitoring data from all parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides monitoring tools to watch Application Auto Scaling, report when something is wrong, and take automatic actions when appropriate.

You can use the following features to help you manage your AWS resources:

AWS CloudTrail

With AWS CloudTrail, you can track the calls made to the Application Auto Scaling API by or on behalf of your AWS account. CloudTrail stores the information in log files in the Amazon S3 bucket that you specify. You can identify which users and accounts called Application Auto Scaling, the source IP address from which the calls were made, and when the calls occurred. For more information, see Log Application Auto Scaling API calls using AWS CloudTrail.

> **Note**
>
> For information about other AWS services that can help you log and collect data about your workloads, see the Logging and monitoring guide for application owners guide in the *AWS Prescriptive Guidance*.

Amazon CloudWatch

Amazon CloudWatch helps you analyze logs and, in real time, monitor the metrics of your AWS resources and hosted applications. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track resource utilization and notify you when utilization is very high or when the metric's alarm has gone into the `INSUFFICIENT_DATA` state. For more information, see Monitor usage of scalable resources using CloudWatch.

CloudWatch also tracks AWS API usage metrics for Application Auto Scaling. You can use these metrics to configure alarms that alert you when your API call volume violates a threshold that you define. For more information, see AWS usage metrics in the *Amazon CloudWatch User Guide*.

Amazon EventBridge

Amazon EventBridge is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This lets you monitor events that happen in services, and build event-driven architectures. For more information, see Monitor Application Auto Scaling events using Amazon EventBridge.

AWS Health Dashboard

The Health Dashboard (PHD) displays information, and also provides notifications that are invoked by changes in the health of AWS resources. The information is presented in two ways: on a dashboard that shows recent and upcoming events organized by category, and in a full event log that shows all events from the past 90 days. For more information, see Getting started with your Health Dashboard.

Monitor usage of scalable resources using CloudWatch

With Amazon CloudWatch, you get nearly continuous visibility into your applications across scalable resources. CloudWatch is a monitoring service for AWS resources. You can use CloudWatch to collect and track metrics, set alarms, and automatically react to changes in your AWS resources. You can also create dashboards to monitor the specific metrics or sets of metrics you need.

When you interact with the services that integrate with Application Auto Scaling, they send the metrics shown in the following table to CloudWatch. In CloudWatch, metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace. These metrics can help you monitor resource usage and plan capacity for your applications. If your application's workload is not constant, this indicates that you should consider using auto scaling. For detailed descriptions of these metrics, see the documentation for the metric of interest.

Contents

- CloudWatch metrics for monitoring resource usage
- Predefined metrics for target tracking scaling policies
- Predictive scaling metrics and dimensions

CloudWatch metrics for monitoring resource usage

The following table lists the CloudWatch metrics that are available to support monitoring resource usage. The list is not exhaustive but will give you a good starting point. If you do not see these metrics in the CloudWatch console, make sure that you have completed the set up of the resource. For more information, see the Amazon CloudWatch User Guide.

Scalable resource	Namespace	CloudWatch metric	Link to documentation
WorkSpaces Applications			
Fleets	AWS/AppStream	Name: AvailableCapacity Dimension: Fleet	WorkSpaces Applications metrics
Fleets	AWS/AppStream	Name: CapacityUtilization Dimension: Fleet	WorkSpaces Applications metrics
Aurora			
Replicas	AWS/RDS	Name: CPUUtilization Dimensions: DBClusterIdentifier, Role (READER)	Aurora cluster-level metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Replicas	AWS/RDS	Name: DatabaseConnections Dimensions: DBClusterIdentifier, Role (READER)	Aurora cluster-level metrics
Amazon Comprehend			
Document classification endpoints	AWS/Comprehend	Name: InferenceUtilization Dimension: EndpointArn	Amazon Comprehend endpoint metrics
Entity recognizer endpoints	AWS/Comprehend	Name: InferenceUtilization Dimension: EndpointArn	Amazon Comprehend endpoint metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
DynamoDB			
Tables and global secondary indexes	AWS/DynamoDB	Name: ProvisionedReadCapacityUnits Dimensions: TableName, GlobalSecondaryIndexName	DynamoDB metrics
Tables and global secondary indexes	AWS/DynamoDB	Name: ProvisionedWriteCapacityUnits Dimensions: TableName, GlobalSecondaryIndexName	DynamoDB metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Tables and global secondary indexes	AWS/DynamoDB	Name: ConsumedReadCapacityUnits Dimensions: TableName, GlobalSecondaryIndexName	DynamoDB metrics
Tables and global secondary indexes	AWS/DynamoDB	Name: ConsumedWriteCapacityUnits Dimensions: TableName, GlobalSecondaryIndexName	DynamoDB metrics
Amazon ECS			

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Services	AWS/ECS	Name: CPUUtilization Dimensions: ClusterName, ServiceName	Amazon ECS metrics
Services	AWS/ECS	Name: MemoryUtilization Dimensions: ClusterName, ServiceName	Amazon ECS metrics
Services	AWS/ApplicationELB	Name: RequestCountPerTarget Dimension: TargetGroup	Application Load Balancer metrics
ElastiCache			

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Clusters (replication groups)	AWS/ElastiCache	Name: DatabaseMemoryUsageCountedForEvictPercentage Dimension: ReplicationGroupId	ElastiCache Valkey and Redis OSS metrics
Clusters (replication groups)	AWS/ElastiCache	Name: DatabaseCapacityUsageCountedForEvictPercentage Dimension: ReplicationGroupId	ElastiCache Valkey and Redis OSS metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Clusters (replication groups)	AWS/ElastiCache	Name: EngineCPUUtilization Dimensions: ReplicationGroupId, Role (Primary)	ElastiCache Valkey and Redis OSS metrics
Clusters (replication groups)	AWS/ElastiCache	Name: EngineCPUUtilization Dimensions: ReplicationGroupId, Role (Replica)	ElastiCache Valkey and Redis OSS metrics
Clusters (cache)	AWS/ElastiCache	Name: EngineCPUUtilization Dimensions: CacheClusterId, Node	ElastiCache Memcached metrics

Scalable resource	Namespac	CloudWatch metric	Link to documentation
Clusters (cache)	AWS/ElastiCache	Name: DatabaseCapacityMemoryUsagePercentage Dimensions: CacheClusterId	ElastiCache Memcached metrics
Amazon EMR			
Clusters	AWS/ElasticMapReduce	Name: YARNMemoryAvailablePercentage Dimension: ClusterId	Amazon EMR metrics
Amazon Keyspaces			

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Tables	AWS/Cassandra	Name: ProvisionedReadCapacityUnits Dimensions: Keyspace, TableName	Amazon Keyspaces metrics
Tables	AWS/Cassandra	Name: ProvisionedWriteCapacityUnits Dimensions: Keyspace, TableName	Amazon Keyspaces metrics
Tables	AWS/Cassandra	Name: ConsumedReadCapacityUnits Dimensions: Keyspace, TableName	Amazon Keyspaces metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Tables	AWS/Cassandra	Name: ConsumedWriteCapacityUnits Dimensions: Keyspace, TableName	Amazon Keyspaces metrics
Lambda			
Provisioned concurrency	AWS/Lambda	Name: ProvisionedConcurrencyUtilization Dimensions: FunctionName, Resource	Lambda function metrics
Amazon MSK			
Broker storage	AWS/Kafka	Name: KafkaDataLogsDiskUsed Dimensions: Cluster Name	Amazon MSK metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Broker storage	AWS/Kafka	Name: KafkaDataLogsDiskUsed Dimensions: Cluster Name, Broker ID	Amazon MSK metrics
Neptune			
Clusters	AWS/Neptune	Name: CPUUtilization Dimensions: DBClusterIdentifier, Role (READER)	Neptune metrics
SageMaker AI			

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Endpoint variants	AWS/SageMaker	Name: InvocationsPerInstance Dimensions: EndpointName, VariantName	Invocation metrics
Inference components	AWS/SageMaker	Name: InvocationsPerCopy Dimensions: InferenceComponentName	Invocation metrics
Provisioned concurrency for a serverless endpoint	AWS/SageMaker	Name: ServerlessProvisionedConcurrencyUtilization Dimensions: EndpointName, VariantName	Serverless endpoint metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Spot Fleet (Amazon EC2)			
Spot Fleets	AWS/EC2Spot	Name: CPUUtilization Dimension: FleetRequestId	Spot Fleet metrics
Spot Fleets	AWS/EC2Spot	Name: NetworkIn Dimension: FleetRequestId	Spot Fleet metrics
Spot Fleets	AWS/EC2Spot	Name: NetworkOut Dimension: FleetRequestId	Spot Fleet metrics

Scalable resource	Namespace	CloudWatch metric	Link to documentation
Spot Fleets	AWS/ApplicationELB	Name: RequestCountPerTarget Dimension: TargetGroup	Application Load Balancer metrics

Predefined metrics for target tracking scaling policies

The following table lists the predefined metric types from the Application Auto Scaling API Reference with their corresponding CloudWatch metric name. Each predefined metric represents an aggregation of the values of the underlying CloudWatch metric. The result is the average resource usage over a one-minute period, based on a percentage unless otherwise noted. The predefined metrics are only used within the context of setting up target tracking scaling policies.

You can find more information about these metrics in the service's documentation that's available from the table in CloudWatch metrics for monitoring resource usage.

Predefined metric type	CloudWatch metric name
WorkSpaces Applications	
`AppStreamAverageCapacityUtilization`	CapacityUtilization
Aurora	
`RDSReaderAverageCPUUtilization`	CPUUtilization
`RDSReaderAverageDatabaseConnections`	DatabaseConnections[1]

Predefined metric type	CloudWatch metric name
Amazon Comprehend	
`ComprehendInferenceUtilization`	InferenceUtilization
DynamoDB	
`DynamoDBReadCapacityUtilization`	ProvisionedReadCapacityUnits, ConsumedReadCapacityUnits[2]
`DynamoDBWriteCapacityUtilization`	ProvisionedWriteCapacityUnits, ConsumedWriteCapacityUnits[2]
Amazon ECS	
`ECSServiceAverageCPUUtilization`	CPUUtilization
`ECSServiceAverageMemoryUtilization`	MemoryUtilization
`ALBRequestCountPerTarget`	RequestCountPerTarget[1]
ElastiCache	
`ElastiCacheDatabaseMemoryUsageCountedForEvictPercentage`	DatabaseMemoryUsageCountedForEvictPercentage
`ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage`	DatabaseCapacityUsageCountedForEvictPercentage
`ElastiCachePrimaryEngineCPUUtilization`	EngineCPUUtilization
`ElastiCacheReplicaEngineCPUUtilization`	EngineCPUUtilization
`ElastiCacheEngineCPUUtilization`	EngineCPUUtilization

Predefined metric type	CloudWatch metric name
`ElastiCacheDatabaseMemoryUsagePercentage`	DatabaseMemoryUsagePercentage
Amazon Keyspaces	
`CassandraReadCapacityUtilization`	ProvisionedReadCapacityUnits, ConsumedReadCapacityUnits[2]
`CassandraWriteCapacityUtilization`	ProvisionedWriteCapacityUnits, ConsumedWriteCapacityUnits[2]
Lambda	
`LambdaProvisionedConcurrencyUtilization`	ProvisionedConcurrencyUtilization
Amazon MSK	
`KafkaBrokerStorageUtilization`	KafkaDataLogsDiskUsed
Neptune	
`NeptuneReaderAverageCPUUtilization`	CPUUtilization
SageMaker AI	
`SageMakerVariantInvocationsPerInstance`	InvocationsPerInstance[1]
`SageMakerInferenceComponentInvocationsPerCopy`	InvocationsPerCopy[1]
`SageMakerVariantProvisionedConcurrencyUtilization`	ServerlessProvisionedConcurrencyUtilization

Predefined metric type	CloudWatch metric name
SageMakerInferenceComponentConcurrentRequestsPerCopyHighResolution	ConcurrentRequestsPerCopy
SageMakerVariantConcurrentRequestsPerModelHighResolution	ConcurrentRequestsPerModel
Spot Fleet	
EC2SpotFleetRequestAverageCPUUtilization	CPUUtilization[3]
EC2SpotFleetRequestAverageNetworkIn[3]	NetworkIn[1] [3]
EC2SpotFleetRequestAverageNetworkOut[3]	NetworkOut[1] [3]
ALBRequestCountPerTarget	RequestCountPerTarget[1]

[1] Metric is based on a count instead of a percentage.

[2] For DynamoDB and Amazon Keyspaces, the predefined metrics are an aggregation of two CloudWatch metrics to support scaling based on provisioned throughput consumption.

[3] For best scaling performance, Amazon EC2 detailed monitoring should be used.

Predictive scaling metrics and dimensions

The AWS/ApplicationAutoScaling namespace includes the following metrics for predictive scaling policies. These metrics are available with a resolution of one hour and can help you evaluate forecast accuracy by comparing forecasted values with actual values.

Metric	Description	Dimensions
PredictiveScalingLoadForecast	The amount of load that's anticipated to be generated by your application. The Average, Minimum, and Maximum statistics are useful, but the Sum statistic is not. **Reporting criteria**: Reported after the initial forecast is created.	ResourceId, ServiceNamespace, PolicyName, ScalableDimension, PairIndex
PredictiveScalingCapacityForecast	The anticipated amount of capacity needed to meet application demand. This is based on the load forecast and target utilization level at which you want to maintain your Application Auto Scaling resources. The Average, Minimum, and Maximum statistics are useful, but the Sum statistic is not. **Reporting criteria**: Reported after the initial forecast is created.	ResourceId, ServiceNamespace, PolicyName, ScalableDimension
PredictiveScalingMetricPairCorrelation	The correlation between the scaling metric and the per-instance average of the load metric. Predictive scaling assumes high correlation. Therefore, if you observe low value for this metric, it's better not to use a metric pair. The Average, Minimum, and Maximum statistics are useful, but the Sum statistic is not. **Reporting criteria**: Reported after the initial forecast is created.	ResourceId, ServiceNamespace, PolicyName, ScalableDimension, PairIndex

Log Application Auto Scaling API calls using AWS CloudTrail

Application Auto Scaling is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls for Application Auto Scaling as events. The calls captured include calls from the AWS Management Console and code calls to the Application Auto Scaling API operations. Using the information collected by CloudTrail, you can determine the request that was made to Application Auto Scaling, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For

more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

Application Auto Scaling management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

Application Auto Scaling logs all Application Auto Scaling control plane operations as management events. For a list of the Application Auto Scaling control plane operations that Application Auto Scaling logs to CloudTrail, see the Application Auto Scaling API Reference.

Application Auto Scaling event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail event that demonstrates the `DescribeScalableTargets` operation.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-21T17:05:42Z"
            }
        }
    },
    "eventTime": "2018-08-16T23:20:32Z",
    "eventSource": "autoscaling.amazonaws.com",
```

```
    "eventName": "DescribeScalableTargets",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "72.21.196.68",
    "userAgent": "EC2 Spot Console",
    "requestParameters": {
        "serviceNamespace": "ec2",
        "scalableDimension": "ec2:spot-fleet-request:TargetCapacity",
        "resourceIds": [
            "spot-fleet-request/sfr-05ceaf79-3ba2-405d-e87b-612857f1357a"
        ]
    },
    "responseElements": null,
    "additionalEventData": {
        "service": "application-autoscaling"
    },
    "requestID": "0737e2ea-fb2d-11e3-bfd8-99133058e7bb",
    "eventID": "3fcfb182-98f8-4744-bd45-b38835ab61cb",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Application Auto Scaling RemoveAction calls on CloudWatch

Your AWS CloudTrail log might show that Application Auto Scaling calls the CloudWatch `RemoveAction` API when Application Auto Scaling instructs CloudWatch to remove the automatic scaling action from an alarm. This could happen if you deregister a scalable target, delete a scaling policy, or if an alarm invokes a nonexistent scaling policy.

Monitor Application Auto Scaling events using Amazon EventBridge

Amazon EventBridge, formerly called CloudWatch Events, helps you monitor events that are specific to Application Auto Scaling and initiate target actions that use other AWS services. Events from AWS services are delivered to EventBridge in near real time.

Using EventBridge, you can create *rules* that match incoming *events* and route them to *targets* for processing.

For more information, see Getting started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

Application Auto Scaling events

The following examples show events for Application Auto Scaling. Events are produced on a best-effort basis.

Only events that are specific to scaled to max and API calls via CloudTrail are currently available for Application Auto Scaling.

Event types

- Event for state change: scaled to max
- Events for API calls via CloudTrail

Event for state change: scaled to max

The following example event shows that Application Auto Scaling increased (scaled out) the capacity of the scalable target to its maximum size limit. If demand increases again, Application Auto Scaling will be prevented from scaling the target to a larger size because it is already scaled to its maximum size.

In the `detail` object, the values for the `resourceId`, `serviceNamespace`, and `scalableDimension` attributes identify the scalable target. The values for the `newDesiredCapacity` and `oldDesiredCapacity` attributes refer to the new capacity after the scale-out event and the original capacity before the scale-out event. The `maxCapacity` is the maximum size limit of the scalable target.


```
{
  "version": "0",
  "id": "11112222-3333-4444-5555-666677778888",
  "detail-type": "Application Auto Scaling Scaling Activity State Change",
  "source": "aws.application-autoscaling",
  "account": "123456789012",
  "time": "2019-06-12T10:23:40Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "startTime": "2022-06-12T10:20:43Z",
```

```
    "endTime": "2022-06-12T10:23:40Z",
    "newDesiredCapacity": 8,
    "oldDesiredCapacity": 5,
    "minCapacity": 2,
    "maxCapacity": 8,
    "resourceId": "table/my-table",
    "scalableDimension": "dynamodb:table:WriteCapacityUnits",
    "serviceNamespace": "dynamodb",
    "statusCode": "Successful",
    "scaledToMax": true,
    "direction": "scale-out"
}
```

To create a rule that captures all `scaledToMax` state change events for all scalable targets, use the following sample event pattern.

```
{
  "source": [
    "aws.application-autoscaling"
  ],
  "detail-type": [
    "Application Auto Scaling Scaling Activity State Change"
  ],
  "detail": {
    "scaledToMax": [
      true
    ]
  }
}
```

Events for API calls via CloudTrail

A *trail* is a configuration that AWS CloudTrail uses to deliver events as log files to an Amazon S3 bucket. CloudTrail log files contain log entries. An event represents a log entry, and it includes information about the requested action, the date and time of the action, and request parameters. To learn how to get started with CloudTrail, see Creating a trail in the *AWS CloudTrail User Guide*.

Events that are delivered via CloudTrail have `AWS API Call via CloudTrail` as the value for `detail-type`.

The following example event represents a CloudTrail log file entry that shows that a console user called the Application Auto Scaling RegisterScalableTarget action.

```
{
  "version": "0",
  "id": "99998888-7777-6666-5555-444433332222",
  "detail-type": "AWS API Call via CloudTrail",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2022-07-13T16:50:15Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "eventVersion": "1.08",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "123456789012",
      "arn": "arn:aws:iam::123456789012:user/Bob",
      "accountId": "123456789012",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "sessionContext": {
        "sessionIssuer": {
          "type": "Role",
          "principalId": "123456789012",
          "arn": "arn:aws:iam::123456789012:role/Admin",
          "accountId": "123456789012",
          "userName": "Admin"
        },
        "webIdFederationData": {},
        "attributes": {
          "creationDate": "2022-07-13T15:17:08Z",
          "mfaAuthenticated": "false"
        }
      }
    },
    "eventTime": "2022-07-13T16:50:15Z",
    "eventSource": "autoscaling.amazonaws.com",
    "eventName": "RegisterScalableTarget",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "EC2 Spot Console",
    "requestParameters": {
      "resourceId": "spot-fleet-request/sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE",
      "serviceNamespace": "ec2",
      "scalableDimension": "ec2:spot-fleet-request:TargetCapacity",
      "minCapacity": 2,
```

```
      "maxCapacity": 10
    },
    "responseElements": null,
    "additionalEventData": {
      "service": "application-autoscaling"
    },
    "requestID": "e9caf887-8d88-11e5-a331-3332aa445952",
    "eventID": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
  }
}
```

To create a rule based on all DeleteScalingPolicy and DeregisterScalableTarget API calls for all scalable targets, use the following sample event pattern:

```
{
  "source": [
    "aws.autoscaling"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "autoscaling.amazonaws.com"
    ],
    "eventName": [
       "DeleteScalingPolicy",
       "DeregisterScalableTarget"
    ],
    "additionalEventData": {
      "service": [
        "application-autoscaling"
      ]
    }
  }
}
```

For more information about using CloudTrail, see Log Application Auto Scaling API calls using AWS CloudTrail.

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	AWS Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to this service, see Code examples for Application Auto Scaling using AWS SDKs.

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Code examples for Application Auto Scaling using AWS SDKs

The following code examples show how to use Application Auto Scaling with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for Application Auto Scaling using AWS SDKs
  - Actions for Application Auto Scaling using AWS SDKs
    - Use DeleteScalingPolicy with an AWS SDK or CLI
    - Use DeleteScheduledAction with a CLI
    - Use DeregisterScalableTarget with a CLI
    - Use DescribeScalableTargets with a CLI
    - Use DescribeScalingActivities with a CLI
    - Use DescribeScalingPolicies with an AWS SDK or CLI
    - Use DescribeScheduledActions with a CLI
    - Use PutScalingPolicy with a CLI
    - Use PutScheduledAction with a CLI
    - Use RegisterScalableTarget with an AWS SDK or CLI

Basic examples for Application Auto Scaling using AWS SDKs

The following code examples show how to use the basics of Application Auto Scaling with AWS SDKs.

Examples

- Actions for Application Auto Scaling using AWS SDKs

- Use DeleteScalingPolicy with an AWS SDK or CLI
- Use DeleteScheduledAction with a CLI
- Use DeregisterScalableTarget with a CLI
- Use DescribeScalableTargets with a CLI
- Use DescribeScalingActivities with a CLI
- Use DescribeScalingPolicies with an AWS SDK or CLI
- Use DescribeScheduledActions with a CLI
- Use PutScalingPolicy with a CLI
- Use PutScheduledAction with a CLI
- Use RegisterScalableTarget with an AWS SDK or CLI

Actions for Application Auto Scaling using AWS SDKs

The following code examples demonstrate how to perform individual Application Auto Scaling actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the Application Auto Scaling API Reference.

Examples

- Use DeleteScalingPolicy with an AWS SDK or CLI
- Use DeleteScheduledAction with a CLI
- Use DeregisterScalableTarget with a CLI
- Use DescribeScalableTargets with a CLI
- Use DescribeScalingActivities with a CLI
- Use DescribeScalingPolicies with an AWS SDK or CLI
- Use DescribeScheduledActions with a CLI
- Use PutScalingPolicy with a CLI
- Use PutScheduledAction with a CLI
- Use RegisterScalableTarget with an AWS SDK or CLI

Use `DeleteScalingPolicy` with an AWS SDK or CLI

The following code examples show how to use `DeleteScalingPolicy`.

CLI

AWS CLI

To delete a scaling policy

This example deletes a scaling policy for the Amazon ECS service web-app running in the default cluster.

Command:

```
aws application-autoscaling delete-scaling-policy --policy-name web-app-cpu-lt-25
 --scalable-dimension ecs:service:DesiredCount --resource-id service/default/web-
app --service-namespace ecs
```

- For API details, see DeleteScalingPolicy in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExcep
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReq
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequ
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsResp
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequ
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesResp
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

            """;
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
```

```
            .build();

        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
DeleteScalingPolicyRequest.builder()
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
            .build();
```

```
        DescribeScalingPoliciesResponse response =
 appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
 DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

- For API details, see DeleteScalingPolicy in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet deletes the specified scaling policy for an Application Auto Scaling scalable target.

```
Remove-AASScalingPolicy -ServiceNamespace AppStream -PolicyName "default-scale-
out" -ResourceId fleet/Test -ScalableDimension appstream:fleet:DesiredCapacity
```

- For API details, see DeleteScalingPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet deletes the specified scaling policy for an Application Auto Scaling scalable target.

```
Remove-AASScalingPolicy -ServiceNamespace AppStream -PolicyName "default-scale-
out" -ResourceId fleet/Test -ScalableDimension appstream:fleet:DesiredCapacity
```

- For API details, see DeleteScalingPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteScheduledAction` with a CLI

The following code examples show how to use `DeleteScheduledAction`.

CLI

AWS CLI

To delete a scheduled action

The following `delete-scheduled-action` example deletes the specified scheduled action from the specified Amazon AppStream 2.0 fleet:

```
aws application-autoscaling delete-scheduled-action \
```

```
    --service-namespace appstream \
    --scalable-dimension appstream:fleet:DesiredCapacity \
    --resource-id fleet/sample-fleet \
    --scheduled-action-name my-recurring-action
```

This command produces no output.

For more information, see Scheduled Scaling in the *Application Auto Scaling User Guide*.

- For API details, see DeleteScheduledAction in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet deletes the specified scheduled action for an Application Auto Scaling scalable target.

```
Remove-AASScheduledAction -ServiceNamespace AppStream -ScheduledActionName
 WeekDaysFleetScaling -ResourceId fleet/MyFleet -ScalableDimension
 appstream:fleet:DesiredCapacity
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-AASScheduledAction (DeleteScheduledAction)" on
 target "WeekDaysFleetScaling".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"): Y
```

- For API details, see DeleteScheduledAction in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet deletes the specified scheduled action for an Application Auto Scaling scalable target.

```
Remove-AASScheduledAction -ServiceNamespace AppStream -ScheduledActionName
 WeekDaysFleetScaling -ResourceId fleet/MyFleet -ScalableDimension
 appstream:fleet:DesiredCapacity
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-AASScheduledAction (DeleteScheduledAction)" on
 target "WeekDaysFleetScaling".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"): Y
```

- For API details, see DeleteScheduledAction in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeregisterScalableTarget` with a CLI

The following code examples show how to use `DeregisterScalableTarget`.

CLI

AWS CLI

To deregister a scalable target

This example deregisters a scalable target for an Amazon ECS service called web-app that is running in the default cluster.

Command:

```
aws application-autoscaling deregister-scalable-target --service-namespace ecs --
scalable-dimension ecs:service:DesiredCount --resource-id service/default/web-app
```

This example deregisters a scalable target for a custom resource. The custom-resource-id.txt file contains a string that identifies the Resource ID, which, for a custom resource, is the path to the custom resource through your Amazon API Gateway endpoint.

Command:

```
aws application-autoscaling deregister-scalable-target --
service-namespace custom-resource --scalable-dimension custom-
resource:ResourceType:Property --resource-id file://~/custom-resource-id.txt
```

Contents of custom-resource-id.txt file:

```
https://example.execute-api.us-west-2.amazonaws.com/prod/
scalableTargetDimensions/1-23456789
```

- For API details, see DeregisterScalableTarget in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet deregisters an Application Auto Scaling scalable target.Deregistering a scalable target deletes the scaling policies that are associated with it.

```
Remove-AASScalableTarget -ResourceId fleet/MyFleet -ScalableDimension
 appstream:fleet:DesiredCapacity -ServiceNamespace AppStream
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-AASScalableTarget (DeregisterScalableTarget)" on
 target "fleet/MyFleet".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"): Y
```

- For API details, see DeregisterScalableTarget in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet deregisters an Application Auto Scaling scalable target.Deregistering a scalable target deletes the scaling policies that are associated with it.

```
Remove-AASScalableTarget -ResourceId fleet/MyFleet -ScalableDimension
 appstream:fleet:DesiredCapacity -ServiceNamespace AppStream
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-AASScalableTarget (DeregisterScalableTarget)" on
 target "fleet/MyFleet".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"): Y
```

- For API details, see DeregisterScalableTarget in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScalableTargets` with a CLI

The following code examples show how to use `DescribeScalableTargets`.

CLI

AWS CLI

To describe scalable targets

The following `describe-scalable-targets` example describes the scalable targets for the `ecs` service namespace.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace ecs
```

Output:

```
{
    "ScalableTargets": [
        {
            "ServiceNamespace": "ecs",
```

```
            "ScalableDimension": "ecs:service:DesiredCount",
            "ResourceId": "service/default/web-app",
            "MinCapacity": 1,
            "MaxCapacity": 10,
            "RoleARN": "arn:aws:iam::123456789012:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService",
            "CreationTime": 1462558906.199,
            "SuspendedState": {
                "DynamicScalingOutSuspended": false,
                "ScheduledScalingSuspended": false,
                "DynamicScalingInSuspended": false
            },
            "ScalableTargetARN": "arn:aws:application-autoscaling:us-
west-2:123456789012:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
        }
    ]
}
```

For more information, see AWS services that you can use with Application Auto Scaling in the *Application Auto Scaling User Guide*.

- For API details, see DescribeScalableTargets in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example will provide information about the Application Autoscaling Scalable targets in the specified namespace.

```
Get-AASScalableTarget -ServiceNamespace "AppStream"
```

Output:

```
CreationTime      : 11/7/2019 2:30:03 AM
MaxCapacity       : 5
MinCapacity       : 1
ResourceId        : fleet/Test
RoleARN           : arn:aws:iam::012345678912:role/aws-
service-role/appstream.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_AppStreamFleet
```

```
ScalableDimension : appstream:fleet:DesiredCapacity
ServiceNamespace  : appstream
SuspendedState    : Amazon.ApplicationAutoScaling.Model.SuspendedState
```

- For API details, see DescribeScalableTargets in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example will provide information about the Application Autoscaling Scalable targets in the specified namespace.

```
Get-AASScalableTarget -ServiceNamespace "AppStream"
```

Output:

```
CreationTime      : 11/7/2019 2:30:03 AM
MaxCapacity       : 5
MinCapacity       : 1
ResourceId        : fleet/Test
RoleARN           : arn:aws:iam::012345678912:role/aws-
service-role/appstream.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_AppStreamFleet
ScalableDimension : appstream:fleet:DesiredCapacity
ServiceNamespace  : appstream
SuspendedState    : Amazon.ApplicationAutoScaling.Model.SuspendedState
```

- For API details, see DescribeScalableTargets in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScalingActivities` with a CLI

The following code examples show how to use `DescribeScalingActivities`.

CLI

AWS CLI

Example 1: To describe scaling activities for the specified Amazon ECS service

The following `describe-scaling-activities` example describes the scaling activities for an Amazon ECS service called `web-app` that is running in the `default` cluster. The output shows a scaling activity initiated by a scaling policy.

```
aws application-autoscaling describe-scaling-activities \
    --service-namespace ecs \
    --resource-id service/default/web-app
```

Output:

```
{
    "ScalingActivities": [
        {
            "ScalableDimension": "ecs:service:DesiredCount",
            "Description": "Setting desired count to 1.",
            "ResourceId": "service/default/web-app",
            "ActivityId": "e6c5f7d1-dbbb-4a3f-89b2-51f33e766399",
            "StartTime": 1462575838.171,
            "ServiceNamespace": "ecs",
            "EndTime": 1462575872.111,
            "Cause": "monitor alarm web-app-cpu-lt-25 in state ALARM triggered
 policy web-app-cpu-lt-25",
            "StatusMessage": "Successfully set desired count to 1. Change
 successfully fulfilled by ecs.",
            "StatusCode": "Successful"
        }
    ]
}
```

For more information, see Scaling activities for Application Auto Scaling in the *Application Auto Scaling User Guide*.

Example 2: To describe scaling activities for the specified DynamoDB table

The following `describe-scaling-activities` example describes the scaling activities for a DynamoDB table called `TestTable`. The output shows scaling activities initiated by two different scheduled actions.

```
aws application-autoscaling describe-scaling-activities \
    --service-namespace dynamodb \
    --resource-id table/TestTable
```

Output:

```
{
    "ScalingActivities": [
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting write capacity units to 10.",
            "ResourceId": "table/my-table",
            "ActivityId": "4d1308c0-bbcf-4514-a673-b0220ae38547",
            "StartTime": 1561574415.086,
            "ServiceNamespace": "dynamodb",
            "EndTime": 1561574449.51,
            "Cause": "maximum capacity was set to 10",
            "StatusMessage": "Successfully set write capacity units to 10. Change
 successfully fulfilled by dynamodb.",
            "StatusCode": "Successful"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting min capacity to 5 and max capacity to 10",
            "ResourceId": "table/my-table",
            "ActivityId": "f2b7847b-721d-4e01-8ef0-0c8d3bacc1c7",
            "StartTime": 1561574414.644,
            "ServiceNamespace": "dynamodb",
            "Cause": "scheduled action name my-second-scheduled-action was
 triggered",
            "StatusMessage": "Successfully set min capacity to 5 and max capacity
 to 10",
            "StatusCode": "Successful"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting write capacity units to 15.",
            "ResourceId": "table/my-table",
            "ActivityId": "d8ea4de6-9eaa-499f-b466-2cc5e681ba8b",
```

```
            "StartTime": 1561574108.904,
            "ServiceNamespace": "dynamodb",
            "EndTime": 1561574140.255,
            "Cause": "minimum capacity was set to 15",
            "StatusMessage": "Successfully set write capacity units to 15. Change
 successfully fulfilled by dynamodb.",
            "StatusCode": "Successful"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Description": "Setting min capacity to 15 and max capacity to 20",
            "ResourceId": "table/my-table",
            "ActivityId": "3250fd06-6940-4e8e-bb1f-d494db7554d2",
            "StartTime": 1561574108.512,
            "ServiceNamespace": "dynamodb",
            "Cause": "scheduled action name my-first-scheduled-action was
 triggered",
            "StatusMessage": "Successfully set min capacity to 15 and max
 capacity to 20",
            "StatusCode": "Successful"
        }
    ]
}
```

For more information, see Scaling activities for Application Auto Scaling in the *Application Auto Scaling User Guide*.

- For API details, see DescribeScalingActivities in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Provides descriptive information about the scaling activities in the specified namespace from the previous six weeks.

```
Get-AASScalingActivity -ServiceNamespace AppStream
```

Output:

```
ActivityId        : 2827409f-b639-4cdb-a957-8055d5d07434
```

```
Cause             : monitor alarm Appstream2-MyFleet-default-scale-in-Alarm in
 state ALARM triggered policy default-scale-in
Description       : Setting desired capacity to 2.
Details           :
EndTime           : 12/14/2019 11:32:49 AM
ResourceId        : fleet/MyFleet
ScalableDimension : appstream:fleet:DesiredCapacity
ServiceNamespace  : appstream
StartTime         : 12/14/2019 11:32:14 AM
StatusCode        : Successful
StatusMessage     : Successfully set desired capacity to 2. Change successfully
 fulfilled by appstream.
```

- For API details, see DescribeScalingActivities in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Provides descriptive information about the scaling activities in the specified namespace from the previous six weeks.

```
Get-AASScalingActivity -ServiceNamespace AppStream
```

Output:

```
ActivityId        : 2827409f-b639-4cdb-a957-8055d5d07434
Cause             : monitor alarm Appstream2-MyFleet-default-scale-in-Alarm in
 state ALARM triggered policy default-scale-in
Description       : Setting desired capacity to 2.
Details           :
EndTime           : 12/14/2019 11:32:49 AM
ResourceId        : fleet/MyFleet
ScalableDimension : appstream:fleet:DesiredCapacity
ServiceNamespace  : appstream
StartTime         : 12/14/2019 11:32:14 AM
StatusCode        : Successful
StatusMessage     : Successfully set desired capacity to 2. Change successfully
 fulfilled by appstream.
```

- For API details, see DescribeScalingActivities in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScalingPolicies` with an AWS SDK or CLI

The following code examples show how to use `DescribeScalingPolicies`.

CLI

AWS CLI

To describe scaling policies

This example command describes the scaling policies for the ecs service namespace.

Command:

```
aws application-autoscaling describe-scaling-policies --service-namespace ecs
```

Output:

```
{
    "ScalingPolicies": [
        {
            "PolicyName": "web-app-cpu-gt-75",
            "ScalableDimension": "ecs:service:DesiredCount",
            "ResourceId": "service/default/web-app",
            "CreationTime": 1462561899.23,
            "StepScalingPolicyConfiguration": {
                "Cooldown": 60,
                "StepAdjustments": [
                    {
                        "ScalingAdjustment": 200,
                        "MetricIntervalLowerBound": 0.0
                    }
                ],
                "AdjustmentType": "PercentChangeInCapacity"
            },
            "PolicyARN": "arn:aws:autoscaling:us-
west-2:012345678910:scalingPolicy:6d8972f3-efc8-437c-92d1-6270f29a66e7:resource/
ecs/service/default/web-app:policyName/web-app-cpu-gt-75",
```

```
            "PolicyType": "StepScaling",
            "Alarms": [
                {
                    "AlarmName": "web-app-cpu-gt-75",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:web-app-cpu-gt-75"
                }
            ],
            "ServiceNamespace": "ecs"
        },
        {
            "PolicyName": "web-app-cpu-lt-25",
            "ScalableDimension": "ecs:service:DesiredCount",
            "ResourceId": "service/default/web-app",
            "CreationTime": 1462562575.099,
            "StepScalingPolicyConfiguration": {
                "Cooldown": 1,
                "StepAdjustments": [
                    {
                        "ScalingAdjustment": -50,
                        "MetricIntervalUpperBound": 0.0
                    }
                ],
                "AdjustmentType": "PercentChangeInCapacity"
            },
            "PolicyARN": "arn:aws:autoscaling:us-
west-2:012345678910:scalingPolicy:6d8972f3-efc8-437c-92d1-6270f29a66e7:resource/
ecs/service/default/web-app:policyName/web-app-cpu-lt-25",
            "PolicyType": "StepScaling",
            "Alarms": [
                {
                    "AlarmName": "web-app-cpu-lt-25",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:web-app-cpu-lt-25"
                }
            ],
            "ServiceNamespace": "ecs"
        }
    ]
}
```

- For API details, see DescribeScalingPolicies in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet describe the Application Auto Scaling scaling policies for the specified service namespace.

```
Get-AASScalingPolicy -ServiceNamespace AppStream
```

Output:

```
Alarms                                    : {Appstream2-LabFleet-default-scale-
out-Alarm}
CreationTime                              : 9/3/2019 2:48:15 AM
PolicyARN                                 : arn:aws:autoscaling:us-
west-2:012345678912:scalingPolicy:5659b069-b5cd-4af1-9f7f-3e956d36233e:resource/
appstream/fleet/LabFleet:
                                            policyName/default-scale-out
PolicyName                                : default-scale-out
PolicyType                                : StepScaling
ResourceId                                : fleet/LabFleet
ScalableDimension                         : appstream:fleet:DesiredCapacity
ServiceNamespace                          : appstream
StepScalingPolicyConfiguration            :
 Amazon.ApplicationAutoScaling.Model.StepScalingPolicyConfiguration
TargetTrackingScalingPolicyConfiguration :

Alarms                                    : {Appstream2-LabFleet-default-scale-in-
Alarm}
CreationTime                              : 9/3/2019 2:48:15 AM
PolicyARN                                 : arn:aws:autoscaling:us-
west-2:012345678912:scalingPolicy:5659b069-b5cd-4af1-9f7f-3e956d36233e:resource/
appstream/fleet/LabFleet:
                                            policyName/default-scale-in
PolicyName                                : default-scale-in
PolicyType                                : StepScaling
ResourceId                                : fleet/LabFleet
ScalableDimension                         : appstream:fleet:DesiredCapacity
ServiceNamespace                          : appstream
StepScalingPolicyConfiguration            :
 Amazon.ApplicationAutoScaling.Model.StepScalingPolicyConfiguration
TargetTrackingScalingPolicyConfiguration :
```

- For API details, see DescribeScalingPolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet describe the Application Auto Scaling scaling policies for the specified service namespace.

```
Get-AASScalingPolicy -ServiceNamespace AppStream
```

Output:

```
Alarms                                    : {Appstream2-LabFleet-default-scale-
out-Alarm}
CreationTime                              : 9/3/2019 2:48:15 AM
PolicyARN                                 : arn:aws:autoscaling:us-
west-2:012345678912:scalingPolicy:5659b069-b5cd-4af1-9f7f-3e956d36233e:resource/
appstream/fleet/LabFleet:
                                            policyName/default-scale-out
PolicyName                                : default-scale-out
PolicyType                                : StepScaling
ResourceId                                : fleet/LabFleet
ScalableDimension                         : appstream:fleet:DesiredCapacity
ServiceNamespace                          : appstream
StepScalingPolicyConfiguration            :
 Amazon.ApplicationAutoScaling.Model.StepScalingPolicyConfiguration
TargetTrackingScalingPolicyConfiguration :

Alarms                                    : {Appstream2-LabFleet-default-scale-in-
Alarm}
CreationTime                              : 9/3/2019 2:48:15 AM
PolicyARN                                 : arn:aws:autoscaling:us-
west-2:012345678912:scalingPolicy:5659b069-b5cd-4af1-9f7f-3e956d36233e:resource/
appstream/fleet/LabFleet:
                                            policyName/default-scale-in
PolicyName                                : default-scale-in
PolicyType                                : StepScaling
ResourceId                                : fleet/LabFleet
ScalableDimension                         : appstream:fleet:DesiredCapacity
ServiceNamespace                          : appstream
StepScalingPolicyConfiguration            :
 Amazon.ApplicationAutoScaling.Model.StepScalingPolicyConfiguration
TargetTrackingScalingPolicyConfiguration :
```

- For API details, see DescribeScalingPolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn show_policies(client: &Client) -> Result<(), Error> {
    let response = client
        .describe_scaling_policies()
        .service_namespace(ServiceNamespace::Ec2)
        .send()
        .await?;
    println!("Auto Scaling Policies:");
    for policy in response.scaling_policies() {
        println!("{:?}\n", policy);
    }
    println!("Next token: {:?}", response.next_token());

    Ok(())
}
```

- For API details, see DescribeScalingPolicies in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScheduledActions` with a CLI

The following code examples show how to use `DescribeScheduledActions`.

CLI

AWS CLI

To describe scheduled actions

The following `describe-scheduled-actions` example displays details for the scheduled actions for the specified service namespace:

```
aws application-autoscaling describe-scheduled-actions \
    --service-namespace dynamodb
```

Output:

```
{
    "ScheduledActions": [
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Schedule": "at(2019-05-20T18:35:00)",
            "ResourceId": "table/my-table",
            "CreationTime": 1561571888.361,
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledAction:2d36aa3b-cdf9-4565-
b290-81db519b227d:resource/dynamodb/table/my-table:scheduledActionName/my-first-
scheduled-action",
            "ScalableTargetAction": {
                "MinCapacity": 15,
                "MaxCapacity": 20
            },
            "ScheduledActionName": "my-first-scheduled-action",
            "ServiceNamespace": "dynamodb"
        },
        {
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "Schedule": "at(2019-05-20T18:40:00)",
            "ResourceId": "table/my-table",
            "CreationTime": 1561571946.021,
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledAction:2d36aa3b-cdf9-4565-
b290-81db519b227d:resource/dynamodb/table/my-table:scheduledActionName/my-second-
scheduled-action",
            "ScalableTargetAction": {
                "MinCapacity": 5,
```

```
                "MaxCapacity": 10
            },
            "ScheduledActionName": "my-second-scheduled-action",
            "ServiceNamespace": "dynamodb"
        }
    ]
}
```

For more information, see Scheduled Scaling in the *Application Auto Scaling User Guide*.

- For API details, see DescribeScheduledActions in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet lists the actions scheduled for your Auto Scaling group that haven't run or that have not reached their end time.

```
Get-AASScheduledAction -ServiceNamespace AppStream
```

Output:

```
CreationTime         : 12/22/2019 9:25:52 AM
EndTime              : 1/1/0001 12:00:00 AM
ResourceId           : fleet/MyFleet
ScalableDimension    : appstream:fleet:DesiredCapacity
ScalableTargetAction : Amazon.ApplicationAutoScaling.Model.ScalableTargetAction
Schedule             : cron(0 0 8 ? * MON-FRI *)
ScheduledActionARN   : arn:aws:autoscaling:us-
west-2:012345678912:scheduledAction:4897ca24-3caa-4bf1-8484-851a089b243c:resource/
appstream/fleet/MyFleet:scheduledActionName
                       /WeekDaysFleetScaling
ScheduledActionName  : WeekDaysFleetScaling
ServiceNamespace     : appstream
StartTime            : 1/1/0001 12:00:00 AM
```

- For API details, see DescribeScheduledActions in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet lists the actions scheduled for your Auto Scaling group that haven't run or that have not reached their end time.

```
Get-AASScheduledAction -ServiceNamespace AppStream
```

Output:

```
CreationTime         : 12/22/2019 9:25:52 AM
EndTime              : 1/1/0001 12:00:00 AM
ResourceId           : fleet/MyFleet
ScalableDimension    : appstream:fleet:DesiredCapacity
ScalableTargetAction : Amazon.ApplicationAutoScaling.Model.ScalableTargetAction
Schedule             : cron(0 0 8 ? * MON-FRI *)
ScheduledActionARN   : arn:aws:autoscaling:us-
west-2:012345678912:scheduledAction:4897ca24-3caa-4bf1-8484-851a089b243c:resource/
appstream/fleet/MyFleet:scheduledActionName
                       /WeekDaysFleetScaling
ScheduledActionName  : WeekDaysFleetScaling
ServiceNamespace     : appstream
StartTime            : 1/1/0001 12:00:00 AM
```

- For API details, see DescribeScheduledActions in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutScalingPolicy` with a CLI

The following code examples show how to use `PutScalingPolicy`.

CLI

AWS CLI

Example 1: To apply a target tracking scaling policy with a predefined metric specification

The following `put-scaling-policy` example applies a target tracking scaling policy with a predefined metric specification to an Amazon ECS service called web-app in the default cluster. The policy keeps the average CPU utilization of the service at 75 percent, with scale-out and scale-in cooldown periods of 60 seconds. The output contains the ARNs and names of the two CloudWatch alarms created on your behalf.

```
aws application-autoscaling put-scaling-policy --service-namespace ecs \
--scalable-dimension ecs:service:DesiredCount \
--resource-id service/default/web-app \
--policy-name cpu75-target-tracking-scaling-policy --policy-
type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration file://config.json
```

This example assumes that you have a config.json file in the current directory with the following contents:

```
{
     "TargetValue": 75.0,
     "PredefinedMetricSpecification": {
         "PredefinedMetricType": "ECSServiceAverageCPUUtilization"
     },
     "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60
}
```

Output:

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
west-2:012345678910:scalingPolicy:6d8972f3-efc8-437c-92d1-6270f29a66e7:resource/
ecs/service/default/web-app:policyName/cpu75-target-tracking-scaling-policy",
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-service/default/web-app-AlarmHigh-
d4f0770c-b46e-434a-a60f-3b36d653feca",
            "AlarmName": "TargetTracking-service/default/web-app-AlarmHigh-
d4f0770c-b46e-434a-a60f-3b36d653feca"
        },
        {
```

```
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-service/default/web-app-
AlarmLow-1b437334-d19b-4a63-a812-6c67aaf2910d",
            "AlarmName": "TargetTracking-service/default/web-app-
AlarmLow-1b437334-d19b-4a63-a812-6c67aaf2910d"
        }
    ]
}
```

Example 2: To apply a target tracking scaling policy with a customized metric specification

The following `put-scaling-policy` example applies a target tracking scaling policy with a customized metric specification to an Amazon ECS service called web-app in the default cluster. The policy keeps the average utilization of the service at 75 percent, with scale-out and scale-in cooldown periods of 60 seconds. The output contains the ARNs and names of the two CloudWatch alarms created on your behalf.

```
aws application-autoscaling put-scaling-policy --service-namespace ecs \
--scalable-dimension ecs:service:DesiredCount \
--resource-id service/default/web-app \
--policy-name cms75-target-tracking-scaling-policy
--policy-type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration file://config.json
```

This example assumes that you have a config.json file in the current directory with the following contents:

```
{
    "TargetValue":75.0,
    "CustomizedMetricSpecification":{
        "MetricName":"MyUtilizationMetric",
        "Namespace":"MyNamespace",
        "Dimensions": [
            {
                "Name":"MyOptionalMetricDimensionName",
                "Value":"MyOptionalMetricDimensionValue"
            }
        ],
        "Statistic":"Average",
        "Unit":"Percent"
```

```
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60
}
```

Output:

```
{
    "PolicyARN": "arn:aws:autoscaling:us-west-2:012345678910:scalingPolicy:
 8784a896-b2ba-47a1-b08c-27301cc499a1:resource/ecs/service/default/web-
app:policyName/cms75-target-tracking-scaling-policy",
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-service/default/web-app-
AlarmHigh-9bc77b56-0571-4276-ba0f-d4178882e0a0",
            "AlarmName": "TargetTracking-service/default/web-app-
AlarmHigh-9bc77b56-0571-4276-ba0f-d4178882e0a0"
        },
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-service/default/web-app-
AlarmLow-9b6ad934-6d37-438e-9e05-02836ddcbdc4",
            "AlarmName": "TargetTracking-service/default/web-app-
AlarmLow-9b6ad934-6d37-438e-9e05-02836ddcbdc4"
        }
    ]
}
```

Example 3: To apply a target tracking scaling policy for scale out only

The following `put-scaling-policy` example applies a target tracking scaling policy to an Amazon ECS service called `web-app` in the default cluster. The policy is used to scale out the ECS service when the `RequestCountPerTarget` metric from the Application Load Balancer exceeds the threshold. The output contains the ARN and name of the CloudWatch alarm created on your behalf.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --scalable-dimension ecs:service:DesiredCount \
    --resource-id service/default/web-app \
    --policy-name alb-scale-out-target-tracking-scaling-policy \
```

```
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration file://config.json
```

Contents of `config.json`:

```
{
     "TargetValue": 1000.0,
     "PredefinedMetricSpecification": {
         "PredefinedMetricType": "ALBRequestCountPerTarget",
         "ResourceLabel": "app/EC2Co-EcsEl-1TKLTMITMM0EO/f37c06a68c1748aa/
targetgroup/EC2Co-Defau-LDNM7Q3ZH1ZN/6d4ea56ca2d6a18d"
     },
     "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "DisableScaleIn": true
}
```

Output:

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
west-2:123456789012:scalingPolicy:6d8972f3-efc8-437c-92d1-6270f29a66e7:resource/
ecs/service/default/web-app:policyName/alb-scale-out-target-tracking-scaling-
policy",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-service/default/web-app-AlarmHigh-
d4f0770c-b46e-434a-a60f-3b36d653feca",
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-service/default/web-app-AlarmHigh-
d4f0770c-b46e-434a-a60f-3b36d653feca"
        }
    ]
}
```

For more information, see Target Tracking Scaling Policies for Application Auto Scaling in the *AWS Application Auto Scaling User Guide*.

- For API details, see PutScalingPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet creates or updates a policy for an Application Auto Scaling scalable target. Each scalable target is identified by a service namespace, resource ID, and scalable dimension.

```
Set-AASScalingPolicy -ServiceNamespace AppStream -PolicyName ASFleetScaleInPolicy
 -PolicyType StepScaling  -ResourceId fleet/MyFleet -ScalableDimension
 appstream:fleet:DesiredCapacity -StepScalingPolicyConfiguration_AdjustmentType
 ChangeInCapacity -StepScalingPolicyConfiguration_Cooldown 360
 -StepScalingPolicyConfiguration_MetricAggregationType Average -
StepScalingPolicyConfiguration_StepAdjustments @{ScalingAdjustment = -1;
 MetricIntervalUpperBound = 0}
```

Output:

```
Alarms    PolicyARN
------    ---------
{}        arn:aws:autoscaling:us-
west-2:012345678912:scalingPolicy:4897ca24-3caa-4bf1-8484-851a089b243c:resource/
appstream/fleet/MyFleet:policyName/ASFleetScaleInPolicy
```

- For API details, see PutScalingPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet creates or updates a policy for an Application Auto Scaling scalable target. Each scalable target is identified by a service namespace, resource ID, and scalable dimension.

```
Set-AASScalingPolicy -ServiceNamespace AppStream -PolicyName ASFleetScaleInPolicy
 -PolicyType StepScaling  -ResourceId fleet/MyFleet -ScalableDimension
 appstream:fleet:DesiredCapacity -StepScalingPolicyConfiguration_AdjustmentType
 ChangeInCapacity -StepScalingPolicyConfiguration_Cooldown 360
 -StepScalingPolicyConfiguration_MetricAggregationType Average -
StepScalingPolicyConfiguration_StepAdjustments @{ScalingAdjustment = -1;
 MetricIntervalUpperBound = 0}
```

Output:

```
Alarms    PolicyARN
------    ---------
{}        arn:aws:autoscaling:us-
west-2:012345678912:scalingPolicy:4897ca24-3caa-4bf1-8484-851a089b243c:resource/
appstream/fleet/MyFleet:policyName/ASFleetScaleInPolicy
```

- For API details, see PutScalingPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutScheduledAction` with a CLI

The following code examples show how to use `PutScheduledAction`.

CLI

AWS CLI

To add a scheduled action to a DynamoDB table

This example adds a scheduled action to a DynamoDB table called TestTable to scale out on a recurring schedule. On the specified schedule (every day at 12:15pm UTC), if the current capacity is below the value specified for MinCapacity, Application Auto Scaling scales out to the value specified by MinCapacity.

Command:

```
aws application-autoscaling put-scheduled-action --service-
namespace dynamodb --scheduled-action-name my-recurring-action --
schedule "cron(15 12 * * ? *)" --resource-id table/TestTable --
scalable-dimension dynamodb:table:WriteCapacityUnits --scalable-target-
action MinCapacity=6
```

For more information, see Scheduled Scaling in the *Application Auto Scaling User Guide*.

- For API details, see PutScheduledAction in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet creates or updates a scheduled action for an Application Auto Scaling scalable target. Each scalable target is identified by a service namespace, resource ID, and scalable dimension.

```
Set-AASScheduledAction -ServiceNamespace AppStream -ResourceId fleet/
MyFleet -Schedule "cron(0 0 8 ? * MON-FRI *)" -ScalableDimension
 appstream:fleet:DesiredCapacity -ScheduledActionName WeekDaysFleetScaling -
ScalableTargetAction_MinCapacity 5 -ScalableTargetAction_MaxCapacity 10
```

- For API details, see PutScheduledAction in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet creates or updates a scheduled action for an Application Auto Scaling scalable target. Each scalable target is identified by a service namespace, resource ID, and scalable dimension.

```
Set-AASScheduledAction -ServiceNamespace AppStream -ResourceId fleet/
MyFleet -Schedule "cron(0 0 8 ? * MON-FRI *)" -ScalableDimension
 appstream:fleet:DesiredCapacity -ScheduledActionName WeekDaysFleetScaling -
ScalableTargetAction_MinCapacity 5 -ScalableTargetAction_MaxCapacity 10
```

- For API details, see PutScheduledAction in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RegisterScalableTarget` with an AWS SDK or CLI

The following code examples show how to use `RegisterScalableTarget`.

CLI

AWS CLI

Example 1: To register an ECS service as a scalable target

The following `register-scalable-target` example registers an Amazon ECS service with Application Auto Scaling. It also adds a tag with the key name `environment` and the value `production` to the scalable target.

```
aws application-autoscaling register-scalable-target \
    --service-namespace ecs \
    --scalable-dimension ecs:service:DesiredCount \
    --resource-id service/default/web-app \
    --min-capacity 1 --max-capacity 10 \
    --tags environment=production
```

Output:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:us-
west-2:123456789012:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

For examples for other AWS services and custom resources, see the topics in AWS services that you can use with Application Auto Scaling in the *Application Auto Scaling User Guide*.

Example 2: To suspend scaling activities for a scalable target

The following `register-scalable-target` example suspends scaling activities for an existing scalable target.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:ReadCapacityUnits \
    --resource-id table/my-table \
    --suspended-
state DynamicScalingInSuspended=true,DynamicScalingOutSuspended=true,ScheduledScalingSusp
```

Output:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:us-
west-2:123456789012:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

For more information, see Suspending and resuming scaling for Application Auto Scaling in the *Application Auto Scaling User Guide*.

Example 3: To resume scaling activities for a scalable target

The following `register-scalable-target` example resumes scaling activities for an existing scalable target.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:ReadCapacityUnits \
    --resource-id table/my-table \
    --suspended-
state DynamicScalingInSuspended=false,DynamicScalingOutSuspended=false,ScheduledScalingSu
```

Output:

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:us-
west-2:123456789012:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

For more information, see Suspending and resuming scaling for Application Auto Scaling in the *Application Auto Scaling User Guide*.

- For API details, see RegisterScalableTarget in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExcep
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequ
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsResp
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequ
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesResp
import software.amazon.awssdk.services.applicationautoscaling.model.PolicyType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PredefinedMetricSpecificati
import
 software.amazon.awssdk.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.RegisterScalableTargetReque
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalingPolicy;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import software.amazon.awssdk.services.applicationautoscaling.model.MetricType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.TargetTrackingScalingPolicy
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <roleARN> <policyName>\s
```

```
            Where:
               tableId - The table Id value (for example, table/Music).
               roleARN - The ARN of the role that has ApplicationAutoScaling
permissions.
               policyName - The name of the policy to create.

            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        System.out.println("This example registers an Amazon DynamoDB table,
which is the resource to scale.");
        String tableId = args[0];
        String roleARN = args[1];
        String policyName = args[2];
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        ApplicationAutoScalingClient appAutoScalingClient =
ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        registerScalableTarget(appAutoScalingClient, tableId, roleARN, ns,
tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        configureScalingPolicy(appAutoScalingClient, tableId, ns, tableWCUs,
policyName);
    }

    public static void registerScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, String roleARN, ServiceNamespace ns,
ScalableDimension tableWCUs) {
        try {
            RegisterScalableTargetRequest targetRequest =
RegisterScalableTargetRequest.builder()
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .roleARN(roleARN)
                .minCapacity(5)
```

```
                .maxCapacity(10)
                .build();

            appAutoScalingClient.registerScalableTarget(targetRequest);
            System.out.println("You have registered " + tableId);

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the target was created.
    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    // Configure a scaling policy.
    public static void configureScalingPolicy(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs, String policyName) {
        // Check if the policy exists before creating a new one.
        DescribeScalingPoliciesResponse describeScalingPoliciesResponse =
appAutoScalingClient.describeScalingPolicies(DescribeScalingPoliciesRequest.builder()
            .serviceNamespace(ns)
            .resourceId(tableId)
            .scalableDimension(tableWCUs)
            .build());

        if (!describeScalingPoliciesResponse.scalingPolicies().isEmpty()) {
            // If policies exist, consider updating an existing policy instead of
creating a new one.
```

```
            System.out.println("Policy already exists. Consider updating it
 instead.");
            List<ScalingPolicy> polList =
 describeScalingPoliciesResponse.scalingPolicies();
            for (ScalingPolicy pol : polList) {
                System.out.println("Policy name:" +pol.policyName());
            }
        } else {
            // If no policies exist, proceed with creating a new policy.
            PredefinedMetricSpecification specification =
 PredefinedMetricSpecification.builder()

 .predefinedMetricType(MetricType.DYNAMO_DB_WRITE_CAPACITY_UTILIZATION)
                .build();

            TargetTrackingScalingPolicyConfiguration policyConfiguration =
 TargetTrackingScalingPolicyConfiguration.builder()
                .predefinedMetricSpecification(specification)
                .targetValue(50.0)
                .scaleInCooldown(60)
                .scaleOutCooldown(60)
                .build();

            PutScalingPolicyRequest putScalingPolicyRequest =
 PutScalingPolicyRequest.builder()
                .targetTrackingScalingPolicyConfiguration(policyConfiguration)
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .policyName(policyName)
                .policyType(PolicyType.TARGET_TRACKING_SCALING)
                .build();

            try {
                appAutoScalingClient.putScalingPolicy(putScalingPolicyRequest);
                System.out.println("You have successfully created a scaling
 policy for an Application Auto Scaling scalable target");
            } catch (ApplicationAutoScalingException e) {
                System.err.println("Error: " +
 e.awsErrorDetails().errorMessage());
            }
        }
    }
}
```

- For API details, see RegisterScalableTarget in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This cmdlet registers or updates a scalable target. A scalable target is a resource that Application Auto Scaling can scale out and scale in.

```
Add-AASScalableTarget -ServiceNamespace AppStream -ResourceId fleet/MyFleet -
ScalableDimension appstream:fleet:DesiredCapacity -MinCapacity 2 -MaxCapacity 10
```

- For API details, see RegisterScalableTarget in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This cmdlet registers or updates a scalable target. A scalable target is a resource that Application Auto Scaling can scale out and scale in.

```
Add-AASScalableTarget -ServiceNamespace AppStream -ResourceId fleet/MyFleet -
ScalableDimension appstream:fleet:DesiredCapacity -MinCapacity 2 -MaxCapacity 10
```

- For API details, see RegisterScalableTarget in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Tagging support for Application Auto Scaling

You can use the AWS CLI or an SDK to tag Application Auto Scaling scalable targets. Scalable targets are the entities that represent the AWS or custom resources that Application Auto Scaling can scale.

Each tag is a label consisting of a user-defined key and value using the Application Auto Scaling API. Tags can help you configure granular access to specific scalable targets according to your organization's needs. For more information, see ABAC with Application Auto Scaling.

You can add tags to new scalable targets when you register them, or you can add them to existing scalable targets.

The commonly used commands for managing tags include:

- register-scalable-target to tag new scalable targets when you register them.
- tag-resource to add tags to an existing scalable target.
- list-tags-for-resource to return the tags on a scalable target.
- untag-resource to delete a tag.

Tagging example

Use the following register-scalable-target command with the `--tags` option. This example tags a scalable target with two tags: a tag key named **`environment`** with the tag value of **`production`**, and a tag key named **`iscontainerbased`** with the tag value of **`true`**.

Replace the sample values for `--min-capacity` and `--max-capacity` and sample text for `--service-namespace` with the namespace of the AWS service you're using with Application Auto Scaling, `--scalable-dimension` with the scalable dimension associated with the resource you're registering, and `--resource-id` with an identifier for the resource. For more information and examples for each service, see the topics in AWS services that you can use with Application Auto Scaling.

```
aws application-autoscaling register-scalable-target \
  --service-namespace namespace \
  --scalable-dimension dimension \
  --resource-id identifier \
```

```
  --min-capacity 1 --max-capacity 10 \
  --tags environment=production,iscontainerbased=true
```

If successful, this command returns the ARN of the scalable target.

```
{
    "ScalableTargetARN": "arn:aws:application-autoscaling:region:account-id:scalable-target/1234abcd56ab78cd901ef1234567890ab123"
}
```

Note

If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Tags for security

Use tags to verify that the requester (such as an IAM user or role) has permissions to perform certain actions. Provide tag information in the condition element of an IAM policy by using one or more of the following condition keys:

- Use `aws:ResourceTag/tag-key: tag-value` to allow (or deny) user actions on scalable targets with specific tags.
- Use `aws:RequestTag/tag-key: tag-value` to require that a specific tag be present (or not present) in a request.
- Use `aws:TagKeys [tag-key, ...]` to require that specific tag keys be present (or not present) in a request.

For example, the following IAM policy grants permissions to use the `DeregisterScalableTarget`, `DeleteScalingPolicy`, and `DeleteScheduledAction` actions. However, it also denies the actions if the scalable target being acted upon has the tag **`environment=production`**.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeleteScheduledAction"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeleteScheduledAction"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "production"
                }
            }
        }
    ]
}
```

Control access to tags

Use tags to verify that the requester (such as an IAM user or role) has permissions to add, modify, or delete tags for scalable targets.

For example, you could create an IAM policy that allows removing only the tag with the **`temporary`** key from scalable targets.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "application-autoscaling:UntagResource",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": { "aws:TagKeys": ["temporary"] }
            }
        }
    ]
}
```

Security in Application Auto Scaling

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. 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 Application Auto Scaling, 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 company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Application Auto Scaling. The following topics show you how to configure Application Auto Scaling to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Application Auto Scaling resources.

Contents

- Data protection in Application Auto Scaling
- Identity and Access Management for Application Auto Scaling
- Access Application Auto Scaling using interface VPC endpoints
- Resilience in Application Auto Scaling
- Infrastructure security in Application Auto Scaling
- Compliance validation for Application Auto Scaling

Data protection in Application Auto Scaling

The AWS shared responsibility model applies to data protection in Application Auto Scaling. 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 Application Auto Scaling 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.

Identity and Access Management for Application Auto Scaling

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 Application Auto Scaling resources. IAM is an AWS service that you can use with no additional charge.

For complete IAM documentation, see the IAM User Guide.

Access control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Application Auto Scaling resources. For example, you must have permissions to create scaling policies, configure scheduled scaling, and so on.

The following sections provide details on how an IAM administrator can use IAM to help secure your AWS resources, by controlling who can perform Application Auto Scaling API actions.

Contents

- How Application Auto Scaling works with IAM
- AWS managed policies for Application Auto Scaling
- Service-linked roles for Application Auto Scaling
- Application Auto Scaling identity-based policy examples
- Troubleshooting access to Application Auto Scaling
- Permissions validation for Application Auto Scaling API calls on target resources

How Application Auto Scaling works with IAM

> **Note**
>
> In December 2017, there was an update for Application Auto Scaling, enabling several service-linked roles for Application Auto Scaling integrated services. Specific IAM permissions *and* an Application Auto Scaling service-linked role (or a service role for Amazon EMR auto scaling) are required so that users can configure scaling.

Before you use IAM to manage access to Application Auto Scaling, learn what IAM features are available to use with Application Auto Scaling.

IAM features you can use with Application Auto Scaling

IAM feature	Application Auto Scaling support
Identity-based policies	Yes
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
Resource-based policies	No
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how Application Auto Scaling and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Application Auto Scaling identity-based policies

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. 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 Application Auto Scaling

To view examples of Application Auto Scaling identity-based policies, see Application Auto Scaling identity-based policy examples.

Actions

Supports policy actions: Yes

In an IAM policy statement, you can specify any API action from any service that supports IAM. For Application Auto Scaling, use the following prefix with the name of the API action: `application-autoscaling:`. For example: `application-autoscaling:RegisterScalableTarget`, `application-autoscaling:PutScalingPolicy`, and `application-autoscaling:DeregisterScalableTarget`.

To specify multiple actions in a single statement, separate them with commas as shown in the following example.

```
"Action": [
      "application-autoscaling:DescribeScalingPolicies",
      "application-autoscaling:DescribeScalingActivities"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "application-autoscaling:Describe*"
```

For a list of Application Auto Scaling actions, see Actions defined by AWS Application Auto Scaling in the *Service Authorization Reference*.

Resources

Supports policy resources: Yes

In an IAM policy statement, the `Resource` element specifies the object or objects that the statement covers. For Application Auto Scaling, each IAM policy statement applies to the scalable targets that you specify using their Amazon Resource Names (ARNs).

The ARN resource format for scalable targets:

```
arn:aws:application-autoscaling:region:account-id:scalable-target/unique-identifier
```

For example, you can indicate a specific scalable target in your statement using its ARN as follows. The unique ID (1234abcd56ab78cd901ef1234567890ab123) is a value assigned by Application Auto Scaling to the scalable target.

```
"Resource": "arn:aws:application-autoscaling:us-east-1:123456789012:scalable-
target/1234abcd56ab78cd901ef1234567890ab123"
```

You can specify all instances that belong to a specific account by replacing the unique identifier with a wildcard (*) as follows.

```
"Resource": "arn:aws:application-autoscaling:us-east-1:123456789012:scalable-target/*"
```

To specify all resources, or if a specific API action does not support ARNs, use a wildcard (*) as the `Resource` element as follows.

```
"Resource": "*"
```

For more information, see Resource types defined by AWS Application Auto Scaling in the *Service Authorization Reference*.

Condition keys

Supports service-specific policy condition keys: Yes

You can specify conditions in the IAM policies that control access to Application Auto Scaling resources. The policy statement is effective only when the conditions are true.

Application Auto Scaling supports the following service-defined condition keys that you can use in identity-based policies to determine who can perform Application Auto Scaling API actions.

- `application-autoscaling:scalable-dimension`
- `application-autoscaling:service-namespace`

To learn which Application Auto Scaling API actions you can use a condition key with, see Actions defined by AWS Application Auto Scaling in the *Service Authorization Reference*. For more

information about using Application Auto Scaling condition keys, see Condition keys for AWS Application Auto Scaling.

To view the global condition keys that are available to all services, see AWS global condition context keys in the *IAM User Guide*.

Resource-based policies

Supports resource-based policies: No

Other AWS services, such as Amazon Simple Storage Service, support resource-based permissions policies. For example, you can attach a permissions policy to an S3 bucket to manage access permissions to that bucket.

Application Auto Scaling does not support resource-based policies.

Access Control Lists (ACLs)

Supports ACLs: No

Application Auto Scaling does not support Access Control Lists (ACLs).

ABAC with Application Auto Scaling

Supports ABAC (tags in policies): Partial

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.

ABAC is possible for resources that support tags, but not everything supports tags. Scheduled actions and scaling policies don't support tags, but scalable targets support tags. For more information, see Tagging support for Application Auto Scaling.

For more information about ABAC, see What is ABAC? 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 Application Auto Scaling

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Service roles

Supports service roles: Yes

If your Amazon EMR cluster uses automatic scaling, this feature allows Application Auto Scaling to assume a service role on your behalf. Similar to a service-linked role, a service role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Application Auto Scaling supports service roles only for Amazon EMR. For documentation for the EMR service role, see Using automatic scaling with a custom policy for instance groups in the *Amazon EMR Management Guide*.

> **Note**
>
> With the introduction of service-linked roles, several legacy service roles are no longer required, for example, for Amazon ECS and Spot Fleet.

Service-linked roles

Supports service-linked roles: Yes

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 information about Application Auto Scaling service-linked roles, see Service-linked roles for Application Auto Scaling.

AWS managed policies for Application Auto Scaling

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: WorkSpaces Applications and CloudWatch

Policy name: AWSApplicationAutoscalingAppStreamFleetPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_AppStreamFleet to allow Application Auto Scaling to call Amazon AppStream and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `appstream:DescribeFleets`
- Action: `appstream:UpdateFleet`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: Aurora and CloudWatch

Policy name: AWSApplicationAutoscalingRDSClusterPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_RDSCluster to allow Application Auto Scaling to call Aurora and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `rds:AddTagsToResource`
- Action: `rds:CreateDBInstance`
- Action: `rds:DeleteDBInstance`
- Action: `rds:DescribeDBClusters`
- Action: `rds:DescribeDBInstance`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: Amazon Comprehend and CloudWatch

Policy name: AWSApplicationAutoscalingComprehendEndpointPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_ComprehendEndpoint to allow Application Auto Scaling to call Amazon Comprehend and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `comprehend:UpdateEndpoint`
- Action: `comprehend:DescribeEndpoint`

- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: DynamoDB and CloudWatch

Policy name: AWSApplicationAutoscalingDynamoDBTablePolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_DynamoDBTable to allow Application Auto Scaling to call DynamoDBand CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `dynamodb:DescribeTable`
- Action: `dynamodb:UpdateTable`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: Amazon ECS and CloudWatch

Policy name: AWSApplicationAutoscalingECSServicePolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_ECSService to allow Application Auto Scaling to call Amazon ECS and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `ecs:DescribeServices`
- Action: `ecs:UpdateService`

- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:GetMetricData`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: ElastiCache and CloudWatch

Policy name: AWSApplicationAutoscalingElastiCacheRGPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG to allow Application Auto Scaling to call ElastiCache and CloudWatch and perform scaling on your behalf. This service-linked role can be used for ElastiCache Memcached, Redis OSS, and Valkey.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on the specified resources:

- Action: `elasticache:DescribeReplicationGroups` on all resources
- Action: `elasticache:ModifyReplicationGroupShardConfiguration` on all resources
- Action: `elasticache:IncreaseReplicaCount` on all resources
- Action: `elasticache:DecreaseReplicaCount` on all resources
- Action: `elasticache:DescribeCacheClusters` on all resources
- Action: `elasticache:DescribeCacheParameters` on all resources
- Action: `elasticache:ModifyCacheCluster` on all resources
- Action: `cloudwatch:DescribeAlarms` on the resource `arn:aws:cloudwatch:*:*:alarm:*`
- Action: `cloudwatch:PutMetricAlarm` on the resource `arn:aws:cloudwatch:*:*:alarm:TargetTracking*`
- Action: `cloudwatch:DeleteAlarms` on the resource `arn:aws:cloudwatch:*:*:alarm:TargetTracking*`

AWS managed policy: Amazon Keyspaces and CloudWatch

Policy name: AWSApplicationAutoscalingCassandraTablePolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_CassandraTable to allow Application Auto Scaling to call Amazon Keyspaces and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on the specified resources:

- Action: `cassandra:Select` on the following resources:
  - `arn:*:cassandra:*:*:/keyspace/system/table/*`
  - `arn:*:cassandra:*:*:/keyspace/system_schema/table/*`
  - `arn:*:cassandra:*:*:/keyspace/system_schema_mcs/table/*`
- Action: `cassandra:Alter` on all resources
- Action: `cloudwatch:DescribeAlarms` on all resources
- Action: `cloudwatch:PutMetricAlarm` on all resources
- Action: `cloudwatch:DeleteAlarms` on all resources

AWS managed policy: Lambda and CloudWatch

Policy name: AWSApplicationAutoscalingLambdaConcurrencyPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_LambdaConcurrency to allow Application Auto Scaling to call Lambda and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `lambda:PutProvisionedConcurrencyConfig`
- Action: `lambda:GetProvisionedConcurrencyConfig`
- Action: `lambda:DeleteProvisionedConcurrencyConfig`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: Amazon MSK and CloudWatch

Policy name: AWSApplicationAutoscalingKafkaClusterPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_KafkaCluster to allow Application Auto Scaling to call Amazon MSK and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `kafka:DescribeCluster`
- Action: `kafka:DescribeClusterOperation`
- Action: `kafka:UpdateBrokerStorage`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: Neptune and CloudWatch

Policy name: AWSApplicationAutoscalingNeptuneClusterPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_NeptuneCluster to allow Application Auto Scaling to call Neptune and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on the specified resources:

- Action: `rds:ListTagsForResource` on all resources
- Action: `rds:DescribeDBInstances` on all resources
- Action: `rds:DescribeDBClusters` on all resources
- Action: `rds:DescribeDBClusterParameters` on all resources
- Action: `cloudwatch:DescribeAlarms` on all resources

- Action: `rds:AddTagsToResource` on resources with the prefix *autoscaled-reader* in the Amazon Neptune database engine (`"Condition":{"StringEquals":{"rds:DatabaseEngine":"neptune"}}`)
- Action: `rds:CreateDBInstance` on resources with the prefix *autoscaled-reader* in all DB clusters (`"Resource":"arn:*:rds:*:*:db:autoscaled-reader*"`, `"arn:aws:rds:*:*:cluster:*"`) in the Amazon Neptune database engine (`"Condition":{"StringEquals":{"rds:DatabaseEngine":"neptune"}}`)
- Action: `rds:DeleteDBInstance` on the resource `arn:aws:rds:*:*:db:autoscaled-reader*`
- Action: `cloudwatch:PutMetricAlarm` on the resource `arn:aws:cloudwatch:*:*:alarm:TargetTracking*`
- Action: `cloudwatch:DeleteAlarms` on the resource `arn:aws:cloudwatch:*:*:alarm:TargetTracking*`

AWS managed policy: SageMaker AI and CloudWatch

Policy name: AWSApplicationAutoscalingSageMakerEndpointPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint to allow Application Auto Scaling to call SageMaker AI and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on the specified resources:

- Action: `sagemaker:DescribeEndpoint` on all resources
- Action: `sagemaker:DescribeEndpointConfig` on all resources
- Action: `sagemaker:DescribeInferenceComponent` on all resources
- Action: `sagemaker:UpdateEndpointWeightsAndCapacities` on all resources
- Action: `sagemaker:UpdateInferenceComponentRuntimeConfig` on all resources
- Action: `cloudwatch:DescribeAlarms` on all resources
- Action: `cloudwatch:GetMetricData` on all resources
- Action: `cloudwatch:PutMetricAlarm` on the resource `arn:aws:cloudwatch:*:*:alarm:TargetTracking*`

- Action: `cloudwatch:DeleteAlarms` on the resource `arn:aws:cloudwatch:*:*:alarm:TargetTracking*`

AWS managed policy: EC2 Spot Fleet and CloudWatch

Policy name: AWSApplicationAutoscalingEC2SpotFleetRequestPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_EC2SpotFleetRequest to allow Application Auto Scaling to call Amazon EC2 and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `ec2:DescribeSpotFleetRequests`
- Action: `ec2:ModifySpotFleetRequest`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

AWS managed policy: WorkSpaces and CloudWatch

Policy name: AWSApplicationAutoscalingWorkSpacesPoolPolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_WorkSpacesPool to allow Application Auto Scaling to call WorkSpaces and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on the specified resources:

- Action: `workspaces:DescribeWorkspacesPools` on all resources from the same account as the SLR
- Action: `workspaces:UpdateWorkspacesPool` on all resources from the same account as the SLR

- Action: `cloudwatch:DescribeAlarms` on all alarms from the same account as the SLR
- Action: `cloudwatch:PutMetricAlarm` on all alarms from the same account as the SLR, where the alarm name starts with TargetTracking
- Action: `cloudwatch:DeleteAlarms` on all alarms from the same account as the SLR, where the alarm name starts with TargetTracking

AWS managed policy: custom resources and CloudWatch

Policy name: AWSApplicationAutoScalingCustomResourcePolicy

This policy is attached to the service-linked role named AWSServiceRoleForApplicationAutoScaling_CustomResource to allow Application Auto Scaling to call your custom resources that are available through API Gateway and CloudWatch and perform scaling on your behalf.

Permission details

The permissions policy allows Application Auto Scaling to complete the following actions on all related resources ("Resource": "*"):

- Action: `execute-api:Invoke`
- Action: `cloudwatch:DescribeAlarms`
- Action: `cloudwatch:PutMetricAlarm`
- Action: `cloudwatch:DeleteAlarms`

Application Auto Scaling updates to AWS managed policies

View details about updates to AWS managed policies for Application Auto Scaling since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Application Auto Scaling Document history page.

Change	Description	Date
AWSApplicationAutoscalingElastiCacheRGPolicy – Update an existing policy	Added permission to call the ElastiCache `ModifyCacheCluster` API action	April 10, 2025

Change	Description	Date
	to support Memcached automatic scaling.	
AWSApplicationAutoscalingECSServicePolicy – Update an existing policy	Added permission to call the CloudWatch `GetMetricData` API action to support predictive scaling.	November 21, 2024
AWSApplicationAutoscalingWorkSpacesPoolPolicy – New policy	Added a managed policy for Amazon WorkSpaces. This policy is attached to a service-linked role that allows Application Auto Scaling to call WorkSpaces and CloudWatch and perform scaling on your behalf.	June 24, 2024
AWSApplicationAutoscalingSageMakerEndpointPolicy – Update to an existing policy	Added permissions to call the SageMaker AI `DescribeInferenceComponent` and `UpdateInferenceComponentRuntimeConfig` API actions to support compatibility for the auto scaling of SageMaker AI resources for an upcoming integration. The policy also now restricts the CloudWatch `PutMetricAlarm` and `DeleteAlarms` API actions to CloudWatch alarms that are used with target tracking scaling policies.	November 13, 2023

Change	Description	Date
AWSApplicationAutoscalingNeptuneClusterPolicy – New policy	Added a managed policy for Neptune. This policy is attached to a service-linked role that allows Application Auto Scaling to call Neptune and CloudWatch and perform scaling on your behalf.	October 6, 2021
AWSApplicationAutoscalingRDSClusterPolicy – New policy	Added a managed policy for ElastiCache. This policy is attached to a service-linked role that allows Application Auto Scaling to call ElastiCache and CloudWatch and perform scaling on your behalf.	August 19, 2021
Application Auto Scaling started tracking changes	Application Auto Scaling started tracking changes for its AWS managed policies.	August 19, 2021

Service-linked roles for Application Auto Scaling

Application Auto Scaling uses service-linked roles for the permissions that it requires to call other AWS services on your behalf. A service-linked role is a unique type of AWS Identity and Access Management (IAM) role that is linked directly to an AWS service. Service-linked roles provide a secure way to delegate permissions to AWS services because only the linked service can assume a service-linked role.

For services that integrate with Application Auto Scaling, Application Auto Scaling creates service-linked roles for you. There is one service-linked role for each service. Each service-linked role trusts the specified service principal to assume it. For more information, see Service-linked role ARN reference.

Application Auto Scaling includes all of the necessary permissions for each service-linked role. These managed permissions are created and managed by Application Auto Scaling, and they define the allowed actions for each resource type. For details about the permissions that each role grants, see AWS managed policies for Application Auto Scaling.

Contents

- Permissions required to create a service-linked role
- Create service-linked roles (automatic)
- Create service-linked roles (manual)
- Edit the service-linked roles
- Delete the service-linked roles
- Supported Regions for Application Auto Scaling service-linked roles
- Service-linked role ARN reference

Permissions required to create a service-linked role

Application Auto Scaling requires permissions to create a service-linked role the first time any user in your AWS account calls `RegisterScalableTarget` for a given service. Application Auto Scaling creates a service-linked role for the target service in your account, if the role does not exist already. The service-linked role grants permissions to Application Auto Scaling so that it can call the target service on your behalf.

For automatic role creation to succeed, users must have permission for the `iam:CreateServiceLinkedRole` action.

```
"Action": "iam:CreateServiceLinkedRole"
```

The following is an identity-based policy that grants permission to create a service-linked role for Spot Fleet. You can specify the service-linked role in the policy's `Resource` field as an ARN, and the service principal for your service-linked role as a condition, as shown. For the ARN for each service, see Service-linked role ARN reference.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/ec2.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_EC2SpotFleetRequest",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":"ec2.application-
autoscaling.amazonaws.com"
                }
            }
        }
    ]
}
```

Note

The `iam:AWSServiceName` IAM condition key specifies the service principal to which the role is attached, which is indicated in this example policy as *`ec2.application-autoscaling`*`.amazonaws.com`. Do not try to guess the service principal. To view the service principal for a service, see AWS services that you can use with Application Auto Scaling.

Create service-linked roles (automatic)

You don't need to manually create a service-linked role. Application Auto Scaling creates the appropriate service-linked role for you when you call `RegisterScalableTarget`. For example, if you set up automatic scaling for an Amazon ECS service, Application Auto Scaling creates the `AWSServiceRoleForApplicationAutoScaling_ECSService` role.

Create service-linked roles (manual)

To create the service-linked role, you can use the IAM console, AWS CLI, or IAM API. For more information, see Create a service-linked role in the *IAM User Guide*.

To create a service-linked role (AWS CLI)

Use the following create-service-linked-role command to create the Application Auto Scaling service-linked role. In the request, specify the service name "prefix".

To find the service name prefix, refer to the information about the service principal for the service-linked role for each service in the AWS services that you can use with Application Auto Scaling section. The service name and the service principal share the same prefix. For example, to create the AWS Lambda service-linked role, use `lambda.application-autoscaling.amazonaws.com`.

```
aws iam create-service-linked-role --aws-service-name prefix.application-autoscaling.amazonaws.com
```

Edit the service-linked roles

With the service-linked roles created by Application Auto Scaling, you can edit only their descriptions. For more information, see Edit a service-linked role description in the *IAM User Guide*.

Delete the service-linked roles

If you no longer use Application Auto Scaling with a supported service, we recommend that you delete the corresponding service-linked role.

You can delete a service-linked role only after first deleting the related AWS resources. This protects you from inadvertently revoking Application Auto Scaling permissions to your resources. For more information, see the documentation for the scalable resource. For example, to delete an Amazon ECS service, see Deleting an Amazon ECS service in the *Amazon Elastic Container Service Developer Guide*.

You can use IAM to delete a service-linked role. For more information, see Delete a service-linked role in the *IAM User Guide*.

After you delete a service-linked role, Application Auto Scaling creates the role again when you call `RegisterScalableTarget`.

Supported Regions for Application Auto Scaling service-linked roles

Application Auto Scaling supports using service-linked roles in all of the AWS Regions where the service is available.

Service-linked role ARN reference

The following table lists the Amazon Resource Name (ARN) of the service-linked role for each AWS service that works with Application Auto Scaling.

Service	ARN
AppStream 2.0	arn:aws:iam:: *012345678910* :role/aws-service-role/appstream.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_AppStreamFleet
Aurora	arn:aws:iam:: *012345678910* :role/aws-service-role/rds.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_RDSCluster
Comprehend	arn:aws:iam:: *012345678910* :role/aws-service-role/comprehend.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_ComprehendEndpoint
DynamoDB	arn:aws:iam:: *012345678910* :role/aws-service-role/dynamodb.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable
ECS	arn:aws:iam:: *012345678910* :role/aws-service-role/ecs.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_ECSService
ElastiCache	arn:aws:iam:: *012345678910* :role/aws-service-role/elasticache.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG
Keyspaces	arn:aws:iam:: *012345678910* :role/aws-service-role/cassandra.application-autoscaling.amazonaws.com/

Service	ARN
	AWSServiceRoleForApplicationAutoScaling_CassandraTable
Lambda	arn:aws:iam:: *012345678910* :role/aws-service-role/lambda.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_LambdaConcurrency
MSK	arn:aws:iam:: *012345678910* :role/aws-service-role/kafka.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_KafkaCluster
Neptune	arn:aws:iam:: *012345678910* :role/aws-service-role/neptune.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_NeptuneCluster
SageMaker AI	arn:aws:iam:: *012345678910* :role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint
Spot Fleets	arn:aws:iam:: *012345678910* :role/aws-service-role/ec2.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_EC2SpotFleetRequest
WorkSpaces	arn:aws:iam:: *012345678910* :role/aws-service-role/workspaces.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_WorkSpacesPool
Custom resources	arn:aws:iam:: *012345678910* :role/aws-service-role/custom-resource.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_CustomResource

Note

You can specify the ARN of a service-linked role for the `RoleARN` property of an AWS::ApplicationAutoScaling::ScalableTarget resource in your CloudFormation stack templates, even if the specified service-linked role doesn't yet exist. Application Auto Scaling automatically creates the role for you.

Application Auto Scaling identity-based policy examples

By default, a brand new user in your AWS account has no permissions to do anything. An IAM administrator must create and assign IAM policies that give an IAM identity (such as a user or role) permission to perform Application Auto Scaling API actions.

To learn how to create an IAM policy using the following example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Contents

- Permissions required for Application Auto Scaling API actions
- Permissions required for API actions on target services and CloudWatch
- Permissions for working in the AWS Management Console

Permissions required for Application Auto Scaling API actions

The following policies grant permissions for common use cases when calling Application Auto Scaling API. Refer to this section when writing identity-based policies. Each policy grants permissions to all or some of the Application Auto Scaling API actions. You also need to make sure that end users have permissions for the target service and CloudWatch (see the next section for details).

The following identity-based policy grants permissions to all Application Auto Scaling API actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
              "application-autoscaling:*"
            ],
            "Resource": "*"
        }
    ]
}
```

The following identity-based policy grants permissions to all Application Auto Scaling API actions that are required to configure scaling policies and not scheduled actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "application-autoscaling:RegisterScalableTarget",
              "application-autoscaling:DescribeScalableTargets",
              "application-autoscaling:DeregisterScalableTarget",
              "application-autoscaling:PutScalingPolicy",
              "application-autoscaling:DescribeScalingPolicies",
              "application-autoscaling:DescribeScalingActivities",
              "application-autoscaling:DeleteScalingPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

The following identity-based policy grants permissions to all Application Auto Scaling API actions that are required to configure scheduled actions and not scaling policies.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "application-autoscaling:RegisterScalableTarget",
              "application-autoscaling:DescribeScalableTargets",
              "application-autoscaling:DeregisterScalableTarget",
              "application-autoscaling:PutScheduledAction",
              "application-autoscaling:DescribeScheduledActions",
              "application-autoscaling:DescribeScalingActivities",
              "application-autoscaling:DeleteScheduledAction"
            ],
            "Resource": "*"
        }
    ]
}
```

Permissions required for API actions on target services and CloudWatch

To successfully configure and use Application Auto Scaling with the target service, end users must be granted permissions for Amazon CloudWatch and for each target service for which they will configure scaling. Use the following policies to grant the minimum permissions required to work with target services and CloudWatch.

Contents

- AppStream 2.0 fleets
- Aurora replicas
- Amazon Comprehend document classification and entity recognizer endpoints
- DynamoDB tables and global secondary indexes
- ECS services
- ElastiCache replication groups
- Amazon EMR clusters
- Amazon Keyspaces tables
- Lambda functions
- Amazon Managed Streaming for Apache Kafka (MSK) broker storage
- Neptune clusters

- SageMaker AI endpoints
- Spot Fleets (Amazon EC2)
- Custom resources

AppStream 2.0 fleets

The following identity-based policy grants permissions to all AppStream 2.0 and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "appstream:DescribeFleets",
              "appstream:UpdateFleet",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Aurora replicas

The following identity-based policy grants permissions to all Aurora and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
              "rds:AddTagsToResource",
              "rds:CreateDBInstance",
              "rds:DeleteDBInstance",
              "rds:DescribeDBClusters",
              "rds:DescribeDBInstances",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Comprehend document classification and entity recognizer endpoints

The following identity-based policy grants permissions to all Amazon Comprehend and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "comprehend:UpdateEndpoint",
              "comprehend:DescribeEndpoint",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

DynamoDB tables and global secondary indexes

The following identity-based policy grants permissions to all DynamoDB and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "dynamodb:DescribeTable",
              "dynamodb:UpdateTable",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

ECS services

The following identity-based policy grants permissions to all ECS and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "ecs:DescribeServices",
              "ecs:UpdateService",
              "cloudwatch:DescribeAlarms",
```

```
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

ElastiCache replication groups

The following identity-based policy grants permissions to all ElastiCache and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "elasticache:ModifyReplicationGroupShardConfiguration",
              "elasticache:IncreaseReplicaCount",
              "elasticache:DecreaseReplicaCount",
              "elasticache:DescribeReplicationGroups",
              "elasticache:DescribeCacheClusters",
              "elasticache:DescribeCacheParameters",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EMR clusters

The following identity-based policy grants permissions to all Amazon EMR and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "elasticmapreduce:ModifyInstanceGroups",
              "elasticmapreduce:ListInstanceGroups",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Keyspaces tables

The following identity-based policy grants permissions to all Amazon Keyspaces and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "cassandra:Select",
              "cassandra:Alter",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Lambda functions

The following identity-based policy grants permissions to all Lambda and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "lambda:PutProvisionedConcurrencyConfig",
              "lambda:GetProvisionedConcurrencyConfig",
              "lambda:DeleteProvisionedConcurrencyConfig",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Managed Streaming for Apache Kafka (MSK) broker storage

The following identity-based policy grants permissions to all Amazon MSK and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
              "kafka:DescribeCluster",
              "kafka:DescribeClusterOperation",
              "kafka:UpdateBrokerStorage",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Neptune clusters

The following identity-based policy grants permissions to all Neptune and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "rds:AddTagsToResource",
              "rds:CreateDBInstance",
              "rds:DescribeDBInstances",
              "rds:DescribeDBClusters",
              "rds:DescribeDBClusterParameters",
              "rds:DeleteDBInstance",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

SageMaker AI endpoints

The following identity-based policy grants permissions to all SageMaker AI and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "sagemaker:DescribeEndpoint",
              "sagemaker:DescribeEndpointConfig",
              "sagemaker:DescribeInferenceComponent",
              "sagemaker:UpdateEndpointWeightsAndCapacities",
              "sagemaker:UpdateInferenceComponentRuntimeConfig",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Spot Fleets (Amazon EC2)

The following identity-based policy grants permissions to all Spot Fleet and CloudWatch API actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "ec2:DescribeSpotFleetRequests",
                "ec2:ModifySpotFleetRequest",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Custom resources

The following identity-based policy grants permission for the API Gateway API executing action. This policy also grants permissions to all CloudWatch actions that are required.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "execute-api:Invoke",
              "cloudwatch:DescribeAlarms",
              "cloudwatch:PutMetricAlarm",
              "cloudwatch:DeleteAlarms"
            ],
            "Resource": "*"
        }
    ]
}
```

Permissions for working in the AWS Management Console

There is no standalone Application Auto Scaling console. Most services that integrate with Application Auto Scaling have features that are dedicated to helping you configure scaling with their console.

In most cases, each service provides AWS managed (predefined) IAM policies that define access to their console, which includes permissions to the Application Auto Scaling API actions. For more information, refer to the documentation for the service whose console you want to use.

You can also create your own custom IAM policies to give users fine-grained permissions to view and work with specific Application Auto Scaling API actions in the AWS Management Console. You can use the example policies in the previous sections; however, they are designed for requests that are made with the AWS CLI or an SDK. The console uses additional API actions for its features, so these policies may not work as expected. For example, to configure step scaling, users might require additional permissions to create and manage CloudWatch alarms.

Tip

To help you work out which API actions are required to perform tasks in the console, you can use a service such as AWS CloudTrail. For more information, see the AWS CloudTrail User Guide.

The following identity-based policy grants permissions to configure scaling policies for Spot Fleet. In addition to the IAM permissions for Spot Fleet, the console user that accesses fleet scaling settings from the Amazon EC2 console must have the appropriate permissions for the services that support dynamic scaling.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:*",
                "ec2:DescribeSpotFleetRequests",
                "ec2:ModifySpotFleetRequest",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
```

```
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DisableAlarmActions",
                "cloudwatch:EnableAlarmActions",
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:Get*",
                "sns:List*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/ec2.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_EC2SpotFleetRequest",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":"ec2.application-
autoscaling.amazonaws.com"
                }
            }
        }
    ]
}
```

This policy allows console users to view and modify scaling policies in the Amazon EC2 console, and to create and manage CloudWatch alarms in the CloudWatch console.

You can adjust the API actions to limit user access. For example, replacing `application-autoscaling:*` with `application-autoscaling:Describe*` means that the user has read-only access.

You can also adjust the CloudWatch permissions as required to limit user access to CloudWatch features. For more information, see Permissions needed for the CloudWatch console in the *Amazon CloudWatch User Guide*.

Troubleshooting access to Application Auto Scaling

If you encounter `AccessDeniedException` or similar difficulties when working with Application Auto Scaling, consult the information in this section.

I am not authorized to perform an action in Application Auto Scaling

If you receive an `AccessDeniedException` when calling an AWS API operation, it means that the AWS Identity and Access Management (IAM) credentials that you are using do not have the required permissions to make that call.

The following example error occurs when the `mateojackson` user tries to view details about a scalable target, but does not have `application-autoscaling:DescribeScalableTargets` permission.

```
An error occurred (AccessDeniedException) when calling the DescribeScalableTargets
 operation: User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: application-autoscaling:DescribeScalableTargets
```

If you receive this or similar errors, then you must contact your administrator for assistance.

An administrator for your account will need to make sure that you have permissions to access all of the API actions that Application Auto Scaling uses to access resources in the target service and CloudWatch. There are different permissions required depending on which resources you are working with. Application Auto Scaling also requires permission to create a service-linked role the first time that a user configures scaling for a given resource.

I'm an administrator and my IAM policy returned an error or isn't working as expected

In addition to Application Auto Scaling actions, your IAM policies must grant permissions to call the target service and CloudWatch. If a user or application doesn't have these additional permissions, their access might be unexpectedly denied. To write IAM policies for users and applications in your accounts, consult the information in Application Auto Scaling identity-based policy examples.

For information about how validation is performed, see Permissions validation for Application Auto Scaling API calls on target resources.

Note that some permission issues can also be due to an issue with creating the service-linked roles used by Application Auto Scaling. For information about creating these service-linked roles, see Service-linked roles for Application Auto Scaling.

Permissions validation for Application Auto Scaling API calls on target resources

Making authorized requests to Application Auto Scaling API actions requires that the API caller has permissions to access AWS resources in the target service and in CloudWatch. Application Auto Scaling validates permissions for requests associated with both the target service and CloudWatch before proceeding with the request. To accomplish this, we issue a series of calls to validate the IAM permissions on target resources. When a response is returned, it is read by Application Auto Scaling. If the IAM permissions do not allow a given action, Application Auto Scaling fails the request and returns an error to the user containing information about the missing permission. This ensures that the scaling configuration that the user wants to deploy functions as intended, and that a useful error is returned if the request fails.

As an example of how this works, the following information provides details about how Application Auto Scaling performs permissions validations with Aurora and CloudWatch.

When a user calls the `RegisterScalableTarget` API against an Aurora DB cluster, Application Auto Scaling performs all of the following checks to verify that the user has the required permissions (in bold).

- **rds:CreateDBInstance**: To determine whether the user has this permission, we send a request to the `CreateDBInstance` API operation, attempting to create a DB instance with invalid parameters (empty instance ID) in the Aurora DB cluster that the user specified. For an authorized user, the API returns an `InvalidParameterValue` error code response after it audits the request. However, for an unauthorized user, we get an `AccessDenied` error and fail the Application Auto Scaling request with a `ValidationException` error to the user that lists the missing permissions.
- **rds:DeleteDBInstance**: We send an empty instance ID to the `DeleteDBInstance` API operation. For an authorized user, this request results in an `InvalidParameterValue` error. For an unauthorized user, it results in `AccessDenied` and sends a validation exception to the user (same treatment as described in the first bullet point).
- **rds:AddTagsToResource**: Because the `AddTagsToResource` API operation requires an Amazon Resource Name (ARN), it is necessary to specify a "dummy" resource using an invalid account ID (12345) and dummy instance ID (non-existing-db) to construct the ARN (`arn:aws:rds:us-east-1:12345:db:non-existing-db`). For an authorized user, this request results in an `InvalidParameterValue` error. For an unauthorized user, it results in `AccessDenied` and sends a validation exception to the user.

- **rds:DescribeDBClusters**: We describe the cluster name for the resource being registered for auto scaling. For an authorized user, we get a valid describe result. For an unauthorized user, it results in `AccessDenied` and sends a validation exception to the user.
- **rds:DescribeDBInstances**: We call the `DescribeDBInstances` API with a `db-cluster-id` filter that filters on the cluster name that was provided by the user to register the scalable target. For an authorized user, we are permitted to describe all of the DB instances in the DB cluster. For an unauthorized user, this call results in `AccessDenied` and sends a validation exception to the user.
- **cloudwatch:PutMetricAlarm**: We call the `PutMetricAlarm` API without any parameters. Because alarm name is missing, the request results in `ValidationError` for an authorized user. For an unauthorized user, it results in `AccessDenied` and sends a validation exception to the user.
- **cloudwatch:DescribeAlarms**: We call the `DescribeAlarms` API with the maximum number of records value set to 1. For an authorized user, we expect information on one alarm in the response. For an unauthorized user, this call results in `AccessDenied` and sends a validation exception to the user.
- **cloudwatch:DeleteAlarms**: Similar to `PutMetricAlarm` above, we provide no parameters to `DeleteAlarms` request. Because an alarm name is missing from the request, this call fails with `ValidationError` for an authorized user. For an unauthorized user, it results in `AccessDenied` and sends a validation exception to the user.

Whenever any one of these validation exceptions occur, it is logged. You can take steps to manually identify which calls failed validation by using AWS CloudTrail. For more information, see the AWS CloudTrail User Guide.

> **Note**
>
> If you receive alerts for Application Auto Scaling events using CloudTrail, these alerts will include the Application Auto Scaling calls to validate user permissions by default. To filter out these alerts, use the `invokedBy` field, which will contain `application-autoscaling.amazonaws.com` for these validation checks.

Access Application Auto Scaling using interface VPC endpoints

You can use AWS PrivateLink to create a private connection between your VPC and Application Auto Scaling. You can access Application Auto Scaling as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access Application Auto Scaling.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Application Auto Scaling.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Contents

- Create an interface VPC endpoint
- Create a VPC endpoint policy

Create an interface VPC endpoint

Create an endpoint for Application Auto Scaling using the following service name:

```
com.amazonaws.region.application-autoscaling
```

For more information, see Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

You do not need to change any other settings. Application Auto Scaling calls other AWS services using either service endpoints or private interface VPC endpoints, whichever are in use.

Create a VPC endpoint policy

You can attach a policy to your VPC endpoint to control access to the Application Auto Scaling API. The policy specifies:

- The principal that can perform actions.

- The actions that can be performed.
- The resource on which the actions can be performed.

The following example shows a VPC endpoint policy that denies everyone permission to delete a scaling policy through the endpoint. The example policy also grants everyone permission to perform all other actions.

```
{
   "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "application-autoscaling:DeleteScalingPolicy",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

For more information, see VPC endpoint policies in the *AWS PrivateLink Guide*.

Resilience in Application Auto Scaling

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 zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS global infrastructure.

Infrastructure security in Application Auto Scaling

As a managed service, Application Auto Scaling 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 Application Auto Scaling 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.

Compliance validation for Application Auto Scaling

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Quotas for Application Auto Scaling

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

To view the quotas for Application Auto Scaling, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **Application Auto Scaling**.

To request a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*.

Your AWS account has the following quotas related to Application Auto Scaling.

Name	Default	Adjustable
Scalable targets per resource type	Amazon DynamoDB: 5,000 | Amazon ECS: 3,000 | Amazon Keyspaces: 1,500 | Other resource types: 500	Yes
Scaling policies per scalable target (both step scaling and target tracking policies)	50	No
Scheduled actions per scalable target	200	No
Step adjustments per step scaling policy	20	Yes

Keep service quotas in mind as you scale out your workloads. For example, when you reach the maximum number of capacity units allowed by a service, scaling out will stop. If demand drops and the current capacity decreases, Application Auto Scaling can scale out again. To avoid reaching this capacity limit again, you can request an increase. Each service has its own default quotas for the maximum capacity of the resource. For information about the default quotas for other Amazon Web Services, see Service endpoints and quotas in the *Amazon Web Services General Reference*.

Document history for Application Auto Scaling

The following table describes important additions to the Application Auto Scaling documentation, beginning in January 2018. For notification about updates to this documentation, you can subscribe to the RSS feed.

Change	Description	Date
Add support for ElastiCache Memcached clusters	Use Application Auto Scaling to horizontally scale the number of nodes for a Memcached cluster. For more information, see ElastiCache and Application Auto Scaling.	April 10, 2025
AWS managed policy updates	Application Auto Scaling updated the `AWSApplicationAutoscalingElastiCacheRGPolicy` policy.	April 10, 2025
Guide changes	New topic in the *Application Auto Scaling User Guide* helps you get started using predictive scaling with Application Auto Scaling. See Application Auto Scaling predictive scaling.	November 21, 2024
AWS managed policy updates	Application Auto Scaling updated the `AWSApplicationAutoscalingECSServicePolicy` policy.	November 21, 2024
Add support for a pool of WorkSpaces	Use Application Auto Scaling to scale a pool of WorkSpaces. For more information, see	June 27, 2024

	Amazon WorkSpaces and Application Auto Scaling. The topic Application Auto Scaling updates to AWS managed policies has been updated to list a new managed policy for the integration with WorkSpaces.	
Guide changes	Updated the *Maximum number of scalable targets per resource type* entry in the quotas documentation. See Quotas for Application Auto Scaling.	January 16, 2024
Support for SageMaker AI inference components	Use Application Auto Scaling to scale copies of an inference component.	November 29, 2023
AWS managed policy updates	Application Auto Scaling updated the `AWSApplicationAutoscalingSageMakerEndpointPolicy` policy.	November 13, 2023
Support for SageMaker AI Serverless provisioned concurrency	Use Application Auto Scaling to scale the provisioned concurrency of a serverless endpoint.	May 9, 2023
Categorize your scalable targets using tags	You can now assign metadata to your Application Auto Scaling scalable targets in the form of tags. See Tagging support for Application Auto Scaling.	March 20, 2023

Support for CloudWatch metric math	You can now use metric math when you create target tracking scaling policies. With metric math, you can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. See Create a target tracking scaling policy for Application Auto Scaling using metric math.	March 14, 2023
Reasons for not scaling	You can now retrieve the machine-readable reasons for Application Auto Scaling not scaling your resources using the Application Auto Scaling API. See Scaling activities for Application Auto Scaling.	January 4, 2023
Guide changes	Updated the *Maximum number of scalable targets per resource type* entry in the quotas documentation. See Quotas for Application Auto Scaling.	May 6, 2022

Add support for Amazon Neptune clusters	Use Application Auto Scaling to scale the number of replicas in an Amazon Neptune DB cluster. For more information, see Amazon Neptune and Application Auto Scaling. The topic Application Auto Scaling updates to AWS managed policies has been updated to list a new managed policy for the integration with Neptune.	October 6, 2021
Application Auto Scaling now reports changes to its AWS managed policies	Beginning August 19, 2021, changes to managed policies are reported in the topic Application Auto Scaling updates to AWS managed policies. The first change listed is the addition of permissions needed for ElastiCache (Redis OSS).	August 19, 2021
Add support for ElastiCache (Redis OSS) replication groups	Use Application Auto Scaling to scale the number of node groups and the number of replicas per node group for an ElastiCache (Redis OSS) replication group (cluster) . For more information, see ElastiCache (Redis OSS) and Application Auto Scaling.	August 19, 2021

Guide changes	New IAM topics in the *Application Auto Scaling User Guide* help you troubleshoot access to Application Auto Scaling. For more informati on, see Identity and Access Management for Applicati on Auto Scaling. Also added new example IAM permissio ns policies for actions on target services and Amazon CloudWatch. For more information, see Example policies for working with the AWS CLI or an SDK.	February 23, 2021
Add support for local time zones	You can now create scheduled actions in the local time zone. If your time zone observes daylight saving time, it automatically adjusts for Daylight Saving Time (DST). For more information, see Scheduled scaling.	February 2, 2021
Guide changes	A new tutorial in the *Applicati on Auto Scaling User Guide* helps you understand how to use target tracking scaling policies and scheduled scaling to increase the availability of your application when using Application Auto Scaling.	October 15, 2020

Add support for Amazon Managed Streaming for Apache Kafka cluster storage	Use a target tracking scaling policy to scale out the amount of broker storage associated with an Amazon MSK cluster.	September 30, 2020
Add support for Amazon Comprehend entity recognizer endpoints	Use Application Auto Scaling to scale the number of inference units provisioned for your Amazon Comprehend entity recognizer endpoints.	September 28, 2020
Add support for Amazon Keyspaces (for Apache Cassandra) tables	Use Application Auto Scaling to scale the provisioned throughput (read and write capacity) of an Amazon Keyspaces table.	April 23, 2020
New "Security" chapter	A new Security chapter in the *Application Auto Scaling User Guide* helps you understand how to apply the shared responsibility model when using Applicati on Auto Scaling. As part of this update, the user guide chapter "Authentication and Access Control" has been replaced by a new, more useful section, Identity and Access Management for Application Auto Scaling.	January 16, 2020
Minor updates	Various improvements and corrections.	January 15, 2020

Add notification functionality	Application Auto Scaling now sends events to Amazon EventBridge and notifications to your AWS Health Dashboard when certain actions occur. For more information, see Application Auto Scaling monitoring.	December 20, 2019
Add support for AWS Lambda functions	Use Application Auto Scaling to scale the provisioned concurrency of a Lambda function.	December 3, 2019
Add support for Amazon Comprehend document classification endpoints	Use Application Auto Scaling to scale the throughput capacity of an Amazon Comprehend document classification endpoint.	November 25, 2019
Add WorkSpaces Applications support for target tracking scaling policies	Use target tracking scaling policies to scale the size of an WorkSpaces Applications fleet.	November 25, 2019
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and Application Auto Scaling. For migration considerations and instructions, see Application Auto Scaling and interface VPC endpoints.	November 22, 2019

Suspend and resume scaling	Added support for suspending and resuming scaling. For more information, see Suspending and resuming scaling for Application Auto Scaling.	August 29, 2019
Guide changes	Improved Application Auto Scaling documentation in the Scheduled scaling, Step scaling policies, and Target tracking scaling policies sections.	March 11, 2019
Add support for custom resources	Use Application Auto Scaling to scale custom resources provided by your own applications or services. For more information, see our GitHub repository.	July 9, 2018
Add support for SageMaker AI endpoint variants	Use Application Auto Scaling to scale the number of endpoint instances provisioned for a variant.	February 28, 2018

The following table describes important changes to the Application Auto Scaling documentation before January 2018.

Change	Description	Date
Add support for Aurora Replicas	Use Application Auto Scaling to scale the desired count. For more information, see Using Amazon Aurora Auto Scaling	November 17, 2017

Change	Description	Date
	with Aurora replicas in the *Amazon RDS User Guide*.	
Add support for scheduled scaling	Use scheduled scaling to scale resources at specific preset times or intervals. For more information, see Scheduled scaling for Application Auto Scaling.	November 8, 2017
Add support for target tracking scaling policies	Use target tracking scaling policies to set up dynamic scaling for your application in just a few simple steps. For more information, see Target tracking scaling policies for Application Auto Scaling.	July 12, 2017
Add support for provisioned read and write capacity for DynamoDB tables and global secondary indexes	Use Application Auto Scaling to scale provisioned throughput (read and write capacity). For more informati on, see Managing throughpu t capacity with DynamoDB Auto Scaling in the *Amazon DynamoDB Developer Guide*.	June 14, 2017
Add support for WorkSpaces Applications fleets	Use Application Auto Scaling to scale the size of the fleet. For more information, see Fleet Auto Scaling for WorkSpaces Applications in the *Amazon WorkSpaces Applications Administration Guide*.	March 23, 2017

Change	Description	Date
Add support for Amazon EMR clusters	Use Application Auto Scaling to scale the core and task nodes. For more information, see Using automatic scaling in Amazon EMR in the *Amazon EMR Management Guide*.	November 18, 2016
Add support for Spot Fleets	Use Application Auto Scaling to scale the target capacity. For more information, see Automatic scaling for Spot fleet in the *Amazon EC2 User Guide*.	September 1, 2016
Add support for Amazon ECS services	Use Application Auto Scaling to scale the desired count. For more information, see Service Auto Scaling in the *Amazon Elastic Container Service Developer Guide*.	August 9, 2016