



User Guide

Amazon CloudWatch Logs



Amazon CloudWatch Logs: User Guide

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

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

Table of Contents

What is Amazon CloudWatch Logs?	1
Features	1
Related AWS services	3
Pricing	4
Concepts	4
Billing and costs	6
Log classes	7
Supported features	7
Getting started	10
Prerequisites	10
Sign up for an AWS account	10
Create a user with administrative access	11
Set up the Command Line Interface	12
Using the unified CloudWatch agent	12
Using the previous CloudWatch agent	13
CloudWatch Logs agent prerequisites	14
Quick Start: Install the agent on a running EC2 Linux instance	14
Quick Start: Install the agent on an EC2 Linux instance at launch	21
Quick Start: Use CloudWatch Logs with Windows Server 2016 instances	25
Quick Start: Use CloudWatch Logs with Windows Server 2012 and Windows Server 2008 instances	36
Report the CloudWatch Logs agent status	46
Start the CloudWatch Logs agent	47
Stop the CloudWatch Logs agent	47
CloudWatch Logs agent reference	48
Quick Start with CloudFormation	59
Working with AWS SDKs	61
Log management	63
Data source discovery and management	63
What is CloudWatch Logs Data Sources?	64
How to get started	64
System fields	66
Accessing Data Sources	66
Relationship to log groups	67

Features enabled by data sources	67
Supported AWS services for data sources	68
Supported third-party sources for data sources	72
Analyzing log data with CloudWatch Logs Insights	76
Supported query languages	79
CloudWatch Logs Insights query language (Logs Insights QL)	79
OpenSearch Piped Processing Language (PPL)	138
OpenSearch Structured Query Language (SQL)	150
Use natural language to generate and update CloudWatch Logs Insights queries	162
Example queries	163
Opting out of using your data for service improvement	166
Supported logs and discovered fields	166
Fields in JSON logs	169
Create field indexes to improve query performance and reduce scan volume	171
Field index syntax and quotas	174
Create an account-level field index policy	179
Create a log-group level field index policy	180
Log group selection options when creating a query	181
Effects of deleting a field index policy	182
Use facets to group and explore logs	182
To run a facet-based query	183
To save a facet-based query	183
To create an account-level facet	184
Facet Management using APIs	184
Pattern analysis	185
Getting started with pattern analysis	185
Details about the pattern command	188
Save and re-run queries	188
Add query to dashboard or export query results	191
View running queries or query history	192
Encrypt query results with AWS Key Management Service	192
Limits	193
Step 1: Create an AWS KMS key	193
Step 2: Set permissions on the KMS key	194
Step 3: Associate a KMS key with your query results	195
Step 4: Disassociate a key from query results in the account	196

Generate a natural language summary from CloudWatch Logs Insights query results	196
How it works	196
Regional availability and data processing	196
Getting started	197
Permissions	198
Data privacy	198
Automating log analysis with scheduled queries	199
Understanding scheduled queries concepts	199
IAM role separation	199
Cross-region and cross-account usage	202
Schedule expressions and timezone handling	203
Choosing a query language	204
Destination selection and use cases	204
Query result format and structure	205
Schedule expression reference	207
Best practices	210
Getting started with scheduled queries	211
Creating a scheduled query	212
Viewing and managing scheduled queries	215
Viewing scheduled query execution history	217
Updating a scheduled query	218
Configuring S3 destinations for scheduled queries	220
Troubleshooting scheduled queries	221
Query execution fails with permission errors	221
Query times out	221
Destination processing fails	222
Invalid query errors	222
Query Concurrency Errors	223
Log anomaly detection	224
Severity and priority of anomalies and patterns	225
Anomaly visibility time	225
Suppressing an anomaly	226
Frequently asked questions	226
Using anomaly detection in CloudWatch Logs Insights	227
Enable anomaly detection on a log group	228
View anomalies that have been found	229

Create alarms on log anomaly detectors	232
Metrics published by log anomaly detectors	234
Encrypt an anomaly detector and its results with AWS KMS	234
Limits	235
Troubleshoot with CloudWatch Logs Live Tail	240
Start a Live Tail session using the AWS CLI	240
print-only	241
interactive	241
Start a Live Tail session in the console	243
Cross-account cross-Region log centralization	247
Data centralization concepts	247
Setting up log centralization	248
Prerequisites	249
Creating a centralization rule	249
Modifying a centralization rule	251
Viewing a centralization rule	252
Deleting a centralization rule	252
Monitoring centralization	253
Monitoring centralization in the console	253
Centralization monitoring	253
Monitoring rule health	254
Monitoring centralization API calls with AWS CloudTrail	254
Working with log groups and log streams	255
Create a log group	255
Send logs to a log group	256
View log data	256
Search log data using filter patterns	257
Search log entries using the console	257
Search log entries using the AWS CLI	258
Pivot from metrics to logs	258
Troubleshooting	259
Change log data retention	259
Protect log groups from deletion	260
Protect log groups from deletion	260
Enabling deletion protection	260
Tag log groups	261

Tag basics	262
Tracking costs using tagging	263
Tag restrictions	263
Tagging log groups using the AWS CLI	264
Tagging log groups using the CloudWatch Logs API	264
Encrypt log data using AWS KMS	265
Limits	266
Step 1: Create an AWS KMS key	193
Step 2: Set permissions on the KMS key	267
Step 3: Associate a KMS key with a log group	239
Step 4: Disassociate key from a log group	239
KMS keys and encryption context	272
Help protect sensitive log data with masking	275
Understanding data protection policies	278
IAM permissions required to create or work with a data protection policy	281
Create an account-wide data protection policy	286
Create a data protection policy for a single log group	289
View unmasked data	292
Audit findings reports	293
Types of data that you can protect	294
Transform logs during ingestion	338
Create and manage log transformers	340
Create an account-level transformer policy	341
Edit or delete an account-level transformer policy	342
Create a log-group-level log transformer from scratch	342
Create a log-group-level transformer by copying an existing one	344
Edit a log-group-level transformer	344
Delete a log-group-level transformer	345
Configurable parser-type processors	346
parseJSON	346
grok	348
csv	379
parseKeyValue	380
Built-in processors for AWS vended logs	382
parseWAF	382
parsePostgres	385

parseCloudfront	386
parseRoute53	388
parseVPC	389
parseToOCSF	390
String mutate processors	391
lowerCaseString	392
upperCaseString	393
splitString	394
substituteString	396
trimString	399
JSON mutate processors	400
addKeys	401
deleteKeys	402
moveKeys	403
renameKeys	405
copyValue	406
listToMap	408
Datatype converter processors	413
typeConverter	414
datetimeConverter	415
Transformation metrics and errors	417
Analyze with Amazon OpenSearch Service	418
Step 1: Create the integration with OpenSearch Service	420
Required permissions	420
Create the integration	427
Step 2: Create vended logs dashboards	429
View, edit, or delete vended logs dashboards	430
View vended logs dashboards in CloudWatch Logs or OpenSearch Service	430
Grant dashboard viewing access to additional IAM roles or IAM users	431
Edit dashboard configuration	431
Delete a vended log dashboard	431
Delete all vended log dashboard integration with OpenSearch Service	432
IAM policies for users	433
Permissions that the integration needs	434
Access logs with S3 Tables Integration	437
Understanding S3 Tables Integration	437

Core Components	437
Data flow to S3 tables	437
When to Use S3 Tables Integration	438
Prerequisites	438
IAM permissions	439
KMS key policy (for encrypted data)	440
Getting Started	441
Metric filters	443
Concepts	444
Filter pattern syntax for metric filters	445
Configuring metric values for a metric filter	446
Publishing dimensions with metric from log events	447
Using values in log events to increment a metric's value	450
Creating metric filters	451
Create a metric filter for a log group	452
Example: Count log events	453
Example: Count occurrences of a term	454
Example: Count HTTP 404 codes	456
Example: Count HTTP 4xx codes	458
Example: Extract fields from an Apache log and assign dimensions	460
Listing metric filters	462
Deleting a metric filter	463
Subscription filters	464
Concepts	465
Log group-level subscription filters	467
Example 1: Subscription filters with Amazon Kinesis Data Streams	467
Example 2: Subscription filters with AWS Lambda	473
Example 3: Subscription filters with Amazon Data Firehose	477
Example 4: Subscription filters with Amazon OpenSearch Service	484
Account-level subscription filters	485
Example 1: Subscription filters with Amazon Kinesis Data Streams	486
Example 2: Subscription filters with AWS Lambda	492
Example 3: Subscription filters with Amazon Data Firehose	496
Cross-account cross-Region subscriptions	504
Cross-account cross-Region log data sharing using Amazon Kinesis Data Streams	505
Cross-account cross-Region log data sharing using Firehose	525

Cross-account cross-Region account-level subscriptions using Amazon Kinesis Data Streams	539
Cross-account cross-Region account-level subscriptions using Firehose	558
Confused deputy prevention	570
Log recursion prevention	571
Filter pattern syntax	573
Supported regular expressions	573
Match terms using regular expressions	576
Match terms in unstructured log events	577
Match terms in JSON log events	580
Match terms in space-delimited log events	589
Enable logging from AWS services	594
Logging that requires additional permissions [V1]	606
Logs sent to CloudWatch Logs	606
Logs sent to Amazon S3	608
Logs sent to Firehose	613
Logging that requires additional permissions [V2]	615
Logs sent to CloudWatch Logs	616
Logs sent to Amazon S3	619
Logs sent to Firehose	623
Traces sent to X-Ray	626
Service-specific permissions	628
Console-specific permissions	629
Cross-account delivery example	630
Create delivery source	631
Configure delivery to an Amazon S3 bucket	632
Configure delivery to a Firehose stream	635
Cross-service confused deputy prevention	637
Exporting log data to Amazon S3	639
Concepts	640
Export log data to Amazon S3 using the console	641
Same-account export (console)	641
Cross-account export (console)	648
Export log data to Amazon S3 using the AWS CLI	657
Same-account export (CLI)	657
Cross-account export (CLI)	664

Describe export tasks (CLI)	673
Cancel an export task (CLI)	675
Streaming data to OpenSearch Service	676
Prerequisites	676
Subscribe a log group to OpenSearch Service	677
Code examples	679
Basics	680
Actions	681
Scenarios	744
Run a large query	745
Use scheduled events to invoke a Lambda function	783
Security	786
Data protection	787
Encryption at rest	788
Encryption in transit	788
Identity and access management	788
Authentication	788
Access control	789
Overview of managing access	789
Using identity-based policies (IAM policies)	795
Compliance validation	820
Resilience	821
Infrastructure security	821
Interface VPC endpoints	821
Availability	822
Creating a VPC endpoint for CloudWatch Logs	822
Testing the connection between your VPC and CloudWatch Logs	822
Controlling access to your CloudWatch Logs VPC endpoint	823
Support for VPC context keys	824
Logging API and console operations with AWS CloudTrail	825
Query generation information in CloudTrail	829
Understanding log file entries	831
Monitoring usage with CloudWatch metrics	833
CloudWatch Logs metrics	833
Dimensions for CloudWatch Logs metrics	838
Log transformer metrics and dimensions	839

CloudWatch Logs service usage metrics	840
Service quotas	843
Managing your CloudWatch Logs service quotas	851
Document history	853
AWS Glossary	867

What is Amazon CloudWatch Logs?

You can use Amazon CloudWatch Logs to monitor, store, and access your log files from Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS CloudTrail, Route 53, and other sources.

CloudWatch Logs enables you to centralize the logs from all of your systems, applications, and AWS services that you use, in a single, highly scalable service. You can then easily view them, search them for specific error codes or patterns, filter them based on specific fields, or archive them securely for future analysis. CloudWatch Logs enables you to see all of your logs, regardless of their source, as a single and consistent flow of events ordered by time.

CloudWatch Logs also supports querying your logs with a powerful query language, auditing and masking sensitive data in logs, and generating metrics from logs using filters or an embedded log format.

CloudWatch Logs supports two *log classes*. Log groups in the *CloudWatch Logs Standard log class* support all CloudWatch Logs features. Log groups in the *CloudWatch Logs Infrequent Access log class* incur lower ingestion charges and support a subset of the Standard class capabilities. For more information, see [Log classes](#).

Features

- **Two log classes for flexibility** – CloudWatch Logs offers two log classes so that you can have a cost-effective option for logs that you access infrequently. You also have a full-featured option for logs that require real-time monitoring or other features. For more information, see [Log classes](#).
- **Query your log data** – You can use CloudWatch Logs Insights to interactively search and analyze your log data. You can perform queries to help you more efficiently and effectively respond to operational issues. CloudWatch Logs Insights includes a purpose-built query language with a few simple but powerful commands. We provide sample queries, command descriptions, query autocompletion, and log field discovery to help you get started. Sample queries are included for several types of AWS service logs. To get started, see [Analyzing log data with CloudWatch Logs Insights](#).
- **Create field indexes to make queries more efficient** – You can create *field indexes* of fields in your log events. When you then use a field index in a CloudWatch Logs Insights query, the query attempts to skip processing log events that are known to not include the indexed field. This

query reduces the scan volume of your queries, making it possible to return results faster. To get started, see [Create field indexes to improve query performance and reduce scan volume](#).

- **Detect and debug using Live Tail** – You can use Live Tail to quickly troubleshoot incidents by viewing a streaming list of new log events as they are ingested. You can view, filter, and highlight ingested logs in near real time, helping you to detect and resolve issues quickly. You can filter the logs based on terms you specify, and also highlight logs that contain specified terms to help you quickly find what you are looking for. For more information, see [Troubleshoot with CloudWatch Logs Live Tail](#).
- **Monitor logs from Amazon EC2 instances** – You can use CloudWatch Logs to monitor applications and systems using log data. For example, CloudWatch Logs can track the number of errors that occur in your application logs and send you a notification whenever the rate of errors exceeds a threshold you specify. CloudWatch Logs uses your log data for monitoring; so, no code changes are required. For example, you can monitor application logs for specific literal terms (such as "NullPointerException") or count the number of occurrences of a literal term at a particular position in log data (such as "404" status codes in an Apache access log). When the term you are searching for is found, CloudWatch Logs reports the data to a CloudWatch metric that you specify. Log data is encrypted while in transit and while it is at rest. To get started, see [Getting started with CloudWatch Logs](#).
- **Monitor AWS CloudTrail logged events** – You can create alarms in CloudWatch and receive notifications of particular API activity as captured by CloudTrail and use the notification to perform troubleshooting. To get started, see [Sending CloudTrail Events to CloudWatch Logs](#) in the *AWS CloudTrail User Guide*.
- **Audit and mask sensitive data** – If you have sensitive data in your logs, you can help safeguard it with *data protection policies*. These policies let you audit and mask the sensitive data. If you enable data protection, then by default, sensitive data that matches the data identifiers you select is masked. For more information, see [Help protect sensitive log data with masking](#).
- **Log retention** – By default, logs are kept indefinitely and never expire. You can adjust the retention policy for each log group, keeping the indefinite retention, or choosing a retention period between 10 years and one day.
- **Deletion protection** – A safeguard that prevents accidental deletion of log groups and their log streams. When enabled on a log group, deletion protection blocks all deletion operations until it is explicitly disabled. By default, deletion protection is not enabled. This optional feature helps protect critical operational and compliance data from unintended removal, such as log groups that contain audit data, and production application logs for troubleshooting and analysis.

- **Archive log data** – You can use CloudWatch Logs to store your log data in highly durable storage. The CloudWatch Logs agent makes it easy to quickly send both rotated and non-rotated log data off of a host and into the log service. You can then access the raw log data when you need it.
- **Log Route 53 DNS queries** – You can use CloudWatch Logs to log information about the DNS queries that Route 53 receives. For more information, see [Logging DNS Queries](#) in the *Amazon Route 53 Developer Guide*.
- **Centralize logs across accounts and regions** – You can use CloudWatch Logs Centralization to define cross-account and cross-region centralization rules that replicate log data ingested across multi-account and region environments into a central region and account. You can configure a backup region within the central account for increased resiliency, define encryption settings for newly created log groups in the central account, and create metric and subscription filters on log events from specific source accounts and regions with enhanced filtering capabilities.

Related AWS services

The following services are used in conjunction with CloudWatch Logs:

- **AWS CloudTrail** is a web service that enables you to monitor the calls made to the CloudWatch Logs API for your account, including calls made by the AWS Management Console, AWS Command Line Interface (AWS CLI), and other services. When CloudTrail logging is turned on, CloudTrail captures API calls in your account and delivers the log files to the Amazon S3 bucket that you specify. Each log file can contain one or more records, depending on how many actions must be performed to satisfy a request. For more information about AWS CloudTrail, see [What Is AWS CloudTrail?](#) in the *AWS CloudTrail User Guide*. For an example of the type of data that CloudWatch writes into CloudTrail log files, see [Logging CloudWatch Logs API and console operations in AWS CloudTrail](#).
- **AWS Identity and Access Management (IAM)** is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). For more information, see [What Is IAM?](#) in the *IAM User Guide*.
- **Amazon Kinesis Data Streams** is a web service you can use for rapid and continuous data intake and aggregation. The type of data used includes IT infrastructure log data, application logs, social media, market data feeds, and web clickstream data. Because the response time for the data intake and processing is in real time, processing is typically lightweight. For more

information, see [What is Amazon Kinesis Data Streams?](#) in the *Amazon Kinesis Data Streams Developer Guide*.

- **AWS Lambda** is a web service you can use to build applications that respond quickly to new information. Upload your application code as Lambda functions and Lambda runs your code on high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging. All you need to do is supply your code in one of the languages that Lambda supports. For more information, see [What is AWS Lambda?](#) in the *AWS Lambda Developer Guide*.

Pricing

When you sign up for AWS, you can get started with CloudWatch Logs for free using the [AWS Free Tier](#).

Standard rates apply for logs stored by other services using CloudWatch Logs (for example, Amazon VPC flow logs and Lambda logs).

For more information about pricing, see [Amazon CloudWatch Pricing](#).

For more information about how to analyze your costs and usage for CloudWatch Logs and CloudWatch, and for best practices about how to reduce your costs, see [CloudWatch billing and cost](#).

Amazon CloudWatch Logs concepts

The terminology and concepts that are central to your understanding and use of CloudWatch Logs are described below.

Log class

CloudWatch Logs offers two classes of log groups. The Standard log class is a full-featured option for logs that require real-time monitoring or logs that you access frequently. The Infrequent Access log class is a lower-cost option for logs that you access less frequently. It supports a subset of the Standard log class capabilities.

Log events

A log event is a record of some activity recorded by the application or resource being monitored. The log event record that CloudWatch Logs understands contains two properties:

the timestamp of when the event occurred, and the raw event message. Event messages must be UTF-8 encoded.

Log streams

A log stream is a sequence of log events that share the same source. More specifically, a log stream is generally intended to represent the sequence of events coming from the application instance or resource being monitored. For example, a log stream may be associated with an Apache access log on a specific host. When you no longer need a log stream, you can delete it using the [aws logs delete-log-stream](#) command.

Log groups

Log groups define groups of log streams that share the same retention, monitoring, and access control settings. Each log stream has to belong to one log group. For example, if you have a separate log stream for the Apache access logs from each host, you could group those log streams into a single log group called `MyWebsite.com/Apache/access_log`.

There is no limit on the number of log streams that can belong to one log group.

Metric filters

You can use metric filters to extract metric observations from ingested events and transform them to data points in a CloudWatch metric. Metric filters are assigned to log groups, and all of the filters assigned to a log group are applied to their log streams.

Retention settings

Retention settings can be used to specify how long log events are kept in CloudWatch Logs. Expired log events get deleted automatically. Just like metric filters, retention settings are also assigned to log groups, and the retention assigned to a log group is applied to their log streams.

Deletion protection

Deletion protection is a safeguard that prevents accidental deletion of log groups and their log streams. When enabled on a log group, deletion protection blocks all deletion operations until it is explicitly disabled. By default, deletion protection is not enabled. This optional feature helps protect critical operational and compliance data from unintended removal, such as log groups that contain audit data, and production application logs for troubleshooting and analysis.

Amazon CloudWatch Logs billing and cost

For detailed information about how to analyze your costs and usage for CloudWatch Logs and CloudWatch, and for best practices about how to reduce your costs, see [CloudWatch billing and cost](#).

For more information about pricing, see [Amazon CloudWatch Pricing](#).

When you sign up for AWS, you can get started with CloudWatch Logs for free using the [AWS Free Tier](#).

Standard rates apply for logs stored by other services using CloudWatch Logs (for example, Amazon VPC flow logs and Lambda logs).

Log classes

CloudWatch Logs offers two classes of log groups:

- The *CloudWatch Logs Standard* log class is a full-featured option for logs that require real-time monitoring or logs that you access frequently.
- The *CloudWatch Logs Infrequent Access* log class is a new log class that you can use to cost-effectively consolidate your logs. This log class offers a subset of CloudWatch Logs capabilities including managed ingestion, storage, cross-account log analytics, and encryption with a lower ingestion price per GB. The Infrequent Access log class is ideal for ad-hoc querying and after-the-fact forensic analysis on infrequently accessed logs.

Note

For charges, the Standard and Infrequent Access log classes differ in ingestion costs only. Storage charges and CloudWatch Logs Insights charges are the same in each log class.

For more information about CloudWatch Logs pricing, see [Amazon CloudWatch Pricing](#).

Important

After a log group is created, its log class can't be changed.

Supported features

The following table lists the features for each log class.

Feature	Standard	Infrequent Access	
Fully managed log ingestion and storage	Yes ✓	Yes ✓	
Cross-account features	Yes ✓	Yes ✓	
Encryption with AWS KMS	Yes ✓	Yes ✓	

Feature	Standard	Infrequent Access
CloudWatch Logs Insights query commands	Yes ✓	Yes ✓ (Most commands—see Logs Insights QL commands supported in log classes.)
CloudWatch Logs Insights discovered fields	Yes ✓	Yes ✓
Facets	Yes ✓	No
Using CloudWatch Pipeline to transform logs	Yes ✓	Yes ✓
Export to Amazon S3	Yes ✓	Yes ✓
S3 Tables Integration	Yes ✓	Yes ✓
Scheduled Queries	Yes ✓	Yes ✓
Using OpenSearch PPL or OpenSearch SQL to query in CloudWatch Logs Insights;	Yes ✓	No
Natural language query assist	Yes ✓	No
CloudWatch Logs Anomaly Detection	Yes ✓	No
Live Tail	Yes ✓	No
Field indexing	Yes ✓	No
Compare to previous time range	Yes ✓	No
Subscription filters	Yes ✓	No

Feature	Standard	Infrequent Access	
GetLogEvents and FilterLogEvents API operations	Yes ✓	Not supported . Use CloudWatch Logs Insights to view log events stored in log groups in the Infrequent Access log class.	
Metric filters	Yes ✓	No	
Container Insights log ingestion	Yes ✓	No	
Lambda Insights log ingestion	Yes ✓	No	
Sensitive data protection with masking	Yes ✓	No	
Embedded metrics format	Yes ✓	No	

Note

In addition to these two log classes, there is a Delivery log class. Use the Delivery log class only for delivering AWS Lambda logs to store in Amazon S3 or Amazon Data Firehose. Log events in log groups in the Delivery class are kept in CloudWatch Logs for only one day. This log class doesn't offer rich CloudWatch Logs capabilities such as CloudWatch Logs Insights queries.

Getting started with CloudWatch Logs

To collect logs from your Amazon EC2 instances and on-premises servers into CloudWatch Logs, use the unified CloudWatch agent. It enables you to collect both logs and advanced metrics with one agent. It offers support across operating systems, including servers running Windows Server. This agent also provides better performance.

If you're using the unified CloudWatch agent to collect CloudWatch metrics, it enables the collection of additional system metrics, for in-guest visibility. It also supports collecting custom metrics using StatsD or collectd.

For more information, see [Installing the CloudWatch Agent](#) in the *Amazon CloudWatch User Guide*.

The older CloudWatch Logs agent, which supports only the collection of logs from servers running Linux, is deprecated and is no longer supported. For information about migrating from the older CloudWatch Logs agent to the unified agent, see [Create the CloudWatch agent configuration file with the wizard](#).

Contents

- [Prerequisites](#)
- [Use the unified CloudWatch agent to get started with CloudWatch Logs](#)
- [Use the previous CloudWatch agent to get started with CloudWatch Logs](#)
- [Quick Start: Use CloudFormation to get started with CloudWatch Logs](#)

Prerequisites

To use Amazon CloudWatch Logs you need an AWS account. Your AWS account allows you to use services (for example, Amazon EC2) to generate logs that you can view in the CloudWatch console, a web-based interface. In addition, you can install and configure the AWS Command Line Interface (AWS CLI).

Sign up for an AWS account

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

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.

2. Follow the online instructions.

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

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

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Set up the Command Line Interface

You can use the AWS CLI to perform CloudWatch Logs operations.

For information about how to install and configure the AWS CLI, see [Getting Set Up with the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

Use the unified CloudWatch agent to get started with CloudWatch Logs

For more information about using the unified CloudWatch agent to get started with CloudWatch Logs, see [Collect Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent](#) in the *Amazon CloudWatch User Guide*. You complete the steps listed in this section to install, configure, and start the agent. If you are not using the agent to also collect CloudWatch metrics, you can ignore any sections that refer to metrics.

If you are currently using the older CloudWatch Logs agent and want to migrate to using the new unified agent, we recommend that you use the wizard included in the new agent package. This wizard can read your current CloudWatch Logs agent configuration file and set up the CloudWatch agent to collect the same logs. For more information about the wizard, see [Create the CloudWatch Agent Configuration File with the Wizard](#) in the *Amazon CloudWatch User Guide*.

Use the previous CloudWatch agent to get started with CloudWatch Logs

Important

CloudWatch includes a unified CloudWatch agent that can collect both logs and metrics from EC2 instances and on-premises servers. The older logs-only agent is deprecated and is no longer supported.

For information about migrating from the older logs-only agent to the unified agent, see [Create the CloudWatch agent configuration file with the wizard](#).

The rest of this section explains the use of the older CloudWatch Logs agent for customers who are still using it.

Using the CloudWatch Logs agent, you can publish log data from Amazon EC2 instances running Linux or Windows Server, and logged events from AWS CloudTrail. We recommend instead using the CloudWatch unified agent to publish your log data. For more information about the new agent, see [Collect Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent](#) in the *Amazon CloudWatch User Guide*.

Contents

- [CloudWatch Logs agent prerequisites](#)
- [Quick Start: Install and configure the CloudWatch Logs agent on a running EC2 Linux instance](#)
- [Quick Start: Install and configure the CloudWatch Logs agent on an EC2 Linux instance at launch](#)
- [Quick Start: Enable your Amazon EC2 instances running Windows Server 2016 to send logs to CloudWatch Logs using the CloudWatch Logs agent](#)
- [Quick Start: Enable your Amazon EC2 instances running Windows Server 2012 and Windows Server 2008 to send logs to CloudWatch Logs](#)
- [Report the CloudWatch Logs agent status](#)

- [Start the CloudWatch Logs agent](#)
- [Stop the CloudWatch Logs agent](#)
- [CloudWatch Logs agent reference](#)

CloudWatch Logs agent prerequisites

The CloudWatch Logs agent requires Python version 2.7, 3.0, or 3.3, and any of the following versions of Linux:

- Amazon Linux version 2014.03.02 or later. Amazon Linux 2 is not supported
- Ubuntu Server version 12.04, 14.04, or 16.04
- CentOS version 6, 6.3, 6.4, 6.5, or 7.0
- Red Hat Enterprise Linux (RHEL) version 6.5 or 7.0
- Debian 8.0

Quick Start: Install and configure the CloudWatch Logs agent on a running EC2 Linux instance

Important

The older logs agent is deprecated. CloudWatch includes a unified agent that can collect both logs and metrics from EC2 instances and on-premises servers. For more information, see [Getting started with CloudWatch Logs](#).

For information about migrating from the older CloudWatch Logs agent to the unified agent, see [Create the CloudWatch agent configuration file with the wizard](#).

The older logs agent supports only versions 2.6 to 3.5 of Python. Additionally, the older CloudWatch Logs agent doesn't support Instance Metadata Service Version 2 (IMDSv2). If your server uses IMDSv2, you must use the newer unified agent instead of the older CloudWatch Logs agent.

The rest of this section explains the use of the older CloudWatch Logs agent for customers who are still using it.

Tip

CloudWatch includes a new unified agent that can collect both logs and metrics from EC2 instances and on-premises servers. If you are not already using the older CloudWatch Logs agent, we recommend that you use the newer unified CloudWatch agent. For more information, see [Getting started with CloudWatch Logs](#).

Additionally, the older agent doesn't support Instance Metadata Service Version 2 (IMDSv2). If your server uses IMDSv2, you must use the newer unified agent instead of the older CloudWatch Logs agent.

The rest of this section explains the use of the older CloudWatch Logs agent.

Configure the older CloudWatch Logs agent on a running EC2 Linux instance

You can use the CloudWatch Logs agent installer on an existing EC2 instance to install and configure the CloudWatch Logs agent. After installation is complete, logs automatically flow from the instance to the log stream you create while installing the agent. The agent confirms that it has started and it stays running until you disable it.

In addition to using the agent, you can also publish log data using the AWS CLI, CloudWatch Logs SDK, or the CloudWatch Logs API. The AWS CLI is best suited for publishing data at the command line or through scripts. The CloudWatch Logs SDK is best suited for publishing log data directly from applications or building your own log publishing application.

Step 1: Configure your IAM role or user for CloudWatch Logs

The CloudWatch Logs agent supports IAM roles and users. If your instance already has an IAM role associated with it, make sure that you include the IAM policy below. If you don't already have an IAM role assigned to your instance, you can use your IAM credentials for the next steps or you can assign an IAM role to that instance. For more information, see [Attaching an IAM Role to an Instance](#).

To configure your IAM role or user for CloudWatch Logs

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose the role by selecting the role name (do not select the check box next to the name).
4. Choose **Attach Policies, Create Policy**.

A new browser tab or window opens.

5. Choose the **JSON** tab and type the following JSON policy document.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

6. When you are finished, choose **Review policy**. The Policy Validator reports any syntax errors.
7. On the **Review Policy** page, type a **Name** and a **Description** (optional) for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.
8. Close the browser tab or window, and return to the **Add permissions** page for your role. Choose **Refresh**, and then choose the new policy to attach it to your role.
9. Choose **Attach Policy**.

Step 2: Install and configure CloudWatch Logs on an existing Amazon EC2 instance

The process for installing the CloudWatch Logs agent differs depending on whether your Amazon EC2 instance is running Amazon Linux, Ubuntu, CentOS, or Red Hat. Use the steps appropriate for the version of Linux on your instance.

To install and configure CloudWatch Logs on an existing Amazon Linux instance

Starting with Amazon Linux AMI 2014.09, the CloudWatch Logs agent is available as an RPM installation with the `awslogs` package. Earlier versions of Amazon Linux can access the `awslogs` package by updating their instance with the `sudo yum update -y` command. By installing the `awslogs` package as an RPM instead of the using the CloudWatch Logs installer, your instance receives regular package updates and patches from AWS without having to manually reinstall the CloudWatch Logs agent.

Warning

Do not update the CloudWatch Logs agent using the RPM installation method if you previously used the Python script to install the agent. Doing so may cause configuration issues that prevent the CloudWatch Logs agent from sending your logs to CloudWatch.

1. Connect to your Amazon Linux instance. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

For more information about connection issues, see [Troubleshooting Connecting to Your Instance](#) in the *Amazon EC2 User Guide*.

2. Update your Amazon Linux instance to pick up the latest changes in the package repositories.

```
sudo yum update -y
```

3. Install the `awslogs` package. This is the recommended method for installing `awslogs` on Amazon Linux instances.

```
sudo yum install -y awslogs
```

4. Edit the `/etc/awslogs/awslogs.conf` file to configure the logs to track. For more information about editing this file, see [CloudWatch Logs agent reference](#).
5. By default, the `/etc/awslogs/awsccli.conf` points to the `us-east-1` Region. To push your logs to a different Region, edit the `awsccli.conf` file and specify that Region.
6. Start the `awslogs` service.

```
sudo service awslogs start
```

If you are running Amazon Linux 2, start the `awslogs` service with the following command.

```
sudo systemctl start awslogsd
```

7. (Optional) Check the `/var/log/awslogs.log` file for errors logged when starting the service.
8. (Optional) Run the following command to start the `awslogs` service at each system boot.

```
sudo chkconfig awslogs on
```

If you are running Amazon Linux 2, use the following command to start the service at each system boot.

```
sudo systemctl enable awslogsd.service
```

9. You should see the newly created log group and log stream in the CloudWatch console after the agent has been running for a few moments.

For more information, see [View log data sent to CloudWatch Logs](#).

To install and configure CloudWatch Logs on an existing Ubuntu Server, CentOS, or Red Hat instance

If you're using an AMI running Ubuntu Server, CentOS, or Red Hat, use the following procedure to manually install the CloudWatch Logs agent on your instance.

1. Connect to your EC2 instance. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

For more information about connection issues, see [Troubleshooting Connecting to Your Instance](#) in the *Amazon EC2 User Guide*.

2. Run the CloudWatch Logs agent installer using one of two options. You can run it directly from the internet, or download the files and run it standalone.

Note

If you are running CentOS 6.x, Red Hat 6.x, or Ubuntu 12.04, use the steps for downloading and running the installer standalone. Installing the CloudWatch Logs agent directly from the internet is not supported on these systems.

Note

On Ubuntu, run `apt-get update` before running the commands below.

To run it directly from the internet, use the following commands and follow the prompts:

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
```

```
sudo python ./awslogs-agent-setup.py --region us-east-1
```

If the preceding command does not work, try the following:

```
sudo python3 ./awslogs-agent-setup.py --region us-east-1
```

To download and run it standalone, use the following commands and follow the prompts:

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
```

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/AgentDependencies.tar.gz -O
```

```
tar xvf AgentDependencies.tar.gz -C /tmp/
```

```
sudo python ./awslogs-agent-setup.py --region us-east-1 --dependency-path /tmp/AgentDependencies
```

You can install the CloudWatch Logs agent by specifying the us-east-1, us-west-1, us-west-2, ap-south-1, ap-northeast-2, ap-southeast-1, ap-southeast-2, ap-northeast-1, eu-central-1, eu-west-1, or sa-east-1 Regions.

 Note

For more information about the current version and the version history of `awslogs-agent-setup`, see [CHANGELOG.txt](#).

The CloudWatch Logs agent installer requires certain information during setup. Before you start, you need to know which log file to monitor and its time stamp format. You should also have the following information ready.

Item	Description
AWS access key ID	Press Enter if using an IAM role. Otherwise, enter your AWS access key ID.
AWS secret access key	Press Enter if using an IAM role. Otherwise, enter your AWS secret access key.
Default Region name	Press Enter. The default is us-east-2. You can set this to us-east-1, us-west-1, us-west-2, ap-south-1, ap-northeast-2, ap-southeast-1, ap-southeast-2, ap-northeast-1, eu-central-1, eu-west-1, or sa-east-1.
Default output format	Leave blank and press Enter.
Path of log file to upload	The location of the file that contains the log data to send. The installer suggests a path for you.
Destination Log Group name	The name for your log group. The installer suggests a log group name for you.
Destination Log Stream name	By default, this is the name of the host. The installer suggests a host name for you.

Item	Description
Timestamp format	Specify the format of the time stamp within the specified log file. Choose custom to specify your own format.
Initial position	How data is uploaded. Set this to start_of_file to upload everything in the data file. Set to end_of_file to upload only newly appended data.

After you have completed these steps, the installer asks about configuring another log file. You can run the process as many times as you like for each log file. If you have no more log files to monitor, choose **N** when prompted by the installer to set up another log. For more information about the settings in the agent configuration file, see [CloudWatch Logs agent reference](#).

 Note

Configuring multiple log sources to send data to a single log stream is not supported.

3. You should see the newly created log group and log stream in the CloudWatch console after the agent has been running for a few moments.

For more information, see [View log data sent to CloudWatch Logs](#).

Quick Start: Install and configure the CloudWatch Logs agent on an EC2 Linux instance at launch

 Tip

The older CloudWatch Logs agent discussed in this section is on the path to deprecation. We strongly recommend that you instead use the new unified CloudWatch agent that can collect both logs and metrics. Additionally, the older CloudWatch Logs agent requires Python 3.3 or earlier, and these versions are not installed on new EC2 instances by default. For more information about the unified CloudWatch agent, see [Installing the CloudWatch Agent](#).

The rest of this section explains the use of the older CloudWatch Logs agent.

Installing the older CloudWatch Logs agent on an EC2 Linux instance at launch

You can use Amazon EC2 user data, a feature of Amazon EC2 that allows parametric information to be passed to the instance on launch, to install and configure the CloudWatch Logs agent on that instance. To pass the CloudWatch Logs agent installation and configuration information to Amazon EC2, you can provide the configuration file in a network location such as an Amazon S3 bucket.

Configuring multiple log sources to send data to a single log stream is not supported.

Prerequisite

Create an agent configuration file that describes all your log groups and log streams. This is a text file that describes the log files to monitor as well as the log groups and log streams to upload them to. The agent consumes this configuration file and starts monitoring and uploading all the log files described in it. For more information about the settings in the agent configuration file, see [CloudWatch Logs agent reference](#).

The following is a sample agent configuration file for Amazon Linux 2

```
[general]
state_file = /var/lib/awslogs/state/agent-state

[/var/log/messages]
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {instance_id}
datetime_format = %b %d %H:%M:%S
```

The following is a sample agent configuration file for Ubuntu

```
[general]
state_file = /var/awslogs/state/agent-state

[/var/log/syslog]
file = /var/log/syslog
log_group_name = /var/log/syslog
log_stream_name = {instance_id}
datetime_format = %b %d %H:%M:%S
```

To configure your IAM role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.

2. In the navigation pane, choose **Policies, Create Policy**.
3. On the **Create Policy** page, for **Create Your Own Policy**, choose **Select**. For more information about creating custom policies, see [IAM Policies for Amazon EC2](#) in the *Amazon EC2 User Guide*.
4. On the **Review Policy** page, for **Policy Name**, type a name for the policy.
5. For **Policy Document**, paste in the following policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    }
  ]
}
```

6. Choose **Create Policy**.
7. In the navigation pane, choose **Roles, Create New Role**.
8. On the **Set Role Name** page, type a name for the role and then choose **Next Step**.
9. On the **Select Role Type** page, choose **Select** next to **Amazon EC2**.

10. On the **Attach Policy** page, in the table header, choose **Policy Type, Customer Managed**.
11. Select the IAM policy that you created and then choose **Next Step**.
12. Choose **Create Role**.

For more information about users and policies, see [IAM Users and Groups](#) and [Managing IAM Policies](#) in the *IAM User Guide*.

To launch a new instance and enable CloudWatch Logs

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch Instance**.

For more information, see [Launching an Instance](#) in *Amazon EC2 User Guide*.

3. On the **Step 1: Choose an Amazon Machine Image (AMI)** page, select the Linux instance type to launch, and then on the **Step 2: Choose an Instance Type** page, choose **Next: Configure Instance Details**.

Make sure that [cloud-init](#) is included in your Amazon Machine Image (AMI). Amazon Linux AMIs, and AMIs for Ubuntu and RHEL already include cloud-init, but CentOS and other AMIs in the AWS Marketplace might not.

4. On the **Step 3: Configure Instance Details** page, for **IAM role**, select the IAM role that you created.
5. Under **Advanced Details**, for **User data**, paste the following script into the box. Then update that script by changing the value of the **-c** option to the location of your agent configuration file:

```
#!/bin/bash
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-
setup.py -O
chmod +x ./awslogs-agent-setup.py
./awslogs-agent-setup.py -n -r us-east-1 -c s3://amzn-s3-demo-bucket/my-config-file
```

6. Make any other changes to the instance, review your launch settings, and then choose **Launch**.
7. You should see the newly created log group and log stream in the CloudWatch console after the agent has been running for a few moments.

For more information, see [View log data sent to CloudWatch Logs](#).

Quick Start: Enable your Amazon EC2 instances running Windows Server 2016 to send logs to CloudWatch Logs using the CloudWatch Logs agent

Tip

CloudWatch includes a new unified agent that can collect both logs and metrics from EC2 instances and on-premises servers. We recommend that you use the newer unified CloudWatch agent. For more information, see [Getting started with CloudWatch Logs](#). The rest of this section explains the use of the older CloudWatch Logs agent.

Enable your Amazon EC2 instances running Windows Server 2016 to send logs to CloudWatch Logs using the older CloudWatch Logs agent

There are multiple methods you can use to enable instances running Windows Server 2016 to send logs to CloudWatch Logs. The steps in this section use Systems Manager Run Command. For information about the other possible methods, see [Sending Logs, Events, and Performance Counters to Amazon CloudWatch](#).

Steps

- [Download the sample configuration file](#)
- [Configure the JSON file for CloudWatch](#)
- [Create an IAM role for Systems Manager](#)
- [Verify Systems Manager prerequisites](#)
- [Verify internet access](#)
- [Enable CloudWatch Logs using Systems Manager Run Command](#)

Download the sample configuration file

Download the following sample file to your computer: [AWS.EC2.Windows.CloudWatch.json](#).

Configure the JSON file for CloudWatch

You determine which logs to send to CloudWatch by specifying your choices in a configuration file. The process of creating this file and specifying your choices can take 30 minutes or more to

complete. After you have completed this task once, you can reuse the configuration file on all of your instances.

Steps

- [Step 1: Enable CloudWatch Logs](#)
- [Step 2: Configure settings for CloudWatch](#)
- [Step 3: Configure the data to send](#)
- [Step 4: Configure flow control](#)
- [Step 5: Save JSON content](#)

Step 1: Enable CloudWatch Logs

At the top of the JSON file, change "false" to "true" for `IsEnabled`:

```
"IsEnabled": true,
```

Step 2: Configure settings for CloudWatch

Specify credentials, Region, a log group name, and a log stream namespace. This enables the instance to send log data to CloudWatch Logs. To send the same log data to different locations, you can add additional sections with unique IDs (for example, "CloudWatchLogs2" and CloudWatchLogs3") and a different Region for each ID.

To configure settings to send log data to CloudWatch Logs

1. In the JSON file, locate the `CloudWatchLogs` section.

```
{
  "Id": "CloudWatchLogs",
  "FullName":
"AWS.EC2.Windows.CloudWatch.CloudWatchLogsOutput,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "AccessKey": "",
    "SecretKey": "",
    "Region": "us-east-1",
    "LogGroup": "Default-Log-Group",
    "LogStream": "{instance_id}"
  }
},
```

2. Leave the `AccessKey` and `SecretKey` field blank. You configure credentials using an IAM role.
3. For `Region`, type the Region to which to send log data (for example, `us-east-2`).
4. For `LogGroup`, type the name for your log group. This name appears on the **Log Groups** screen in the CloudWatch console.
5. For `LogStream`, type the destination log stream. This name appears on the **Log Groups > Streams** screen in the CloudWatch console.

If you use `{instance_id}`, the default, the log stream name is the instance ID of this instance.

If you specify a log stream name that doesn't already exist, CloudWatch Logs automatically creates it for you. You can define a log stream name using a literal string, the predefined variables `{instance_id}`, `{hostname}`, and `{ip_address}`, or a combination of these.

Step 3: Configure the data to send

You can send event log data, Event Tracing for Windows (ETW) data, and other log data to CloudWatch Logs.

To send Windows application event log data to CloudWatch Logs

1. In the JSON file, locate the `ApplicationEventLog` section.

```
{
  "Id": "ApplicationEventLog",
  "FullName":
    "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Application",
    "Levels": "1"
  }
},
```

2. For `Levels`, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.
 - **2** - Upload only warning messages.

- **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send security log data to CloudWatch Logs

1. In the JSON file, locate the `SecurityEventLog` section.

```
{
  "Id": "SecurityEventLog",
  "FullName":
    "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Security",
    "Levels": "7"
  }
},
```

2. For `Levels`, type **7** to upload all messages.

To send system event log data to CloudWatch Logs

1. In the JSON file, locate the `SystemEventLog` section.

```
{
  "Id": "SystemEventLog",
  "FullName":
    "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "System",
    "Levels": "7"
  }
},
```

2. For `Levels`, specify the type of messages to upload. You can specify one of the following values:

- **1** - Upload only error messages.

- **2** - Upload only warning messages.
- **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send other types of event log data to CloudWatch Logs

1. In the JSON file, add a new section. Each section must have a unique Id.

```
{
  "Id": "Id-name",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Log-name",
    "Levels": "7"
  }
},
```

2. For Id, type a name for the log to upload (for example, **WindowsBackup**).
3. For LogName, type the name of the log to upload. You can find the name of the log as follows.
 - a. Open Event Viewer.
 - b. In the navigation pane, choose **Applications and Services Logs**.
 - c. Navigate to the log, and then choose **Actions, Properties**.
4. For Levels, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.
 - **2** - Upload only warning messages.
 - **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send Event Tracing for Windows data to CloudWatch Logs

ETW (Event Tracing for Windows) provides an efficient and detailed logging mechanism that applications can write logs to. Each ETW is controlled by a session manager that can start and stop the logging session. Each session has a provider and one or more consumers.

1. In the JSON file, locate the ETW section.

```
{
  "Id": "ETW",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Microsoft-Windows-WinINet/Analytic",
    "Levels": "7"
  }
},
```

2. For LogName, type the name of the log to upload.
3. For Levels, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.
 - **2** - Upload only warning messages.
 - **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send custom logs (any text-based log file) to CloudWatch Logs

1. In the JSON file, locate the CustomLogs section.

```
{
  "Id": "CustomLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
```

```
    "LogDirectoryPath": "C:\\CustomLogs\\",  
    "TimestampFormat": "MM/dd/yyyy HH:mm:ss",  
    "Encoding": "UTF-8",  
    "Filter": "",  
    "CultureName": "en-US",  
    "TimeZoneKind": "Local",  
    "LineCount": "5"  
  },  
},
```

2. For `LogDirectoryPath`, type the path where logs are stored on your instance.
3. For `TimestampFormat`, type the time stamp format to use. For more information about supported values, see the [Custom Date and Time Format Strings](#) topic on MSDN.

 Important

Your source log file must have the time stamp at the beginning of each log line and there must be a space following the time stamp.

4. For `Encoding`, type the file encoding to use (for example, UTF-8). For a list of supported values, see the [Encoding Class](#) topic on MSDN.

 Note

Use the encoding name, not the display name.

5. (Optional) For `Filter`, type the prefix of log names. Leave this parameter blank to monitor all files. For more information about supported values, see the [FileSystemWatcherFilter Property](#) topic on MSDN.
6. (Optional) For `CultureName`, type the locale where the time stamp is logged. If `CultureName` is blank, it defaults to the same locale currently used by your Windows instance. For more information about, see the `Language` tag column in the table in the [Product Behavior](#) topic on MSDN.

 Note

The `div`, `div-MV`, `hu`, and `hu-HU` values are not supported.

7. (Optional) For `TimeZoneKind`, type `Local` or `UTC`. You can set this to provide time zone information when no time zone information is included in your log's time stamp. If this parameter is left blank and if your time stamp doesn't include time zone information, CloudWatch Logs defaults to the local time zone. This parameter is ignored if your time stamp already contains time zone information.
8. (Optional) For `LineCount`, type the number of lines in the header to identify the log file. For example, IIS log files have virtually identical headers. You could enter `5`, which would read the first three lines of the log file header to identify it. In IIS log files, the third line is the date and time stamp, but the time stamp is not always guaranteed to be different between log files. For this reason, we recommend including at least one line of actual log data to uniquely fingerprint the log file.

To send IIS log data to CloudWatch Logs

1. In the JSON file, locate the `IISLog` section.

```
{
  "Id": "IISLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogDirectoryPath": "C:\\inetpub\\logs\\LogFiles\\W3SVC1",
    "TimestampFormat": "yyyy-MM-dd HH:mm:ss",
    "Encoding": "UTF-8",
    "Filter": "",
    "CultureName": "en-US",
    "TimeZoneKind": "UTC",
    "LineCount": "5"
  }
},
```

2. For `LogDirectoryPath`, type the folder where IIS logs are stored for an individual site (for example, `C:\\inetpub\\logs\\LogFiles\\W3SVC1`).

Note

Only W3C log format is supported. IIS, NCSA, and Custom formats are not supported.

3. For `TimestampFormat`, type the time stamp format to use. For more information about supported values, see the [Custom Date and Time Format Strings](#) topic on MSDN.
4. For `Encoding`, type the file encoding to use (for example, UTF-8). For more information about supported values, see the [Encoding Class](#) topic on MSDN.

Note

Use the encoding name, not the display name.

5. (Optional) For `Filter`, type the prefix of log names. Leave this parameter blank to monitor all files. For more information about supported values, see the [FileSystemWatcherFilter Property](#) topic on MSDN.
6. (Optional) For `CultureName`, type the locale where the time stamp is logged. If `CultureName` is blank, it defaults to the same locale currently used by your Windows instance. For more information about supported values, see the `Language` tag column in the table in the [Product Behavior](#) topic on MSDN.

Note

The `div`, `div-MV`, `hu`, and `hu-HU` values are not supported.

7. (Optional) For `TimeZoneKind`, enter `Local` or `UTC`. You can set this to provide time zone information when no time zone information is included in your log's time stamp. If this parameter is left blank and if your time stamp doesn't include time zone information, CloudWatch Logs defaults to the local time zone. This parameter is ignored if your time stamp already contains time zone information.
8. (Optional) For `LineCount`, type the number of lines in the header to identify the log file. For example, IIS log files have virtually identical headers. You could enter `5`, which would read the first five lines of the log file's header to identify it. In IIS log files, the third line is the date and time stamp, but the time stamp is not always guaranteed to be different between log files. For this reason, we recommend including at least one line of actual log data for uniquely fingerprinting the log file.

Step 4: Configure flow control

Each data type must have a corresponding destination in the Flows section. For example, to send the custom log, ETW log, and system log to CloudWatch Logs, add (CustomLogs, ETW, SystemEventLog), CloudWatchLogs to the Flows section.

Warning

Adding a step that is not valid blocks the flow. For example, if you add a disk metric step, but your instance doesn't have a disk, all steps in the flow are blocked.

You can send the same log file to more than one destination. For example, to send the application log to two different destinations that you defined in the CloudWatchLogs section, add ApplicationEventLog, (CloudWatchLogs, CloudWatchLogs2) to the Flows section.

To configure flow control

1. In the AWS.EC2.Windows.CloudWatch.json file, locate the Flows section.

```
"Flows": {
  "Flows": [
    "PerformanceCounter,CloudWatch",
    "(PerformanceCounter,PerformanceCounter2), CloudWatch2",
    "(CustomLogs, ETW, SystemEventLog),CloudWatchLogs",
    "CustomLogs, CloudWatchLogs2",
    "ApplicationEventLog,(CloudWatchLogs, CloudWatchLogs2)"
  ]
}
```

2. For Flows, add each data type that is to be uploaded (for example, ApplicationEventLog) and its destination (for example, CloudWatchLogs).

Step 5: Save JSON content

You are now finished editing the JSON file. Save it, and paste the file contents into a text editor in another window. You will need the file contents in a later step of this procedure.

Create an IAM role for Systems Manager

An IAM role for instance credentials is required when you use Systems Manager Run Command. This role enables Systems Manager to perform actions on the instance. For more information, see [Configuring Security Roles for Systems Manager](#) in the *AWS Systems Manager User Guide*. For information about how to attach an IAM role to an existing instance, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide*.

Verify Systems Manager prerequisites

Before you use Systems Manager Run Command to configure integration with CloudWatch Logs, verify that your instances meet the minimum requirements. For more information, see [Systems Manager Prerequisites](#) in the *AWS Systems Manager User Guide*.

Verify internet access

Your Amazon EC2 Windows Server instances and managed instances must have outbound internet access in order to send log and event data to CloudWatch. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

Enable CloudWatch Logs using Systems Manager Run Command

Run Command enables you to manage the configuration of your instances on demand. You specify a Systems Manager document, specify parameters, and execute the command on one or more instances. The SSM agent on the instance processes the command and configures the instance as specified.

To configure integration with CloudWatch Logs using Run Command

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Open the SSM console at <https://console.aws.amazon.com/systems-manager/>.
3. In the navigation pane, choose **Run Command**.
4. Choose **Run a command**.
5. For **Command document**, choose **AWS-ConfigureCloudWatch**.
6. For **Target instances**, choose the instances to integrate with CloudWatch Logs. If you do not see an instance in this list, it might not be configured for Run Command. For more information, see [Systems Manager Prerequisites](#) in the *Amazon EC2 User Guide*.
7. For **Status**, choose **Enabled**.
8. For **Properties**, copy and paste the JSON content you created in the previous tasks.

9. Complete the remaining optional fields and choose **Run**.

Use the following procedure to view the results of command execution in the Amazon EC2 console.

To view command output in the console

1. Select a command.
2. Choose the **Output** tab.
3. Choose **View Output**. The command output page shows the results of your command execution.

Quick Start: Enable your Amazon EC2 instances running Windows Server 2012 and Windows Server 2008 to send logs to CloudWatch Logs

Tip

CloudWatch includes a new unified agent that can collect both logs and metrics from EC2 instances and on-premises servers. We recommend that you use the newer unified CloudWatch agent. For more information, see [Getting started with CloudWatch Logs](#). The rest of this section explains the use of the older CloudWatch Logs agent.

Enable your Amazon EC2 instances running Windows Server 2012 and Windows Server 2008 to send logs to CloudWatch Logs

Use the following steps to enable your instances running Windows Server 2012 and Windows Server 2008 to send logs to CloudWatch Logs.

Download the sample configuration file

Download the following sample JSON file to your computer:
[AWS.EC2.Windows.CloudWatch.json](#). You edit it in the following steps.

Configure the JSON file for CloudWatch

You determine which logs to send to CloudWatch by specifying your choices in the JSON configuration file. The process of creating this file and specifying your choices can take 30 minutes

or more to complete. After you have completed this task once, you can reuse the configuration file on all of your instances.

Steps

- [Step 1: Enable CloudWatch Logs](#)
- [Step 2: Configure settings for CloudWatch](#)
- [Step 3: Configure the data to send](#)
- [Step 4: Configure flow control](#)

Step 1: Enable CloudWatch Logs

At the top of the JSON file, change "false" to "true" for `IsEnabled`:

```
"IsEnabled": true,
```

Step 2: Configure settings for CloudWatch

Specify credentials, Region, a log group name, and a log stream namespace. This enables the instance to send log data to CloudWatch Logs. To send the same log data to different locations, you can add additional sections with unique IDs (for example, "CloudWatchLogs2" and CloudWatchLogs3") and a different Region for each ID.

To configure settings to send log data to CloudWatch Logs

1. In the JSON file, locate the `CloudWatchLogs` section.

```
{
  "Id": "CloudWatchLogs",
  "FullName":
"AWS.EC2.Windows.CloudWatch.CloudWatchLogsOutput,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "AccessKey": "",
    "SecretKey": "",
    "Region": "us-east-1",
    "LogGroup": "Default-Log-Group",
    "LogStream": "{instance_id}"
  }
},
```

2. Leave the `AccessKey` and `SecretKey` field blank. You configure credentials using an IAM role.
3. For `Region`, type the Region to which to send log data (for example, `us-east-2`).
4. For `LogGroup`, type the name for your log group. This name appears on the **Log Groups** screen in the CloudWatch console.
5. For `LogStream`, type the destination log stream. This name appears on the **Log Groups > Streams** screen in the CloudWatch console.

If you use `{instance_id}`, the default, the log stream name is the instance ID of this instance.

If you specify a log stream name that doesn't already exist, CloudWatch Logs automatically creates it for you. You can define a log stream name using a literal string, the predefined variables `{instance_id}`, `{hostname}`, and `{ip_address}`, or a combination of these.

Step 3: Configure the data to send

You can send event log data, Event Tracing for Windows (ETW) data, and other log data to CloudWatch Logs.

To send Windows application event log data to CloudWatch Logs

1. In the JSON file, locate the `ApplicationEventLog` section.

```
{
  "Id": "ApplicationEventLog",
  "FullName":
    "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Application",
    "Levels": "1"
  }
},
```

2. For `Levels`, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.
 - **2** - Upload only warning messages.

- **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send security log data to CloudWatch Logs

1. In the JSON file, locate the `SecurityEventLog` section.

```
{
  "Id": "SecurityEventLog",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Security",
    "Levels": "7"
  }
},
```

2. For `Levels`, type **7** to upload all messages.

To send system event log data to CloudWatch Logs

1. In the JSON file, locate the `SystemEventLog` section.

```
{
  "Id": "SystemEventLog",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "System",
    "Levels": "7"
  }
},
```

2. For `Levels`, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.

- **2** - Upload only warning messages.
- **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send other types of event log data to CloudWatch Logs

1. In the JSON file, add a new section. Each section must have a unique Id.

```
{
  "Id": "Id-name",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Log-name",
    "Levels": "7"
  }
},
```

2. For Id, type a name for the log to upload (for example, **WindowsBackup**).
3. For LogName, type the name of the log to upload. You can find the name of the log as follows.
 - a. Open Event Viewer.
 - b. In the navigation pane, choose **Applications and Services Logs**.
 - c. Navigate to the log, and then choose **Actions, Properties**.
4. For Levels, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.
 - **2** - Upload only warning messages.
 - **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send Event Tracing for Windows data to CloudWatch Logs

ETW (Event Tracing for Windows) provides an efficient and detailed logging mechanism that applications can write logs to. Each ETW is controlled by a session manager that can start and stop the logging session. Each session has a provider and one or more consumers.

1. In the JSON file, locate the ETW section.

```
{
  "Id": "ETW",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Microsoft-Windows-WinINet/Analytic",
    "Levels": "7"
  }
},
```

2. For LogName, type the name of the log to upload.
3. For Levels, specify the type of messages to upload. You can specify one of the following values:
 - **1** - Upload only error messages.
 - **2** - Upload only warning messages.
 - **4** - Upload only information messages.

You can combine values to include more than one type of message. For example, a value of **3** uploads error messages (**1**) and warning messages (**2**). A value of **7** uploads error messages (**1**), warning messages (**2**), and information messages (**4**).

To send custom logs (any text-based log file) to CloudWatch Logs

1. In the JSON file, locate the CustomLogs section.

```
{
  "Id": "CustomLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
```

```
    "LogDirectoryPath": "C:\\CustomLogs\\",
    "TimestampFormat": "MM/dd/yyyy HH:mm:ss",
    "Encoding": "UTF-8",
    "Filter": "",
    "CultureName": "en-US",
    "TimeZoneKind": "Local",
    "LineCount": "5"
  },
},
```

2. For `LogDirectoryPath`, type the path where logs are stored on your instance.
3. For `TimestampFormat`, type the time stamp format to use. For more information about supported values, see the [Custom Date and Time Format Strings](#) topic on MSDN.

 Important

Your source log file must have the time stamp at the beginning of each log line and there must be a space following the time stamp.

4. For `Encoding`, type the file encoding to use (for example, UTF-8). For more information about supported values, see the [Encoding Class](#) topic on MSDN.

 Note

Use the encoding name, not the display name.

5. (Optional) For `Filter`, type the prefix of log names. Leave this parameter blank to monitor all files. For more information about supported values, see the [FileSystemWatcherFilter Property](#) topic on MSDN.
6. (Optional) For `CultureName`, type the locale where the time stamp is logged. If `CultureName` is blank, it defaults to the same locale currently used by your Windows instance. For more information about supported values, see the `Language` tag column in the table in the [Product Behavior](#) topic on MSDN.

 Note

The `div`, `div-MV`, `hu`, and `hu-HU` values are not supported.

7. (Optional) For `TimeZoneKind`, type `Local` or `UTC`. You can set this to provide time zone information when no time zone information is included in your log's time stamp. If this parameter is left blank and if your time stamp doesn't include time zone information, CloudWatch Logs defaults to the local time zone. This parameter is ignored if your time stamp already contains time zone information.
8. (Optional) For `LineCount`, type the number of lines in the header to identify the log file. For example, IIS log files have virtually identical headers. You could enter `5`, which would read the first three lines of the log file header to identify it. In IIS log files, the third line is the date and time stamp, but the time stamp is not always guaranteed to be different between log files. For this reason, we recommend including at least one line of actual log data to uniquely fingerprint the log file.

To send IIS log data to CloudWatch Logs

1. In the JSON file, locate the `IISLog` section.

```
{
  "Id": "IISLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogDirectoryPath": "C:\\inetpub\\logs\\LogFiles\\W3SVC1",
    "TimestampFormat": "yyyy-MM-dd HH:mm:ss",
    "Encoding": "UTF-8",
    "Filter": "",
    "CultureName": "en-US",
    "TimeZoneKind": "UTC",
    "LineCount": "5"
  }
},
```

2. For `LogDirectoryPath`, type the folder where IIS logs are stored for an individual site (for example, `C:\\inetpub\\logs\\LogFiles\\W3SVCn`).

Note

Only W3C log format is supported. IIS, NCSA, and Custom formats are not supported.

3. For `TimestampFormat`, type the time stamp format to use. For more information about supported values, see the [Custom Date and Time Format Strings](#) topic on MSDN.
4. For `Encoding`, type the file encoding to use (for example, UTF-8). For more information about supported values, see the [Encoding Class](#) topic on MSDN.

Note

Use the encoding name, not the display name.

5. (Optional) For `Filter`, type the prefix of log names. Leave this parameter blank to monitor all files. For more information about supported values, see the [FileSystemWatcherFilter Property](#) topic on MSDN.
6. (Optional) For `CultureName`, type the locale where the time stamp is logged. If `CultureName` is blank, it defaults to the same locale currently used by your Windows instance. For more information about supported values, see the `Language` tag column in the table in the [Product Behavior](#) topic on MSDN.

Note

The `div`, `div-MV`, `hu`, and `hu-HU` values are not supported.

7. (Optional) For `TimeZoneKind`, enter `Local` or `UTC`. You can set this to provide time zone information when no time zone information is included in your log's time stamp. If this parameter is left blank and if your time stamp doesn't include time zone information, CloudWatch Logs defaults to the local time zone. This parameter is ignored if your time stamp already contains time zone information.
8. (Optional) For `LineCount`, type the number of lines in the header to identify the log file. For example, IIS log files have virtually identical headers. You could enter `5`, which would read the first five lines of the log file's header to identify it. In IIS log files, the third line is the date and time stamp, but the time stamp is not always guaranteed to be different between log files. For this reason, we recommend including at least one line of actual log data for uniquely fingerprinting the log file.

Step 4: Configure flow control

Each data type must have a corresponding destination in the Flows section. For example, to send the custom log, ETW log, and system log to CloudWatch Logs, add (CustomLogs, ETW, SystemEventLog), CloudWatchLogs to the Flows section.

Warning

Adding a step that is not valid blocks the flow. For example, if you add a disk metric step, but your instance doesn't have a disk, all steps in the flow are blocked.

You can send the same log file to more than one destination. For example, to send the application log to two different destinations that you defined in the CloudWatchLogs section, add ApplicationEventLog, (CloudWatchLogs, CloudWatchLogs2) to the Flows section.

To configure flow control

1. In the AWS.EC2.Windows.CloudWatch.json file, locate the Flows section.

```
"Flows": {
  "Flows": [
    "PerformanceCounter,CloudWatch",
    "(PerformanceCounter,PerformanceCounter2), CloudWatch2",
    "(CustomLogs, ETW, SystemEventLog),CloudWatchLogs",
    "CustomLogs, CloudWatchLogs2",
    "ApplicationEventLog,(CloudWatchLogs, CloudWatchLogs2)"
  ]
}
```

2. For Flows, add each data type that is to be uploaded (for example, ApplicationEventLog) and its destination (for example, CloudWatchLogs).

You are now finished editing the JSON file. You use it in a later step.

Start the agent

To enable an Amazon EC2 instance running Windows Server 2012 or Windows Server 2008 to send logs to CloudWatch Logs, use the EC2Config service (EC2Config.exe). Your instance should have EC2Config 4.0 or later, and you can use this procedure.

To configure CloudWatch using EC2Config 4.x

1. Check the encoding of the `AWS.EC2.Windows.CloudWatch.json` file that you edited earlier in this procedure. Only UTF-8 without BOM encoding is supported. Then save the file in the following folder on your Windows Server 2008 - 2012 R2 instance: `C:\Program Files\Amazon\SSM\Plugins\awsCloudWatch\`.
2. Start or restart the SSM agent (`AmazonSSMAgent.exe`) using the Windows Services control panel or using the following PowerShell command:

```
PS C:\> Restart-Service AmazonSSMAgent
```

After the SSM agent restarts, it detects the configuration file and configures the instance for CloudWatch integration. If you change parameters and settings in the local configuration file, you need to restart the SSM agent to pick up the changes. To disable CloudWatch integration on the instance, change `IsEnabled` to `false` and save your changes in the configuration file.

Report the CloudWatch Logs agent status

Use the following procedure to report the status of the CloudWatch Logs agent on your EC2 instance.

To report the agent status

1. Connect to your EC2 instance. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

For more information about connection issues, see [Troubleshooting Connecting to Your Instance](#) in the *Amazon EC2 User Guide*

2. At a command prompt, type the following command:

```
sudo service awslogs status
```

If you are running Amazon Linux 2, type the following command:

```
sudo service awslogsd status
```

3. Check the `/var/log/awslogs.log` file for any errors, warnings, or issues with the CloudWatch Logs agent.

Start the CloudWatch Logs agent

If the CloudWatch Logs agent on your EC2 instance did not start automatically after installation, or if you stopped the agent, you can use the following procedure to start the agent.

To start the agent

1. Connect to your EC2 instance. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

For more information about connection issues, see [Troubleshooting Connecting to Your Instance](#) in the *Amazon EC2 User Guide*.

2. At a command prompt, type the following command:

```
sudo service awslogs start
```

If you are running Amazon Linux 2, type the following command:

```
sudo service awslogsd start
```

Stop the CloudWatch Logs agent

Use the following procedure to stop the CloudWatch Logs agent on your EC2 instance.

To stop the agent

1. Connect to your EC2 instance. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

For more information about connection issues, see [Troubleshooting Connecting to Your Instance](#) in the *Amazon EC2 User Guide*.

2. At a command prompt, type the following command:

```
sudo service awslogs stop
```

If you are running Amazon Linux 2, type the following command:

```
sudo service awslogsd stop
```

CloudWatch Logs agent reference

Important

This section is a reference for those using the deprecated CloudWatch Logs agent. If you're using Instance Metadata Service Version 2 (IMDSv2), you must use the new unified CloudWatch agent. However, even if you're not using IMDSv2, we strongly recommend using the newer unified CloudWatch agent instead of the deprecated CloudWatch Logs agent. For information about the newer unified CloudWatch agent, see [Collecting metrics and logs from Amazon EC2 instance and on-premises servers with the CloudWatch agent](#). For information about migrating from the deprecated CloudWatch Logs agent to the unified agent, [Create the CloudWatch agent configuration file with the wizard](#).

The CloudWatch Logs agent provides an automated way to send log data to CloudWatch Logs from Amazon EC2 instances. The agent includes the following components:

- A plug-in to the AWS CLI that pushes log data to CloudWatch Logs.
- A script (daemon) that initiates the process to push data to CloudWatch Logs.
- A cron job that ensures that the daemon is always running.

Agent configuration file

The CloudWatch Logs agent configuration file describes information needed by the CloudWatch Logs agent. The agent configuration file's [general] section defines common configurations that apply to all log streams. The [logstream] section defines the information necessary to send a local file to a remote log stream. You can have more than one [logstream] section, but each must have a unique name within the configuration file, e.g., [logstream1], [logstream2], and so on. The [logstream] value along with the first line of data in the log file, define the log file's identity.

```
[general]
state_file = value
logging_config_file = value
```

```

use_gzip_http_content_encoding = [true | false]

[logstream1]
log_group_name = value
log_stream_name = value
datetime_format = value
time_zone = [LOCAL|UTC]
file = value
file_fingerprint_lines = integer | integer-integer
multi_line_start_pattern = regex | {datetime_format}
initial_position = [start_of_file | end_of_file]
encoding = [ascii|utf_8|..]
buffer_duration = integer
batch_count = integer
batch_size = integer

[logstream2]
...

```

state_file

Specifies where the state file is stored.

logging_config_file

(Optional) Specifies the location of the agent logging config file. If you do not specify an agent logging config file here, the default file `awslogs.conf` is used. The default file location is `/var/awslogs/etc/awslogs.conf` if you installed the agent with a script, and is `/etc/awslogs/awslogs.conf` if you installed the agent with rpm. The file is in Python configuration file format (<https://docs.python.org/2/library/logging.config.html#logging-config-fileformat>). Loggers with the following names can be customized.

```

cwlogs.push
cwlogs.push.reader
cwlogs.push.publisher
cwlogs.push.event
cwlogs.push.batch
cwlogs.push.stream
cwlogs.push.watcher

```

The sample below changes the level of reader and publisher to WARNING while the default value is INFO.

```
[loggers]
keys=root,cwlogs,reader,publisher

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=INFO
handlers=consoleHandler

[logger_cwlogs]
level=INFO
handlers=consoleHandler
qualname=cwlogs.push
propagate=0

[logger_reader]
level=WARNING
handlers=consoleHandler
qualname=cwlogs.push.reader
propagate=0

[logger_publisher]
level=WARNING
handlers=consoleHandler
qualname=cwlogs.push.publisher
propagate=0

[handler_consoleHandler]
class=logging.StreamHandler
level=INFO
formatter=simpleFormatter
args=(sys.stderr,)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(process)d - %(threadName)s -
%(message)s
```

use_gzip_http_content_encoding

When set to true (default), enables gzip http content encoding to send compressed payloads to CloudWatch Logs. This decreases CPU usage, lowers NetworkOut, and decreases put latency. To disable this feature, add **use_gzip_http_content_encoding = false** to the **[general]** section of the CloudWatch Logs agent configuration file, and then restart the agent.

Note

This setting is only available in awscli-cwlogs version 1.3.3 and later.

log_group_name

Specifies the destination log group. A log group is created automatically if it doesn't already exist. Log group names can be between 1 and 512 characters long. Allowed characters include a-z, A-Z, 0-9, '_' (underscore), '-' (hyphen), '/' (forward slash), and '.' (period).

log_stream_name

Specifies the destination log stream. You can use a literal string or predefined variables ({instance_id}, {hostname}, {ip_address}), or combination of both to define a log stream name. A log stream is created automatically if it doesn't already exist.

datetime_format

Specifies how the timestamp is extracted from logs. The timestamp is used for retrieving log events and generating metrics. The current time is used for each log event if the **datetime_format** isn't provided. If the provided **datetime_format** value is invalid for a given log message, the timestamp from the last log event with a successfully parsed timestamp is used. If no previous log events exist, the current time is used.

The common datetime_format codes are listed below. You can also use any datetime_format codes supported by Python, datetime.strptime(). The timezone offset (%z) is also supported even though it's not supported until python 3.2, [+ -]HHMM without colon(:). For more information, see [strftime\(\) and strptime\(\) Behavior](#).

%y: Year without century as a zero-padded decimal number. 00, 01, ..., 99

%Y: Year with century as a decimal number. 1970, 1988, 2001, 2013

%b: Month as locale's abbreviated name. Jan, Feb, ..., Dec (en_US);

%B: Month as locale's full name. January, February, ..., December (en_US);

%m: Month as a zero-padded decimal number. 01, 02, ..., 12

%d: Day of the month as a zero-padded decimal number. 01, 02, ..., 31

%H: Hour (24-hour clock) as a zero-padded decimal number. 00, 01, ..., 23

%I: Hour (12-hour clock) as a zero-padded decimal number. 01, 02, ..., 12

%p: Locale's equivalent of either AM or PM.

%M: Minute as a zero-padded decimal number. 00, 01, ..., 59

%S: Second as a zero-padded decimal number. 00, 01, ..., 59

%f: Microsecond as a decimal number, zero-padded on the left. 000000, ..., 999999

%z: UTC offset in the form +HHMM or -HHMM. +0000, -0400, +1030

Example formats:

Syslog: '%b %d %H:%M:%S', e.g. Jan 23 20:59:29

Log4j: '%d %b %Y %H:%M:%S', e.g. 24 Jan 2014 05:00:00

ISO8601: '%Y-%m-%dT%H:%M:%S%z', e.g. 2014-02-20T05:20:20+0000

time_zone

Specifies the time zone of log event timestamp. The two supported values are UTC and LOCAL. The default is LOCAL, which is used if time zone can't be inferred based on **datetime_format**.

file

Specifies log files that you want to push to CloudWatch Logs. File can point to a specific file or multiple files (using wildcards such as /var/log/system.log*). Only the latest file is pushed to CloudWatch Logs based on file modification time. We recommend that you use wildcards to specify a series of files of the same type, such as access_log.2014-06-01-01, access_log.2014-06-01-02, and so on, but not multiple kinds of files, such as access_log_80 and access_log_443. To specify multiple kinds of files, add another log stream entry to the configuration file so each kind of log file goes to a different log stream. Zipped files are not supported.

file_fingerprint_lines

Specifies the range of lines for identifying a file. The valid values are one number or two dash delimited numbers, such as '1', '2-5'. The default value is '1' so the first line is used to calculate fingerprint. Fingerprint lines are not sent to CloudWatch Logs unless all the specified lines are available.

multi_line_start_pattern

Specifies the pattern for identifying the start of a log message. A log message is made of a line that matches the pattern and any following lines that don't match the pattern. The valid values are regular expression or {datetime_format}. When using {datetime_format}, the datetime_format option should be specified. The default value is '^([^\s])' so any line that begins with non-whitespace character closes the previous log message and starts a new log message.

initial_position

Specifies where to start to read data (start_of_file or end_of_file). The default is start_of_file. It's only used if there is no state persisted for that log stream.

encoding

Specifies the encoding of the log file so that the file can be read correctly. The default is utf_8. Encodings supported by Python codecs.decode() can be used here.

Warning

Specifying an incorrect encoding might cause data loss because characters that cannot be decoded are replaced with some other character.

Below are some common encodings:

ascii, big5, big5hkscs, cp037, cp424, cp437, cp500, cp720, cp737, cp775, cp850, cp852, cp855, cp856, cp857, cp858, cp860, cp861, cp862, cp863, cp864, cp865, cp866, cp869, cp874, cp875, cp932, cp949, cp950, cp1006, cp1026, cp1140, cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258, euc_jp, euc_jis_2004, euc_jisx0213, euc_kr, gb2312, gbk, gb18030, hz, iso2022_jp, iso2022_jp_1, iso2022_jp_2, iso2022_jp_2004, iso2022_jp_3, iso2022_jp_ext, iso2022_kr, latin_1, iso8859_2, iso8859_3, iso8859_4, iso8859_5, iso8859_6, iso8859_7,

iso8859_8, iso8859_9, iso8859_10, iso8859_13, iso8859_14, iso8859_15, iso8859_16, johab, koi8_r, koi8_u, mac_cyrillic, mac_greek, mac_iceland, mac_latin2, mac_roman, mac_turkish, ptcp154, shift_jis, shift_jis_2004, shift_jisx0213, utf_32, utf_32_be, utf_32_le, utf_16, utf_16_be, utf_16_le, utf_7, utf_8, utf_8_sig

buffer_duration

Specifies the time duration for the batching of log events. The minimum value is 5000ms and default value is 5000ms.

batch_count

Specifies the max number of log events in a batch, up to 10000. The default value is 10000.

batch_size

Specifies the max size of log events in a batch, in bytes, up to 1048576 bytes. The default value is 1048576 bytes. This size is calculated as the sum of all event messages in UTF-8, plus 26 bytes for each log event.

Using the CloudWatch Logs agent with HTTP proxies

You can use the CloudWatch Logs agent with HTTP proxies.

Note

HTTP proxies are supported in awslogs-agent-setup.py version 1.3.8 or later.

To use the CloudWatch Logs agent with HTTP proxies

1. Do one of the following:
 - a. For a new installation of the CloudWatch Logs agent, run the following commands:

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
```

```
sudo python awslogs-agent-setup.py --region us-east-1 --http-proxy http://your/proxy --https-proxy http://your/proxy --no-proxy 169.254.169.254
```

In order to maintain access to the Amazon EC2 metadata service on EC2 instances, use **--no-proxy 169.254.169.254** (recommended). For more information, see [Instance Metadata and User Data](#) in the *Amazon EC2 User Guide*.

In the values for `http-proxy` and `https-proxy`, you specify the entire URL.

- b. For an existing installation of the CloudWatch Logs agent, edit `/var/awslogs/etc/proxy.conf`, and add your proxies:

```
HTTP_PROXY=  
HTTPS_PROXY=  
NO_PROXY=
```

2. Restart the agent for the changes to take effect:

```
sudo service awslogs restart
```

If you are using Amazon Linux 2, use the following command to restart the agent:

```
sudo service awslogsd restart
```

Compartmentalizing CloudWatch Logs agent configuration files

If you're using `awslogs-agent-setup.py` version 1.3.8 or later with `awscli-cwlogs` 1.3.3 or later, you can import different stream configurations for various components independently of one another by creating additional configuration files in the `/var/awslogs/etc/config/` directory. When the CloudWatch Logs agent starts, it includes any stream configurations in these additional configuration files. Configuration properties in the `[general]` section must be defined in the main configuration file (`/var/awslogs/etc/awslogs.conf`) and are ignored in any additional configuration files found in `/var/awslogs/etc/config/`.

If you don't have a `/var/awslogs/etc/config/` directory because you installed the agent with `rpm`, you can use the `/etc/awslogs/config/` directory instead.

Restart the agent for the changes to take effect:

```
sudo service awslogs restart
```

If you are using Amazon Linux 2, use the following command to restart the agent:

```
sudo service awslogsd restart
```

CloudWatch Logs agent FAQ

What kinds of file rotations are supported?

The following file rotation mechanisms are supported:

- Renaming existing log files with a numerical suffix, then re-creating the original empty log file. For example, `/var/log/syslog.log` is renamed `/var/log/syslog.log.1`. If `/var/log/syslog.log.1` already exists from a previous rotation, it is renamed `/var/log/syslog.log.2`.
- Truncating the original log file in place after creating a copy. For example, `/var/log/syslog.log` is copied to `/var/log/syslog.log.1` and `/var/log/syslog.log` is truncated. There might be data loss for this case, so be careful about using this file rotation mechanism.
- Creating a new file with a common pattern as the old one. For example, `/var/log/syslog.log.2014-01-01` remains and `/var/log/syslog.log.2014-01-02` is created.

The fingerprint (source ID) of the file is calculated by hashing the log stream key and the first line of file content. To override this behavior, the **file_fingerprint_lines** option can be used. When file rotation happens, the new file is supposed to have new content and the old file is not supposed to have content appended; the agent pushes the new file after it finishes reading the old file.

How can I determine which version of agent am I using?

If you used a setup script to install the CloudWatch Logs agent, you can use `/var/awslogs/bin/awslogs-version.sh` to check what version of the agent you are using. It prints out the version of the agent and its major dependencies. If you used yum to install the CloudWatch Logs agent, you can use `"yum info awslogs"` and `"yum info aws-cli-plugin-cloudwatch-logs"` to check the version of the CloudWatch Logs agent and plugin.

How are log entries converted to log events?

Log events contain two properties: the timestamp of when the event occurred, and the raw log message. By default, any line that begins with non-whitespace character closes the previous log message if there is one, and starts a new log message. To override this behavior, the **multi_line_start_pattern** can be used and any line that matches the pattern starts a new log message. The pattern could be any regex or `{datetime_format}`. For example, if the first line of every log message contains a timestamp like `'2014-01-02T13:13:01Z'`, then the **multi_line_start_pattern** can be set to `'\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}Z'`. To simplify the

configuration, the '{datetime_format}' variable can be used if the **datetime_format** option is specified. For the same example, if **datetime_format** is set to '%Y-%m-%dT%H:%M:%S%z', then **multi_line_start_pattern** could be simply '{datetime_format}'.

The current time is used for each log event if the **datetime_format** isn't provided. If the provided **datetime_format** is invalid for a given log message, the timestamp from the last log event with a successfully parsed timestamp is used. If no previous log events exist, the current time is used. A warning message is logged when a log event falls back to the current time or time of previous log event.

Timestamps are used for retrieving log events and generating metrics, so if you specify the wrong format, log events could become non-retrievable and generate wrong metrics.

How are log events batched?

A batch becomes full and is published when any of the following conditions are met:

1. The **buffer_duration** amount of time has passed since the first log event was added.
2. Less than **batch_size** of log events have been accumulated but adding the new log event exceeds the **batch_size**.
3. The number of log events has reached **batch_count**.
4. Log events from the batch don't span more than 24 hours, but adding the new log event exceeds the 24 hours constraint.

What would cause log entries, log events, or batches to be skipped or truncated?

To follow the constraint of the PutLogEvents operation, the following issues could cause a log event or batch to be skipped.

Note

The CloudWatch Logs agent writes a warning to its log when data is skipped.

1. If the size of a log event exceeds 256 KB, the log event is skipped completely.
2. If the timestamp of log event is more than 2 hours in future, the log event is skipped.
3. If the timestamp of log event is more than 14 days in past, the log event is skipped.
4. If any log event is older than the retention period of log group, the whole batch is skipped.
5. If the batch of log events in a single PutLogEvents request spans more than 24 hours, the PutLogEvents operation fails.

Does stopping the agent cause data loss/duplicates?

Not as long as the state file is available and no file rotation has happened since the last run. The CloudWatch Logs agent can start from where it stopped and continue pushing the log data.

Can I point different log files from the same or different hosts to the same log stream?

Configuring multiple log sources to send data to a single log stream is not supported.

What API calls does the agent make (or what actions should I add to my IAM policy)?

The CloudWatch Logs agent requires permission to perform `CreateLogGroup`, `CreateLogStream`, `DescribeLogStreams`, `DescribeLogGroups`, `PutLogEvents` and `PutRetentionPolicy` actions. If you're using the latest agent, `DescribeLogStreams` is not needed. See the sample IAM policy below.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    }
  ]
}
```

I don't want the CloudWatch Logs agent to create either log groups or log streams automatically. How can I prevent the agent from recreating both log groups and log streams?

In your IAM policy, you can restrict the agent to only the following operations: `DescribeLogStreams`, `PutLogEvents`.

Before you revoke the `CreateLogGroup` and `CreateLogStream` permissions from the agent, be sure to create both the log groups and log streams that you want the agent to use. The logs agent cannot create log streams in a log group that you have created unless it has both the `CreateLogGroup` and `CreateLogStream` permissions.

What logs should I look at when troubleshooting?

The agent installation log is at `/var/log/awslogs-agent-setup.log` and the agent log is at `/var/log/awslogs.log`.

Quick Start: Use CloudFormation to get started with CloudWatch Logs

AWS CloudFormation enables you to describe and provision your AWS resources in JSON format. The advantages of this method include being able to manage a collection of AWS resources as a single unit, and easily replicating your AWS resources across Regions.

When you provision AWS using CloudFormation, you create templates that describe the AWS resources to use. The following example is a template snippet that creates a log group and a metric filter that counts 404 occurrences and sends this count to the log group.

```
"WebServerLogGroup": {
  "Type": "AWS::Logs::LogGroup",
  "Properties": {
    "RetentionInDays": 7
  }
},

"404MetricFilter": {
  "Type": "AWS::Logs::MetricFilter",
  "Properties": {
    "LogGroupName": {
      "Ref": "WebServerLogGroup"
    },
    "FilterPattern": "[ip, identity, user_id, timestamp, request, status_code = 404, size, ...]",
    "MetricTransformations": [
      {
        "MetricValue": "1",
        "MetricNamespace": "test/404s",
        "MetricName": "test404Count"
```

```
}  
  ]  
}  
}
```

This is a basic example. You can set up much richer CloudWatch Logs deployments using CloudFormation. For more information about template examples, see [Amazon CloudWatch Logs Template Snippets](#) in the *AWS CloudFormation User Guide*. For more information about getting started, see [Getting Started with AWS CloudFormation](#) in the *AWS CloudFormation User Guide*.

Using CloudWatch Logs 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 CloudWatch Logs, see [Code examples for CloudWatch Logs 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.

Log management

CloudWatch Logs provides advanced log management capabilities that help you organize, transform, and analyze your log data more effectively. These features include [cross-account and cross-region data centralization](#), [automatic data discovery and schema management](#), [log transformation during ingestions](#), and [enhanced analytics with facets for interactive log exploration](#).

Topics

- [Data source discovery and management](#)
- [Features enabled by data sources](#)
- [Supported AWS services for data sources](#)
- [Supported third-party sources for data sources](#)

Data source discovery and management

CloudWatch Logs automatically discovers and categorizes your log data by data source and type, making it easier to understand and manage your logs at scale. This feature provides schema discovery for AWS vended sources such as Amazon VPC Flow Logs, CloudTrail, and Route 53, as well as third-party security tools.

The Logs Management console provides a high-level view of your logs organized by data source and type, rather than just log groups. This organization helps you:

- View logs categorized by AWS services, third-party sources (such as Okta or CrowdStrike), and custom sources
- Understand the schema and structure of your log data automatically
- Create field index policies based on discovered schema fields
- Manage logs more efficiently across different data sources
- Query logs by different data sources

When you [enable CloudWatch Logs logging for supported AWS services](#), CloudWatch Logs automatically applies the appropriate schema to your logs. This automatic schema application helps maintain consistency and provides immediate insights into your log structure.

What is CloudWatch Logs Data Sources?

CloudWatch Logs Data Sources is a feature that provides a new way to organize and categorize your logs data based on the source that generates the logs. While CloudWatch Logs traditionally uses log groups to organize logs, Data Sources offers an additional layer of organization that groups logs by their originating service and log type.

How Data Sources work

Data Sources provide service-based log organization and simplified discovery across your AWS infrastructure. You can easily locate logs from specific services and filter by log type without needing to know individual log group names or structures.

For third-party sources and optionally for application logs sources, Data Sources work with CloudWatch pipelines to categorize your logs. When you configure a pipeline to ingest and transform logs, you specify the data source name and type. CloudWatch Logs then automatically categorizes all logs that the pipeline processes. For more information, see [CloudWatch pipelines](#) in the *Amazon CloudWatch User Guide*.

Data Sources categorize logs using two key identifiers:

- **Data Source Name:** The AWS service, third-party source, or application that generates the logs (for example, Route 53, Amazon VPC, CloudTrail, Okta SSO, or CrowdStrike Falcon).
- **Data Source Type:** The specific type of log generated by that service.

A schema defines the structure of log data, including what fields are present and how information is organized. A single data source can produce multiple types of logs with different schemas and purposes. For example, the AWS CloudTrail data source has two types: management events (which track control plane operations like creating or deleting resources) and data events (which track data plane operations like S3 object access). Each type has a different schema because they capture different kinds of information.

How to get started

CloudWatch Logs categorizes your logs into data sources based on their origin. The method depends on the type of logs you're working with:

AWS service logs

Logs from [supported AWS services](#) are automatically grouped by data source without any configuration required. CloudWatch Logs recognizes these logs and applies the appropriate data source name and type based on the originating service.

Third-party logs

Third-party logs require pipelines for data source categorization. When you configure a pipeline to ingest logs from supported third-party sources such as Microsoft Office 365, Okta, CrowdStrike, or Palo Alto Networks, you specify the [data source name and type](#) in the pipeline configuration. CloudWatch Logs automatically categorizes all logs that the pipeline processes using those identifiers.

Pipelines can optionally transform third-party logs into Open Cybersecurity Schema Framework (OCSF) format for standardized security event analysis. When OCSF transformation is enabled, the data source name and type are automatically determined based on the OCSF schema mapping. Without OCSF transformation, you specify the data source name and type in the pipeline configuration.

Application logs

For custom application logs, you can categorize them by data source using one of these methods:

- **Log group tags** - Add tags to your log groups using the keys `cw:datasource:name` and `cw:datasource:type` to specify the data source name and type respectively for all logs ingested in the log group. Tag values can be up to 64 characters and may contain only lower case letters, numbers and underscore. They must start with either a letter or a number and they may not contain double underscores(`__`) .
- **Pipeline configuration** - Configure data source information through log processing pipelines when ingesting your application logs.

Note

Data source names cannot start with "aws" or "amazon" to avoid conflicts with AWS service logs.

System fields

CloudWatch Logs automatically adds three system fields to logs that are categorized by data source. These fields serve as default facets:

- `@data_source_name` - Contains the name of the data source, or "Unknown" if not determined
- `@data_source_type` - Contains the type of the data source, or "Unknown" if not determined
- `@data_format` - Indicates the format of the log data

When the data source name or type cannot be determined, these fields are set to "Unknown". Data sources with "Unknown" values are still visible in facets and in the data sources table under "Log Management" in Console, allowing you to identify uncategorized logs and from which log group they come from.

The `@data_format` field can contain one of the following values:

- `Default` - Logs ingested without modification.
- `Custom` - Logs processed through pipeline processors or logs ingested into log group with data source name/type tags.
- `OCSF-<version>` - Logs processed with OCSF (Open Cybersecurity Schema Framework) processors in pipelines.
- `AWS-OTEL-LOG-V<version>` - OpenTelemetry logs ingested through the CloudWatch OTLP endpoint.
- `AWS-OTEL-TRACE-V<version>` - OpenTelemetry traces ingested through the CloudWatch OTLP endpoint.

These system fields enable you to filter and query your logs based on their source and format, making it easier to work with logs from different origins and processing pipelines.

Accessing Data Sources

Console

In the CloudWatch Logs console, you use the **Log Management** tab to access your data sources. CloudWatch Logs automatically consolidates your log data by data sources and types, continuously discovering newly ingested data. From the data sources list, you can create pipelines, define field indexes and facets.

AWS CLI

Use the following command to list distinct data sources and types of logs in your account:

```
aws logs list-aggregate-log-group-summaries --group-by DATA_SOURCE_NAME_AND_TYPE
```

Relationship to log groups

Data sources complement rather than replace log groups. Your logs continue to be stored in log groups as before, but now they're also automatically tagged with data source information. This dual organization allows you to:

- Use log groups for fine-grained access control and retention policies
- Use data sources for service-based log discovery and analysis
- Query logs using either organizational method depending on your needs

Data sources make it easier to work with logs at scale by providing a service-centric view of your log data across your AWS infrastructure.

Features enabled by data sources

Data sources enable advanced log processing and analytics capabilities through field discovery and consistent data structures.

- **Facets:** Facets are indexed log fields that provide interactive filtering and analysis without writing queries. CloudWatch Logs automatically creates facets for data source name and type, and you can create facet policies on discovered fields to accelerate troubleshooting. Facets display value distributions and counts in CloudWatch Logs Insights, making it easy to identify patterns through point-and-click exploration.
- **Pipelines:** Create transformation pipelines that apply to all logs from a specific data source name and type. This allows you to define consistent processing rules for logs from the same source.
- **Field discovery:** CloudWatch Logs automatically discovers fields and their data types for each data source name and type combination based on pipeline processors. For AWS managed logs, field structures are predefined. For application logs, we recommend maintaining consistent log formats to maximize compatibility with analytics tools such as Amazon S3 tables that require well-defined field structures.

You can view the complete list of fields and their types for any data source using the `GetLogFields` API:

```
aws logs get-log-fields --data-source-name <name> --data-source-type <type>
```

This field discovery and consistency enables advanced analytics and integrations, as external tools can work with predictable field structures when processing your log data.

Supported AWS services for data sources

The following table lists the AWS services that are automatically categorized by CloudWatch Logs as data sources:

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
amazon_api_gateway	access
amazon_bedrock_agentcore	browser_usage
amazon_bedrock_agentcore	code_interpreter_application
amazon_bedrock_agentcore	code_interpreter_usage
amazon_bedrock_agentcore	gateway_application
amazon_bedrock_agentcore	identity_workload_application
amazon_bedrock_agentcore	memory_application
amazon_bedrock_agentcore	online_evaluation_config
amazon_bedrock_agentcore	runtime_application
amazon_bedrock_agentcore	runtime_usage
amazon_bedrock_agents	application
amazon_bedrock_agents	event

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
amazon_bedrock_knowledge_bases	application
amazon_cloudfront	access
amazon_cloudfront	connection
amazon_cloudwatch	rum_app_monitor
amazon_cognito	user_pool
amazon_ec2	verified_access
amazon_eks	api_server
amazon_eks	audit
amazon_eks	authenticator
amazon_eks	controller_manager
amazon_eks	scheduler
amazon_elasticache	cluster
amazon_eventbridge	eventbus_error
amazon_eventbridge	eventbus_info
amazon_eventbridge	pipes_execution
amazon_interactive_video_service	chat
amazon_managed_prometheus	scraper
amazon_managed_prometheus	workspace
amazon_msk	broker
amazon_msk	connect

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
amazon_opensearch_service	pipeline
amazon_q_business	events
amazon_q_business	sync_job
amazon_q_connect	events
amazon_route53	global_resolver_query
amazon_route53	hosted_zones
amazon_route53	profiles_resolver_query
amazon_route53	resolver_query
amazon_sagemaker	workteam_activity
amazon_ses	ingress_endpoints
amazon_ses	rule_sets
amazon_ses	traffic_policy
amazon_vpc	flow
amazon_vpc	route_server_peer
amazon_vpc_lattice	access
amazon_vpc_lattice	resource_access
amazon_workmail	access_control
amazon_workmail	authentication
amazon_workmail	personal_access
amazon_workmail	workmail_access

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
amazon_workmail	workmail_availability
aws_b2b_data_interchange	execution
aws_backup	data_access
aws_backup	hypervisor
aws_clean_rooms	analysis
aws_client_vpn	connection
aws_client_vpn	event
aws_cloudtrail	data
aws_cloudtrail	management
aws_elemental_mediapackage	egress_access
aws_elemental_mediapackage	ingress_access
aws_elemental_mediatailor	ad_decision
aws_elemental_mediatailor	manifest
aws_elemental_mediatailor	transcode
aws_entity_resolution	id_mapping_workflow
aws_entity_resolution	matching_workflow
aws_iot_fleetwise	error
aws_mainframe_modernization	batch_job
aws_mainframe_modernization	config
aws_mainframe_modernization	console

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
aws_mainframe_modernization	dataset_import
aws_network_firewall	alert
aws_network_firewall	flow
aws_network_firewall	tls
aws_nlb	access
aws_pcs	job_completion
aws_pcs	scheduler
aws_shield	protection_flow
aws_step_functions	express
aws_step_functions	standard
aws_transfer_family	server
aws_waf	access

Supported third-party sources for data sources

The following table lists the third-party sources that are automatically categorized by CloudWatch Logs as data sources when ingested through pipelines:

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
crowdstrike_falcon	detection_finding
crowdstrike_falcon	process_activity

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
github_auditlogs	account_change
github_auditlogs	api_activity
github_auditlogs	entity_management
microsoft_entraid	account_change
microsoft_entraid	authentication
microsoft_entraid	entity_management
microsoft_entraid	user_access_management
microsoft_office365	account_change
microsoft_office365	application_lifecycle
microsoft_office365	authentication
microsoft_office365	compliance_finding
microsoft_office365	detection_finding
microsoft_office365	email_activity
microsoft_office365	file_hosting_activity
microsoft_office365	group_management
microsoft_office365	incident_finding
microsoft_office365	user_access_management
microsoft_office365	vulnerability_finding
microsoft_office365	web_resources_activity
microsoft_windows	account_change

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
microsoft_windows	authentication
microsoft_windows	entity_management
microsoft_windows	event_log_activity
microsoft_windows	file_system_activity
microsoft_windows	group_management
microsoft_windows	kernel_activity
okta_auth0	api_activity
okta_auth0	authentication
okta_sso	api_activity
okta_sso	authentication
okta_sso	detection_finding
okta_sso	entity_management
paloaltonetworks_nextgenerationfirewall	authentication
paloaltonetworks_nextgenerationfirewall	detection_finding
paloaltonetworks_nextgenerationfirewall	network_activity
paloaltonetworks_nextgenerationfirewall	process_activity
sentinelone_endpointsecurity	dns_activity

Data Source Name (@data_source_name field)	Data Source Type (@data_source_type field)
sentinelone_endpointsecurity	file_system_activity
sentinelone_endpointsecurity	http_activity
sentinelone_endpointsecurity	process_activity
servicenow_cmdb	api_activity
servicenow_cmdb	datastore_activity
servicenow_cmdb	entity_management
wiz_cnapp	api_activity
wiz_cnapp	authentication
wiz_cnapp	compliance_finding
wiz_cnapp	detection_finding
wiz_cnapp	vulnerability_finding
zscaler_internetaccess	authentication
zscaler_internetaccess	dns_activity
zscaler_internetaccess	http_activity
zscaler_internetaccess	network_activity

Analyzing log data with CloudWatch Logs Insights

With CloudWatch Logs Insights, you can interactively search and analyze your log data in Amazon CloudWatch Logs. You can perform queries to help you more efficiently and effectively respond to operational issues. In addition to querying using log groups, you can query using facets, data source, and data type. If an issue occurs, you can use CloudWatch Logs Insights to identify potential causes and validate deployed fixes.

CloudWatch Logs Insights supports three query languages that you can use for your queries:

- A purpose-built **Logs Insights query language (Logs Insights QL)** with a few simple but powerful commands.
- **OpenSearch Service Piped Processing Language (PPL)**. OpenSearch PPL enables you to analyze your logs using a set of commands delimited by pipes (`|`).

With OpenSearch PPL you can retrieve, query, and analyze data by using commands that are piped together, making it easier to understand and compose complex queries. The syntax enables the chaining of commands to transform and process data. With PPL, you can filter and aggregate data, and use a rich set of math, string, date, conditional and other functions for analysis.

- **OpenSearch Service Structured Query Language (SQL)**. With OpenSearch SQL queries, you can analyze your logs in a declarative manner. You can use commands such as `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, and various other commands and functions available in SQL. You can execute `JOINS` across log groups, correlate data across logs using sub-queries, and use the rich set of JSON, Mathematical, String, Conditional and other SQL functions to perform powerful analysis on logs.

When you use either SQL or PPL commands, make sure to enclose fields with special characters (non-alphabetic and non-numeric) in backticks to successfully query them. For example, enclose `@message`, `Operation.Export`, and `Test::Field` in backticks. You don't need to enclose fields with purely alphabetical names in backticks.

CloudWatch Logs Insights offers the following features that are available for use with any of the query languages.

- Automatic [discovery of log fields](#) in logs from AWS services such as Amazon Route 53, AWS Lambda, AWS CloudTrail, and Amazon VPC, and any application or custom log that emits log events as JSON.
- Creating [field indexes](#) to reduce costs and speed results, especially for queries of large number of log groups or log events. After creating field indexes of fields that are common in your log events, you can use them in a query. The query skips processing log events that are known to not include the indexed field, and processes less data.

 Note

The `filterIndex` command is available only in Logs Insights QL.

- [Detection and analysis of patterns](#) in your log events. A pattern is a shared text structure that recurs among your log fields. When you view the results of a query, you can choose the **Patterns** tab to see the patterns that CloudWatch Logs found based on a sample of your results.
- [Saving queries](#), seeing your query history, and re-running saved queries. This can help you run complex queries when you need, without having to re-create them each time that you want to run them.
- [Adding queries to dashboards](#).
- [Encrypting query results with AWS Key Management Service](#).
- [Query generation using natural language](#) lets you use natural language to create CloudWatch Logs Insights queries. You can ask questions about or describe the data you're looking for, then the AI generates a query based on your prompt and provides a line-by-line explanation of how the query works.
- [Use facets to group, filter, and interactively explore your logs](#).

The following CloudWatch Logs Insights features are supported only when you use Logs Insights QL.

- Querying logs in the [Infrequent Access log class](#).
- [Comparison queries](#) that compare log events in a log group with log events from a previous time period.

⚠ Important

CloudWatch Logs Insights can't access log events with timestamps that pre-date the creation time of the log group.

If you are signed in to an account set up as a monitoring account in CloudWatch cross-account observability, you can run CloudWatch Logs Insights queries on log groups in source accounts linked to this monitoring account. You can run a query that queries multiple log groups located in different accounts. For more information, see [CloudWatch cross-account observability](#).

When you create queries using Logs Insights QL, you can also use natural language to create CloudWatch Logs Insights queries. To do so, ask questions about or describe the data you're looking for. This AI-assisted capability generates a query based on your prompt and provides a line-by-line explanation of how the query works. For more information, see [Use natural language to generate and update CloudWatch Logs Insights queries](#).

Queries using any of the supported query languages time out after 60 minutes, if they have not completed. Query results are available for seven days.

CloudWatch Logs Insights queries incur charges based on the amount of data that is queried, regardless of query language. For more information, see [Amazon CloudWatch Pricing](#).

You can use CloudWatch Logs Insights to search log data that was sent to CloudWatch Logs on November 5, 2018 or later.

⚠ Important

If your network security team doesn't allow the use of web sockets, you can't currently access the CloudWatch Logs Insights portion of the CloudWatch console. You can use the CloudWatch Logs Insights query capabilities using APIs. For more information, see [StartQuery](#) in the *Amazon CloudWatch Logs API Reference*.

Contents

- [Supported query languages](#)
- [Use natural language to generate and update CloudWatch Logs Insights queries](#)
- [Supported logs and discovered fields](#)

- [Create field indexes to improve query performance and reduce scan volume](#)
- [Use facets to group and explore logs](#)
- [Pattern analysis](#)
- [Save and re-run CloudWatch Logs Insights queries](#)
- [Add query to dashboard or export query results](#)
- [View running queries or query history](#)
- [Encrypt query results with AWS Key Management Service](#)
- [Generate a natural language summary from CloudWatch Logs Insights query results](#)

Supported query languages

The following sections list the commands supported in each query language. They also describe the syntax format and provide sample queries.

Topics

- [CloudWatch Logs Insights query language \(Logs Insights QL\)](#)
- [OpenSearch Piped Processing Language \(PPL\)](#)
- [OpenSearch Structured Query Language \(SQL\)](#)

CloudWatch Logs Insights query language (Logs Insights QL)

This section includes full documentation of Logs Insights QL commands and functions. It also includes sample queries for this language.

For information on other query languages you can use, see [OpenSearch Service PPL](#), [OpenSearch Service SQL](#), and [CloudWatch Metrics Insights](#).

Topics

- [CloudWatch Logs Insights language query syntax](#)
- [Get started with Logs Insights QL: Query tutorials](#)
- [Sample queries](#)
- [Compare \(diff\) with previous time ranges](#)
- [Visualize log data in graphs](#)

CloudWatch Logs Insights language query syntax

This section provides details about the Logs Insights QL. The query syntax supports different functions and operations that include but aren't limited to general functions, arithmetic and comparison operations, and regular expressions.

Important

To avoid incurring excessive charges by running large queries, keep in mind the following best practices:

- Select only the necessary log groups for each query.
- Always specify the narrowest possible time range for your queries.
- When you use the console to run queries, cancel all your queries before you close the CloudWatch Logs Insights console page. Otherwise, queries continue to run until completion.
- When you add a CloudWatch Logs Insights widget to a dashboard, ensure that the dashboard is not refreshing at a high frequency, because each refresh starts a new query.

To create queries that contain multiple commands, separate the commands with the pipe character (`|`).

To create queries that contain comments, set off the comments with the hash character (`#`).

Note

CloudWatch Logs Insights automatically discovers fields for different log types and generates fields that start with the `@` character. For more information about these fields, see [Supported logs and discovered fields](#) in the *Amazon CloudWatch User Guide*.

The following table briefly describes each command. Following this table is a more comprehensive description of each command, with examples.

Note

All Logs Insights QL query commands are supported on log groups in the Standard log class. Log groups in the Infrequent Access log class support all Logs Insights QL query commands except `pattern`, `diff`, and `unmask`.

<u>anomaly</u>	Identifies unusual patterns in your log data using machine learning.
<u>display</u>	Displays a specific field or fields in query results.
<u>fields</u>	Displays specific fields in query results and supports functions and operations you can use to modify field values and create new fields to use in your query.
<u>filter</u>	Filters the query to return only the log events that match one or more conditions.
<u>filterIndex</u>	Forces a query to attempt to scan only the log groups that are both indexed on the field mentioned in a field index and also contain a value for the that field index. This reduces scanned volume by attempting to scan only log events from these log groups that contain the value specified in the query for this field index. This command is not supported for log groups in the Infrequent Access log class.
<u>pattern</u>	Automatically clusters your log data into patterns. A pattern is shared text structure that recurs among your log fields. CloudWatch Logs Insights provides ways for you to analyze the patterns found in your log events. For more information, see Pattern analysis .
<u>diff</u>	Compares the log events found in your requested time period with the log events from a previous time period of equal length, so that you can look for trends and find out if certain log events are new.

<u>parse</u>	Extracts data from a log field to create an extracted field that you can process in your query. parse supports both glob mode using wildcards , and regular expressions.
<u>sort</u>	Displays the returned log events in ascending (asc) or descending (desc) order.
<u>SOURCE</u>	Including SOURCE in a query is a useful way to specify a large amount of log groups based on log group name prefix, account identifiers, and log group class to include in a query. This command is supported only when you create a query in the AWS CLI or programmatically, not in the CloudWatch console.
<u>stats</u>	Calculate aggregate statistics using values in the log fields.
<u>limit</u>	Specifies a maximum number of log events that you want your query to return. Useful with sort to return "top 20" or "most recent 20" results.
<u>dedup</u>	Removes duplicate results based on specific values in fields that you specify.
<u>unmask</u>	Displays all the content of a log event that has some content masked because of a data protection policy. For more information about data protection in log groups, see Help protect sensitive log data with masking .
<u>unnest</u>	Flattens a list taken as input to produce multiple records with a single record for each element in the list.
<u>Other operations and functions</u>	CloudWatch Logs Insights also supports many comparison, arithmetic, datetime, numeric, string, IP address, and general functions and operations.

The following sections provide more details about the CloudWatch Logs Insights query commands.

Topics

- [Logs Insights QL commands supported in log classes](#)

- [anomaly](#)
- [display](#)
- [fields](#)
- [filter](#)
- [filterIndex](#)
- [SOURCE](#)
- [pattern](#)
- [diff](#)
- [parse](#)
- [sort](#)
- [stats](#)
- [limit](#)
- [dedup](#)
- [unmask](#)
- [unnest](#)
- [Boolean, comparison, numeric, datetime, and other functions](#)
- [Fields that contain special characters](#)
- [Use aliases and comments in queries](#)

Logs Insights QL commands supported in log classes

All Logs Insights QL query commands are supported on log groups in the Standard log class. Log groups in the Infrequent Access log class support all query commands except `pattern`, `diff`, `filterIndex`, and `unmask`.

anomaly

Use `anomaly` to automatically identify unusual patterns and potential issues within your log data using machine learning.

The `anomaly` command extends the existing `pattern` functionality and leverages advanced analytics to help identify potential anomalies in log data. You can use `anomaly` to reduce the time it takes to identify and resolve operational issues by automatically surfacing unusual patterns or behaviors in your logs.

The `anomaly` command works with the [pattern](#) command to first identify log patterns, then detect anomalies within those patterns. You can also combine `anomaly` with the [filter](#) or [sort](#) commands to focus anomaly detection on specific subsets of your data.

Anomaly Command Input

The `anomaly` command is typically used after the [pattern](#) command to analyze the patterns identified in your log data. The command does not require additional parameters and analyzes the output from preceding commands in your query.

Types of Anomalies Identified

The `anomaly` command identifies five distinct types of anomalies:

- *Pattern Frequency Anomalies*: Unusual frequencies of specific log patterns, such as when an application starts generating more error messages than usual.
- *New Pattern Anomalies*: Previously unseen log patterns that may indicate new types of errors or messages appearing in your logs.
- *Token Variation Anomalies*: Unexpected changes in log message contents that may indicate unusual variations in expected log formats.
- *Numerical Token Anomalies*: Unusual changes in numerical values within logs that can help detect potential performance issues or unexpected metric variations.
- *HTTP Error Code Anomalies*: Patterns related to HTTP error responses, particularly useful when monitoring web applications and APIs.

Anomaly Command Output

The `anomaly` command preserves all fields from the input data and adds anomaly detection results to help identify unusual patterns in your log data.

Examples

The following command identifies patterns in your log data and then detects anomalies within those patterns:

```
fields @timestamp, @message
| pattern @message
| anomaly
```


The anomaly command can be used with filtering to focus on specific log types:

```
fields @timestamp, @message
| filter @type = "REPORT"
| pattern @message
| anomaly
```

The anomaly command can be combined with sorting to organize results:

```
fields @timestamp, @message
| filter @type = "ERROR"
| pattern @message
| anomaly
| sort @timestamp desc
```

display

Use display to show a specific field or fields in query results.

The display command shows only the fields you specify. If your query contains multiple display commands, the query results show only the field or fields that you specified in the final display command.

Example: Display one field

The code snippet shows an example of a query that uses the parse command to extract data from @message to create the extracted fields loggingType and loggingMessage. The query returns all log events where the values for loggingType are **ERROR**. display shows only the values for loggingMessage in the query results.

```
fields @message
| parse @message "[*] *" as loggingType, loggingMessage
| filter loggingType = "ERROR"
| display loggingMessage
```

Tip

Use display only once in a query. If you use display more than once in a query, the query results show the field specified in the last occurrence of display command being used.

fields

Use `fields` to show specific fields in query results.

If your query contains multiple `fields` commands and doesn't include a `display` command, the results display all of the fields that are specified in the `fields` commands.

Example: Display specific fields

The following example shows a query that returns 20 log events and displays them in descending order. The values for `@timestamp` and `@message` are shown in the query results.

```
fields @timestamp, @message
| sort @timestamp desc
| limit 20
```

Use `fields` instead of `display` when you want to use the different functions and operations supported by `fields` for modifying field values and creating new fields that can be used in queries.

You can use the `fields` command with the keyword `as` to create extracted fields that use fields and functions in your log events. For example, `fields ispresent as isRes` creates an extracted field named `isRes`, and the extracted field can be used in the rest of your query.

filter

Use `filter` to get log events that match one or more conditions.

Example: Filter log events using one condition

The code snippet shows an example of a query that returns all log events where the value for `range` is greater than **3000**. The query limits the results to 20 log events and sorts the logs events by `@timestamp` and in descending order.

```
fields @timestamp, @message
| filter (range>3000)
| sort @timestamp desc
| limit 20
```

Example: Filter log events using more than one condition

You can use the keywords `and` and `or` to combine more than one condition.

The code snippet shows an example of a query that returns log events where the value for `range` is greater than **3000** and value for `accountId` is equal to **123456789012**. The query limits the results to 20 log events and sorts the logs events by `@timestamp` and in descending order.

```
fields @timestamp, @message
| filter (range>3000 and accountId=123456789012)
| sort @timestamp desc
| limit 20
```

Indexed fields and the filter command

If you have created field indexes for a log group, you can leverage those field indexes to make your filter queries more efficient and reduce scanned volume. For example, suppose you have created a field index for `requestId`. Then, any CloudWatch Logs Insights query on that log group that includes `filter requestId = value` or `filter requestId IN [value, value, ...]` will attempt to skip processing log events that are known not to include the indexed field. By attempting to scan only the log events that are known to contain that indexed field, scan volume can be reduced and the query is faster.

For more information about field indexes and how to create them, see [Create field indexes to improve query performance and reduce scan volume](#).

Important

Only queries with `filter fieldName = ...` and `filter fieldName IN...` will benefit from the field index improvements. Queries with `filter fieldName like` don't use indexes and always scan all log events in the selected log groups.

Example: Find log events that are related to a certain request ID, using indexes

This example assumes that you have created a field index on `requestId`. For log groups that use this field index, the query will leverage field indexes to attempt to scan the least amount of log events to find events with `requestId` with a value of 123456

```
fields @timestamp, @message
| filter requestId = "123456"
| limit 20
```

Matches and regular expressions in the filter command

The filter command supports the use of regular expressions. You can use the following comparison operators (=, !=, <, <=, >, >=) and Boolean operators (and, or, and not).

You can use the keyword `in` to test for set membership and check for elements in an array. To check for elements in an array, put the array after `in`. You can use the Boolean operator `not` with `in`. You can create queries that use `in` to return log events where fields are string matches. The fields must be complete strings. For example, the following code snippet shows a query that uses `in` to return log events where the field `logGroup` is the complete string `example_group`.

```
fields @timestamp, @message
| filter logGroup in ["example_group"]
```

You can use the keyword phrases `like` and `not like` to match substrings. You can use the regular expression operator `=~` to match substrings. To match a substring with `like` and `not like`, enclose the substring that you want to match in single or double quotation marks. You can use regular expression patterns with `like` and `not like`. To match a substring with the regular expression operator, enclose the substring that you want to match in forward slashes. The following examples contain code snippets that show how you can match substrings using the `filter` command.

Examples: Match substrings

The following examples return log events where `f1` contains the word ***Exception***. All three examples are case sensitive.

The first example matches a substring with `like`.

```
fields f1, f2, f3
| filter f1 like "Exception"
```

The second example matches a substring with `like` and a regular expression pattern.

```
fields f1, f2, f3
| filter f1 like /Exception/
```

The third example matches a substring with a regular expression.

```
fields f1, f2, f3
```

```
| filter f1 =~ /Exception/
```

Example: Match substrings with wildcards

You can use the period symbol (.) as a wildcard in regular expressions to match substrings. In the following example, the query returns matches where the value for f1 begins with the string `ServiceLog`.

```
fields f1, f2, f3
| filter f1 like /ServiceLog./
```

You can place the asterisk symbol after the period symbol (.*) to create a greedy quantifier that returns as many matches as possible. For example, the following query returns matches where the value for f1 not only begins with the string `ServiceLog`, but also includes the string `ServiceLog`.

```
fields f1, f2, f3
| filter f1 like /ServiceLog.*/
```

Possible matches can be formatted like the following:

- `ServiceLogSampleApiLogGroup`
- `SampleApiLogGroupServiceLog`

Example: Exclude substrings from matches

The following example shows a query that returns log events where f1 doesn't contain the word ***Exception***. The example is case sensitive.

```
fields f1, f2, f3
| filter f1 not like "Exception"
```

Example: Match substrings with case-insensitive patterns

You can match substrings that are case insensitive with `like` and regular expressions. Place the following parameter (**?i**) before the substring you want to match. The following example shows a query that returns log events where f1 contains the word ***Exception*** or ***exception***.

```
fields f1, f2, f3
```

```
| filter f1 like /(?!i)Exception/
```

filterIndex

Use `filterIndex` to return indexed data only, by forcing a query to scan only log groups that are indexed on a field that you specify in the query. For these log groups that are indexed on this field, it further optimizes the query by skipping the log groups that do not have any log events containing the field specified in the query for the indexed field. It further reduces scanned volume by attempting to scan only log events from these log groups that match the value specified in the query for this field index. For more information about field indexes and how to create them, see [Create field indexes to improve query performance and reduce scan volume](#).

Using `filterIndex` with indexed fields can help you query log groups that include petabytes of log data efficiently by limiting the actual search space to log groups and log events that have field indexes.

For example, suppose that you have created a field index for `IPAddress` in some of the log groups in your account. You can then create the following query and choose to query all log groups in the account to find log events that include the value `198.51.100.0` in the `IPAddress` field.

```
fields @timestamp, @message
| filterIndex IPAddress = "198.51.100.0"
| limit 20
```

The `filterIndex` command causes this query to attempt to skip all log groups that are not indexed for `IPAddress`. Additionally, within the log groups that are indexed, the query skips log events that have an `IPAddress` field but not observed `198.51.100.0` as the value for that field.

Use the `IN` operator to expand the results to any of multiple values for the indexed fields. The following example finds logs events that include either the value `198.51.100.0` or `198.51.100.1` in the `IPAddress` field.

```
fields @timestamp, @message
| filterIndex IPAddress in ["198.51.100.0", "198.51.100.1"]
| limit 20
```

CloudWatch Logs provides default field indexes for all log groups in the Standard log class. Default field indexes are automatically available for the following fields:

- `@logStream`

- `@aws.region`
- `@aws.account`
- `@source.log`
- `@data_source_name`
- `@data_source_type`
- `@data_format`
- `traceId`
- `severityText`
- `attributes.session.id`

CloudWatch Logs provides default field indexes for certain data source name and type combinations as well. Default field indexes are automatically available for the following data source name and type combinations:

Data Source Name and Type	Default Field Indexes
<code>amazon_vpc.flow</code>	<code>action</code> <code>logStatus</code> <code>region</code> <code>flowDirection</code> <code>type</code>
<code>amazon_route53_resolver_query</code>	<code>query_type</code> <code>transport</code> <code>rcode</code>
<code>aws_waf.access</code>	<code>action</code> <code>httpRequest.country</code>
<code>aws_cloudtrail.data</code>	<code>eventSource</code>

Data Source Name and Type	Default Field Indexes
aws_cloudtrail.management	eventName awsRegion userAgent errorCode eventType managementEvent readOnly eventCategory requestId

Default field indexes are in addition to any custom field indexes you define within your policy. Default field indexes are not counted towards your [field index quota](#).

filterIndex compared to filter

To illustrate the difference between `filterIndex` and `filter`, consider the following example queries. Assume that you have created a field index for `IPaddress`, for four of your log groups, but not for a fifth log group. The following query using `filterIndex` will skip scanning the log group that doesn't have the field indexed. For each indexed log group, it attempts to scan only log events that have the indexed field, and it also returns only results from after the field index was created.

```
fields @timestamp, @message
| filterIndex IPaddress = "198.51.100.0"
| limit 20
```

In contrast, if you use `filter` instead of `filterIndex` for a query of the same five log groups, the query will attempt to scan not only the log events that contain the value in the indexed log groups, but will also scan the fifth log group that isn't indexed, and it will scan every log event in that fifth log group.


```
fields @timestamp, @message
| filter IPaddress = "198.51.100.0"
| limit 20
```

SOURCE

Including SOURCE in a query is a useful way to specify the log groups and/or data sources to include in a query when you are using the AWS CLI or API to create a query. The SOURCE command is supported only in the AWS CLI and API, not in the CloudWatch console. When you use the CloudWatch console to start a query, you use the console interface to specify the log groups.

Query log groups

To use SOURCE to specify the log groups to query, you can use the following keywords:

- `namePrefix` runs the query against log groups that have names that start with the string that you specify. If you omit this, all log groups are queried.

You can include as many as five prefixes in the list.

- `accountIdentifier` runs the query against log groups in the specified AWS account. This works only when you run the query in a monitoring account. If you omit this, the default is to query all linked source accounts and the current monitoring account. For more information about cross-account observability, see [CloudWatch cross-account observability](#).

You can include as many as 20 account identifiers in the list.

- `logGroupClass` runs the query against log groups that are in the specified log class, either Standard or Infrequent Access. If you omit this, the default of Standard log class is used. For more information about log classes, see [Log classes](#).

Because you can specify large numbers of log groups to query this way, we recommend that you use SOURCE only in queries that leverage field indexes that you have created. For more information about indexing fields in log groups, see [Create field indexes to improve query performance and reduce scan volume](#)

The following example selects all log groups in the account. If this is a monitoring account then the log groups across monitoring and all the source accounts will be selected. If the total number of log groups exceed 10,000 then you will see an error prompting you to reduce the number of log groups by using a different log group selection method.

```
SOURCE logGroups()
```

The following example selects the log groups in the 111122223333 source account. If you start a query in a monitoring account in CloudWatch cross-account observability, log groups in all source accounts and in the monitoring account are selected by default.

```
SOURCE logGroups(accountIdentifiers:['111122223333'])
```

The next example selects log groups based on name prefixes.

```
SOURCE logGroups(namePrefix: ['namePrefix1', 'namePrefix2'])
```

The following example selects all log groups in the Infrequent Access log class. If you don't include the `class` identifier, the query applies only to log groups in the Standard log class, which is the default.

```
SOURCE logGroups(class: ['INFREQUENT_ACCESS'])
```

The next example selects log groups in the 111122223333 account that start with specific name prefixes and are in the Standard log class. The class is not mentioned in the command because Standard is the default log class value.

```
SOURCE logGroups(accountIdentifiers:['111122223333'], namePrefix: ['namePrefix1',  
'namePrefix2'])
```

The final example displays how to use the `SOURCE` command with the `start-query` AWS CLI command.

```
aws logs start-query  
--region us-east-1  
--start-time 1729728200  
--end-time 1729728215  
--query-string "SOURCE logGroups(namePrefix: ['Query']) | fields @message | limit 5"
```

Query data sources

To use `SOURCE` to specify the data sources to query, you can use the `dataSource` keyword. You can include as many as ten data sources in the list.

The following example selects the `amazon_vpc.flow` data source.

```
SOURCE dataSource(['amazon_vpc.flow'])
```

The following example selects the `amazon_vpc.flow` data source and limits the log groups based on a log group name prefix.

```
SOURCE dataSource(['amazon_vpc.flow']) logGroups(namePrefix: ['namePrefix1'])
```

pattern

Use `pattern` to automatically cluster your log data into patterns.

A pattern is shared text structure that recurs among your log fields. You can use `pattern` to surface emerging trends, monitor known errors, and identify frequently occurring or high-cost log lines. CloudWatch Logs Insights also provides a console experience you can use to find and further analyze patterns in your log events. For more information, see [Pattern analysis](#).

Because the `pattern` command automatically identifies common patterns, you can use it as a starting point to search and analyze your logs. You can also combine `pattern` with the [filter](#), [parse](#), or [sort](#) commands to identify patterns in more fine-tuned queries.

Pattern Command Input

The `pattern` command expects one of the following inputs: the `@message` field, an extracted field created using the [parse](#) command, or a string manipulated using one or more [String functions](#).

If CloudWatch Logs can't infer the type of data that a dynamic token represents, displays it as `<Token-number>`, and *number* indicates where in the pattern this token appears, compared to the other dynamic tokens.

Common examples of dynamic tokens include error codes, IP addresses, timestamps, and request IDs.

Pattern Command Output

The `pattern` command produces the following output:

- `@pattern`: A shared text structure that recurs among your log event fields. Fields that vary within a pattern, such as a request ID or timestamp, are represented by *tokens*. If CloudWatch Logs can determine the type of data that a dynamic token represents, it displays the token as

<*string-number*>. The *string* is a description of the type of data that the token represents. The *number* shows where in the pattern this token appears, compared to the other dynamic tokens.

CloudWatch Logs assigns the string part of the name based on analyzing the content of the log events that contain it.

If CloudWatch Logs can't infer the type of data that a dynamic token represents, displays it as <Token-*number*>, and *number* indicates where in the pattern this token appears, compared to the other dynamic tokens.

For example, [INFO] Request time: <Time-1> ms is a potential output for the log message [INFO] Request time: 327 ms.

- @ratio: The ratio of log events from a selected time period and specified log groups that match an identified pattern. For example, if half of the log events in the selected log groups and time period match the pattern, @ratio returns 0.50
- @sampleCount: A count of the number of log events from a selected time period and specified log groups that match an identified pattern.
- @severityLabel: The log severity or level, which indicates the type of information contained in a log. For example, Error, Warning, Info, or Debug.

Examples

The following command identifies logs with similar structures in specified log group(s) over the selected time range, grouping them by pattern and count

```
pattern @message
```

The pattern command can be used in combination with the [filter](#) command

```
filter @message like /ERROR/  
| pattern @message
```

The pattern command can be use with the [parse](#) and [sort](#) commands

```
filter @message like /ERROR/  
| parse @message 'Failed to do: *' as cause  
| pattern cause
```

```
| sort @sampleCount asc
```

diff

Compares the log events found in your requested time period with the log events from a previous time period of equal length. This way, you can look for trends and find whether specific log events are new.

Add a modifier to the `diff` command to specify the time period that you want to compare with:

- `diff` compares the log events in the currently selected time range to the log events of the immediately preceding time range.
- `diff previousDay` compares the log events in the currently selected time range to the log events from the same time the preceding day.
- `diff previousWeek` compares the log events in the currently selected time range to the log events from the same time the preceding week.
- `diff previousMonth` compares the log events in the currently selected time range to the log events from the same time the preceding month.

For more information, see [Compare \(diff\) with previous time ranges](#).

parse

Use `parse` to extract data from a log field and create an extracted field that you can process in your query. **parse** supports both glob mode using wildcards, and regular expressions. For information about regular expression syntax, see [Supported regular expressions \(regex\) syntax](#).

You can parse nested JSON fields with a regular expression.

Example: Parsing a nested JSON field

The code snippet shows how to parse a JSON log event that's been flattened during ingestion.

```
{'fieldsA': 'logs', 'fieldsB': [{'fA': 'a1'}, {'fA': 'a2'}]}
```

The code snippet shows a query with a regular expression that extracts the values for `fieldsA` and `fieldsB` to create the extracted fields `fld` and `array`.

```
parse @message "'fieldsA': '*, 'fieldsB': ['*']" as fld, array
```

Named capturing groups

When you use **parse** with a regular expression, you can use named capturing groups to capture a pattern into a field. The syntax is `parse @message (?<Name>pattern)`

The following example uses a capturing group on a VPC flow log to extract the ENI into a field named `NetworkInterface`.

```
parse @message /(?(<NetworkInterface>eni-.*?)/ | display NetworkInterface, @message
```

Note

JSON log events are flattened during ingestion. Currently, parsing nested JSON fields with a glob expression isn't supported. You can only parse JSON log events that include no more than 200 log event fields. When you parse nested JSON fields, you must format the regular expression in your query to match the format of your JSON log event.

Examples of the parse command

Use a glob expression to extract the fields `@user`, `@method`, and `@latency` from the log field `@message` and return the average latency for each unique combination of `@method` and `@user`.

```
parse @message "user=*, method:*, latency := *" as @user,  
    @method, @latency | stats avg(@latency) by @method,  
    @user
```

Use a regular expression to extract the fields `@user2`, `@method2`, and `@latency2` from the log field `@message` and return the average latency for each unique combination of `@method2` and `@user2`.

```
parse @message /user=(?(<user2>.*?), method:(?(<method2>.*?),  
    latency := (?(<latency2>.*?)/ | stats avg(latency2) by @method2,  
    @user2
```

Extracts the fields `loggingTime`, `loggingType` and `loggingMessage`, filters down to log events that contain `ERROR` or `INFO` strings, and then displays only the `loggingMessage` and `loggingType` fields for events that contain an `ERROR` string.

```

FIELDS @message
| PARSE @message "*" [*] "*" as loggingTime, loggingType, loggingMessage
| FILTER loggingType IN ["ERROR", "INFO"]
| DISPLAY loggingMessage, loggingType = "ERROR" as isError

```

sort

Use `sort` to display log events in ascending (`asc`) or descending (`desc`) order by a specified field. You can use this with the `limit` command to create "top N" or "bottom N" queries.

The sorting algorithm is an updated version of natural sorting. If you sort in ascending order, the following logic is used.

- All non-number values come before all number values. *Number values* are values that include only numbers, not a mix of numbers and other characters.
- For non-number values, the algorithm groups consecutive numeric characters and consecutive alphabetic characters into separate chunks for comparison. It orders non-numeric portions by their Unicode values, and it orders numeric portions by their length first and then by their numerical value.

For more information about Unicode order, see [List of Unicode characters](#).

For example, the following is the result of a sort in ascending order.

[illegible]

3

If you sort in descending order, the sort results are the reverse.

For example, the following query for Amazon VPC flow logs finds the top 15 packet transfers across hosts.

```
stats sum(packets) as packetsTransferred by srcAddr, dstAddr
| sort packetsTransferred desc
| limit 15
```

stats

Use `stats` to create visualizations of your log data such as bar charts, line charts, and stacked area charts. This helps you more efficiently identify patterns in your log data. CloudWatch Logs Insights generates visualizations for queries that use the `stats` function and one or more aggregation functions.

For example, the following query in a Route 53 log group returns visualizations showing the distribution of Route 53 records per hour, by query type.

```
stats count(*) by queryType, bin(1h)
```

All such queries can produce bar charts. If your query uses the `bin()` function to group the data by one field over time, you can also see line charts and stacked area charts.

The following time units and abbreviations are supported with the `bin` function. For all units and abbreviations that include more than one character, adding `s` to pluralize is supported. So both `hr` and `hrs` work to specify hours.

- millisecond `ms` `msec`
- second `s` `sec`
- minute `m` `min`
- hour `h` `hr`
- day `d`
- week `w`
- month `mo` `mon`
- quarter `q` `qtr`

- `year y yr`

Topics

- [Visualize time series data](#)
- [Visualize log data grouped by fields](#)
- [Use multiple stats commands in a single query](#)
- [Functions to use with stats](#)

Visualize time series data

Time series visualizations work for queries with the following characteristics:

- The query contains one or more aggregation functions. For more information, see [Aggregation Functions in the Stats Command](#).
- The query uses the `bin()` function to group the data by one field.

These queries can produce line charts, stacked area charts, bar charts, and pie charts.

Examples

For a complete tutorial, see [the section called “Tutorial: Run a query that produces a time series visualization”](#).

Here are more example queries that work for time series visualization.

The following query generates a visualization of the average values of the `myfield1` field, with a data point created every five minutes. Each data point is the aggregation of the averages of the `myfield1` values from the logs from the previous five minutes.

```
stats avg(myfield1) by bin(5m)
```

The following query generates a visualization of three values based on different fields, with a data point created every five minutes. The visualization is generated because the query contains aggregate functions and uses `bin()` as the grouping field.

```
stats avg(myfield1), min(myfield2), max(myfield3) by bin(5m)
```

Line chart and stacked area chart restrictions

Queries that aggregate log entry information but don't use the `bin()` function can generate bar charts. However, the queries cannot generate line charts or stacked area charts. For more information about these types of queries, see [the section called “Visualize log data grouped by fields”](#).

Visualize log data grouped by fields

You can produce bar charts for queries that use the `stats` function and one or more aggregation functions. For more information, see [Aggregation Functions in the Stats Command](#).

To see the visualization, run your query. Then choose the **Visualization** tab, select the arrow next to **Line**, and choose **Bar**. Visualizations are limited to up to 100 bars in the bar chart.

Examples

For a complete tutorial, see [the section called “Tutorial: Run a query that produces a visualization grouped by log fields”](#). The following paragraphs include more example queries for visualization by fields.

The following VPC flow log query finds the average number of bytes transferred per session for each destination address.

```
stats avg(bytes) by dstAddr
```

You can also produce a chart that includes more than one bar for each resulting value. For example, the following VPC flow log query finds the average and maximum number of bytes transferred per session for each destination address.

```
stats avg(bytes), max(bytes) by dstAddr
```

The following query finds the number of Amazon Route 53 query logs for each query type.

```
stats count(*) by queryType
```

Use multiple stats commands in a single query

You can use as many as two `stats` commands in a single query. This enables you to perform an additional aggregation on the output of the first aggregation.

Example: Query with two stats commands

For example, the following query first find the total traffic volume in 5-minute bins, then calculates the highest, lowest, and average traffic volume among those 5-minute bins.

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length)/1024/1024 as logs_mb BY bin(5m)
| STATS max(logs_mb) AS peak_ingest_mb,
      min(logs_mb) AS min_ingest_mb,
      avg(logs_mb) AS avg_ingest_mb
```

Example: Combine multiple stats commands with other functions such as filter, fields, bin

You can combine two stats commands with other commands such as `filter` and `fields` in a single query. For example, the following query finds the number of distinct IP addresses in sessions and finds the number of sessions by client platform, filters those IP addresses, and then finally finds the average of session requests per client platform.

```
STATS count_distinct(client_ip) AS session_ips,
      count(*) AS requests BY session_id, client_platform
| FILTER session_ips > 1
| STATS count(*) AS multiple_ip_sessions,
      sum(requests) / count(*) AS avg_session_requests BY client_platform
```

You can use `bin` and `dateceil` functions in queries with multiple stats commands. For example, the following query first combines messages into 5-minute blocks, then aggregates those 5-minute blocks into 10-minute blocks and calculates the highest, lowest, and average traffic volumes within each 10-minute block.

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length) / 1024 / 1024 AS logs_mb BY BIN(5m) as @t
| STATS max(logs_mb) AS peak_ingest_mb,
      min(logs_mb) AS min_ingest_mb,
      avg(logs_mb) AS avg_ingest_mb BY dateceil(@t, 10m)
```

Notes and limitations

A query can have a maximum of two stats commands. This quota can't be changed.

If you use a `sort` or `limit` command, it must appear after the second stats command. If it is before the second stats command, the query is not valid.

When a query has two stats commands, the partial results from the query do not begin displaying until the first stats aggregation is complete.

In the second stats command in a single query, you can refer only to fields that are defined in the first stats command. For example, the following query is not valid because the @message field won't be available after the first stats aggregation.

```
FIELDS @message
| STATS SUM(Fault) by Operation
# You can only reference `SUM(Fault)` or Operation at this point
| STATS MAX(strlen(@message)) AS MaxMessageSize # Invalid reference to @message
```

Any fields that you reference after the first stats command must be defined in that first stats command.

```
STATS sum(x) as sum_x by y, z
| STATS max(sum_x) as max_x by z
# You can only reference `max(sum_x)`, max_x or z at this point
```

Important

The bin function always implicitly uses the @timestamp field. This means that you can't use bin in the second stats command without using the first stats command to propagate the timestamp field. For example, the following query is not valid.

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length) AS ingested_bytes BY @logStream
| STATS avg(ingested_bytes) BY bin(5m) # Invalid reference to @timestamp field
```

Instead, define the @timestamp field in the first stats command, and then you can use it with dateceil in the second stats command as in the following example.

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length) AS ingested_bytes, max(@timestamp) as @t BY @logStream
| STATS avg(ingested_bytes) BY dateceil(@t, 5m)
```

Functions to use with stats

CloudWatch Logs Insights supports both stats aggregation functions and stats non-aggregation functions.

Use stats aggregation functions in the stats command and as arguments for other functions.

Function	Result type	Description
<code>avg(fieldName: NumericLogField)</code>	number	The average of the values in the specified field.
<code>count()</code> <code>count(fieldName: LogField)</code>	number	Counts the log events. <code>count()</code> (or <code>count(*)</code>) counts all events returned by the query, while <code>count(fieldName)</code> counts all records that include the specified field name.
<code>count_distinct(fieldName: LogField)</code>	number	Returns the number of unique values for the field. If the field has very high cardinality (contains many unique values), the value returned by <code>count_distinct</code> is just an approximation.
<code>max(fieldName: LogField)</code>	LogFieldValue	The maximum of the values for this log field in the queried logs.
<code>min(fieldName: LogField)</code>	LogFieldValue	The minimum of the values for this log field in the queried logs.
<code>pct(fieldName: LogFieldValue, percent: number)</code>	LogFieldValue	A percentile indicates the relative standing of a value in a dataset. For example, <code>pct(@duration, 95)</code> returns the <code>@duration</code> value at which 95 percent of the values of <code>@duration</code> are lower than this value, and 5 percent are higher than this value.
<code>stddev(fieldName: NumericLogField)</code>	number	The standard deviation of the values in the specified field.
<code>sum(fieldName: NumericLogField)</code>	number	The sum of the values in the specified field.

Stats non-aggregation functions

Use non-aggregation functions in the `stats` command and as arguments for other functions.

Function	Result type	Description
<code>earliest(fieldName: LogField)</code>	LogField	Returns the value of <code>fieldName</code> from the log event that has the earliest timestamp in the queried logs.
<code>latest(fieldName: LogField)</code>	LogField	Returns the value of <code>fieldName</code> from the log event that has the latest timestamp in the queried logs.
<code>sortsFirst(fieldName: LogField)</code>	LogField	Returns the value of <code>fieldName</code> that sorts first in the queried logs.
<code>sortsLast(fieldName: LogField)</code>	LogField	Returns the value of <code>fieldName</code> that sorts last in the queried logs.

limit

Use `limit` to specify the number of log events that you want your query to return. If you omit `limit`, the query will return as many as 10,000 log events in the results.

For example, the following example returns only the 25 most recent log events

```
fields @timestamp, @message | sort @timestamp desc | limit 25
```

dedup

Use `dedup` to remove duplicate results based on specific values in fields that you specify. You can use `dedup` with one or more fields. If you specify one field with `dedup`, only one log event is returned for each unique value of that field. If you specify multiple fields, then one log event is returned for each unique combination of values for those fields.

Duplicates are discarded based on the sort order, with only the first result in the sort order being kept. We recommend that you sort your results before putting them through the `dedup` command.

If the results are not sorted before being run through dedup, then the default descending sort order using `@timestamp` is used.

Null values are not considered duplicates for evaluation. Log events with null values for any of the specified fields are retained. To eliminate fields with null values, use **filter** using the `isPresent(field)` function.

The only query command that you can use in a query after the dedup command is `limit`.

When you use dedup in a query, the console displays a message such as **Showing X of Y records**, where X is the number of deduplicated results and Y is the total number of records matched before deduplication. This indicates that duplicate records were removed and does not mean that data is missing.

Example: See only the most recent log event for each unique value of the field named server

The following example displays the `timestamp`, `server`, `severity`, and `message` fields for only the most recent event for each unique value of `server`.

```
fields @timestamp, server, severity, message
| sort @timestamp desc
| dedup server
```

For more samples of CloudWatch Logs Insights queries, see [General queries](#).

unmask

Use `unmask` to display all the content of a log event that has some content masked because of a data protection policy. To use this command, you must have the `logs:Unmask` permission.

For more information about data protection in log groups, see [Help protect sensitive log data with masking](#).

unnest

Use `unnest` to flatten a list taken as input to produce multiple records with a single record for each element in the list. Based on the number of items a field contains, this command discards the current record and generates new records. Each record includes the `unnested_field`, which represents an item. All other fields come from the original record.

The input for `unnest` is `LIST`, which comes from the `jsonParse` function. For more information, see [Structure types](#). Any other types, such as `MAP`, `String` and numbers, are treated as a list with one item in `unnest`.

Command structure

The following example describes the format of this command.

```
unnest field into unnested_field
```

Example query

The following example parses a JSON object string and expands a list of field events.

```
fields jsonParse(@message) as json_message
| unnest json_message.events into event
| display event.name
```

The log event for this example query could be a JSON string as follows:

```
{
  "events": [
    {
      "name": "exception"
    },
    {
      "name": "user action"
    }
  ]
}
```

In this case, the sample query produces two records in the query result, one with `event.name` as `exception` and another with `event.name` as **user action**

Example query

The following example flattens a list and then filters out items.

```
fields jsonParse(@message) as js
| unnest js.accounts into account
| filter account.type = "internal"
```


Example query

The following example flattens a list for aggregation.

```
fields jsonParse(trimmedData) as accounts
| unnest accounts into account
| stats sum(account.droppedSpans) as n by account.accountId
| sort n desc
| limit 10
```

Boolean, comparison, numeric, datetime, and other functions

CloudWatch Logs Insights supports many other operations and functions in queries, as explained in the following sections.

Topics

- [Arithmetic operators](#)
- [Boolean operators](#)
- [Comparison operators](#)
- [Numeric operators](#)
- [Structure types](#)
- [Datetime functions](#)
- [General functions](#)
- [JSON functions](#)
- [IP address string functions](#)
- [String functions](#)

Arithmetic operators

Arithmetic operators accept numeric data types as arguments and return numeric results. Use arithmetic operators in the `filter` and `fields` commands and as arguments for other functions.

Operation	Description
<code>a + b</code>	Addition

Operation	Description
$a - b$	Subtraction
$a * b$	Multiplication
a / b	Division
$a ^ b$	Exponentiation ($2 ^ 3$ returns 8)
$a \% b$	Remainder or modulus ($10 \% 3$ returns 1)

Boolean operators

Use the Boolean operators **and**, **or**, and **not**.

Note

Use Boolean operators only in functions that return a value of **TRUE** or **FALSE**.

Comparison operators

Comparison operators accept all data types as arguments and return a Boolean result. Use comparison operations in the `filter` command and as arguments for other functions.

Operator	Description
<code>=</code>	Equal
<code>!=</code>	Not equal
<code><</code>	Less than
<code>></code>	Greater than
<code><=</code>	Less than or equal to
<code>>=</code>	Greater than or equal to

Numeric operators

Numeric operations accept numeric data types as arguments and return numeric results. Use numeric operations in the `filter` and `fields` commands and as arguments for other functions.

Operation	Result type	Description
<code>abs(a: number)</code>	number	Absolute value
<code>ceil(a: number)</code>	number	Round to ceiling (the smallest integer that is greater than the value of a)
<code>floor(a: number)</code>	number	Round to floor (the largest integer that is smaller than the value of a)
<code>greatest(a: number, ...numbers: number[])</code>	number	Returns the largest value
<code>least(a: number, ...numbers: number[])</code>	number	Returns the smallest value
<code>log(a: number)</code>	number	Natural log
<code>sqrt(a: number)</code>	number	Square root

Structure types

A map or list is a structure type in CloudWatch Logs Insights that allows you to access and use attributes for queries.

Example: To get a map or list

Use `jsonParse` to parse a field that's a json string into a map or a list.

```
fields jsonParse(@message) as json_message
```

Example: To access attributes

Use the dot access operator (`map.attribute`) to access items in a map.. If an attribute in a map contains special characters, use backticks to enclose the attribute name (`map.attributes.`special.char``).

```
fields jsonParse(@message) as json_message
| stats count() by json_message.status_code
```

Use the bracket access operator (`list[index]`) to retrieve an item at a specific position within the list.

```
fields jsonParse(@message) as json_message
| filter json_message.users[1].action = "PutData"
```

Wrap special characters in backticks (``) when special characters are present in the key name.

```
fields jsonParse(@message) as json_message
| filter json_message.`user.id` = "123"
```

Example: empty results

Maps and lists are treated as null for string, number, and datetime functions.

```
fields jsonParse(@message) as json_message
| display toupper(json_message)
```

Comparing map and list to any other fields result in false.

Note

Using map and list in dedup,pattern, sort, and stats isn't supported.

Datetime functions

Datetime functions

Use datetime functions in the `fields` and `filter` commands and as arguments for other functions. Use these functions to create time buckets for queries with aggregate functions. Use time periods that consist of a number and one of the following:

- ms for milliseconds
- s for seconds
- m for minutes
- h for hours

For example, 10m is 10 minutes, and 1h is 1 hour.

Note

Use the most appropriate time unit for your datetime function. CloudWatch Logs caps your request according to the time unit that you choose. For example, it caps 60 as the maximum value for any request that uses s. So, if you specify `bin(300s)`, CloudWatch Logs actually implements this as 60 seconds, because 60 is the number of seconds in a minute so CloudWatch Logs won't use a number higher than 60 with s. To create a 5-minute bucket, use `bin(5m)` instead.

The cap for ms is 1000, the caps for s and m are 60, and the cap for h is 24.

The following table contains a list of the different datetime functions that you can use in query commands. The table lists each function's result type and contains a description of each function.

Tip

When you create a query command, you can use the time interval selector to select a time period that you want to query. For example, you can set a time period between 5 and 30-minute intervals; 1, 3, and 12-hour intervals; or a custom time frame. You also can set time periods between specific dates.

Function	Result type	Description
<code>bin(period: Period)</code>	Timestamp	Rounds the value of <code>@timestamp</code> to the given time period and then truncates. For example, <code>bin(5m)</code> rounds the value of <code>@timestamp</code> to the nearest 5 minutes.

Function	Result type	Description
		You can use this to group multiple log entries together in a query. The following example returns the count of exceptions per hour: <pre>filter @message like /Exception/ stats count(*) as exceptionCount by bin(1h) sort exceptionCount desc</pre> The following time units and abbreviations are supported with the <code>bin</code> function. For all units and abbreviations that include more than one character, adding <code>s</code> to pluralize is supported. So both <code>hr</code> and <code>hrs</code> work to specify hours. • millisecond <code>ms msec</code> • second <code>s sec</code> • minute <code>m min</code> • hour <code>h hr</code> • day <code>d</code> • week <code>w</code> • month <code>mo mon</code> • quarter <code>q qtr</code> • year <code>y yr</code>
<code>datefloor(timestamp: Timestamp, period: Period)</code>	Timestamp	Truncates the timestamp to the given period. For example, <code>datefloor(@timestamp, 1h)</code> truncates all values of <code>@timestamp</code> to the bottom of the hour.
<code>dateceil(timestamp : Timestamp, period: Period)</code>	Timestamp	Rounds up the timestamp to the given period and then truncates. For example, <code>dateceil(@timestamp, 1h)</code> truncates all values of <code>@timestamp</code> to the top of the hour.

Function	Result type	Description
<code>fromMillis(fieldName: number)</code>	Timestamp	Interprets the input field as the number of milliseconds since the Unix epoch and converts it to a timestamp.
<code>toMillis(fieldName: Timestamp)</code>	number	Converts the timestamp found in the named field into a number representing the milliseconds since the Unix epoch. For example, <code>toMillis(@timestamp)</code> converts the timestamp <code>2022-01-14T13:18:03.000-08:00</code> to <code>1642195111000</code> .
<code>now()</code>	number	Returns the time that the query processing was started, in epoch seconds. This function takes no arguments. You can use this to filter your query results according to the current time. For example, the following query returns all 4xx errors from the past two hours: <pre> parse @message "Status Code: *;" as statusCode\n filter statusCode >= 400 and statusCode <= 499 \n filter toMillis(@timestamp) >= (now() * 1000 - 7200000) </pre> The following example returns all log entries from the past five hours that contain either the word <code>error</code> or <code>failure</code> <pre> fields @timestamp, @message filter @message like /(?!)(error failure)/ filter toMillis(@timestamp) >= (now() * 1000 - 18000000) </pre>

Note

Currently, CloudWatch Logs Insights doesn't support filtering logs with human readable timestamps.

General functions

General functions

Use general functions in the `fields` and `filter` commands and as arguments for other functions.

Function	Result type	Description
<code>ispresent(fieldName: LogField)</code>	Boolean	Returns true if the field exists
<code>coalesce(fieldName: LogField, ...fieldNames: LogField[])</code>	LogField	Returns the first non-null value from the list

JSON functions

JSON functions

Use JSON functions in the `fields` and `filter` commands and as arguments for other functions.

Function	Result type	Description
<code>jsonParse(fieldName: string)</code>	Map List Empty	Returns a map or list when the input is a string representation of JSON object or a JSON array. Returns an empty value, if the input is not one of the representation.
<code>jsonStringify(fieldName: Map List)</code>	String	Returns a JSON string from a map or list data.

IP address string functions

IP address string functions

Use IP address string functions in the `filter` and `fields` commands and as arguments for other functions.

Function	Result type	Description
<code>isValidIp(fieldName: string)</code>	boolean	Returns <code>true</code> if the field is a valid IPv4 or IPv6 address.
<code>isValidIPv4(fieldName: string)</code>	boolean	Returns <code>true</code> if the field is a valid IPv4 address.
<code>isValidIPv6(fieldName: string)</code>	boolean	Returns <code>true</code> if the field is a valid IPv6 address.
<code>isIpInSubnet(fieldName: string, subnet: string)</code>	boolean	Returns <code>true</code> if the field is a valid IPv4 or IPv6 address within the specified v4 or v6 subnet. When you specify the subnet, use CIDR notation such as <code>192.0.2.0/24</code> or <code>2001:db8::/32</code> , where <code>192.0.2.0</code> or <code>2001:db8::</code> is the start of the CIDR block.
<code>isIPv4InSubnet(fieldName: string, subnet: string)</code>	boolean	Returns <code>true</code> if the field is a valid IPv4 address within the specified v4 subnet. When you specify the subnet, use CIDR notation such as <code>192.0.2.0/24</code> where <code>192.0.2.0</code> is the start of the CIDR block..
<code>isIPv6InSubnet(fieldName: string, subnet: string)</code>	boolean	Returns <code>true</code> if the field is a valid IPv6 address within the specified v6 subnet. When you specify the subnet, use CIDR notation such as <code>2001:db8::/32</code> where <code>2001:db8::</code> is the start of the CIDR block.

String functions

String functions

Use string functions in the `fields` and `filter` commands and as arguments for other functions.

Function	Result type	Description
<code>isempty(fieldName: string)</code>	Number	Returns 1 if the field is missing or is an empty string.
<code>isblank(fieldName: string)</code>	Number	Returns 1 if the field is missing, an empty string, or contains only white space.
<code>concat(str: string, ...strings: string[])</code>	string	Concatenates the strings.
<code>ltrim(str: string)</code> <code>ltrim(str: string, trimChars: string)</code>	string	If the function does not have a second argument, it removes white space from the left of the string. If the function has a second string argument, it does not remove white space. Instead, it removes the characters in <code>trimChars</code> from the left of <code>str</code>. For example, <code>ltrim("xy ZxyfooxyZ", "xyZ")</code> returns "fooxyZ".
<code>rtrim(str: string)</code> <code>rtrim(str: string, trimChars: string)</code>	string	If the function does not have a second argument, it removes white space from the right of the string. If the function has a second string argument, it does not remove white space.

Function	Result type	Description
		Instead, it removes the characters of <code>trimChars</code> from the right of <code>str</code> . For example, <code>rtrim("xyZfooxyxyZ", "xyZ")</code> returns <code>"xyZfoo"</code> .
<code>trim(str: string)</code> <code>trim(str: string, trimChars: string)</code>	string	If the function does not have a second argument, it removes white space from both ends of the string. If the function has a second string argument, it does not remove white space. Instead, it removes the characters of <code>trimChars</code> from both sides of <code>str</code> . For example, <code>trim("xyZxyfooxyxyZ", "xyZ")</code> returns <code>"foo"</code> .
<code>strlen(str: string)</code>	number	Returns the length of the string in Unicode code points.
<code>toupper(str: string)</code>	string	Converts the string to uppercase.
<code>tolower(str: string)</code>	string	Converts the string to lowercase.

Function	Result type	Description
<code>substr(str: string, startIndex: number)</code> <code>substr(str: string, startIndex: number, length: number)</code>	string	Returns a substring from the index specified by the number argument to the end of the string. If the function has a second number argument, it contains the length of the substring to be retrieved. For example, <code>substr("xyzfooxyz", 3, 3)</code> returns "foo".
<code>replace(fieldName: string, searchValue: string, replaceValue: string)</code>	string	Replaces all instances of <code>searchValue</code> in <code>fieldName: string</code> with <code>replaceValue</code> . For example, the function <code>replace(logGroup, "smoke_test", "Smoke")</code> searches for log events where the field <code>logGroup</code> contains the string value <code>smoke_test</code> and replaces the value with the string <code>Smoke</code>.
<code>strcontains(str: string, searchValue: string)</code>	number	Returns 1 if <code>str</code> contains <code>searchValue</code> and 0 otherwise.

Fields that contain special characters

If a field contains non-alphanumeric characters other than the @ symbol or the period (.), you must surround the field with backtick characters (`). For example, the log field `foo-bar` must be

enclosed in backticks (``foo-bar``) because it contains a non-alphanumeric character, the hyphen (-).

Use aliases and comments in queries

Create queries that contain aliases. Use aliases to rename log fields or when extracting values into fields. Use the keyword `as` to give a log field or result an alias. You can use more than one alias in a query. You can use aliases in the following commands:

- `fields`
- `parse`
- `sort`
- `stats`

The following examples show how to create queries that contain aliases.

Example

The query contains an alias in the `fields` command.

```
fields @timestamp, @message, accountId as ID
| sort @timestamp desc
| limit 20
```

The query returns the values for the fields `@timestamp`, `@message`, and `accountId`. The results are sorted in descending order and limited to 20. The values for `accountId` are listed under the alias `ID`.

Example

The query contains aliases in the `sort` and `stats` commands.

```
stats count(*) by duration as time
| sort time desc
```

The query counts the number of times the field `duration` occurs in the log group and sorts the results in descending order. The values for `duration` are listed under the alias `time`.

Use comments

CloudWatch Logs Insights supports comments in queries. Use the hash character (#) to set off comments. You can use comments to ignore lines in queries or document queries.

Example: Query

When the following query is run, the second line is ignored.

```
fields @timestamp, @message, accountId
# | filter accountId not like "7983124201998"
| sort @timestamp desc
| limit 20
```

Get started with Logs Insights QL: Query tutorials

The following sections include sample query tutorials to help you get started with Logs Insights QL.

Topics

- [Tutorial: Run and modify a sample query](#)
- [Tutorial: Run a query with an aggregation function](#)
- [Tutorial: Run a query that produces a visualization grouped by log fields](#)
- [Tutorial: Run a query that produces a time series visualization](#)

Tutorial: Run and modify a sample query

The following tutorial helps you get started with CloudWatch Logs Insights. You run a sample query in Logs Insights QL, and then see how to modify and rerun it.

To run a query, you must already have logs stored in CloudWatch Logs. If you are already using CloudWatch Logs and have log groups and log streams set up, you are ready to start. You may also already have logs if you use services such as AWS CloudTrail, Amazon Route 53, or Amazon VPC and you have set up logs from those services to go to CloudWatch Logs. For more information about sending logs to CloudWatch Logs, see [Getting started with CloudWatch Logs](#).

Queries in CloudWatch Logs Insights return either a set of fields from log events or the result of a mathematical aggregation or other operation performed on log events. This tutorial demonstrates a query that returns a list of log events.

Run a sample query

To run a CloudWatch Logs Insights sample query

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.

On the **Logs Insights** page, the query editor contains a default query in Logs Insights QL that returns the 20 most recent log events.

3. In the **Select log group(s)** drop down, choose one or more log groups to query.

If this is a monitoring account in CloudWatch cross-account observability, you can select log groups in the source accounts as well as the monitoring account. A single query can query logs from different accounts at once.

You can filter the log groups by log group name, account ID, or account label.

When you select a log group in the Standard log class, CloudWatch Logs Insights automatically detects data fields in the group. To see discovered fields, select the **Fields** menu near the top right of the page.

Note

Discovered fields is supported only for log groups in the Standard log class. For more information about log classes, see [Log classes](#).

4. (Optional) Use the time interval selector to select a time period that you want to query.

You can choose between 5 and 30-minute intervals; 1, 3, and 12-hour intervals; or a custom time frame.

5. Choose **Run** to view the results.

For this tutorial, the results include the 20 most recently added log events.

CloudWatch Logs displays a bar graph of log events in the log group over time. The bar graph shows not only the events in the table, but also the distribution of events in the log group that match the query and timeframe.

6. To see all fields for a returned log event, choose the triangular dropdown icon left of the numbered event.

Modify the sample query

In this tutorial, you modify the sample query to show the 50 most recent log events.

If you haven't already run the previous tutorial, do that now. This tutorial starts where that previous tutorial ends.

Note

Some sample queries provided with CloudWatch Logs Insights use `head` or `tail` commands instead of `limit`. These commands are being deprecated and have been replaced with `limit`. Use `limit` instead of `head` or `tail` in all queries that you write.

To modify the CloudWatch Logs Insights sample query

1. In the query editor, change **20** to **50**, and then choose **Run**.

The results of the new query appear. Assuming there is enough data in the log group in the default time range, there are now 50 log events listed.

2. (Optional) You can save queries that you have created. To save this query, choose **Save**. For more information, see [Save and re-run CloudWatch Logs Insights queries](#).

Add a filter command to the sample query

This tutorial shows how to make a more powerful change to the query in the query editor. In this tutorial, you filter the results of the previous query based on a field in the retrieved log events.

If you haven't already run the previous tutorials, do that now. This tutorial starts where that previous tutorial ends.

To add a filter command to the previous query

1. Decide on a field to filter. To see the most common fields that CloudWatch Logs has detected in the log events contained in the selected log groups in the past 15 minutes, and the percentage of those log events in which each field appears, select **Fields** on the right side of the page.

To see the fields contained in a particular log event, choose the icon to the left of that row.

The **awsRegion** field might appear in your log event, depending on which events are in your logs. For the rest of this tutorial, we use **awsRegion** as the filter field, but you can use a different field if that field isn't available.

2. In the query editor box, place your cursor after **50** and press Enter.
3. On the new line, first enter | (the pipe character) and a space. Commands in a CloudWatch Logs Insights query must be separated by the pipe character.
4. Enter **filter awsRegion="us-east-1"**.
5. Choose **Run**.

The query runs again, and now displays the 50 most recent results that match the new filter.

If you filtered on a different field and got an error result, you might need to escape the field name. If the field name includes non-alphanumeric characters, you must put backtick characters (```) before and after the field name (for example, ``error-code`="102"`).

You must use the backtick characters for field names that contain non-alphanumeric characters, but not for values. Values are always contained in quotation marks (").

Logs Insights QL includes powerful query abilities, including several commands and support for regular expressions, mathematical, and statistical operations. For more information, see [CloudWatch Logs Insights language query syntax](#).

Tutorial: Run a query with an aggregation function

You can use aggregation functions with the `stats` command and as arguments for other functions. In this tutorial, you run a query command that counts the number of log events containing a specified field. The query command returns a total count that's grouped by the specified field's value or values. For more information about aggregation functions, see [Supported operations and functions](#) in the *Amazon CloudWatch Logs User Guide*.

To run a query with an aggregation function

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. Confirm that the **Logs Insights QL** tab is selected.
4. In the **Select log group(s)** drop down, choose one or more log groups to query.

If this is a monitoring account in CloudWatch cross-account observability, you can select log groups in the source accounts as well as the monitoring account. A single query can query logs from different accounts at once.

You can filter the log groups by log group name, account ID, or account label.

When you select a log group, CloudWatch Logs Insights automatically detects data fields in the log group if it is a Standard class log group. To see discovered fields, select the **Fields** menu near the top right of the page.

5. Delete the default query in the query editor, and enter the following command:

```
stats count(*) by fieldName
```

6. Replace *fieldName* with a discovered field from the **Fields** menu.

The **Fields** menu is located at the top right of the page and displays all of the discovered fields that CloudWatch Logs Insights detects in your log group.

7. Choose **Run** to view the query results.

The query results show the number of records in your log group that match the query command and the total count that's grouped by the specified field's value or values.

Tutorial: Run a query that produces a visualization grouped by log fields

When you run a query that uses the `stats` function to group the returned results by the values of one or more fields in the log entries, you can view the results as a bar chart, pie chart, line graph or stacked area graph. This helps you more efficiently visualize trends in your logs.

To run a query for visualization

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. In the **Select log group(s)** drop down, choose one or more log groups to query.

If this is a monitoring account in CloudWatch cross-account observability, you can select log groups in the source accounts as well as the monitoring account. A single query can query logs from different accounts at once.

You can filter the log groups by log group name, account ID, or account label.

4. In the query editor, delete the current contents, enter the following stats function, and then choose **Run query**.

```
stats count(*) by @logStream
| limit 100
```

The results show the number of log events in the log group for each log stream. The results are limited to only 100 rows.

5. Choose the **Visualization** tab.
6. Select the arrow next to **Line**, and then choose **Bar**.

The bar chart appears, showing a bar for each log stream in the log group.

Tutorial: Run a query that produces a time series visualization

When you run a query that uses the `bin()` function to group the returned results by a time period, you can view the results as a line graph, stacked area graph, pie chart, or bar chart. This helps you more efficiently visualize trends in log events over time.

To run a query for visualization

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. Confirm that the **Logs Insights QL** tab is selected.
4. In the **Select log group(s)** drop down, choose one or more log groups to query.

If this is a monitoring account in CloudWatch cross-account observability, you can select log groups in the source accounts as well as the monitoring account. A single query can query logs from different accounts at once.

You can filter the log groups by log group name, account ID, or account label.

5. In the query editor, delete the current contents, enter the following stats function, and then choose **Run query**.

```
stats count(*) by bin(30s)
```

The results show the number of log events in the log group that were received by CloudWatch Logs for each 30-second period.

6. Choose the **Visualization** tab.

The results are shown as a line graph. To switch to a bar chart, pie chart, or stacked area chart, choose the arrow next to **Line** at the upper left of the graph.

Sample queries

This section contains a list of general and useful query commands that you can run in the [CloudWatch console](#). For information about how to run a query command, see [Tutorial: Run and modify a sample query](#) in the *Amazon CloudWatch Logs User Guide*.

For more information about query syntax, see [CloudWatch Logs Insights language query syntax](#).

Topics

- [General queries](#)
- [Queries for Lambda logs](#)
- [Queries for Amazon VPC flow logs](#)
- [Queries for Route 53 logs](#)
- [Queries for CloudTrail logs](#)
- [Queries for Amazon API Gateway](#)
- [Queries for NAT gateway](#)
- [Queries for Apache server logs](#)
- [Queries for Amazon EventBridge](#)
- [Examples of the parse command](#)

General queries

Find the 25 most recently added log events.

```
fields @timestamp, @message | sort @timestamp desc | limit 25
```

Get a list of the number of exceptions per hour.

```
filter @message like /Exception/  
  | stats count(*) as exceptionCount by bin(1h)  
  | sort exceptionCount desc
```

Get a list of log events that aren't exceptions.

```
fields @message | filter @message not like /Exception/
```

Get the most recent log event for each unique value of the server field.

```
fields @timestamp, server, severity, message  
  | sort @timestamp asc  
  | dedup server
```

Get the most recent log event for each unique value of the server field for each severity type.

```
fields @timestamp, server, severity, message  
  | sort @timestamp desc  
  | dedup server, severity
```

Queries for Lambda logs

Determine the amount of overprovisioned memory.

```
filter @type = "REPORT"  
  | stats max(@memorySize / 1000 / 1000) as provisionedMemoryMB,  
    min(@maxMemoryUsed / 1000 / 1000) as smallestMemoryRequestMB,  
    avg(@maxMemoryUsed / 1000 / 1000) as avgMemoryUsedMB,  
    max(@maxMemoryUsed / 1000 / 1000) as maxMemoryUsedMB,  
    provisionedMemoryMB - maxMemoryUsedMB as overProvisionedMB
```

Create a latency report.

```
filter @type = "REPORT" |  
  stats avg(@duration), max(@duration), min(@duration) by bin(5m)
```

Search for slow function invocations, and eliminate duplicate requests that can arise from retries or client-side code. In this query, @duration is in milliseconds.

```
fields @timestamp, @requestId, @message, @logStream
| filter @type = "REPORT" and @duration > 1000
| sort @timestamp desc
| dedup @requestId
| limit 20
```

Queries for Amazon VPC flow logs

Find the top 15 packet transfers across hosts:

```
stats sum(packets) as packetsTransferred by srcAddr, dstAddr
| sort packetsTransferred desc
| limit 15
```

Find the top 15 byte transfers for hosts on a given subnet.

```
filter isIpv4InSubnet(srcAddr, "192.0.2.0/24")
| stats sum(bytes) as bytesTransferred by dstAddr
| sort bytesTransferred desc
| limit 15
```

Find the IP addresses that use UDP as a data transfer protocol.

```
filter protocol=17 | stats count(*) by srcAddr
```

Find the IP addresses where flow records were skipped during the capture window.

```
filter logStatus="SKIPDATA"
| stats count(*) by bin(1h) as t
| sort t
```

Find a single record for each connection, to help troubleshoot network connectivity issues.

```
fields @timestamp, srcAddr, dstAddr, srcPort, dstPort, protocol, bytes
```

```
| filter logStream = 'vpc-flow-logs' and interfaceId = 'eni-0123456789abcdef0'  
| sort @timestamp desc  
| dedup srcAddr, dstAddr, srcPort, dstPort, protocol  
| limit 20
```

Queries for Route 53 logs

Find the distribution of records per hour by query type.

```
stats count(*) by queryType, bin(1h)
```

Find the 10 DNS resolvers with the highest number of requests.

```
stats count(*) as numRequests by resolverIp  
  | sort numRequests desc  
  | limit 10
```

Find the number of records by domain and subdomain where the server failed to complete the DNS request.

```
filter responseCode="SERVFAIL" | stats count(*) by queryName
```

Queries for CloudTrail logs

Find the number of log entries for each service, event type, and AWS Region.

```
stats count(*) by eventSource, eventName, awsRegion
```

Find the Amazon EC2 hosts that were started or stopped in a given AWS Region.

```
filter (eventName="StartInstances" or eventName="StopInstances") and awsRegion="us-east-2"
```

Find the AWS Regions, user names, and ARNs of newly created IAM users.

```
filter eventName="CreateUser"  
  | fields awsRegion, requestParameters.userName, responseElements.user.arn
```

Find the number of records where an exception occurred while invoking the API UpdateTrail.

```
filter eventName="UpdateTrail" and ispresent(errorCode)
  | stats count(*) by errorCode, errorMessage
```

Find log entries where TLS 1.0 or 1.1 was used

```
filter tlsDetails.tlsVersion in [ "TLSv1", "TLSv1.1" ]
  | stats count(*) as numOutdatedTlsCalls by userIdentity.accountId, recipientAccountId,
    eventSource, eventName, awsRegion, tlsDetails.tlsVersion, tlsDetails.cipherSuite,
    userAgent
  | sort eventSource, eventName, awsRegion, tlsDetails.tlsVersion
```

Find the number of calls per service that used TLS versions 1.0 or 1.1

```
filter tlsDetails.tlsVersion in [ "TLSv1", "TLSv1.1" ]
  | stats count(*) as numOutdatedTlsCalls by eventSource
  | sort numOutdatedTlsCalls desc
```

Queries for Amazon API Gateway**Find the last 10 4XX errors**

```
fields @timestamp, status, ip, path, httpMethod
  | filter status>=400 and status<=499
  | sort @timestamp desc
  | limit 10
```

Identify the 10 longest-running Amazon API Gateway requests in your Amazon API Gateway access log group

```
fields @timestamp, status, ip, path, httpMethod, responseLatency
  | sort responseLatency desc
  | limit 10
```


Return the list of the most popular API paths in your Amazon API Gateway access log group

```
stats count(*) as requestCount by path
| sort requestCount desc
| limit 10
```

Create an integration latency report for your Amazon API Gateway access log group

```
filter status=200
| stats avg(integrationLatency), max(integrationLatency),
min(integrationLatency) by bin(1m)
```

Queries for NAT gateway

If you notice higher than normal costs in your AWS bill, you can use CloudWatch Logs Insights to find the top contributors. For more information about the following query commands, see [How can I find the top contributors to traffic through the NAT gateway in my VPC?](#) at the AWS premium support page.

Note

In the following query commands, replace "x.x.x.x" with the private IP of your NAT gateway, and replace "y.y" with the first two octets of your VPC CIDR range.

Find the instances that are sending the most traffic through your NAT gateway.

```
filter (dstAddr like 'x.x.x.x' and srcAddr like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

Determine the traffic that's going to and from the instances in your NAT gateways.

```
filter (dstAddr like 'x.x.x.x' and srcAddr like 'y.y.') or (srcAddr like 'xxx.xx.xx.xx'
and dstAddr like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

Determine the internet destinations that the instances in your VPC communicate with most often for uploads and downloads.

For uploads

```
filter (srcAddr like 'x.x.x.x' and dstAddr not like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

For downloads

```
filter (dstAddr like 'x.x.x.x' and srcAddr not like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

Queries for Apache server logs

You can use CloudWatch Logs Insights to query Apache server logs. For more information about the following queries, see [Simplifying Apache server logs with CloudWatch Logs Insights](#) at the AWS Cloud Operations & Migrations Blog.

Find the most relevant fields, so you can review your access logs and check for traffic in the */admin* path of your application.

```
fields @timestamp, remoteIP, request, status, filename| sort @timestamp desc
| filter filename="/var/www/html/admin"
| limit 20
```

Find the number unique GET requests that accessed your main page with status code "200" (success).

```
fields @timestamp, remoteIP, method, status
| filter status="200" and referrer= http://34.250.27.141/ and method= "GET"
| stats count_distinct(remoteIP) as UniqueVisits
| limit 10
```

Find the number of times your Apache service restarted.

```
fields @timestamp, function, process, message
| filter message like "resuming normal operations"
| sort @timestamp desc
| limit 20
```

Queries for Amazon EventBridge

Get the number of EventBridge events grouped by event detail type

```
fields @timestamp, @message
| stats count(*) as numberOfEvents by `detail-type`
| sort numberOfEvents desc
```

Examples of the parse command

Use a glob expression to extract the fields @user, @method, and @latency from the log field @message and return the average latency for each unique combination of @method and @user.

```
parse @message "user=*, method:*, latency := *" as @user,
    @method, @latency | stats avg(@latency) by @method,
    @user
```

Use a regular expression to extract the fields @user2, @method2, and @latency2 from the log field @message and return the average latency for each unique combination of @method2 and @user2.

```
parse @message /user=(?<user2>.*?), method:(?<method2>.*?),
    latency := (?<latency2>.*?)/ | stats avg(latency2) by @method2,
    @user2
```

Extracts the fields loggingTime, loggingType and loggingMessage, filters down to log events that contain ERROR or INFO strings, and then displays only the loggingMessage and loggingType fields for events that contain an ERROR string.

```
FIELDS @message
| PARSE @message "*" [*] "*" as loggingTime, loggingType, loggingMessage
| FILTER loggingType IN ["ERROR", "INFO"]
| DISPLAY loggingMessage, loggingType = "ERROR" as isError
```

Compare (diff) with previous time ranges

You can use CloudWatch Logs Insights with the Logs Insights QL to compare changes in your log events over time. You can compare the log events ingested during a recent time range with the logs from the immediately previous time period. Alternatively, you can compare with similar past time periods. This can help you find whether an error in your logs was recently introduced or was already occurring, and can help you find other trends.

Comparison queries return only patterns in the results, not raw log events. The patterns returned will help you quickly see the trends and changes in the log events over time. After you run a comparison query and have the pattern results, you can see sample raw log events for the patterns that you're interested in. For more information about log patterns, see [Pattern analysis](#).

When you run a comparison query, your query is analyzed against two different time periods: the original query period that you select, and the comparison period. The comparison period is always of equal length to your original query period. The default time intervals for the comparisons are the following.

- **Previous period**— Compares to the time period immediately before your query time period.
- **Previous day**— Compares to the time period one day before your query time period.
- **Previous week**— Compares to the time period one week before your query time period.
- **Previous month**— Compares to the time period one month before your query time period.

Note

Queries using comparisons incur charges similar to running a single CloudWatch Logs Insights query over the combined time range. For more information, see [Amazon CloudWatch Pricing](#).

To run a comparison query

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Logs Insights**.

A default query appears in the query box.

3. Confirm that the **Logs Insights QL** tab is selected.

4. Keep the default query or enter a different query.
5. In the **Select log group(s)** drop-down, choose one or more log groups to query.
6. (Optional) Use the time interval selector to select a time period that you want to query. The default query is for the previous hour of log data.
7. By the time range selector, choose **Compare**. Then choose the previous time period that you want to compare the original logs with, and choose **Apply**.
8. Choose **Run query**.

To cause the query to fetch the data from the comparison period, the `diff` command is appended to your query.

9. Choose the **Patterns** tab to see the results.

The table displays the following information:

- Each **Pattern**, with variable parts of the pattern replaced by the dynamic token symbol `<string-number>`. The *string* is a description of the type of data that the token represents. The *number* shows where in the pattern this token appears, compared to the other dynamic tokens. For more information, see [Pattern analysis](#).
 - **Event count** is the number of log events with that pattern in the original, more current time period.
 - **Difference event count** is the difference between the number of matching log events in the current time period versus the comparison time period. A positive difference means there are more such events in the current time period.
 - **Difference description** briefly summarizes the change in that pattern between the current time period and the comparison period.
 - **Severity type** is the probable severity of the logs events with this pattern, based on words found in the log events such as FATAL, ERROR, and WARN.
10. To further inspect one of the patterns in the list, choose the icon in the **Inspect** column for one of the patterns.

The **Pattern inspect** pane appears and displays the following:

- The **Pattern**. Select a token within the pattern to analyze that token's values.
- A histogram showing the number of occurrences of the pattern over the queried time range. This can help you to identify interesting trends such as a sudden increase in occurrence of a pattern.

- The **Log samples** tab displays a few of the log events that match the selected pattern.
- The **Token Values** tab displays the values of the selected dynamic token, if you have selected one.

 Note

A maximum of 10 token values is captured for each token. Token counts might not be precise. CloudWatch Logs uses a probabilistic counter to generate the token count, not the absolute value.

- The **Related patterns** tab displays other patterns that frequently occurred near the same time as the pattern that you are inspecting. For example, if a pattern for an ERROR message was usually accompanied by another log event marked as INFO with additional details, that pattern is displayed here.

Visualize log data in graphs

You can use visualizations such as bar charts, line charts, and stacked area charts to more efficiently identify patterns in your log data. CloudWatch Logs Insights generates visualizations for queries that use the `stats` function and one or more aggregation functions. For more information, see [stats](#).

OpenSearch Piped Processing Language (PPL)

This section contains a basic introduction to querying CloudWatch Logs using OpenSearch PPL. With PPL, you can retrieve, query, and analyze data using piped-together commands, making it easier to understand and compose complex queries. Its syntax is based on Unix pipes, and enables chaining of commands to transform and process data. With PPL, you can filter and aggregate data, and use a rich set of math, string, date, conditional, and other functions for analysis.

Including `SOURCE` in a PPL query is a useful way to specify the log groups field indexes, and data sources to include in a query when you are using the AWS CLI or API to create a query. The `SOURCE` command is supported only in the AWS CLI and API, not in the CloudWatch console. When you use the CloudWatch console to start a query, you use the console interface to specify the log groups and data source name and type.

Use `aws:fieldIndex` to return indexed data only, by forcing a query to scan only log groups that are indexed on a field that you specify in the query. The relevant log groups are automatically

selected, based on the fields specified in the `filterIndex` command. This reduces scanned volume by skipping log groups that do not have any log events containing the field specified in the query, and only scanning log groups that match the value specified in the query for this field index. Use `aws:fieldIndex` to specify the field name, along with the field name and value in the source command to query only indexed data containing the field and value specified. For more information, see [Create field indexes to improve query performance and reduce scan volume](#)

You can use OpenSearch PPL for queries of log groups in the Standard Log Class.

Note

For information about all OpenSearch PPL query commands supported in CloudWatch Logs and detailed information about syntax and restrictions, see [Supported PPL commands](#) in the OpenSearch Service Developer Guide.

For information on other query languages you can use see, [CloudWatch Logs Insights](#), [OpenSearch Service SQL](#), and [CloudWatch Metrics Insights](#)

Command or function	Example query	Description
<code>fields</code>	<code>fields field1, field2</code>	Displays a set of fields which needs projection.
<code>join</code>	<code>LEFT JOIN left=l, right=r on l.id = r.id `join_right_lg` fields l.field_1, r.field_2</code>	Joins two datasets together.
<code>where</code>	<code>where field1="success" where field2 != "i-023fe0a90929d8822" fields field3, field4, field5,field6 head 1000</code>	Filters the data based on the conditions that you specify.
<code>aws:fieldIndex</code>	<code>source = [`aws:fieldIndex`="region", `region` = "us-west-2"] where status = 200 head 10</code>	Returns indexed data only, by forcing a query

Command or function	Example query	Description
		to scan only log groups that are indexed on a field that you specify in the query.
stats	<code>stats count(), count(field1), min(field1), max(field1), avg(field1) by field2 head 1000</code>	Performs aggregations and calculations
parse	<code>parse field1 ".*/(?<field2>[^/]+\$)" where field2 = "requestId" fields field1, field2 head 1000</code>	Extracts a regular expression (regex) pattern from a string and displays the extracted pattern. The extracted pattern can be further used to create new fields or filter data.
sort	<code>stats count(), count(field1), min(field1) as field1Alias, max(`field1`), avg(`field1`) by field2 sort -field1Alias head 1000</code>	Sort the displayed results by a field name. Use <code>sort -FieldName</code> to sort in descending order.

Command or function	Example query	Description
eval	<pre>eval field2 = field1 * 2 fields field1, field2 head 20</pre>	Modifies or processes the value of a field and stores it in a different field. This is useful to mathematically modify a column, apply string functions to a column, or apply date functions to a column.
rename	<pre>rename field2 as field1 fields field1;</pre>	Renames one or more fields in the search result.
head	<pre>fields `@message` head 20</pre>	Limits the displayed query results to the first N rows.
top	<pre>top 2 field1 by field2</pre>	Finds the most frequent values for a field.
dedup	<pre>dedup field1 fields field1, field2, field3</pre>	Removes duplicate entries based on the fields that you specify.

Command or function	Example query	Description
rare	<code>rare field1 by field2</code>	Finds the least frequent values of all fields in the field list.
subquery	<code>where field_1 IN [search source=`subquery_lg` fields field_2] fields id, field_1</code>	Performs complex, nested queries within your PPL statements.
trendline	<code>trendline sma(2, field1) as field1Alias</code>	Calculates the moving averages of fields.
eventStats	<code>eventstats sum(field1) by field2</code>	Enriches your event data with calculated summary statistics. It analyzes specified fields within your events, computes various statistical measures, and then appends these results to each original event as new fields.

Command or function	Example query	Description
expand	<pre>eval tags_array_string = json_extract(`@message`, '\$.tags') eval tags_array = json_array(json_extract(tags_string, '\$[0]'), json_extract(tags_string, '\$[1]')) expand tags_array as color_tags</pre>	Breaks down a field containing multiple values into separate rows, creating a new row for each value in the specified field.
fillnull	<pre>fields `@timestamp`, error_code, status_code fillnull using status_code = "UNKNOWN", error_code = "UNKNOWN"</pre>	Fills null fields with the value that you provide. It can be used in one or more fields.
flatten	<pre>eval metadata_struct = json_object('size', json_extract(metadata_string, '\$.size'), 'color', json_extract(metadata_string, '\$.color')) flatten metadata_struct as (meta_size, meta_color)</pre>	Flattens a field. The field must be of this type: struct<?,?> or array<struct<?,?>> .
cidrmatch	<pre>where cidrmatch(ip, '2003:db8::/32') fields ip</pre>	Checks if the specified IP address is within the given CIDR range.

Command or function	Example query	Description
fieldsummary	<code>where field1 != 200 fieldsummary includefields= field1 nulls=true</code>	Calculates basic statistics for each field (count, distinct count, min, max, avg, stddev, and mean).
grok	<code>grok email '.*@%{HOSTNAME:host}' fields email, host</code>	Parses a text field with a grok pattern and appends the results to the search result.
String functions	<code>eval field1Len = LENGTH(field1) fields field1Len</code>	Built-in functions in PPL that can manipulate and transform string and text data within PPL queries. For example, converting case, combining strings, extracting parts, and cleaning text.

Command or function	Example query	Description
Date-Time functions	<code>eval newDate = ADDDATE(DATE('2020-08-26'), 1) fields newDate</code>	Built-in functions for handling and transforming date and timestamp data in PPL queries. For example, <code>date_add</code> , <code>date_format</code> , <code>datediff</code> , <code>date_sub</code> , <code>timestamp add</code> , <code>timestamp diff</code> , <code>current_timezone</code> , <code>utc_timestamp</code> , and <code>current_date</code> .
Condition functions	<code>eval field2 = isnull(field1) fields field2, field1, field3</code>	Built-in functions that check for specific field conditions, and evaluate expressions conditionally. For example, if <code>field1</code> is null, return <code>field2</code> .

Command or function	Example query	Description
Math functions	<code>eval field2 = ACOS(field1) fields field1</code>	Built-in functions for performing mathematical calculations and transformations in PPL queries. For example, <code>abs</code> (absolute value), <code>round</code> (rounds numbers), <code>sqrt</code> (square root), <code>pow</code> (power calculation), and <code>ceil</code> (rounds up to nearest integer).
CryptoGraphic functions	<code>eval crypto = MD5(field) head 1000</code>	To calculate the hash of given field

Command or function	Example query	Description
JSON functions	<pre>eval valid_json = json('[1,2,3,{"f1":1,"f2":[5,6]},4]') fields valid_json</pre>	Built-in functions for handling JSON including arrays, extracting, and validation. For example, <code>json_object</code> , <code>json_array</code> , <code>to_json_string</code> , <code>json_array_length</code> , <code>json_extract</code> , <code>json_keys</code> , and <code>json_valid</code> .

Query scope

Including `SOURCE` in a query is a useful way to specify the log groups to include in a query when you are using the AWS CLI or API to create a query. The `SOURCE` command is supported only in the AWS CLI and API, not in the CloudWatch console. When you use the CloudWatch console to start a query, you use the console interface to specify the log groups and data source name and type.

PPL's `source` command now support multiple ways to specify them:

1. Log group
2. Field indexes - New
3. Data source and type - New

Log Group

Log Group source selection can be used when customers know which exact log group(s) need to be searched

```
source = [lg:`/aws/lambda/my-function`] | where status = 200 | head 10
```

Field Indexes

Field index-based source selection reduces the amount of data queried by limiting results to only indexed data when your filters target fields that have been indexed. The relevant log groups are automatically selected, based on the fields specified in the `filterIndex` command. For more information about field indexes and how to create them, see [Create field indexes to improve query performance and reduce scan volume](#).

Use `aws:fieldIndex` to return indexed data only, by forcing a query to scan only log groups that are indexed on a field that you specify in the query. For these log groups that are indexed on this field, it further optimizes the query by skipping the log groups that do not have any log events containing the field specified in the query for the indexed field. It further reduces scanned volume by attempting to scan only log events from these log groups that match the value specified in the query for this field index. For more information about field indexes and how to create them, see [Create field indexes to improve query performance and reduce scan volume](#).

In PPL, `aws:fieldIndex` is used to specify which key value pairs should be treated as indexes. The syntax is as follows

```
source = [aws:fieldIndex`="region", `region` = "us-west-2"] | where status = 200 | head 10
```

where,

1. `aws:fieldIndex`="region"` identifies region as field Index.
 - a. Note: Instead of `=` customers can use `IN` to specify multiple indexes (example below)
2. ``region`="us-west-2"` identifies the filter condition to be applied
 - a. Note: Instead of `=` customers can use `IN` to specify multiple values (example below)

Customers can specify multiple fieldIndexes as follows

```
source = [aws:fieldIndex` IN ("status", "region"), `status` = 200, `region` IN ("us-west-2", "us-east-1")] | head 10
```


Data Source and Type

Data source and type based source selection can be used when customers know which exact data sources need to be queried. This query is executed over one or more log groups which contain the specified data source and type.

```
source = [ds:`data_source.type`] | where status = 200 | head 10
```

Supported PPL for data source queries

To support the use case for querying data sources in PPL, you can use the dynamic source selector clause. Using this syntax, you can query data sources by specifying them in the search command. You can specify up to 10 data sources.

Syntax

```
source=[ds:`DataSource1.Type1`, ds:`DataSource2.Type2`, ...ds:`DataSourceN.TypeN`]
```

Example query

```
search source=[ds:`DataSource1.Type1`, ds:`DataSource2.Type2`] | fields field1, field2
```

Combined example

Customers can specify all the source selection operators in any order & the results would be the intersection of the all the conditions applied.

For example, `/aws/lambda/my-function-1` might contain multiple data source & types including wide variety of indexes, when the following query was ran, the results returned will only have events of source and type `DataSource1.Type1` and matching the criteria of `'status' = 200`.

```
search source=[
  ds:`DataSource1.Type1`,
  lg:`/aws/lambda/my-function-1`,
  `aws:fieldIndex` IN ("status"), `status` = 200
]
```

Restrictions

The following restrictions apply when you use OpenSearch PPL to query in CloudWatch Logs Insights.

- You cannot use join or subquery commands with data source queries.

OpenSearch Structured Query Language (SQL)

This section contains a basic introduction to querying CloudWatch Logs using OpenSearch SQL. It provides a familiar option if you're used to working with relational databases. OpenSearch SQL offers a subset of SQL functionality, making it a good choice for performing ad-hoc queries and data analysis tasks. With OpenSearch SQL, you can use commands such as SELECT, FROM, WHERE, GROUP BY, HAVING, and various other SQL commands and functions. You can execute JOINS across log groups, correlate data across log groups using sub-queries, and use the rich set of JSON, mathematical, string, conditional, and other SQL functions to perform powerful analysis on log and security data.

Use `filterIndex` to return indexed data only, by forcing a query to scan only log groups that are indexed on a field that you specify in the query. Reduce scanned volume by skipping log groups that do not have any log events containing the field specified in the query, and only scanning log groups that match the value specified in the query for this field index. Use `filterIndex` to specify the field name, along with the field name and value to query only indexed data containing the field and value specified.

You can use OpenSearch SQL for queries of log groups in the Standard Log Class. SQL also supports querying using data source name and data source type.

Note

The following table lists the SQL commands and functions supported in CloudWatch Logs. For information about all OpenSearch SQL commands including syntax, see [Supported SQL commands](#) in the OpenSearch Service Developer Guide. For information on other query languages you can use, see [CloudWatch Logs Insights](#), [OpenSearch Service PPL](#), and [CloudWatch Metrics Insights](#).

Supported SQL commands

Note

In the example query column, replace *<logGroup>* as needed depending on which data source you're querying.

Command or function	Example query	Description
SELECT	SELECT `@message`, Operation FROM `LogGroupA`	Displays projected values.
FROM	SELECT `@message`, Operation FROM `LogGroupA`	Built-in clause that specifies the source table(s) or view(s) from which to retrieve data, supporting various types of joins and subqueries.
WHERE	SELECT * FROM `LogGroupA` WHERE Operation = 'x'	Filters log events based on the provided field criteria.
filterIndex	SELECT * FROM `filterIndex('region' = 'us-east-1')` WHERE status = 200 LIMIT 10;	Returns indexed data only, by forcing a query to scan only log groups that are indexed on a field that you

Command or function	Example query	Description
		specify in the query.
GROUP BY	<pre>SELECT `@logStream`, COUNT(*) as log_count FROM `LogGroupA` GROUP BY `@logStream`</pre>	Groups log events based on category and finds the average based on stats.
HAVING	<pre>SELECT `@logStream`, COUNT(*) as log_count FROM `LogGroupA` GROUP BY `@logStream` HAVING log_count > 100</pre>	Filters the results based on grouping conditions.
ORDER BY	<pre>SELECT * FROM `LogGroupA` ORDER BY `@timestamp` DESC</pre>	Orders the results based on fields in the ORDER BY clause. You can sort in either descending or ascending order.
JOIN	<pre>SELECT A.`@message`, B.`@timestamp` FROM `LogGroupA` as A INNER JOIN `LogGroupB` as B ON A.`requestId` = B.`requestId`</pre>	Joins the results for two tables based on common fields. Inner JOIN or Left Outer Join must be specified

Command or function	Example query	Description
LIMIT	Select * from `LogGroupA` limit 10	Limits the displayed query results to the first N rows.
String functions	SELECT upper(Operation) , lower(Operation), Operation FROM `LogGroupA`	Built-in functions in SQL that can manipulate and transform string and text data within SQL queries. For example, converting case, combining strings, extracting parts, and cleaning text.
Date functions	SELECT current_date() as today, date_add(current_date(), 30) as thirty_days_later, last_day(current_date()) as month_end FROM `LogGroupA`	Built-in functions for handling and transforming date and timestamp data in SQL queries. For example, date_add, date_format, datediff, and current_date.

Command or function	Example query	Description
Conditional functions	<code>SELECT Operation, IF(Error > 0, 'High', 'Low') as error_category FROM `LogGroup A`;</code>	Built-in functions that perform actions based on specified conditions, or that evaluate expressions conditionally. For example, CASE and IF.
Aggregate functions	<code>SELECT AVG(bytes) as bytesWritten FROM `LogGroupA`</code>	Built-in functions that perform calculations on multiple rows to produce a single summarized value. For example, SUM, COUNT, AVG, MAX, and MIN.

Command or function	Example query	Description
JSON functions	<code>SELECT get_json_object(json_column, '\$.name') as name FROM `LogGroupA`</code>	Built-in functions for parsing, extracting, modifying, and querying JSON-formatted data within SQL queries (e.g., <code>from_json</code> , <code>to_json</code> , <code>get_json_object</code> , <code>json_tuple</code>) allowing manipulation of JSON structures in datasets.
Array functions	<code>SELECT scores, size(scores) as length, array_contains(scores, 90) as has_90 FROM `LogGroupA`;</code>	Built-in functions for working with array-type columns in SQL queries, allowing operations like accessing, modifying, and analyzing array data (e.g., <code>size</code> , <code>explode</code> , <code>array_contains</code>).

Command or function	Example query	Description
Window functions	<pre>SELECT field1, field2, RANK() OVER (ORDER BY field2 DESC) as field2Rank FROM `LogGroupA`;</pre>	Built-in functions that perform calculations across a specified set of rows related to the current row (window), enabling operations like ranking, running totals, and moving averages. For example, ROW_NUMBER, RANK, LAG, and LEAD

Command or function	Example query	Description
Conversion functions	<pre>SELECT CAST('123' AS INT) as converted_number, CAST(123 AS STRING) as converted_string FROM `LogGroupA`</pre>	Built-in functions for converting data from one type to another within SQL queries, enabling data type transformations and format conversions. For example, CAST, TO_DATE, TO_TIMESTAMP, and BINARY.
Predicate functions	<pre>SELECT scores, size(scores) as length, array_contains(scores, 90) as has_90 FROM `LogGroupA`;</pre>	Built-in functions that evaluate conditions and return boolean values (true/false) based on specified criteria or patterns. For example, IN, LIKE, BETWEEN, IS NULL, and EXISTS.
Select multiple log groups	<pre>SELECT lg1.field1, lg1.field2 from `logGroups( logGroupIdentifier: ['LogGroup1', 'LogGroup2'])` as lg1 where lg1.field3= "Success"</pre>	Enables you to specify multiple log groups in a SELECT statement

Command or function	Example query	Description
Select multiple data sources	<pre>SELECT ds1.field1, ds1.field2 from `dataSource(['DataSource1', 'DataSour ce2'])` as ds1 where ds1.field3= "Success"</pre>	Enables you to specify multiple data sources in a SELECT statement

Supported SQL for multi-log-group queries

To support the use case for querying multiple log groups in SQL, you can use the `logGroups` command. Using this syntax, you can query multiple log groups by specifying them in the `FROM` command.

Syntax:

```
`logGroups(
  logGroupIdentifier: ['LogGroup1', 'LogGroup2', ... 'LogGroupn']
)
```

In this syntax, you can specify up to 50 log groups in the `logGroupIdentifier` parameter. To reference log groups in a monitoring account, use ARNs instead of LogGroup names.

Example query:

```
SELECT LG1.Column1, LG1.Column2 from `logGroups(
  logGroupIdentifier: ['LogGroup1', 'LogGroup2']
)` as LG1 WHERE LG1.Column1 = 'ABC'
```

The following syntax involving multiple log groups after the `FROM` statement is NOT supported when querying CloudWatch Logs.

```
SELECT Column1, Column2 FROM 'LogGroup1', 'LogGroup2', ... 'LogGroupn'
WHERE Column1 = 'ABC'
```

Supported SQL for data source queries

To support the use case for querying data sources in SQL, you can use the `dataSource` command. Using this syntax, you can query data sources by specifying them in the `FROM` command. You can specify up to 10 data sources.

Syntax

```
`dataSource(  
    ['DataSource1', 'DataSource2', ...'DataSourcen']  
)`
```

Example query

```
SELECT DS1.Column1, DS1.Column2 from `dataSource(  
    ['DataSource1', 'DataSource2']  
)` as DS1 WHERE DS1.Column1 = 'ABC'
```

Query scope

In the AWS CLI and API, you can specify which logs to query by using the log group, data source and type, and field indexes.

Log Group

Log Group source selection can be used when customers know which exact log group(s) need to be searched

```
SELECT * FROM `logGroups(logGroupIdentifier: ['/aws/lambda/my-function'])`;
```

Data Source and Type

Customers can query their logs using data source name and data source type.

Data source and type based source selection can be used when customers know which exact data sources need to be queried. This query is executed over one or more log groups which contain the specified data source and type.

To support the use case for querying data sources in SQL, you can use the `dataSource` command. Using this syntax, you can query data sources by specifying them in the `FROM` command. You can specify up to 10 data sources.

Syntax:

```
`dataSource(  
    ['DataSource1.Type1', 'DataSource2.Type2', ... 'DataSourceN.TypeN']  
)`
```

Example query:

```
SELECT DS1.Column1, DS1.Column2 from `dataSource(  
    ['DataSource1.Type1', 'DataSource2.Type2']  
)` as DS1 WHERE DS1.Column1 = 'ABC'
```

For more information on querying by data sources, see [Use facets to group and explore logs](#).

Combined example

Customers can specify all the source selection operators within the backticks in any order and the results would be based on the intersection of the all the conditions applied.

For example, `/aws/lambda/my-function-1` might contain multiple data source & types including wide variety of indexes, when the following query was ran, the results returned will only have events of source and type `DataSource1.Type1` and matching the criteria of `'status' = 200`.

```
SELECT * FROM `  
    logGroups(logGroupIdentifier: ['/aws/lambda/my-function'])  
    filterIndex('status' = 200)  
    dataSource(['DataSource1.Type1'])  
`;
```

Field Indexes

Field Index-based source selection automatically identifies relevant log groups when your filters target indexed fields, reducing scan volume and query runtime.

Use `filterIndex` to return indexed data only, by forcing a query to scan only log groups that are indexed on a field that you specify in the query. For these log groups that are indexed on this field, it further optimizes the query by skipping the log groups that do not have any log events containing the field specified in the query for the indexed field. It further reduces scanned volume by attempting to scan only log events from these log groups that match the value specified in the

query for this field index. For more information about field indexes and how to create them, see [Create field indexes to improve query performance and reduce scan volume](#).

In SQL, `filterIndex` is used to specify which key value pairs should be treated as indexes. The syntax is as follows

```
SELECT * FROM `filterIndex('region' = 'us-east-1')`;
```

where,

1. `filterIndex(...)` specifies, treat the key values within them as field indexes. Each key value pair is separated by a comma (example below)
2. `'region' = 'us-east-1'` specifies the actual condition to be applied
 - a. Note: Instead of `=` customers can use `IN` to specify multiple values (example below)

Using multiple `filterIndex` would combine the conditions using "AND". In the example, logs matching `status=200` and region in `us-east-1` or `us-west-2` would be queried.

```
SELECT * FROM `filterIndex('status' = 200, 'region' IN ['us-east-1', 'us-west-2'])`;
```

Restrictions

The following restrictions apply when you use OpenSearch SQL to query in CloudWatch Logs Insights.

- You can include only one JOIN in a SELECT statement.
- You cannot use JOIN or subqueries with data source queries.
- Only one level of nested subqueries is supported.
- Multiple statement queries separated by semi-colons (;) aren't supported.
- Queries containing field names that are identical but differ only in case (such as `field1` and `FIELD1`) are not supported.

For example, the following query isn't supported:

```
Select AWSAccountId, AwsAccountId from LogGroup
```

However, the following query is supported because the field name (@logStream) is identical in both log groups:

```
Select a.`@logStream`, b.`@logStream` from Table A INNER Join Table B on a.id = b.id
```

- Functions and expressions must operate on field names and be part of a SELECT statement with a log group specified in the FROM clause.

For example, this query is not supported:

```
SELECT cos(10) FROM LogGroup
```

This query is supported:

```
SELECT cos(field1) FROM LogGroup
```

- When using SQL or PPL commands, enclose certain fields in backticks to successfully query them. Backticks are necessary for fields with special characters (non-alphabetic and non-numeric). For example, enclose @message, Operation.Export, and Test::Field in backticks. You don't need to enclose fields with purely alphabetic names in backticks.

Example query with simple fields:

```
SELECT SessionToken, Operation, StartTime FROM `LogGroup-A`  
LIMIT 1000;
```

Similar query with backticks appended:

```
SELECT `@SessionToken`, `@Operation`, `@StartTime` FROM `LogGroup-A` LIMIT 1000;
```

Use natural language to generate and update CloudWatch Logs Insights queries

CloudWatch Logs supports a natural language query capability to help you generate and update queries for [CloudWatch Logs Insights](#), [OpenSearch Service PPL](#), [OpenSearch Service SQL](#), and [CloudWatch Metrics Insights](#).

With this capability, you can ask questions about or describe the CloudWatch Logs data you're looking for in plain English. The natural language capability generates a query based on a prompt that you enter and provides a line-by-line explanation of how the query works. You can also update your query to further investigate your data.

Depending on your environment, you can enter prompts like "What are the top 100 source IP addresses by bytes transferred?" and "Find the 10 slowest Lambda function requests."

Note

The natural-language query feature is a Regional service. For some Regions, the feature makes cross-Region calls to Regions in the United States to process the query prompts. For more information, see [Amazon CloudWatch expands region support for natural language query result summarization and query generation](#).

To generate a CloudWatch Logs Insights query with this capability, open the CloudWatch Logs Insights query editor, select the log group you want to query, and choose **Generate query**.

Important

To use the natural language query capability, you must be signed in with the [CloudWatchLogsFullAccess](#), [CloudWatchLogsReadOnlyAccess](#), [AdministratorAccess](#), or [ReadOnlyAccess](#) IAM policies, or have the `cloudwatch:GenerateQuery` permission.

Example queries

The examples in this section describe how to generate and update queries using the natural language capability.

Note

For more information on the CloudWatch Logs Insights query editor and syntax, see [CloudWatch Logs Insights query syntax](#).

Examples: Generate a natural language query

To generate a query using natural language, enter a prompt and choose **Generate new query**. These example shows queries that perform a basic search.

Prompt

The following is an example of a prompt that directs the capability to search for the 10 slowest Lambda function invocations.

```
Find the 10 slowest requests
```

Query

The following is the query using the CloudWatch Logs Insights query language that the natural language capability generated based on the prompt. Notice how the prompt appears in a comment before the query. After the query, you can read an explanation that describes how the query works.

```
# Find the 10 slowest requests
fields @timestamp, @message, @duration
| sort @duration desc
| limit 10
# This query retrieves the timestamp, message and duration fields from the logs and
sorts them in descending order by duration to find the 10 slowest requests.
```

Note

To turn off the appearance of your prompt and the explanation of how the query works, use the gear icon in your editor.

Prompt

To generate an OpenSearch SQL query, select the OpenSearch SQL tab, then open the query generator prompt box to enter your natural language prompt. The following is an example of a prompt that uses the natural language capability to generate an OpenSearch SQL query.

```
Give me the number of errors and exceptions per hour
```


Query

The following is the SQL query generated by that prompt that you can use to find the number of errors and exceptions aggregated per hour:

```
SELECT DATE_FORMAT(`@timestamp`, 'yyyy-MM-dd HH') AS hour,  
       COUNT(*) AS error_count  
FROM `/aws/lambda/CloudWatchOdysseyQueryGen`  
WHERE `@message` LIKE '%error%'  
      OR `@message` LIKE '%exception%'  
GROUP BY DATE_FORMAT(`@timestamp`, 'yyyy-MM-dd HH')  
ORDER BY hour
```

Prompt

To generate an OpenSearch PPL query, select the OpenSearch PPL tab, then open the query generator prompt box to enter your natural language prompt. The following is an example of a prompt that uses the natural language capability to generate an OpenSearch PPL query.

```
Give me all unique exception messages
```

Query

The following is the PPL query generated by that prompt that you can use to find the unique exception messages in your logs:

```
dedup @message  
| fields @message
```

Example: Update a natural language query

You can update a query by editing the initial prompt and then choosing **Update query**.

Updated prompt

The following example shows an updated version of the previous prompt. Instead of a prompt that searches for the 10 slowest Lambda function invocations, this prompt now directs the capability to

search for the 20 slowest Lambda function invocations and include another column for additional log events.

Show top 20 slowest requests instead and display requestId as a column

Updated query

The following is an example of the updated query using the CloudWatch Logs Insights query language. Notice how the updated prompt appears in a comment before the updated query. After the query, you can read an explanation that describes how the original query has been updated.

```
# Show top 20 slowest requests instead and display requestId as a column
fields @timestamp, @message, @requestId, @duration
| sort @duration desc
| limit 20
# This query modifies the original query by replacing the @message field with the
  @requestId field and changing the limit from 10 to 20 to return the top 20 log events
  by duration instead of the top 10.
```

Opting out of using your data for service improvement

The natural language prompt data you provide to train the AI model and generate relevant queries is used solely to provide and maintain your service. This data might be used to improve the quality of CloudWatch Logs Insights. Your trust and privacy, as well as the security of your content, is our highest priority. For more information, see [AWS Service Terms](#) and [AWS responsible AI policy](#).

You can opt out of having your content used to develop or improve the quality of natural language queries by creating an AI service opt-out policy. To opt-out of data collection for all CloudWatch Logs AI features, including the query generation capability, you must create an opt-out policy for CloudWatch Logs. For more information, see [AI services opt-out policies](#) in the *AWS Organizations User Guide*.

Supported logs and discovered fields

CloudWatch Logs Insights supports different log types. For every log that's sent to a Standard class log group in Amazon CloudWatch Logs, CloudWatch Logs Insights automatically generates five system fields:

- `@message` contains the raw unparsed log event. This is the equivalent to the `message` field in [InputLogevent](#).
- `@timestamp` contains the event timestamp in the log event's `timestamp` field. This is the equivalent to the `timestamp` field in [InputLogevent](#).
- `@ingestionTime` contains the time when CloudWatch Logs received the log event.
- `@logStream` contains the name of the log stream that the log event was added to. Log streams group logs through the same process that generated them.
- `@log` is a log group identifier in the form of *account-id:log-group-name*. When querying multiple log groups, this can be useful to identify which log group a particular event belongs to.
- `@entity` contains flattened JSON related to entities for the [Explore related telemetry](#) feature.

For example, this JSON can represent an entity.

```
{
  "Entity": {
    "KeyAttributes": {
      "Type": "Service",
      "Name": "PetClinic"
    },
    "Attributes": {
      "PlatformType": "AWS::EC2",
      "EC2.InstanceId": "i-1234567890123"
    }
  }
}
```

For this entity, the extracted system fields would be the following:

```
@entity.KeyAttributes.Type = Service
@entity.KeyAttributes.Name = PetClinic
@entity.Attributes.PlatformType = AWS::EC2
@entity.Attributes.EC2.InstanceId = i-1234567890123
```

Note

Field discovery is supported only for log groups in the Standard log class. For more information about log classes, see [Log classes](#).

CloudWatch Logs Insights inserts the **@** symbol at the start of fields that it generates.

For many log types, CloudWatch Logs also automatically discovers the log fields contained in the logs. These automatic discovery fields are shown in the following table.

For other types of logs with fields that CloudWatch Logs Insights doesn't automatically discover, you can use the `parse` command to extract and create extracted fields for use in that query. For more information, see [CloudWatch Logs Insights language query syntax](#).

If the name of a discovered log field starts with the `@` character, CloudWatch Logs Insights displays it with an additional `@` appended to the beginning. For example, if a log field name is `@example.com`, this field name is displayed as `@@example.com`.

Note

Except for `@message`, `@timestamp`, or `@log`, you can create field indexes for discovered fields. For more information about field indexes, see [Create field indexes to improve query performance and reduce scan volume](#).

Log type	Discovered log fields
Amazon VPC flow logs	<code>@timestamp</code> , <code>@logStream</code> , <code>@message</code> , <code>accountId</code> , <code>endTime</code> , <code>interfaceId</code> , <code>logStatus</code> , <code>startTime</code> , <code>version</code> , <code>action</code> , <code>bytes</code> , <code>dstAddr</code> , <code>dstPort</code> , <code>packets</code> , <code>protocol</code> , <code>srcAddr</code> , <code>srcPort</code>
Route 53 logs	<code>@timestamp</code> , <code>@logStream</code> , <code>@message</code> , <code>edgeLocation</code> , <code>ednsClientSubnet</code> , <code>hostZoneId</code> , <code>protocol</code> , <code>queryName</code> , <code>queryTimestamp</code> , <code>queryType</code> , <code>resolverIp</code> , <code>responseCode</code> , <code>version</code>
Lambda logs	<code>@timestamp</code> , <code>@logStream</code> , <code>@message</code>, <code>@requestId</code> , <code>@duration</code>, <code>@billedDuration</code> , <code>@type</code>, <code>@maxMemoryUsed</code> , <code>@memorySize</code> If a Lambda log line contains an X-Ray trace ID, it also includes the following fields: <code>@xrayTraceId</code> and <code>@xraySegmentId</code> . CloudWatch Logs Insights automatically discovers log fields in Lambda logs, but only for the first embedded JSON fragment in each log event. If a Lambda log event contains multiple JSON fragments, you can parse and

Log type	Discovered log fields
	extract the log fields by using the parse command. For more information, see Fields in JSON logs .
CloudTrail logs Logs in JSON format	For more information, see Fields in JSON logs .
Other log types	@timestamp , @ingestionTime , @logStream , @message, @log.

Fields in JSON logs

With CloudWatch Logs Insights, you use dot notation to represent JSON fields. This section contains an example JSON event and code snippet that show how you can access JSON fields using dot notation.

Example: JSON event

```
{
  "eventVersion": "1.0",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn: aws: iam: : 123456789012: user/Alice",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "accountId": "123456789012",
    "userName": "Alice"
  },
  "eventTime": "2014-03-06T21: 22: 54Z",
  "eventSource": "ec2.amazonaws.com",
  "eventName": "StartInstances",
  "awsRegion": "us-east-2",
  "sourceIpAddress": "192.0.2.255",
  "userAgent": "ec2-api-tools1.6.12.2",
  "requestParameters": {
    "instancesSet": {
      "items": [
        {
          "instanceId": "i-abcde123"
```

```

        }
      ]
    }
  },
  "responseElements": {
    "instancesSet": {
      "items": [
        {
          "instanceId": "i-abcde123",
          "currentState": {
            "code": 0,
            "name": "pending"
          },
          "previousState": {
            "code": 80,
            "name": "stopped"
          }
        }
      ]
    }
  }
}

```

The example JSON event contains an object that's named `userIdentity`. `userIdentity` contains a field that's named `type`. To represent value of `type` using dot notation, you use `userIdentity.type`.

The example JSON event contains arrays that flatten to lists of nested field names and values. To represent the value of `instanceId` for the first item in `requestParameters.instancesSet`, you use `requestParameters.instancesSet.items.0.instanceId`. The number `0` that's placed before the field `instanceId` refers to the position of values for the field `items`. The following example contains a code snippet that shows how you can access nested JSON fields in a JSON log event.

Example: Query

```

fields @timestamp, @message
| filter requestParameters.instancesSet.items.0.instanceId="i-abcde123"
| sort @timestamp desc

```

The code snippet shows a query that uses dot notation with the `filter` command to access the value of the nested JSON field `instanceId`. The query filters on messages where the value of

`instanceId` equals "i-abcde123" and returns all of the log events that contain the specified value.

 Note

CloudWatch Logs Insights can extract a maximum of 200 log event fields from a JSON log. For additional fields that aren't extracted, you can use the `parse` command to extract the fields from the raw unparsed log event in the message field. For more information about the `parse` command, see [Query syntax](#) in the Amazon CloudWatch User Guide.

Create field indexes to improve query performance and reduce scan volume

You can create *field indexes* of fields in your log events for efficient equality-based searches. When you then use a field index in a CloudWatch Logs Insights query, the query attempts to skip processing log events that are known to not include the indexed field. This reduces the scan volume of your queries that use field indexes, making it possible to return results faster. This can help you quickly search petabytes of total logs across thousands of log groups, and hone in on relevant logs faster. Good fields to index are fields that you often need to query for. Fields that have high cardinality of values are also good candidates for field indexes because a query using these field indexes will complete faster because it limits the log events that are being matched to the target value.

For example, suppose you have created a field index for `requestId`. Then, any CloudWatch Logs Insights query on that log group that includes `requestId = value` or `requestId IN [value, value, ...]` will attempt to process only the log events that are known to contain that indexed field and the queried value, and that CloudWatch Logs has detected a value for that field in the past.

You can also leverage your field indexes to create efficient queries of larger numbers of log groups. When you use the `filterIndex` command in your query instead of the `filter` command, the query will run against selected log groups on log events that have field indexes. These queries can scan as many as 10,000 log groups which you choose by specifying as many as five log group name prefixes. If this is a monitoring account in CloudWatch cross-account observability, you can choose all the source accounts or specify individual source accounts to select the log groups".

Indexed fields are case-sensitive. For example, a field index of `RequestId` won't match a log event containing `requestId`.

Fields indexes are supported only for the structured log formats of JSON and service logs.

CloudWatch Logs provides default field indexes for all log groups in the Standard log class. Default field indexes are automatically available for the following fields:

- `@logStream`
- `@aws.region`
- `@aws.account`
- `@source.log`
- `@data_source_name`
- `@data_source_type`
- `@data_format`
- `traceId`
- `severityText`
- `attributes.session.id`

CloudWatch Logs provides default field indexes for certain data source name and type combinations as well. Default field indexes are automatically available for the following data source name and type combinations:

Data Source Name and Type	Default Field Indexes
amazon_vpc.flow	action logStatus region flowDirection type
amazon_route53_resolver_query	query_type transport

Data Source Name and Type	Default Field Indexes
	rcode
aws_waf.access	action httpRequest.country
aws_cloudtrail.data	eventSource
aws_cloudtrail.management	eventName awsRegion userAgent errorCode eventType managementEvent readOnly eventCategory requestId

Default field indexes are in addition to any custom field indexes you define within your policy. Default field indexes are not counted towards your [field index quota](#).

CloudWatch Logs indexes only the log events ingested after an index policy is created. It doesn't index log events ingested before the policy was created. After you create a field index, each matching log event remains indexed for 30 days from the log event's ingestion time.

Note

If you create a field index policy in a monitoring account, that policy is not used for log groups in linked source accounts. A field index policy applies only in the account where it is created.

The rest of the topics in this section explain how to create field indexes. For information about referring to field indexes in your queries, see [filterIndex](#) and [filter](#).

Topics

- [Field index syntax and quotas](#)
- [Create an account-level field index policy](#)
- [Create a log-group level field index policy](#)
- [Log group selection options when creating a query](#)
- [Effects of deleting a field index policy](#)

Field index syntax and quotas

You create field indexes by creating *field index policies*. You can create account-level index policies that apply to your whole account, and you can also create policies that apply to only a single log group. For account-wide index policies, you can have one that applies to all log groups in the account. You can also create account-level index policies that apply to a subset of log groups in the account, selected by the prefixes of their log group names. If you have multiple account-level policies in the same account, the log group name prefixes for these policies can't overlap. Similarly, you can create account-level index policies that apply to a specific data source name and type combination. Only one account policy can be created per data source name and type combination.

Log group-level field index policies override account-level field index policies: which apply to the log group as a whole (such as, account-level policies with no selection criteria or with log group name prefix based selection criteria). Account-level policies which match at the log event level (such as, for a given data source name and type combination) will apply in addition to policies which match the log group as a whole. If you create log-group level index policy, that log group does not use account-level policies that match at the log group level.

Matches of log events to the names of field indexes are case-sensitive. For example, a field index of `RequestId` won't match a log event containing `requestId`.

You can have as many as 40 account-level index policies, of these policies 20 can use log group name prefix selection criteria and 20 can use data source based selection criteria. If you have multiple account-level index policies filtered to log group name prefixes, no two of them can use the same or overlapping log group name prefixes. For example, if you have one policy filtered to log groups that start with `my-log`, you can't have another field index policy filtered to `my-logpprod` or `my-logging`. Similarly, if you have multiple account-level index policies filtered to

data source name and type combinations, no two of them can use the same data source name and type. For example, if you have one policy filtered to the data source name `amazon_vpc` and data source type `flow` you cannot create another policy with this combination.

If you have an account-level index policy that has no name prefixes and applies to all log groups, then no other account-level index policy with log group name prefix filters can be created; you can create account-level index policies which use data source name and type filters.

Each index policy has the following quotas and restrictions:

- As many as 20 fields can be included in the policy.
- Each field name can include as many as 100 characters.
- To create an index of a custom field in your log groups that starts with `@`, you must specify the field with an extra `@` at the beginning of the field name. For example, if your log events include a field named `@userId`, you must specify `@@userId` to create an index for this field.

For account-level index policies with data source name and type based selection criteria an additional restriction applies: all of the fields must be primitive data types, nested primitives are only supported for structs.

Generated fields and reserved fields

CloudWatch Logs Insights automatically generates system fields in each log event. These generated fields are prefixed with `@`. For more information about generated fields, see [Supported logs and discovered fields](#).

Of these generated fields, the following are supported for use as field indexes:

- `@logStream`
- `@ingestionTime`
- `@requestId`
- `@type`
- `@initDuration`
- `@duration`
- `@billedDuration`
- `@memorySize`
- `@maxMemoryUsed`

- `@xrayTraceId`
- `@xraySegmentId`

To index these generated fields, you don't need to add an extra `@` when specifying them, as you have to do for custom fields that start with `@`. For example, to create a field index for `@logStream`, just specify `@logStream` as the field index.

CloudWatch Logs provides default field indexes for all log groups in the Standard log class. Default field indexes are automatically available for the following fields:

- `@logStream`
- `@aws.region`
- `@aws.account`
- `@source.log`
- `@data_source_name`
- `@data_source_type`
- `@data_format`
- `traceId`
- `severityText`
- `attributes.session.id`

CloudWatch Logs provides default field indexes for certain data source name and type combinations as well. Default field indexes are automatically available for the following data source name and type combinations:

Data Source Name and Type	Default Field Indexes
amazon_vpc.flow	action
	logStatus
	region
	flowDirection
	type

Data Source Name and Type	Default Field Indexes
amazon_route53.resolver_query	query_type transport rcode
aws_waf.access	action httpRequest.country
aws_cloudtrail.data	eventSource
aws_cloudtrail.management	eventName awsRegion userAgent errorCode eventType managementEvent readOnly eventCategory requestId

Default field indexes are in addition to any custom field indexes you define within your policy. Default field indexes are not counted towards your [field index quota](#).

Child fields and array fields in JSON logs

You can index fields that are nested child fields or array fields in JSON logs.

For example, you can create an index of the `accessKeyId` child field within the `userIdentity` field within this log:

```
{
  "eventVersion": "1.0",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn: aws: iam: : 123456789012: user/Alice",
    "accessKeyId": "11112222",
    "accountId": "123456789012",
    "userName": "Alice"
  },
  "eventTime": "2014-03-06T21: 22: 54Z",
  "eventSource": "ec2.amazonaws.com",
  "eventName": "StartInstances",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.255",
  "userAgent": "ec2-api-tools1.6.12.2",
  "requestParameters": {
    "instancesSet": {
      "items": [{
        "instanceId": "i-abcde123",
        "currentState": {
          "code": 0,
          "name": "pending"
        },
        "previousState": {
          "code": 80,
          "name": "stopped"
        }
      }]
    }
  }
}
```

To create this field, you refer to it using dot notation (`userIdentity.accessKeyId`) both when creating the field index and when specifying it in a query. The query could look like this:

```
fields @timestamp, @message
| filterIndex userIdentity.accessKeyId = "11112222"
```

In the previous example event, the `instanceId` field is in an array within `requestParameters.instancesSet.items`. To represent this field both when creating the field index and when querying, refer to it as

`requestParameters.instancesSet.items.0.instanceId` The 0 refers to that field's place in the array.

Then a query for this field could be the following:

```
fields @timestamp, @message
| filterIndex requestParameters.instancesSet.items.0.instanceId="i-abcde123"
```

Create an account-level field index policy

Use the steps in this section to create a field index policy that applies to all log groups in the account, or to multiple log groups that have log group names that start with the same string.

To create an account-level field index policy

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Settings** and then choose the **Logs** tab.
3. In the **Account level index policies** section, choose **Manage**.
4. Choose **Create index policy**.
5. For **Policy name**, enter a name for your new policy.
6. For **Select scope of policy**, do one of the following:
 - Choose **All standard log groups** to have the index policy apply to all Standard Class log groups in the account.
 - Choose **Log groups by prefix match** to apply the policy to a subset of log groups that all have names that start with the same string. Then, enter the prefix for these log groups in **Enter a prefix name**.

After you enter your prefix, you can choose **Preview prefix matched log groups** to confirm that your prefix matches the log groups that you expected.

Choose **Logs Data by Data source** to apply the policy to a specific data source name and type combination. You can then select the **Data source** and **Data type** from the drop-down menu.

After you select the data source name and type you can select **Get fields** to populate the **Configure field indexes and facets** section with relevant information such as the fields available, included log groups, as well as default and custom field indexes.

7. For **Custom index field configuration**, choose **Add field path** to enter the first field to index.

Then enter the string to use as the value of the field name or select a field from the drop-down menu. This must be an exact case match to what appears in the log events. For example, if your log events include `requestId`, you must enter `requestId` here. `RequestId`, `requestID`, and `request Id` wouldn't match.

If you want to index a custom log field that starts with the `@` character, you must include an extra `@` character when you enter the index string. For example, if you have a custom log field `@emailname`, enter `@emailname` in the **Add field path** box.

You can also create indexes for the `@ingestionTime` and `@logStream` fields that CloudWatch Logs automatically generates. If you do, you don't need to add an extra `@` when specifying them.

8. (Optional) In addition to specifying the field path, you can select **Set as a facet** to create the field as a facet.
9. Repeat the previous step to add as many as 20 field indexes.
10. When you have finished, choose **Create**.

Create a log-group level field index policy

Use the steps in this section to create a field index policy that applies to a single log group.

To create a log-group level field index policy

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs, Log groups**.
3. Choose the name of the log group.
4. Choose the **Field indexes** tab.
5. Choose **Manage field indexes for this log group**.
6. For **Manage log group level field indexes**, choose **Add field path** to enter the first field to index.

Then enter the string to use as the value of the field name. This must be an exact case match to what appears in the log events. For example, if your log events include `requestId`, you must enter `requestId` here. `RequestId`, `requestID`, and `request Id` would not match.

If you want to index a custom log field that starts with the @ character, you must include an extra @ character when you enter the index string. For example, if you have a custom log field @emailname, enter @@emailname in the **Add field path** box.

You can also create indexes for the @ingestionTime and @logStream fields that CloudWatch Logs automatically generates. If you do, do not need to add an extra @ when specifying them.

7. (Optional) In addition to specifying the field path, you can select **Set as a facet** to create the field as a facet.
8. Repeat the previous step to add as many as 20 field indexes.
9. When you have finished, choose **Save**.

Log group selection options when creating a query

This section explains the various ways that you can select log groups to include in a query.

To select log groups for a query in the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Logs Insights**.
3. Select the query language you want to use for this query. You can choose either: **Logs Insights QL**, **OpenSearch PPL**, or **OpenSearch SQL**.
4. There are three ways to select log groups for the query:
 - Use the **Log group name** box. This is the default selection method. You can enter as many as 50 log group names with this method. If this is a monitoring account in CloudWatch cross-account observability, you can select log groups in the source accounts as well as the monitoring account. A single query can query logs from different accounts at once.
 - Use the **Log group criteria** section. In this section, you can choose log groups based on the prefix of the log group names. You can include as many as five prefixes in one query. Log groups having these prefixes in their names will be selected. Alternatively, the **All log groups** option selects all the log groups from the account.
 - If this is a monitoring account in CloudWatch cross-account observability, you can select **All accounts** in the account dropdown menu to select the log groups from all linked accounts. Alternatively, you can individually select which accounts should be included for this query.

If your choices match more than 10,000 log groups, you'll see an error that prompts you to narrow your selection.

5. The default log class for a query is **Standard**. You can use **Log class** to change it to **Infrequent access**.

Using the AWS CLI

To make these types of selections when you start a query from the command line, you can use the `source` command in your query. For more information and examples, see [SOURCE](#).

Effects of deleting a field index policy

If you delete a field index policy that has been in effect for a time, the following happens:

- For up to 30 days after the policy is deleted, queries can still benefit from the indexed log events.
- If you delete a log-group level index policy, and there is already an account-level policy in place that would apply to that log group, the account-level policy will eventually apply to that log group.

Use facets to group and explore logs

Facets are useful for analyzing logs as they allow you to interactively filter and group your data without running queries. A facet is a field in your logs (such as `ServiceName` or `StatusCode`) that enables filtering, aggregation and analysis across log groups. You can view the list of faceted fields in the CloudWatch Logs Insights console, along with the count of log events for each facet value based on your selected time range. As you select different facets and values, the facet values and counts are updated in real-time, enabling you to interactively explore your logs.

Each facet shows available values and counts automatically extracted from fields in your logs based on the selected time range and query scope, and retained for 30 days. Facet counts shown are approximate. You can use the default facets such as data source name or data source type to explore your logs, or create custom facets on any of the fields in your logs. Data Source name is an AWS service or application that generates the logs (for example, Route 53, Amazon VPC, or CloudTrail) and Data Source type is the specific type of log generated by that service. Default facets are created by CloudWatch and include `@aws.region`, `@data_source_name`, `@data_source_type`, and `@data_format`. For more information, see [Log management](#). Facets

are only available for logs that are ingested in the account. If you have set up cross account observability, the monitoring account cannot view facets based on logs from source accounts.

To create additional facets, select the fields in your logs that are relevant to your troubleshooting and configure them using the index policies. For custom facets, we recommend creating them on low-cardinality fields (fields having less than 100 unique values per day such as Status and ApplicationName). Facets with more than 100 unique values per day are classified as high-cardinality and values for these facets are not displayed. Select one or more facets and click to run queries across your logs.

To get started with facets in CloudWatch Logs Insights:

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Logs Insights**.
3. (Optional) Use the time range selector to select a time period that you want to analyze. For the selected time range, available facets and values are shown in the panel.
4. Select facets to explore your data and see real-time updates of the value distributions across the facets.

Facets with more than 100 unique values are not displayed. To query specific values, use filters in your query instead.

To run a facet-based query

1. Select one or more values across facets.
2. The event count will update based on the facets and values selected.
3. As facet values are selected, the query scope is updated to reflect the selection.
4. After selecting the facets values, click run to execute your query.
5. The maximum number of unique values supported per facet is 100. For example, if there are more than 100 values for a facet, then all the counts are displayed as "-", indicating that the values are unknown.

To save a facet-based query

1. Create your query using one or more facet values.

2. The rest of the steps are the same as saving a Logs Insights query. See [Saving CloudWatch Logs Insights queries](#).
3. Your saved queries are available in the Saved Queries section. When you retrieve a saved query, it will automatically include the facets and values used for the query, making it easy to analyze your logs.

To create an account-level facet

1. To create facets, you need to first create the field as an index and configure it as a facet. In the navigation pane, choose **Settings, Logs, Account level index policies**. Alternatively, you can select **Manage facets** on the facets panel.
2. Choose **Create new index policy**. For details on creating index policies, see [Create an account-level field index policy](#).
3. To create a facet, check **Set as facet** for the selected field in the index policy creation page.

Facet Management using APIs

Facet management can be done using the field index policy. See [field index](#) APIs for details.

Field Index APIs

No.	Name	Description
1	PutIndexPolicy	Creates or updates a field index policy for the specific log group
2	PutAccountPolicy	Creates an account-level data protection policy, subscription filter policy, field index policy, transformer policy, or metric extraction policy that applies to all log groups or a subset of log groups in the account
3	DeleteIndexPolicy	Deletes a log-group level field index policy that was applied to a single log group
4	DeleteAccountPolicy	Deletes a CloudWatch Logs account policy

Pattern analysis

CloudWatch Logs Insights uses machine learning algorithms to find *patterns* when you query your logs. A pattern is a shared text structure that recurs among your log fields. When you view the results of a query, you can choose the **Patterns** tab to see the patterns that CloudWatch Logs found based on a sample of your results. Alternatively, you can append the `pattern` command to your query to analyze the patterns in the entire set of matching log events.

Patterns are useful for analyzing large log sets because a large number of log events can often be compressed into a few patterns.

Consider the following sample of three log events.

```
2023-01-01 19:00:01 [INFO] Calling DynamoDB to store for resource id 12342342k124-12345
2023-01-01 19:00:02 [INFO] Calling DynamoDB to store for resource id 324892398123-12345
2023-01-01 19:00:03 [INFO] Calling DynamoDB to store for resource id 3ff231242342-12345
```

In the previous sample, all three log events follow one pattern:

```
<Time-1> [INFO] Calling DynamoDB to store for resource id <ID-2>
```

Fields within a pattern are called *tokens*. Fields that vary within a pattern, such as a request ID or timestamp, are *dynamic tokens*. Each dynamic token is represented by `<string-number>`. The *string* is a description of the type of data that the token represents. The *number* shows where in the pattern this token appears, compared to the other dynamic tokens.

Common examples of dynamic tokens include error codes, timestamps, and request IDs. A *token value* represents a particular value of a dynamic token. For example, if a dynamic token represents an HTTP error code, then a token value could be 501.

Pattern detection is also used in the CloudWatch Logs anomaly detector and compare features. For more information, see [Log anomaly detection](#) and [Compare \(diff\) with previous time ranges](#).

Getting started with pattern analysis

Pattern detection is automatically performed in any CloudWatch Logs Insights query. Queries that don't include the `pattern` command get both log events and patterns in the results.

If you include the `pattern` command in your query, pattern analysis is performed on the entire matched set of log events. This gives you more accurate pattern results, but the raw log events

are not returned when you use the `pattern` command. When a query doesn't include `pattern`, the pattern results are based either on the first 1000 returned log events, or on the limit value you used in your query. If you include `pattern` in the query, then the results displayed in the **Patterns** tab are derived from all log events matched by the query.

To get started with pattern analysis in CloudWatch Logs Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Logs Insights**.

On the **Logs Insights** page, the query editor contains a default query that returns the 20 most recent log events.

3. Remove the `| limit 20` line in the query box, so that the query looks like the following:

```
fields @timestamp, @message, @logStream, @log
| sort @timestamp desc
```

4. In the **Select log group(s)** drop-down, choose one or more log groups to query.
5. (Optional) Use the time interval selector to select a time period that you want to query.

You can choose between 5-minute and 30-minute intervals; 1-hour, 3-hour, and 12-hour intervals; or a custom time frame.

6. Choose **Run query** to start the query.

When the query finishes running, the **Logs** tab displays a table of log events returned by the query. Above the table is a message about how many records matched the query, similar to **Showing 10,000 of 71,101 records matched**.

7. Choose the **Patterns** tab.
8. The table now displays the patterns found in the query. Because the query did not include the `pattern` command, this tab displays only the patterns discovered among the 10,000 log events that were shown in the table in the **Logs** tab.

For each pattern, the following information is displayed:

- The **Pattern**, with each dynamic token displayed as `<string-number>`. The *string* is a description of the type of data that the token represents. The *number* shows where in the pattern this token appears, compared to the other dynamic tokens.

- The **Event count**, which is the number of times that the pattern appeared in the queried log events. Choose the **Event count** column heading to sort the patterns by frequency.
- The **Event ratio**, which is the percentage of the queried log events that contain this pattern.
- The **Severity type**, which will be one of the following:
 - **ERROR** if the pattern contains the word **Error**.
 - **WARN** if the pattern contains the word **Warn** but doesn't contain **Error**.
 - **INFO** if the pattern doesn't contain either **Warn** or **Error**.

Choose the **Severity info** column heading to sort the patterns by severity.

9. Now change the query. Replace the `| sort @timestamp desc` line in the query with `| pattern @message`, so that the complete query is as follows:

```
fields @timestamp, @message, @logStream, @log
| pattern @message
```

10. Choose **Run query**.

When the query finishes, there are no results in the **Logs** tab. However, the **Patterns** tab likely has a larger number of patterns listed, depending on the total number of log events that were queried.

11. Regardless of whether you included pattern in your query, you can further inspect the patterns that the query returns. To do so, choose the icon in the **Inspect** column for one of the patterns.

The **Pattern inspect** pane appears and displays the following:

- The **Pattern**. Select a token within the pattern to analyze that token's values.
- A histogram showing the number of occurrences of the pattern over the queried time range. This can help you to identify interesting trends such as a sudden increase in occurrence of a pattern.
- The **Log samples** tab displays a few of the log events that match the selected pattern.
- The **Token Values** tab displays the values of the selected dynamic token, if you have selected one.

Note

A maximum of 10 token values is captured for each token. Token counts might not be precise. CloudWatch Logs uses a probabilistic counter to generate the token count, not the absolute value.

- The **Related patterns** tab displays other patterns that frequently occurred near the same time as the pattern that you are inspecting. For example, if a pattern for an ERROR message was usually accompanied by another log event marked as INFO with additional details, that pattern is displayed here.

Details about the pattern command

This section contains more details about the pattern command and its uses.

- In the previous tutorial, we removed the `sort` command when we added `pattern` because a query is not valid if it includes a pattern command after a sort command. It is valid to have a pattern before a sort.

For more details about pattern syntax, see [pattern](#).

- When you use `pattern` in a query, `@message` must be one of the fields selected in the `pattern` command.
- You can include the `filter` command before a `pattern` command to cause only the filtered set of log events to be used as input for pattern analysis.
- To see pattern results for a particular field, such as a field derived from the `parse` command, use `pattern @fieldname`.
- Queries with non-log output, such as queries with the `stats` command, do not return pattern results.

Save and re-run CloudWatch Logs Insights queries

After you create a query, you can save it, and run it again later. Queries are saved in a folder structure, so you can organize them. You can save as many as 1000 queries per region and per account.

Queries are saved on a Region-specific level, not a user-specific level. If you create and save a query, other users with access to CloudWatch Logs in the same Region can see all saved queries and their folder structures in the Region.

To save a query, you must be logged into a role that has the permission `logs:PutQueryDefinition`. To see a list of your saved queries, you must be logged into a role that has the permission `logs:DescribeQueryDefinitions`.

To save a query

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. In the query editor, create a query.
4. Choose **Save**.

If you don't see a **Save** button, you need to change to the new design for the CloudWatch Logs console. To do so:

- a. In the navigation pane, choose **Log groups**.
 - b. Choose **Try the new design**.
 - c. In the navigation pane, choose **Insights** and return to step 3 of this procedure.
5. Enter a name for the query.
 6. (Optional) Choose a folder where you want to save the query. Select **Create new** to create a folder. If you create a new folder, you can use slash (/) characters in the folder name to define a folder structure. For example, naming a new folder **folder-level-1/folder-level-2** creates a top-level folder called **folder-level-1**, with another folder called **folder-level-2** inside that folder. The query is saved in **folder-level-2**.
 7. (Optional) Change the query's log groups or query text.
 8. Choose **Save**.

Tip

You can create a folder for saved queries with `PutQueryDefinition`. To create a folder for your saved queries, use a forward slash (/) to prefix your desired query name with your

desired folder name: `<folder-name>/<query-name>`. For more information about this action, see [PutQueryDefinition](#).

To run a saved query

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. On the right, choose **Queries**.
4. Select your query from **Saved queries** list. It appears in the query editor.
5. Choose **Run**.

To save a new version of a saved query

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. On the right, choose **Queries**.
4. Select your query from **Saved queries** list. It appears in the query editor.
5. Modify the query. If you need to run it to check your work, choose **Run query**.
6. When you are ready to save the new version, choose **Actions, Save as**.
7. Enter a name for the query.
8. (Optional) Choose a folder where you want to save the query. Select **Create new** to create a folder. If you create a new folder, you can use slash (/) characters in the folder name to define a folder structure. For example, naming a new folder **folder-level-1/folder-level-2** creates a top-level folder called **folder-level-1**, with another folder called **folder-level-2** inside that folder. The query is saved in **folder-level-2**.
9. (Optional) Change the query's log groups or query text.
10. Choose **Save**.

To delete a query, you must be logged in to a role that has the `logs:DeleteQueryDefinition` permission.

To edit or delete a saved query

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. On the right, choose **Queries**.
4. Select your query from **Saved queries** list. It appears in the query editor.
5. Choose **Actions, Edit** or **Actions, Delete**.

Add query to dashboard or export query results

After you run a query, you can add the query to a CloudWatch dashboard or copy the results to the clipboard.

Queries added to dashboards run every time you load the dashboard and every time that the dashboard refreshes. These queries count toward your limit of 30 concurrent CloudWatch Logs Insights queries.

To add query results to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. Choose one or more log groups and run a query.
4. Choose **Add to dashboard**.
5. Select the dashboard, or choose **Create new** to create a dashboard for the query results.
6. Select the widget type to use for the query results.
7. Enter a name for the widget.
8. Choose **Add to dashboard**.

To copy query results to the clipboard or download the query results

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. Choose one or more log groups and run a query.
4. Choose **Export results**, and then choose the option you want.

View running queries or query history

You can view the queries currently in progress as well as your recent query history.

Queries currently running includes queries you have added to a dashboard. You are limited to 30 concurrent CloudWatch Logs Insights queries per account, including queries added to dashboards. Only 15 of these 30 queries can use either OpenSearch Service PPL or OpenSearch Service SQL.

To view your recent query history

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Logs Insights**.
3. Choose **History**, if you are using the new design for the CloudWatch Logs console. If you are using the old design, choose **Actions**, **View query history for this account**.

A list of your recent queries appears. You can run any of them again by selecting the query and choosing **Run**.

Under **Status**, CloudWatch Logs displays **In progress** for any queries that are currently running.

Encrypt query results with AWS Key Management Service

By default, CloudWatch Logs encrypts the stored results of your CloudWatch Logs Insights queries using the default CloudWatch Logs server-side encryption method. You can choose to use a AWS KMS key to encrypt these results instead. If you associate a AWS KMS key with your encryption results, then CloudWatch Logs uses that key to encrypt the stored results of all queries in the account.

If you later disassociate a the key from your query results, CloudWatch Logs goes back to the default encryption method for later queries. But the queries that ran while the key was associated are still encrypted with that key. CloudWatch Logs can still return those results after the KMS key is disassociated, because CloudWatch Logs can still continue to reference the key. However, if the key is later disabled, then CloudWatch Logs is unable to read the query results that were encrypted with that key.

⚠ Important

CloudWatch Logs supports only symmetric KMS keys. Do not use an asymmetric key to encrypt your query results. For more information, see [Using Symmetric and Asymmetric Keys](#).

Limits

- To perform the following steps, you must have the following permissions: `kms:CreateKey`, `kms:GetKeyPolicy`, and `kms:PutKeyPolicy`.
- After you associate or disassociate a key from your query results, it can take up to five minutes for the operation to take effect.
- If you revoke CloudWatch Logs access to an associated key or delete an associated KMS key, your encrypted data in CloudWatch Logs can no longer be retrieved.
- You can't use the CloudWatch console to associate a key, you must use the AWS CLI or CloudWatch Logs API.

Step 1: Create an AWS KMS key

To create a KMS key use the following [create-key](#) command:

```
aws kms create-key
```

The output contains the key ID and Amazon Resource Name (ARN) of the key. The following is example output:

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
```

```
    "Arn": "arn:aws:kms:us-west-2:123456789012:key/6f815f63-e628-448c-8251-
e40cb0d29f59",
    "AWSAccountId": "123456789012",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ]
}
```

Step 2: Set permissions on the KMS key

By default, all KMS keys are private. Only the resource owner can use it to encrypt and decrypt data. However, the resource owner can grant permissions to access the key to other users and resources. With this step, you give the CloudWatch Logs service principal permission to use the key. This service principal must be in the same AWS Region where the key is stored.

As a best practice, we recommend that you restrict the use of the key to only those AWS accounts that you specify.

First, save the default policy for your KMS key as `policy.json` using the following [get-key-policy](#) command:

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./
policy.json
```

Open the `policy.json` file in a text editor and add the section in bold from one of the following statements. Separate the existing statement from the new statement with a comma. These statements use Condition sections to enhance the security of the AWS KMS key. For more information, see [AWS KMS keys and encryption context](#).

The Condition section in this example limits the use of the AWS KMS key to the CloudWatch Logs Insights query results in the specified account.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
```

```

        "Sid": "Enable IAM User Permissions",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::111122223333:root"
        },
        "Action": "kms:*",
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.region.amazonaws.com"
        },
        "Action": [
            "kms:Encrypt*",
            "kms:Decrypt*",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:Describe*"
        ],
        "Resource": "*",
        "Condition": {
            "ArnEquals": {
                "aws:SourceArn": "arn:aws:logs:us-east-1:111122223333:query-
result:*"
            },
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            }
        }
    }
]
}

```

Finally, add the updated policy using the following [put-key-policy](#) command:

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

Step 3: Associate a KMS key with your query results

To associate the KMS key with the query results in the account

Use the [disassociate-kms-key](#) command as follows:

```
aws logs associate-kms-key --resource-identifier "arn:aws:logs:region:account-id:query-  
result:*" --kms-key-id "key-arn"
```

Step 4: Disassociate a key from query results in the account

To disassociate the KMS key associated with query results, use the following [disassociate-kms-key](#) command:

```
aws logs disassociate-kms-key --resource-identifier "arn:aws:logs:region:account-  
id:query-result:*"
```

Generate a natural language summary from CloudWatch Logs Insights query results

Analyzing log data is crucial for understanding your applications' behavior, but interpreting large volumes of log entries can be time-consuming. CloudWatch Logs Insights now offers a natural language summarization capability that transforms complex query results into clear, concise summaries. This capability helps you quickly identify issues and gain actionable insights from your log data.

How it works

CloudWatch Logs Insights can generate a human-readable summary from your query results using Amazon Bedrock. The feature supports all CloudWatch Logs Insights query languages and provides clear, actionable insights from your log data.

Regional availability and data processing

Important

When you use this feature, your query results might be processed in a different AWS Region. For example, if you run a query in US East (N. Virginia), the summarization might occur in US West (Oregon).

The following table lists the possible processing AWS Region for the different geographies in which the query results feature is available:

Supported CloudWatch Logs geography	Possible Processing Region
United States (US)	US East (N. Virginia) Region
	US East (Ohio) Region
	US West (Oregon) Region
Europe	Europe (Frankfurt) Region
	Europe (Ireland) Region
	Europe (Paris) Region
	Europe (Stockholm) Region
	Europe (London) Region
Asia Pacific	US East (N. Virginia) Region
	US East (Ohio) Region
	US West (Oregon) Region
South America	US East (N. Virginia) Region
	US East (Ohio) Region
	US West (Oregon) Region

Getting started

To generate a natural language summary

1. Run your CloudWatch Logs Insights query.
2. After the query completes, select **Summarize results**.

Permissions

You must have one of the following:

- `CloudWatchLogsFullAccess` permission
- `CloudWatchLogsReadOnlyAccess` permission
- Custom IAM policy including the `cloudwatch:GenerateQueryResultsSummary`, `logs:GetQueryResults`, `logs:DescribeQueries` and `logs:FilterLogEvents` actions

Data privacy

Your query results are processed securely and aren't used to train or improve CloudWatch Logs Insights or Amazon Bedrock. If you choose to provide feedback on the query results summary using the feedback buttons, your feedback indicates your level of satisfaction with the capability provided in CloudWatch Logs Insights.

Automating log analysis with scheduled queries

Scheduled queries enable you to automate the execution of CloudWatch Logs Insights queries on a regular schedule. Instead of manually running queries to analyze your log data, you can configure scheduled queries to run automatically and deliver results to destinations such as Amazon S3 buckets or Amazon EventBridge event buses. This automation is ideal for generating regular reports, monitoring trends, or triggering downstream processes based on log analysis results.

Scheduled queries support all three query languages available in CloudWatch Logs Insights:

- [Logs Insights query language \(Logs Insights QL\)](#)
- [OpenSearch Service Piped Processing Language \(PPL\)](#)
- [OpenSearch Service Structured Query Language \(SQL\)](#)

Contents

- [Understanding scheduled queries concepts](#)
- [Schedule expression reference](#)
- [Best practices](#)
- [Getting started with scheduled queries](#)
- [Configuring S3 destinations for scheduled queries](#)
- [Troubleshooting scheduled queries](#)

Understanding scheduled queries concepts

Before creating scheduled queries, understand these key concepts that affect how your queries run and where results are delivered.

IAM role separation

Scheduled queries require two separate IAM roles: one for executing queries and another for delivering results to Amazon S3 or Amazon EventBridge event buses. Understanding why this separation exists helps you configure permissions correctly and use the security and operational benefits it provides.

The two-role architecture divides responsibilities between data access and data delivery. The query execution role accesses your log data and runs queries, while the destination delivery role writes results to your chosen destination. This separation follows the principle of least privilege—each role has only the permissions it needs for its specific function.

Query execution role

Allows CloudWatch Logs to run CloudWatch Logs Insights queries on your behalf. This role needs permissions to access your log groups and execute queries, but doesn't need access to destination resources. Required permissions:

- `logs:StartQuery`
- `logs:StopQuery`
- `logs:GetQueryResults`
- `logs:DescribeLogGroups`
- `logs:Unmask` if unmask data is required

For KMS-encrypted log groups: `kms:Decrypt` and `kms:DescribeKey` permissions for the KMS key used to encrypt the log groups. These permissions need to be added as well.

Trust relationship requirement: The query execution role must include a trust policy that allows the CloudWatch Logs service (`logs.amazonaws.com`) to assume the role. Without this trust relationship, scheduled queries will fail with permission errors.

Example trust policy for the query execution role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Example permissions policy for the query execution role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:GetQueryResults",
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

Destination delivery role

Allows CloudWatch Logs to deliver query results to your chosen destination. This role only needs permissions for the specific destination service, following the principle of least privilege. Required permissions vary by destination type.

Trust relationship requirement: The destination delivery role must also include a trust policy that allows the CloudWatch Logs service (`logs.amazonaws.com`) to assume the role.

Example permissions policy for S3 destination delivery role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::your-scheduled-query-results-bucket/*"
    }
  ]
}
```

This separation provides practical benefits for your operations. From a security perspective, if you need to change where results are delivered, you only modify the destination delivery role without changing the query execution permissions. For compliance and auditing, you can clearly track which role accesses sensitive log data and which role writes to external systems. This makes it easier to demonstrate that your log analysis infrastructure follows security best practices.

Cross-region and cross-account usage

A scheduled query is created in a specific region and runs in that region. However, you can query log groups and deliver results across regions and accounts. You need to set up one or more AWS accounts as *monitoring accounts* and link them with multiple *source accounts*. A monitoring account is a central AWS account that can view and interact with observability data generated from source accounts. A source account is an individual AWS account that generates observability data for the resources that reside in it. Source accounts share their observability data with the monitoring account. So you can setup scheduled queries from the monitoring account using the log groups of all linked accounts.

Querying cross-region log groups

Your scheduled query can access log groups in any region. Specify log groups using their full ARN format: `arn:aws:logs:region:account-id:log-group:log-group-name`. The query execution role needs `logs:StartQuery` and `logs:GetQueryResults` permissions for log groups in all target regions.

Important

When querying log groups or delivering results across regions, log data crosses regional boundaries. Consider the following:

- **Data residency requirements** - Ensure cross-region data transfer complies with your organization's data governance policies and regulatory requirements
- **Data transfer costs** - Cross-region data transfer incurs additional charges
- **Network latency** - Queries accessing log groups in distant regions may experience higher latency

For optimal performance and cost efficiency, create scheduled queries in the same region as your primary log groups.

Alternative approach: Use [CloudWatch Logs centralization](#) to replicate log data from multiple accounts and regions into a central monitoring account. This allows you to create scheduled queries in a single region that access all your centralized logs, avoiding cross-region queries and simplifying IAM permissions management.

Schedule expressions and timezone handling

The schedule you define determines when your query runs and how often it executes. Choosing the right schedule expression affects when you receive results and how much data you query. Understanding the expression types helps you choose between simplicity and precision.

Cron expressions provide precise control over timing, allowing you to specify exact times, days of the week, or days of the month. Use cron expressions when you need queries to run at specific business hours or align with operational schedules. In the console you can also schedule queries using easy calendar options.

Cron expressions

Run queries at specific times. Format: `cron(minute hour day-of-month month day-of-week year)`. Examples:

- `cron(0 9 * * ? *)` - Every day at 9:00 AM UTC
- `cron(0 18 ? * MON-FRI *)` - Weekdays at 6:00 PM UTC
- `cron(0 0 1 * ? *)` - First day of every month at midnight UTC
- `cron(0 12 ? * SUN *)` - Every Sunday at noon UTC
- `cron(30 8 1 1 ? *)` - January 1st at 8:30 AM UTC

All scheduled queries run in UTC, regardless of your local timezone or where your AWS resources are located. This is particularly important when you schedule queries for business hours or time-sensitive analysis. For example, if your business operates in US Eastern Time and you want a daily report at 9 AM ET, you need to account for the UTC offset (14:00 UTC during daylight saving time, 13:00 UTC otherwise). Plan your schedule expressions with UTC in mind to ensure queries run at the intended times.

Choosing a query language

Scheduled queries support three different query languages, and your choice affects both how you write queries and how easily your team can maintain them. The right language depends on your analysis requirements and your team's existing skills.

If you are primarily filtering and aggregating log data, CloudWatch Logs Insights Query Language offers the most straightforward syntax. For complex data transformations where you need to reshape or enrich data through multiple steps, PPL's pipeline approach makes the logic easier to follow. When you need to perform joins or complex aggregations similar to database operations, SQL provides familiar syntax that database-experienced teams can adopt quickly.

CloudWatch Logs Insights Query Language (CWLI)

Purpose-built for log analysis with intuitive syntax. Best for:

- Text-based log analysis and filtering
- Time-series aggregations and statistics
- Teams new to log analysis

OpenSearch Service Piped Processing Language (PPL)

Pipeline-based query language with powerful data transformation capabilities. Best for:

- Complex data transformations and enrichment
- Multi-step data processing workflows
- Teams familiar with pipeline-based processing

OpenSearch Service Structured Query Language (SQL)

Standard SQL syntax for familiar database-style queries. Best for:

- Complex joins and aggregations
- Business intelligence and reporting
- Teams with strong SQL experience

Destination selection and use cases

Where you send query results determines what you can do with them. This choice shapes your entire downstream workflow—whether you are building long-term analytics, triggering automated

responses, or both. Understanding the strengths of each destination type helps you design the right architecture for your use case.

Amazon S3 destinations are optimized for storage and batch processing. When you need to keep query results for months or years, analyze trends over time, or feed data into analytics platforms, Amazon S3 provides cost-effective storage with unlimited retention. EventBridge destinations are optimized for real-time automation. When query results should trigger immediate actions—like sending alerts, starting workflows, or updating systems—EventBridge delivers results as events that your applications can respond to instantly. By default all query completion events are automatically sent as events to the default event bus, enabling integration with downstream processing systems, Lambda functions, or other event-driven architectures. Results are only published to destinations when query is executed successfully.

Amazon S3 destinations

Store query results as JSON files for long-term retention and batch processing. Best for:

- Historical analysis and data archiving
- Integration with data lakes and analytics platforms
- Compliance and audit requirements
- Cost-effective storage of large result sets

EventBridge destinations

Send query results as events for real-time processing and automation. You can retrieve query results using the `queryId` sent in the event upto 30 days only as we store results for 30 days. Best for:

- Triggering automated responses to query results
- Integration with serverless workflows and Lambda functions
- Real-time alerting and notification systems
- Event-driven architectures and microservices

Query result format and structure

For Amazon S3 destinations - Query results are delivered in JSON format with the same structure as the `GetQueryResults` API response. For Amazon EventBridge understanding the format of scheduled query results helps you design downstream processing and integration workflows.

Query results are delivered in JSON format with the following structure:

```
{
  "version": "0",
  "id": "be72061b-eca2-e068-a7e1-83e01d6fe807",
  "detail-type": "Scheduled Query Completed",
  "source": "aws.logs",
  "account": "123456789012",
  "time": "2025-11-18T11:31:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:logs:us-east-1:123456789012:scheduled-query:477b4380-b098-474e-9c5e-e10a8cc2e6e7"
  ],
  "detail": {
    "queryId": "2038fd57-ab4f-4018-bb2f-61d363f4a004",
    "queryString": "fields @timestamp, @message, @logStream, @log\n| sort\n@timestamp desc\n| limit 10000",
    "logGroupIdentifiers": [
      "/aws/lambda/my-function"
    ],
    "status": "Complete",
    "startTime": 1763465460,
    "statistics": {
      "recordsMatched": 0,
      "recordsScanned": 0,
      "estimatedRecordsSkipped": 0,
      "bytesScanned": 0,
      "estimatedBytesSkipped": 0,
      "logGroupsScanned": 1
    }
  }
}
```

Key elements include:

- **statistics** - Query performance metrics including records matched, scanned, bytes processed, and estimated skipped data
- **startTime** - When the query execution started (Unix timestamp)
- **queryString** - The actual query that was executed
- **queryId** - Query id of the query using which results can be retrieved

- `logGroupIdentifiers` - List of log groups that were queried
- `status` - Query execution status (Complete, Failed, etc.)

Schedule expression reference

Use these reference tables to construct schedule expressions for your scheduled queries. All times are in UTC.

Cron expression syntax

Format: `cron(minute hour day-of-month month day-of-week year)`

Use Case	Cron Expression	Description	Use When
Daily Schedules	<code>cron(0 9 * * ? *)</code>	Every day at 9:00 AM UTC	Daily reports
	<code>cron(0 */6 * * ? *)</code>	Every 6 hours (00:00, 06:00, 12:00, 18:00 UTC)	Frequent monitoring
	<code>cron(30 2 * * ? *)</code>	Every day at 2:30 AM UTC	Off-peak analysis
Business Hours	<code>cron(0 9-17 ? * MON-FRI *)</code>	Every hour from 9 AM to 5 PM, Monday to Friday UTC	Business monitoring
	<code>cron(0 18 ? * MON-FRI *)</code>	Weekdays at 6:00 PM UTC	End of business day
	<code>cron(0 8,12,17 ? * MON-FRI *)</code>	8 AM, noon, and 5 PM on weekdays UTC	Key business times
Weekly Schedules	<code>cron(0 12 ? * SUN *)</code>	Every Sunday at noon UTC	Weekly summaries

Use Case	Cron Expression	Description	Use When
	<code>cron(0 9 ? * MON *)</code>	Every Monday at 9:00 AM UTC	Week start reports
	<code>cron(0 23 ? * FRI *)</code>	Every Friday at 11:00 PM UTC	Week end cleanup
Monthly Schedules	<code>cron(0 0 1 * ? *)</code>	First day of every month at midnight UTC	Monthly reports
	<code>cron(0 9 L * ? *)</code>	Last day of every month at 9:00 AM UTC	Month end processing
	<code>cron(0 10 1 1,4,7,10 ? *)</code>	First day of each quarter at 10:00 AM UTC	Quarterly analysis
High Frequency	<code>cron(* /15 * * * ? *)</code>	Every 15 minutes	Real-time monitoring
	<code>cron(0,30 * * * ? *)</code>	Every 30 minutes (at :00 and :30)	Frequent checks
	<code>cron(0 */2 * * ? *)</code>	Every 2 hours	Regular intervals
Special Cases	<code>cron(30 8 1 1 ? *)</code>	January 1st at 8:30 AM UTC	Annual reports
	<code>cron(0 6 * * SAT,SUN *)</code>	Weekends at 6:00 AM UTC	Weekend processing
	<code>cron(0 0 ? * MON#1 *)</code>	First Monday of every month at midnight UTC	Monthly planning

Cron expression field reference

Field	Values	Wildcards	Examples
Minute (1st)	0-59	* , - /	0 (top of hour), */15 (every 15 min), 0, 30 (twice hourly)
Hour (2nd)	0-23	* , - /	9 (9 AM), */2 (every 2 hrs), 9-17 (business hours)
Day-of-month (3rd)	1-31, L, W	* , - / ?	1 (1st day), L (last day), ? (when using day-of-week)
Month (4th)	1-12 or JAN-DEC	* , - /	1 (January), JAN, 1, 4, 7, 10 (quarterly)
Day-of-week (5th)	1-7 or SUN-SAT	* , - / ? # L	MON-FRI (weekdays), SUN, MON#1 (1st Monday)
Year (6th)	1970-2199	* , - /	* (every year), 2024 (specific year), 2024-2026 (range)

Wildcard characters and special expressions

* (asterisk)

Matches all values in the field. Example: * in the hour field means every hour.

? (question mark)

No specific value. Use in day-of-month or day-of-week when the other is specified. Example: Use ? in day-of-month when specifying MON-FRI in day-of-week.

- (dash)

Range of values. Example: MON-FRI (Monday through Friday), 9-17 (9 AM through 5 PM).

, (comma)

Multiple specific values. Example: MON, WED, FRI (Monday, Wednesday, Friday), 8, 12, 17 (8 AM, noon, 5 PM).

/ (slash)

Step values or increments. Example: 0/15 in minutes means every 15 minutes starting at minute 0 (0, 15, 30, 45). */2 in hours means every 2 hours.

L (last)

Last day of month or last occurrence of weekday. Example: L in day-of-month means last day of month. FRIL means last Friday of month.

W (weekday)

Nearest weekday. Example: 15W means nearest weekday to the 15th of the month.

(nth occurrence)

Nth occurrence of weekday in month. Example: MON#1 means first Monday of month, FRI#2 means second Friday of month.

Common patterns and best practices

- **For business applications:** Use MON-FRI and business hours (e.g., 9-17) to avoid running queries during weekends or off-hours.
- **For high-frequency monitoring:** Use increments like */15 (every 15 minutes) but be mindful of query concurrency limits.
- **For resource efficiency:** Schedule resource-intensive queries during off-peak hours using early morning times like 2-6 UTC.
- **For monthly reports:** Use L for last day of month or specific dates like 1 for first day to ensure consistent timing.

Best practices

Follow these best practices to ensure reliable and efficient scheduled query operations:

Query optimization

- Test queries manually before scheduling to verify performance and results
- Use filter indexes early in your query to reduce data processing
- Limit time ranges to avoid timeouts with high-volume log groups
- Consider query complexity and execution time limits

Schedule planning

- Avoid overlapping executions by ensuring queries complete before the next scheduled run
- Consider log ingestion delays when setting time ranges
- Use cron expressions for specific times
- Spread out the schedules to ensure you do not hit query concurrency limit

Monitoring and maintenance

- Monitor execution history regularly to identify failures or performance issues
- Review and update IAM roles periodically to maintain security
- Test destination accessibility before deploying to production

Authorization

- All the APIs for scheduled query authorize on the scheduled query resource and not on the resources it takes in the input like log groups. Set up IAM policies accordingly
- Manage authorization of log groups using the execution role passed in the APIs

Getting started with scheduled queries

When you create a scheduled query, you'll configure several key components that define how your query runs and where results are delivered. Understanding these components will help you set up effective automated log analysis.

Each scheduled query consists of the following key components:

Query configuration

The CloudWatch Logs Insights query string, target log groups, and query language to use for analysis.

Schedule expression

A cron expression or frequency calendar that defines when the query runs. You can specify timezone settings to ensure queries run at the correct local time. The console displays a human-readable description of your schedule, such as "Run query on every Tuesday at 15:10 for a time range of 5 minutes, effective immediately, on UTC, until indefinitely."

Time range

The lookback period for each query execution, defined by a start time offset from the execution time. This determines how much historical data each query execution will analyze.

Execution schedule preview

The console shows the next three scheduled query runs with exact dates and times (for example, 2025/10/28 15:10, UTC; 2025/11/04 15:10, UTC; 2025/11/11 15:10, UTC), helping you verify that the schedule is configured correctly.

Destinations

Where query results are delivered after successful execution. Supported destinations include Amazon S3 buckets and by default result metadata is send to the default event bus.

Execution role

An IAM role that CloudWatch Logs assumes to execute the query and deliver results to the specified destinations.

Before creating scheduled queries, ensure you have the necessary permissions and resources configured.

Creating a scheduled query

Create a scheduled query that automatically runs CloudWatch Logs Insights queries and delivers results to your chosen destinations.

Prerequisites

Before creating a scheduled query, ensure you have the following:

- **Log groups** - One or more log groups containing the data you want to analyze
- **Execution IAM role** - An IAM role with the following permissions:
 - `logs:StartQuery` - Permission to start CloudWatch Logs Insights queries
 - `logs:GetQueryResults` - Permission to retrieve query results
 - `logs:DescribeLogGroups` - Permission to access log group information. This is only required for prefix based log groups for log group discovery
- **Destination permissions** - Additional IAM permissions for your chosen destination:
 - For Amazon S3 destinations: `s3:PutObject`
- **For AWS CLI and API usage** - Configured AWS credentials with permissions to call CloudWatch Logs APIs

For detailed IAM policy examples, see [Identity and access management for Amazon CloudWatch Logs](#). Also to be noted you can have only 1000 scheduled queries per account.

Console

To create a scheduled query (console)

1. Open the CloudWatch Logs console at <https://console.aws.amazon.com/cloudwatch/home#logsV2:>.
2. In the navigation pane, choose **Logs Insights**.
3. Choose **Create scheduled query**.
4. In the **Query definition** section:
 - a. For **Query language**, choose the query language to use from the list.
 - b. For **Query string**, enter your CloudWatch Logs Insights query in the box.
 - c. For **Log groups**, select the log groups to query from the list.
5. In the **Schedule setup** section:
 - a. For **Schedule expression**, configure when the query runs. Choose from predefined options or enter a custom cron expression.
 - b. For **Effective upon creation**, specify when the schedule becomes active. Choose to start immediately or at a specific date and time using YYYY/MM/DD format.
 - c. For **Time range**, specify the lookback period for each query execution. Enter the duration in minutes that defines how far back from the execution time to query.
 - d. For **Continue indefinitely**, specify when the schedule ends. Choose to run indefinitely or until a specific date and time using YYYY/MM/DD format.
6. The console displays the next three scheduled query runs based on your configuration, showing the exact dates and times in UTC when the query will execute.
7. In the **Post query results to S3 - optional** section (if using S3 destination):
 - a. For **Amazon S3 URI**, enter the Amazon S3 bucket and prefix where results will be stored (for example, `s3://my-bucket/query-results/`).
 - b. Choose **View Amazon S3** to open the Amazon S3 console in a new tab and verify the bucket configuration.
 - c. Choose **Browse Amazon S3** to select an existing Amazon S3 location using the Amazon S3 browser.

8. In the **IAM role for posting query results to Amazon S3** section:
 - a. For **Select an IAM role**, choose an existing IAM role with the required policies, or choose **create a new role in IAM console** to create a new role.
 - b. Use the search field to find and select the appropriate IAM role from the list.
9. In the **IAM role for scheduled query execution** section:
 - a. For **Select an IAM role**, choose an existing IAM role with the required policies, or choose **create a new role in IAM console** to create a new role.
 - b. Use the search field to find and select the appropriate IAM role from the list.
10. Choose **Create schedule** to create the scheduled query.

AWS CLI

To create a scheduled query (AWS CLI)

- Use the `create-scheduled-query` command to create a new scheduled query:

```
aws logs create-scheduled-query \
  --name "ErrorAnalysisQuery" \
  --query-language "CWL" \
  --query-string "fields @timestamp, @message | filter @message like /ERROR/ |
stats count() by bin(5m)" \
  --schedule-expression "cron(8 * * * ? *)" \
  --execution-role-arn "arn:aws:iam::123456789012:role/
CloudWatchLogsScheduledQueryRole" \
  --log-group-identifiers "/aws/lambda/my-function" "/aws/apigateway/my-api" \
  --state "ENABLED"
```

API

To create a scheduled query (API)

- Use the `CreateScheduledQuery` action to create a new scheduled query. The following example creates a scheduled query that runs every hour:

```
{
  "name": "ErrorAnalysisQuery",
  "queryLanguage": "CWL",
```

```
"queryString": "fields @timestamp, @message | filter @message like /ERROR/ |  
stats count() by bin(5m)",  
"scheduleExpression": "cron(8 * * * ? *)",  
"executionRoleArn": "arn:aws:iam::123456789012:role/  
CloudWatchLogsScheduledQueryRole",  
"logGroupIdentifiers": ["/aws/lambda/my-function", "/aws/apigateway/my-  
api"],  
"state": "ENABLED"  
}
```

After creating the scheduled query, you can view and manage it from the **Scheduled queries** page and using ListScheduledQueries API, which shows all your scheduled queries with their names, creation dates, status of last run, last triggered time, and repeat frequency.

Viewing and managing scheduled queries

The following information is available for each query:

Name

The unique name you assigned to the scheduled query. Select the name to view detailed configuration and execution history.

Creation date

The date when the scheduled query was created, displayed in YYYY-MM-DD format.

Status of last run

The execution status of the most recent query run. Possible values include:

- **Complete** - The query executed successfully and results were delivered to all configured destinations.
- **Failed** - The query execution or result delivery failed. Check the execution history for error details.
- **Invalid Query** - The query is invalid and has syntactical issues
- **Timeout** - The query has timed out. A query times out automatically after 60 mins

Last triggered time

The date and time when the query was last executed, displayed in YYYY-MM-DD HH:MM:SS format. Shows **Never** if the query has not yet run.

Repeating every

The schedule frequency for the query. Shows **Custom** for queries using cron expressions or specific frequency descriptions for simpler schedules.

The **Scheduled queries** page provides an overview of all your scheduled queries, showing their current status and execution history so that you can view, monitor, and manage all your scheduled queries from a centralized location. Use this information to monitor query performance, identify issues, and manage your automated log analysis workflows.

Console

To view scheduled queries (console)

1. Open the CloudWatch Logs console at <https://console.aws.amazon.com/cloudwatch/home#logsV2:>.
2. In the navigation pane, choose **Scheduled queries**.

AWS CLI

To list scheduled queries (AWS CLI)

- Use the `list-scheduled-queries` command to list all scheduled queries:

```
aws logs list-scheduled-queries --max-results 10
```

API

To list scheduled queries (API)

- Use the `ListScheduledQueries` action to retrieve all scheduled queries:

```
{  
  "maxResults": 10  
}
```

The **Scheduled queries** page header shows the total number of scheduled queries in your account, helping you track your usage and manage your automated log analysis workflows effectively.

Viewing scheduled query execution history

Use the execution history to monitor the performance of your scheduled queries and troubleshoot any issues with query execution or result delivery.

The execution history shows the status of each query run, including successful executions, failures, and destination processing results. You can use this information to identify patterns, diagnose problems, and verify that your queries are running as expected.

Console

To view execution history (console)

1. In the CloudWatch Logs console, choose **Scheduled queries**.
2. Select the scheduled query you want to examine.
3. Choose the **Execution history** tab.

AWS CLI

To view execution history (AWS CLI)

1. Use the `get-scheduled-query-history` command to retrieve execution history for a scheduled query:

```
aws logs get-scheduled-query-history \
  --identifier "DailyErrorMonitoring" \
  --start-time 1743379200 \
  --end-time 1743465600 \
  --max-results 10
```

2. To filter by execution status, add the `--execution-statuses` parameter:

```
aws logs get-scheduled-query-history \
  --identifier "DailyErrorMonitoring" \
  --start-time 1743379200 \
  --end-time 1743465600 \
  --max-results 1 \
  --execution-statuses "Failed"
```

```
--execution-statuses "SUCCEEDED"
```

API

To view execution history (API)

- Use the `GetScheduledQueryHistory` action to retrieve execution history:

```
{
  "identifier": "DailyErrorMonitoring",
  "startTime": 1743379200,
  "endTime": 1743465600,
  "maxResults": 10,
  "executionStatuses": ["SUCCEEDED", "FAILED"]
}
```

The execution history displays:

- **Execution status** - Running, Complete, Failed, Timeout, or InvalidQuery
- **Triggered time** - When the query was executed
- **Destinations** - Processing status for each configured destination including S3 and EventBridge
- **Error messages** - Details about any failures in query execution or destination processing

Updating a scheduled query

Modify your scheduled query configuration to change the query string, schedule, destinations, or execution role as your requirements evolve.

You can update any aspect of a scheduled query, including the query string, schedule expression, destinations, and execution role. Changes take effect immediately for future executions.

Console

To update a scheduled query (console)

1. In the CloudWatch Logs console, choose **Scheduled queries**.
2. Select the scheduled query you want to update.

3. Choose **Edit**.
4. Modify the configuration as needed.
5. Choose **Save changes**.

AWS CLI

To update a scheduled query (AWS CLI)

- Use the `update-scheduled-query` command to modify an existing scheduled query:

```
aws logs update-scheduled-query \
  --identifier "arn:aws:logs:us-east-1:111122223333:scheduled-
query:5e0c0228-1c29-4d26-904f-59f1f1ba3c8f" \
  --description "Monitor for ERROR level logs daily" \
  --query-language "LogsQL" \
  --query-string "fields @timestamp, @message | filter @message like /ERROR/"
\
  --log-group-identifiers "/aws/lambda/my-function-1" "/aws/lambda/my-
function-2"
```

API

To update a scheduled query (API)

1. Use the `UpdateScheduledQuery` action to modify scheduled query configuration:

```
{
  "identifier": "arn:aws:logs:us-east-1:111122223333:scheduled-
query:5e0c0228-1c29-4d26-904f-59f1f1ba3c8f",
  "queryString": "fields @timestamp, @message | filter @message like /WARNING|
ERROR/ | stats count() by bin(5m)",
  "scheduleExpression": "cron(0 */2 * * ? *)",
  "state": "ENABLED"
}
```

2. To update multiple configuration parameters at once:

```
{
  "identifier": "arn:aws:logs:us-east-1:111122223333:scheduled-
query:5e0c0228-1c29-4d26-904f-59f1f1ba3c8f",
```

```
"queryString": "fields @timestamp, @message, @level | filter @level = 'ERROR'",
"scheduleExpression": "cron(0 8,12,16 * * ? *)",
"executionRoleArn": "arn:aws:iam::111122223333:role/UpdatedScheduledQueryRole",
"logGroupIdentifiers": ["/aws/lambda/my-function", "/aws/lambda/another-function"],
"destinationConfiguration": {
  "s3Configuration": {
    "destinationIdentifier": "s3://111122223333-sqn-results-bucket/processed-results",
    "roleArn": "arn:aws:iam::111122223333:role/Admin"
  }
}
```

Configuring S3 destinations for scheduled queries

Configure Amazon S3 as a destination to store your scheduled query results as JSON files for long-term retention and analysis.

When using Amazon S3 as a destination, query results are stored as JSON files in the specified bucket and prefix. This option is ideal for archiving results, performing batch analysis, or integrating with other AWS services that process S3 data.

The Amazon S3 destination configuration includes the following options:

Amazon S3 URI

The bucket and optional prefix where results will be stored (for example, `s3://my-bucket/query-results/`). Enter the full Amazon S3 URI in the box.

View Amazon S3

Choose this option to open the Amazon S3 console in a new tab, allowing you to verify the bucket exists and check its configuration.

Browse Amazon S3

Choose this option to open the Amazon S3 browser, where you can navigate and select an existing Amazon S3 location without manually typing the URI.

Required IAM permissions for Amazon S3 destinations:

- `s3:PutObject` - Permission to write query result files to the specified Amazon S3 bucket

The IAM role for posting query results to Amazon S3 must be configured separately from the IAM role for scheduled query execution. This separation allows for fine-grained access control, where the execution role can run queries while the Amazon S3 role specifically handles result delivery.

Troubleshooting scheduled queries

Use these troubleshooting topics to resolve common issues with scheduled queries.

Query execution fails with permission errors

Resolve permission errors that prevent scheduled queries from executing or delivering results to destinations.

Permission errors occur when the execution role lacks the necessary permissions to read from log groups or write to destination resources.

To resolve permission errors

1. Verify that the execution role has `logs:StartQuery`, `logs:GetQueryResults`, and `logs:DescribeLogGroups` permissions for the target log groups.
2. Ensure the execution role has write permissions for the destination resources (such as `s3:PutObject` for S3 buckets).
3. Check that the trust policy allows CloudWatch Logs to assume the execution role. The role should trust the logs service principal (`logs.amazonaws.com`) in its trust policy.

Common causes include missing IAM permissions, incorrect resource ARNs in the policy, or trust policy configuration issues.

To prevent permission errors, use the principle of least privilege when creating execution roles and test permissions before deploying scheduled queries to production.

Query times out

Resolve timeout errors that occur when scheduled queries exceed the maximum execution time limit.

Query timeouts happen when the query takes more than 60 mins to process the specified data range, often due to large datasets or complex query logic.

To resolve timeout errors

1. Reduce the time range by decreasing the start time offset to process less data per execution.
2. Optimize the query by adding filters early in the query to reduce the amount of data processed. Use filter indexes to reduce the data scan size.
3. Consider breaking complex queries into simpler, more focused queries.

Common causes include querying large time ranges, processing high-volume log groups, or using complex aggregations without proper filtering.

To prevent timeouts, test queries manually in CloudWatch Logs Insights with your expected data volume and optimize performance before scheduling.

Destination processing fails

Resolve failures that occur when scheduled query results cannot be delivered to the configured destinations.

Destination processing failures happen when the target Amazon S3 bucket or EventBridge event bus is inaccessible or incorrectly configured.

To resolve failures where query results are not getting published to destination

1. Verify that the specified Amazon S3 bucket exists and is accessible.
2. Check the destination configuration for correct URIs.
3. Ensure the execution role has the necessary permissions to write to the destination.

Common causes include deleted or renamed destination resources, incorrect destination URIs, or network connectivity issues.

To prevent destination failures, regularly validate destination configurations and monitor destination resource availability.

Invalid query errors

Resolve syntax and logic errors in scheduled query strings that prevent successful execution.

Invalid query errors occur when the query string contains syntax errors, references non-existent fields, or uses unsupported query language features.

To resolve invalid query errors

1. Test your query manually in CloudWatch Logs Insights to verify syntax and logic.
2. Check that all referenced log fields exist in your target log groups.
3. Verify that the query language features you're using are supported for scheduled queries.

Common causes include typos in field names, incorrect query syntax, or using query features not supported in the scheduled execution environment.

To prevent invalid query errors, always test queries interactively before scheduling and use field discovery features to verify field names.

Query Concurrency Errors

There are some important points mentioned below to keep in mind when concurrency errors are seen as scheduled queries use the same quota as of Cloudwatch Logs insights queries. It is recommended to spread out your schedules to avoid hitting the concurrency limit.

- **Quota:** You can run up to 30 concurrent CloudWatch Logs Insights queries per AWS account.
- **Dashboards:** Queries added to CloudWatch dashboards also count towards this concurrency limit, as they are executed when the dashboard is loaded or refreshed.
- **OpenSearch Service PPL/SQL:** Within the 30 concurrent queries, a subset (currently 15) can utilize either OpenSearch PPL or OpenSearch SQL.
- **Cross-account queries:** The concurrency quota applies to both single and cross-account queries. When using CloudWatch cross-account observability, queries initiated in a monitoring account against a linked source account also count towards the monitoring account's concurrency limit.
- **Infrequent Access Log Groups:** For log groups in the infrequent access log class, the maximum number of concurrent Logs Insights queries is limited to five.

Log anomaly detection

You can detect anomalies in your log data in two ways: by creating a *log anomaly detector* for continuous monitoring, or by using the [anomaly detection](#) command in CloudWatch Logs Insights queries for on-demand analysis.

A log anomaly detector scans the log events ingested into a log group and finds anomalies in the log data automatically. Anomaly detection uses machine-learning and pattern recognition to establish baselines of typical log content. For on-demand analysis, you can use the `anomaly detection` command in CloudWatch Logs Insights queries to identify unusual patterns in time-series data. For more information about query-based anomaly detection, see [Using anomaly detection in CloudWatch Logs Insights](#).

After you create an anomaly detector for a log group, it trains using the past two weeks of log events in the log group for training. The training period can take up to 15 minutes. After the training is complete, it begins to analyze incoming logs to identify anomalies, and the anomalies are displayed in the CloudWatch Logs console for you to examine.

CloudWatch Logs pattern recognition extracts log patterns by identifying static and dynamic content in your logs. Patterns are useful for analyzing large log sets because a large number of log events can often be compressed into a few patterns.

For example, see the following sample of three log events.

```
2023-01-01 19:00:01 [INFO] Calling DynamoDB to store for ResourceID: 12342342k124-12345
2023-01-01 19:00:02 [INFO] Calling DynamoDB to store for ResourceID: 324892398123-1234R
2023-01-01 19:00:03 [INFO] Calling DynamoDB to store for ResourceID: 3ff231242342-12345
```

In the previous sample, all three log events follow one pattern:

```
<Date-1> <Time-2> [INFO] Calling DynamoDB to store for resource id <ResourceID-3>
```

Fields within a pattern are called *tokens*. Fields that vary within a pattern, such as a request ID or timestamp, are referred to as *dynamic tokens*. Each different value found for a dynamic token is called a *token value*.

If CloudWatch Logs can infer the type of data that a dynamic token represents, it displays the token as `<string-number>`. The *string* is a description of the type of data that the token

represents. The *number* shows where in the pattern this token appears, compared to the other dynamic tokens.

CloudWatch Logs assigns the string part of the name based on analyzing the content of the log events that contain it.

If CloudWatch Logs can't infer the type of data that a dynamic token represents, it displays the token as <Token-*number*>, and *number* indicates where in the pattern this token appears, compared to the other dynamic tokens.

Common examples of dynamic tokens include error codes, IP addresses, timestamps, and request IDs.

Logs anomaly detection uses these patterns to find anomalies. After the anomaly detector model training period, logs are evaluated against known trends. The anomaly detector flags significant fluctuations as anomalies.

This chapter describes how to enable anomaly detection, view anomalies, create alarms for log anomaly detectors, and metrics that log anomaly detectors publish. It also describes how to encrypt anomaly detector and its results with AWS Key Management Service.

Creating log anomaly detectors doesn't incur charges.

Severity and priority of anomalies and patterns

Each anomaly found by a log anomaly detector is assigned a *priority*. Each pattern found is assigned a *severity*.

- *Priority* is automatically computed, and is based on both the severity level of the pattern and the amount of deviation from expected values. For example, if a certain token value suddenly increases by 500%, that anomaly might be designated as HIGH priority even if its severity is NONE.
- *Severity* is based only on keywords found in the patterns such as FATAL, ERROR, and WARN. If none of these keywords are found, the severity of a pattern is marked as NONE.

Anomaly visibility time

When you create an anomaly detector, you specify the maximum anomaly visibility period for it. This is the number of days that the anomaly is displayed in the console and is returned by the

[ListAnomalies](#) API operation. After this time period has elapsed for an anomaly, if it continues to happen, it's automatically accepted as regular behavior and the anomaly detector model stops flagging it as an anomaly.

If you don't adjust the visibility time when you create an anomaly detector, 21 days is used as the default.

Suppressing an anomaly

After an anomaly has been found, you can choose to suppress it temporarily or permanently. Suppressing an anomaly causes the anomaly detector to stop flagging this occurrence as an anomaly for the amount of time that you specify. When you suppress an anomaly, you can choose to suppress only that specific anomaly, or suppress all anomalies related to the pattern that the anomaly was found in.

You can still view suppressed anomalies in the console. You can also choose to stop suppressing them.

Frequently asked questions

Does AWS use my data to train machine-learning algorithms for AWS use or for other customers?

No. The anomaly detection model created by the training is based on the log events in a log group and is used only within that log group and that AWS account.

What types of log events work well with anomaly detection?

Log anomaly detection is well-suited for: Application logs and other types of logs where most log entries fit typical patterns. Log groups with events that contain a log level or severity keywords such as **INFO**, **ERROR**, and **DEBUG** are especially well-suited to log anomaly detection.

Log anomaly detection is not suited for: Log events with extremely long JSON structures, such as CloudTrail Logs. Pattern analysis analyzes only up to the first 1500 characters of a log line, so any characters beyond that limit are skipped.

Audit or access logs, such as VPC flow logs, will also have less success with anomaly detection. Anomaly detection is meant to find application issues, so it might not be well-suited for network or access anomalies.

To help you determine whether an anomaly detector is suited to a certain log group, use CloudWatch Logs pattern analysis to find the number of patterns in the log events in the group. If the number of patterns is no more than about 300, anomaly detection might work well. For more information about pattern analysis, see [Pattern analysis](#).

What gets flagged as an anomaly?

The following occurrences can cause a log event to be flagged as an anomaly:

- A log event with a pattern not seen before in the log group.
- A significant variation to a known pattern.
- A new value for a dynamic token that has a discrete set of usual values.
- A large change in the number of occurrences of a value for a dynamic token.

While all the preceding items might be flagged as anomalies, they don't all mean that the application is performing poorly. For example, a higher-than-usual number of 200 success values might be flagged as an anomaly. In cases like this, you might consider suppressing these anomalies that don't indicate problems.

What happens with sensitive data that is being masked?

Any parts of log events that are masked as sensitive data are not scanned for anomalies. For more information about masking sensitive data, see [Help protect sensitive log data with masking](#).

Using anomaly detection in CloudWatch Logs Insights

In addition to creating log anomaly detectors for continuous monitoring, you can also use the `anomaly` command in CloudWatch Logs Insights queries to identify unusual patterns in your log data on-demand. This command extends the existing `pattern` functionality and uses machine learning to detect five types of anomalies including pattern frequency changes, new patterns, and token variations.

The `anomaly` command is particularly useful for:

- Ad-hoc analysis of historical log data to identify unusual patterns
- Investigating specific time periods for anomalous behavior
- Monitoring applications like Lambda functions for execution issues

For more information about using the `anomaly` command in your queries, see [anomaly](#).

This query-based anomaly detection complements the continuous anomaly detectors described in the following sections, giving you both real-time monitoring and on-demand analysis capabilities.

Enable anomaly detection on a log group

Use the following steps to use the CloudWatch console to create a log anomaly detector that scans a log group for anomalies.

You can also create anomaly detectors programmatically. For more information, see [CreateLogAnomalyDetector](#).

To create a log anomaly detector

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Logs, Log Anomalies**.
3. Choose **Create anomaly detector**.
4. Select the log group to create this anomaly detector for.
5. Enter a name for the detector in **Anomaly detector name**.
6. (Optional) Change the **Evaluation frequency** from the default of 5 minutes. Set this value according to the frequency that the log group receives new logs. For example, if the log group receives new log events in batches every 10 minutes, then setting the evaluation frequency to 15 minutes might be appropriate.
7. (Optional) To configure the anomaly detector to look for anomalies only in log events that contain certain words or strings, choose **Filter patterns**.

Then, enter a pattern in **Anomaly detection filter pattern**. For more information about pattern syntax, [Filter pattern syntax for metric filters, subscription filters, filter log events, and Live Tail](#).

(Optional) To test your filter pattern, enter some log messages into **Log event messages** and then choose **Test Pattern**.

8. (Optional) To change the anomaly visibility period from the default or to associate an AWS KMS key with this anomaly detector, choose **Advanced configuration**.
 - a. To change the anomaly visibility period from the default, enter a new value in **Maximum anomaly visibility period (days)**.

- b. To associate an AWS KMS key with this anomaly detector, enter the ARN in **KMS key ARN**. If you assign a key, the anomaly information found by this detector is encrypted at rest with the key. Users must have permissions for this key and for the anomaly detector to retrieve information about the anomalies that it finds.

You must also ensure that the CloudWatch Logs service principal has permission to use the key. For more information, see [Encrypt an anomaly detector and its results with AWS KMS](#).

9. Choose **Enable Anomaly Detection**.

The anomaly detector is created and starts training its model, based on the log events the log group is ingesting. After about 15 minutes, anomaly detection is active and begins to find and surface anomalies.

View anomalies that have been found

After you create one or more log anomaly detectors, you can use the CloudWatch console to view the anomalies that they have found.

You can view anomalies programmatically. For more information, see [ListAnomalies](#).

To view the anomalies found by all of your log anomaly detectors

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Logs, Log Anomalies**.

The **Logs anomalies** table appears. The number at the top next to **Log anomalies** displays how many log anomalies are listed in the table. Each row in the table displays the following information:

- The **Anomaly** column displays a short summary of the anomaly. These summaries are generated by CloudWatch Logs.
- The **Priority** of the anomaly. Priority is automatically computed based on the amount of change in the log events, key words such as `Exception` occurring in a log event, and more.
- The **Log pattern** that the anomaly is based on. For more information about patterns, see [Log anomaly detection](#).
- **Anomaly log trend** displays a histogram depicting the volume of logs matching the pattern.

- **Last detection time** displays the most recent time that this anomaly was found.
 - **First detection time** displays the first time that this anomaly was found.
 - **Anomaly detector** displays the name of the log group containing the log events related to this anomaly. You can choose this name to see the log group details page.
3. To further inspect one anomaly, choose the radio button in its row.

The **Pattern inspect** pane appears and displays the following:

- The **Pattern** that this anomaly is based on. Select a token within the pattern to analyze that token's values.
- A histogram showing the number of occurrences of the anomaly over the queried time range.
- The **Log samples** tab displays a few of the log events that are part of the anomaly.
- The **Token Values** tab displays the values of the selected dynamic token, if you have selected one.

 Note

A maximum of 10 token values is captured for each token. Token counts might not be precise. CloudWatch Logs uses a probabilistic counter to generate the token count, not the absolute value.

4. To suppress an anomaly, choose the radio button in its row and then do the following:
- a. Choose **Actions, Suppress Anomaly**.
 - b. Then specify how long you want the anomaly to be suppressed.
 - c. To suppress all anomalies related to this pattern, select **Suppress Pattern**.
 - d. Choose **Suppress anomaly**.

To view the anomalies found in a single log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Logs, Log groups**.
3. Choose the name of a log group, and then choose the **Anomaly detection** tab.

The **Anomaly detection** table appears. The number at the top next to **Log anomalies** displays how many log anomalies are listed in the table. Each row in the table displays the following information:

- The **Anomaly** column displays a short summary of the anomaly. These summaries are generated by CloudWatch Logs.
 - The **Priority** of the anomaly. Priority is automatically computed based on the amount of change in the log events, key words such as `Exception` occurring in a log event, and more.
 - The **Log pattern** that the anomaly is based on. For more information about patterns, see [Log anomaly detection](#).
 - **Anomaly log trend** displays a histogram depicting the volume of logs matching the pattern.
 - **Last detection time** displays the most recent time that this anomaly was found.
 - **First detection time** displays the first time that this anomaly was found.
4. To further inspect one anomaly, choose the radio button in its row.

The **Pattern inspect** pane appears and displays the following:

- The **Pattern** that this anomaly is based on. Select a token within the pattern to analyze that token's values.
- A histogram showing the number of occurrences of the anomaly over the queried time range.
- The **Log samples** tab displays a few of the log events that are part of the anomaly.
- The **Token Values** tab displays the values of the selected dynamic token, if you have selected one.

 Note

A maximum of 10 token values is captured for each token. Token counts might not be precise. CloudWatch Logs uses a probabilistic counter to generate the token count, not the absolute value.

5. To suppress an anomaly, choose the radio button in its row and then do the following:
- a. Choose **Actions, Suppress Anomaly**.
 - b. Then specify how long you want the anomaly to be suppressed.

- c. To suppress all anomalies related to this pattern, select **Suppress Pattern**.
- d. Choose **Suppress anomaly**.

Create alarms on log anomaly detectors

You can create an alarm for a log anomaly detector in a log group. You can specify for the alarm to go into ALARM state when a specified number of anomalies are found in the log group during a specified period of time. You can also use filters so that only anomalies of specified priorities are counted by the alarm.

To create an alarm for a log anomaly detector

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Log Anomalies**.

The table of log anomaly detectors appears.

3. Choose the radio button for the anomaly detector that you want to set the alarm for, and choose **Create alarm**.

The CloudWatch alarm creation wizard appears. The **LogAnomalyDetector** field displays the name of the anomaly detector that you chose. The **Metric name** field displays **AnomalyCount**.

4. (Optional) To filter this alarm for anomaly priority, do one of the following:
 - To have the alarm count only high-priority anomalies, enter **HIGH** for **LogAnomalyPriority**.
 - To have the alarm count only high- and medium-priority anomalies, enter **MEDIUM** for **LogAnomalyPriority**.

For more information about priority levels, see [Severity and priority of anomalies and patterns](#).

5. Choose to use a static or metric anomaly detection threshold for the alarm. This selection determines how the alarm threshold is set. A **Static** threshold means that the alarm threshold is a static, constant number that you choose. An **Anomaly detection** threshold means that CloudWatch determines a range of usual values, and the alarm triggers if the actual count crosses the threshold of this band. You don't have to choose **Anomaly detection** for a log anomaly detection alarm. For more information about metric anomaly detection, see [Using CloudWatch anomaly detection](#).

6. For **Whenever *your-metric-name* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal**, or **Lower**. Then for **than . . .**, specify a number for your threshold value. The alarm goes into ALARM state if the anomaly detector finds more than this number of alarms during a time specified by **Period**.
7. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see [Evaluating an alarm](#).

8. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
9. Choose **Next**.
10. For **Notification**, choose **Add notification**, and then specify an Amazon SNS topic to notify when your alarm transitions to the ALARM, OK, or INSUFFICIENT_DATA state.
 - a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

 Note

We recommend that you set the alarm to take actions when it goes into **Insufficient data** state in addition to when it goes into **Alarm** state. This is because many issues with the Lambda function that connects to the data source can cause the alarm to transition to **Insufficient data**.

- b. (Optional) To not send Amazon SNS notifications, choose **Remove**.
11. (Optional) If you want your alarm to perform actions for Amazon EC2 Auto Scaling, Amazon EC2, tickets, or AWS Systems Manager, choose the appropriate button, and specify the alarm state and action.

Note

Your alarm can perform Systems Manager actions only when it's in the ALARM state. For information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems](#) and [Incident creation](#).

12. Choose **Next**.
13. Under **Name and description**, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

Tip

The alarm name must contain only UTF-8 characters. It can't contain ASCII control characters.

14. Under **Preview and create**, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Metrics published by log anomaly detectors

CloudWatch Logs publishes the **AnomalyCount** metric to CloudWatch metrics. This metric is published to the AWS/Logs namespace.

The **AnomalyCount** metric is published with the following dimensions:

- **LogAnomalyDetector**– The name of the anomaly detector
- **LogAnomalyPriority**– The priority level of the anomaly

Encrypt an anomaly detector and its results with AWS KMS

Anomaly detector data is always encrypted in CloudWatch Logs. By default, CloudWatch Logs uses server-side encryption for the data at rest. As an alternative, you can use AWS Key Management Service for this encryption. If you do, the encryption is done using an AWS KMS key. Encryption

using AWS KMS is enabled at the anomaly detector level, by associating a KMS key with an anomaly detector.

Important

CloudWatch Logs supports only symmetric KMS keys. Do not use an asymmetric key to encrypt the data in your log groups. For more information, see [Using Symmetric and Asymmetric Keys](#).

Limits

- To perform the following steps, you must have the following permissions: `kms:CreateKey`, `kms:GetKeyPolicy`, and `kms:PutKeyPolicy`.
- After you associate or disassociate a key from an anomaly detector, it can take up to five minutes for the operation to take effect.
- If you revoke CloudWatch Logs access to an associated key or delete an associated KMS key, your encrypted data in CloudWatch Logs can no longer be retrieved.

Step 1: Create an AWS KMS key

To create an KMS key, use the following [create-key](#) command:

```
aws kms create-key
```

The output contains the key ID and Amazon Resource Name (ARN) of the key. The following is example output:

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "key-default-1",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
```

```
{
  "Arn": "arn:aws:kms:us-west-2:123456789012:key/key-default-1",
  "AWSAccountId": "123456789012",
  "EncryptionAlgorithms": [
    "SYMMETRIC_DEFAULT"
  ]
}
```

Step 2: Set permissions on the KMS key

By default, all AWS KMS keys are private. Only the resource owner can use it to encrypt and decrypt data. However, the resource owner can grant permissions to access the KMS key to other users and resources. With this step, you give the CloudWatch Logs service principal permission to use the key. This service principal must be in the same AWS Region where the KMS key is stored.

As a best practice, we recommend that you restrict the use of the KMS key to only those AWS accounts or anomaly detectors that you specify.

First, save the default policy for your KMS key as `policy.json` using the following [get-key-policy](#) command:

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./
policy.json
```

Open the `policy.json` file in a text editor and add the section in bold from one of the following statements. Separate the existing statement from the new statement with a comma. These statements use Condition sections to enhance the security of the AWS KMS key. For more information, see [AWS KMS keys and encryption context](#).

The Condition section in this example limits the use of the AWS KMS key to the specified account, but it can be used for any anomaly detector.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "EnableIAMUserPermissions",
```



```

    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:root"
    },
    "Action": "kms:*",
    "Resource": "*"
  },
  {
    "Sid": "AllowCloudWatchLogsEncryption",
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.us-east-1.amazonaws.com"
    },
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt",
      "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "StringLike": {
        "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:123456789012:anomaly-detector:*"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:logs:us-east-1:123456789012:anomaly-detector:*"
      }
    }
  },
  {
    "Sid": "AllowCloudWatchLogsDescribeKey",
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.us-east-1.amazonaws.com"
    },
    "Action": "kms:DescribeKey",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}

```

```

        }
    },
    {
        "Sid": "AllowCloudWatchLogsReEncryption",
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.us-east-1.amazonaws.com"
        },
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
            },
            "StringLike": {
                "kms:EncryptionContext:aws-crypto-ec:aws:logs:arn":
                "arn:aws:logs:us-east-1:123456789012:anomaly-detector:*"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:logs:us-
east-1:123456789012:anomaly-detector:*"
            }
        }
    },
    {
        "Sid": "AllowCloudWatchLogsDescribeKeyForReEncryption",
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.us-east-1.amazonaws.com"
        },
        "Action": "kms:DescribeKey",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
            }
        }
    }
}

```

```
]
}
```

Finally, add the updated policy using the following [put-key-policy](#) command:

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

Step 3: Associate a KMS key with an anomaly detector

You can associate a KMS key with an anomaly detector when you create it in the console or using the AWS CLI or APIs.

Step 4: Disassociate key from an anomaly detector

After a key has been associated with an anomaly detector, you can't update the key. The only way to remove the key is to delete the anomaly detector, and then re-create it.

Troubleshoot with CloudWatch Logs Live Tail

CloudWatch Logs Live Tail helps you quickly troubleshoot incidents by viewing a streaming list of new log events as they are ingested. You can view, filter, and highlight ingested logs in near real time, helping you to detect and resolve issues quickly. You can filter the logs based on terms you specify, and also highlight logs that contain specified terms to help you quickly find what you are looking for.

Live Tail sessions incur costs by session usage time, per minute. For more information about pricing, see the **Logs** tab at [Amazon CloudWatch Pricing](#).

Live Tail is supported only for log groups in the Standard log class. For more information about log classes, see [Log classes](#).

The following sections explain how to use Live Tail in the console and in the AWS CLI. You can also start a Live Tail session programatically. For more information, see [StartLiveTail](#). For SDK examples, see [Start a Live Tail session using an AWS SDK](#).

You can also use Live Tail in the AWS Toolkit for Visual Studio Code. To start a Live Tail session from the VS Code Command Palette, see the [Amazon CloudWatch Logs Live Tail section](#) of the AWS Toolkit for Visual Studio Code User Guide.

The Live Tail feature is available in all commercial AWS [Regions](#). It is not available in the China Regions or the AWS GovCloud (US) Regions.

Note

The `StartLiveTail` API routes requests using SDK host prefix injection. SDK versions released before April 1, 2026 route to `streaming-logs.Region.amazonaws.com`, which does not support VPC endpoints. SDK versions released on or after April 1, 2026 route to `stream-logs.Region.amazonaws.com`, which supports VPC endpoints. To set up a VPC endpoint for this API, see [Creating a VPC endpoint for CloudWatch Logs](#).

Start a Live Tail session using the AWS CLI

The `start-live-tail` AWS CLI command starts a Live Tail streaming session for one or more log groups in a terminal. A Live Tail session can last for up to three hours. If more than 500 log

events per second match the filter, the log events that are displayed are a sample of the total log events, to provide a real-time tailing experience. For more information about the `start-live-tail` command, see [start-live-tail](#)

You can use the `start-live-tail` in two modes:

- **print-only**– this is the default mode
- **interactive**

print-only

In `print-only` mode, log events are streamed on the terminal. New events are added at the bottom every second, creating a near real-time tailing experience similar to `tail -f` on Linux.

To start a Live Tail session in `print-only` mode, enter the following command.

```
aws logs start-live-tail --log-group-identifiers arn:aws:logs:us-east-1:111111222222:log-group:my-logs
```

When you use `print-only` mode, you can also pipe it with other Linux commands to increase its analytical capabilities. The following example filters log events with the `error` keyword and prints the second and fourth column of these events to help you extract particular information.

```
aws logs start-live-tail --log-group-identifiers arn:aws:logs:us-east-1:111111222222:log-group:my-logs --mode print-only | grep "error" | awk '{print $2, $4}'
```

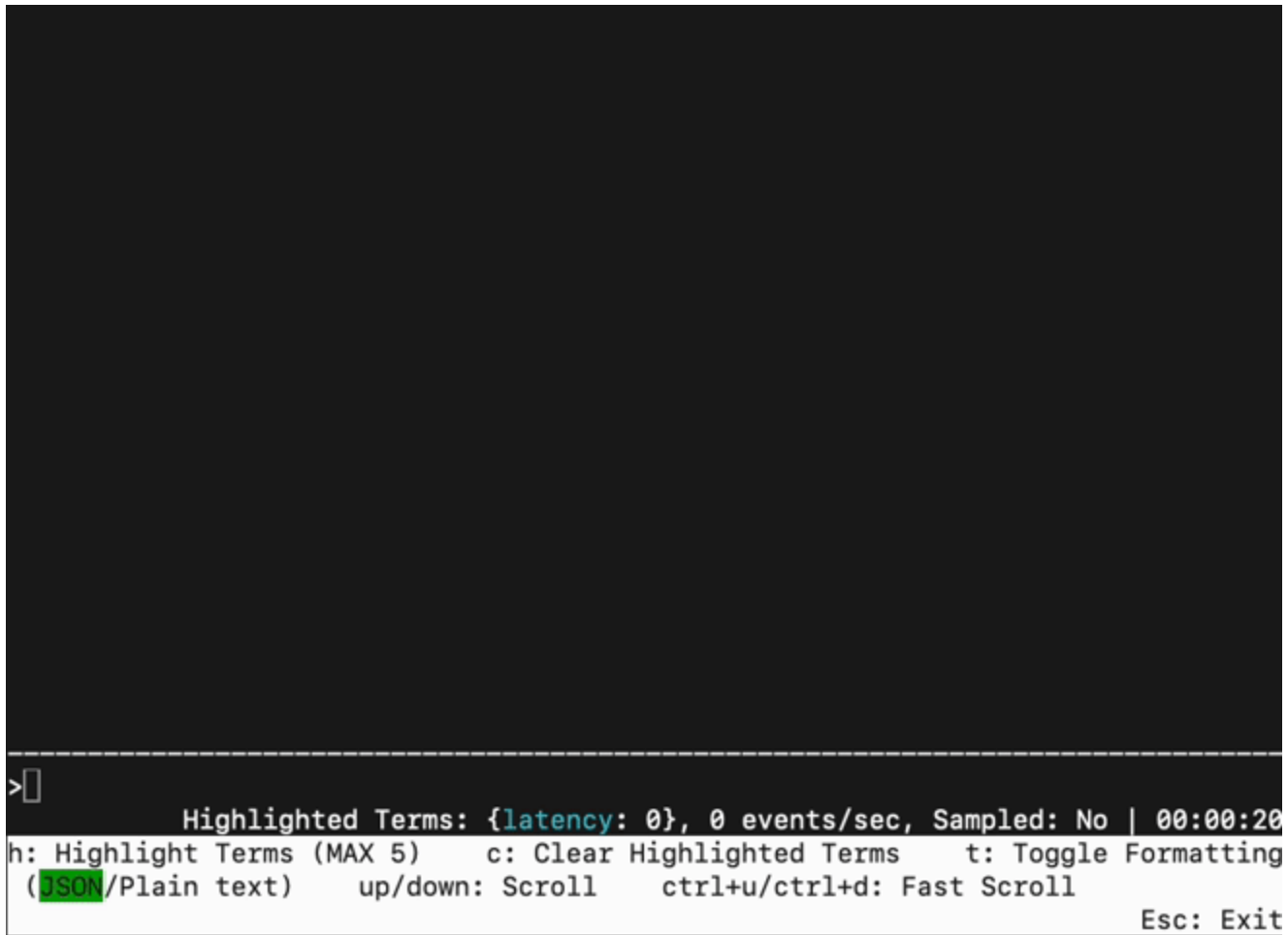
interactive

In `interactive` mode, you can highlight terms and toggle the format of the output log events between JSON and plain text. Interactive mode also displays information about the Live Tail session such as session duration, whether the session is being sampled, and the current highlighted terms and the count of the times they have been encountered.

To start a Live Tail session in `interactive` mode, enter the following command.

```
aws logs start-live-tail --log-group-identifiers arn:aws:logs:us-east-1:111111222222:log-group:my-logs --mode interactive
```

The Live Tail session begins. The following video shows part of an example session.



To highlight a term in the streaming logs, press **h** and then enter the term. The following shows the screen after the term `latency` has been highlighted.

To clear a highlighted term, press **c** and then type the number that represents the term that you want to stop highlighting.

You can press **t** to toggle the display format of incoming events between JSON and plain text. This toggle functionality is best-effort and happens only if the log event format is compatible.

You can use the up arrow and down arrow keys to scroll, and use CTRL+u and CTRL+d to scroll faster.

The following image displays the highlighting of the `latency` term during a Live Tail session.

```

2024-06-27 12:34:56 [INFO] User login successful
2024-06-27 12:34:56 [ERROR] Disk space exhausted
2024-06-27 12:34:56 [WARN] Unauthorized access attempt
2024-06-27 12:34:56 [WARN] Disk space running low
2024-06-27 12:34:56 [INFO] User logout successful
2024-06-27 12:34:56 [WARN] High latency in network.
2024-06-27 12:34:57 [ERROR] Database connection failed
2024-06-27 12:34:57 [INFO] Database connection established
2024-06-27 12:34:57 [WARN] SSL certificate is about to expire
2024-06-27 12:34:57 [INFO] Scheduled task started
2024-06-27 12:34:57 [WARN] Network latency detected.
2024-06-27 12:34:57 [WARN] Outdated library version
2024-06-27 12:34:58 [INFO] New user registered
2024-06-27 12:34:58 [INFO] Database query executed
2024-06-27 12:34:58 [INFO] File uploaded successfully
2024-06-27 12:34:58 [WARN] Memory usage is high
2024-06-27 12:34:59 [ERROR] Unable to connect to server
[INFO] Connection established with the server
[WARN] SSL certificate is about to expire
[INFO] Scheduled task started

```

Highlighted Terms: {latency: 2}, 0 events/sec, Sampled: No | 00:08:21

h: Highlight Terms (MAX 5) c: Clear Highlighted Terms t: Toggle Formatting
 (JSON/Plain text) up/down: Scroll ctrl+u/ctrl+d: Fast Scroll Esc: Exit

Start a Live Tail session in the console

You use the CloudWatch console to start a Live Tail session. The following procedure explains how to start a Live Tail session by using the **Live tail** option in the left navigation pane. You can also start Live Tail sessions from the Log Groups page or the CloudWatch Logs Insights page.

If you are using data protection policies to mask sensitive data in a log group that you are viewing with Live Tail, the sensitive data always appears masked in the Live Tail session. For more information about masking sensitive data in log groups, see [Help protect sensitive log data with masking](#).

Important

If your network security team doesn't allow the use of web sockets, you can't currently access the Live Tail portion of the CloudWatch console. You can use Live Tail with the AWS CLI or APIs. For more information, see [Start a Live Tail session using the AWS CLI](#) and [StartLiveTail](#).

To start a Live Tail session

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Live tail**.
3. For **Select log groups**, select the log groups that you want to view events from, in the Live Tail session. You can select as many as 10 log groups.
4. (Optional) If you selected only one log group, you can filter your Live Tail session further by selecting one or more log streams to view log events from. To do so, under **Select log streams**, select the names of the log streams from the drop down list. Alternatively, you can use the second box under **Select log streams** to enter a log stream name prefix, and then all log streams with names that match the prefix will be selected.
5. (Optional) To display only log events that contain certain words or other strings, enter the word or string in **Add filter patterns**.

For example, to display only log events that include the word **Warning**, enter **Warning**. The filters field is case-sensitive. You can include multiple terms and pattern operators in this field:

- **error 404** displays only log events that include both **error** and **404**
- **?Error ?error** displays log events that include either **Error** or **error**
- **-INFO** displays all log events that don't include **INFO**
- **{ \$.eventType = "UpdateTrail" }** displays all JSON log events where the value of the event type field is **UpdateTrail**

You can also use regular expression (regex) to filter:

- **%ERROR%** uses regex to display all log events consisting of the **ERROR** keyword
- **{ \$.names = %Steve% }** uses regex to display JSON log events where **Steve** is in the property "name"

- [**w1 = %abc%, w2**] uses regex to display space-delimited log events where the first word is **abc**

For more information about pattern syntax, see [Filter pattern syntax](#).

6. (Optional) To highlight some of the displayed log events, enter a term to search for and highlight under **Live Tail**. Enter highlight terms one at a time. If you add multiple terms to highlight, a different color is assigned to represent each term. A highlight indicator is displayed to the left of any log event that contains the specified term, and also appears under the term itself when you expand the log event in the main window to view the full log event.

You can use filtering along with highlighting to quickly troubleshoot issues. For example, you might filter the events to display only the events that contain `Error`, and then also highlight the events that contain `404`.

7. To start the session, choose **Apply filters**

Matching log events begin appearing in the window. The following information is also displayed:

- The timer displays how long the Live Tail session has been active.
 - **events/sec** displays how many ingested log events per second match the filters that you have set.
 - To keep the session from scrolling too fast because many events match the filters, CloudWatch Logs might display only some matching events. If this happens, the percentage of matching events that are displayed on screen is shown in **% displayed**.
8. To pause the flow of events to investigate what is currently displayed, click anywhere in the events window.
 9. During the session, you can use the following to see more details about each log event.
 - To display the entire text for a log event in the main window, choose the arrow next to that log event.
 - To display the entire text for a log event in a side window, choose the **+** magnifying glass next to that log event. The event flow pauses and the side window appears.

Displaying a log event text in the side window can be useful to compare its text to other events in the main window.

10. To stop the Live Tail session, choose **Stop**.

11. To restart the session, optionally use the **Filter** panel to modify the filtering criteria, and choose **Apply filters**. Then choose **Start**.

Cross-account cross-Region log centralization

Amazon CloudWatch Logs data centralization works with AWS Organizations to collect log data from multiple member accounts into one data repository using cross-account and cross-region centralization rules. You define the rules that automatically replicate log data from multiple accounts and AWS Regions into a centralized account within your organization. This capability streamlines log consolidation for improved centralized monitoring, analysis, and compliance across your entire AWS infrastructure.

CloudWatch Logs data centralization offers configuration flexibility to meet operational and security requirements, such as the ability to configure a backup region during rule setup within the destination account to ensure increased resiliency. Additionally, you have full control over encryption behavior for log groups copied from source accounts to handle data originally encrypted with customer-managed KMS keys.

Note

The CloudWatch Logs centralization feature only processes new log data that arrives in source accounts after you create the centralization rule. Historical log data (logs that existed before rule creation) is not centralized.

Data centralization concepts

Before you begin using CloudWatch Logs data centralization, familiarize yourself with the following concepts:

Centralization rule

A configuration that defines how log data from source accounts and regions is replicated to a destination account and region. Rules specify source criteria and destination settings.

Source account

The AWS account where log data originates. Log events from source accounts are replicated to the destination account based on the centralization rules you define.

Destination account

The destination AWS account where replicated log data is stored. This account serves as the centralized location for log analysis and monitoring.

Backup region

An optional secondary region within the destination account where log data can be replicated for increased resiliency and disaster recovery purposes.

Encryption in CloudWatch Logs

Log group data is always encrypted in CloudWatch Logs. By default, CloudWatch Logs uses server-side encryption with 256-bit Advanced Encryption Standard Galois/Counter Mode (AES-GCM) to encrypt log data at rest. As an alternative, you can use AWS Key Management Service for this encryption. If you do, the encryption is done using either an AWS owned KMS key or a customer managed KMS key. KMS key Encryption using AWS KMS is enabled at the log group level, by associating a KMS key with a log group, either when you create the log group or after it exists. After you associate a KMS key with a log group, all newly ingested data for the log group is encrypted using this key. This data is stored in encrypted format throughout its retention period. CloudWatch Logs decrypts this data whenever it is requested. CloudWatch Logs must have permissions for the KMS key whenever encrypted data is requested, such as when a log centralization rule is run against a source account. If you are using customer managed KMS keys, update the KMS keys associated with the source and destination log groups with the tag `LogsManaged = true`. For more information, see [AWS KMS keys](#) in the *AWS Key Management Service Developer Guide*.

Setting up log centralization

To set up CloudWatch Logs Centralization, you need to configure centralization rules that define how log data flows from log groups in source accounts to log groups in your destination account.

Once the centralization rule is enabled and log events are being replicated to the destination account, you can create metric, subscription, and account filters on centralized log groups with enhanced filtering capabilities. These filters can target log events from specific source accounts and regions, and can emit source account and region information as metric dimensions. For more information, see [Creating metrics from log events using filters](#).

Prerequisites

- AWS Organizations must be set up and the source and destination accounts must both belong to the organization.
- Trusted access must be enabled for CloudWatch, the management account and the destination account so provide access to the log data.

Creating a centralization rule

Use the following procedure to create a centralization rule that replicates log data from source accounts to your destination account.

To create a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Configure rule**.
5. Specify source details by setting the following fields, then choose **Next**:
 - a. **Centralization rule name**: Enter a unique name for the centralization rule.
 - b. **Source accounts**: Define source selection criteria to pick accounts from which telemetry data will be centralized. The selection criteria can include:
 - A list of member accounts in the organization
 - A list of organization units in the organization
 - The entire organization

You can provide the selection criteria in two modes:

- **Builder**: A click-based experience to generate the source selection criteria
- **Editor**: A free-form text box to provide the source selection criteria

Supported syntax for source selection criteria:

- *Supported Keys:* OrganizationId | OrganizationUnitId | AccountId | *
 - *Supported Operators:* = | IN | OR
- c. **Source Regions:** Select a list of regions to look for the telemetry data to centralize.
6. Specify destination details by setting the following fields, then choose **Next**:
 - a. **Destination account:** Select an account in the organization that acts as a central destination for telemetry data.
 - b. **Destination Region:** Select a primary region that stores a copy of the centralized telemetry data.
 - c. **Backup Region:** Optionally select a region that stores a second copy of the centralized telemetry data.
 7. Specify telemetry data by setting the following fields, then choose **Next**:
 - a. **Log groups:** Choose one of the following options:
 - **All log groups:** Centralize logs from all log groups in the source accounts.
 - **Filter log group:** Centralize logs from a subset of log groups in the source accounts, matching a log group selection criteria. You can provide the selection criteria in two modes:
 - **Builder:** A click-based experience to generate the log group selection criteria
 - **Editor:** A free-form text box to provide the log group selection criteria

Supported syntax for log group selection criteria:

- *Supported Keys:* LogGroupName | *
- *Supported Operators:* = | != | IN | NOT IN | AND | OR | LIKE | NOT LIKE

- b. **KMS Encrypted Log Group**

 Important

CloudWatch centralization rules will fail to deliver logs from the source account to the destination log groups if the KMS Key provided in the Centralization rule doesn't permit CloudWatch Logs to use it. For more information, see [Step 2: Set permissions on the KMS key](#).

Choose one of the following options:

- **Do not centralize log groups encrypted with Customer Managed KMS keys:** Skip centralization of log events from source log groups encrypted with Customer Managed KMS Keys.
- **Centralize log groups encrypted with Customer Managed KMS keys in destination account with an AWS Managed KMS key:** Centralize log events from source log groups encrypted with Customer Managed KMS Keys into destination log groups that are not associated with Customer Managed KMS Keys but instead use an AWS Managed KMS key.

When this setting is selected, you must also set the following:

- **Destination encryption key ARN:** ARN of the KMS Key belonging to the destination account and the primary destination region, to be associated with newly created destination log groups.
- **Backup destination encryption key ARN (optional):** ARN of the KMS Key belonging to the destination account and the backup destination region, to be associated with newly created destination log groups.

 Note

Note that this setting only applies when the source log group is encrypted using Customer Managed KMS Keys and only applies to newly created log groups in the destination account.

8. Review the centralization rule, optionally make any last-minute edits, and choose **Create Centralization policy**.

Modifying a centralization rule

Use the following procedure to modify an existing centralization rule.

To modify a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Manage rules**.
5. Select the rule to update and choose **Edit**.
6. Update the rule configuration as needed, choosing **Next** to proceed through each step.
7. In Step 4, Review and configure, choose **Update centralization policy**.

Viewing a centralization rule

Use the following procedure to view details of an existing centralization rule.

To view a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Manage rules**.
5. View a list of all existing centralization rules and choose a specific rule name to view its details.

Deleting a centralization rule

Use the following procedure to delete an existing centralization rule.

To delete a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.

4. Choose **Manage rules**.
5. Select the rule to delete and choose **Delete**.
6. Confirm deletion and choose **Delete**.

Monitoring centralization

You can monitor the status and performance of your centralization rules using CloudWatch metrics, the CloudWatch Logs console, and AWS CloudTrail logs. This helps you ensure that log data is being replicated successfully and identify any issues with your centralization configuration.

Monitoring centralization in the console

Use the CloudWatch Logs console to view the status and activity of your centralization rules.

To monitor centralization rules in the console

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Manage rules**.
5. Review the centralization rules list, which displays:
 - **Rule name**: The name of each centralization rule
 - **Rule status**: Current operational status (Active, Inactive, Error)
 - **Creation date**: When the rule was created
 - **Destination account ID**: The account ID of the destination account
 - **Destination Region**: The Region of the destination account
6. Choose a specific rule name to view the rule configuration details

Centralization monitoring

You can monitor centralization rules using the console interface and API operations.

Current monitoring capabilities include:

- *Rule health status*: Monitor the overall health of centralization rules through the console or `GetCentralizationRuleForOrganization` API
- *Rule configuration*: Review rule settings and last update timestamps
- *Failure reasons*: View detailed failure information when rules are marked as UNHEALTHY
- *API activity*: Track centralization API calls through CloudTrail logs

Monitoring rule health

Each centralization rule has a health status that indicates whether it's operating correctly. You can check rule health through the console or programmatically using the API.

Rule health statuses include:

- **HEALTHY**: The rule is operating normally and replicating log data as configured
- **UNHEALTHY**: The rule has encountered issues and may not be replicating data correctly
- **PROVISIONING**: Centralization for the organization is in the process of being set up.

When a rule is marked as UNHEALTHY, the `FailureReason` field provides details about the specific issue that needs to be addressed.

Monitoring centralization API calls with AWS CloudTrail

AWS CloudTrail logs API calls made to the centralization service, allowing you to track configuration changes and troubleshoot issues for accounts that are members of your AWS Organizations.

Key CloudTrail events for centralization include:

- `CreateCentralizationRuleForOrganization`: When a new centralization rule is created
- `UpdateCentralizationRuleForOrganization`: When an existing rule is modified
- `DeleteCentralizationRuleForOrganization`: When a rule is deleted
- `GetCentralizationRuleForOrganization`: When rule details are retrieved
- `ListCentralizationRulesForOrganization`: When rules are listed

You can use CloudTrail logs to audit centralization configuration changes and correlate them with performance issues or replication failures.

Working with log groups and log streams

A log stream is a sequence of log events that share the same source. Each separate source of logs in CloudWatch Logs makes up a separate log stream.

A log group is a group of log streams that share the same retention, monitoring, and access control settings. You can define log groups and specify which streams to put into each group. There is no limit on the number of log streams that can belong to one log group.

For organizations that need to consolidate log data from multiple accounts and regions, you can use CloudWatch Logs Centralization to automatically replicate log groups to a central account. For more information, see [Cross-account cross-Region log centralization](#).

You can use the procedures in this section to work with log groups and log streams.

Create a log group in CloudWatch Logs

When you install the CloudWatch Logs agent on an Amazon EC2 instance using the steps in previous sections of the Amazon CloudWatch Logs User Guide, the log group is created as part of that process. You can also create a log group directly in the CloudWatch console.

To create a log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log Management**.
3. Choose **Actions**, and then choose **Create log group**.
4. Enter a name for the log group, and then choose **Create log group**.

Tip

You can favorite log groups, as well as dashboards and alarms, from the ***Favorites and recents*** menu in the navigation pane. Under the ***Recently visited*** column, hover over the log group that you want to favorite, and choose the star symbol next to it.

Send logs to a log group

CloudWatch Logs automatically receives log events from several AWS services. You can also send other log events to CloudWatch Logs using one of the following methods:

- **CloudWatch agent**— The unified CloudWatch agent can send both metrics and logs to CloudWatch Logs. For information about installing and using the CloudWatch agent, see [Collecting Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent](#) in the *Amazon CloudWatch User Guide*.
- **AWS CLI**—The [put-log-events](#) uploads batches of log events to CloudWatch Logs.
- **Programmatically**— The [PutLogEvents](#) API enables you to programmatically upload batches of log events to CloudWatch Logs.

View log data sent to CloudWatch Logs

You can view and scroll through log data on a stream-by-stream basis as sent to CloudWatch Logs by the CloudWatch Logs agent. You can specify the time range for the log data to view.

To view log data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log Management**.
3. For **Log Groups**, choose the log group to view the streams.
4. In the list of log groups, choose the name of the log group that you want to view.
5. In the list of log streams, choose the name of the log stream that you want to view.
6. To change how the log data is displayed, do one of the following:
 - To expand a single log event, choose the arrow next to that log event.
 - To expand all log events and view them as plain text, above the list of log events, choose **Text**.
 - To filter the log events, enter the desired search filter in the search field. For more information, see [Creating metrics from log events using filters](#).
 - To view log data for a specified date and time range, next to the search filter, choose the arrow next to the date and time. To specify a date and time range, choose **Absolute**. To

choose a predefined number of minutes, hours, days, or weeks, choose **Relative**. You can also switch between UTC and local time zone.

Search log data using filter patterns

You can search your log data using the [Filter pattern syntax for metric filters, subscription filters, filter log events, and Live Tail](#). You can search all the log streams within a log group, or by using the AWS CLI you can also search specific log streams. When each search runs, it returns up to the first page of data found and a token to retrieve the next page of data or to continue searching. If no results are returned, you can continue searching.

You can set the time range you want to query to limit the scope of your search. You could start with a larger range to see where the log lines you are interested in fall, and then shorten the time range to scope the view to logs in the time range that interest you.

You can also pivot directly from your logs-extracted metrics to the corresponding logs.

If you are signed in to an account set up as a monitoring account in CloudWatch cross-account observability, you can search and filter log events from the source accounts linked to this monitoring account. For more information, see [CloudWatch cross-account observability](#).

Search log entries using the console

You can search for log entries that meet a specified criteria using the console.

To search your logs using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log Management**.
3. For **Log Groups**, choose the name of the log group containing the log stream to search.
4. For **Log Streams**, choose the name of the log stream to search.
5. Under **Log events**, enter the filter syntax to use.

To search all log entries for a time range using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log Management**.

3. For **Log Groups**, choose the name of the log group containing the log stream to search.
4. Choose **Search log group**.
5. For **Log events**, select the date and time range, and enter the filter syntax.

Search log entries using the AWS CLI

You can search for log entries that meet a specified criteria using the AWS CLI.

To search log entries using the AWS CLI

At a command prompt, run the following [filter-log-events](#) command. Use `--filter-pattern` to limit the results to the specified filter pattern and `--log-stream-names` to limit the results to the specified log streams.

```
aws logs filter-log-events --log-group-name my-group [--log-stream-  
names LIST_OF_STREAMS_TO_SEARCH] [--filter-pattern VALID_METRIC_FILTER_PATTERN]
```

To search log entries over a given time range using the AWS CLI

At a command prompt, run the following [filter-log-events](#) command:

```
aws logs filter-log-events --log-group-name my-group [--log-stream-  
names LIST_OF_STREAMS_TO_SEARCH] [--start-time 1482197400000] [--end-  
time 148221758365] [--filter-pattern VALID_METRIC_FILTER_PATTERN]
```

Pivot from metrics to logs

You can get to specific log entries from other parts of the console.

To get from dashboard widgets to logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose a dashboard.
4. On the widget, choose the **View logs** icon, and then choose **View logs in this time range**. If there is more than one metric filter, select one from the list. If there are more metric filters than we can display in the list, choose **More metric filters** and select or search for a metric filter.

To get from metrics to logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the search field on the **All metrics** tab, type the name of the metric and press Enter.
4. Select one or more metrics from the results of your search.
5. Choose **Actions, View logs**. If there is more than one metric filter, select one from the list. If there are more metric filters than we can display in the list, choose **More metric filters** and select or search for a metric filter.

Troubleshooting

Search takes too long to complete

If you have a lot of log data, search might take a long time to complete. To speed up a search, you can do the following:

- If you are using the AWS CLI, you can limit the search to just the log streams you are interested in. For example, if your log group has 1000 log streams, but you just want to see three log streams that you know are relevant, you can use the AWS CLI to limit your search to only those three log streams within the log group.
- Use a shorter, more granular time range, which reduces the amount of data to be searched and speeds up the query.

Change log data retention in CloudWatch Logs

By default, log data is stored in CloudWatch Logs indefinitely. However, you can configure how long to store log data in a log group. Any data older than the current retention setting is deleted. You can change the log retention for each log group at any time.

Note

CloudWatch Logs doesn't immediately delete log events when they reach their retention setting. It typically takes up to 72 hours after that before log events are deleted, but in rare situations might take longer.

This means that if you change a log group to have a longer retention setting when it contains log events that are past the expiration date, but haven't been actually deleted,

those log events will take up to 72 hours to be deleted after the new retention date is reached. To make sure that log data is deleted permanently, keep a log group at its lower retention setting until 72 hours has passed after the end of the previous retention period, or you have confirmed that the older log events are deleted.

When log events reach their retention setting they are marked for deletion. After they are marked for deletion, they do not add to your archival storage costs anymore, even if they are not actually deleted until later. These log events marked for deletion are also not included when you use an API to retrieve the `storedBytes` value to see how many bytes a log group is storing.

To change the logs retention setting

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Log groups**.
3. Find the log group to update.
4. In the **Retention** column for that log group, choose the current retention setting, such as **Never Expire**.
5. In **Retention setting**, for **Expire events after**, choose a log retention value, and then choose **Save**.

Protecting log groups from deletion

You can optionally enable deletion protection to prevent accidental deletion of important log groups. For detailed information about deletion protection, see [Protecting log groups from deletion](#).

Protecting log groups from deletion

Enabling deletion protection

You can enable deletion protection when creating a new log group or on existing log groups. During log group creation, select "Enabled deletion protection" or by passing the parameter `--deletion-protection-enabled`. By default, deletion protection is not enabled.

To enable or disable deletion protection on an existing log group (console)

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log Management**.
3. Select the log group you want to protect.
4. Choose **Actions, Edit deletion protection**.
5. In the dialog box, review and then submit changes.

If using the AWS CLI, to enable deletion protection on an existing log group:

```
aws logs put-log-group-deletion-protection \  
--log-group-identifier "/my-application/logs" \  
--deletion-protection-enabled
```

To remove deletion protection on an existing log group:

```
aws logs put-log-group-deletion-protection \  
--log-group-identifier "/my-application/logs" \  
--no-deletion-protection-enabled
```

Error handling

If you attempt to delete a log group with deletion protection enabled, you receive a `ValidationException` with the message: "Cannot delete log group with deletion protection enabled. Disable deletion protection first."

Tag log groups in Amazon CloudWatch Logs

You can assign your own metadata to the log groups you create in Amazon CloudWatch Logs in the form of *tags*. A tag is a key-value pair that you define for a log group. Using tags is a simple yet powerful way to manage AWS resources and organize data, including billing data.

Note

You can use tags to control access to CloudWatch Logs resources, including log groups and destinations. Access to log streams is controlled at the log group level, because of the hierarchical relation between log groups and log streams. For more information about

using tags to control access, see [Controlling access to Amazon Web Services resources using tags](#).

Contents

- [Tag basics](#)
- [Tracking costs using tagging](#)
- [Tag restrictions](#)
- [Tagging log groups using the AWS CLI](#)
- [Tagging log groups using the CloudWatch Logs API](#)

Tag basics

You use AWS CloudFormation the AWS CLI, or CloudWatch Logs API to complete the following tasks:

- Add tags to a log group when you create it.
- Add tags to an existing log group.
- List the tags for a log group.
- Remove tags from a log group.

You can use tags to categorize your log groups. For example, you can categorize them by purpose, owner, or environment. Because you define the key and value for each tag, you can create a custom set of categories to meet your specific needs. For example, you might define a set of tags that helps you track log groups by owner and associated application. Here are several examples of tags:

- Project: Project name
- Owner: Name
- Purpose: Load testing
- Application: Application name
- Environment: Production

Tracking costs using tagging

You can use tags to categorize and track your AWS costs. When you apply tags to your AWS resources, including log groups, your AWS cost allocation report includes usage and costs aggregated by tags. You can apply tags that represent business categories (such as cost centers, application names, or owners) to organize your costs across multiple services. For more information, see [Use Cost Allocation Tags for Custom Billing Reports](#) in the *AWS Billing User Guide*.

Tag restrictions

The following restrictions apply to tags.

Basic restrictions

- The maximum number of tags per log group is 50.
- Tag keys and values are case sensitive.
- You can't change or edit tags for a deleted log group.

Tag key restrictions

- Each tag key must be unique. If you add a tag with a key that's already in use, your new tag overwrites the existing key-value pair.
- You can't start a tag key with `aws:` because this prefix is reserved for use by AWS. AWS creates tags that begin with this prefix on your behalf, but you can't edit or delete them.
- Tag keys must be between 1 and 128 Unicode characters in length.
- Tag keys must consist of the following characters: Unicode letters, digits, white space, and the following special characters: `_ . / = + - @`.

Tag value restrictions

- Tag values must be between 0 and 255 Unicode characters in length.
- Tag values can be blank. Otherwise, they must consist of the following characters: Unicode letters, digits, white space, and any of the following special characters: `_ . / = + - @`.

Tagging log groups using the AWS CLI

You can add, list, and remove tags using the AWS CLI. For examples, see the following documentation:

[create-log-group](#)

Creates a log group. You can optionally add tags when you create the log group.

[tag-resource](#)

Assigns one or more tags (key-value pairs) to the specified CloudWatch Logs resource.

[list-tags-for-resource](#)

Displays the tags the are associated with a CloudWatch Logs resource.

[untag-resource](#)

Removes one or more tags from the specified CloudWatch Logs resource.

Tagging log groups using the CloudWatch Logs API

You can add, list, and remove tags using the CloudWatch Logs API. For examples, see the following documentation:

[CreateLogGroup](#)

Creates a log group. You can optionally add tags when you create the log group.

[TagResource](#)

Assigns one or more tags (key-value pairs) to the specified CloudWatch Logs resource.

[ListTagsForResource](#)

Displays the tags the are associated with a CloudWatch Logs resource.

[UntagResource](#)

Removes one or more tags from the specified CloudWatch Logs resource.

Encrypt log data in CloudWatch Logs using AWS Key Management Service

Log group data is always encrypted in CloudWatch Logs. By default, CloudWatch Logs uses server-side encryption with 256-bit Advanced Encryption Standard Galois/Counter Mode (AES-GCM) to encrypt log data at rest. As an alternative, you can use AWS Key Management Service for this encryption. If you do, the encryption is done using an AWS KMS key. Encryption using AWS KMS is enabled at the log group level, by associating a KMS key with a log group, either when you create the log group or after it exists.

Important

CloudWatch Logs now supports encryption context, using `kms:EncryptionContext:aws:logs:arn` as the key and the ARN of the log group as the value for that key. If you have log groups that you have already encrypted with a KMS key, and you would like to restrict the key to be used with a single account and log group, you should assign a new KMS key that includes a condition in the IAM policy. For more information, see [AWS KMS keys and encryption context](#).

Important

CloudWatch Logs now supports `kms:ViaService` which allows logs to make AWS KMS calls on your behalf. You should add this to your roles which call CloudWatch Logs in either your Key Policy or in IAM. For more information, see [kms:ViaService](#).

After you associate a KMS key with a log group, all newly ingested data for the log group is encrypted using this key. This data is stored in encrypted format throughout its retention period. CloudWatch Logs decrypts this data whenever it is requested. CloudWatch Logs must have permissions for the KMS key whenever encrypted data is requested.

If you later disassociate a KMS key from a log group, CloudWatch Logs encrypts newly ingested data using the CloudWatch Logs default encryption method. All previously ingested data that was encrypted with the KMS key remains encrypted with the KMS key. CloudWatch Logs can still return that data after the KMS key is disassociated, because CloudWatch Logs can still continue to

reference the key. However, if the key is later disabled, then CloudWatch Logs is unable to read the logs that were encrypted with that key.

Important

CloudWatch Logs supports only symmetric KMS keys. Do not use an asymmetric key to encrypt the data in your log groups. For more information, see [Using Symmetric and Asymmetric Keys](#).

Limits

- To perform the following steps, you must have the following permissions: `kms:CreateKey`, `kms:GetKeyPolicy`, and `kms:PutKeyPolicy`.
- After you associate or disassociate a key from a log group, it can take up to five minutes for the operation to take effect.
- If you revoke CloudWatch Logs access to an associated key or delete an associated KMS key, your encrypted data in CloudWatch Logs can no longer be retrieved.
- You can't associate a KMS key with an existing log group using the CloudWatch console.

Step 1: Create an AWS KMS key

To create an KMS key, use the following [create-key](#) command:

```
aws kms create-key
```

The output contains the key ID and Amazon Resource Name (ARN) of the key. The following is example output:

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
```

```
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
    "Arn": "arn:aws:kms:us-west-2:123456789012:key/6f815f63-e628-448c-8251-
e40cb0d29f59",
    "AWSAccountId": "123456789012",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ]
}
```

Step 2: Set permissions on the KMS key

By default, all AWS KMS keys are private. Only the resource owner can use it to encrypt and decrypt data. However, the resource owner can grant permissions to access the KMS key to other users and resources. With this step, you give the CloudWatch Logs service principal and the caller role permission to use the key. This service principal must be in the same AWS Region where the KMS key is stored.

As a best practice, we recommend that you restrict the use of the KMS key to only those AWS accounts or log groups you specify.

First, save the default policy for your KMS key as `policy.json` using the following [get-key-policy](#) command:

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./
policy.json
```

Open the `policy.json` file in a text editor and add the section in bold from one of the following statements. Separate the existing statement from the new statement with a comma. These statements use `Condition` sections to enhance the security of the AWS KMS key. For more information, see [AWS KMS keys and encryption context](#).

The `Condition` section in this example restricts the key to a single log group ARN.

JSON

```
{
    "Version": "2012-10-17",
    "Id": "key-default-1",
```

```

"Statement": [
  {
    "Sid": "Enable IAM User Permissions",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:root"
    },
    "Action": "kms:*",
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.us-east-1.amazonaws.com"
    },
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt",
      "kms:ReEncrypt*",
      "kms:GenerateDataKey*",
      "kms:Describe*"
    ],
    "Resource": "*",
    "Condition": {
      "ArnEquals": {
        "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:111122223333:log-group:log-group-name"
      }
    }
  }
]
}

```

The Condition section in this example limits the use of the AWS KMS key to the specified account, but it can be used for any log group.

JSON

```

{
  "Version": "2012-10-17",
  "Id": "key-default-1",

```



```

    "Statement": [
      {
        "Sid": "Enable IAM User Permissions",
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::123456789012:root"
        },
        "Action": "kms:*",
        "Resource": "*"
      },
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.us-east-1.amazonaws.com"
        },
        "Action": [
          "kms:Encrypt",
          "kms:Decrypt",
          "kms:ReEncrypt*",
          "kms:GenerateDataKey*",
          "kms:Describe*"
        ],
        "Resource": "*",
        "Condition": {
          "ArnLike": {
            "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:123456789012:"
          }
        }
      }
    ]
  }
}

```

Next, add permissions to the role which will be calling the CloudWatch Logs. You can do this by adding an additional statement to the AWS KMS Key Policy or through IAM on the role itself. CloudWatch Logs uses `kms:ViaService` to make calls to AWS KMS on the customer's behalf. For more information, see [kms:ViaService](#).

To add permissions in the AWS KMS Key Policy, add the following additional statement to your key policy. If you use this method, as best practice, scope the policy to only the roles that will be interacting with AWS KMS encrypted log groups.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::account_id:role/role_name"
  },
  "Action": [
    "kms:Encrypt",
    "kms:ReEncrypt*",
    "kms:Decrypt",
    "kms:GenerateDataKey*",
    "kms:Describe*"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "logs.region.amazonaws.com"
      ]
    }
  }
}
```

Alternatively if you would like to manage role permissions in IAM, you can add equivalent permissions through the following policy. This can be added to an existing role policy or attached to a role as an additional separate policy. If you use this method, as best practice, scope the policy to only the AWS KMS keys which will be used for log encryption. For more information, see [Edit IAM policies](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:ReEncrypt*",
        "kms:Decrypt",
```

```

        "kms:GenerateDataKey*",
        "kms:Describe*"
    ],
    "Condition": {
        "StringEquals": {
            "kms:ViaService": [
                "logs.us-east-1.amazonaws.com"
            ]
        }
    },
    "Resource": "arn:aws:kms:us-east-1:444455556666:key/key_id"
}
]
}

```

Finally, add the updated policy using the following [put-key-policy](#) command:

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

Step 3: Associate a KMS key with a log group

You can associate a KMS key with a log group when you create it or after it exists.

To find whether a log group already has a KMS key associated, use the following [describe-log-groups](#) command:

```
aws logs describe-log-groups --log-group-name-prefix "log-group-name-prefix"
```

If the output includes a `kmsKeyId` field, the log group is associated with the key displayed for the value of that field.

To associate the KMS key with a log group when you create it

Use the [create-log-group](#) command as follows:

```
aws logs create-log-group --log-group-name my-log-group --kms-key-id "key-arn"
```

To associate the KMS key with an existing log group

Use the [associate-kms-key](#) command as follows:

```
aws logs associate-kms-key --log-group-name my-log-group --kms-key-id "key-arn"
```

Step 4: Disassociate key from a log group

To disassociate the KMS key associated with a log group, use the following [disassociate-kms-key](#) command:

```
aws logs disassociate-kms-key --log-group-name my-log-group
```

AWS KMS keys and encryption context

To enhance the security of your AWS Key Management Service keys and your encrypted log groups, CloudWatch Logs now puts log group ARNs as part of the *encryption context* used to encrypt your log data. Encryption context is a set of key-value pairs that are used as additional authenticated data. The encryption context enables you to use IAM policy conditions to limit access to your AWS KMS key by AWS account and log group. For more information, see [Encryption context](#) and [IAM JSON Policy Elements: Condition](#).

We recommend that you use different KMS keys for each of your encrypted log groups.

If you have a log group that you encrypted previously and now want to change the log group to use a new KMS key that works only for that log group, follow these steps.

To convert an encrypted log group to use a KMS key with a policy limiting it to that log group

1. Enter the following command to find the ARN of the log group's current key:

```
aws logs describe-log-groups
```

The output includes the following line. Make a note of the ARN. You need to use it in step 7.

```
...  
"kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/01234567-89ab-  
cdef-0123-456789abcdef"  
...
```

2. Enter the following command to create a new KMS key:

```
aws kms create-key
```

3. Enter the following command to save the new key's policy to a `policy.json` file:

```
aws kms get-key-policy --key-id new-key-id --policy-name default --output text > ./policy.json
```

4. Use a text editor to open `policy.json` and add a Condition expression to the policy:

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:Describe*"
      ],
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:111122223333:log-group:LOG-GROUP-NAME"
        }
      }
    }
  ]
}
```

```

    }
  }
}

```

5. Enter the following command to add the updated policy to the new KMS key:

```
aws kms put-key-policy --key-id new-key-ARN --policy-name default --policy file://policy.json
```

6. Enter the following command to associate the policy with your log group:

```
aws logs associate-kms-key --log-group-name my-log-group --kms-key-id new-key-ARN
```

CloudWatch Logs now encrypts all new data using the new key.

7. Next, revoke all permissions except Decrypt from the old key. First, enter the following command to retrieve the old policy:

```
aws kms get-key-policy --key-id old-key-ARN --policy-name default --output text
> ./policy.json
```

8. Use a text editor to open `policy.json` and remove all values from the Action list, except for `kms:Decrypt`

JSON

```

{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {

```

```
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.region.amazonaws.com"
        },
        "Action": [
            "kms:Decrypt"
        ],
        "Resource": "*"
    }
}
```

9. Enter the following command to add the updated policy to the old key:

```
aws kms put-key-policy --key-id old-key-ARN --policy-name default --policy file://
policy.json
```

Help protect sensitive log data with masking

You can help safeguard sensitive data that's ingested by CloudWatch Logs by using log group *data protection policies*. These policies let you audit and mask sensitive data that appears in log events ingested by the log groups in your account.

When you create a data protection policy, then by default, sensitive data that matches the data identifiers you've selected is masked at all egress points, including CloudWatch Logs Insights, metric filters, and subscription filters. Only users who have the `logs:Unmask` IAM permission can view unmasked data.

You can create a data protection policy for all log groups in your account, and you can also create a data protection policies for individual log groups. When you create a policy for your entire account, it applies to both existing log groups and log groups that are created in the future.

If you create a data protection policy for your entire account and you also create a policy for a single log group, both policies apply to that log group. All managed data identifiers that are specified in either policy are audited and masked in that log group.

 Note

Masking sensitive data is supported only for log groups in the Standard log class. If you create a data protection policy for all log groups in your account, it applies only to log groups in the Standard log class. For more information about log classes, see [Log classes](#).

Each log group can have only one log group-level data protection policy, but that policy can specify many managed data identifiers to audit and mask. The limit for a data protection policy is 30,720 characters.

 Important

Sensitive data is detected and masked when it is ingested into the log group. When you set a data protection policy, log events ingested to the log group before that time are not masked.

CloudWatch Logs supports many *managed data identifiers*, which offer preconfigured data types you can select to protect financial data, personal health information (PHI), and personally identifiable information (PII). CloudWatch Logs data protection allows you to leverage pattern matching and machine learning models to detect sensitive data. For some types of managed data identifiers, the detection depends on also finding certain keywords in proximity with the sensitive data. You can also use custom data identifiers to create data identifiers tailored to your specific use case.

A metric is emitted to CloudWatch when sensitive data is detected that matches the data identifiers you select. This is the **LogEventsWithFindings** metric and it is emitted in the **AWS/Logs** namespace. You can use this metric to create CloudWatch alarms, and you can visualize it in graphs and dashboards. Metrics emitted by data protection are vended metrics and are free of charge. For more information about metrics that CloudWatch Logs sends to CloudWatch, see [Monitoring with CloudWatch metrics](#).

Each managed data identifier is designed to detect a specific type of sensitive data, such as credit card numbers, AWS secret access keys, or passport numbers for a particular country or region. When you create a data protection policy, you can configure it to use these identifiers to analyze logs ingested by the log group, and take actions when they are detected.

CloudWatch Logs data protection can detect the following categories of sensitive data by using managed data identifiers:

- Credentials, such as private keys or AWS secret access keys
- Financial information, such as credit card numbers
- Personally Identifiable Information (PII) such as driver's licenses or social security numbers
- Protected Health Information (PHI) such as health insurance or medical identification numbers
- Device identifiers, such as IP addresses or MAC addresses

For details about the types of data that you can protect, see [Types of data that you can protect](#).

Contents

- [Understanding data protection policies](#)
 - [What are data protection policies?](#)
 - [How is the data protection policy structured?](#)
 - [JSON properties for the data protection policy](#)
 - [JSON properties for a policy statement](#)
 - [JSON properties for a policy statement operation](#)
- [IAM permissions required to create or work with a data protection policy](#)
 - [Permissions required for account-level data protection policies](#)
 - [Permissions required for data protection policies for a single log group](#)
 - [Sample data protection policy](#)
- [Create an account-wide data protection policy](#)
 - [Console](#)
 - [AWS CLI](#)
 - [Data protection policy syntax for AWS CLI or API operations](#)
- [Create a data protection policy for a single log group](#)
 - [Console](#)
 - [AWS CLI](#)
 - [Data protection policy syntax for AWS CLI or API operations](#)
- [View unmasked data](#)
- [Audit findings reports](#)

- [Required key policy to send audit findings to an bucket protected by AWS KMS](#)
- [Types of data that you can protect](#)
- [CloudWatch Logs managed data identifiers for sensitive data types](#)
 - [Credentials](#)
 - [Data identifier ARNs for credential data types](#)
 - [Device identifiers](#)
 - [Data identifier ARNs for device data types](#)
 - [Financial information](#)
 - [Data identifier ARNs for financial data types](#)
 - [Protected health information \(PHI\)](#)
 - [Data identifier ARNs for protected health information data types \(PHI\)](#)
 - [Personally identifiable information \(PII\)](#)
 - [Keywords for driver's license identification numbers](#)
 - [Keywords for national identification numbers](#)
 - [Keywords for passport numbers](#)
 - [Keywords for taxpayer identification and reference numbers](#)
 - [Data identifier ARNs for personally identifiable information \(PII\)](#)
 - [Custom data identifiers](#)
 - [What are custom data identifiers?](#)
 - [Custom data identifier constraints](#)
 - [Using custom data identifiers in the console](#)
 - [Using custom data identifiers in your data protection policy](#)

Understanding data protection policies

Topics

- [What are data protection policies?](#)
- [How is the data protection policy structured?](#)

What are data protection policies?

CloudWatch Logs uses **data protection policies** to select the sensitive data for which you want to scan, and the actions that you want to take to protect that data. To select the sensitive data of interest, you use [data identifiers](#). CloudWatch Logs data protection then detects the sensitive data by using machine learning and pattern matching. To act upon data identifiers that are found, you can define **audit** and **de-identify** operations. These operations let you log the sensitive data that is found (or not found), and to mask the sensitive data when the log events are viewed.

How is the data protection policy structured?

As illustrated in the following figure, a data protection policy document includes the following elements:

- Optional policy-wide information at the top of the document
- One statement that defines the audit and de-identify actions

Only one data protection policy can be defined per CloudWatch Logs log group. The data protection policy can have one or more deny or de-identify statements, but only one audit statement.

JSON properties for the data protection policy

A data protection policy requires the following basic policy information for identification:

- **Name** – The policy name.
- **Description** (Optional) – The policy description.
- **Version** – The policy language version. The current version is 2021-06-01.
- **Statement** – A list of statements that specifies data protection policy actions.

```
{
  "Name": "CloudWatchLogs-PersonalInformation-Protection",
  "Description": "Protect basic types of sensitive data",
  "Version": "2021-06-01",
  "Statement": [
    ...
  ]
}
```

JSON properties for a policy statement

A policy statement sets the detection context for the data protection operation.

- **Sid** (Optional) – The statement identifier.
- **DataIdentifier** – The sensitive data for which CloudWatch Logs should scan. For example, name, address, or phone number.
- **Operation** – The follow-on actions, either **Audit** or **De-identify**. CloudWatch Logs performs these actions when it finds sensitive data.

```
{
  ...
  "Statement": [
    {
      "Sid": "audit-policy",
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/Address"
      ],
      "Operation": {
        "Audit": {
          "FindingsDestination": {}
        }
      }
    }
  ],
},
```

JSON properties for a policy statement operation

A policy statement sets one of the following data protection operations.

- **Audit** – Emits metrics and findings reports without interrupting logging. Strings that match increment the **LogEventsWithFindings** metric that CloudWatch Logs publishes to the **AWS/Logs** namespace in CloudWatch. You can use these metrics to create alarms.

For an example of a findings report, see [Audit findings reports](#).

For more information about metrics that CloudWatch Logs sends to CloudWatch, see [Monitoring with CloudWatch metrics](#).

- **De-identify** – Mask the sensitive data without interrupting logging.

IAM permissions required to create or work with a data protection policy

To be able to work with data protection policies for log groups, you must have certain permissions as shown in the following tables. The permissions are different for account-wide data protection policies and for data protection policies that apply to a single log group.

Permissions required for account-level data protection policies

Note

If you are performing any of these operations inside a Lambda function, the Lambda execution role and permissions boundary must also include the following permissions.

Operation	IAM permission needed	Resource
Create a data protection policy with no audit destinations	<code>logs:PutAccountPolicy</code>	*
	<code>logs:PutDataProtectionPolicy</code>	*
Create a data protection policy with CloudWatch Logs as an audit destination	<code>logs:PutAccountPolicy</code>	*
	<code>logs:PutDataProtectionPolicy</code>	*
	<code>logs:CreateLogDelivery</code>	*
	<code>logs:PutResourcePolicy</code>	*
	<code>logs:DescribeResourcePolicies</code>	*

Operation	IAM permission needed	Resource
Create a data protection policy with Firehose as an audit destination	logs:DescribeLogGroups	*
	logs:PutAccountPolicy	*
	logs:PutDataProtectionPolicy	*
	logs:CreateLogDelivery	*
	firehose:TagDeliveryStream	arn:aws:logs::deliverystream/ <i>YOUR_DELIVERY_STREAM</i>
Create a data protection policy with Amazon S3 as an audit destination	logs:PutAccountPolicy	*
	logs:PutDataProtectionPolicy	*
	logs:CreateLogDelivery	*
	s3:GetBucketPolicy	arn:aws:s3::: <i>YOUR_BUCKET</i>
	s3:PutBucketPolicy	arn:aws:s3::: <i>YOUR_BUCKET</i>
Unmask masked log events in a specified log group	logs:Unmask	arn:aws:logs::log-group:*
View an existing data protection policy	logs:GetDataProtectionPolicy	*

Operation	IAM permission needed	Resource
Delete a data protection policy	logs:DeleteAccountPolicy	*
	logs:DeleteDataProtectionPolicy	*

If any data protection audit logs are already being sent to a destination, then other policies that send logs to the same destination only need the `logs:PutDataProtectionPolicy` and `logs:CreateLogDelivery` permissions.

Permissions required for data protection policies for a single log group

Note

If you are performing any of these operations inside a Lambda function, the Lambda execution role and permissions boundary must also include the following permissions.

Operation	IAM permission needed	Resource
Create a data protection policy with no audit destinations	logs:PutDataProtectionPolicy	arn:aws:logs::log-group: YOUR_LOG_GROUP :*
Create a data protection policy with CloudWatch Logs as an audit destination	logs:PutDataProtectionPolicy	arn:aws:logs::log-group: YOUR_LOG_GROUP :*
	logs:CreateLogDelivery	*
	logs:PutResourcePolicy	*
	logs:DescribeResourcePolicies	*

Operation	IAM permission needed	Resource
	logs:DescribeLogGroups	
Create a data protection policy with Firehose as an audit destination	logs:PutDataProtectionPolicy logs:CreateLogDelivery firehose:TagDeliveryStream	arn:aws:logs::log-group: YOUR_LOG_GROUP :* * arn:aws:logs::deliverystream/ YOUR_DELIVERY_STREAM
Create a data protection policy with Amazon S3 as an audit destination	logs:PutDataProtectionPolicy logs:CreateLogDelivery s3:GetBucketPolicy s3:PutBucketPolicy	arn:aws:logs::log-group: YOUR_LOG_GROUP :* * arn:aws:s3::: YOUR_BUCKET arn:aws:s3::: YOUR_BUCKET
Unmask masked log events	logs:Unmask	arn:aws:logs::log-group: YOUR_LOG_GROUP :*
View an existing data protection policy	logs:GetDataProtectionPolicy	arn:aws:logs::log-group: YOUR_LOG_GROUP :*
Delete a data protection policy	logs:DeleteDataProtectionPolicy	arn:aws:logs::log-group: YOUR_LOG_GROUP :*

If any data protection audit logs are already being sent to a destination, then other policies that send logs to the same destination only need the `logs:PutDataProtectionPolicy` and `logs:CreateLogDelivery` permissions.

Sample data protection policy

The following sample policy allows a user to create, view, and delete data protection policies that can sending audit findings to all three types of audit destinations. It does not permit the user to view unmasked data.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLogDeliveryConfiguration",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:PutResourcePolicy",
        "logs:DescribeLogGroups",
        "logs:DescribeResourcePolicies"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowDataProtectionAndBucketConfiguration",
      "Effect": "Allow",
      "Action": [
        "logs:GetDataProtectionPolicy",
        "logs>DeleteDataProtectionPolicy",
        "logs:PutDataProtectionPolicy",
        "s3:PutBucketPolicy",
        "firehose:TagDeliveryStream",
        "s3:GetBucketPolicy"
      ],
      "Resource": [
        "arn:aws:firehose:us-east-1:111122223333:deliverystream/delivery-  
stream-name",
        "arn:aws:s3:::amzn-s3-demo-destination-bucket",

```

```
        "arn:aws:logs:us-east-1:111122223333:log-group:log-group-name:*"  
      ]  
    }  
  ]  
}
```

Create an account-wide data protection policy

You can use the CloudWatch Logs console or AWS CLI commands to create a data protection policy to mask sensitive data for all log groups in your account. Doing so affects both current log groups and log groups that you create in the future.

Important

Sensitive data is detected and masked when it is ingested into the log group. When you set a data protection policy, log events ingested to the log group before that time are not masked.

Topics

- [Console](#)
- [AWS CLI](#)

Console

To use the console to create an account-wide data protection policy

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**. It is located near the bottom of the list.
3. Choose the **Logs** tab.
4. Choose **Configure**.
5. For **Managed data identifiers**, select the types of data that you want to audit and mask for all of your log groups. You can type in the selection box to find the identifiers that you want.

We recommend that you select only the data identifiers that are relevant for your log data and your business. Choosing many types of data can lead to false positives.

For details about which types of data that you can protect, see [Types of data that you can protect](#).

6. (Optional) If you want to audit and mask other types of data by using custom data identifiers, choose **Add custom data identifier**. Then enter a name for the data type and the regular expression to use to search for that type of data in the log events. For more information, see [Custom data identifiers](#).

A single data protection policy can include up to 10 custom data identifiers. Each regular expression that defines a custom data identifier must be 200 characters or fewer.

7. (Optional) Choose one or more services to send the audit findings to. Even if you choose not to send audit findings to any of these services, the sensitive data types that you select will still be masked.
8. Choose **Activate data protection**.

AWS CLI

To use the AWS CLI to create a data protection policy

1. Use a text editor to create a policy file named `DataProtectionPolicy.json`. For information about the policy syntax, see the following section.
2. Enter the following command:

```
aws logs put-account-policy \
--policy-name TEST_POLICY --policy-type "DATA_PROTECTION_POLICY" \
--policy-document file://policy.json \
--scope "ALL" \
--region us-west-2
```

Data protection policy syntax for AWS CLI or API operations

When you create a JSON data protection policy to use in an AWS CLI command or API operation, the policy must include two JSON blocks:

- The first block must include both a `DataIdentifier` array and an `Operation` property with an `Audit` action. The `DataIdentifier` array lists the types of sensitive data that you want to mask. For more information about the available options, see [Types of data that you can protect](#).

The `Operation` property with an `Audit` action is required to find the sensitive data terms. This `Audit` action must contain a `FindingsDestination` object. You can optionally use that `FindingsDestination` object to list one or more destinations to send audit findings reports to. If you specify destinations such as log groups, Amazon Data Firehose streams, and S3 buckets, they must already exist. For an example of an audit findings report, see [Audit findings reports](#).

- The second block must include both a `DataIdentifier` array and an `Operation` property with an `Deidentify` action. The `DataIdentifier` array must exactly match the `DataIdentifier` array in the first block of the policy.

The `Operation` property with the `Deidentify` action is what actually masks the data, and it must contain the `"MaskConfig": {}` object. The `"MaskConfig": {}` object must be empty.

The following is an example of a data protection policy using only managed data identifiers. This policy masks email addresses and United States driver's licenses.

For information about policies that specify custom data identifiers, see [Using custom data identifiers in your data protection policy](#).

```
{
  "Name": "data-protection-policy",
  "Description": "test description",
  "Version": "2021-06-01",
  "Statement": [{
    "Sid": "audit-policy",
    "DataIdentifier": [
      "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
      "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
    ],
    "Operation": {
      "Audit": {
        "FindingsDestination": {
          "CloudWatchLogs": {
            "LogGroup": "EXISTING_LOG_GROUP_IN_YOUR_ACCOUNT",
          },
          "Firehose": {
            "DeliveryStream": "EXISTING_STREAM_IN_YOUR_ACCOUNT"
          }
        }
      }
    }
  ]
}
```

```

        "S3": {
            "Bucket": "EXISTING_BUCKET"
        }
    }
},
{
    "Sid": "redact-policy",
    "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
        "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
    ],
    "Operation": {
        "Deidentify": {
            "MaskConfig": {}
        }
    }
}
]
}

```

Create a data protection policy for a single log group

You can use the CloudWatch Logs console or AWS CLI commands to create a data protection policy to mask sensitive data.

You can assign one data protection policy to each log group. Each data protection policy can audit for multiple types of information. Each data protection policy can include one audit statement.

Topics

- [Console](#)
- [AWS CLI](#)

Console

To use the console to create a data protection policy

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Log groups**.

3. Choose the name of the log group.
4. Choose **Actions, Create data protection policy**.
5. For **Managed data identifiers**, select the types of data that you want to audit and mask in this log group. You can type in the selection box to find the identifiers that you want.

We recommend that you select only the data identifiers that are relevant for your log data and your business. Choosing many types of data can lead to false positives.

For details about which types of data that you can protect by using managed data identifiers, see [Types of data that you can protect](#).

6. (Optional) If you want to audit and mask other types of data by using custom data identifiers, choose **Add custom data identifier**. Then enter a name for the data type and the regular expression to use to search for that type of data in the log events. For more information, see [Custom data identifiers](#).

A single data protection policy can include up to 10 custom data identifiers. Each regular expression that defines a custom data identifier must be 200 characters or fewer.

7. (Optional) Choose one or more services to send the audit findings to. Even if you choose not to send audit findings to any of these services, the sensitive data types that you select will still be masked.
8. Choose **Activate data protection**.

AWS CLI

To use the AWS CLI to create a data protection policy

1. Use a text editor to create a policy file named `DataProtectionPolicy.json`. For information about the policy syntax, see the following section.
2. Enter the following command:

```
aws logs put-data-protection-policy --log-group-identifier "my-log-group" --policy-document file:///Path/DataProtectionPolicy.json --region us-west-2
```

Data protection policy syntax for AWS CLI or API operations

When you create a JSON data protection policy to use in an AWS CLI command or API operation, the policy must include two JSON blocks:

- The first block must include both a `DataIdentifier` array and an `Operation` property with an `Audit` action. The `DataIdentifier` array lists the types of sensitive data that you want to mask. For more information about the available options, see [Types of data that you can protect](#).

The `Operation` property with an `Audit` action is required to find the sensitive data terms. This `Audit` action must contain a `FindingsDestination` object. You can optionally use that `FindingsDestination` object to list one or more destinations to send audit findings reports to. If you specify destinations such as log groups, Amazon Data Firehose streams, and S3 buckets, they must already exist. For an example of an audit findings report, see [Audit findings reports](#).

- The second block must include both a `DataIdentifier` array and an `Operation` property with an `Deidentify` action. The `DataIdentifier` array must exactly match the `DataIdentifier` array in the first block of the policy.

The `Operation` property with the `Deidentify` action is what actually masks the data, and it must contain the `"MaskConfig": {}` object. The `"MaskConfig": {}` object must be empty.

The following is an example of a data protection policy that masks email addresses and United States driver's licenses.

```
{
  "Name": "data-protection-policy",
  "Description": "test description",
  "Version": "2021-06-01",
  "Statement": [{
    "Sid": "audit-policy",
    "DataIdentifier": [
      "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
      "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
    ],
    "Operation": {
      "Audit": {
        "FindingsDestination": {
          "CloudWatchLogs": {
```

```

        "LogGroup": "EXISTING_LOG_GROUP_IN_YOUR_ACCOUNT",
    },
    "Firehose": {
        "DeliveryStream": "EXISTING_STREAM_IN_YOUR_ACCOUNT"
    },
    "S3": {
        "Bucket": "EXISTING_BUCKET"
    }
}
}
},
{
    "Sid": "redact-policy",
    "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
        "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
    ],
    "Operation": {
        "Deidentify": {
            "MaskConfig": {}
        }
    }
}
]
}

```

View unmasked data

To view unmasked data, a user must have the `logs:Unmask` permission. Users with this permission can see the unmasked data in the following ways:

- When viewing the events in a log stream, choose **Display, Unmask**.
- Use a CloudWatch Logs Insights query that includes the **`unmask(@message)`** command. The following example query displays the 20 most recent log events in the stream, unmasked:

```

fields @timestamp, @message, unmask(@message)
| sort @timestamp desc
| limit 20

```


For more information about CloudWatch Logs Insights commands, see [CloudWatch Logs Insights language query syntax](#).

- Use a [GetLogEvents](#) or [FilterLogEvents](#) operation with the unmask parameter.

The **CloudWatchLogsFullAccess** policy includes the `logs:Unmask` permission. To grant `logs:Unmask` to a user who does not have **CloudWatchLogsFullAccess**, you can attach a custom IAM policy to that user. For more information, see [Adding permissions to a user \(console\)](#).

Audit findings reports

If you set up CloudWatch Logs data protection audit policies to write audit reports to CloudWatch Logs, Amazon S3, or Firehose, these findings reports are similar to the following example. CloudWatch Logs writes one findings report for each log event that contains sensitive data.

```
{
  "auditTimestamp": "2023-01-23T21:11:20Z",
  "resourceArn": "arn:aws:logs:us-west-2:111122223333:log-group:/aws/lambda/MyLogGroup:*",
  "dataIdentifiers": [
    {
      "name": "EmailAddress",
      "count": 2,
      "detections": [
        {
          "start": 13,
          "end": 26
        },
        {
          "start": 30,
          "end": 43
        }
      ]
    }
  ]
}
```

The fields in the report are as follows:

- The `resourceArn` field displays the log group where the sensitive data was found.

- The `dataIdentifiers` object displays information about the findings for one type of sensitive data that you are auditing.
- The `name` field identifies which type of sensitive data this section is reporting about.
- The `count` field displays the number of times this type of sensitive data appears in the log event.
- The `start` and `end` fields show where in the log event, by character count, each occurrence of the sensitive data appears.

The previous example shows a report of finding two email addresses in one log event. The first email address starts at the 13th character of the log event and ends at the 26th character. The second email address runs from the 30th character to the 43rd character. Even though this log event has two email addresses, the value of the `LogEventsWithFindings` metric is incremented only by one, because that metric counts the number of log events that contain sensitive data, not the number of occurrences of sensitive data.

Required key policy to send audit findings to an bucket protected by AWS KMS

You can protect the data in an Amazon S3 bucket by enabling either Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3) or Server-Side Encryption with KMS Keys (SSE-KMS). For more information, see [Protecting data using server-side encryption](#) in the Amazon S3 User Guide.

If you send audit findings to a bucket that is protected with SSE-S3, no additional configuration is required. Amazon S3 handles the encryption key.

If you send audit findings to a bucket that is protected with SSE-KMS, you must update the key policy for your KMS key so that the log delivery account can write to your S3 bucket. For more information about the required key policy for use with SSE-KMS, see [Amazon S3](#) in the Amazon CloudWatch Logs User Guide.

Types of data that you can protect

This section contains information about the types of data that you can protect in a CloudWatch Logs data protection policy. CloudWatch Logs managed data identifiers offer preconfigured data types for protecting financial data, personal health information (PHI), and personally identifiable information (PII). You can also use custom data identifiers to create data identifiers tailored to your specific use case.

Contents

- [CloudWatch Logs managed data identifiers for sensitive data types](#)

- [Credentials](#)
 - [Data identifier ARNs for credential data types](#)
- [Device identifiers](#)
 - [Data identifier ARNs for device data types](#)
- [Financial information](#)
 - [Data identifier ARNs for financial data types](#)
- [Protected health information \(PHI\)](#)
 - [Data identifier ARNs for protected health information data types \(PHI\)](#)
- [Personally identifiable information \(PII\)](#)
 - [Keywords for driver's license identification numbers](#)
 - [Keywords for national identification numbers](#)
 - [Keywords for passport numbers](#)
 - [Keywords for taxpayer identification and reference numbers](#)
 - [Data identifier ARNs for personally identifiable information \(PII\)](#)
- [Custom data identifiers](#)
 - [What are custom data identifiers?](#)
 - [Custom data identifier constraints](#)
 - [Using custom data identifiers in the console](#)
 - [Using custom data identifiers in your data protection policy](#)

CloudWatch Logs managed data identifiers for sensitive data types

This section contains information about the types of data that you can protect using managed data identifiers, and which countries and regions are relevant for each of those types of data.

For some types of sensitive data, CloudWatch Logs data protection scans for keywords in the proximity of the data, and finds a match only if it finds that keyword. If a keyword has to be in proximity of a particular type of data, the keyword typically has to be within 30 characters (inclusively) of the data.

If a keyword contains a space, CloudWatch Logs data protection automatically matches keyword variations that are missing the space or that contain an underscore (`_`) or hyphen (`-`) instead of the space. In some cases, CloudWatch Logs also expands or abbreviates a keyword to address common variations of the keyword.

The following tables lists the types of credential, device, financial, medical, and protected health information (PHI) that CloudWatch Logs can detect using managed data identifiers. These are in addition to certain types of data that might also qualify as personally identifiable information (PII).

Supported identifiers that are language and region independent

Identifier	Category
Address	Personal
AwsSecretKey	Credentials
CreditCardExpiration	Financial
CreditCardNumber	Financial
CreditCardSecurityCode	Financial
EmailAddress	Personal
IpAddress	Personal
LatLong	Personal
Name	Personal
OpenSshPrivateKey	Credentials
PgpPrivateKey	Credentials
PkcsPrivateKey	Credentials
PuttyPrivateKey	Credentials
VehicleIdentificationNumber	Personal

Region-dependent data identifiers must include the identifier name, then a hyphen, and then the two-letter (ISO 3166-1 alpha-2) codes. For example, `DriversLicense-US`.

Supported identifiers that must include a two-letter country or region code

Identifier	Category	Countries and languages
BankAccountNumber	Financial	DE, ES, FR, GB, IT, US
CepCode	Personal	BR
Cnpj	Personal	BR
CpfCode	Personal	BR
DriversLicense	Personal	AT, AU, BE, BG, CA, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IT, LT, LU, LV, MT, NL, PL, PT, RO, SE, SI, SK, US
DrugEnforcementAgencyNumber	Health	US
ElectoralRollNumber	Personal	GB
HealthInsuranceCardNumber	Health	EU
HealthInsuranceClaimNumber	Health	US
HealthInsuranceNumber	Health	FR
HealthcareProcedureCode	Health	US
IndividualTaxIdentificationNumber	Personal	US
InseeCode	Personal	FR
MedicareBeneficiaryNumber	Health	US
NationalDrugCode	Health	US
NationalIdentificationNumber	Personal	DE, ES, IT
NationalInsuranceNumber	Personal	GB

Identifier	Category	Countries and languages
NationalProviderId	Health	US
NhsNumber	Health	GB
NieNumber	Personal	ES
NifNumber	Personal	ES
PassportNumber	Personal	CA, DE, ES, FR, GB, IT, US
PermanentResidenceNumber	Personal	CA
PersonalHealthNumber	Health	CA
PhoneNumber	Personal	BR, DE, ES, FR, GB, IT, US
PostalCode	Personal	CA
RgNumber	Personal	BR
SocialInsuranceNumber	Personal	CA
Ssn	Personal	ES, US
TaxId	Personal	DE, ES, FR, GB
ZipCode	Personal	US

Credentials

CloudWatch Logs data protection can find the following types of credentials.

Type of data	Data identifier ID	Keyword required	Countries and regions
AWS secret access key	AwsSecretKey	aws_secret_access_key , credentials ,	All

Type of data	Data identifier ID	Keyword required	Countries and regions
		secret access key, secret key, set-aws-credential	
OpenSSH private key	OpenSSHPrivateKey	None	All
PGP private key	PgpPrivateKey	None	All
Pkcs Private Key	PkcsPrivateKey	None	All
PuTTY private key	PuttyPrivateKey	None	All

Data identifier ARNs for credential data types

The following lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

Credential data identifier ARNs

```
arn:aws:dataprotection::aws:data-identifier/AwsSecretKey
```

```
arn:aws:dataprotection::aws:data-identifier/OpenSshPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PgpPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PkcsPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PuttyPrivateKey
```

Device identifiers

CloudWatch Logs data protection can find the following types of device identifiers.

Type of data	Data identifier ID	Keyword required	Countries and regions
IP address	IpAddress	None	All

Data identifier ARNs for device data types

The following lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

Device data identifier ARN
arn:aws:dataprotection::aws:data-identifier/IpAddress

Financial information

CloudWatch Logs data protection can find the following types of financial information.

If you set a data protection policy, CloudWatch Logs scans for the data identifiers that you specify no matter what geolocation the log group is located in. The information in the **Countries and regions** column in this table designates whether two-letter country codes must be appended to the data identifier to detect the appropriate keywords for those countries and regions.

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Bank account number	BankAccountNumber	Yes. Different keywords apply to different countries. For details, see the Keywords for bank account numbers table later in this section.	France, Germany, Italy, Spain, United Kingdom, United States	Includes International Bank Account Numbers (IBANs) that

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
				consist of up to 34 alphanumeric characters, including elements such as country codes.
Credit card expiration date	CreditCardExpiration	exp d, exp m, exp y, expiration , expiry	All	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Credit card number	CreditCardNumber	account number, american express, amex, bank card, card, card number, card num, cc #, ccn, check card, credit, credit card#, dankort, debit, debit card, diners club, discover, electron, japanese card bureau, jcb, mastercard , mc, pan, payment account number, payment card number, pcn, union pay, visa	All	Detection requires the data to be a 13–19 digit sequence that adheres to the Luhn check formula, and uses a standard card number prefix for any of the following types of credit cards: American Express, Dankort, Diner's Club,

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
				Discover, Electron, Japanese Card Bureau (JCB), Mastercard, UnionPay, and Visa.
Credit card verification code	CreditCardSecurity Code	card id, card identification code, card identification number , card security code, card validation code , card validation number , card verification data , card verification value, cvc, cvc2, cvv, cvv2, elo verification code	All	

Keywords for bank account numbers

Use the following keywords to bank account numbers. This includes International Bank Account Numbers (IBANs) that consist of up to 34 alphanumeric characters, including elements such as country codes.

Country	Keywords
France	account code, account number, accountno# , accountnumber# , bban, code bancaire, compte bancaire, customer account id, customer account number, customer bank account id, iban, numéro de compte
Germany	account code, account number, accountno# , accountnumber# , bankleitzahl , bban, customer account id, customer account number, customer bank account id, geheimzahl , iban, kartennummer , kontonummer , kreditkartennummer , sepa
Italy	account code, account number, accountno# , accountnumber# , bban, codice bancario, conto bancario, customer account id, customer account number, customer bank account id, iban, numero di conto
Spain	account code, account number, accountno# , accountnumber# , bban, código cuenta, código cuenta bancaria, cuenta cliente id, customer account ID, customer account number, customer bank account id, iban, número cuenta bancaria cliente, número cuenta cliente
United Kingdom	account code, account number, accountno# , accountnumber# , bban, customer account ID, customer account number, customer bank account id, iban, sepa
United States	bank account, bank acct, checking account, checking acct, deposit account, deposit acct, savings account, savings acct, chequing account, chequing acct

CloudWatch Logs doesn't report occurrences of the following sequences, which credit card issuers have reserved for public testing.

```
1220000000000003, 2222405343248877, 2222990905257051, 2223007648726984,
2223577120017656,
```

```
30569309025904, 34343434343434, 3528000700000000, 3530111333300000, 3566002020360505,
36148900647913,
36700102000000, 371449635398431, 378282246310005, 378734493671000, 38520000023237,
401288888881881,
4111111111111111, 422222222222, 4444333322221111, 4462030000000000, 4484070000000000,
49118300000000,
4917300800000000, 4917610000000000, 4917610000000000003, 5019717010103742,
5105105105105100,
5111010030175156, 5185540810000019, 5200828282828210, 5204230080000017,
5204740009900014, 5420923878724339,
5454545454545454, 5455330760000018, 5506900490000436, 5506900490000444,
5506900510000234, 5506920809243667,
5506922400634930, 5506927427317625, 5553042241984105, 5555553753048194,
555555555554444, 5610591081018250,
6011000990139424, 6011000400000000, 6011111111111117, 630490017740292441,
630495060000000000,
6331101999990016, 6759649826438453, 6799990100000000019, and 76009244561.
```

Data identifier ARNs for financial data types

The following lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

Financial data identifier ARNs

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-DE
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-ES
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-FR
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-IT
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardExpiration
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardNumber
```

Financial data identifier ARNs

```
arn:aws:dataprotection::aws:data-identifier/CreditCardSecurityCode
```

Protected health information (PHI)

CloudWatch Logs data protection can find the following types of protected health information (PHI).

If you set a data protection policy, CloudWatch Logs scans for the data identifiers that you specify no matter what geolocation the log group is located in. The information in the **Countries and regions** column in this table designates whether two-letter country codes must be appended to the data identifier to detect the appropriate keywords for those countries and regions.

Type of data	Data identifier ID	Keyword required	Countries and regions
Drug Enforcement Agency (DEA) registration number	DrugEnforcementAgencyNumber	dea number, dea registration	United States
Health Insurance Card Number (EHIC)	HealthInsuranceCardNumber	assicurazione sanitaria numero, carta assicurazione numero, carte d'assurance maladie , carte européenne d'assurance maladie , ceam, ehic, ehic#, finlandehicnumber# , gesundheitskarte , hälsokort , health card, health card number, health insurance card, health insurance number,	European Union

Type of data	Data identifier ID	Keyword required	Countries and regions
		insurance card number, krankenversicherungskarte , krankenversicherungsnnummer , medical account number, numero conto medico, numéro d'assurance maladie , numéro de carte d'assurance , numéro de compte medical, número de cuenta médica, número de seguro de salud, número de tarjeta de seguro, sairaanho itokortin , sairausva kuutuskortti , sairausvakuutusnumero , sjukförsäkringsnummer, sjukförsäkringskort , suomi ehic-numero , tarjeta de salud, terveyskortti , tessera sanitaria assicurazione numero , versicherungsnummer	
Health Insurance Claim Number (HICN)	HealthInsuranceClaimNumber	health insurance claim number, hic no, hic no., hic number, hic#, hicn, hicn#, hicno#	United States

Type of data	Data identifier ID	Keyword required	Countries and regions
Health insurance or medical identification number	HealthInsuranceNumber	carte d'assuré social, carte vitale, insurance card	France
Healthcare Common Procedure Coding System (HCPCS) code	HealthcareProcedureCode	current procedural terminology , hcpcs, healthcare common procedure coding system	United States
Medicare Beneficiary Number (MBN)	MedicareBeneficiaryNumber	mbi, medicare beneficiary	United States
National Drug Code (NDC)	NationalDrugCode	national drug code, ndc	United States
National Provider Identifier (NPI)	NationalProviderId	hipaa, n.p.i., national provider, npi	United States
National Health Service (NHS) number	NhsNumber	national health service, NHS	Great Britain
Personal Health Number	PersonalHealthNumber	canada healthcare number, msp number, care number, phn, soins de santé	Canada

Data identifier ARNs for protected health information data types (PHI)

The following lists the data identifier Amazon Resource Names (ARNs) that can be used in protected health information (PHI) data protection policies.

PHI data identifier ARNs

```
arn:aws:dataprotection::aws:data-identifier/DrugEnforcementAgencyNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/HealthcareProcedureCode-US
```

```
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceCardNumber-EU
```

```
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceClaimNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceNumber-FR
```

```
arn:aws:dataprotection::aws:data-identifier/MedicareBeneficiaryNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/NationalDrugCode-US
```

```
arn:aws:dataprotection::aws:data-identifier/NationalInsuranceNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/NationalProviderId-US
```

```
arn:aws:dataprotection::aws:data-identifier/NhsNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/PersonalHealthNumber-CA
```

Personally identifiable information (PII)

CloudWatch Logs data protection can find the following types of personally identifiable information (PII).

If you set a data protection policy, CloudWatch Logs scans for the data identifiers that you specify no matter what geolocation the log group is located in. The information in the **Countries and**

regions column in this table designates whether two-letter country codes must be appended to the data identifier to detect the appropriate keywords for those countries and regions.

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Birth date	DateOfBirth	dob, date of birth, birthdate , birth date, birthday, b-day, bday	Any	Support includes most date formats, such as all digits and combinations of digits and names of months. Date components can be separated by spaces, slashes (/), or hyphens (-).
Código de Endereçamento Postal (CEP)	CepCode	cep, código de endereçamento	Brazil	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
		postal, código de endereçamento postal		
Cadastro Nacional da Pessoa Jurídica (CNPJ)	Cnpj	cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj	Brazil	
Cadastro de Pessoas Físicas (CPF)	CpfCode	Cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro de pessoa física, cadastro de pessoa fisica, cpf	Brazil	
Driver's license identification number	DriversLicense	Yes. Different keywords apply to different countries. For details, see the Drivers license identification numbers table later in this section.	Many countries. For details, see the Drivers license identification numbers table.	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Electoral roll number	Electoral RollNumber	electoral #, electoral number, electoral roll #, electoral roll no., electoral roll number, electoral rollno	United Kingdom	
Individual taxpayer identification	IndividualTaxIdentificationNumber	Yes. Different keywords apply to different countries. For details, see the Individual taxpayer identification numbers table later in this section.	Brazil, France, Germany Spain, United Kingdom	
National Institute for Statistics and Economic Studies (INSEE)	InseeCode	Yes. Different keywords apply to different countries. For details, see the Keywords for national identification numbers table later in this section.	France	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
National Identification Number	NationalIdentificationNumber	Yes. For details, see the Keywords for national identification numbers table later in this section.	Germany, Italy, Spain	This includes Documento Nacional de Identidad (DNI) identifiers (Spain), Codice fiscale codes (Italy), and National Identity Card numbers (German).
National Insurance Number (NINO)	NationalInsuranceNumber	insurance no., insurance number, insurance# ,national insurance number, nationalinsurance# ,nationalinsurancenumber ,nin, nino	United Kingdom	–

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Número de identidad de extranjero (NIE)	NieNumber	Yes. Different keywords apply to different countries. For details, see the Individual taxpayer identification numbers table later in this section.	Spain	
Número de Identificación Fiscal (NIF)	NifNumber	Yes. Different keywords apply to different countries. For details, see the Individual taxpayer identification numbers table later in this section.	Spain	
Passport number	PassportNumber	Yes. Different keywords apply to different countries. For details, see the Keywords for passport numbers table later in this section.	Canada, France, Germany, Italy, Spain, United Kingdom, United States	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Permanent residence number	Permanent Residence Number	carte résident permanent , numéro carte résident permanent , numéro résident permanent , permanent resident card, permanent resident card number, permanent resident no, permanent resident no., permanent resident number, pr no, pr no., pr non, pr number, résident permanent no., résident permanent non	Canada	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Phone number	PhoneNumber	Brazil: keywords also include: cel, celular, fone, móvel, número residencial , numero residencial , telefone Others: cell, contact, fax, fax number, mobile, phone, phone number, tel, telephone , telephone number	Brazil, Canada, France, Germany, Italy, Spain, United Kingdom, United States	This includes toll-free numbers in the United States and fax numbers. If a keyword is in proximity of the data, the number doesn't have to include a country code. If a keyword isn't in proximity of the data,

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
				the number has to include a country code.
Postal Code	PostalCode	None	Canada	
Registro Geral (RG)	RgNumber	Yes. Different keywords apply to different countries. For details, see the Individual taxpayer identification numbers table later in this section.	Brazil	
Social Insurance Number (SIN)	SocialInsuranceNumber	canadian id, numéro d'assurance sociale, social insurance number, sin	Canada	
Social Security Number (SSN)	Ssn	Spain – número de la seguridad social, social security no., social security no. número de la seguridad social, social security number, socialsecurityno# , ssn, ssn# United States – social security, ss#, ssn	Spain, United States	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Taxpayer identification or reference number	TaxId	Yes. Different keywords apply to different countries. For details, see the Individual taxpayer identification numbers table later in this section.	France, Germany, Spain, United Kingdom	This includes TIN (France); Steueridentifikationsnummer (Germany); CIF (Spain); and TRN, UTR (United Kingdom).
ZIP code	ZipCode	zip code, zip+4	United States	United States postal code.

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Mailing address	Address	None	Australia, Canada, France, Germany, Italy, Spain, United Kingdom, United States	Although a keyword isn't required, detection requires the address to include the name of a city or place and a ZIP code or Postal Code.
Electronic mail address	EmailAddress	None	Any	

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Global Positioning System (GPS) coordinates	LatLong	coordinate , coordinates , lat long, latitude longitude , location, position	Any	CloudWatch Logs can detect GPS coordinates if the latitude and longitude coordinates are stored as a pair and they're in Decimal Degrees (DD) format, for example, 41.948614,-87.655311. Support doesn't include coordinates

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
				es in Degrees Decimal Minutes (DDM) format, for example 41°56.916 8'N 87°39.318 7'W, or Degrees, Minutes, Seconds (DMS) format, for example 41°56'55.0104"N 87°39'19.1196"W.

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Full name	Name	None	Any	CloudWatch Logs can detect full names only. Support is limited to Latin character sets.

Type of data	Data identifier ID	Keyword required	Countries and regions	Notes
Vehicle Identification Number (VIN)	VehicleIdentificationNumber	Fahrgestellnummer , niv, numarul de identificare , numarul seriei de sasiu, serie sasiu, numer VIN, Número de Identificação do Veículo, Número de Identificación de Automóviles , numéro d'identification du véhicule, vehicle identification number, vin, VIN numeris	Any	CloudWatch Logs can detect VINs that consist of a 17-character sequence and adhere to the ISO 3779 and 3780 standards . These standards were designed for worldwide use.

Keywords for driver's license identification numbers

To detect various types of driver's license identification numbers, CloudWatch Logs requires a keyword to be in proximity of the numbers. The following table lists the keywords that CloudWatch Logs recognizes for specific countries and regions.

Country or region	Keywords
Australia	dl# dl:, dl :, dlno# driver licence, driver license, driver permit, drivers lic., drivers licence, driver's licence, drivers license, driver's license, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
Austria	führerschein, fuhrerschein, führerschein republik österreich, fuhrerschein republik osterreich
Belgium	fuehrerschein, fuehrerschein- nr, fuehrersc heinnummer, fuhrerschein, führerschein, fuhrerschein- nr, führerschein- nr, fuhrersch einnummer, führerscheinnummer, numéro permis conduire, permis de conduire, rijbewijs, rijbewijsnummer
Bulgaria	превозно средство, свидетелство за управление на моторно, свидетелство за управление на мпс, сумпс, шофьорска книжка
Canada	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit,

Country or region	Keywords
	drivers permit number, driving licence, driving license, driving permit, permis de conduire
Croatia	vozačka dozvola
Cyprus	άδεια οδήγησης
Czech Republic	číslo licence, číslo licence řidiče, číslo řidičské o průkazu, ovladače lic., povolení k jízdě, povolení řidiče, řidiči povolení, řidičský průkaz, řidičský průkaz
Denmark	kørekort, kørekortnummer
Estonia	juhi litsentsi number, juhiloa number, juhiluba, juhiluba number
Finland	ajokortin numero, ajokortti, förare lic., körkort, körkort nummer, kuljettaja lic., permis de conduire
France	permis de conduire
Germany	fuehrerschein, fuehrerschein- nr, fuehrerscheinnummer, fuhrerschein, fuhrerschein, fuhrerschein- nr, fuhrerschein- nr, fuhrersch einnummer, fuhrerscheinnummer
Greece	δεια οδήγησης, adeia odigisis
Hungary	illesztőprogramok lic, jogosítvány, jogsí, licensszám, vezető engedély, vezetői engedély
Ireland	ceadúnas tiomána
Italy	patente di guida, patente di guida numero, patente guida, patente guida numero

Country or region	Keywords
Latvia	autovadītāja apliecība, licences numurs, vadītāja apliecība, vadītāja apliecības numurs, vadītāja atļauja, vadītāja licences numurs, vadītāji lic.
Lithuania	vairuotojo pažymėjimas
Luxembourg	fahrerlaubnis, führerschein
Malta	licenzja tas-sewqan
Netherlands	permis de conduire, rijbewijs, rijbewijsnummer
Poland	numer licencyjny, prawo jazdy, zezwolenie na prowadzenie
Portugal	carta de condução, carteira de habilitação, carteira de motorist, carteira habilitação, carteira motorist, licença condução, licença de condução, número de licença, número licença, permissão condução, permissão de condução
Romania	numărul permisului de conducere, permis de conducere
Slovakia	číslo licencie, číslo vodičského preukazu, ovládače lic., povolenia vodičov, povolenie jazdu, povolenie na jazdu, povolenie vodiča, vodičský preukaz
Slovenia	vozniško dovoljenje

Country or region	Keywords
Spain	carnet conducir, el carnet de conducir, licencia conducir, licencia de manejo, número carnet conducir, número de carnet de conducir, número de permiso conducir, número de permiso de conducir, número licencia conducir, número permiso conducir, permiso conducción, permiso conducir, permiso de conducción
Sweden	ajokortin numero, dlno# ajokortti, drivere lic., förare lic., körkort, körkort nummer, körkortsn ummer, kuljettajat lic.
United Kingdom	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
United States	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit

Keywords for national identification numbers

To detect various types of national identification numbers, CloudWatch Logs requires a keyword to be in close proximity to the numbers. This includes Documento Nacional de Identidad (DNI)

identifiers (Spain), French National Institute for Statistics and Economic Studies (INSEE) codes, German National Identity Card numbers, and Registro Geral (RG) numbers (Brazil).

The following table lists the keywords that CloudWatch Logs recognizes for specific countries and regions.

Country or region	Keywords
Brazil	registro geral, rg
France	assurance sociale, carte nationale d'identité, cni, code sécurité sociale, French social security number, fssn#, insee, insurance number, national id number, nationalid#, numéro d'assurance, sécurité sociale, sécurité sociale non., sécurité sociale numéro, social, social security, social security number, socialsecuritynumber, ss#, ssn, ssn#
Germany	ausweisnummer, id number, identification number, identity number, insurance number, personal id, personalausweis
Italy	codice fiscale, dati anagrafici, ehic, health card, health insurance card, p. iva, partita i.v.a., personal data, tax code, tessera sanitaria
Spain	dni, dni#, dninúmero#, documento nacional de identidad, identidad único, identidad único#, insurance number, national identification number, national identity, nationalid#, nationalidno#, número nacional identidad, personal identification number, personal identity no, unique identity number, uniqueid#

Keywords for passport numbers

To detect various types of passport numbers, CloudWatch Logs requires a keyword to be in proximity of the numbers. The following table lists the keywords that CloudWatch Logs recognizes for specific countries and regions.

Country or region	Keywords
Canada	paspassport, paspassport#, passport, passport#, passportno, passportno#
France	numéro de paspassport, paspassport, paspassport #, paspassport #, paspassportn °, paspassport n °, paspassportNon, paspassport non
Germany	ausstellungsdatum, ausstellungsort, geburtsdatum, passport, passports, reisepass, reisepass-nr, reisepassnummer
Italy	italian passport number, numéro paspassport , numéro paspassport italien, passaporto, passaporto italiana, passaporto numero, passport number, repubblica italiana passaporto
Spain	españa pasaporte, libreta pasaporte, número pasaporte, pasaporte, passport, passport book, passport no, passport number, spain passport
United Kingdom	paspassport #, paspassport n °, paspassportNon, paspassport non, paspassportn °, passport #, passport no, passport number, passport#, passportid
United States	passport, travel document

Keywords for taxpayer identification and reference numbers

To detect various types of taxpayer identification and reference numbers, CloudWatch Logs requires a keyword to be in proximity of the numbers. The following table lists the keywords that CloudWatch Logs recognizes for specific countries and regions.

Country or region	Keywords
Brazil	cadastro de pessoa física, cadastro de pessoa física, cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro nacional da pessoa jurídica, cadastro nacional da pessoa jurídica, cnpj, cpf
France	numéro d'identification fiscale, tax id, tax identification number, tax number, tin, tin#
Germany	identifikationsnummer, steuer id, steueridentifikationsnummer, steuernummer, tax id, tax identification number, tax number
Spain	cif, cif número, cifnúmero#, nie, nif, número de contribuyente, número de identidad de extranjero, número de identificación fiscal, número de impuesto corporativo, personal tax number, tax id, tax identification number, tax number, tin, tin#
United Kingdom	paye, tax id, tax id no., tax id number, tax identification, tax identification#, tax no., tax number, tax reference, tax#, taxid#, temporary reference number, tin, trn, unique tax reference, unique taxpayer reference, utr
United States	individual taxpayer identification number, itin, i.t.i.n.

Data identifier ARNs for personally identifiable information (PII)

The following table lists the Amazon Resource Names (ARNs) for the personally identifiable information (PII) data identifiers that you can add to your data protection policies.

PII data identifier ARNs

```
arn:aws:dataprotection::aws:data-identifier/Address
```

```
arn:aws:dataprotection::aws:data-identifier/CepCode-BR
```

```
arn:aws:dataprotection::aws:data-identifier/Cnpj-BR
```

```
arn:aws:dataprotection::aws:data-identifier/CpfCode-BR
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-AT
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-AU
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-BE
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-BG
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CA
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CY
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CZ
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-DE
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-DK
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-EE
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-ES
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-FI
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-FR
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-GB
```

PII data identifier ARNs

arn:aws:dataprotection::aws:data-identifier/DriversLicense-GR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-HR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-HU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-IE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-IT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LV

arn:aws:dataprotection::aws:data-identifier/DriversLicense-MT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-NL

arn:aws:dataprotection::aws:data-identifier/DriversLicense-PL

arn:aws:dataprotection::aws:data-identifier/DriversLicense-PT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-RO

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SI

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SK

arn:aws:dataprotection::aws:data-identifier/DriversLicense-US

arn:aws:dataprotection::aws:data-identifier/ElectoralRollNumber-GB

arn:aws:dataprotection::aws:data-identifier/EmailAddress

PII data identifier ARNs

arn:aws:dataprotection::aws:data-identifier/IndividualTaxIdentificationNumber-US

arn:aws:dataprotection::aws:data-identifier/InseeCode-FR

arn:aws:dataprotection::aws:data-identifier/LatLong

arn:aws:dataprotection::aws:data-identifier/Name

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-DE

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-ES

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-IT

arn:aws:dataprotection::aws:data-identifier/NieNumber-ES

arn:aws:dataprotection::aws:data-identifier/NifNumber-ES

arn:aws:dataprotection::aws:data-identifier/PassportNumber-CA

arn:aws:dataprotection::aws:data-identifier/PassportNumber-DE

arn:aws:dataprotection::aws:data-identifier/PassportNumber-ES

arn:aws:dataprotection::aws:data-identifier/PassportNumber-FR

arn:aws:dataprotection::aws:data-identifier/PassportNumber-GB

arn:aws:dataprotection::aws:data-identifier/PassportNumber-IT

arn:aws:dataprotection::aws:data-identifier/PassportNumber-US

arn:aws:dataprotection::aws:data-identifier/PermanentResidenceNumber-CA

PII data identifier ARNs

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-BR

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-DE

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-ES

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-FR

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-GB

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-IT

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US

arn:aws:dataprotection::aws:data-identifier/PostalCode-CA

arn:aws:dataprotection::aws:data-identifier/RgNumber-BR

arn:aws:dataprotection::aws:data-identifier/SocialInsuranceNumber-CA

arn:aws:dataprotection::aws:data-identifier/Ssn-ES

arn:aws:dataprotection::aws:data-identifier/Ssn-US

arn:aws:dataprotection::aws:data-identifier/TaxId-DE

arn:aws:dataprotection::aws:data-identifier/TaxId-ES

arn:aws:dataprotection::aws:data-identifier/TaxId-FR

arn:aws:dataprotection::aws:data-identifier/TaxId-GB

arn:aws:dataprotection::aws:data-identifier/VehicleIdentificationNumber

arn:aws:dataprotection::aws:data-identifier/ZipCode-US

Custom data identifiers

Topics

- [What are custom data identifiers?](#)
- [Custom data identifier constraints](#)
- [Using custom data identifiers in the console](#)
- [Using custom data identifiers in your data protection policy](#)

What are custom data identifiers?

Custom data identifiers (CDIs) let you define your own custom regular expressions that can be used in your data protection policy. Using custom data identifiers, you can target business-specific personally identifiable information (PII) use cases that [managed data identifiers](#) can't provide. For example, you can use a custom data identifier to look for company-specific employee IDs. Custom data identifiers can be used in conjunction with managed data identifiers.

Custom data identifier constraints

CloudWatch Logs custom data identifiers have the following limitations:

- A maximum of 10 custom data identifiers are supported for each data protection policy.
- Custom data identifier names have a maximum length of 128 characters. The following characters are supported:
 - Alphanumeric: (a-zA-Z0-9)
 - Symbols: ('_' | '-')
- RegEx has a maximum length of 200 characters. The following characters are supported:
 - Alphanumeric: (a-zA-Z0-9)
 - Symbols: ('_' | '#' | '=' | '@' | '/' | ';' | ':' | '-' | '')
 - RegEx reserved characters: ('^' | '\$' | '?' | '[' | ']' | '{' | '}' | '|' | '\\' | '*' | '+' | '!')
- Custom data identifiers cannot share the same name as a managed data identifier.
- Custom data identifiers can be specified within an account-level data protection policy or in log group-level data protection policies. Similar to managed data identifiers, custom data identifiers defined within an account-level policy work in combination with custom data identifiers defined in a log group-level policy.

Using custom data identifiers in the console

When you use the CloudWatch console to create or edit a data protection policy, to specify a custom data identifier you just enter a name and regular expression for the data identifier. For example, you might enter **Employee_ID** for the name and **EmployeeID-\d{9}** as the regular expression. This regular expression will detect and mask log events with nine numbers after EmployeeID-. For example, EmployeeID-123456789

Using custom data identifiers in your data protection policy

If you are using the AWS CLI or AWS API to specify a custom data identifier, you need to include the data identifier name and regular expression in the JSON policy used to define the data protection policy. The following data protection policy detects and masks log events that carry company-specific employee IDs.

1. Create a Configuration block within your data protection policy.
2. Enter a Name for your custom data identifier. For example, **EmployeeId**.
3. Enter a Regex for your custom data identifier. For example, **EmployeeID-\d{9}**. This regular expression will match log events containing EmployeeID- that have nine digits after EmployeeID-. For example, EmployeeID-123456789
4. Refer to the following custom data identifier in a policy statement.

```
{
  "Name": "example_data_protection_policy",
  "Description": "Example data protection policy with custom data identifiers",
  "Version": "2021-06-01",
  "Configuration": {
    "CustomDataIdentifier": [
      {
        "Name": "EmployeeId",
        "Regex": "EmployeeId-\\d{9}"
      }
    ]
  },
  "Statement": [
    {
      "Sid": "audit-policy",
      "DataIdentifier": [
        "EmployeeId"
      ],
      "Operation": {
        "Audit": {
          "FindingsDestination": {
            "S3": {
```

```

        "Bucket": "EXISTING_BUCKET"
      }
    }
  },
  {
    "Sid": "redact-policy",
    "DataIdentifier": [
      "EmployeeId"
    ],
    "Operation": {
      "Deidentify": {
        "MaskConfig": {
        }
      }
    }
  }
]
}

```

5. (Optional) Continue to add additional **custom data identifiers** to the Configuration block as needed. Data protection policies currently support a maximum of 10 custom data identifiers.

Transform logs during ingestion

With logs transformation and enrichment, you can normalize all your logs in a consistent and context-rich format at the time of ingestion into CloudWatch Logs. You can add structure to your logs by using pre-configured templates for common AWS services such as AWS WAF and Amazon Route 53, or build custom transformers with native parsers such as [Grok](#). You can also rename existing attributes and add additional metadata to your logs such as account ID, and Region.

Log transformation helps simplify and shorten your log queries across your applications, and helps simplify creating alerts on your logs. This feature provides transformation for common log types with out-of-the-box transformation templates for major AWS log sources like VPC Flow logs, Route 53, and Amazon RDS for PostgreSQL. You can use pre-configured transformation templates or create custom transformers to suit your needs.

Log transformation helps you manage logs emitted from various sources that vary widely in format and attribute names.

After you create a transformer, ingested log events are converted and stored in a standard format. You can leverage these transformed logs to accelerate your analytics experience with the following features:

- [Field indexes](#)
- [CloudWatch Logs Insights discovered fields](#)
- Flexibility in alarming using [metric filters](#)
- Forwarding via [subscription filters](#)
- Creating metric data from log events with [Contributor Insights](#), where you can choose to have the Contributor Insights rule evaluate log events either before or after they are transformed.

Transformations happen only during log ingestion. You can't transform log events that have already been ingested. Transformations are not reversible. Both original and transformed logs are stored in CloudWatch Logs with the same retention policy. Log transformation and enrichment capability is included in the existing Standard log class ingestion price. Log storage costs will be based on log size after transformation, which might exceed the original log volume.

⚠ Important

After log events have been transformed, you must use CloudWatch Logs Insights queries to view the transformed versions of the logs. The [GetLogEvents](#) and [FilterLogEvents](#) actions return only the original versions of the log events, before they were transformed.

⚠ Important

Despite a single log event in PutLogEvents allowing upto 1MB, logs transformation can only handle log event of size less than 512kb. Any log events greather than 512kb will fail in transformation and emit an error. Total size of PutLogEvents can still go over 512kb.

In addition to transforming into different formats, you can also enrich your logs with additional context, such as account ID, Region, and keyword. These are extracted from the log group name and from static keywords.

Log transformation helps you with logs emitted from various sources that vary widely in format and attribute names.

Log transformation and enrichment is supported only for log groups in the Standard log class.

You can create transformers for individual log groups, and you can also create account-level transformers that apply to all or many log groups in your account. If a log group has a log group-level transformer, that transformer overrides any account-level transformer that would otherwise apply to that log group.

Topics

- [Create and manage log transformers](#)
- [Configurable parser-type processors](#)
- [Built-in processors for AWS vended logs](#)
- [String mutate processors](#)
- [JSON mutate processors](#)
- [Datatype converter processors](#)
- [Transformation metrics and errors](#)

Create and manage log transformers

A log transformer includes one or more *processors* that are in a logical pipeline together. Each processor is applied to a log event, one after the other in the order that they are listed in the transformer configuration.

Some processors are of the *parser* type. Each transformer must have at least one parser, and the first processor in a transformer must be a parser.

Some of the parsers are built-in parsers that are configured for a certain type of AWS vended log.

Other processor types are string mutators, JSON mutators, and data processors.

You can create transformers for individual log groups, and you can also create account-level transformers that apply to all or many log groups in your account. If a log group has a log group-level transformer, that transformer overrides any account-level transformer that would otherwise apply to that log group. You can have as many as 20 account-level transformers in a Region in your account.

You must follow these guidelines when you create a transformer:

- If you include a pre-configured parser for a type of AWS vended logs, it must be the first processor listed in the transformer. You can include only one such processor in a transformer.
- You can include only one `grok` processor in a transformer.
- You must have at least one parser-type processor in a transformer. You can include as many as five parser-type processors. This limit of five includes both built-in parsers and configurable parsers.
- You can have as many as 20 processors in a transformer.
- You can include only one **addKeys** processor in a transformer.
- You can include only one **copyValue** processor in a transformer.
- Each transformer can extract up to 200 fields from a log event.
- Each log event **MUST** be below 512KB. Total size of log events can still go over 512KB.

Topics

- [Create an account-level transformer policy](#)
- [Edit or delete an account-level transformer policy](#)

- [Create a log-group-level log transformer from scratch](#)
- [Create a log-group-level transformer by copying an existing one](#)
- [Edit a log-group-level transformer](#)
- [Delete a log-group-level transformer](#)

Create an account-level transformer policy

Use the steps in this section to create a transformer policy that applies to all log groups in the account, or to multiple log groups that have log group names that start with the same string (prefix). You can have as many as 20 account-level transformer policies in a Region.

You can't create two transformer policies in the same Region that use the same prefix or have one prefix contained within another. For example, if you create one transformer policy for the string prefix `/aws/lambda`, you can't create another with the prefix `/aws`. But you could have one transformer for `/aws/lambda` and another for `/aws/waf`.

To create an account-level transformer policy

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Settings** and then choose the **Logs** tab.
3. In the **Transformer policy for account** section, choose **Create transformer policy**.
4. For **Transformer policy name**, enter a name for your new policy.
5. For **Select log groups**, do one of the following:
 - Choose **All standard log groups** to have the transformer policy apply to all Standard Class log groups in the account.
 - choose **Log groups by prefix match** to apply the policy to a subset of log groups that all have names that start with the same string. Then, enter the prefix for these log groups in **Selection criteria**.
6. In the **Select parsers** area, use **Parsers** to select a parser to include in your transformer.

If it is a pre-configured parser for a type of AWS vended log, you don't have to specify any configuration for it.

If it is a different parser, you need to specify its configuration. For more information, see the information for that processor in [Configurable parser-type processors](#).

7. To add another processor, choose **Select processor**. Then select the processor that you want in the **Processor** box, and fill in the configuration parameters.

Remember that processors operate on the log events in the order that you add them to the transformer.

8. (Optional) To add additional processors, choose **+ Processor** and repeat the previous step.
9. (Optional) At any time, you can test the transformer that you have built so far on a sample log event. To do so, do one of the following in the **Transformer preview** section:
 - Select as many as five log groups in **Select log groups** and then choose **Load latest log events**. Then choose **Test transformer**.
 - Copy log events directly into **Sample log events** and then choose **Test transformer**.

The transformed version of the log then appears.

10. When you are finished adding processors and satisfied with the tests on sample logs, choose **Save**.
11. When you have finished, choose **Create**.

Edit or delete an account-level transformer policy

Use the steps in this section to edit or delete an account-level transformer policy.

To edit or delete an account-level transformer policy

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Settings** and then choose the **Logs** tab.
3. In the **Transformer account policy** section, choose **Manage**.
4. Select the button by the transformer policy that you want to manage, and then choose **Edit** or **Delete**.

If you're editing the policy, see steps 5-11 in [Configurable parser-type processors](#) to see your options.

Create a log-group-level log transformer from scratch

Use these steps to create a log-group-level transformer from scratch.

To use the console to create a log transformer for a log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Log groups**.
3. Choose the log group that you want to create the transformer for.
4. Choose the **Transformer** tab. You might have to scroll the tab list to the right to see it.
5. Choose **Create transformer**.
6. In the **Choose a parser** box, select a parser to include in your transformer.

If it is a pre-configured parser for a type of AWS vended log, you don't have to specify any configuration for it.

If it is a different parser, you need to specify its configuration. For more information, see the information for that processor in [Configurable parser-type processors](#).

7. To add another processor, choose **+ Add processor**. Then select the processor that you want in the **Choose processors** box, and fill in the configuration parameters.

Remember that processors operate on the log events in the order that you add them to the transformer.

8. (Optional) At any time, you can test the transformer that you have built so far on a sample log event. To do so, do the following:
 - In the **Transformation preview** section, either choose **Load sample log** to load a sample log event from the log group that this transformer is for, or paste a log event into the text box.

Choose **Test transformer**. The transformed version of the log appears

9. When you are finished adding processors and satisfied with the tests on sample logs, choose **Save**.

To use the AWS CLI to create a log transformer from scratch

- Use the `aws logs put-transformer` command. When using `parseJSON` as the first processor, you must parse the entire log event using `@message` as the source field. After the initial JSON parsing, you can then manipulate specific fields in subsequent processors. The following is an example that creates a transformer that includes the `parseJSON` and `addKeys` processors:

```
aws logs put-transformer \  
  --transformer-config '[{"parseJSON":{"source":"@message"}}, {"addKeys":  
{"entries":[{"key":"metadata.transformed_in", "value":"CloudWatchLogs"},  
{"key":"feature", "value":"Transformation"}]}], {"trimString":{"withKeys":  
["status"]}]}' \  
  --log-group-identifier my-log-group-name
```

Create a log-group-level transformer by copying an existing one

You can use the console to copy the JSON configuration of an existing transformer. You can then use that code to create an identical transformer by using the AWS CLI, or you can modify the configuration first.

To create a log transformer by copying an existing one

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Log groups**.
3. Choose the log group that has the transformer that you want to copy.
4. Choose the **Transformations** tab. You might have to scroll the tab list to the right to see it.
5. Choose **Manage transformer**.
6. Choose **Copy transformer**. This copies the transformer JSON to your clipboard.
7. Create a file and paste in the transformer configuration. In this example we'll call the file `CopiedTransformer.json`
8. Use the AWS CLI to create a new transformer with that configuration.

```
aws logs put-transformer --log-group-identifier my-log-group-name \  
  --transformer-config file://CopiedTransformer.json
```

Edit a log-group-level transformer

Use these steps to edit an existing log transformer.

To edit a log transformer

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Logs, Log groups**.
3. Choose the log group that has the transformer that you want to edit.
4. Choose the **Transformations** tab. You might have to scroll the tab list to the right to see it.
5. Choose **Manage transformer**.
6. In the **Parsers** and **Processors** sections, make your changes.
7. To add another processor, choose **+ Add Processor**. Then select the processor that you want in the **Processor** box, and fill in the configuration parameters.

Remember that processors operate on the log events in the order that you add them to the transformer.

8. (Optional) At any time, you can test the transformer that you have built so far on a sample log event. To do so, do the following:
 - In the **Transformation Preview** section, either choose **Load Sample Log** to load a sample log event from the log group that this transformer is for, or paste a log event into the text box.

Choose **Test Transformation**. The transformed version of the log appears

9. When you are finished adding processors and satisfied with the tests on sample logs, choose **Save**.

Delete a log-group-level transformer

Use these steps to delete a log transformer.

To delete a log transformer

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs, Log groups**.
3. Choose the log group that has the transformer that you want to edit.
4. Choose the **Transformations** tab. You might have to scroll the tab list to the right to see it.
5. Choose **Delete**.
6. In the confirmation box, choose **Delete Policy**.

Configurable parser-type processors

This section contains information about the configurable data parser processors that you can use in a log event transformer.

Contents

- [parseJSON](#)
- [grok](#)
 - [Grok examples](#)
 - [Example 1: Use grok to extract a field from unstructured logs](#)
 - [Example 2: Use grok in combination with parseJSON to extract fields from a JSON log event](#)
 - [Example 3: Grok pattern with dotted annotation in FIELD_NAME](#)
 - [Supported grok patterns](#)
 - [Common log format examples](#)
 - [Apache log example](#)
 - [NGINX log example](#)
 - [Syslog Protocol \(RFC 5424\) log example](#)
- [csv](#)
- [parseKeyValue](#)

parseJSON

The **parseJSON** processor parses JSON log events and inserts extracted JSON key-value pairs under the destination. If you don't specify a destination, the processor places the key-value pair under the root node. When using parseJSON as the first processor, you must parse the entire log event using @message as the source field. After the initial JSON parsing, you can then manipulate specific fields in subsequent processors.

The original @message content is not changed, the new keys are added to the message.

Field	Description	Required?	Default	Limits
source	Path to the field in the log event that will be parsed. Use dot	No	@message	Maximum length: 128

Field	Description	Required?	Default	Limits
	notation to access child fields. For example, <code>store.book</code>			Maximum nested key depth: 3
destination	The destination field of the parsed JSON	No	Parent JSON node	Maximum length: 128 Maximum nested key depth: 3

Example

Suppose an ingested log event looks like this:

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

Then if we have this **parseJSON** processor:

```
[
  {
    "parseJSON": {
      "destination": "new_key"
    }
  }
]
```

The transformed log event would be the following.

```
{
  "new_key": {
    "outer_key": {
      "inner_key": "inner_value"
    }
  }
}
```

grok

Use the grok processor to parse and structure unstructured data using pattern matching. This processor can also extract fields from log messages.

Field	Description	Required	Default	Limits	Notes
source	Path of the field to apply Grok matching on	No	@message	Maximum length: 128 Maximum nested key depth: 3	
match	The grok pattern to match against the log event	Yes		Maximum length: 512 Maximum grok patterns: 20 Some grok pattern types have individual usage limits. Any combination of the following patterns can be used as many as five times: {URI, URIPARAM, URIPATHPARAM, SPACE, DATA, GREEDYDATA, GREEDYDATA_MULTILINE} Grok patterns don't support type conversions.	See all supported Grok patterns

Field	Description	Required	Default	Limits	Notes
				For common log format patterns (APACHE_ACCESS_LOG, NGINX_ACCESS_LOG, SYSLOG5424), only DATA, GREEDYDATA, or GREEDYDATA_MULTILINE patterns are supported to be included after the common log pattern.	

Structure of a Grok Pattern

This is the supported grok pattern structure:

```
%{PATTERN_NAME:FIELD_NAME}
```

- **PATTERN_NAME:** Refers to a pre-defined regular expression for matching a specific type of data. Only predefined [grok patterns](#) are supported. Creating custom patterns is not allowed.
- **FIELD_NAME:** Assigns a name to the extracted value. FIELD_NAME is optional, but if you don't specify this value then the extracted data will be dropped from the transformed log event. If FIELD_NAME uses dotted notation (e.g., "parent.child"), it is considered as a JSON path.
- **Type Conversion:** Explicit type conversions are not supported. Use [TypeConverter processor](#) to convert the datatype of any value extracted by grok.

To create more complex matching expressions, you can combine several grok patterns. As many as 20 grok patterns can be combined to match a log event. For example, this combination of patterns

`%{NUMBER:timestamp} [%{NUMBER:db} %{IP:client_ip}:%{NUMBER:client_port}]`
`%{GREEDYDATA:data}` can be used to extract fields from a Redis slow log entry like this:

```
1629860738.123456 [0 127.0.0.1:6379] "SET" "key1" "value1"
```

Grok examples

Example 1: Use grok to extract a field from unstructured logs

Sample log:

```
293750 server-01.internal-network.local OK "[Thread-000] token generated"
```

Transformer used:

```
[
  {
    "grok": {
      "match": "%{NUMBER:version} %{HOSTNAME:hostname} %{NOTSPACE:status}
%{QUOTEDSTRING:logMsg}"
    }
  }
]
```

Output:

```
{
  "version": "293750",
  "hostname": "server-01.internal-network.local",
  "status": "OK",
  "logMsg": "[Thread-000] token generated"
}
```

Sample log:

```
23/Nov/2024:10:25:15 -0900 172.16.0.1 200
```

Transformer used:

```
[
```

```
{
  "grok": {
    "match": "%{HTTPDATE:timestamp} %{IPORHOST:clientip}
%{NUMBER:response_status}"
  }
}
```

Output:

```
{
  "timestamp": "23/Nov/2024:10:25:15 -0900",
  "clientip": "172.16.0.1",
  "response_status": "200"
}
```

Example 2: Use grok in combination with parseJSON to extract fields from a JSON log event

Sample log:

```
{
  "timestamp": "2024-11-23T16:03:12Z",
  "level": "ERROR",
  "logMsg": "GET /page.html HTTP/1.1"
}
```

Transformer used:

```
[
  {
    "parseJSON": {}
  },
  {
    "grok": {
      "source": "logMsg",
      "match": "%{WORD:http_method} %{NOTSPACE:request} HTTP/
%{NUMBER:http_version}"
    }
  }
]
```

Output:

```
{
  "timestamp": "2024-11-23T16:03:12Z",
  "level": "ERROR",
  "logMsg": "GET /page.html HTTP/1.1",
  "http_method": "GET",
  "request": "/page.html",
  "http_version": "1.1"
}
```

Example 3: Grok pattern with dotted annotation in FIELD_NAME

Sample log:

```
192.168.1.1 GET /index.html?param=value 200 1234
```

Transformer used:

```
[
  {
    "grok": {
      "match": "%{IP:client.ip} %{WORD:method} %{URIPATHPARAM:request.uri}
%{NUMBER:response.status} %{NUMBER:response.bytes}"
    }
  }
]
```

Output:

```
{
  "client": {
    "ip": "192.168.1.1"
  },
  "method": "GET",
  "request": {
    "uri": "/index.html?param=value"
  },
  "response": {
    "status": "200",
    "bytes": "1234"
  }
}
```

Supported grok patterns

The following tables list the patterns that are supported by the grok processor.

General grok patterns

Grok Pattern	Description	Maximum pattern limit	Example
USERNAME or USER	Matches one or more characters that can include lowercase letters (a-z), uppercase letters (A-Z), digits (0-9), dots (.), underscores (_), or hyphens (-)	20	Input: user123.name-TEST Pattern: %{USERNAME:name} Output: {"name": "user123.name-TEST"}
INT	Matches an optional plus or minus sign followed by one or more digits.	20	Input: -456 Pattern: %{INT:num} Output: {"num": "-456"}
BASE10NUM	Matches an integer or a floating-point number with optional sign and decimal point	20	Input: -0.67 Pattern: %{BASE10NUM:num} Output: {"num": "-0.67"}
BASE16NUM	Matches decimal and hexadecimal numbers with an optional sign (+ or -) and an optional 0x prefix	20	Input: +0xA1B2 Pattern: %{BASE16NUM:num} Output: {"num": "+0xA1B2"}
POSINT	Matches whole positive integers without leading	20	Input: 123

Grok Pattern	Description	Maximum pattern limit	Example
	zeros, consisting of one or more digits (1-9 followed by 0-9)		Pattern: <code>%{POSINT:num}</code> Output: <code>{"num": "123"}</code>
NONNEGINT	Matches any whole numbers (consisting of one or more digits 0-9) including zero and numbers with leading zeros.	20	Input: 007 Pattern: <code>%{NONNEGINT:num}</code> Output: <code>{"num": "007"}</code>
WORD	Matches whole words composed of one or more word characters (<code>\w</code>), including letters, digits, and underscores	20	Input: user_123 Pattern: <code>%{WORD:user}</code> Output: <code>{"user": "user_123"}</code>
NOTSPACE	Matches one or more non-whitespace characters.	5	Input: hello_world123 Pattern: <code>%{NOTSPACE:msg}</code> Output: <code>{"msg": "hello_world123"}</code>
SPACE	Matches zero or more whitespace characters.	5	Input: " " Pattern: <code>%{SPACE:extra}</code> Output: <code>{"extra": " "}</code>
DATA	Matches any character (except newline) zero or more times, non-greedy.	5	Input: abc def ghi Pattern: <code>%{DATA:x}</code> <code>%{DATA:y}</code> Output: <code>{"x": "abc", "y": "def ghi"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
GREEDYDATA	Matches any character (except newline) zero or more times, greedy.	5	Input: abc def ghi Pattern: <code>%{GREEDYDATA:x}%{GREEDYDATA:y}</code> Output: <code>{"x": "abc def", "y": "ghi"}</code>
GREEDYDATA_MULTILINE	Matches any character (including newline) zero or more times, greedy.	1	Input: abc def ghi Pattern: <code>%{GREEDYDATA_MULTILINE:data}</code> Output: <code>{"data": "abc\ndef\nghi"}</code>
QUOTEDSTRING	Matches quoted strings (single or double quotes) with escaped characters.	20	Input: "Hello, world!" Pattern: <code>%{QUOTEDSTRING:msg}</code> Output: <code>{"msg": "Hello, world!"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
UUID	Matches a standard UUID format: 8 hexadecimal characters, followed by three groups of 4 hexadecimal characters, and ending with 12 hexadecimal characters, all separated by hyphens.	20	Input: 550e8400-e29b-41d4-a716-446655440000 Pattern: <code>%{UUID:id}</code> Output: <code>{"id": "550e8400-e29b-41d4-a716-446655440000"}</code>
URN	Matches URN (Uniform Resource Name) syntax.	20	Input: urn:isbn:0451450523 Pattern: <code>%{URN:urn}</code> Output: <code>{"urn": "urn:isbn:0451450523"}</code>

AWS grok patterns

Pattern	Description	Maximum pattern limit	Example
ARN	Matches AWS Amazon Resource Names (ARNs), capturing the partition (aws, aws-cn, or aws-us-gov), service, Region, account ID, and up to 5 hierarchical resource identifiers separated by slashes. It will not match ARNs that are	5	Input: arn:aws:iam:us-east-1:123456789012:user/johndoe Pattern: <code>%{ARN:arn}</code> Output: <code>{"arn": "arn:aws:iam:us-east-1:123456789012:user/johndoe"}</code>

Pattern	Description	Maximum pattern limit	Example

missing information between colons.

Networking grok patterns

Grok Pattern	Description	Maximum pattern limit	Example

CISCOMAC	Matches a MAC address in 4-4-4 hexadecimal format.	20	Input: 0123.4567.89AB Pattern: <code>%{CISCOMAC:MacAddress}</code> Output: <code>{"MacAddress": "0123.4567.89AB"}</code>

WINDOWSMAC	Matches a MAC address in hexadecimal format with hyphens	20	Input: 01-23-45-67-89-AB Pattern: <code>%{WINDOWSMAC:MacAddress}</code> Output: <code>{"MacAddress": "01-23-45-67-89-AB"}</code>

COMMONMAC	Matches a MAC address in hexadecimal format with colons.	20	Input: 01:23:45:67:89:AB Pattern: <code>%{COMMONMAC:MacAddress}</code> Output: <code>{"MacAddress": "01:23:45:67:89:AB"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
MAC	Matches one of CISCOMAC, WINDOWSMAC or COMMONMAC grok patterns	20	Input: 01:23:45:67:89:AB Pattern: <code>%{MAC:m1}</code> Output: <code>{"m1": "01:23:45:67:89:AB"}</code>
IPV6	Matches IPv6 addresses, including compressed forms and IPv4-mapped IPv6 addresses.	5	Input: 2001:db8:3333:4444:5555:6666:7777:8888 Pattern: <code>%{IPV6:ip}</code> Output: <code>{"ip": "2001:db8:3333:4444:5555:6666:7777:8888"}</code>
IPV4	Matches an IPv4 address.	20	Input: 192.168.0.1 Pattern: <code>%{IPV4:ip}</code> Output: <code>{"ip": "192.168.0.1"}</code>
IP	Matches either IPv6 addresses as supported by <code>%{IPv6}</code> or IPv4 addresses as supported by <code>%{IPv4}</code>	5	Input: 192.168.0.1 Pattern: <code>%{IP:ip}</code> Output: <code>{"ip": "192.168.0.1"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
HOSTNAME or HOST	Matches domain names, including subdomains	5	Input: server-01 .internal-network. local Pattern: <code>%{HOST:host}</code> Output: <code>{"host": "server-01.internal-network.local"}</code>
IPORHOST	Matches either a hostname or an IP address	5	Input: 2001:db8: 3333:4444:5555:666 6:7777:8888 Pattern: <code>%{IPORHOST:ip}</code> Output: <code>{"ip": "2001:db8:3333:4444:5555:6666:7777:8888"}</code>
HOSTPORT	Matches an IP address or hostname as supported by <code>%{IPORHOST}</code> pattern followed by a colon and a port number, capturing the port as "PORT" in the output.	5	Input: 192.168.1.1:8080 Pattern: <code>%{HOSTPORT:ip}</code> Output: <code>{"ip": "192.168.1.1:8080", "PORT": "8080"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
URIHOST	Matches an IP address or hostname as supported by <code>%{IPORHOST}</code> pattern, optionally followed by a colon and a port number, capturing the port as "port" if present.	5	Input: <code>example.com:443</code> <code>10.0.0.1</code> Pattern: <code>%{URIHOST:host}</code> <code>%{URIHOST:ip}</code> Output: <code>{"host": "example.com:443", "port": "443", "ip": "10.0.0.1"}</code>

Path grok patterns

Grok Pattern	Description	Maximum pattern limit	Example
UNIXPATH	Matches URL paths, potentially including query parameters.	20	Input: <code>/search?q=regex</code> Pattern: <code>%{UNIXPATH:path}</code> Output: <code>{"path": "/search?q=regex"}</code>
WINPATH	Matches Windows file paths.	5	Input: <code>C:\Users\John\Documents\file.txt</code> Pattern: <code>%{WINPATH:path}</code> Output: <code>{"path": "C:\\Users\\John\\Documents\\file.txt"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
PATH	Matches either URL or Windows file paths	5	Input: /search?q=regex Pattern: %{PATH:path} Output: {"path":"/search?q=regex"}
TTY	Matches Unix device paths for terminals and pseudo-terminals.	20	Input: /dev/tty1 Pattern: %{TTY:path} Output: {"path":"/dev/tty1"}
URIPROTO	Matches letters, optionally followed by a plus (+) character and additional letters or plus (+) characters	20	Input: web+transformer Pattern: %{URIPROTO:protocol} Output: {"protocol":"web+transformer"}
URIPATH	Matches the path component of a URI	20	Input: /category/sub-category/product_name Pattern: %{URIPATH:path} Output: {"path":"/category/sub-category/product_name"}

Grok Pattern	Description	Maximum pattern limit	Example
URIPARAM	Matches URL query parameters	5	Input: ?param1=value1&param2=value2 Pattern: %{URIPARAM:url} Output: {"url": "?param1=value1&param2=value2"}
URIPATHPARAM	Matches a URI path optionally followed by query parameters	5	Input: /category/sub-category/product?id=12345&color=red Pattern: %{URIPATHPARAM:path} Output: {"path": "/category/sub-category/product?id=12345&color=red"}
URI	Matches a complete URI	5	Input: https://user:password@example.com/path/to/resource?param1=value1&param2=value2 Pattern: %{URI:uri} Output: {"path": "https://user:password@example.com/path/to/resource?param1=value1&param2=value2"}

Date and time grok patterns

Grok Pattern	Description	Maximum pattern limit	Example
MONTH	Matches full or abbreviated english month names as whole words	20	Input: Jan Pattern: <code>%{MONTH:month}</code> Output: <code>{"month": "Jan"}</code> Input: January Pattern: <code>%{MONTH:month}</code> Output: <code>{"month": "January"}</code>
MONTHNUM	Matches month numbers from 1 to 12, with optional leading zero for single-digit months.	20	Input: 5 Pattern: <code>%{MONTHNUM:month}</code> Output: <code>{"month": "5"}</code> Input: 05 Pattern: <code>%{MONTHNUM:month}</code> Output: <code>{"month": "05"}</code>
MONTHNUM	Matches two-digit month numbers from 01 to 12.	20	Input: 05 Pattern: <code>%{MONTHNUM2:month}</code>

Grok Pattern	Description	Maximum pattern limit	Example
			Output: {"month": "05"}
MONTHDAY	Matches day of the month from 1 to 31, with optional leading zero.	20	Input: 31 Pattern: %{MONTHDAY:monthDay} Output: {"monthDay": "31"}
YEAR	Matches year in two or four digits	20	Input: 2024 Pattern: %{YEAR:year} Output: {"year": "2024"} Input: 24 Pattern: %{YEAR:year} Output: {"year": "24"}
DAY	Matches full or abbreviated day names.	20	Input: Tuesday Pattern: %{DAY:day} Output: {"day": "Tuesday"}

Grok Pattern	Description	Maximum pattern limit	Example
HOURL	Matches hour in 24-hour format with an optional leading zero (0)0-23.	20	Input: 22 Pattern: %{HOURL:hour} Output: {"hour": "22"}
MINUTE	Matches minutes (00-59).	20	Input: 59 Pattern: %{MINUTE:min} Output: {"min": "59"}

Grok Pattern	Description	Maximum pattern limit	Example
SECOND	Matches a number representing seconds (0)0-60, optionally followed by a decimal point or colon and one or more digits for fractional minutes	20	Input: 3 Pattern: %{SECOND:second} Output: {"second": "3"} Input: 30.5 Pattern: %{SECOND:minSec} Output: {"minSec": "30.5"} Input: 30:5 Pattern: %{SECOND:minSec} Output: {"minSec": "30:5"}
TIME	Matches a time format with hours, minutes, and seconds in the format (H)H:mm:(s)s. Seconds include leap second (0)0-60.	20	Input: 09:45:32 Pattern: %{TIME:time} Output: {"time": "09:45:32"}

Grok Pattern	Description	Maximum pattern limit	Example
DATE_US	Matches a date in the format of (M)M/(d)d/(yy)yy or (M)M-(d)d-(yy)yy.	20	Input: 11/23/2024 Pattern: %{DATE_US:date} Output: {"date": "11/23/2024"} Input: 1-01-24 Pattern: %{DATE_US:date} Output: {"date": "1-01-24"}
DATE_EU	Matches date in format of (d)d/(M)M/(yy)yy, (d)d-(M)M-(yy)yy, or (d)d.(M)M.(yy)yy.	20	Input: 23/11/2024 Pattern: %{DATE_EU:date} Output: {"date": "23/11/2024"} Input: 1.01.24 Pattern: %{DATE_EU:date} Output: {"date": "1.01.24"}

Grok Pattern	Description	Maximum pattern limit	Example
ISO8601_TIMEZONE	Matches UTC offset 'Z' or time zone offset with optional colon in format <code>[+-(H)H(:)mm</code> .	20	Input: <code>+05:30</code> Pattern: <code>%{ISO8601_TIMEZONE:tz}</code> Output: <code>{"tz":"+05:30"}</code> Input: <code>-530</code> Pattern: <code>%{ISO8601_TIMEZONE:tz}</code> Output: <code>{"tz":"-530"}</code> Input: <code>Z</code> Pattern: <code>%{ISO8601_TIMEZONE:tz}</code> Output: <code>{"tz":"Z"}</code>
ISO8601_SECONDS	Matches a number representing seconds <code>(0)0-60</code> , optionally followed by a decimal point or colon and one or more digits for fractional seconds	20	Input: <code>60</code> Pattern: <code>%{ISO8601_SECONDS:second}</code> Output: <code>{"second":"60"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
TIMESTAMP_ISO8601	Matches ISO8601 datetime format (yy)yy-(M)M-(d)dT(H)H:mm:((s)s)(Z [-+](H)H:mm) with optional seconds and timezone.	20	Input: 2023-05-15T14:30:00+05:30 Pattern: %{TIMESTAMP_ISO8601}1:timestamp} Output: {"timestamp": "2023-05-15T14:30:00+05:30"} Input: 23-5-1T1:25+5:30 Pattern: %{TIMESTAMP_ISO8601}1:timestamp} Output: {"timestamp": "23-5-1T1:25+5:30"} Input: 23-5-1T1:25Z Pattern: %{TIMESTAMP_ISO8601}1:timestamp} Output: {"timestamp": "23-5-1T1:25Z"}

Grok Pattern	Description	Maximum pattern limit	Example
DATE	Matches either a date in the US format using <code>%{DATE_US}</code> or in the EU format using <code>%{DATE_EU}</code>	20	Input: 11/29/2024 Pattern: <code>%{DATE:date}</code> Output: <code>{"date":"11/29/2024"}</code> Input: 29.11.2024 Pattern: <code>%{DATE:date}</code> Output: <code>{"date":"29.11.2024"}</code>
DATESTAMP	Matches <code>%{DATE}</code> followed by <code>%{TIME}</code> pattern, separated by space or hyphen.	20	Input: 29-11-2024 14:30:00 Pattern: <code>%{DATESTAMP:dateTime}</code> Output: <code>{"dateTime":"29-11-2024 14:30:00"}</code>
TZ	Matches common time zone abbreviations (PST, PDT, MST, MDT, CST CDT, EST, EDT, UTC).	20	Input: PDT Pattern: <code>%{TZ:tz}</code> Output: <code>{"tz":"PDT"}</code>

Grok Pattern	Description	Maximum pattern limit	Example
DATESTAMP _RFC822	Matches date and time in format: Day MonthName (D)D (YY)YY (H)H:mm:(s)s Timezone	20	Input: Monday Jan 5 23 1:30:00 CDT Pattern: % DATESTAMP_RFC822 :dateTime} Output: {"dateTim e": "Monday Jan 5 23 1:30:00 CDT"} Input: Mon January 15 2023 14:30:00 PST Pattern: % DATESTAMP_RFC822 :dateTime} Output: {"dateTim e": "Mon January 15 2023 14:30:00 PST"}

Grok Pattern	Description	Maximum pattern limit	Example
DATESTAMP_RFC2822	Matches RFC2822 date-time format: Day, (d)d MonthName (yy)yy (H)H:mm:(s)s Z [-+](H)H:mm	20	Input: Mon, 15 May 2023 14:30:00 +0530 Pattern: % DATESTAMP_RFC2822:dateTime} Output: {"dateTime": "Mon, 15 May 2023 14:30:00 +0530"} Input: Monday, 15 Jan 23 14:30:00 Z Pattern: % DATESTAMP_RFC2822:dateTime} Output: {"dateTime": "Monday, 15 Jan 23 14:30:00 Z"}
DATESTAMP_OTHER	Matches date and time in format: Day MonthName (d)d (H)H:mm:(s)s Timezone (yy)yy	20	Input: Mon May 15 14:30:00 PST 2023 Pattern: % DATESTAMP_OTHER:dateTime} Output: {"dateTime": "Mon May 15 14:30:00 PST 2023"}

Grok Pattern	Description	Maximum pattern limit	Example
DATESTAMP_EVENTLOG	Matches compact datetime format without separators: (yy)yyMM(d)d(H)Hmm(s)s	20	Input: 20230515143000 Pattern: %{DATESTAMP_EVENTLOG:dateTime} Output: {"dateTime": "20230515143000"}

Log grok patterns

Grok Pattern	Description	Maximum pattern limit	Example
LOGLEVEL	Matches standard log levels in different capitalizations and abbreviations, including the following: Alert/ALERT , Trace/TRACE , Debug/DEBUG , Notice/NOTICE , Info/INFO , Warn/Warning/WARN/ WARNING , Err/Error /ERR/ERROR , Crit/Critical/CRIT/ CRITICAL , Fatal/FATAL , Severe/SEVERE , Emerg/Emergency/EMERG/EMERGENCY	20	Input: INFO Pattern: %{LOGLEVEL:logLevel} Output: {"logLevel": "INFO"}

Grok Pattern	Description	Maximum pattern limit	Example
HTTPDATE	Matches date and time format often used in log files. Format: (d)d/ MonthName/(yy)yy: (H)H:mm:(s)s Timezone MonthName: Matches full or abbreviated english month names (Example: "Jan" or "January") Timezone: Matches %{INT} grok pattern	20	Input: 23/Nov/20 24:14:30:00 +0640 Pattern: %{HTTPDATE:date} Output: {"date": "23/Nov/20 24:14:30:00 +0640"}
SYSLOGTIMESTAMP	Matches date format with MonthName (d)d (H)H:mm:(s)s MonthName: Matches full or abbreviated english month names (Example: "Jan" or "January")	20	Input: Nov 29 14:30:00 Pattern: %{SYSLOGTIMESTAMP:dateTime} Output: {"dateTime": "Nov 29 14:30:00"}
PROG	Matches a program name consisting of string of letters, digits, dot, underscore, forward slash, percent sign, and hyphen characters.	20	Input: user.profile/settings-page Pattern: %{PROG:program} Output: {"program": "user.profile/settings-page"}

Grok Pattern	Description	Maximum pattern limit	Example
SYSLOGPROG	Matches PROG grok pattern optionally followed by a process ID in square brackets.	20	Input: user.profile/settings-page[1234] Pattern: %{SYSLOGPROG:programWithId} Output: {"programWithId": "user.profile/settings-page[1234]", "program": "user.profile/settings-page", "pid": "1234"}
SYSLOGHOST	Matches either a %{HOST} or %{IP} pattern	5	Input: 2001:db8:3333:4444:5555:6666:7777:8888 Pattern: %{SYSLOGHOST:ip} Output: {"ip": "2001:db8:3333:4444:5555:6666:7777:8888"}

Grok Pattern	Description	Maximum pattern limit	Example
SYSLOGFACILITY	Matches syslog priority in decimal format. The value should be enclosed in angular brackets (<>).	20	Input: <13.6> Pattern: %{SYSLOGFACILITY:syslog} Output: {"syslog": "<13.6>", "facility": "13", "priority": "6"}

Common log grok patterns

You can use pre-defined custom grok patterns to match Apache, NGINX and Syslog Protocol (RFC 5424) log formats. When you use these specific patterns, they must be the first patterns in your matching configuration, and no other patterns can precede them. Also, you can follow them only with exactly one **DATA**, **GREEDYDATA** or **GREEDYDATA_MULTILINE** pattern.

Grok pattern	Description	Maximum pattern limit
APACHE_ACCESS_LOG	Matches Apache access logs	1
NGINX_ACCESS_LOG	Matches NGINX access logs	1
SYSLOG5424	Matches Syslog Protocol (RFC 5424) logs	1

The following shows valid and invalid examples for using these common log format patterns.

```

"%{NGINX_ACCESS_LOG} %{DATA}" // Valid
"%{SYSLOG5424}%{DATA:logMsg}" // Valid
"%{APACHE_ACCESS_LOG} %{GREEDYDATA:logMsg}" // Valid
"%{APACHE_ACCESS_LOG} %{SYSLOG5424}" // Invalid (multiple common log patterns used)
"%{NGINX_ACCESS_LOG} %{NUMBER:num}" // Invalid (Only GREEDYDATA and DATA patterns are
supported with common log patterns)
"%{GREEDYDATA:logMsg} %{SYSLOG5424}" // Invalid (GREEDYDATA and DATA patterns are
supported only after common log patterns)

```

Common log format examples

Apache log example

Sample log:

```
127.0.0.1 - - [03/Aug/2023:12:34:56 +0000] "GET /page.html HTTP/1.1" 200 1234
```

Transformer:

```

[
  {
    "grok": {
      "match": "%{APACHE_ACCESS_LOG}"
    }
  }
]

```

Output:

```

{
  "request": "/page.html",
  "http_method": "GET",
  "status_code": 200,
  "http_version": "1.1",
  "response_size": 1234,
  "remote_host": "127.0.0.1",
  "timestamp": "2023-08-03T12:34:56Z"
}

```

NGINX log example

Sample log:

```
192.168.1.100 - Foo [03/Aug/2023:12:34:56 +0000] "GET /account/login.html HTTP/1.1"
200 42 "https://www.amazon.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36"
```

Transformer:

```
[
  {
    "grok": {
      "match": "%{NGINX_ACCESS_LOG}"
    }
  }
]
```

Output:

```
{
  "request": "/account/login.html",
  "referrer": "https://www.amazon.com/",
  "agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/92.0.4515.131 Safari/537.36",
  "http_method": "GET",
  "status_code": 200,
  "auth_user": "Foo",
  "http_version": "1.1",
  "response_size": 42,
  "remote_host": "192.168.1.100",
  "timestamp": "2023-08-03T12:34:56Z"
}
```

Syslog Protocol (RFC 5424) log example

Sample log:

```
<165>1 2003-10-11T22:14:15.003Z mymachine.example.com evntslog - ID47
[exampleSDID@32473 iut="3" eventSource= "Application" eventID="1011"]
[examplePriority@32473 class="high"]
```

Transformer:

```
[
  {
```

```
      "grok": {
        "match": "%{SYSLOG5424}"
      }
    }
  ]
}
```

Output:

```
{
  "pri": 165,
  "version": 1,
  "timestamp": "2003-10-11T22:14:15.003Z",
  "hostname": "mymachine.example.com",
  "app": "evntslog",
  "msg_id": "ID47",
  "structured_data": "exampleSDID@32473 iut=\"3\" eventSource= \"Application\" eventID= \"1011\"",
  "message": "[examplePriority@32473 class=\"high\"]"
}
```

CSV

The **csv** processor parses comma-separated values (CSV) from the log events into columns.

Field	Description	Required?	Default	Limits
source	Path to the field in the log event that will be parsed	No	@message	Maximum length: 128 Maximum nested key depth: 3
delimiter	The character used to separate each column in the original comma-separated value log event	No	,	Maximum length: 1 unless the value is \t or \s
quoteCharacter	Character used as a text qualifier for a single column of data	No	"	Maximum length: 1
columns	List of names to use for the columns in the transformed log event.	No	[column_1, column_2	Maximum CSV columns: 100

Field	Description	Required?	Default	Limits
				Maximum length: 128 Maximum nested key depth: 3

Setting `delimiter` to `\t` will separate each column on a tab character, and `\t` will separate each column on a single space character.

Example

Suppose part of an ingested log event looks like this:

```
'Akua Mansa':28:'New York: USA'
```

Suppose we use only the **csv** processor:

```
[
  {
    "csv": {
      "delimiter": ":",
      "quoteCharacter": "'"
    }
  }
]
```

The transformed log event would be the following.

```
{
  "column_1": "Akua Mansa",
  "column_2": "28",
  "column_3": "New York: USA"
}
```

parseKeyValue

Use the **parseKeyValue** processor to parse a specified field into key-value pairs. You can customize the processor to parse field information with the following options.

Field	Description	Required?	Default	Limits
source	Path to the field in the log event that will be parsed	No	@message	Maximum length: 128 Maximum nested key depth: 3
destination	The destination field to put the extracted key-value pairs into	No		Maximum length: 128
fieldDelimiter	The field delimiter string that is used between key-value pairs in the original log events	No	&	Maximum length: 128
keyValueDelimiter	The delimiter string to use between the key and value in each pair in the transformed log event	No	=	Maximum length: 128
nonMatchValue	A value to insert into the value field in the result, when a key-value pair is not successfully split.	No		Maximum length: 128
keyPrefix	If you want to add a prefix to all transformed keys, specify it here.	No		Maximum length: 128
overwriteIfExists	Whether to overwrite the value if the destination key already exists	No	false	

Example

Take the following example log event:

```
key1:value1!key2:value2!key3:value3!key4
```

Suppose we use the following processor configuration:

```
[
  {
```

```
    "parseKeyValue": {
      "destination": "new_key",
      "fieldDelimiter": "!",
      "keyValueDelimiter": ":",
      "nonMatchValue": "defaultValue",
      "keyPrefix": "parsed_"
    }
  }
]
```

The transformed log event would be the following.

```
{
  "new_key": {
    "parsed_key1": "value1",
    "parsed_key2": "value2",
    "parsed_key3": "value3",
    "parsed_key4": "defaultValue"
  }
}
```

Built-in processors for AWS vended logs

This section contains information about the built-in processors that you can use with AWS services that vend logs.

Contents

- [parseWAF](#)
- [parsePostgres](#)
- [parseCloudfront](#)
- [parseRoute53](#)
- [parseVPC](#)
- [parseToOCSF](#)

parseWAF

Use this processor to parse AWS WAF vended logs. It takes the contents of `httpRequest.headers` and creates JSON keys from each header name, with the corresponding

value. It also does the same for labels. These transformations can make querying AWS WAF logs much easier. For more information about AWS WAF log format, see [Log examples for web ACL traffic](#).

This processor accepts only @message as the input.

Important

If you use this processor, it must be the first processor in your transformer.

Example

Take the following example log event:

```
{
  "timestamp": 1576280412771,
  "formatVersion": 1,
  "webaclId": "arn:aws:wafv2:ap-southeast-2:111122223333:regional/webacl/STMTTest/1EXAMPLE-2ARN-3ARN-4ARN-123456EXAMPLE",
  "terminatingRuleId": "STMTTest_SQLi_XSS",
  "terminatingRuleType": "REGULAR",
  "action": "BLOCK",
  "terminatingRuleMatchDetails": [
    {
      "conditionType": "SQL_INJECTION",
      "sensitivityLevel": "HIGH",
      "location": "HEADER",
      "matchedData": ["10", "AND", "1"]
    }
  ],
  "httpSourceName": "-",
  "httpSourceId": "-",
  "ruleGroupList": [],
  "rateBasedRuleList": [],
  "nonTerminatingMatchingRules": [],
  "httpRequest": {
    "clientIp": "1.1.1.1",
    "country": "AU",
    "headers": [
      { "name": "Host", "value": "localhost:1989" },
      { "name": "User-Agent", "value": "curl/7.61.1" },
      { "name": "Accept", "value": "*/*" }
    ]
  }
}
```

```

    { "name": "x-stm-test", "value": "10 AND 1=1" }
  ],
  "uri": "/myUri",
  "args": "",
  "httpVersion": "HTTP/1.1",
  "httpMethod": "GET",
  "requestId": "rid"
},
"labels": [{ "name": "value" }]
}

```

The processor configuration is this:

```

[
  {
    "parseWAF": {}
  }
]

```

The transformed log event would be the following.

```

{
  "httpRequest": {
    "headers": {
      "Host": "localhost:1989",
      "User-Agent": "curl/7.61.1",
      "Accept": "*/*",
      "x-stm-test": "10 AND 1=1"
    },
    "clientIp": "1.1.1.1",
    "country": "AU",
    "uri": "/myUri",
    "args": "",
    "httpVersion": "HTTP/1.1",
    "httpMethod": "GET",
    "requestId": "rid"
  },
  "labels": { "name": "value" },
  "timestamp": 1576280412771,
  "formatVersion": 1,
  "webaclId": "arn:aws:wafv2:ap-southeast-2:111122223333:regional/webacl/
STMTest/1EXAMPLE-2ARN-3ARN-4ARN-123456EXAMPLE",
  "terminatingRuleId": "STMTest_SQLi_XSS",

```

```
{
  "terminatingRuleType": "REGULAR",
  "action": "BLOCK",
  "terminatingRuleMatchDetails": [
    {
      "conditionType": "SQL_INJECTION",
      "sensitivityLevel": "HIGH",
      "location": "HEADER",
      "matchedData": ["10", "AND", "1"]
    }
  ],
  "httpSourceName": "-",
  "httpSourceId": "-",
  "ruleGroupList": [],
  "rateBasedRuleList": [],
  "nonTerminatingMatchingRules": []
}
```

parsePostgres

Use this processor to parse Amazon RDS for PostgreSQL vended logs, extract fields, and convert them to JSON format. For more information about RDS for PostgreSQL log format, see [RDS for PostgreSQL database log files](#).

This processor accepts only @message as the input.

Important

If you use this processor, it must be the first processor in your transformer.

Example

Take the following example log event:

```
2019-03-10 03:54:59 UTC:10.0.0.123(52834):postgres@logtestdb:[20175]:ERROR: column
"wrong_column_name" does not exist at character 8
```

The processor configuration is this:

```
[
  {
    "parsePostgres": {}
  }
]
```

```
}
]
```

The transformed log event would be the following.

```
{
  "logTime": "2019-03-10 03:54:59 UTC",
  "srcIp": "10.0.0.123(52834)",
  "userName": "postgres",
  "dbName": "logtestdb",
  "processId": "20175",
  "logLevel": "ERROR"
}
```

parseCloudfront

Use this processor to parse Amazon CloudFront vended logs, extract fields, and convert them into JSON format. Encoded field values are decoded. Values that are integers and doubles are treated as such. For more information about Amazon CloudFront log format, see [Configure and use standard logs \(access logs\)](#).

This processor accepts only @message as the input.

Important

If you use this processor, it must be the first processor in your transformer.

Example

Take the following example log event:

```
2019-12-04 21:02:31 LAX1 392 192.0.2.24 GET
d1111111abcdef8.cloudfront.net /index.html 200 - Mozilla/5.0%20(Windows
%20NT%2010.0;%20Win64;%20x64)%20AppleWebKit/537.36%20(KHTML,
%20like%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Hit
SOX4xwn4XV6Q4rgb7XiVG0Hms_BG1TAC4KyHmureZmBNrjGdRLiNIQ==
d1111111abcdef8.cloudfront.net https 23 0.001 - TLSv1.2 ECDHE-RSA-AES128-GCM-
SHA256 Hit HTTP/2.0 - - 11040 0.001 Hit text/html 78 - -
```

The processor configuration is this:

```
[
  {
    "parseCloudfront": {}
  }
]
```

The transformed log event would be the following.

```
{
  "date": "2019-12-04",
  "time": "21:02:31",
  "x-edge-location": "LAX1",
  "sc-bytes": 392,
  "c-ip": "192.0.2.24",
  "cs-method": "GET",
  "cs(Host)": "d111111abcdef8.cloudfront.net",
  "cs-uri-stem": "/index.html",
  "sc-status": 200,
  "cs(Referer)": "-",
  "cs(User-Agent)": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36",
  "cs-uri-query": "-",
  "cs(Cookie)": "-",
  "x-edge-result-type": "Hit",
  "x-edge-request-id": "S0X4xwn4XV6Q4rgb7XiVG0Hms_BG1TAC4KyHmureZmBNrjGdRLiNIQ==",
  "x-host-header": "d111111abcdef8.cloudfront.net",
  "cs-protocol": "https",
  "cs-bytes": 23,
  "time-taken": 0.001,
  "x-forwarded-for": "-",
  "ssl-protocol": "TLSv1.2",
  "ssl-cipher": "ECDHE-RSA-AES128-GCM-SHA256",
  "x-edge-response-result-type": "Hit",
  "cs-protocol-version": "HTTP/2.0",
  "fle-status": "-",
  "fle-encrypted-fields": "-",
  "c-port": 11040,
  "time-to-first-byte": 0.001,
  "x-edge-detailed-result-type": "Hit",
  "sc-content-type": "text/html",
  "sc-content-len": 78,
  "sc-range-start": "-",
  "sc-range-end": "-"
}
```

```
}
```

parseRoute53

Use this processor to parse Amazon Route 53 Public Data Plane vended logs, extract fields, and convert them into JSON format. Encoded field values are decoded. This processor does not support Amazon Route 53 Resolver logs.

This processor accepts only @message as the input.

Important

If you use this processor, it must be the first processor in your transformer.

Example

Take the following example log event:

```
1.0 2017-12-13T08:15:50.235Z Z123412341234 example.com AAAA NOERROR TCP IAD12 192.0.2.0
198.51.100.0/24
```

The processor configuration is this:

```
[
  {
    "parseRoute53": {}
  }
]
```

The transformed log event would be the following.

```
{
  "version": 1.0,
  "queryTimestamp": "2017-12-13T08:15:50.235Z",
  "hostZoneId": "Z123412341234",
  "queryName": "example.com",
  "queryType": "AAAA",
  "responseCode": "NOERROR",
  "protocol": "TCP",
  "edgeLocation": "IAD12",
```



```
"resolverIp": "192.0.2.0",
"ednsClientSubnet": "198.51.100.0/24"
}
```

parseVPC

Use this processor to parse Amazon VPC vended logs, extract fields, and convert them into JSON format. Encoded field values are decoded.

This processor accepts only @message as the input.

Important

If you use this processor, it must be the first processor in your transformer.

Example

Take the following example log event:

```
2 123456789010 eni-abc123de 192.0.2.0 192.0.2.24 20641 22 6 20 4249 1418530010
1418530070 ACCEPT OK
```

The processor configuration is this:

```
[
  {
    "parseVPC": {}
  }
]
```

The transformed log event would be the following.

```
{
  "version": 2,
  "accountId": "123456789010",
  "interfaceId": "eni-abc123de",
  "srcAddr": "192.0.2.0",
  "dstAddr": "192.0.2.24",
  "srcPort": 20641,
  "dstPort": 22,
}
```

```
"protocol": 6,  
"packets": 20,  
"bytes": 4249,  
"start": 1418530010,  
"end": 1418530070,  
"action": "ACCEPT",  
"logStatus": "OK"  
}
```

parseToOCSF

The `parseToOCSF` processor converts logs into Open Cybersecurity Schema Framework (OCSF) events. OCSF is an open standard that provides a common schema for security data, enabling better interoperability and analysis across different security tools and platforms.

This processor is particularly useful for security analytics workflows where you need to standardize log formats from various AWS services into a consistent schema for downstream analysis.

Parameters

`eventSource` (required)

Specifies the AWS service or process that produces the log events to be converted. Valid values are:

- `CloudTrail` - CloudTrail logs
- `Route53Resolver` - Route 53 Resolver logs
- `VPCFlow` - Amazon VPC Flow Logs
- `EKSAudit` - Amazon EKS audit logs
- `AWSWAF` - AWS WAF logs

`ocsfVersion` (required)

Specifies which version of the OCSF schema to use for the transformed log events. Currently supported versions: `V1.1`, `V1.5`

`mappingVersion` (optional)

Specifies the OCSF transformation mapping version. Controls which transformation logic is applied when converting logs to OCSF format. If not specified, uses the latest available version at policy creation time. Existing policies do not automatically upgrade when new mapping versions are released. Current latest version: `v1.5.0`.

Note: Not supported when `ocsfVersion` is `V1.1`.

`source` (optional)

The path to the field in the log event that you want to parse. If omitted, the entire log message is parsed.

Example

The following example shows how to use `parseToOCSF` to convert VPC Flow Logs to OCSF format:

```
{
  "parseToOCSF": {
    "eventSource": "VPCFlow",
    "ocsfVersion": "V1.1"
  }
}
```

The following example shows how to specify a particular mapping version for consistent transformation behavior:

```
{
  "parseToOCSF": {
    "eventSource": "CloudTrail",
    "ocsfVersion": "V1.5",
    "mappingVersion": "v1.5.0"
  }
}
```

String mutate processors

This section contains information about the string mutate processors that you can use with a log event transformer.

Contents

- [lowerCaseString](#)
- [upperCaseString](#)
- [splitString](#)
- [substituteString](#)

- [trimString](#)

lowerCaseString

The lowerCaseString processor converts a string to its lowercase version.

Field	Description	Required?	Default	Limits
withKeys	A list of keys to convert to lowercase	Yes		Maximum entries: 10

Example

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "INNER_VALUE"
  }
}
```

The transformer configuration is this, using lowerCaseString with parseJSON:

```
[
  {
    "parseJSON": {}
  },
  {
    "lowerCaseString": {
      "withKeys": ["outer_key.inner_key"]
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

```
}
```

upperCaseString

The upperCaseString processor converts a string to its uppercase version.

Field	Description	Required?	Default	Limits
withKeys	A list of keys to convert to uppercase	Yes		Maximum entries: 10

Example

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

The transformer configuration is this, using upperCaseString with parseJSON:

```
[
  {
    "parseJSON": {}
  },
  {
    "upperCaseString": {
      "withKeys": ["outer_key.inner_key"]
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {
    "inner_key": "INNER_VALUE"
  }
}
```

```
}
```

splitString

The `splitString` processor is a type of string mutate processor which splits a field into an array using a delimiting character.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array must contain <code>source</code> and <code>delimiter</code> fields.	Yes		Maximum entries: 10
source	The key of the field value to split	Yes		Maximum length: 128
delimiter	The delimiter string to split the field value on	Yes		Maximum length: 128

Example 1

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

The transformer configuration is this, using `splitString` with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
    "splitString": {
      "entries": [
        {
          "source": "outer_key.inner_key",
          "delimiter": "_"
        }
      ]
    }
  }
]
```

```

    }
  ]
}
]

```

The transformed log event would be the following.

```

{
  "outer_key": {
    "inner_key": [
      "inner",
      "value"
    ]
  }
}

```

Example 2

The delimiter to split the string on can be multiple characters long.

Take the following example log event:

```

{
  "outer_key": {
    "inner_key": "item1, item2, item3"
  }
}

```

The transformer configuration is as follows:

```

[
  {
    "parseJSON": {}
  },
  {
    "splitString": {
      "entries": [
        {
          "source": "outer_key.inner_key",
          "delimiter": ", "
        }
      ]
    }
  }
]

```

```
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {
    "inner_key": [
      "item1",
      "item2",
      "item3"
    ]
  }
}
```

substituteString

The `substituteString` processor is a type of string mutate processor which matches a key’s value against a regular expression and replaces all matches with a replacement string.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array must contain <code>source</code> , <code>from</code> , and <code>to</code> fields.	Yes		Maximum entries: 10
source	The key of the field to modify	Yes		Maximum length: 128 Maximum nested key depth: 3
from	The regular expression string to be replaced. Special regex characters such as <code>[</code> and <code>]</code> must be escaped using <code>\\</code> when using double quotes and with <code>\</code> when using single quotes or when configured from the AWS Management Console. For more	Yes		Maximum length: 128

Field	Description	Required?	Default	Limits
	information, see Class Pattern on the Oracle web site. You can wrap a pattern in (. . .) to create a numbered capturing group and create (?P<group_name> . . .) named capturing groups that can be referenced in the to field.			
to	The string to be substituted for each match of from. Backreferences to capturing groups can be used. Use the form \$n for numbered groups such as \$1 and use \${group_name} for named groups such as \${my_group} .>	Yes		Maximum length: 128 Maximum number of backreferences: 10 Maximum number of duplicate backreferences: 2

Example 1

Take the following example log event:

```
{
  "outer_key": {
    "inner_key1": "[]",
    "inner_key2": "123-345-567",
    "inner_key3": "A cat takes a catnap."
  }
}
```

The transformer configuration is this, using substituteString with parseJSON:

```
[
  {
    "parseJSON": {}
  },
  {
```

```

    "substituteString": {
      "entries": [
        {
          "source": "outer_key.inner_key1",
          "from": "\\[\\]",
          "to": "value1"
        },
        {
          "source": "outer_key.inner_key2",
          "from": "[0-9]{3}-[0-9]{3}-[0-9]{3}",
          "to": "xxx-xxx-xxx"
        },
        {
          "source": "outer_key.inner_key3",
          "from": "cat",
          "to": "dog"
        }
      ]
    }
  }
}

```

The transformed log event would be the following.

```

{
  "outer_key": {
    "inner_key1": "value1",
    "inner_key2": "xxx-xxx-xxx",
    "inner_key3": "A dog takes a dognap."
  }
}

```

Example 2

Take the following example log event:

```

{
  "outer_key": {
    "inner_key1": "Tom, Dick, and Harry",
    "inner_key2": "arn:aws:sts::123456789012:assumed-role/MyImportantRole/MySession"
  }
}

```

The transformer configuration is this, using `substituteString` with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
    "substituteString": {
      "entries": [
        {
          "source": "outer_key.inner_key1",
          "from": "(\\w+), (\\w+), and (\\w+)",
          "to": "$1 and $3"
        },
        {
          "source": "outer_key.inner_key2",
          "from": "^arn:aws:sts::(?P<account_id>\\d{12}):assumed-role/(?P<role_name>[\\w+=,\\.@-]+)/(?P<role_session_name>[\\w+=,\\.@-]+)$",
          "to": "${account_id}:${role_name}:${role_session_name}"
        }
      ]
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {
    "inner_key1": "Tom and Harry",
    "inner_key2": "123456789012:MyImportantRole:MySession"
  }
}
```

trimString

The `trimString` processor removes whitespace from the beginning and end of a key.

Field	Description	Required?	Default	Limits
<code>withKeys</code>	A list of keys to trim	Yes		Maximum entries: 10

Example

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "  inner_value  "
  }
}
```

The transformer configuration is this, using `trimString` with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
    "trimString": {
      "withKeys": ["outer_key.inner_key"]
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

JSON mutate processors

This section contains information about the JSON mutate processors that you can use with a log event transformer.

Contents

- [addKeys](#)
- [deleteKeys](#)
- [moveKeys](#)

- [renameKeys](#)
- [copyValue](#)
- [listToMap](#)

addKeys

Use the addKeys processor to add new key-value pairs to the log event.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array can contain key, value, and <code>overwriteIfExists</code> fields.	Yes		Maximum entries: 5
key	The key of the new entry to be added	Yes		Maximum length: 128 Maximum nested key depth: 3
value	The value of the new entry to be added	Yes		Maximum length: 256
overwriteIfExists	If you set this to <code>true</code> , the existing value is overwritten if key already exists in the event. The default value is <code>false</code> .	No	<code>false</code>	No limit

Example

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

The transformer configuration is this, using addKeys with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
    "addKeys": {
      "entries": [
        {
          "key": "outer_key.new_key",
          "value": "new_value"
        }
      ]
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {
    "inner_key": "inner_value",
    "new_key": "new_value"
  }
}
```

deleteKeys

Use the deleteKeys processor to delete fields from a log event. These fields can include key-value pairs.

Field	Description	Required?	Default	Limits
withKeys	The list of keys to delete.	Yes	No limit	Maximum entries: 5

Example

Take the following example log event:

```
{
  "outer_key": {
```

```
      "inner_key": "inner_value"
    }
  }
```

The transformer configuration is this, using `deleteKeys` with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
    "deleteKeys": {
      "withKeys":["outer_key.inner_key"]
    }
  }
]
```

The transformed log event would be the following.

```
{
  "outer_key": {}
}
```

moveKeys

Use the `moveKeys` processor to move a key from one field to another.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array can contain <code>source</code> , <code>target</code> , and <code>overwriteIfExists</code> fields.	Yes		Maximum entries: 5
source	The key to move	Yes		Maximum length: 128 Maximum nested key depth: 3
target	The key to move to	Yes		Maximum length: 128

Field	Description	Required?	Default	Limits
				Maximum nested key depth: 3
overwriteIfExists	If you set this to <code>true</code> , the existing value is overwritten if key already exists in the event. The default value is <code>false</code> .	No	false	No limit

Example

Take the following example log event:

```
{
  "outer_key1": {
    "inner_key1": "inner_value1"
  },
  "outer_key2": {
    "inner_key2": "inner_value2"
  }
}
```

The transformer configuration is this, using `moveKeys` with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
    "moveKeys": {
      "entries": [
        {
          "source": "outer_key1.inner_key1",
          "target": "outer_key2"
        }
      ]
    }
  }
]
```


The transformed log event would be the following.

```
{
  "outer_key1": {},
  "outer_key2": {
    "inner_key2": "inner_value2",
    "inner_key1": "inner_value1"
  }
}
```

renameKeys

Use the `renameKeys` processor to rename keys in a log event.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array can contain <code>key</code> , <code>target</code> , and <code>overwriteIfExists</code> fields.	Yes	No limit	Maximum entries: 5
key	The key to rename	Yes	No limit	Maximum length: 128
target	The new key name	Yes	No limit	Maximum length: 128 Maximum nested key depth: 3
overwriteIfExists	If you set this to <code>true</code> , the existing value is overwritten if key already exists in the event. The default value is <code>false</code> .	No	false	No limit

Example

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

```
}  
}
```

The transformer configuration is this, using `renameKeys` with `parseJSON`:

```
[  
  {  
    "parseJSON": {}  
  },  
  {  
    "renameKeys": {  
      "entries": [  
        {  
          "key": "outer_key",  
          "target": "new_key"  
        }  
      ]  
    }  
  }  
]
```

The transformed log event would be the following.

```
{  
  "new_key": {  
    "inner_key": "inner_value"  
  }  
}
```

copyValue

Use the `copyValue` processor to copy values within a log event. You can also use this processor to add metadata to log events, by copying the values of the following metadata keys into the log events: `@logGroupName`, `@logGroupStream`, `@accountId`, `@regionName`. This is illustrated in the following example.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array can contain source,	Yes		Maximum entries: 5

Field	Description	Required?	Default	Limits
	target, and overwrite IfExists fields.			
source	The key to copy	Yes		Maximum length: 128 Maximum nested key depth: 3
target	The key to copy the value to	Yes	No limit	Maximum length: 128 Maximum nested key depth: 3
overwrite IfExists	If you set this to true, the existing value is overwritten if key already exists in the event. The default value is false.	No	false	No limit

Example

Take the following example log event:

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

The transformer configuration is this, using copyValue with parseJSON:

```
[
  {
    "parseJSON": {}
  },
  {
    "copyValue": {
      "entries": [
        {
```

```

        "key": "outer_key.new_key",
        "target": "new_key"
    },
    {
        "source": "@logGroupName",
        "target": "log_group_name"
    },
    {
        "source": "@logGroupStream",
        "target": "log_group_stream"
    },
    {
        "source": "@accountId",
        "target": "account_id"
    },
    {
        "source": "@regionName",
        "target": "region_name"
    }
]
}
]

```

The transformed log event would be the following.

```

{
  "outer_key": {
    "inner_key": "inner_value"
  },
  "new_key": "inner_value",
  "log_group_name": "myLogGroupName",
  "log_group_stream": "myLogStreamName",
  "account_id": "012345678912",
  "region_name": "us-east-1"
}

```

listToMap

The `listToMap` processor takes a list of objects that contain key fields, and converts them into a map of target keys.

Field	Description	Required?	Default	Limits
source	The key in the ProcessingEvent with a list of objects that will be converted to a map	Yes		Maximum length: 128 Maximum nested key depth: 3
key	The key of the fields to be extracted as keys in the generated map	Yes		Maximum length: 128
valueKey	If this is specified, the values that you specify in this parameter will be extracted from the source objects and put into the values of the generated map. Otherwise, original objects in the source list will be put into the values of the generated map.	No		Maximum length: 128
target	The key of the field that will hold the generated map	No	Root node	Maximum length: 128 Maximum nested key depth: 3
flatten	A Boolean value to indicate whether the list will be flattened into single items or if the values in the generated map will be lists. By default the values for the matching keys will be represented in an array. Set <code>flatten</code> to <code>true</code> to convert the array to a single value based on the value of <code>flattenedElement</code> .	No	false	
flattenedElement	If you set <code>flatten</code> to <code>true</code> , use <code>flattenedElement</code> to specify	Required when		Value can only be first or last

Field	Description	Required?	Default	Limits
	which element, first or last, to keep.	flatten is set to true		

Example

Take the following example log event:

```
{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ]
}
```

Transformer for use case 1: flatten is false

```
[
  {
    "parseJSON": {}
  },
  {
    "listToMap": {
      "source": "outer_key"
      "key": "inner_key",
      "valueKey": "inner_value",
```

```

        "flatten": false
    }
}
]
```

The transformed log event would be the following.

```

{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ],
  "a": [
    "val-a"
  ],
  "b": [
    "val-b1",
    "val-b2"
  ],
  "c": [
    "val-c"
  ]
}
```

Transformer for use case 2: flatten is true and flattenedElement is first

```

[
  {
    "parseJSON": {}
  }
]
```

```

    },
    {
      "listToMap": {
        "source": "outer_key"
        "key": "inner_key",
        "valueKey": "inner_value",
        "flatten": true,
        "flattenedElement": "first"
      }
    }
  ]
}

```

The transformed log event would be the following.

```

{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ],
  "a": "val-a",
  "b": "val-b1",
  "c": "val-c"
}

```

Transformer for use case 3: flatten is true and flattenedElement is last

```

[
  {
    "parseJSON": {}
  }
]

```



```
    },
    {
      "listToMap": {
        "source": "outer_key"
        "key": "inner_key",
        "valueKey": "inner_value",
        "flatten": true,
        "flattenedElement": "last"
      }
    }
  ]
}
```

The transformed log event would be the following.

```
{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ],
  "a": "val-a",
  "b": "val-b2",
  "c": "val-c"
}
```

Datatype converter processors

This section contains information about the datatype converter processors that you can use with a log event transformer.

Contents

- [typeConverter](#)
- [datetimeConverter](#)

typeConverter

Use the typeConverter processor to convert a value type associated with the specified key to the specified type. It's a casting processor that changes the types of the specified fields. Values can be converted into one of the following datatypes: integer, double, string and boolean.

Field	Description	Required?	Default	Limits
entries	Array of entries. Each item in the array must contain key and type fields.	Yes		Maximum entries: 10
key	The key with the value that is to be converted to a different type	Yes		Maximum length: 128 Maximum nested key depth: 3
type	The type to convert to. Valid values are integer, double, string and boolean.	Yes		

Example

Take the following example log event:

```
{
  "name": "value",
  "status": "200"
}
```

The transformer configuration is this, using typeConverter with parseJSON:

```
[
```

```
{
  "parseJSON": {}
},
{
  "typeConverter": {
    "entries": [
      {
        "key": "status",
        "type": "integer"
      }
    ]
  }
}
]
```

The transformed log event would be the following.

```
{
  "name": "value",
  "status": 200
}
```

datetimeConverter

Use the `datetimeConverter` processor to convert a datetime string into a format that you specify.

Field	Description	Required?	Default	Limits
source	The key to apply the date conversion to.	Yes		Maximum entries: 10
matchPatterns	A list of patterns to match against the source field	Yes		Maximum entries: 5
target	The JSON field to store the result in.	Yes		Maximum length: 128 Maximum nested key depth: 3

Field	Description	Required?	Default	Limits
targetFormat	The datetime format to use for the converted data in the target field.	No	yyyy-MM-dd'T'HH:mm:ss.SSS'Z'	Maximum length:64
sourceTimezone	The time zone of the source field. For a list of possible values, see Java Supported Zone Ids and Offsets .	No	UTC	Minimum length:1
targetTimezone	The time zone of the target field. For a list of possible values, see Java Supported Zone Ids and Offsets .	No	UTC	Minimum length:1
locale	The locale of the source field. For a list of possible values, see Locale.getAvailableLocales() Method in Java with Examples .	Yes		Minimum length:1

Example

Take the following example log event:

```
{"german_datetime": "Samstag 05. Dezember 1998 11:00:00"}
```

The transformer configuration is this, using `dateTimeConverter` with `parseJSON`:

```
[
  {
    "parseJSON": {}
  },
  {
```

```
    "dateTimeConverter": {
      "source": "german_datetime",
      "target": "target_1",
      "locale": "de",
      "matchPatterns": ["EEEE dd. MMMM yyyy HH:mm:ss"],
      "sourceTimezone": "Europe/Berlin",
      "targetTimezone": "America/New_York",
      "targetFormat": "yyyy-MM-dd'T'HH:mm:ss z"
    }
  }
}
```

The transformed log event would be the following.

```
{
  "german_datetime": "Samstag 05. Dezember 1998 11:00:00",
  "target_1": "1998-12-05T17:00:00 MEZ"
}
```

Transformation metrics and errors

CloudWatch Logs publishes transformation metrics to CloudWatch. These metrics include `TransformedLogEvents`, `TransformedBytes`, and `TransformationErrors`. For more information, see [Log transformer metrics and dimensions](#).

Whenever CloudWatch Logs tries and fails to transform a log event, it adds a `@transformationError` system field to that log event. When you run a CloudWatch Logs Insights query, you will see this field in all log events that had a transformation failure. You can query for this field with a query such as `filter ispresent(@transformationError)` to find all the failed transformation events.

Analyze with Amazon OpenSearch Service

CloudWatch Logs integrates with Amazon OpenSearch Service to enable you to create automatic curated dashboards that display key metrics that OpenSearch Service derives from logs vended from AWS services. The following dashboards are available:

- An **Amazon VPC flow logs dashboard** captures network flow data for Amazon VPC. It helps you analyze network traffic, detect unusual patterns, and monitor resource usage. Key metrics displayed include the following:
 - Total flows and acceptance and rejection of these flows
 - Traffic patterns over time
 - A Sankey diagram that illustrates data flow between source and destination IPs (top talkers)
 - Top IPs by bytes and packets transferred

Note

Currently only VPC version 2 fields format is supported.

- An **AWS WAF logs dashboard** provides insights into web traffic being monitored by AWS WAF. This dashboard helps you identify traffic patterns, blocked requests, and potential threats from specific regions or IPs. Key metrics displayed include the following:
 - Total requests, including by “ALLOW” and “BLOCK” counts.
 - Request history over time, displaying allowed and blocked requests.
 - Breakdowns of requests by Web ACL name, blocked requests by terminating rule, and source IPs.
 - A geographic distribution of request origins.
 - Top client IPs and terminating rules by request count.
- A **CloudTrail logs** dashboard provides an overview of API activity within your AWS environment using CloudTrail logs. It’s useful for monitoring API activity, auditing actions, and identifying potential security or compliance issues. Key metrics displayed include the following:
 - Total event count and event history over time
 - A breakdown of events by account IDs, categories, and Regions.
 - Top APIs, services, and source IPs involved in generating events.

- A table of the top users that are generating events, detailing user account information and event counts.
- An **AWS Network Firewall** dashboard provides enhanced visibility into network traffic, offering valuable insights for security monitoring and analysis. This dashboard offers a comprehensive view of various network metrics and patterns, to quickly identify potential security issues and optimize network configurations. Key metrics displayed include the following:
 - Top talkers and protocols
 - Insights into PrivateLink endpoints
 - Allowed and blocked TLS Server Name Indication traffic

The metrics displayed in these curated dashboards are derived from Amazon OpenSearch Service analytics.

Before you can view these dashboards, you must create an IAM role and perform a one-time integration of CloudWatch Logs with Amazon OpenSearch Service. This one-time integration configures the Amazon OpenSearch Service resources needed to create and render the dashboard. You will incur charges for the OpenSearch services used. For more information, see [Amazon CloudWatch Pricing](#).

You can create these curated dashboards only for log groups in the Standard Log Class.

Important

Don't use [log transformers](#) for any log groups that you want to create vended logs dashboards for. Transforming log events will cause the dashboards to have empty data.

Topics

- [Step 1: Create the integration with OpenSearch Service](#)
- [Step 2: Create vended logs dashboards](#)
- [View, edit, or delete vended logs dashboards](#)
- [IAM policies for users](#)
- [Permissions that the integration needs](#)

Step 1: Create the integration with OpenSearch Service

The first step is creating the integration with OpenSearch Service, which you need to do only once. Creating the integration will create the following resources in your account.

- [An OpenSearch Service time series collection](#) without high availability.

A collection is a set of OpenSearch Service *indexes* that work together to support a workload.

- **Two security policies** for the collection. One defines the encryption type, which is either with a customer managed AWS KMS key or a service owned key. The other policy defines network access, allowing the OpenSearch Service application to access the collection. For more information, see [Encryption of data at rest for Amazon OpenSearch Service](#).
- [An OpenSearch Service data access policy](#) that defines who can access data in the collection.
- [An OpenSearch Service direct query data source](#) with CloudWatch Logs defined as the source.
- [An OpenSearch Service application](#) with the name `aws-analytics`. The application will be configured to allow the creation of a workspace. If an application named `aws-analytics` already exists, it will be updated to add this collection as a data source.
- [A OpenSearch Service workspace](#) that will host the dashboards and allows everyone who has been granted access to read from the workspace.

Topics

- [Required permissions](#)
- [Create the integration](#)

Required permissions

To create the integration, you must be signed on to an account that has the **CloudWatchOpenSearchDashboardsFullAccess** managed IAM policy or equivalent permissions, shown here. You must also have these permissions to delete the integration, create, edit, and delete dashboards, and to refresh the dashboard manually.

JSON

```
{  
  "Version": "2012-10-17",
```



```

    "Statement": [{
      "Sid": "CloudWatchOpenSearchDashboardsIntegration",
      "Effect": "Allow",
      "Action": [
        "logs:ListIntegrations",
        "logs:GetIntegration",
        "logs>DeleteIntegration",
        "logs:PutIntegration",
        "logs:DescribeLogGroups",
        "opensearch:ApplicationAccessAll",
        "iam:ListRoles",
        "iam:ListUsers"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsOpensearchReadAPIs",
      "Effect": "Allow",
      "Action": [
        "aoss:BatchGetCollection",
        "aoss:BatchGetLifecyclePolicy",
        "es:ListApplications"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaFirst": "logs.amazonaws.com"
        }
      }
    },
    {
      "Sid": "CloudWatchLogsOpensearchCreateServiceLinkedAccess",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": "arn:aws:iam::*:role/aws-service-role/
opensearchservice.amazonaws.com/AWSServiceRoleForAmazonOpenSearchService",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "opensearchservice.amazonaws.com",
          "aws:CalledViaFirst": "logs.amazonaws.com"
        }
      }
    }
  ]
}

```

```

    },
    {
      "Sid": "CloudWatchLogsObservabilityCreateServiceLinkedAccess",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": "arn:aws:iam::*:role/aws-service-role/
observability.aoss.amazonaws.com/AWSServiceRoleForAmazonOpenSearchServerless",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "observability.aoss.amazonaws.com",
          "aws:CalledViaFirst": "logs.amazonaws.com"
        }
      }
    },
    {
      "Sid": "CloudWatchLogsCollectionRequestAccess",
      "Effect": "Allow",
      "Action": [
        "aoss:CreateCollection"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaFirst": "logs.amazonaws.com",
          "aws:RequestTag/CloudWatchOpenSearchIntegration": [
            "Dashboards"
          ]
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": "CloudWatchOpenSearchIntegration"
        }
      }
    },
    {
      "Sid": "CloudWatchLogsApplicationRequestAccess",
      "Effect": "Allow",
      "Action": [
        "es:CreateApplication"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {

```

```

        "aws:CalledViaFirst": "logs.amazonaws.com",
        "aws:RequestTag/OpenSearchIntegration": [
            "Dashboards"
        ]
    },
    "ForAllValues:StringEquals": {
        "aws:TagKeys": "OpenSearchIntegration"
    }
},
{
    "Sid": "CloudWatchLogsCollectionResourceAccess",
    "Effect": "Allow",
    "Action": [
        "aoss:DeleteCollection"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaFirst": "logs.amazonaws.com",
            "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                "Dashboards"
            ]
        }
    }
},
{
    "Sid": "CloudWatchLogsApplicationResourceAccess",
    "Effect": "Allow",
    "Action": [
        "es:UpdateApplication",
        "es:GetApplication"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaFirst": "logs.amazonaws.com",
            "aws:ResourceTag/OpenSearchIntegration": [
                "Dashboards"
            ]
        }
    }
},
{

```

```

        "Sid": "CloudWatchLogsCollectionPolicyAccess",
        "Effect": "Allow",
        "Action": [
            "aoss:CreateSecurityPolicy",
            "aoss:CreateAccessPolicy",
            "aoss:DeleteAccessPolicy",
            "aoss:DeleteSecurityPolicy",
            "aoss:GetAccessPolicy",
            "aoss:GetSecurityPolicy"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "aoss:collection": "cloudwatch-logs-*",
                "aws:CalledViaFirst": "logs.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchLogsAPIAccessAll",
        "Effect": "Allow",
        "Action": [
            "aoss:APIAccessAll"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "aoss:collection": "cloudwatch-logs-*"
            }
        }
    },
    {
        "Sid": "CloudWatchLogsIndexPolicyAccess",
        "Effect": "Allow",
        "Action": [
            "aoss:CreateAccessPolicy",
            "aoss:DeleteAccessPolicy",
            "aoss:GetAccessPolicy",
            "aoss:CreateLifecyclePolicy",
            "aoss:DeleteLifecyclePolicy"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {

```

```

        "aoss:index": "cloudwatch-logs-*",
        "aws:CalledViaFirst": "logs.amazonaws.com"
    }
},
{
    "Sid": "CloudWatchLogsDQSRequestQueryAccess",
    "Effect": "Allow",
    "Action": [
        "es:AddDirectQueryDataSource"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/cloudwatch_logs_*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaFirst": "logs.amazonaws.com",
            "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                "Dashboards"
            ]
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "CloudWatchOpenSearchIntegration"
        }
    }
},
{
    "Sid": "CloudWatchLogsStartDirectQueryAccess",
    "Effect": "Allow",
    "Action": [
        "opensearch:StartDirectQuery",
        "opensearch:GetDirectQuery"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/cloudwatch_logs_*"
},
{
    "Sid": "CloudWatchLogsDQSResourceQueryAccess",
    "Effect": "Allow",
    "Action": [
        "es:GetDirectQueryDataSource",
        "es>DeleteDirectQueryDataSource"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/cloudwatch_logs_*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaFirst": "logs.amazonaws.com",

```

```

        "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
            "Dashboards"
        ]
    },
    {
        "Sid": "CloudWatchLogsPassRoleAccess",
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "iam:PassedToService":
"directquery.opensearchservice.amazonaws.com",
                "aws:CalledViaFirst": "logs.amazonaws.com"
            }
        },
    },
    {
        "Sid": "CloudWatchLogsAossTagsAccess",
        "Effect": "Allow",
        "Action": [
            "aoss:TagResource"
        ],
        "Resource": "arn:aws:aoss:*:*:collection/*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaFirst": "logs.amazonaws.com",
                "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                    "Dashboards"
                ]
            },
            "ForAllValues:StringEquals": {
                "aws:TagKeys": "CloudWatchOpenSearchIntegration"
            }
        },
    },
    {
        "Sid": "CloudWatchLogsEsApplicationTagsAccess",
        "Effect": "Allow",
        "Action": [

```

```

        "es:AddTags"
    ],
    "Resource": "arn:aws:opensearch:*:*:application/*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/OpenSearchIntegration": [
                "Dashboards"
            ],
            "aws:CalledViaFirst": "logs.amazonaws.com"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "OpenSearchIntegration"
        }
    }
},
{
    "Sid": "CloudWatchLogsEsDataSourceTagsAccess",
    "Effect": "Allow",
    "Action": [
        "es:AddTags"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                "Dashboards"
            ],
            "aws:CalledViaFirst": "logs.amazonaws.com"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "CloudWatchOpenSearchIntegration"
        }
    }
}
]
}

```

Create the integration

Use these steps to create the integration.

To integrate CloudWatch Logs with Amazon OpenSearch Service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs Insights** and then choose the **Analyze with OpenSearch** tab.
3. Choose **Create integration**.
4. For **Integration name**, enter a name for the integration.
5. (Optional) To encrypt the data written to OpenSearch Service Serverless, enter the ARN of the AWS KMS key that you want to use in **KMS key ARN**. For more information, see [Encryption at rest](#) in the Amazon OpenSearch Service Developer Guide.
6. For **Data retention**, enter the amount of time that you want the OpenSearch Service data indexes to be retained. This also defines the maximum time period for which you can view data in the dashboards. Choosing a longer data retention period will incur additional searching and indexing costs. For more information, see [OpenSearch Service Serverless Pricing](#).

The maximum retention period is 30 days.

The data retention length will also be used to create the OpenSearch Service collection lifecycle policy.

7. For **IAM role for writing to OpenSearch collection**, create a new IAM role or select an existing IAM role to be used to write to the OpenSearch Service collection.

Creating a new role is the simplest method, and the role will be created with the necessary permissions.

Note

If you create a role, it will have permissions to read from all log groups in the account.

If you want to select an existing role, it should have the permissions listed in [Permissions that the integration needs](#). Alternatively, you can choose **Use an existing role** and then in the **Verify access permissions of the selected role** section you can choose **Create role**. This way you can use the permissions listed in [Permissions that the integration needs](#) as a template and modify it. For example, if you want to specify a finer-grain control of log groups.

8. For **IAM roles and users who can view dashboards**, you select how you want to grant access to IAM roles and IAM users for vended logs dashboard access:

- To limit the dashboard access to just some users, choose **Select IAM roles and users who can view dashboards** and then in the text box search for and select the IAM roles and IAM users that you want to grant access to.
- To grant dashboard access to all users, choose **Allow all roles and users in this account to view dashboards**.

 Important

Selecting roles or users, or choosing all users, only adds them to the [data access policy](#) needed for accessing OpenSearch Service collection that stores the dashboard data. **For them to be able to view the vendored logs dashboards, you must also grant those roles and users the [CloudWatchOpenSearchDashboardAccess](#) managed IAM policy.**

9. Choose **Create integration**

Creating the integration will take a few minutes.

Step 2: Create vendored logs dashboards

After you have created the integration, you can create dashboards. Dashboards are available for Amazon VPC flow logs, CloudTrail logs, and AWS WAF logs.

To create a vendored log dashboard with metrics derived by OpenSearch Service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs Insights** and then choose the **Analyze with OpenSearch** tab.
3. Choose **Create dashboard**.
4. Choose which type of logs to create the dashboard for, AWS WAF, Amazon VPC flow logs, CloudTrail, or AWS Network Firewall.
5. Enter a name for the dashboard, and optionally enter a description.
6. For **Data synchronization frequency**, enter how often that you want OpenSearch Service to query CloudWatch so the metrics and indexes created in OpenSearch Service can be synchronized and updated with new data. OpenSearch Service creates metrics and indexes on your logs for rendering the dashboard.

Choosing a shorter time keeps the data more up to date and incurs higher costs.

7. Select the log groups to collect data from for this dashboard. Be sure to select log groups that match the type of dashboard that you are creating.

You can use the **Browse log groups** button and the **View log samples from selected log groups** option as you make these choices, to make sure that you get the log groups that you want.

8. Choose **Create dashboard**.

At first, the dashboard appears without any data. After a few minutes, data will appear in the dashboard. When the data first appears, it will be for the most recent 15 minutes of log entries.

View, edit, or delete vended logs dashboards

View vended logs dashboards in CloudWatch Logs or OpenSearch Service

To be able to view dashboards, you must be signed in to an IAM principal that has the **CloudWatchOpenSearchDashboardAccess** IAM policy.

To view vended log dashboards

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs Insights** and then choose the **Analyze with OpenSearch** tab.
3. Choose the dashboard in the **OpenSearch dashboards** box.
4. (Optional) In the upper right, choose **View in OpenSearch**.

The OpenSearch Service console opens and you see the same dashboard there. In the OpenSearch Service console, you can make changes to the dashboard and its widgets, and these changes will also be visible when you view the dashboard in CloudWatch Logs.

Grant dashboard viewing access to additional IAM roles or IAM users

To grant access to additional IAM principals after you've created the integration, take the following steps.

To grant vended log dashboard access to additional IAM roles or users

1. Edit the data access policy for the collection to add these roles or users. For more information, see [Data access control for Amazon OpenSearch Service Serverless](#) in the OpenSearch Service Developer Guide.
2. Grant the **CloudWatchOpenSearchDashboardAccess** to these users. For more information about the contents of this policy, see [CloudWatchOpenSearchDashboardAccess](#).

Edit dashboard configuration

You can edit the name, description, and synchronization frequency of existing vended log dashboards.

To edit a vended log dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs Insights** and then choose the **Analyze with OpenSearch** tab.
3. Choose the dashboard in the **OpenSearch dashboards** box.
4. Choose **Actions, Change dashboard details**.
5. Make your changes, then choose **Confirm changes**.

Delete a vended log dashboard

You can delete a vended log dashboard. If you do so, the dashboard, the metrics, and indexes created in the OpenSearch Service collection are all deleted.

Note

After you delete a vended log dashboard, wait at least six hours before trying to re-create that same dashboard. If you don't wait, the re-created dashboard won't work correctly.

To delete a vended log dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs Insights** and then choose the **Analyze with OpenSearch** tab.
3. Choose the dashboard in the **OpenSearch dashboards** box.
4. Choose **Actions, Delete**.
5. Confirm your decision by entering **delete**, then choose **Delete**.

Delete all vended log dashboard integration with OpenSearch Service

You can delete your entire OpenSearch integration. If you do, all vended logs dashboards and the data that was displayed in them is deleted.

Important

To avoid ongoing costs, we strongly recommend that you manually delete the following resources before you delete the integration. Deleting the integration doesn't automatically delete these resources, and after you delete the integration you won't be able to access these resources to delete them. To find the names of the resources to delete, see the following procedure.

- [The data source](#)
- [The collection](#)
- [The data access policy](#)
- [The encryption policy](#)
- [The network policy](#)
- [The lifecycle policy](#)

To delete your entire vended log dashboard integration with OpenSearch Service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Settings**.
3. Choose the **Logs** tab.

4. In the **OpenSearch integration** section, choose **Delete integration**.

The next screen displays the names of the OpenSearch Service resources that you should delete before deleting the integration.

5. Confirm your decision by entering **delete**, then choose **Delete integration**.

IAM policies for users

CloudWatch Logs has created two IAM policies, **CloudWatchOpenSearchDashboardsFullAccess** and **CloudWatchOpenSearchDashboardAccess**. The following table lists which actions each of these policies enables.

Action	IAM policy	Additional permissions needed
Create integration	CloudWatchOpenSearchDashboardsFullAccess	
Delete integration	CloudWatchOpenSearchDashboardsFullAccess	
Create dashboard	CloudWatchOpenSearchDashboardsFullAccess	
Edit dashboard	CloudWatchOpenSearchDashboardsFullAccess	
Delete dashboard	CloudWatchOpenSearchDashboardsFullAccess	
Refresh dashboard using Synchronize now	CloudWatchOpenSearchDashboardsFullAccess	
View integration in Settings	CloudWatchOpenSearchDashboardAccess or CloudWatchOpenSearchDashboardsFullAccess	

Action	IAM policy	Additional permissions needed
View dashboard	CloudWatchOpenSearchDashboardAccess or CloudWatchOpenSearchDashboardsFullAccess	Specify the role or user when you create the integration, or edit the data access policy for the collection to add these roles or users. For more information, see Data access control for Amazon OpenSearch Service Serverless in the OpenSearch Service Developer Guide.
View dashboard in OpenSearch Service console	CloudWatchOpenSearchDashboardAccess or CloudWatchOpenSearchDashboardsFullAccess	Specify the role or user when you create the integration, or edit the data access policy for the collection to add these roles or users. For more information, see Data access control for Amazon OpenSearch Service Serverless in the OpenSearch Service Developer Guide.

Permissions that the integration needs

If you create an IAM role for the integration to use, instead of allowing CloudWatch Logs to create the role, it must include the following permissions and trust policy. For more information about how to create an IAM role, see [Create a role to delegate permissions to an AWS service](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

{
  "Sid": "CloudWatchLogsAccess",
  "Effect": "Allow",
  "Action": [
    "logs:StartQuery",
    "logs:GetLogGroupFields",
    "logs:GetQueryResults"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Sid": "CloudWatchLogsDescribeLogGroupsAccess",
  "Effect": "Allow",
  "Action": [
    "logs:DescribeLogGroups"
  ],
  "Resource": "*"
},
{
  "Sid": "AmazonOpenSearchCollectionAccess",
  "Effect": "Allow",
  "Action": [
    "aoss:APIAccessAll"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aoss:collection": "cloudwatch-logs-*"
    }
  }
}
]
}

```

Note

The previous role grants access to read from all log groups in the account, to enable you to create dashboards for any log account, including cross-account log groups. If you want to restrict access to specific log groups and create dashboards for only those log groups, you can update the first statement in that policy to the following:

```
{
  "Sid": "CloudWatchLogsAccess",
  "Effect": "Allow",
  "Action": [
    "logs:StartQuery",
    "logs:GetLogGroupFields",
    "logs:GetQueryResults"
  ],
  "Resource": [
    "arn:aws:logs:us-east-1:123456789012:log-group:myLogGroup:*",
    "arn:aws:logs:us-east-1:123456789012:log-group:myLogGroup"
  ]
}
```


Access logs with S3 Tables Integration

The S3 Tables Integration with CloudWatch allows you to access log data ingested into CloudWatch using analytics engines such as Amazon Athena, Amazon Redshift, and third-party tools that support connection to Apache Iceberg-compatible stores. This integration enables you to perform comprehensive log analysis using tools of your preference and correlate data in CloudWatch Logs with non-CloudWatch data.

Understanding S3 Tables Integration

Amazon S3 Tables Integration is a fully managed solution that makes your logs in CloudWatch Logs available as managed Amazon S3 tables. With this integration, you gain greater flexibility on how you analyze your logs in addition to CloudWatch Logs features.

The integration works by creating a managed Amazon S3 table bucket (`aws-cloudwatch`) and associating specific log sources with Amazon S3 Tables based on data source name and type (that can be managed from the **Log Management > Data Sources** tab in CloudWatch Logs Console). Once associated, CloudWatch Logs data becomes accessible through Amazon S3 Tables using the Apache Iceberg format. This format provides a standardized way for various analytics engines to query the data efficiently.

Core Components

Data Source Association

The process of linking specific CloudWatch Logs sources to the S3 Tables integration based on data source and type criteria.

Apache Iceberg Tables

The underlying table format used by S3 Tables that provides structured data storage and enables compatibility with multiple analytics engines.

Data flow to S3 tables

Understanding how data flows between CloudWatch Logs and S3 Tables helps you plan your integration and manage your log data effectively.

When you create an association, CloudWatch Logs automatically sends new log events that match the associated data source name and type to a CloudWatch-managed S3 table bucket. You can find these events in the logs namespace under the corresponding table for that data source. The integration processes only log events added after you create the association and does not backfill logs from before the association was created.

Data retention in the S3 table bucket matches the retention policy set for the log group. For example, if you set a log group to 1-day retention, CloudWatch Logs removes the data from both CloudWatch Logs and the S3 Table after one day. When you delete a log group or log stream, CloudWatch Logs also removes the data from the S3 table bucket.

When to Use S3 Tables Integration

Consider using S3 Tables integration to correlate log data with other external or non-CloudWatch data or when you prefer using other analytics tools such as Amazon Athena to perform analytics on CloudWatch Logs data. Use this integration when you need capabilities that go beyond what's available in CloudWatch Logs. This integration is particularly valuable when:

- You need to run complex SQL-like queries across large volumes of log data
- You want to integrate log analysis with existing analytics workflows and tools
- You require comprehensive log analysis capabilities that span multiple data sources

There are no additional storage or table maintenance charges for S3 tables created through this integration, beyond existing CloudWatch ingestion and storage pricing.

Prerequisites

Before implementing the integration, ensure you have the following:

- Existing CloudWatch Logs data
- Appropriate IAM permissions for cross-service access between CloudWatch Logs and S3 Tables, as described in the following section

IAM permissions

To integrate CloudWatch Logs with S3 Tables, you need to configure IAM permissions for two separate entities: the user or role that sets up the integration, and the service role that CloudWatch Logs assumes to write data to S3 Tables.

For the role or user creating the integration

The user or role that sets up the integration requires the following permissions:

- `observabilityadmin:CreateS3TableIntegration` to create the integration and `logs:AssociateSourceToS3TableIntegration` to add sources
- `s3tables:CreateTableBucket`, `s3tables:PutTableBucketEncryption`, and `s3tables:PutTableBucketPolicy` to configure the S3 table bucket

For the service role

Attach the following IAM policy to the IAM service role that CloudWatch Logs uses to write data to the table bucket. This policy grants permission to write to the tables. Replace *aws-region*, *123456789012*, and *log-group-name* with your AWS Region, account ID, and log group name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:integrateWithS3Table"
      ],
      "Resource": ["arn:aws:logs:aws-region:123456789012:log-group:log-group-name"],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Attach the following trust policy to the IAM service role that CloudWatch Logs will assume to write log data to S3 Tables. You create or select this role during the integration setup. The conditions restrict the role so that CloudWatch Logs can only assume it for the specified account and log group. Replace *aws-region*, *123456789012*, and *log-group-name* with your AWS Region, account ID, and log group name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": ["arn:aws:logs:aws-region:123456789012:log-
group:log-group-name"]
        }
      }
    }
  ]
}
```

KMS key policy (for encrypted data)

If you use a customer managed key to encrypt your log data, you must grant the CloudWatch service principal and the S3 Tables maintenance service principal access to the key. Add the following statements to your KMS key policy. Replace the placeholder values with your AWS account ID, Region, KMS key ID, and S3 table or table bucket ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnableSystemTablesKeyUsage",
      "Effect": "Allow",
```

```

    "Principal": {
      "Service": "systemtables.cloudwatch.amazonaws.com"
    },
    "Action": [
      "kms:DescribeKey",
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:aws-region:123456789012:key/key-id",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  },
  {
    "Sid": "EnableKeyUsage",
    "Effect": "Allow",
    "Principal": {
      "Service": "maintenance.s3tables.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:aws-region:123456789012:key/key-id",
    "Condition": {
      "StringLike": {
        "kms:EncryptionContext:aws:s3:arn": "<table-or-table-bucket-arn>/*"
      }
    }
  }
]
}

```

Getting Started

To get started with S3 Tables Integration, you need to set up the integration between your CloudWatch Logs and S3 Tables. This process involves configuring data source associations and setting up appropriate IAM permissions.

To create an S3 Tables Integration

1. Open the CloudWatch Logs console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Settings, Global, Create S3 Table Integration**.
3. Customize how logs will be encrypted in S3 Tables, and the role that CloudWatch Logs will use to write your logs into S3 Tables.
4. Choose **Create S3 Table Integration**.

To associate sources to an S3 Table Integration

1. Open the CloudWatch Logs console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Settings, Global, Manage S3 Table Integration**.
3. Choose **Associate data source**.
4. Select the data source name and data source type that you want to enable integration for.
5. Choose **Associate data source**.

To associate sources to an S3 Table Integration from the Log Management Page

1. Open the CloudWatch Logs console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Log Management** in the navigation pane.
3. Select **Data Sources** tab.
4. Choose the data source name and data source type that you want to integrate.
5. Choose **Data source actions**.
6. Select **Associate with S3 Tables Integration**.
7. Review the data sources, and then choose **Associate Data source**.

Creating metrics from log events using filters

You can search and filter the log data coming into CloudWatch Logs by creating one or more *metric filters*. Metric filters define the terms and patterns to look for in log data as it is sent to CloudWatch Logs. CloudWatch Logs uses these metric filters to turn log data into numerical CloudWatch metrics that you can graph or set an alarm on.

When you create a metric from a log filter, you can also choose to assign dimensions and a unit to the metric. If you specify a unit, be sure to specify the correct one when you create the filter. Changing the unit for the filter later will have no effect.

If you have configured AWS Organizations and are working with member accounts you can use log centralization to collect log data from source accounts into a central monitoring account.

When working with centralized log groups you can use these system fields dimensions when creating metric filters.

- `@aws.account` - This dimension represents the AWS account ID from which the log event originated.
- `@aws.region` - This dimension represents the AWS region where the log event was generated.

These dimensions help in identifying the source of log data, allowing for more granular filtering and analysis of metrics derived from centralized logs. For more information, see [Cross-account cross-Region log centralization](#).

If a log group with a subscription uses log transformation, the filter pattern is applied to the transformed versions of the log events. For more information, see [Transform logs during ingestion](#).

Note

Metric filters are supported only for log groups in the Standard log class. For more information about log classes, see [Log classes](#).

You can use any type of CloudWatch statistic, including percentile statistics, when viewing these metrics or setting alarms.

Note

Percentile statistics are supported for a metric only if none of the metric's values are negative. If you set up your metric filter so that it can report negative numbers, percentile statistics will not be available for that metric when it has negative numbers as values. For more information, see [Percentiles](#).

Filters do not retroactively filter data. Filters only publish the metric data points for events that happen after the filter was created. When testing a filter pattern, the **Filter results** preview shows up to the first 50 matching log lines for validation purposes. If the timestamp on the filtered results is earlier than the metric creation time, no logs are displayed.

Contents

- [Concepts](#)
- [Filter pattern syntax for metric filters](#)
- [Creating metric filters](#)
- [Listing metric filters](#)
- [Deleting a metric filter](#)

Concepts

Each metric filter is made up of the following key elements:

default value

The value reported to the metric filter during a period when logs are ingested but no matching logs are found. By setting this to 0, you ensure that data is reported during every such period, preventing "spotty" metrics with periods of no matching data. If no logs are ingested during a one-minute period, then no value is reported.

If you assign dimensions to a metric created by a metric filter, you can't assign a default value for that metric.

dimensions

Dimensions are the key-value pairs that further define a metric. You can assign dimensions to the metric created from a metric filter. Because dimensions are part of the unique identifier for

a metric, whenever a unique name/value pair is extracted from your logs, you are creating a new variation of that metric.

filter pattern

A symbolic description of how CloudWatch Logs should interpret the data in each log event. For example, a log entry may contain timestamps, IP addresses, strings, and so on. You use the pattern to specify what to look for in the log file.

metric name

The name of the CloudWatch metric to which the monitored log information should be published. For example, you may publish to a metric called `ErrorCount`.

metric namespace

The destination namespace of the new CloudWatch metric.

metric value

The numerical value to publish to the metric each time a matching log is found. For example, if you're counting the occurrences of a particular term like "Error", the value will be "1" for each occurrence. If you're counting the bytes transferred, you can increment by the actual number of bytes found in the log event.

Filter pattern syntax for metric filters

Note

How metric filters differ from CloudWatch Logs Insights queries

Metric filters differ from CloudWatch Logs Insights queries in that a specified numerical value is added to a metric filter each time a matching log is found. For more information, see [Configuring metric values for a metric filter](#).

For information about how to query your log groups with the Amazon CloudWatch Logs Insights query language, see [CloudWatch Logs Insights language query syntax](#).

Generic filter pattern examples

For more information on generic filter pattern syntax applicable to metric filters as well as [subscription filters](#) and [filter log events](#), see [Filter pattern syntax for metric filters, subscription filters, and filter log events](#), which includes the following examples:

- Supported regular expressions (regex) syntax

- Matching terms in unstructured log events
- Matching terms in JSON log events
- Matching terms in space-delimited log events

Metric filters allow you to search and filter log data coming into CloudWatch Logs, extract metric observations from the filtered log data, and transform the data points into a CloudWatch Logs metric. You define the terms and patterns to look for in log data as it is sent to CloudWatch Logs. Metric filters are assigned to log groups, and all of the filters assigned to a log group are applied to their log streams.

When a metric filter matches a term, it increments the metric's count by a specified numerical value. For example, you can create a metric filter that counts the number of times the word **ERROR** occurs in your log events.

You can assign units of measure and dimensions to metrics. For example, if you create a metric filter that counts the number of times the word **ERROR** occurs in your log events, you can specify a dimension that's called `ErrorCode` to show the total number of log events that contain the word **ERROR** and filter data by reported error codes.

 Tip

When you assign a unit of measure to a metric, make sure to specify the correct one. If you change the unit later, your change might not take effect. For the complete list of the units that CloudWatch supports, see [MetricDatum](#) in the Amazon CloudWatch API Reference.

Topics

- [Configuring metric values for a metric filter](#)
- [Publishing dimensions with metrics from values in JSON or space-delimited log events](#)
- [Using values in log events to increment a metric's value](#)

Configuring metric values for a metric filter

When you create a metric filter, you define your filter pattern and specify your metric's value and default value. You can set metric values to numbers, named identifiers, or numeric identifiers. If

you don't specify a default value, CloudWatch won't report data when your metric filter doesn't find a match. We recommend that you specify a default value, even if the value is 0. Setting a default value helps CloudWatch report data more accurately and prevents CloudWatch from aggregating spotty metrics. CloudWatch aggregates and reports metric values every minute.

When your metric filter finds a match in your log events, it increments your metric's count by your metric's value. If your metric filter doesn't find a match, CloudWatch reports the metric's default value. For example, your log group publishes two records every minute, the metric value is 1, and the default value is 0. If your metric filter finds matches in both log records within the first minute, the metric value for that minute is 2. If your metric filter doesn't find matches in either records during the second minute, the default value for that minute is 0. If you assign dimensions to metrics that metric filters generate, you can't specify default values for those metrics.

You also can set up a metric filter to increment a metric with a value extracted from a log event, instead of a static value. For more information, see [Using values in log events to increment a metric's value](#).

Publishing dimensions with metrics from values in JSON or space-delimited log events

You can use the CloudWatch console or AWS CLI to create metric filters that publish dimensions with metrics that JSON and space-delimited log events generate. Dimensions are name/value value pairs and only available for JSON and space-delimited filter patterns. You can create JSON and space-delimited metric filters with up to three dimensions. For more information about dimensions and information about how to assign dimensions to metrics, see the following sections:

- [Dimensions](#) in the *Amazon CloudWatch User guide*
- [Example: Extract fields from an Apache log and assign dimensions](#) in the *Amazon CloudWatch Logs User Guide*

Important

Dimensions contain values that gather charges the same as custom metrics. To prevent unexpected charges, don't specify high-cardinality fields, such as `IPAddress` or `requestID`, as dimensions.

If you extract metrics from log events, you're charged for custom metrics. To prevent you from collecting accidental high charges, Amazon might disable your metric filter if it

generates 1000 different name/value pairs for specified dimensions over a certain amount of time.

You can create billing alarms that notify you of your estimated charges. For more information, see [Creating a billing alarm to monitor your estimated AWS charges](#).

Publishing dimensions with metrics from JSON log events

The following examples contain code snippets that describe how to specify dimensions in a JSON metric filter.

Example: JSON log event

```
{
  "eventType": "UpdateTrail",
  "sourceIPAddress": "111.111.111.111",
  "arrayKey": [
    "value",
    "another value"
  ],
  "objectList": [
    {"name": "a",
     "id": 1
    },
    {"name": "b",
     "id": 2
    }
  ]
}
```

Note

If you test the example metric filter with the example JSON log event, you must enter the example JSON log on a single line.

Example: Metric filter

The metric filter increments the metric whenever a JSON log event contain the properties `eventType` and `"sourceIPAddress"`.

```
{ $.eventType = "*" && $.sourceIPAddress != 123.123.* }
```

When you create a JSON metric filter, you can specify any of the properties in the metric filter as a dimension. For example, to set `eventType` as a dimension, use the following:

```
"eventType" : $.eventType
```

The example metric contains a dimension that's named `"eventType"`, and the dimension's value in the example log event is `"UpdateTrail"`.

Publishing dimensions with metrics from space-delimited log events

The following examples contain code snippets that describe how to specify dimensions in a space-delimited metric filter.

Example: Space-delimited log event

```
127.0.0.1 Prod frank [10/Oct/2000:13:25:15 -0700] "GET /index.html HTTP/1.0" 404  
1534
```

Example: Metric filter

```
[ip, server, username, timestamp, request, status_code, bytes > 1000]
```

The metric filter increments the metric when a space-delimited log event includes any of the fields that are specified in the filter. For example, the metric filter finds following fields and values in the example space-delimited log event.

```
{
  "$bytes": "1534",
  "$status_code": "404",

  "$request": "GET /index.html HTTP/1.0",
  "$timestamp": "10/Oct/2000:13:25:15 -0700",
  "$username": "frank",
  "$server": "Prod",
  "$ip": "127.0.0.1"
}
```

When you create a space-delimited metric filter, you can specify any of the fields in the metric filter as a dimension. For example, to set `server` as a dimension, use the following:

```
"server" : $server
```

The example metric filter has a dimension that's named `server`, and the dimension's value in the example log event is `"Prod"`.

Example: Match terms with AND (&&) and OR (||)

You can use the logical operators AND ("`&&`") and OR ("`||`") to create space-delimited metric filters that contain conditions. The following metric filter returns log events where the first word in the events is `ERROR` or any superstring of `WARN`.

```
[w1=ERROR || w1=%WARN%, w2]
```

Using values in log events to increment a metric's value

You can create metric filters that publish numeric values found in your log events. The procedure in this section uses the following example metric filter to show how you can publish a numeric value in a JSON log event to a metric.

```
{ $.latency = * } metricValue: $.latency
```

To create a metric filter that publishes a value in a log event

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Log groups**.
3. Select or create a log group.

For information about how to create a log group, see [Create a log group in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

4. Choose **Actions**, and then choose **Create metric filter**.
5. For **Filter Pattern**, enter `{ $.latency = * }`, and then choose **Next**.
6. For **Metric Name**, enter **myMetric**.
7. For **Metric Value**, enter `$.latency`.
8. (Optional) For **Default Value**, enter **0**, and then choose **Next**.

We recommend that you specify a default value, even if the value is 0. Setting a default value helps CloudWatch report data more accurately and prevents CloudWatch from aggregating spotty metrics. CloudWatch aggregates and reports metric values every minute.

9. Choose **Create metric filter**.

The example metric filter matches the term "latency" in the example JSON log event and publishes a numeric value of 50 to the metric **myMetric**.

```
{
  "latency": 50,
  "requestType": "GET"
}
```

Creating metric filters

The following procedure and examples show how to create metric filters.

Examples

- [Create a metric filter for a log group](#)
- [Example: Count log events](#)
- [Example: Count occurrences of a term](#)

- [Example: Count HTTP 404 codes](#)
- [Example: Count HTTP 4xx codes](#)
- [Example: Extract fields from an Apache log and assign dimensions](#)

Create a metric filter for a log group

To create a metric filter for a log group, follow these steps. The metric won't be visible until there are some data points for it.

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Logs**, and then choose **Log groups**.
3. Choose the name of the log group.
4. Choose **Actions**, and then choose **Create metric filter**.
5. For **Filter pattern**, enter a filter pattern. For more information, see [Filter pattern syntax for metric filters, subscription filters, filter log events, and Live Tail](#).
6. (Optional) If you are using centralized log groups, under **Filter selection criteria**, you can specify filters based on source account (@aws.account), source region (@aws.region), or both conditions.
7. (Optional) To test your filter pattern, under **Test Pattern**, enter one or more log events to test the pattern. Each log event must be formatted on one line. Line breaks are used to separate log events in the **Log event messages** box.
8. Choose **Next**, and then enter a name for your metric filter.
9. Under **Metric details**, for **Metric namespace**, enter a name for the CloudWatch namespace where the metric will be published. If the namespace doesn't already exist, make sure that **Create new** is selected.
10. For **Metric name**, enter a name for the new metric.
11. For **Metric value**, if your metric filter is counting occurrences of the keywords in the filter, enter 1. This increments the metric by 1 for each log event that includes one of the keywords.

Alternatively, enter a token, such as **\$size**. This increments the metric by the value of the number in the **size** field for every log event that contains a **size** field.
12. (Optional) For **Unit**, select a unit to assign to the metric. If you do not specify a unit, the unit is set as None.

13. (Optional) Enter the names and tokens for as many as three dimensions for the metric. If you assign dimensions to metrics that metric filters create, you cannot assign default values for those metrics.

 Note

Dimensions are supported only in JSON or space-delimited metric filters.

14. Choose **Create metric filter**. You can find the metric filter that you created from the navigation pane. Choose **Logs**, and then choose **Log groups**. Choose the name of the log group that you created your metric filter for, and then select the **Metric filters** tab.

Example: Count log events

The simplest type of log event monitoring is to count the number of log events that occur. You might want to do this to keep a count of all events, to create a "heartbeat" style monitor or just to practice creating metric filters.

In the following CLI example, a metric filter called MyAppAccessCount is applied to the log group MyApp/access.log to create the metric EventCount in the CloudWatch namespace MyNamespace. The filter is configured to match any log event content and to increment the metric by "1".

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of a log group.
4. Choose **Actions**, **Create metric filter**.
5. Leave **Filter Pattern** and **Select Log Data to Test** blank.
6. Choose **Next**, and then for **Filter Name**, type **EventCount**.
7. Under **Metric Details**, for **Metric Namespace**, type **MyNameSpace**.
8. For **Metric Name**, type **MyAppEventCount**.
9. Confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every log event.

10. For **Default Value** enter 0, and then choose **Next**. Specifying a default value ensures that data is reported even during periods when no log events occur, preventing spotty metrics where data sometimes does not exist.
11. Choose **Create metric filter**.

To create a metric filter using the AWS CLI

At a command prompt, run the following command:

```
aws logs put-metric-filter \  
  --log-group-name MyApp/access.log \  
  --filter-name EventCount \  
  --filter-pattern " " \  
  --metric-transformations \  
    metricName=MyAppEventCount,metricNamespace=MyNamespace,metricValue=1,defaultValue=0
```

You can test this new policy by posting any event data. You should see data points published to the metric `MyAppAccessEventCount`.

To post event data using the AWS CLI

At a command prompt, run the following command:

```
aws logs put-log-events \  
  --log-group-name MyApp/access.log --log-stream-name TestStream1 \  
  --log-events \  
    timestamp=1394793518000,message="Test event 1" \  
    timestamp=1394793518000,message="Test event 2" \  
    timestamp=1394793528000,message="This message also contains an Error"
```

Example: Count occurrences of a term

Log events frequently include important messages that you want to count, maybe about the success or failure of operations. For example, an error may occur and be recorded to a log file if a given operation fails. You may want to monitor these entries to understand the trend of your errors.

In the example below, a metric filter is created to monitor for the term `Error`. The policy has been created and added to the log group **MyApp/message.log**. CloudWatch Logs publishes a data point

to the CloudWatch custom metric **ErrorCount** in the **MyApp/message.log** namespace with a value of "1" for every event containing **Error**. If no event contains the word **Error**, then a value of 0 is published. When graphing this data in the CloudWatch console, be sure to use the sum statistic.

After you create a metric filter, you can view the metric in the CloudWatch console. When you are selecting the metric to view, select the metric namespace that matches the log group name. For more information, see [Viewing Available Metrics](#).

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the log group.
4. Choose **Actions, Create metric filter**.
5. For **Filter Pattern**, enter **Error**.

Note

All entries in **Filter Pattern** are case-sensitive.

6. (Optional) To test your filter pattern, under **Test Pattern**, enter one or more log events to use to test the pattern. Each log event must be within one line, because line breaks are used to separate log events in the **Log event messages** box.
7. Choose **Next**, and then on the **Assign metric** page, for **Filter Name**, type **MyAppErrorCount**.
8. Under **Metric Details**, for **Metric Namespace**, type **MyNameSpace**.
9. For **Metric Name**, type **ErrorCount**.
10. Confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every log event containing "Error".
11. For **Default Value** type 0, and then choose **Next**.
12. Choose **Create metric filter**.

To create a metric filter using the AWS CLI

At a command prompt, run the following command:

```
aws logs put-metric-filter \
```

```
--log-group-name MyApp/message.log \  
--filter-name MyAppErrorCount \  
--filter-pattern 'Error' \  
--metric-transformations \  
    metricName=ErrorCount,metricNamespace=MyNamespace,metricValue=1,defaultValue=0
```

You can test this new policy by posting events containing the word "Error" in the message.

To post events using the AWS CLI

At a command prompt, run the following command. Note that patterns are case-sensitive.

```
aws logs put-log-events \  
  --log-group-name MyApp/access.log --log-stream-name TestStream1 \  
  --log-events \  
    timestamp=1394793518000,message="This message contains an Error" \  
    timestamp=1394793528000,message="This message also contains an Error"
```

Example: Count HTTP 404 codes

Using CloudWatch Logs, you can monitor how many times your Apache servers return a HTTP 404 response, which is the response code for page not found. You might want to monitor this to understand how often your site visitors do not find the resource they are looking for. Assume that your log records are structured to include the following information for each log event (site visit):

- Requestor IP Address
- RFC 1413 Identity
- Username
- Timestamp
- Request method with requested resource and protocol
- HTTP response code to request
- Bytes transferred in request

An example of this might look like the following:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 404 2326
```

You could specify a rule which attempts to match events of that structure for HTTP 404 errors, as shown in the following example:

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose **Actions**, **Create metric filter**.
4. For **Filter Pattern**, type **[IP, UserInfo, User, Timestamp, RequestInfo, StatusCode=404, Bytes]**.
5. (Optional) To test your filter pattern, under **Test Pattern**, enter one or more log events to use to test the pattern. Each log event must be within one line, because line breaks are used to separate log events in the **Log event messages** box.
6. Choose **Next**, and then for **Filter Name**, type **HTTP404Errors**.
7. Under **Metric Details**, for **Metric Namespace**, enter **MyNameSpace**.
8. For **Metric Name**, enter **ApacheNotFoundErrorCode**.
9. Confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every 404 Error event.
10. For **Default Value** enter 0, and then choose **Next**.
11. Choose **Create metric filter**.

To create a metric filter using the AWS CLI

At a command prompt, run the following command:

```
aws logs put-metric-filter \
  --log-group-name MyApp/access.log \
  --filter-name HTTP404Errors \
  --filter-pattern '[ip, id, user, timestamp, request, status_code=404, size]' \
  --metric-transformations \
    metricName=ApacheNotFoundErrorCode,metricNamespace=MyNameSpace,metricValue=1
```

In this example, literal characters such as the left and right square brackets, double quotes and character string 404 were used. The pattern needs to match with the entire log event message for the log event to be considered for monitoring.

You can verify the creation of the metric filter by using the **describe-metric-filters** command. You should see output that looks like this:

```
aws logs describe-metric-filters --log-group-name MyApp/access.log

{
  "metricFilters": [
    {
      "filterName": "HTTP404Errors",
      "metricTransformations": [
        {
          "metricValue": "1",
          "metricNamespace": "MyNamespace",
          "metricName": "ApacheNotFoundErrorCode"
        }
      ],
      "creationTime": 1399277571078,
      "filterPattern": "[ip, id, user, timestamp, request, status_code=404,
size]"
    }
  ]
}
```

Now you can post a few events manually:

```
aws logs put-log-events \
--log-group-name MyApp/access.log --log-stream-name hostname \
--log-events \
timestamp=1394793518000,message="127.0.0.1 - bob [10/Oct/2000:13:55:36 -0700] \"GET /
apache_pb.gif HTTP/1.0\" 404 2326" \
timestamp=1394793528000,message="127.0.0.1 - bob [10/Oct/2000:13:55:36 -0700] \"GET /
apache_pb2.gif HTTP/1.0\" 200 2326"
```

Soon after putting these sample log events, you can retrieve the metric named in the CloudWatch console as `ApacheNotFoundErrorCode`.

Example: Count HTTP 4xx codes

As in the previous example, you might want to monitor your web service access logs and monitor the HTTP response code levels. For example, you might want to monitor all of the HTTP 400-level errors. However, you might not want to specify a new metric filter for every return code.

The following example demonstrates how to create a metric that includes all 400-level HTTP code responses from an access log using the Apache access log format from the [Example: Count HTTP 404 codes](#) example.

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the log group for the Apache server.
4. Choose Actions, **Create metric filter**.
5. For **Filter pattern**, enter `[ip, id, user, timestamp, request, status_code=4*, size]`.
6. (Optional) To test your filter pattern, under **Test Pattern**, enter one or more log events to use to test the pattern. Each log event must be within one line, because line breaks are used to separate log events in the **Log event messages** box.
7. Choose **Next**, and then for **Filter name**, type **HTTP4xxErrors**.
8. Under **Metric details**, for **Metric namespace**, enter **MyNameSpace**.
9. For **Metric name**, enter **HTTP4xxErrors**.
10. For **Metric value**, enter 1. This specifies that the count is incremented by 1 for every log containing a 4xx error.
11. For **Default value** enter 0, and then choose **Next**.
12. Choose **Create metric filter**.

To create a metric filter using the AWS CLI

At a command prompt, run the following command:

```
aws logs put-metric-filter \  
  --log-group-name MyApp/access.log \  
  --filter-name HTTP4xxErrors \  
  --filter-pattern '[ip, id, user, timestamp, request, status_code=4*, size]' \  
  --metric-transformations \  
    metricName=HTTP4xxErrors,metricNamespace=MyNameSpace,metricValue=1,defaultValue=0
```

You can use the following data in put-event calls to test this rule. If you did not remove the monitoring rule in the previous example, you will generate two different metrics.

```
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /~test/ HTTP/1.1" 200 3
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308
127.0.0.1 - - [24/Sep/2013:11:51:34 -0700] "GET /~test/index.html HTTP/1.1" 200 3
```

Example: Extract fields from an Apache log and assign dimensions

Sometimes, instead of counting, it is helpful to use values within individual log events for metric values. This example shows how you can create an extraction rule to create a metric that measures the bytes transferred by an Apache webserver.

This example also shows how to assign dimensions to the metric that you are creating.

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the log group for the Apache server.
4. Choose **Actions**, **Create metric filter**.
5. For **Filter pattern**, enter `[ip, id, user, timestamp, request, status_code, size]`.
6. (Optional) To test your filter pattern, under **Test Pattern**, enter one or more log events to use to test the pattern. Each log event must be within one line, because line breaks are used to separate log events in the **Log event messages** box.
7. Choose **Next**, and then for **Filter name**, type `size`.
8. Under **Metric details**, for **Metric namespace**, enter `MyNameSpace`. Because this is a new namespace, be sure that **Create new** is selected.
9. For **Metric name**, enter `BytesTransferred`.
10. For **Metric value**, enter `$size`.
11. For **Unit**, select **Bytes**.
12. For **Dimension Name**, type `IP`.
13. For **Dimension Value**, type `$ip` and then choose **Next**.
14. Choose **Create metric filter**.

To create this metric filter using the AWS CLI

At a command prompt, run the following command

```
aws logs put-metric-filter \
--log-group-name MyApp/access.log \
--filter-name BytesTransferred \
--filter-pattern '[ip, id, user, timestamp, request, status_code, size]' \
--metric-transformations \
metricName=BytesTransferred,metricNamespace=MyNamespace,metricValue='$size'
```

```
aws logs put-metric-filter \
--log-group-name MyApp/access.log \
--filter-name BytesTransferred \
--filter-pattern '[ip, id, user, timestamp, request, status_code, size]' \
--metric-transformations \
metricName=BytesTransferred,metricNamespace=MyNamespace,metricValue='$size',unit=Bytes,dimension1=$ip}}'
```

Note

In this command, use this format to specify multiple dimensions.

```
aws logs put-metric-filter \
--log-group-name my-log-group-name \
--filter-name my-filter-name \
--filter-pattern 'my-filter-pattern' \
--metric-transformations \
metricName=my-metric-name,metricNamespace=my-metric-namespace,metricValue=my-token,unit=unit,dimensions='{dimension1=$dim,dimension2=$dim2,dim3=$dim3}'
```

You can use the following data in put-log-event calls to test this rule. This generates two different metrics if you did not remove monitoring rule in the previous example.

```
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /~test/ HTTP/1.1" 200 3
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308
```

```
127.0.0.1 - - [24/Sep/2013:11:51:34 -0700] "GET /~test/index.html HTTP/1.1" 200 3
```

Listing metric filters

You can list all metric filters in a log group.

To list metric filters using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. In the contents pane, in the list of log groups, in the **Metric Filters** column, choose the number of filters.

The **Log Groups > Filters for** screen lists all metric filters associated with the log group.

To list metric filters using the AWS CLI

At a command prompt, run the following command:

```
aws logs describe-metric-filters --log-group-name MyApp/access.log
```

The following is example output:

```
{
  "metricFilters": [
    {
      "filterName": "HTTP404Errors",
      "metricTransformations": [
        {
          "metricValue": "1",
          "metricNamespace": "MyNamespace",
          "metricName": "ApacheNotFoundErrorCount"
        }
      ],
      "creationTime": 1399277571078,
      "filterPattern": "[ip, id, user, timestamp, request, status_code=404,
size]"
    }
  ]
}
```

Deleting a metric filter

A policy is identified by its name and the log group it belongs to.

To delete a metric filter using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. In the contents pane, in the **Metric Filter** column, choose the number of metric filters for the log group.
4. Under **Metric Filters** screen, select the check box to the right of the name of the filter that you want to delete. Then choose **Delete**.
5. When prompted for confirmation, choose **Delete**.

To delete a metric filter using the AWS CLI

At a command prompt, run the following command:

```
aws logs delete-metric-filter --log-group-name MyApp/access.log \  
--filter-name MyFilterName
```

Real-time processing of log data with subscriptions

You can use subscriptions to get access to a real-time feed of log events from CloudWatch Logs and have it delivered to other services such as an Amazon Kinesis stream, an Amazon Data Firehose stream, or AWS Lambda for custom processing, analysis, or loading to other systems. When log events are sent to the receiving service, they are base64 encoded and compressed with the gzip format.

You can also use CloudWatch Logs centralization to replicate log data from multiple accounts and regions into a central location. For more information, see [Cross-account cross-Region log centralization](#).

To begin subscribing to log events, create the receiving resource, such as a Amazon Kinesis Data Streams stream, where the events will be delivered. A subscription filter defines the filter pattern to use for filtering which log events get delivered to your AWS resource, as well as information about where to send matching log events to. Log events are sent to the receiving resource soon after being ingested, usually less than three minutes.

Note

If a log group with a subscription uses log transformation, the filter pattern is compared to the transformed versions of the log events. For more information, see [Transform logs during ingestion](#).

You can create subscriptions at the account level and at the log group level. Each account can have one account-level subscription filter per Region. Each log group can have up to two subscription filters associated with it.

Note

If the destination service returns a retryable error such as a throttling exception or a retryable service exception (HTTP 5xx for example), CloudWatch Logs continues to retry delivery for up to 24 hours. CloudWatch Logs doesn't try to re-deliver if the error is a non-retryable error, such as `AccessDeniedException` or `ResourceNotFoundException`. In these cases the subscription filter is disabled for up to 10 minutes, and then CloudWatch Logs retries sending logs to the destination. During this disabled period, logs are skipped.

CloudWatch Logs also produces CloudWatch metrics about the forwarding of log events to subscriptions. For more information, see [Monitoring with CloudWatch metrics](#).

You can also use a CloudWatch Logs subscription to stream log data in near real time to an Amazon OpenSearch Service cluster. For more information, see [Streaming CloudWatch Logs data to Amazon OpenSearch Service](#).

Subscriptions are supported only for log groups in the Standard log class. For more information about log classes, see [Log classes](#).

 Note

Subscription filters might batch log events to optimize transmission and reduce the amount of calls made to the destination. Batching is not guaranteed but is used when possible.

For batch processing and analysis of log data on a schedule, consider using [Automating log analysis with scheduled queries](#). Scheduled queries run CloudWatch Logs Insights queries automatically and deliver results to destinations such as Amazon S3 buckets or Amazon EventBridge event buses.

Contents

- [Concepts](#)
- [Log group-level subscription filters](#)
- [Account-level subscription filters](#)
- [Cross-account cross-Region subscriptions](#)
- [Confused deputy prevention](#)
- [Log recursion prevention](#)

Concepts

Each subscription filter is made up of the following key elements:

filter pattern

A symbolic description of how CloudWatch Logs should interpret the data in each log event, along with filtering expressions that restrict what gets delivered to the destination AWS

resource. For more information about the filter pattern syntax, see [Filter pattern syntax for metric filters, subscription filters, filter log events, and Live Tail](#).

destination arn

The Amazon Resource Name (ARN) of the Amazon Kinesis Data Streams stream, Firehose stream, or Lambda function you want to use as the destination of the subscription feed.

role arn

An IAM role that grants CloudWatch Logs the necessary permissions to put data into the chosen destination. This role is not needed for Lambda destinations because CloudWatch Logs can get the necessary permissions from access control settings on the Lambda function itself.

distribution

The method used to distribute log data to the destination, when the destination is a stream in Amazon Kinesis Data Streams. By default, log data is grouped by log stream. For a more even distribution, you can group log data randomly.

For log group-level subscriptions, the following key element is also included:

log group name

The log group to associate the subscription filter with. All log events uploaded to this log group would be subject to the subscription filter, and those that match the filter are delivered to the destination service that is receiving the matching log events.

For account-level subscriptions, the following key element is also included:

selection criteria

The criteria used for selecting which log groups have the account-level subscription filter applied. If you don't specify this, the account-level subscription filter is applied to all log groups in the account. This field is used to prevent infinite log loops. For more information about the infinite log loop issue, see [Log recursion prevention](#).

Selection criteria has a size limit of 25 KB.

For centralized log groups, the following key elements are also included. These elements can be used as field selection criteria to help in identifying the source of log data, allowing for more granular filtering and analysis of metrics derived from centralized logs.

@aws.account

This field identifies the AWS account ID from which the log event originated.

@aws.region

This field identifies the AWS Region where the log event was generated.

Log group-level subscription filters

You can use a subscription filter with Amazon Kinesis Data Streams, AWS Lambda, Amazon Data Firehose, or Amazon OpenSearch Service. Logs sent to a service through a subscription filter are base64 encoded and compressed with the gzip format. If you are using centralized logs with your AWS Organizations, you can choose to emit the `@aws.account` and `@aws.region` system field to identify which data comes from which accounts and regions in your organization. This section provides examples you can follow to create a CloudWatch Logs subscription filter that sends log data to Firehose, Lambda, Amazon Kinesis Data Streams, and OpenSearch Service.

Note

If you want to search your log data, see [Filter and pattern syntax](#).

Examples

- [Example 1: Subscription filters with Amazon Kinesis Data Streams](#)
- [Example 2: Subscription filters with AWS Lambda](#)
- [Example 3: Subscription filters with Amazon Data Firehose](#)
- [Example 4: Subscription filters with Amazon OpenSearch Service](#)

Example 1: Subscription filters with Amazon Kinesis Data Streams

The following example associates a subscription filter with a log group containing AWS CloudTrail events. The subscription filter delivers every logged activity made by "Root" AWS credentials to a

stream in Amazon Kinesis Data Streams called "RootAccess." For more information about how to send AWS CloudTrail events to CloudWatch Logs, see [Sending CloudTrail Events to CloudWatch Logs](#) in the *AWS CloudTrail User Guide*.

Note

Before you create the stream, calculate the volume of log data that will be generated. Be sure to create a stream with enough shards to handle this volume. If the stream does not have enough shards, the log stream will be throttled. For more information about stream volume limits, see [Quotas and Limits](#).

Throttled deliverables are retried for up to 24 hours. After 24 hours, the failed deliverables are dropped.

To mitigate the risk of throttling, you can take the following steps:

- Specify random for distribution when you create the subscription filter with [PutSubscriptionFilter](#) or [put-subscription-filter](#). By default, the stream filter distribution is by log stream and this can cause throttling.
- Monitor your stream using CloudWatch metrics. This helps you identify any throttling and adjust your configuration accordingly. For example, the `DeliveryThrottling` metric can be used to track the number of log events for which CloudWatch Logs was throttled when forwarding data to the subscription destination. For more information about monitoring, see [Monitoring with CloudWatch metrics](#).
- Use the on-demand capacity mode for your stream in Amazon Kinesis Data Streams. On-demand mode instantly accommodates your workloads as they ramp up or down. More information about on-demand capacity mode, see [On-demand mode](#).
- Restrict your CloudWatch subscription filter pattern to match the capacity of your stream in Amazon Kinesis Data Streams. If you are sending too much data to the stream, you might need to reduce the filter size or adjust the filter criteria.

To create a subscription filter for Amazon Kinesis Data Streams

1. Create a destination stream using the following command:

```
$ C:\> aws kinesis create-stream --stream-name "RootAccess" --shard-count 1
```

2. Wait until the stream becomes Active (this might take a minute or two). You can use the following Amazon Kinesis Data Streams [describe-stream](#) command

to check the **StreamDescription.StreamStatus** property. In addition, note the **StreamDescription.StreamARN** value, as you will need it in a later step:

```
aws kinesis describe-stream --stream-name "RootAccess"
```

The following is example output:

```
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "RootAccess",
    "StreamARN": "arn:aws:kinesis:us-east-1:123456789012:stream/RootAccess",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "340282366920938463463374607431768211455",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
            "49551135218688818456679503831981458784591352702181572610"
        }
      }
    ]
  }
}
```

3. Create the IAM role that will grant CloudWatch Logs permission to put data into your stream. First, you'll need to create a trust policy in a file (for example, `~/TrustPolicyForCWL-Kinesis.json`). Use a text editor to create this policy. Do not use the IAM console to create it.

This policy includes a `aws:SourceArn` global condition context key to help prevent the confused deputy security problem. For more information, see [Confused deputy prevention](#).

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": { "aws:SourceArn": "arn:aws:logs:region:123456789012:*" }
    }
  }
}
```

```

    }
  }
}

```

4. Use the **create-role** command to create the IAM role, specifying the trust policy file. Note the returned **Role.Arn** value, as you will also need it for a later step:

```
aws iam create-role --role-name CWLtoKinesisRole --assume-role-policy-document
file://~/TrustPolicyForCWL-Kinesis.json
```

The following is an example of the output.

```

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Effect": "Allow",
          "Principal": {
            "Service": "logs.amazonaws.com"
          },
          "Condition": {
            "StringLike": {
              "aws:SourceArn": [ "arn:aws:logs:region:123456789012:*" ]
            }
          }
        }
      ]
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
  }
}

```

5. Create a permissions policy to define what actions CloudWatch Logs can do on your account. First, you'll create a permissions policy in a file (for example, ~/PermissionsForCWL-Kinesis.json). Use a text editor to create this policy. Do not use the IAM console to create it.

```

{
  "Statement": [

```

```
{
  "Effect": "Allow",
  "Action": "kinesis:PutRecord",
  "Resource": "arn:aws:kinesis:region:123456789012:stream/RootAccess"
}
```

6. Associate the permissions policy with the role using the following [put-role-policy](#) command:

```
aws iam put-role-policy --role-name CWLtoKinesisRole --policy-name Permissions-
Policy-For-CWL --policy-document file://~/PermissionsForCWL-Kinesis.json
```

7. After the stream is in **Active** state and you have created the IAM role, you can create the CloudWatch Logs subscription filter. The subscription filter immediately starts the flow of real-time log data from the chosen log group to your stream:

```
aws logs put-subscription-filter \
  --log-group-name "CloudTrail/logs" \
  --filter-name "RootAccess" \
  --filter-pattern "{$.userIdentity.type = Root}" \
  --destination-arn "arn:aws:kinesis:region:123456789012:stream/RootAccess" \
  --role-arn "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
```

8. After you set up the subscription filter, CloudWatch Logs forwards all the incoming log events that match the filter pattern to your stream. You can verify that this is happening by grabbing a Amazon Kinesis Data Streams shard iterator and using the Amazon Kinesis Data Streams `get-records` command to fetch some Amazon Kinesis Data Streams records:

```
aws kinesis get-shard-iterator --stream-name RootAccess --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON
```

```
{
  "ShardIterator":
  "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRgB9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
}
```

```
aws kinesis get-records --limit 10 --shard-iterator "AAAAAAAAAAGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRgB9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWiK20Sh0uP"
```

Note that you might need to make this call a few times before Amazon Kinesis Data Streams starts to return data.

You should expect to see a response with an array of records. The **Data** attribute in a Amazon Kinesis Data Streams record is base64 encoded and compressed with the gzip format. You can examine the raw data from the command line using the following Unix commands:

```
echo -n "<Content of Data>" | base64 -d | zcat
```

The base64 decoded and decompressed data is formatted as JSON with the following structure:

```
{
  "owner": "111111111111",
  "logGroup": "CloudTrail/logs",
  "logStream": "111111111111_CloudTrail/logs_us-east-1",
  "subscriptionFilters": [
    "Destination"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "31953106606966983378809025079804211143289615424298221568",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221569",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}",
    },
    {
```

```
{
  "id": "31953106606966983378809025079804211143289615424298221570",
  "timestamp": 1432826855000,
  "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}\"
}
```

The key elements in the above data structure are the following:

owner

The AWS Account ID of the originating log data.

logGroup

The log group name of the originating log data.

logStream

The log stream name of the originating log data.

subscriptionFilters

The list of subscription filter names that matched with the originating log data.

messageType

Data messages will use the "DATA_MESSAGE" type. Sometimes CloudWatch Logs may emit Amazon Kinesis Data Streams records with a "CONTROL_MESSAGE" type, mainly for checking if the destination is reachable.

logEvents

The actual log data, represented as an array of log event records. The "id" property is a unique identifier for every log event.

Example 2: Subscription filters with AWS Lambda

In this example, you'll create a CloudWatch Logs subscription filter that sends log data to your AWS Lambda function.

Note

Before you create the Lambda function, calculate the volume of log data that will be generated. Be sure to create a function that can handle this volume. If the function does not have enough volume, the log stream will be throttled. For more information about Lambda limits, see [AWS Lambda Limits](#).

To create a subscription filter for Lambda

1. Create the AWS Lambda function.

Ensure that you have set up the Lambda execution role. For more information, see [Step 2.2: Create an IAM Role \(execution role\)](#) in the *AWS Lambda Developer Guide*.

2. Open a text editor and create a file named `helloWorld.js` with the following contents:

```
var zlib = require('zlib');
exports.handler = function(input, context) {
  var payload = Buffer.from(input.awslogs.data, 'base64');
  zlib.gunzip(payload, function(e, result) {
    if (e) {
      context.fail(e);
    } else {
      result = JSON.parse(result.toString());
      console.log("Event Data:", JSON.stringify(result, null, 2));
      context.succeed();
    }
  });
};
```

3. Zip the file `helloWorld.js` and save it with the name `helloWorld.zip`.
4. Use the following command, where the role is the Lambda execution role you set up in the first step:

```
aws lambda create-function \
  --function-name helloworld \
  --zip-file fileb://file-path/helloWorld.zip \
  --role lambda-execution-role-arn \
  --handler helloworld.handler \
  --runtime nodejs12.x
```

5. Grant CloudWatch Logs the permission to execute your function. Use the following command, replacing the placeholder account with your own account and the placeholder log group with the log group to process:

```
aws lambda add-permission \  
  --function-name "helloworld" \  
  --statement-id "helloworld" \  
  --principal "logs.amazonaws.com" \  
  --action "lambda:InvokeFunction" \  
  --source-arn "arn:aws:logs:region:123456789123:log-group:TestLambda:*" \  
  --source-account "123456789012"
```

6. Create a subscription filter using the following command, replacing the placeholder account with your own account and the placeholder log group with the log group to process:

```
aws logs put-subscription-filter \  
  --log-group-name myLogGroup \  
  --filter-name demo \  
  --filter-pattern "" \  
  --destination-arn arn:aws:lambda:region:123456789123:function:helloworld
```

7. (Optional) Test using a sample log event. At a command prompt, run the following command, which will put a simple log message into the subscribed stream.

To see the output of your Lambda function, navigate to the Lambda function where you will see the output in `/aws/lambda/helloworld`:

```
aws logs put-log-events --log-group-name myLogGroup --log-stream-name stream1 --  
log-events "[{\"timestamp\":<CURRENT TIMESTAMP MILLIS> , \"message\": \"Simple  
Lambda Test\"}]"
```

You should expect to see a response with an array of Lambda. The **Data** attribute in the Lambda record is base64 encoded and compressed with the gzip format. The actual payload that Lambda receives is in the following format `{ "awslogs": { "data": "BASE64ENCODED_GZIP_COMPRESSED_DATA" } }` You can examine the raw data from the command line using the following Unix commands:

```
echo -n "<BASE64ENCODED_GZIP_COMPRESSED_DATA>" | base64 -d | zcat
```

The base64 decoded and decompressed data is formatted as JSON with the following structure:

```
{
  "owner": "123456789012",
  "logGroup": "CloudTrail",
  "logStream": "123456789012_CloudTrail_us-east-1",
  "subscriptionFilters": [
    "Destination"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "31953106606966983378809025079804211143289615424298221568",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221569",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221570",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    }
  ]
}
```

The key elements in the above data structure are the following:

owner

The AWS Account ID of the originating log data.

logGroup

The log group name of the originating log data.

logStream

The log stream name of the originating log data.

subscriptionFilters

The list of subscription filter names that matched with the originating log data.

messageType

Data messages will use the "DATA_MESSAGE" type. Sometimes CloudWatch Logs may emit Lambda records with a "CONTROL_MESSAGE" type, mainly for checking if the destination is reachable.

logEvents

The actual log data, represented as an array of log event records. The "id" property is a unique identifier for every log event.

Example 3: Subscription filters with Amazon Data Firehose

In this example, you'll create a CloudWatch Logs subscription that sends any incoming log events that match your defined filters to your Amazon Data Firehose delivery stream. Data sent from CloudWatch Logs to Amazon Data Firehose is already compressed with gzip level 6 compression, so you do not need to use compression within your Firehose delivery stream. You can then use the decompression feature in Firehose to automatically decompress the logs. For more information, see [Send CloudWatch Logs to Firehose](#).

Note

Before you create the Firehose stream, calculate the volume of log data that will be generated. Be sure to create a Firehose stream that can handle this volume. If the stream cannot handle the volume, the log stream will be throttled. For more information about Firehose stream volume limits, see [Amazon Data Firehose Data Limits](#).

To create a subscription filter for Firehose

1. Create an Amazon Simple Storage Service (Amazon S3) bucket. We recommend that you use a bucket that was created specifically for CloudWatch Logs. However, if you want to use an existing bucket, skip to step 2.

Run the following command, replacing the placeholder Region with the Region you want to use:

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration
LocationConstraint=region
```

The following is example output:

```
{
  "Location": "/amzn-s3-demo-bucket2"
}
```

2. Create the IAM role that grants Amazon Data Firehose permission to put data into your Amazon S3 bucket.

For more information, see [Controlling Access with Amazon Data Firehose](#) in the *Amazon Data Firehose Developer Guide*.

First, use a text editor to create a trust policy in a file `~/TrustPolicyForFirehose.json` as follows:

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "firehose.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

3. Use the **create-role** command to create the IAM role, specifying the trust policy file. Note of the returned **Role.Arn** value, as you will need it in a later step:

```
aws iam create-role \
  --role-name FirehoseToS3Role \
  --assume-role-policy-document file:///~/TrustPolicyForFirehose.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
```

```

        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
            "Service": "firehose.amazonaws.com"
        }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "FirehoseToS3Role",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/FirehoseToS3Role"
}

```

4. Create a permissions policy to define what actions Firehose can do on your account. First, use a text editor to create a permissions policy in a file `~/PermissionsForFirehose.json`:

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:PutObject" ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket2",
        "arn:aws:s3:::amzn-s3-demo-bucket2/*" ]
    }
  ]
}

```

5. Associate the permissions policy with the role using the following `put-role-policy` command:

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name Permissions-
Policy-For-Firehose --policy-document file://~/PermissionsForFirehose.json
```

6. Create a destination Firehose delivery stream as follows, replacing the placeholder values for **RoleARN** and **BucketARN** with the role and bucket ARNs that you created:

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::123456789012:role/FirehoseToS3Role", "BucketARN":
  "arn:aws:s3:::amzn-s3-demo-bucket2"}'
```

Note that Firehose automatically uses a prefix in YYYY/MM/DD/HH UTC time format for delivered Amazon S3 objects. You can specify an extra prefix to be added in front of the time format prefix. If the prefix ends with a forward slash (/), it appears as a folder in the Amazon S3 bucket.

7. Wait until the stream becomes active (this might take a few minutes). You can use the Firehose **describe-delivery-stream** command to check the **DeliveryStreamDescription.DeliveryStreamStatus** property. In addition, note the **DeliveryStreamDescription.DeliveryStreamARN** value, as you will need it in a later step:

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
{
  "DeliveryStreamDescription": {
    "HasMoreDestinations": false,
    "VersionId": "1",
    "CreateTimestamp": 1446075815.822,
    "DeliveryStreamARN": "arn:aws:firehose:us-east-1:123456789012:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamName": "my-delivery-stream",
    "Destinations": [
      {
        "DestinationId": "destinationId-000000000001",
        "S3DestinationDescription": {
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "RoleARN": "delivery-stream-role",
          "BucketARN": "arn:aws:s3::amzn-s3-demo-bucket2",
          "BufferingHints": {
            "IntervalInSeconds": 300,
            "SizeInMBs": 5
          }
        }
      }
    ]
  }
}
```

```

    }
  }
]
}
}

```

8. Create the IAM role that grants CloudWatch Logs permission to put data into your Firehose delivery stream. First, use a text editor to create a trust policy in a file `~/TrustPolicyForCWL.json`:

This policy includes a `aws:SourceArn` global condition context key to help prevent the confused deputy security problem. For more information, see [Confused deputy prevention](#).

```

{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
      }
    }
  }
}

```

9. Use the **create-role** command to create the IAM role, specifying the trust policy file. Note of the returned **Role.Arn** value, as you will need it in a later step:

```

aws iam create-role \
--role-name CWLtoKinesisFirehoseRole \
--assume-role-policy-document file:///~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {

```

```

        "StringLike": {
            "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
        }
    },
    "RoleId": "AA01IAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"
}

```

10. Create a permissions policy to define what actions CloudWatch Logs can do on your account. First, use a text editor to create a permissions policy file (for example, ~/PermissionsForCWL.json):

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:PutRecord"],
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-
name"]
      }
    ]
  }
}

```

11. Associate the permissions policy with the role using the put-role-policy command:

```

aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-
name Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json

```

12. After the Amazon Data Firehose delivery stream is in active state and you have created the IAM role, you can create the CloudWatch Logs subscription filter. The subscription filter immediately starts the flow of real-time log data from the chosen log group to your Amazon Data Firehose delivery stream:

```

aws logs put-subscription-filter \
  --log-group-name "CloudTrail" \

```

```
--filter-name "Destination" \
--filter-pattern "{$.userIdentity.type = Root}" \
--destination-arn "arn:aws:firehose:region:123456789012:deliverystream/my-
delivery-stream" \
--role-arn "arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"
```

13. After you set up the subscription filter, CloudWatch Logs will forward all the incoming log events that match the filter pattern to your Amazon Data Firehose delivery stream. Your data will start appearing in your Amazon S3 based on the time buffer interval set on your Amazon Data Firehose delivery stream. Once enough time has passed, you can verify your data by checking your Amazon S3 Bucket.

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket2' --prefix 'firehose/'
{
  "Contents": [
    {
      "LastModified": "2015-10-29T00:01:25.000Z",
      "ETag": "\"a14589f8897f4089d3264d9e2d1f1610\"",
      "StorageClass": "STANDARD",
      "Key": "firehose/2015/10/29/00/my-delivery-stream-2015-10-29-00-01-21-
a188030a-62d2-49e6-b7c2-b11f1a7ba250",
      "Owner": {
        "DisplayName": "cloudwatch-logs",
        "ID": "1ec9cf700ef6be062b19584e0b7d84ecc19237f87b5"
      },
      "Size": 593
    },
    {
      "LastModified": "2015-10-29T00:35:41.000Z",
      "ETag": "\"a7035b65872bb2161388ffb63dd1aec5\"",
      "StorageClass": "STANDARD",
      "Key": "firehose/2015/10/29/00/my-delivery-
stream-2015-10-29-00-35-40-7cc92023-7e66-49bc-9fd4-fc9819cc8ed3",
      "Owner": {
        "DisplayName": "cloudwatch-logs",
        "ID": "1ec9cf700ef6be062b19584e0b7d84ecc19237f87b6"
      },
      "Size": 5752
    }
  ]
}
```

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket2' --key 'firehose/2015/10/29/00/
my-delivery-stream-2015-10-29-00-01-21-a188030a-62d2-49e6-b7c2-b11f1a7ba250'
testfile.gz

{
  "AcceptRanges": "bytes",
  "ContentType": "application/octet-stream",
  "LastModified": "Thu, 29 Oct 2015 00:07:06 GMT",
  "ContentLength": 593,
  "Metadata": {}
}
```

The data in the Amazon S3 object is compressed with the gzip format. You can examine the raw data from the command line using the following Unix command:

```
zcat testfile.gz
```

Example 4: Subscription filters with Amazon OpenSearch Service

In this example, you'll create a CloudWatch Logs subscription that sends incoming log events that match your defined filters to your OpenSearch Service domain.

To create a subscription filter for OpenSearch Service

1. Create an OpenSearch Service domain. For more information, see [Creating OpenSearch Service domains](#)
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Log groups**.
4. Select the name of the log group.
5. Choose **Actions, Subscription filters, Create Amazon OpenSearch Service subscription filter**.
6. Choose whether you want to stream to a cluster in this account or another account.
 - If you chose **This account**, select the domain that you created in step 1.
 - If you chose **Another account**, enter ARN and endpoint of that domain.
7. If you chose another account, provide the domain ARN and endpoint.

8. For Amazon OpenSearch Service cluster choose the name of the cluster where the log group data will be delivered
9. Choose a log format.
10. For **Subscription filter pattern**, enter the terms or pattern to find in your log events. This ensures that you send only the data that you're interested in to your OpenSearch Service cluster. For more information, see [Filter pattern syntax for metric filters](#).
11. (Optional) For **Select log data to test**, select a log stream and then choose **Test pattern** to verify that your search filter is returning the results you expect.
12. Choose **Start streaming**.

Account-level subscription filters

Important

There is a risk of causing an infinite recursive loop with subscription filters that can lead to a large increase in ingestion billing if not addressed. To mitigate this risk, we recommend that you use selection criteria in your account-level subscription filters to exclude log groups that ingest log data from resources that are part of the subscription delivery workflow. For more information on this problem and determining which log groups to exclude, see [Log recursion prevention](#).

You can set an account-level subscription policy that includes a subset of log groups in the account. The account subscription policy can work with Amazon Kinesis Data Streams, AWS Lambda, or Amazon Data Firehose. Logs sent to a service through an account-level subscription policy are base64 encoded and compressed with the gzip format. This section provides examples you can follow to create an account-level subscription for Amazon Kinesis Data Streams, Lambda, and Firehose.

Note

To view a list of all subscription filter policies in your account, use the `describe-account-policies` command with a value of `SUBSCRIPTION_FILTER_POLICY` for the `--policy-type` parameter. For more information, see [describe-account-policies](#).

Examples

- [Example 1: Subscription filters with Amazon Kinesis Data Streams](#)
- [Example 2: Subscription filters with AWS Lambda](#)
- [Example 3: Subscription filters with Amazon Data Firehose](#)

Example 1: Subscription filters with Amazon Kinesis Data Streams

Before you create a Amazon Kinesis Data Streams data stream to use with an account-level subscription policy, calculate the volume of log data that will be generated. Be sure to create a stream with enough shards to handle this volume. If a stream doesn't have enough shards, it is throttled. For more information about stream volume limits, see [Quotas and Limits](#) in the Amazon Kinesis Data Streams documentation.

Warning

Because the log events of multiple log groups are forwarded to the destination, there is a risk of throttling. Throttled deliverables are retried for up to 24 hours. After 24 hours, the failed deliverables are dropped.

To mitigate the risk of throttling, you can take the following steps:

- Monitor your Amazon Kinesis Data Streams stream with CloudWatch metrics. This helps you identify throttling and adjust your configuration accordingly. For example, the `DeliveryThrottling` metric tracks the number of log events for which CloudWatch Logs was throttled when forwarding data to the subscription destination. For more information, see [Monitoring with CloudWatch metrics](#).
- Use the on-demand capacity mode for your stream in Amazon Kinesis Data Streams. On-demand mode instantly accommodates your workloads as they ramp up or down. For more information, see [On-demand mode](#).
- Restrict your CloudWatch Logs subscription filter pattern to match the capacity of your stream in Amazon Kinesis Data Streams. If you are sending too much data to the stream, you might need to reduce the filter size or adjust the filter criteria.

The following example uses an account-level subscription policy to forward all log events to a stream in Amazon Kinesis Data Streams. The filter pattern matches any log events with the text `Test` and forwards them to the stream in Amazon Kinesis Data Streams.

To create an account-level subscription policy for Amazon Kinesis Data Streams

1. Create a destination stream using the following command:

```
$ C:\> aws kinesis create-stream --stream-name "TestStream" --shard-count 1
```

2. Wait a few minutes for the stream to become active. You can verify whether the stream is active by using the [describe-stream](#) command to check the **StreamDescription.StreamStatus** property.

```
aws kinesis describe-stream --stream-name "TestStream"
```

The following is example output:

```
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "TestStream",
    "StreamARN": "arn:aws:kinesis:region:123456789012:stream/TestStream",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "EXAMPLE8463463374607431768211455",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
            "EXAMPLE688818456679503831981458784591352702181572610"
        }
      }
    ]
  }
}
```

3. Create the IAM role that will grant CloudWatch Logs permission to put data into your stream. First, you'll need to create a trust policy in a file (for example, `~/TrustPolicyForCWL-Kinesis.json`). Use a text editor to create this policy.

This policy includes a `aws:SourceArn` global condition context key to help prevent the confused deputy security problem. For more information, see [Confused deputy prevention](#).

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": { "aws:SourceArn": "arn:aws:logs:region:123456789012:*" }
    }
  }
}
```

4. Use the **create-role** command to create the IAM role, specifying the trust policy file. Note the returned **Role.Arn** value, as you will also need it for a later step:

```
aws iam create-role --role-name CWLtoKinesisRole --assume-role-policy-document
file:///~/TrustPolicyForCWL-Kinesis.json
```

The following is an example of the output.

```
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
          "StringLike": {
            "aws:SourceArn": { "arn:aws:logs:region:123456789012:*" }
          }
        }
      }
    },
    "RoleId": "EXAMPLE450GAB4HC5F431",
    "CreateDate": "2023-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
  }
}
```

```
}
```

5. Create a permissions policy to define what actions CloudWatch Logs can do on your account. First, you'll create a permissions policy in a file (for example, `~/PermissionsForCWL-Kinesis.json`). Use a text editor to create this policy. Don't use the IAM console to create it.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:region:123456789012:stream/TestStream"
    }
  ]
}
```

6. Associate the permissions policy with the role using the following [put-role-policy](#) command:

```
aws iam put-role-policy --role-name CWLtoKinesisRole --policy-name Permissions-
Policy-For-CWL --policy-document file://~/PermissionsForCWL-Kinesis.json
```

7. After the stream is in the **Active** state and you have created the IAM role, you can create the CloudWatch Logs subscription filter policy. The policy immediately starts the flow of real-time log data to your stream. In this example, all log events that contain the string `ERROR` are streamed, except those in the log groups named `LogGroupToExclude1` and `LogGroupToExclude2`.

```
aws logs put-account-policy \
  --policy-name "ExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document '{"RoleArn":"arn:aws:iam::123456789012:role/
CWLtoKinesisRole", "DestinationArn":"arn:aws:kinesis:region:123456789012:stream/
TestStream", "FilterPattern": "Test", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

8. After you set up the subscription filter, CloudWatch Logs forwards all the incoming log events that match the filter pattern and selection criteria to your stream.

The `selection-criteria` field is optional, but is important for excluding log groups that can cause an infinite log recursion from a subscription filter. For more information about this

issue and determining which log groups to exclude, see [Log recursion prevention](#). Currently, NOT IN is the only supported operator for selection-criteria.

You can verify that the flow of log events by using a Amazon Kinesis Data Streams shard iterator and using the Amazon Kinesis Data Streams `get-records` command to fetch some Amazon Kinesis Data Streams records::

```
aws kinesis get-shard-iterator --stream-name TestStream --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON
```

```
{
  "ShardIterator":
  "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRgB9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
}
```

```
aws kinesis get-records --limit 10 --shard-iterator "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRgB9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
```

You might need to use this command a few times before Amazon Kinesis Data Streams starts to return data.

You should expect to see a response with an array of records. The **Data** attribute in a Amazon Kinesis Data Streams record is base64 encoded and compressed with the gzip format. You can examine the raw data from the command line using the following Unix commands:

```
echo -n "<Content of Data>" | base64 -d | zcat
```

The base64 decoded and decompressed data is formatted as JSON with the following structure:

```
{
  "messageType": "DATA_MESSAGE",
```

```

    "owner": "123456789012",
    "logGroup": "Example1",
    "logStream": "logStream1",
    "subscriptionFilters": [
        "ExamplePolicy"
    ],
    "logEvents": [
        {
            "id": "31953106606966983378809025079804211143289615424298221568",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}",
        },
        {
            "id": "31953106606966983378809025079804211143289615424298221569",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}",
        },
        {
            "id": "31953106606966983378809025079804211143289615424298221570",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}",
        }
    ],
    "policyLevel": "ACCOUNT_LEVEL_POLICY"
}

```

The key elements in the data structure are the following:

messageType

Data messages will use the "DATA_MESSAGE" type. Sometimes CloudWatch Logs might emit Amazon Kinesis Data Streams records with a "CONTROL_MESSAGE" type, mainly for checking if the destination is reachable.

owner

The AWS Account ID of the originating log data.

logGroup

The log group name of the originating log data.

logStream

The log stream name of the originating log data.

subscriptionFilters

The list of subscription filter names that matched with the originating log data.

logEvents

The actual log data, represented as an array of log event records. The "id" property is a unique identifier for every log event.

policyLevel

The level at which the policy was enforced. "ACCOUNT_LEVEL_POLICY" is the `policyLevel` for an account-level subscription filter policy.

Example 2: Subscription filters with AWS Lambda

In this example, you'll create a CloudWatch Logs account-level subscription filter policy that sends log data to your AWS Lambda function.

** Warning**

Before you create the Lambda function, calculate the volume of log data that will be generated. Be sure to create a function that can handle this volume. If the function can't handle the volume, the log stream will be throttled. Because the log events of either all log groups or a subset of the account's log groups are forwarded to the destination, there is a risk of throttling. For more information about Lambda limits, see [AWS Lambda Limits](#).

To create an account-level subscription filter policy for Lambda

1. Create the AWS Lambda function.

Ensure that you have set up the Lambda execution role. For more information, see [Step 2.2: Create an IAM Role \(execution role\)](#) in the *AWS Lambda Developer Guide*.

2. Open a text editor and create a file named `helloWorld.js` with the following contents:

```
var zlib = require('zlib');
```



```
exports.handler = function(input, context) {
  var payload = Buffer.from(input.awslogs.data, 'base64');
  zlib.gunzip(payload, function(e, result) {
    if (e) {
      context.fail(e);
    } else {
      result = JSON.parse(result.toString());
      console.log("Event Data:", JSON.stringify(result, null, 2));
      context.succeed();
    }
  });
};
```

3. Zip the file `helloWorld.js` and save it with the name `helloWorld.zip`.
4. Use the following command, where the role is the Lambda execution role you set up in the first step:

```
aws lambda create-function \
  --function-name helloworld \
  --zip-file fileb://file-path/helloWorld.zip \
  --role lambda-execution-role-arn \
  --handler helloworld.handler \
  --runtime nodejs18.x
```

5. Grant CloudWatch Logs the permission to execute your function. Use the following command, replacing the placeholder account with your own account.

```
aws lambda add-permission \
  --function-name "helloworld" \
  --statement-id "helloworld" \
  --principal "logs.amazonaws.com" \
  --action "lambda:InvokeFunction" \
  --source-arn "arn:aws:logs:region:123456789012:log-group:*" \
  --source-account "123456789012"
```

6. Create an account-level subscription filter policy using the following command, replacing the placeholder account with your own account. In this example, all log events that contain the string `ERROR` are streamed, except those in the log groups named `LogGroupToExclude1` and `LogGroupToExclude2`.

```
aws logs put-account-policy \
  --policy-name "ExamplePolicyLambda" \
```

```
--policy-type "SUBSCRIPTION_FILTER_POLICY" \
--policy-document
'{"DestinationArn":"arn:aws:lambda:region:123456789012:function:helloWorld",
"FilterPattern": "Test", "Distribution": "Random"}' \
--selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
--scope "ALL"
```

After you set up the subscription filter, CloudWatch Logs forwards all the incoming log events that match the filter pattern and selection criteria to your stream.

The `selection-criteria` field is optional, but is important for excluding log groups that can cause an infinite log recursion from a subscription filter. For more information about this issue and determining which log groups to exclude, see [Log recursion prevention](#). Currently, `NOT IN` is the only supported operator for `selection-criteria`.

7. (Optional) Test using a sample log event. At a command prompt, run the following command, which will put a simple log message into the subscribed stream.

To see the output of your Lambda function, navigate to the Lambda function where you will see the output in `/aws/lambda/helloworld`:

```
aws logs put-log-events --log-group-name Example1 --log-stream-name logStream1 --
log-events "[{\"timestamp\":CURRENT TIMESTAMP MILLIS , \"message\": \"Simple Lambda
Test\"}]"
```

You should expect to see a response with an array of Lambda. The **Data** attribute in the Lambda record is base64 encoded and compressed with the gzip format. The actual payload that Lambda receives is in the following format `{ "awslogs": { "data": "BASE64ENCODED_GZIP_COMPRESSED_DATA" } }` You can examine the raw data from the command line using the following Unix commands:

```
echo -n "<BASE64ENCODED_GZIP_COMPRESSED_DATA>" | base64 -d | zcat
```

The base64 decoded and decompressed data is formatted as JSON with the following structure:

```
{
  "messageType": "DATA_MESSAGE",
  "owner": "123456789012",
```

```

    "logGroup": "Example1",
    "logStream": "logStream1",
    "subscriptionFilters": [
        "ExamplePolicyLambda"
    ],
    "logEvents": [
        {
            "id": "31953106606966983378809025079804211143289615424298221568",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}
        },
        {
            "id": "31953106606966983378809025079804211143289615424298221569",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}
        },
        {
            "id": "31953106606966983378809025079804211143289615424298221570",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}
        }
    ],
    "policyLevel": "ACCOUNT_LEVEL_POLICY"
}

```

Note

The account-level subscription filter will not be applied to the destination Lambda function's log group. This is to prevent an infinite log recursion that can lead to an increase in ingestion billing. For more information about this problem, see [Log recursion prevention](#).

The key elements in the data structure are the following:

messageType

Data messages will use the "DATA_MESSAGE" type. Sometimes CloudWatch Logs might emit Amazon Kinesis Data Streams records with a "CONTROL_MESSAGE" type, mainly for checking if the destination is reachable.

owner

The AWS Account ID of the originating log data.

logGroup

The log group name of the originating log data.

logStream

The log stream name of the originating log data.

subscriptionFilters

The list of subscription filter names that matched with the originating log data.

logEvents

The actual log data, represented as an array of log event records. The "id" property is a unique identifier for every log event.

policyLevel

The level at which the policy was enforced. "ACCOUNT_LEVEL_POLICY" is the `policyLevel` for an account-level subscription filter policy.

Example 3: Subscription filters with Amazon Data Firehose

In this example, you'll create a CloudWatch Logs account-level subscription filter policy that sends incoming log events that match your defined filters to your Amazon Data Firehose delivery stream. Data sent from CloudWatch Logs to Amazon Data Firehose is already compressed with gzip level 6 compression, so you do not need to use compression within your Firehose delivery stream. You can then use the decompression feature in Firehose to automatically decompress the logs. For more information, see [Writing to Kinesis Data Firehose Using CloudWatch Logs](#).

⚠ Warning

Before you create the Firehose stream, calculate the volume of log data that will be generated. Be sure to create a Firehose stream that can handle this volume. If the stream cannot handle the volume, the log stream will be throttled. For more information about Firehose stream volume limits, see [Amazon Data Firehose Data Limits](#).

To create a subscription filter for Firehose

1. Create an Amazon Simple Storage Service (Amazon S3) bucket. We recommend that you use a bucket that was created specifically for CloudWatch Logs. However, if you want to use an existing bucket, skip to step 2.

Run the following command, replacing the placeholder Region with the Region you want to use:

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration
LocationConstraint=region
```

The following is example output:

```
{
  "Location": "/amzn-s3-demo-bucket2"
}
```

2. Create the IAM role that grants Amazon Data Firehose permission to put data into your Amazon S3 bucket.

For more information, see [Controlling Access with Amazon Data Firehose](#) in the *Amazon Data Firehose Developer Guide*.

First, use a text editor to create a trust policy in a file `~/TrustPolicyForFirehose.json` as follows:

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "firehose.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

```
}
}
```

3. Use the **create-role** command to create the IAM role, specifying the trust policy file. Keep a note of the returned **Role.Arn** value, as you will need it in a later step:

```
aws iam create-role \
  --role-name FirehoseToS3Role \
  --assume-role-policy-document file://~/TrustPolicyForFirehose.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        }
      }
    },
    "RoleId": "EXAMPLE50GAB4HC5F431",
    "CreateDate": "2023-05-29T13:46:29.431Z",
    "RoleName": "FirehoseToS3Role",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/FirehoseToS3Role"
  }
}
```

4. Create a permissions policy to define what actions Firehose can do on your account. First, use a text editor to create a permissions policy in a file `~/PermissionsForFirehose.json`:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:PutObject" ]
    }
  ]
}
```

```

    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket2",
      "arn:aws:s3:::amzn-s3-demo-bucket2/*" ]
    }
  ]
}

```

5. Associate the permissions policy with the role using the following put-role-policy command:

```

aws iam put-role-policy --role-name FirehoseToS3Role --policy-name Permissions-
Policy-For-Firehose --policy-document file://~/PermissionsForFirehose.json

```

6. Create a destination Firehose delivery stream as follows, replacing the placeholder values for **RoleARN** and **BucketARN** with the role and bucket ARNs that you created:

```

aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::123456789012:role/FirehoseToS3Role", "BucketARN":
  "arn:aws:s3:::amzn-s3-demo-bucket2"}'

```

Firehose automatically uses a prefix in YYYY/MM/DD/HH UTC time format for delivered Amazon S3 objects. You can specify an extra prefix to be added in front of the time format prefix. If the prefix ends with a forward slash (/), it appears as a folder in the Amazon S3 bucket.

7. Wait a few minutes for the stream becomes active. You can use the Firehose **describe-delivery-stream** command to check the **DeliveryStreamDescription.DeliveryStreamStatus** property. In addition, note the **DeliveryStreamDescription.DeliveryStreamARN** value, as you will need it in a later step:

```

aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
{
  "DeliveryStreamDescription": {
    "HasMoreDestinations": false,
    "VersionId": "1",
    "CreateTimestamp": 1446075815.822,
    "DeliveryStreamARN": "arn:aws:firehose:us-
east-1:123456789012:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamName": "my-delivery-stream",

```

```

    "Destinations": [
      {
        "DestinationId": "destinationId-0000000000001",
        "S3DestinationDescription": {
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "RoleARN": "delivery-stream-role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket2",
          "BufferingHints": {
            "IntervalInSeconds": 300,
            "SizeInMBs": 5
          }
        }
      }
    ]
  }
}

```

8. Create the IAM role that grants CloudWatch Logs permission to put data into your Firehose delivery stream. First, use a text editor to create a trust policy in a file `~/TrustPolicyForCWL.json`:

This policy includes a `aws:SourceArn` global condition context key to help prevent the confused deputy security problem. For more information, see [Confused deputy prevention](#).

```

{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
      }
    }
  }
}

```

9. Use the **create-role** command to create the IAM role, specifying the trust policy file. Make a note of the returned **Role.Arn** value, as you will need it in a later step:


```
aws iam create-role \
--role-name CWLtoKinesisFirehoseRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
          "StringLike": {
            "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
          }
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"
  }
}
```

10. Create a permissions policy to define what actions CloudWatch Logs can do on your account. First, use a text editor to create a permissions policy file (for example, `~/PermissionsForCWL.json`):

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:PutRecord"],
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-name"
      ]
    }
  ]
}
```

```
}
```

11. Associate the permissions policy with the role using the `put-role-policy` command:

```
aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-  
name Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json
```

12. After the Amazon Data Firehose delivery stream is in the active state and you have created the IAM role, you can create the CloudWatch Logs account-level subscription filter policy. The policy immediately starts the flow of real-time log data from the chosen log group to your Amazon Data Firehose delivery stream:

```
aws logs put-account-policy \  
  --policy-name "ExamplePolicyFirehose" \  
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \  
  --policy-document '{"RoleArn":"arn:aws:iam::123456789012:role/  
CWLtoKinesisFirehoseRole", "DestinationArn":"arn:aws:firehose:us-  
east-1:123456789012:deliverystream/delivery-stream-name", "FilterPattern": "Test",  
"Distribution": "Random"}' \  
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",  
"LogGroupToExclude2"]' \  
  --scope "ALL"
```

13. After you set up the subscription filter, CloudWatch Logs forwards the incoming log events that match the filter pattern to your Amazon Data Firehose delivery stream.

The `selection-criteria` field is optional, but is important for excluding log groups that can cause an infinite log recursion from a subscription filter. For more information about this issue and determining which log groups to exclude, see [Log recursion prevention](#). Currently, `NOT IN` is the only supported operator for `selection-criteria`.

Your data will start appearing in your Amazon S3 based on the time buffer interval set on your Amazon Data Firehose delivery stream. Once enough time has passed, you can verify your data by checking your Amazon S3 Bucket.

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket2' --prefix 'firehose/'  
{  
  "Contents": [  
    {  
      "LastModified": "2023-10-29T00:01:25.000Z",  
      "ETag": "\"a14589f8897f4089d3264d9e2d1f1610\"",
```

```

        "StorageClass": "STANDARD",
        "Key": "firehose/2015/10/29/00/my-delivery-stream-2015-10-29-00-01-21-
a188030a-62d2-49e6-b7c2-b11f1a7ba250",
        "Owner": {
            "DisplayName": "cloudwatch-logs",
            "ID": "1ec9cf700ef6be062b19584e0b7d84ecc19237f87b5"
        },
        "Size": 593
    },
    {
        "LastModified": "2015-10-29T00:35:41.000Z",
        "ETag": "\"a7035b65872bb2161388ffb63dd1aec5\"",
        "StorageClass": "STANDARD",
        "Key": "firehose/2023/10/29/00/my-delivery-stream-2023-10-29-00-35-40-
EXAMPLE-7e66-49bc-9fd4-fc9819cc8ed3",
        "Owner": {
            "DisplayName": "cloudwatch-logs",
            "ID": "EXAMPLE6be062b19584e0b7d84ecc19237f87b6"
        },
        "Size": 5752
    }
]
}

```

```

aws s3api get-object --bucket 'amzn-s3-demo-bucket2' --key 'firehose/2023/10/29/00/
my-delivery-stream-2023-10-29-00-01-21-a188030a-62d2-49e6-b7c2-b11f1a7ba250'
testfile.gz

```

```

{
    "AcceptRanges": "bytes",
    "ContentType": "application/octet-stream",
    "LastModified": "Thu, 29 Oct 2023 00:07:06 GMT",
    "ContentLength": 593,
    "Metadata": {}
}

```

The data in the Amazon S3 object is compressed with the gzip format. You can examine the raw data from the command line using the following Unix command:

```
zcat testfile.gz
```

Cross-account cross-Region subscriptions

You can collaborate with an owner of a different AWS account and receive their log events on your AWS resources, such as an Amazon Kinesis or Amazon Data Firehose stream (this is known as cross-account data sharing). For example, this log event data can be read from a centralized Amazon Kinesis Data Streams or Firehose stream to perform custom processing and analysis. Custom processing is especially useful when you collaborate and analyze data across many accounts.

For example, a company's information security group might want to analyze data for real-time intrusion detection or anomalous behaviors so it could conduct an audit of accounts in all divisions in the company by collecting their federated production logs for central processing. A real-time stream of event data across those accounts can be assembled and delivered to the information security groups, who can use Amazon Kinesis Data Streams to attach the data to their existing security analytic systems.

Note

The log group and the destination must be in the same AWS Region. However, the AWS resource that the destination points to can be located in a different Region. In the examples in the following sections, all Region-specific resources are created in US East (N. Virginia)).

If you have configured AWS Organizations and are working with member accounts you can use log centralization to collect log data from source accounts into a central monitoring account.

When working with centralized log groups you can use these system fields dimensions when creating subscription filters:

- `@aws.account` - This dimension represents the AWS account ID from which the log event originated.
- `@aws.region` - This dimension represents the AWS region where the log event was generated.

These dimensions help in identifying the source of log data, allowing for more granular filtering and analysis of metrics derived from centralized logs.

Topics

- [Cross-account cross-Region log data sharing using Amazon Kinesis Data Streams](#)
- [Cross-account cross-Region log data sharing using Firehose](#)

- [Cross-account cross-Region account-level subscriptions using Amazon Kinesis Data Streams](#)
- [Cross-account cross-Region account-level subscriptions using Firehose](#)

Cross-account cross-Region log data sharing using Amazon Kinesis Data Streams

When you create a cross-account subscription, you can specify a single account or an organization to be the sender. If you specify an organization, then this procedure enables all accounts in the organization to send logs to the receiver account.

To share log data across accounts, you need to establish a log data sender and receiver:

- **Log data sender**—gets the destination information from the recipient and lets CloudWatch Logs know that it's ready to send its log events to the specified destination. In the procedures in the rest of this section, the log data sender is shown with a fictional AWS account number of 111111111111.

If you're going to have multiple accounts in one organization send logs to one recipient account, you can create a policy that grants all accounts in the organization the permission to send logs to the recipient account. You still have to set up separate subscription filters for each sender account.

- **Log data recipient**—sets up a destination that encapsulates a Amazon Kinesis Data Streams stream and lets CloudWatch Logs know that the recipient wants to receive log data. The recipient then shares the information about this destination with the sender. In the procedures in the rest of this section, the log data recipient is shown with a fictional AWS account number of 999999999999.

To start receiving log events from cross-account users, the log data recipient first creates a CloudWatch Logs destination. Each destination consists of the following key elements:

Destination name

The name of the destination you want to create.

Target ARN

The Amazon Resource Name (ARN) of the AWS resource that you want to use as the destination of the subscription feed.

Role ARN

An AWS Identity and Access Management (IAM) role that grants CloudWatch Logs the necessary permissions to put data into the chosen stream.

Access policy

An IAM policy document (in JSON format, written using IAM policy grammar) that governs the set of users that are allowed to write to your destination.

Note

The log group and the destination must be in the same AWS Region. However, the AWS resource that the destination points to can be located in a different Region. In the examples in the following sections, all Region-specific resources are created in US East (N. Virginia).

Topics

- [Setting up a new cross-account subscription](#)
- [Updating an existing cross-account subscription](#)

Setting up a new cross-account subscription

Follow the steps in these sections to set up a new cross-account log subscription.

Topics

- [Step 1: Create a destination](#)
- [Step 2: \(Only if using an organization\) Create an IAM role](#)
- [Step 3: Add/validate IAM permissions for the cross-account destination](#)
- [Step 4: Create a subscription filter](#)
- [Validate the flow of log events](#)
- [Modify destination membership at runtime](#)

Step 1: Create a destination

Important

All steps in this procedure are to be done in the log data recipient account.

For this example, the log data recipient account has an AWS account ID of 999999999999, while the log data sender AWS account ID is 111111111111.

This example creates a destination using an Amazon Kinesis Data Streams stream called `RecipientStream`, and a role that enables CloudWatch Logs to write data to it.

When the destination is created, CloudWatch Logs sends a test message to the destination on the recipient account's behalf. When the subscription filter is active later, CloudWatch Logs sends log events to the destination on the source account's behalf.

To create a destination

1. In the recipient account, create a destination stream in Amazon Kinesis Data Streams. At a command prompt, type:

```
aws kinesis create-stream --stream-name "RecipientStream" --shard-count 1
```

2. Wait until the stream becomes active. You can use the **aws kinesis describe-stream** command to check the **StreamDescription.StreamStatus** property. In addition, take note of the **StreamDescription.StreamARN** value because you will pass it to CloudWatch Logs later:

```
aws kinesis describe-stream --stream-name "RecipientStream"
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "RecipientStream",
    "StreamARN": "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "34028236692093846346337460743176EXAMPLE",
          "StartingHashKey": "0"
        }
      }
    ]
  }
}
```

```

        "SequenceNumberRange": {
            "StartingSequenceNumber":
"4955113521868881845667950383198145878459135270218EXAMPLE"
        }
    }
]
}
}

```

It might take a minute or two for your stream to show up in the active state.

3. Create the IAM role that grants CloudWatch Logs the permission to put data into your stream. First, you'll need to create a trust policy in a file `~/TrustPolicyForCWL.json`. Use a text editor to create this policy file, do not use the IAM console.

This policy includes a `aws:SourceArn` global condition context key that specifies the `sourceAccountId` to help prevent the confused deputy security problem. If you don't yet know the source account ID in the first call, we recommend that you put the destination ARN in the source ARN field. In the subsequent calls, you should set the source ARN to be the actual source ARN that you gathered from the first call. For more information, see [Confused deputy prevention](#).

```

{
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.amazonaws.com"
    },
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:",
          "arn:aws:logs:region:recipientAccountId:"
        ]
      }
    },
    "Action": "sts:AssumeRole"
  }
}

```


4. Use the **aws iam create-role** command to create the IAM role, specifying the trust policy file. Take note of the returned Role.Arn value because it will also be passed to CloudWatch Logs later:

```
aws iam create-role \
--role-name CWLtoKinesisRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:*",
              "arn:aws:logs:region:recipientAccountId:"
            ]
          }
        },
        "Principal": {
          "Service": "logs.amazonaws.com"
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole"
  }
}
```

5. Create a permissions policy to define which actions CloudWatch Logs can perform on your account. First, use a text editor to create a permissions policy in a file **~/PermissionsForCWL.json**:

```
{
  "Statement": [
    {
```

```

    "Effect": "Allow",
    "Action": "kinesis:PutRecord",
    "Resource": "arn:aws:kinesis:region:999999999999:stream/RecipientStream"
  }
]
}

```

6. Associate the permissions policy with the role by using the **aws iam put-role-policy** command:

```

aws iam put-role-policy \
  --role-name CWLtoKinesisRole \
  --policy-name Permissions-Policy-For-CWL \
  --policy-document file://~/PermissionsForCWL.json

```

7. After the stream is in the active state and you have created the IAM role, you can create the CloudWatch Logs destination.
 - a. This step doesn't associate an access policy with your destination and is only the first step out of two that completes a destination creation. Make a note of the **DestinationArn** that is returned in the payload:

```

aws logs put-destination \
  --destination-name "testDestination" \
  --target-arn "arn:aws:kinesis:region:999999999999:stream/RecipientStream" \
  --role-arn "arn:aws:iam::999999999999:role/CWLtoKinesisRole"

{
  "DestinationName" : "testDestination",
  "RoleArn" : "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
  "DestinationArn" : "arn:aws:logs:us-
east-1:999999999999:destination:testDestination",
  "TargetArn" : "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream"
}

```

- b. After step 7a is complete, in the log data recipient account, associate an access policy with the destination. This policy must specify the **logs:PutSubscriptionFilter** action and grants permission to the sender account to access the destination.

The policy grants permission to the AWS account that sends logs. You can specify just this one account in the policy, or if the sender account is a member of an organization, the policy can specify the organization ID of the organization. This way, you can create just

one policy to allow multiple accounts in one organization to send logs to this destination account.

Use a text editor to create a file named `~/AccessPolicy.json` with one of the following policy statements.

This first example policy allows all accounts in the organization that have an ID of `o-1234567890` to send logs to the recipient account.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

This next example allows just the log data sender account (111111111111) to send logs to the log data recipient account.

JSON

```
{
  "Version": "2012-10-17",
```

```
"Statement": [  
  {  
    "Sid": "",  
    "Effect": "Allow",  
    "Principal": {  
      "AWS": "111111111111"  
    },  
    "Action": "logs:PutSubscriptionFilter",  
    "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination"  
  }  
]
```

- c. Attach the policy you created in the previous step to the destination.

```
aws logs put-destination-policy \  
  --destination-name "testDestination" \  
  --access-policy file://~/AccessPolicy.json
```

This access policy enables users in the AWS Account with ID 111111111111 to call **PutSubscriptionFilter** against the destination with ARN `arn:aws:logs:region:999999999999:destination:testDestination`. Any other user's attempt to call `PutSubscriptionFilter` against this destination will be rejected.

To validate a user's privileges against an access policy, see [Using Policy Validator](#) in the *IAM User Guide*.

When you have finished, if you're using AWS Organizations for your cross-account permissions, follow the steps in [Step 2: \(Only if using an organization\) Create an IAM role](#). If you're granting permissions directly to the other account instead of using Organizations, you can skip that step and proceed to [Step 4: Create a subscription filter](#).

Step 2: (Only if using an organization) Create an IAM role

In the previous section, if you created the destination by using an access policy that grants permissions to the organization that account 111111111111 is in, instead of granting permissions directly to account 111111111111, then follow the steps in this section. Otherwise, you can skip to [Step 4: Create a subscription filter](#).

The steps in this section create an IAM role, which CloudWatch can assume and validate whether the sender account has permission to create a subscription filter against the recipient destination.

Perform the steps in this section in the sender account. The role must exist in the sender account, and you specify the ARN of this role in the subscription filter. In this example, the sender account is 111111111111.

To create the IAM role necessary for cross-account log subscriptions using AWS Organizations

1. Create the following trust policy in a file /
TrustPolicyForCWLSubscriptionFilter.json. Use a text editor to create this policy file; do not use the IAM console.

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. Create the IAM role that uses this policy. Take note of the Arn value that is returned by the command, you will need it later in this procedure. In this example, we use CWLtoSubscriptionFilterRole for the name of the role we're creating.

```
aws iam create-role \
  --role-name CWLtoSubscriptionFilterRole \
  --assume-role-policy-document file:///~/
TrustPolicyForCWLSubscriptionFilter.json
```

3. Create a permissions policy to define the actions that CloudWatch Logs can perform on your account.
 - a. First, use a text editor to create the following permissions policy in a file named ~/PermissionsForCWLSubscriptionFilter.json.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
```

```
        "Resource": "arn:aws:logs:region:111111111111:log-  
group:LogGroupOnWhichSubscriptionFilterIsCreated:*"  
      }  
    ]  
  }  
}
```

- b. Enter the following command to associate the permissions policy you just created with the role that you created in step 2.

```
aws iam put-role-policy  
  --role-name CWLtoSubscriptionFilterRole  
  --policy-name Permissions-Policy-For-CWL-Subscription-filter  
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

When you have finished, you can proceed to [Step 4: Create a subscription filter](#).

Step 3: Add/validate IAM permissions for the cross-account destination

According to AWS cross-account policy evaluation logic, in order to access any cross-account resource (such as an Kinesis or Firehose stream used as a destination for a subscription filter) you must have an identity-based policy in the sending account which provides explicit access to the cross-account destination resource. For more information about policy evaluation logic, see [Cross-account policy evaluation logic](#).

You can attach the identity-based policy to the IAM role or IAM user that you are using to create the subscription filter. This policy must be present in the sending account. If you are using the Administrator role to create the subscription filter, you can skip this step and move on to [Step 4: Create a subscription filter](#).

To add or validate the IAM permissions needed for cross-account

1. Enter the following command to check which IAM role or IAM user is being used to run AWS logs commands.

```
aws sts get-caller-identity
```

The command returns output similar to the following:

```
{  
  "UserId": "User ID",
```

```
"Account": "sending account id",
"Arn": "arn:aws:sending account id:role/user:RoleName/UserName"
}
```

Make note of the value represented by *RoleName* or *UserName*.

2. Sign into the AWS Management Console in the sending account and search for the attached policies with the IAM role or IAM user returned in the output of the command you entered in step 1.
3. Verify that the policies attached to this role or user provide explicit permissions to call `logs:PutSubscriptionFilter` on the cross-account destination resource.

The following policy provides permissions to create a subscription filter on any destination resource only in a single AWS account, account 999999999999:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSubscriptionFiltersOnAccountResources",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs::*:log-group:*",
        "arn:aws:logs:*:123456789012:destination:*"
      ]
    }
  ]
}
```

The following policy provides permissions to create a subscription filter only on a specific destination resource named `sampleDestination` in single AWS account, account 123456789012:

JSON

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "AllowSubscriptionFilteronAccountResource",
    "Effect": "Allow",
    "Action": "logs:PutSubscriptionFilter",
    "Resource": [
      "arn:aws:logs:*:*:log-group:*",
      "arn:aws:logs:*:123456789012:destination:sampleDestination"
    ]
  }
]
```

Step 4: Create a subscription filter

After you create a destination, the log data recipient account can share the destination ARN (arn:aws:logs:us-east-1:999999999999:destination:testDestination) with other AWS accounts so that they can send log events to the same destination. These other sending accounts users then create a subscription filter on their respective log groups against this destination. The subscription filter immediately starts the flow of real-time log data from the chosen log group to the specified destination.

Note

If you are granting permissions for the subscription filter to an entire organization, you will need to use the ARN of the IAM role that you created in [Step 2: \(Only if using an organization\) Create an IAM role](#).

In the following example, a subscription filter is created in a sending account. the filter is associated with a log group containing AWS CloudTrail events so that every logged activity made by "Root" AWS credentials is delivered to the destination you previously created. That destination encapsulates a stream called "RecipientStream".

The rest of the steps in the following sections assume that you have followed the directions in [Sending CloudTrail Events to CloudWatch Logs](#) in the *AWS CloudTrail User Guide* and created a log group that contains your CloudTrail events. These steps assume that the name of this log group is CloudTrail/logs.

When you enter the following command, be sure you are signed in as the IAM user or using the IAM role that you added the policy for, in [Step 3: Add/validate IAM permissions for the cross-account destination](#).

```
aws logs put-subscription-filter \
  --log-group-name "CloudTrail/logs" \
  --filter-name "RecipientStream" \
  --filter-pattern "${.userIdentity.type = Root}" \
  --destination-arn "arn:aws:logs:region:999999999999:destination:testDestination"
```

The log group and the destination must be in the same AWS Region. However, the destination can point to an AWS resource such as a Amazon Kinesis Data Streams stream that is located in a different Region.

Validate the flow of log events

After you create the subscription filter, CloudWatch Logs forwards all the incoming log events that match the filter pattern to the stream that is encapsulated within the destination stream called "**RecipientStream**". The destination owner can verify that this is happening by using the **aws kinesis get-shard-iterator** command to grab a Amazon Kinesis Data Streams shard, and using the **aws kinesis get-records** command to fetch some Amazon Kinesis Data Streams records:

```
aws kinesis get-shard-iterator \
  --stream-name RecipientStream \
  --shard-id shardId-000000000000 \
  --shard-iterator-type TRIM_HORIZON

{
  "ShardIterator":
    "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
}

aws kinesis get-records \
  --limit 10 \
  --shard-iterator
    "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
```

```
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f
+0IK8zM5My8ID+g6rMo7UKWeI4+IWIKEXAMPLE"
```

Note

You might need to rerun the `get-records` command a few times before Amazon Kinesis Data Streams starts to return data.

You should see a response with an array of Amazon Kinesis Data Streams records. The data attribute in the Amazon Kinesis Data Streams record is compressed in gzip format and then base64 encoded. You can examine the raw data from the command line using the following Unix command:

```
echo -n "<Content of Data>" | base64 -d | zcat
```

The base64 decoded and decompressed data is formatted as JSON with the following structure:

```
{
  "owner": "111111111111",
  "logGroup": "CloudTrail/logs",
  "logStream": "111111111111_CloudTrail/logs_us-east-1",
  "subscriptionFilters": [
    "RecipientStream"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
    }\",
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
    }\",
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
```

```
    "timestamp": 1432826855000,
    "message": "{\"eventVersion\":\"1.03\", \"userIdentity\": {\"type\": \"Root
  \"}\"
    }
  ]
}
```

The key elements in this data structure are as follows:

owner

The AWS Account ID of the originating log data.

logGroup

The log group name of the originating log data.

logStream

The log stream name of the originating log data.

subscriptionFilters

The list of subscription filter names that matched with the originating log data.

messageType

Data messages use the "DATA_MESSAGE" type. Sometimes CloudWatch Logs may emit Amazon Kinesis Data Streams records with a "CONTROL_MESSAGE" type, mainly for checking if the destination is reachable.

logEvents

The actual log data, represented as an array of log event records. The ID property is a unique identifier for every log event.

Modify destination membership at runtime

You might encounter situations where you have to add or remove membership of some users from a destination that you own. You can use the `put-destination-policy` command on your destination with a new access policy. In the following example, a previously added account **111111111111** is stopped from sending any more log data, and account **222222222222** is enabled.

1. Fetch the policy that is currently associated with the destination **testDestination** and make a note of the **AccessPolicy**:

```
aws logs describe-destinations \
  --destination-name-prefix "testDestination"

{
  "Destinations": [
    {
      "DestinationName": "testDestination",
      "RoleArn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
      "DestinationArn":
        "arn:aws:logs:region:999999999999:destination:testDestination",
      "TargetArn": "arn:aws:kinesis:region:999999999999:stream/RecipientStream",
      "AccessPolicy": "{\n\"Version\": \"2012-10-17\", \"Statement\":
        [{\n\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\n\"AWS\":
        \"111111111111\"}, \"Action\": \"logs:PutSubscriptionFilter\", \"Resource\":
        \"arn:aws:logs:region:999999999999:destination:testDestination\"}]}\"
    }
  ]
}
```

2. Update the policy to reflect that account **111111111111** is stopped, and that account **222222222222** is enabled. Put this policy in the **~/NewAccessPolicy.json** file:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "222222222222"
      },
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-
east-1:999999999999:destination:testDestination"
    }
  ]
}
```

```
}
```

3. Call **PutDestinationPolicy** to associate the policy defined in the **NewAccessPolicy.json** file with the destination:

```
aws logs put-destination-policy \  
--destination-name "testDestination" \  
--access-policy file://~/NewAccessPolicy.json
```

This will eventually disable the log events from account ID **111111111111**. Log events from account ID **222222222222** start flowing to the destination as soon as the owner of account **222222222222** creates a subscription filter.

Updating an existing cross-account subscription

If you currently have a cross-account logs subscription where the destination account grants permissions only to specific sender accounts, and you want to update this subscription so that the destination account grants access to all accounts in an organization, follow the steps in this section.

Topics

- [Step 1: Update the subscription filters](#)
- [Step 2: Update the existing destination access policy](#)

Step 1: Update the subscription filters

Note

This step is needed only for cross-account subscriptions for logs that are created by the services listed in [Enable logging from AWS services](#). If you are not working with logs created by one of these log groups, you can skip to [Step 2: Update the existing destination access policy](#).

In certain cases, you must update the subscription filters in all the sender accounts that are sending logs to the destination account. The update adds an IAM role, which CloudWatch can assume and validate that the sender account has permission to send logs to the recipient account.

Follow the steps in this section for every sender account that you want to update to use organization ID for the cross-account subscription permissions.

In the examples in this section, two accounts, 111111111111 and 222222222222 already have subscription filters created to send logs to account 999999999999. The existing subscription filter values are as follows:

```
## Existing Subscription Filter parameter values
\ --log-group-name "my-log-group-name"
\ --filter-name "RecipientStream"
\ --filter-pattern "{$.userIdentity.type = Root}"
\ --destination-arn "arn:aws:logs:region:999999999999:destination:testDestination"
```

If you need to find the current subscription filter parameter values, enter the following command.

```
aws logs describe-subscription-filters
\ --log-group-name "my-log-group-name"
```

To update a subscription filter to start using organization IDs for cross-account log permissions

1. Create the following trust policy in a file `~/TrustPolicyForCWL.json`. Use a text editor to create this policy file; do not use the IAM console.

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. Create the IAM role that uses this policy. Take note of the Arn value of the Arn value that is returned by the command, you will need it later in this procedure. In this example, we use `CWLtoSubscriptionFilterRole` for the name of the role we're creating.

```
aws iam create-role
\ --role-name CWLtoSubscriptionFilterRole
\ --assume-role-policy-document file://~/TrustPolicyForCWL.json
```

3. Create a permissions policy to define the actions that CloudWatch Logs can perform on your account.

- a. First, use a text editor to create the following permissions policy in a file named `/PermissionsForCWLSubscriptionFilter.json`.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:region:111111111111:log-
group:LogGroupOnWhichSubscriptionFilterIsCreated:*"
    }
  ]
}
```

- b. Enter the following command to associate the permissions policy you just created with the role that you created in step 2.

```
aws iam put-role-policy
  --role-name CWLtoSubscriptionFilterRole
  --policy-name Permissions-Policy-For-CWL-Subscription-filter
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

4. Enter the following command to update the subscription filter.

```
aws logs put-subscription-filter
  \ --log-group-name "my-log-group-name"
  \ --filter-name "RecipientStream"
  \ --filter-pattern "${.userIdentity.type = Root}"
  \ --destination-arn
  "arn:aws:logs:region:999999999999:destination:testDestination"
  \ --role-arn "arn:aws:iam::111111111111:role/CWLtoSubscriptionFilterRole"
```

Step 2: Update the existing destination access policy

After you have updated the subscription filters in all of the sender accounts, you can update the destination access policy in the recipient account.

In the following examples, the recipient account is 999999999999 and the destination is named `testDestination`.

The update enables all accounts that are part of the organization with ID `o-1234567890` to send logs to the recipient account. Only the accounts that have subscription filters created will actually send logs to the recipient account.

To update the destination access policy in the recipient account to start using an organization ID for permissions

1. In the recipient account, use a text editor to create a `~/AccessPolicy.json` file with the following contents.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

2. Enter the following command to attach the policy that you just created to the existing destination. To update a destination to use an access policy with an organization ID instead of an access policy that lists specific AWS account IDs, include the `force` parameter.

⚠ Warning

If you are working with logs sent by an AWS service listed in [Enable logging from AWS services](#), then before doing this step, you must have first updated the subscription filters in all the sender accounts as explained in [Step 1: Update the subscription filters](#).

```
aws logs put-destination-policy
\ --destination-name "testDestination"
\ --access-policy file://~/AccessPolicy.json
\ --force
```

Cross-account cross-Region log data sharing using Firehose

To share log data across accounts, you need to establish a log data sender and receiver:

- **Log data sender**—gets the destination information from the recipient and lets CloudWatch Logs know that it is ready to send its log events to the specified destination. In the procedures in the rest of this section, the log data sender is shown with a fictional AWS account number of 111111111111.
- **Log data recipient**—sets up a destination that encapsulates a Amazon Kinesis Data Streams stream and lets CloudWatch Logs know that the recipient wants to receive log data. The recipient then shares the information about this destination with the sender. In the procedures in the rest of this section, the log data recipient is shown with a fictional AWS account number of 222222222222.

The example in this section uses a Firehose delivery stream with Amazon S3 storage. You can also set up Firehose delivery streams with different settings. For more information, see [Creating a Firehose Delivery Stream](#).

i Note

The log group and the destination must be in the same AWS Region. However, the AWS resource that the destination points to can be located in a different Region.

Note

Firehose subscription filter for a **same account** and **cross-Region** delivery stream is supported.

Topics

- [Step 1: Create a Firehose delivery stream](#)
- [Step 2: Create a destination](#)
- [Step 3: Add/validate IAM permissions for the cross-account destination](#)
- [Step 4: Create a subscription filter](#)
- [Validating the flow of log events](#)
- [Modifying destination membership at runtime](#)

Step 1: Create a Firehose delivery stream**⚠ Important**

Before you complete the following steps, you must use an access policy, so Firehose can access your Amazon S3 bucket. For more information, see [Controlling Access](#) in the *Amazon Data Firehose Developer Guide*.

All of the steps in this section (Step 1) must be done in the log data recipient account. US East (N. Virginia) is used in the following sample commands. Replace this Region with the correct Region for your deployment.

To create a Firehose delivery stream to be used as the destination

1. Create an Amazon S3 bucket:

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --create-bucket-configuration
LocationConstraint=us-east-1
```

2. Create the IAM role that grants Firehose permission to put data into the bucket.
 - a. First, use a text editor to create a trust policy in a file `~/TrustPolicyForFirehose.json`.

```
{ "Statement": { "Effect": "Allow", "Principal": { "Service":
  "firehose.amazonaws.com" }, "Action": "sts:AssumeRole", "Condition":
  { "StringEquals": { "sts:ExternalId": "222222222222" } } } }
```

- b. Create the IAM role, specifying the trust policy file that you just made.

```
aws iam create-role \
  --role-name FirehoseToS3Role \
  --assume-role-policy-document file://~/TrustPolicyForFirehose.json
```

- c. The output of this command will look similar to the following. Make a note of the role name and the role ARN.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "FirehoseToS3Role",
    "RoleId": "AROAR3BXASEKW7K635M53",
    "Arn": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
    "CreateDate": "2021-02-02T07:53:10+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "222222222222"
          }
        }
      }
    }
  }
}
```

3. Create a permissions policy to define the actions that Firehose can perform in your account.
 - a. First, use a text editor to create the following permissions policy in a file named `~/PermissionsForFirehose.json`. Depending on your use case, you might need to add more permissions to this file.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  }]
}
```

- b. Enter the following command to associate the permissions policy that you just created with the IAM role.

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name
Permissions-Policy-For-Firehose-To-S3 --policy-document file:///~/
PermissionsForFirehose.json
```

4. Enter the following command to create the Firehose delivery stream. Replace *my-role-arn* and *amzn-s3-demo-bucket2-arn* with the correct values for your deployment.

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role", "BucketARN":
  "arn:aws:s3:::amzn-s3-demo-bucket"}'
```

The output should look similar to the following:

```
{
  "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream"
}
```

Step 2: Create a destination

Important

All steps in this procedure are to be done in the log data recipient account.

When the destination is created, CloudWatch Logs sends a test message to the destination on the recipient account's behalf. When the subscription filter is active later, CloudWatch Logs sends log events to the destination on the source account's behalf.

To create a destination

1. Wait until the Firehose stream that you created in [Step 1: Create a Firehose delivery stream](#) becomes active. You can use the following command to check the **StreamDescription.StreamStatus** property.

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
```

In addition, take note of the **DeliveryStreamDescription.DeliveryStreamARN** value, because you will need to use it in a later step. Sample output of this command:

```
{
  "DeliveryStreamDescription": {
    "DeliveryStreamName": "my-delivery-stream",
    "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamEncryptionConfiguration": {
      "Status": "DISABLED"
    },
    "DeliveryStreamType": "DirectPut",
    "VersionId": "1",
    "CreateTimestamp": "2021-02-01T23:59:15.567000-08:00",
    "Destinations": [
      {
        "DestinationId": "destinationId-000000000001",
        "S3DestinationDescription": {
          "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
          "BufferingHints": {
```

```

        "SizeInMBs": 5,
        "IntervalInSeconds": 300
    },
    "CompressionFormat": "UNCOMPRESSED",
    "EncryptionConfiguration": {
        "NoEncryptionConfig": "NoEncryption"
    },
    "CloudWatchLoggingOptions": {
        "Enabled": false
    }
},
"ExtendedS3DestinationDescription": {
    "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
    "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
    "BufferingHints": {
        "SizeInMBs": 5,
        "IntervalInSeconds": 300
    },
    "CompressionFormat": "UNCOMPRESSED",
    "EncryptionConfiguration": {
        "NoEncryptionConfig": "NoEncryption"
    },
    "CloudWatchLoggingOptions": {
        "Enabled": false
    },
    "S3BackupMode": "Disabled"
}
},
],
"HasMoreDestinations": false
}
}

```

It might take a minute or two for your delivery stream to show up in the active state.

2. When the delivery stream is active, create the IAM role that will grant CloudWatch Logs the permission to put data into your Firehose stream. First, you'll need to create a trust policy in a file **~/TrustPolicyForCWL.json**. Use a text editor to create this policy. For more information about CloudWatch Logs endpoints, see [Amazon CloudWatch Logs endpoints and quotas](#).

This policy includes a `aws:SourceArn` global condition context key that specifies the `sourceAccountId` to help prevent the confused deputy security problem. If you don't yet know the source account ID in the first call, we recommend that you put the destination ARN

in the source ARN field. In the subsequent calls, you should set the source ARN to be the actual source ARN that you gathered from the first call. For more information, see [Confused deputy prevention](#).

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.region.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:",
          "arn:aws:logs:region:recipientAccountId:"
        ]
      }
    }
  }
}
```

3. Use the **aws iam create-role** command to create the IAM role, specifying the trust policy file that you just created.

```
aws iam create-role \
  --role-name CWLtoKinesisFirehoseRole \
  --assume-role-policy-document file://~/TrustPolicyForCWL.json
```

The following is a sample output. Take note of the returned `Role.Arn` value, because you will need to use it in a later step.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "RoleId": "AR0AR3BXASEKYJYWF243H",
    "Arn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
    "CreateDate": "2021-02-02T08:10:43+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
```

```

        "Principal": {
            "Service": "logs.region.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringLike": {
                "aws:SourceArn": [
                    "arn:aws:logs:region:sourceAccountId:*",
                    "arn:aws:logs:region:recipientAccountId:"
                ]
            }
        }
    }
}

```

4. Create a permissions policy to define which actions CloudWatch Logs can perform on your account. First, use a text editor to create a permissions policy in a file `~/PermissionsForCWL.json`:

```

{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["firehose:*"],
            "Resource": ["arn:aws:firehose:region:222222222222:*"]
        }
    ]
}

```

5. Associate the permissions policy with the role by entering the following command:

```

aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-name
Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json

```

6. After the Firehose delivery stream is in the active state and you have created the IAM role, you can create the CloudWatch Logs destination.
 - a. This step will not associate an access policy with your destination and is only the first step out of two that completes a destination creation. Make a note of the ARN of

the new destination that is returned in the payload, because you will use this as the `destination.arn` in a later step.

```
aws logs put-destination \

    --destination-name "testFirehoseDestination" \
    --target-arn "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-
delivery-stream" \
    --role-arn "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole"

{
  "destination": {
    "destinationName": "testFirehoseDestination",
    "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
    "roleArn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
    "arn": "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"}
}
```

- b. After the previous step is complete, in the log data recipient account (222222222222), associate an access policy with the destination.

This policy enables the log data sender account (111111111111) to access the destination in just the log data recipient account (222222222222). You can use a text editor to put this policy in the `~/AccessPolicy.json` file:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111111111111"
      },
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"
    }
  ]
}
```

```
]
}
```

- c. This creates a policy that defines who has write access to the destination. This policy must specify the **logs:PutSubscriptionFilter** action to access the destination. Cross-account users will use the **PutSubscriptionFilter** action to send log events to the destination:

```
aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \
  --access-policy file:///~/AccessPolicy.json
```

Step 3: Add/validate IAM permissions for the cross-account destination

According to AWS cross-account policy evaluation logic, in order to access any cross-account resource (such as an Kinesis or Firehose stream used as a destination for a subscription filter) you must have an identity-based policy in the sending account which provides explicit access to the cross-account destination resource. For more information about policy evaluation logic, see [Cross-account policy evaluation logic](#).

You can attach the identity-based policy to the IAM role or IAM user that you are using to create the subscription filter. This policy must be present in the sending account. If you are using the Administrator role to create the subscription filter, you can skip this step and move on to [Step 4: Create a subscription filter](#).

To add or validate the IAM permissions needed for cross-account

1. Enter the following command to check which IAM role or IAM user is being used to run AWS logs commands.

```
aws sts get-caller-identity
```

The command returns output similar to the following:

```
{
  "UserId": "User ID",
  "Account": "sending account id",
  "Arn": "arn:aws:sending account id:role/user:RoleName/UserName"
}
```

Make note of the value represented by *RoleName* or *UserName*.

2. Sign into the AWS Management Console in the sending account and search for the attached policies with the IAM role or IAM user returned in the output of the command you entered in step 1.
3. Verify that the policies attached to this role or user provide explicit permissions to call `logs:PutSubscriptionFilter` on the cross-account destination resource.

The following policy provides permissions to create a subscription filter on any destination resource only in a single AWS account, account 999999999999:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":
"AllowSubscriptionFiltersOnAnyResourceInOneSpecificAccount",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:123456789012:destination:*"
      ]
    }
  ]
}
```

The following policy provides permissions to create a subscription filter only on a specific destination resource named `sampleDestination` in single AWS account, account 123456789012:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

    {
      "Sid": "AllowSubscriptionFiltersOnSpecificResource",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:123456789012:destination:amzn-s3-demo-bucket"
      ]
    }
  ]
}

```

Step 4: Create a subscription filter

Switch to the sending account, which is 111111111111 in this example. You will now create the subscription filter in the sending account. In this example, the filter is associated with a log group containing AWS CloudTrail events so that every logged activity made by "Root" AWS credentials is delivered to the destination you previously created. For more information about how to send AWS CloudTrail events to CloudWatch Logs, see [Sending CloudTrail Events to CloudWatch Logs](#) in the *AWS CloudTrail User Guide*.

When you enter the following command, be sure you are signed in as the IAM user or using the IAM role that you added the policy for, in [Step 3: Add/validate IAM permissions for the cross-account destination](#).

```

aws logs put-subscription-filter \
  --log-group-name "aws-cloudtrail-logs-111111111111-300a971e" \
  --filter-name "firehose_test" \
  --filter-pattern "${$.userIdentity.type = AssumedRole}" \
  --destination-arn "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"

```

The log group and the destination must be in the same AWS Region. However, the destination can point to an AWS resource such as a Firehose stream that is located in a different Region.

Validating the flow of log events

After you create the subscription filter, CloudWatch Logs forwards all the incoming log events that match the filter pattern to the Firehose delivery stream. The data starts appearing in your Amazon S3 bucket based on the time buffer interval that is set on the Firehose delivery stream.

Once enough time has passed, you can verify your data by checking the Amazon S3 bucket. To check the bucket, enter the following command:

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket'
```

The output of that command will be similar to the following:

```
{
  "Contents": [
    {
      "Key": "2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba",
      "LastModified": "2021-02-02T09:00:26+00:00",
      "ETag": "\"EXAMPLEa817fb88fc770b81c8f990d\"",
      "Size": 198,
      "StorageClass": "STANDARD",
      "Owner": {
        "DisplayName": "firehose+2test",
        "ID": "EXAMPLE27fd05889c665d2636218451970ef79400e3d2aecca3adb1930042e0"
      }
    }
  ]
}
```

You can then retrieve a specific object from the bucket by entering the following command. Replace the value of key with the value you found in the previous command.

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket' --key '2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba' testfile.gz
```

The data in the Amazon S3 object is compressed with the gzip format. You can examine the raw data from the command line using one of the following commands:

Linux:

```
zcat testfile.gz
```

macOS:

```
zcat <testfile.gz
```

Modifying destination membership at runtime

You might encounter situations where you have to add or remove log senders from a destination that you own. You can use the **PutDestinationPolicy** action on your destination with new access policy. In the following example, a previously added account **111111111111** is stopped from sending any more log data, and account **333333333333** is enabled.

1. Fetch the policy that is currently associated with the destination **testDestination** and make a note of the **AccessPolicy**:

```
aws logs describe-destinations \
  --destination-name-prefix "testFirehoseDestination"

{
  "destinations": [
    {
      "destinationName": "testFirehoseDestination",
      "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
      "roleArn": "arn:aws:iam:: 222222222222:role/CWLtoKinesisFirehoseRole",
      "accessPolicy": "{\n  \"Version\" : \"2012-10-17\",\n  \"Statement
\" : [\n    {\n      \"Sid\" : \"\", \n      \"Effect\" : \"Allow\", \n
      \"Principal\" : {\n        \"AWS\" : \"111111111111 \"\n      }, \n      \"Action
\" : \"logs:PutSubscriptionFilter\", \n      \"Resource\" : \"arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination\"\n    } \n  ] \n} \n \n",
      "arn": "arn:aws:logs:us-east-1:
222222222222:destination:testFirehoseDestination",
      "creationTime": 1612256124430
    }
  ]
}
```

2. Update the policy to reflect that account **111111111111** is stopped, and that account **333333333333** is enabled. Put this policy in the **~/NewAccessPolicy.json** file:

JSON

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
```

```
"Sid" : "",
"Effect" : "Allow",
"Principal" : {
  "AWS" : "333333333333 "
},
"Action" : "logs:PutSubscriptionFilter",
"Resource" : "arn:aws:logs:us-east-1:222222222222:destination:testFirehoseDestination"
}
]
```

3. Use the following command to associate the policy defined in the **NewAccessPolicy.json** file with the destination:

```
aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \
  --access-policy file://~/NewAccessPolicy.json
```

This eventually disables the log events from account ID **111111111111**. Log events from account ID **333333333333** start flowing to the destination as soon as the owner of account **333333333333** creates a subscription filter.

Cross-account cross-Region account-level subscriptions using Amazon Kinesis Data Streams

When you create a cross-account subscription, you can specify a single account or an organization to be the sender. If you specify an organization, then this procedure enables all accounts in the organization to send logs to the receiver account.

To share log data across accounts, you need to establish a log data sender and receiver:

- **Log data sender**—gets the destination information from the recipient and lets CloudWatch Logs know that it's ready to send its log events to the specified destination. In the procedures in the rest of this section, the log data sender is shown with a fictional AWS account number of 111111111111.

If you're going to have multiple accounts in one organization send logs to one recipient account, you can create a policy that grants all accounts in the organization the permission to send logs

to the recipient account. You still have to set up separate subscription filters for each sender account.

- **Log data recipient**—sets up a destination that encapsulates a Amazon Kinesis Data Streams stream and lets CloudWatch Logs know that the recipient wants to receive log data. The recipient then shares the information about this destination with the sender. In the procedures in the rest of this section, the log data recipient is shown with a fictional AWS account number of 999999999999.

To start receiving log events from cross-account users, the log data recipient first creates a CloudWatch Logs destination. Each destination consists of the following key elements:

Destination name

The name of the destination you want to create.

Target ARN

The Amazon Resource Name (ARN) of the AWS resource that you want to use as the destination of the subscription feed.

Role ARN

An AWS Identity and Access Management (IAM) role that grants CloudWatch Logs the necessary permissions to put data into the chosen stream.

Access policy

An IAM policy document (in JSON format, written using IAM policy grammar) that governs the set of users that are allowed to write to your destination.

Note

The log group and the destination must be in the same AWS Region. However, the AWS resource that the destination points to can be located in a different Region. In the examples in the following sections, all Region-specific resources are created in US East (N. Virginia).

Topics

- [Setting up a new cross-account subscription](#)
- [Updating an existing cross-account subscription](#)

Setting up a new cross-account subscription

Follow the steps in these sections to set up a new cross-account log subscription.

Topics

- [Step 1: Create a destination](#)
- [Step 2: \(Only if using an organization\) Create an IAM role](#)
- [Step 3: Create an account-level subscription filter policy](#)
- [Validate the flow of log events](#)
- [Modify destination membership at runtime](#)

Step 1: Create a destination

Important

All steps in this procedure are to be done in the log data recipient account.

For this example, the log data recipient account has an AWS account ID of 999999999999, while the log data sender AWS account ID is 111111111111.

This example creates a destination using a Amazon Kinesis Data Streams stream called RecipientStream, and a role that enables CloudWatch Logs to write data to it.

When the destination is created, CloudWatch Logs sends a test message to the destination on the recipient account's behalf. When the subscription filter is active later, CloudWatch Logs sends log events to the destination on the source account's behalf.

To create a destination

1. In the recipient account, create a destination stream in Amazon Kinesis Data Streams. At a command prompt, type:

```
aws kinesis create-stream --stream-name "RecipientStream" --shard-count 1
```

2. Wait until the stream becomes active. You can use the **aws kinesis describe-stream** command to check the **StreamDescription.StreamStatus** property. In addition, take note of the **StreamDescription.StreamARN** value because you will pass it to CloudWatch Logs later:

```
aws kinesis describe-stream --stream-name "RecipientStream"
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "RecipientStream",
    "StreamARN": "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "34028236692093846346337460743176EXAMPLE",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
"4955113521868881845667950383198145878459135270218EXAMPLE"
        }
      }
    ]
  }
}
```

It might take a minute or two for your stream to show up in the active state.

3. Create the IAM role that grants CloudWatch Logs the permission to put data into your stream. First, you'll need to create a trust policy in a file **~/TrustPolicyForCWL.json**. Use a text editor to create this policy file, do not use the IAM console.

This policy includes a `aws:SourceArn` global condition context key that specifies the `sourceAccountId` to help prevent the confused deputy security problem. If you don't yet know the source account ID in the first call, we recommend that you put the destination ARN in the source ARN field. In the subsequent calls, you should set the source ARN to be the actual source ARN that you gathered from the first call. For more information, see [Confused deputy prevention](#).

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.amazonaws.com"
    },
  },
}
```

```

    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:*",
          "arn:aws:logs:region:recipientAccountId:"
        ]
      }
    },
    "Action": "sts:AssumeRole"
  }
}

```

4. Use the **aws iam create-role** command to create the IAM role, specifying the trust policy file. Take note of the returned Role.Arn value because it will also be passed to CloudWatch Logs later:

```

aws iam create-role \
--role-name CWLtoKinesisRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:*",
              "arn:aws:logs:region:recipientAccountId:"
            ]
          }
        },
        "Principal": {
          "Service": "logs.amazonaws.com"
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2023-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",

```

```

    "Arn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole"
  }
}

```

5. Create a permissions policy to define which actions CloudWatch Logs can perform on your account. First, use a text editor to create a permissions policy in a file `~/PermissionsForCWL.json`:

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:region:999999999999:stream/RecipientStream"
    }
  ]
}

```

6. Associate the permissions policy with the role by using the `aws iam put-role-policy` command:

```

aws iam put-role-policy \
  --role-name CWLtoKinesisRole \
  --policy-name Permissions-Policy-For-CWL \
  --policy-document file://~/PermissionsForCWL.json

```

7. After the stream is in the active state and you have created the IAM role, you can create the CloudWatch Logs destination.
 - a. This step doesn't associate an access policy with your destination and is only the first step out of two that completes a destination creation. Make a note of the **DestinationArn** that is returned in the payload:

```

aws logs put-destination \
  --destination-name "testDestination" \
  --target-arn "arn:aws:kinesis:region:999999999999:stream/RecipientStream" \
  --role-arn "arn:aws:iam::999999999999:role/CWLtoKinesisRole"

{
  "DestinationName" : "testDestination",
  "RoleArn" : "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
  "DestinationArn" : "arn:aws:logs:us-east-1:999999999999:destination:testDestination",

```

```
"TargetArn" : "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream"
}
```

- b. After step 7a is complete, in the log data recipient account, associate an access policy with the destination. This policy must specify the **logs:PutSubscriptionFilter** action and grants permission to the sender account to access the destination.

The policy grants permission to the AWS account that sends logs. You can specify just this one account in the policy, or if the sender account is a member of an organization, the policy can specify the organization ID of the organization. This way, you can create just one policy to allow multiple accounts in one organization to send logs to this destination account.

Use a text editor to create a file named `~/AccessPolicy.json` with one of the following policy statements.

This first example policy allows all accounts in the organization that have an ID of `o-1234567890` to send logs to the recipient account.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

```
    }
  ]
}
```

This next example allows just the log data sender account (111111111111) to send logs to the log data recipient account.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111111111111"
      },
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination"
    }
  ]
}
```

- c. Attach the policy you created in the previous step to the destination.

```
aws logs put-destination-policy \
  --destination-name "testDestination" \
  --access-policy file://~/AccessPolicy.json
```

This access policy enables users in the AWS Account with ID 111111111111 to call **PutSubscriptionFilter** against the destination with ARN `arn:aws:logs:region:999999999999:destination:testDestination`. Any other user's attempt to call `PutSubscriptionFilter` against this destination will be rejected.

To validate a user's privileges against an access policy, see [Using Policy Validator](#) in the *IAM User Guide*.

When you have finished, if you're using AWS Organizations for your cross-account permissions, follow the steps in [Step 2: \(Only if using an organization\) Create an IAM role](#). If you're granting permissions directly to the other account instead of using Organizations, you can skip that step and proceed to [Step 3: Create an account-level subscription filter policy](#).

Step 2: (Only if using an organization) Create an IAM role

In the previous section, if you created the destination by using an access policy that grants permissions to the organization that account 111111111111 is in, instead of granting permissions directly to account 111111111111, then follow the steps in this section. Otherwise, you can skip to [Step 3: Create an account-level subscription filter policy](#).

The steps in this section create an IAM role, which CloudWatch can assume and validate whether the sender account has permission to create a subscription filter against the recipient destination.

Perform the steps in this section in the sender account. The role must exist in the sender account, and you specify the ARN of this role in the subscription filter. In this example, the sender account is 111111111111.

To create the IAM role necessary for cross-account log subscriptions using AWS Organizations

1. Create the following trust policy in a file / `TrustPolicyForCWLSubscriptionFilter.json`. Use a text editor to create this policy file; do not use the IAM console.

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. Create the IAM role that uses this policy. Take note of the `Arn` value that is returned by the command, you will need it later in this procedure. In this example, we use `CWLtoSubscriptionFilterRole` for the name of the role we're creating.

```
aws iam create-role \  
    --role-name CWLtoSubscriptionFilterRole \  
    --assume-role-policy-document file://~/TrustPolicyForCWLSubscriptionFilter.json
```

3. Create a permissions policy to define the actions that CloudWatch Logs can perform on your account.
 - a. First, use a text editor to create the following permissions policy in a file named ~/PermissionsForCWLSubscriptionFilter.json.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "logs:PutLogEvents",  
      "Resource": "arn:aws:logs:region:111111111111:log-  
group:LogGroupOnWhichSubscriptionFilterIsCreated:*"  
    }  
  ]  
}
```

- b. Enter the following command to associate the permissions policy you just created with the role that you created in step 2.

```
aws iam put-role-policy  
    --role-name CWLtoSubscriptionFilterRole  
    --policy-name Permissions-Policy-For-CWL-Subscription-filter  
    --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

When you have finished, you can proceed to [Step 3: Create an account-level subscription filter policy](#).

Step 3: Create an account-level subscription filter policy

After you create a destination, the log data recipient account can share the destination ARN (arn:aws:logs:us-east-1:999999999999:destination:testDestination) with other AWS accounts so that they can send log events to the same destination. These other sending accounts users then create a subscription filter on their respective log groups against this destination. The subscription

filter immediately starts the flow of real-time log data from the chosen log group to the specified destination.

Note

If you are granting permissions for the subscription filter to an entire organization, you will need to use the ARN of the IAM role that you created in [Step 2: \(Only if using an organization\) Create an IAM role](#).

In the following example, an account-level subscription filter policy is created in a sending account. The filter is associated with the sender account 111111111111 so that every log event matching the filter and selection criteria is delivered to the destination you previously created. That destination encapsulates a stream called "RecipientStream".

The `selection-criteria` field is optional, but is important for excluding log groups that can cause an infinite log recursion from a subscription filter. For more information about this issue and determining which log groups to exclude, see [Log recursion prevention](#). Currently, NOT IN is the only supported operator for `selection-criteria`.

```
aws logs put-account-policy \
  --policy-name "CrossAccountStreamsExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document
'{"DestinationArn":"arn:aws:logs:region:999999999999:destination:testDestination",
"FilterPattern": "", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

The sender account's log groups and the destination must be in the same AWS Region. However, the destination can point to an AWS resource such as an Amazon Kinesis Data Streams stream that is located in a different Region.

Validate the flow of log events

After you create the account-level subscription filter policy, CloudWatch Logs forwards all the incoming log events that match the filter pattern and selection criteria to the stream that is encapsulated within the destination stream called "**RecipientStream**". The destination owner can verify that this is happening by using the `aws kinesys get-shard-iterator` command to grab

a Amazon Kinesis Data Streams shard, and using the **aws kinesis get-records** command to fetch some Amazon Kinesis Data Streams records:

```
aws kinesis get-shard-iterator \
  --stream-name RecipientStream \
  --shard-id shardId-000000000000 \
  --shard-iterator-type TRIM_HORIZON

{
  "ShardIterator":
  "AAAAAAAAAAGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGB9v4scv+3vaq+f
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
}

aws kinesis get-records \
  --limit 10 \
  --shard-iterator
  "AAAAAAAAAAGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGB9v4scv+3vaq+f
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
```

Note

You might need to rerun the `get-records` command a few times before Amazon Kinesis Data Streams starts to return data.

You should see a response with an array of Amazon Kinesis Data Streams records. The data attribute in the Amazon Kinesis Data Streams record is compressed in gzip format and then base64 encoded. You can examine the raw data from the command line using the following Unix command:

```
echo -n "<Content of Data>" | base64 -d | zcat
```

The base64 decoded and decompressed data is formatted as JSON with the following structure:

```
{
```

```

    "owner": "111111111111",
    "logGroup": "CloudTrail/logs",
    "logStream": "111111111111_CloudTrail/logs_us-east-1",
    "subscriptionFilters": [
        "RecipientStream"
    ],
    "messageType": "DATA_MESSAGE",
    "logEvents": [
        {
            "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
        \",\"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}, \"eventTime\":\"2015-05-28T14:28:26.855Z\", \"eventSource\":\"aws:cloudtrail\", \"eventName\":\"CreateUser\", \"awsEventDetails\":{\"userIdentity\":{\"type\":\"Root\", \"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}, \"eventTime\":\"2015-05-28T14:28:26.855Z\", \"eventSource\":\"aws:cloudtrail\", \"eventName\":\"CreateUser\", \"awsEventDetails\":{\"userIdentity\":{\"type\":\"Root\", \"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}}"
        },
        {
            "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
        \",\"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}, \"eventTime\":\"2015-05-28T14:28:26.855Z\", \"eventSource\":\"aws:cloudtrail\", \"eventName\":\"CreateUser\", \"awsEventDetails\":{\"userIdentity\":{\"type\":\"Root\", \"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}, \"eventTime\":\"2015-05-28T14:28:26.855Z\", \"eventSource\":\"aws:cloudtrail\", \"eventName\":\"CreateUser\", \"awsEventDetails\":{\"userIdentity\":{\"type\":\"Root\", \"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}}"
        },
        {
            "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
        \",\"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}, \"eventTime\":\"2015-05-28T14:28:26.855Z\", \"eventSource\":\"aws:cloudtrail\", \"eventName\":\"CreateUser\", \"awsEventDetails\":{\"userIdentity\":{\"type\":\"Root\", \"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}, \"eventTime\":\"2015-05-28T14:28:26.855Z\", \"eventSource\":\"aws:cloudtrail\", \"eventName\":\"CreateUser\", \"awsEventDetails\":{\"userIdentity\":{\"type\":\"Root\", \"principalId\":\"AIDAJLPLWS337U737225D5656F6598090\", \"arn\":\"arn:aws:iam::111111111111:root\", \"accountId\":\"111111111111\", \"userName\":\"root\"}}}"
        }
    ]
}

```

The key elements in the data structure are the following:

messageType

Data messages will use the "DATA_MESSAGE" type. Sometimes CloudWatch Logs might emit Amazon Kinesis Data Streams records with a "CONTROL_MESSAGE" type, mainly for checking if the destination is reachable.

owner

The AWS Account ID of the originating log data.

logGroup

The log group name of the originating log data.

logStream

The log stream name of the originating log data.

subscriptionFilters

The list of subscription filter names that matched with the originating log data.

logEvents

The actual log data, represented as an array of log event records. The "id" property is a unique identifier for every log event.

policyLevel

The level at which the policy was enforced. "ACCOUNT_LEVEL_POLICY" is the policyLevel for an account-level subscription filter policy.

Modify destination membership at runtime

You might encounter situations where you have to add or remove membership of some users from a destination that you own. You can use the `put-destination-policy` command on your destination with a new access policy. In the following example, a previously added account **111111111111** is stopped from sending any more log data, and account **222222222222** is enabled.

1. Fetch the policy that is currently associated with the destination **testDestination** and make a note of the **AccessPolicy**:

```
aws logs describe-destinations \
  --destination-name-prefix "testDestination"

{
  "Destinations": [
    {
      "DestinationName": "testDestination",
      "RoleArn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
      "DestinationArn":
        "arn:aws:logs:region:999999999999:destination:testDestination",
      "TargetArn": "arn:aws:kinesis:region:999999999999:stream/RecipientStream",
      "AccessPolicy": "{\n\"Version\": \"2012-10-17\", \"Statement\":
        [{\n\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\n\"AWS\":
        \n\"111111111111\"}, \"Action\": \"logs:PutSubscriptionFilter\", \"Resource\":
        \n\"arn:aws:logs:region:999999999999:destination:testDestination\"}] }"
```

```
}
]
}
```

2. Update the policy to reflect that account **111111111111** is stopped, and that account **222222222222** is enabled. Put this policy in the `~/NewAccessPolicy.json` file:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "222222222222"
      },
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination"
    }
  ]
}
```

3. Call **PutDestinationPolicy** to associate the policy defined in the `NewAccessPolicy.json` file with the destination:

```
aws logs put-destination-policy \
--destination-name "testDestination" \
--access-policy file://~/NewAccessPolicy.json
```

This will eventually disable the log events from account ID **111111111111**. Log events from account ID **222222222222** start flowing to the destination as soon as the owner of account **222222222222** creates a subscription filter.

Updating an existing cross-account subscription

If you currently have a cross-account logs subscription where the destination account grants permissions only to specific sender accounts, and you want to update this subscription so that the destination account grants access to all accounts in an organization, follow the steps in this section.

Topics

- [Step 1: Update the subscription filters](#)
- [Step 2: Update the existing destination access policy](#)

Step 1: Update the subscription filters

Note

This step is needed only for cross-account subscriptions for logs that are created by the services listed in [Enable logging from AWS services](#). If you are not working with logs created by one of these log groups, you can skip to [Step 2: Update the existing destination access policy](#).

In certain cases, you must update the subscription filters in all the sender accounts that are sending logs to the destination account. The update adds an IAM role, which CloudWatch can assume and validate that the sender account has permission to send logs to the recipient account.

Follow the steps in this section for every sender account that you want to update to use organization ID for the cross-account subscription permissions.

In the examples in this section, two accounts, 111111111111 and 222222222222 already have subscription filters created to send logs to account 999999999999. The existing subscription filter values are as follows:

```
## Existing Subscription Filter parameter values
{
  "DestinationArn": "arn:aws:logs:region:999999999999:destination:testDestination",
  "FilterPattern": "{\"$.userIdentity.type = Root}\"",
  "Distribution": "Random"
}
```

If you need to find the current subscription filter parameter values, enter the following command.

```
aws logs describe-account-policies \  
--policy-type "SUBSCRIPTION_FILTER_POLICY" \  
--policy-name "CrossAccountStreamsExamplePolicy"
```

To update a subscription filter to start using organization IDs for cross-account log permissions

1. Create the following trust policy in a file `~/TrustPolicyForCWL.json`. Use a text editor to create this policy file; do not use the IAM console.

```
{  
  "Statement": {  
    "Effect": "Allow",  
    "Principal": { "Service": "logs.amazonaws.com" },  
    "Action": "sts:AssumeRole"  
  }  
}
```

2. Create the IAM role that uses this policy. Take note of the Arn value of the Arn value that is returned by the command, you will need it later in this procedure. In this example, we use `CWLtoSubscriptionFilterRole` for the name of the role we're creating.

```
aws iam create-role  
  \ --role-name CWLtoSubscriptionFilterRole  
  \ --assume-role-policy-document file://~/TrustPolicyForCWL.json
```

3. Create a permissions policy to define the actions that CloudWatch Logs can perform on your account.
 - a. First, use a text editor to create the following permissions policy in a file named `PermissionsForCWLSubscriptionFilter.json`.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "logs:PutLogEvents",  
      "Resource": "arn:aws:logs:region:111111111111:log-  
group:LogGroupOnWhichSubscriptionFilterIsCreated:*"  
    }  
  ]  
}
```

- b. Enter the following command to associate the permissions policy you just created with the role that you created in step 2.

```
aws iam put-role-policy
  --role-name CWLtoSubscriptionFilterRole
  --policy-name Permissions-Policy-For-CWL-Subscription-filter
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

4. Enter the following command to update the subscription filter policy.

```
aws logs put-account-policy \
  --policy-name "CrossAccountStreamsExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document
'{"DestinationArn": "arn:aws:logs:region:999999999999:destination:testDestination",
"FilterPattern": "{$.userIdentity.type = Root}", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

Step 2: Update the existing destination access policy

After you have updated the subscription filters in all of the sender accounts, you can update the destination access policy in the recipient account.

In the following examples, the recipient account is 999999999999 and the destination is named testDestination.

The update enables all accounts that are part of the organization with ID o-1234567890 to send logs to the recipient account. Only the accounts that have subscription filters created will actually send logs to the recipient account.

To update the destination access policy in the recipient account to start using an organization ID for permissions

1. In the recipient account, use a text editor to create a ~/AccessPolicy.json file with the following contents.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

2. Enter the following command to attach the policy that you just created to the existing destination. To update a destination to use an access policy with an organization ID instead of an access policy that lists specific AWS account IDs, include the `force` parameter.

** Warning**

If you are working with logs sent by an AWS service listed in [Enable logging from AWS services](#), then before doing this step, you must have first updated the subscription filters in all the sender accounts as explained in [Step 1: Update the subscription filters](#).

```
aws logs put-destination-policy
\ --destination-name "testDestination"
\ --access-policy file://~/AccessPolicy.json
```

```
\ --force
```

Cross-account cross-Region account-level subscriptions using Firehose

To share log data across accounts, you need to establish a log data sender and receiver:

- **Log data sender**—gets the destination information from the recipient and lets CloudWatch Logs know that it is ready to send its log events to the specified destination. In the procedures in the rest of this section, the log data sender is shown with a fictional AWS account number of 111111111111.
- **Log data recipient**—sets up a destination that encapsulates a Amazon Kinesis Data Streams stream and lets CloudWatch Logs know that the recipient wants to receive log data. The recipient then shares the information about this destination with the sender. In the procedures in the rest of this section, the log data recipient is shown with a fictional AWS account number of 222222222222.

The example in this section uses a Firehose delivery stream with Amazon S3 storage. You can also set up Firehose delivery streams with different settings. For more information, see [Creating a Firehose Delivery Stream](#).

Note

The log group and the destination must be in the same AWS Region. However, the AWS resource that the destination points to can be located in a different Region.

Note

Firehose subscription filter for a **same account** and **cross-Region** delivery stream is supported.

Topics

- [Step 1: Create a Firehose delivery stream](#)
- [Step 2: Create a destination](#)
- [Step 3: Create an account-level subscription filter policy](#)

- [Validating the flow of log events](#)
- [Modifying destination membership at runtime](#)

Step 1: Create a Firehose delivery stream

Important

Before you complete the following steps, you must use an access policy, so Firehose can access your Amazon S3 bucket. For more information, see [Controlling Access](#) in the *Amazon Data Firehose Developer Guide*.

All of the steps in this section (Step 1) must be done in the log data recipient account. US East (N. Virginia) is used in the following sample commands. Replace this Region with the correct Region for your deployment.

To create a Firehose delivery stream to be used as the destination

1. Create an Amazon S3 bucket:

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --create-bucket-configuration
LocationConstraint=us-east-1
```

2. Create the IAM role that grants Firehose permission to put data into the bucket.

- a. First, use a text editor to create a trust policy in a file `~/TrustPolicyForFirehose.json`.

```
{ "Statement": { "Effect": "Allow", "Principal": { "Service":
"firehose.amazonaws.com" }, "Action": "sts:AssumeRole", "Condition":
{ "StringEquals": { "sts:ExternalId": "222222222222" } } } }
```

- b. Create the IAM role, specifying the trust policy file that you just made.

```
aws iam create-role \
--role-name FirehoseToS3Role \
--assume-role-policy-document file:///~/TrustPolicyForFirehose.json
```

- c. The output of this command will look similar to the following. Make a note of the role name and the role ARN.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "FirehoseToS3Role",
    "RoleId": "AR0AR3BXASEKW7K635M53",
    "Arn": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
    "CreateDate": "2021-02-02T07:53:10+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "222222222222"
          }
        }
      }
    }
  }
}
```

3. Create a permissions policy to define the actions that Firehose can perform in your account.
 - a. First, use a text editor to create the following permissions policy in a file named `~/PermissionsForFirehose.json`. Depending on your use case, you might need to add more permissions to this file.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  }]
}
```

```
}]
}
```

- b. Enter the following command to associate the permissions policy that you just created with the IAM role.

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name
Permissions-Policy-For-Firehose-To-S3 --policy-document file://~/
PermissionsForFirehose.json
```

4. Enter the following command to create the Firehose delivery stream. Replace *my-role-arn* and *amzn-s3-demo-bucket2-arn* with the correct values for your deployment.

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
    '{"RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role", "BucketARN":
    "arn:aws:s3:::amzn-s3-demo-bucket"}'
```

The output should look similar to the following:

```
{
  "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream"
}
```

Step 2: Create a destination

Important

All steps in this procedure are to be done in the log data recipient account.

When the destination is created, CloudWatch Logs sends a test message to the destination on the recipient account's behalf. When the subscription filter is active later, CloudWatch Logs sends log events to the destination on the source account's behalf.

To create a destination

1. Wait until the Firehose stream that you created in [Step 1: Create a Firehose delivery stream](#) becomes active. You can use the following command to check the **StreamDescription.StreamStatus** property.

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
```

In addition, take note of the **DeliveryStreamDescription.DeliveryStreamARN** value, because you will need to use it in a later step. Sample output of this command:

```
{
  "DeliveryStreamDescription": {
    "DeliveryStreamName": "my-delivery-stream",
    "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamEncryptionConfiguration": {
      "Status": "DISABLED"
    },
    "DeliveryStreamType": "DirectPut",
    "VersionId": "1",
    "CreateTimestamp": "2021-02-01T23:59:15.567000-08:00",
    "Destinations": [
      {
        "DestinationId": "destinationId-000000000001",
        "S3DestinationDescription": {
          "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
          "BufferingHints": {
            "SizeInMBs": 5,
            "IntervalInSeconds": 300
          },
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "CloudWatchLoggingOptions": {
            "Enabled": false
          }
        },
        "ExtendedS3DestinationDescription": {
```

```

        "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
        "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
        "BufferingHints": {
            "SizeInMBs": 5,
            "IntervalInSeconds": 300
        },
        "CompressionFormat": "UNCOMPRESSED",
        "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
        },
        "CloudWatchLoggingOptions": {
            "Enabled": false
        },
        "S3BackupMode": "Disabled"
    }
},
    "HasMoreDestinations": false
}
}

```

It might take a minute or two for your delivery stream to show up in the active state.

2. When the delivery stream is active, create the IAM role that will grant CloudWatch Logs the permission to put data into your Firehose stream. First, you'll need to create a trust policy in a file **~/TrustPolicyForCWL.json**. Use a text editor to create this policy. For more information about CloudWatch Logs endpoints, see [Amazon CloudWatch Logs endpoints and quotas](#).

This policy includes a `aws:SourceArn` global condition context key that specifies the `sourceAccountId` to help prevent the confused deputy security problem. If you don't yet know the source account ID in the first call, we recommend that you put the destination ARN in the source ARN field. In the subsequent calls, you should set the source ARN to be the actual source ARN that you gathered from the first call. For more information, see [Confused deputy prevention](#).

```

{
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
    }
}

```

```

    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:",
          "arn:aws:logs:region:recipientAccountId:"
        ]
      }
    }
  }
}

```

3. Use the **aws iam create-role** command to create the IAM role, specifying the trust policy file that you just created.

```

aws iam create-role \
  --role-name CWLtoKinesisFirehoseRole \
  --assume-role-policy-document file://~/TrustPolicyForCWL.json

```

The following is a sample output. Take note of the returned `Role.Arn` value, because you will need to use it in a later step.

```

{
  "Role": {
    "Path": "/",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "RoleId": "AROAR3BXASEKYJYWF243H",
    "Arn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
    "CreateDate": "2023-02-02T08:10:43+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:",
              "arn:aws:logs:region:recipientAccountId:"
            ]
          }
        }
      }
    }
  }
}

```



```

    }
  }
}

```

4. Create a permissions policy to define which actions CloudWatch Logs can perform on your account. First, use a text editor to create a permissions policy in a file `~/PermissionsForCWL.json`:

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:*"],
      "Resource": ["arn:aws:firehose:region:222222222222:*"]
    }
  ]
}

```

5. Associate the permissions policy with the role by entering the following command:

```

aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-name
Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json

```

6. After the Firehose delivery stream is in the active state and you have created the IAM role, you can create the CloudWatch Logs destination.
 - a. This step will not associate an access policy with your destination and is only the first step out of two that completes a destination creation. Make a note of the ARN of the new destination that is returned in the payload, because you will use this as the `destination.arn` in a later step.

```

aws logs put-destination \

    --destination-name "testFirehoseDestination" \
    --target-arn "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-
delivery-stream" \
    --role-arn "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole"

{
  "destination": {

```

```

    "destinationName": "testFirehoseDestination",
    "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
    "roleArn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
    "arn": "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"}
}

```

- b. After the previous step is complete, in the log data recipient account (222222222222), associate an access policy with the destination. This policy enables the log data sender account (111111111111) to access the destination in just the log data recipient account (222222222222). You can use a text editor to put this policy in the ~/AccessPolicy.json file:

JSON

```

{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "111111111111"
      },
      "Action" : ["logs:PutSubscriptionFilter", "logs:PutAccountPolicy"],
      "Resource" : "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"
    }
  ]
}

```

- c. This creates a policy that defines who has write access to the destination. This policy must specify the `logs:PutSubscriptionFilter` and `logs:PutAccountPolicy` actions to access the destination. Cross-account users will use the `PutSubscriptionFilter` and `PutAccountPolicy` actions to send log events to the destination.

```

aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \
  --access-policy file://~/AccessPolicy.json

```

Step 3: Create an account-level subscription filter policy

Switch to the sending account, which is 111111111111 in this example. You will now create the account-level subscription filter policy in the sending account. In this example, the filter causes every log event containing the string `ERROR` in all but two log groups to be delivered to the destination you previously created.

```
aws logs put-account-policy \
  --policy-name "CrossAccountFirehoseExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document '{"DestinationArn":"arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination", "FilterPattern":
"{$.userIdentity.type = AssumedRole}", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

The sending account's log groups and the destination must be in the same AWS Region. However, the destination can point to an AWS resource such as a Firehose stream that is located in a different Region.

Validating the flow of log events

After you create the subscription filter, CloudWatch Logs forwards all the incoming log events that match the filter pattern and selection criteria to the Firehose delivery stream. The data starts appearing in your Amazon S3 bucket based on the time buffer interval that is set on the Firehose delivery stream. Once enough time has passed, you can verify your data by checking the Amazon S3 bucket. To check the bucket, enter the following command:

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket'
```

The output of that command will be similar to the following:

```
{
  "Contents": [
    {
      "Key": "2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba",
      "LastModified": "2023-02-02T09:00:26+00:00",
      "ETag": "\"EXAMPLEa817fb88fc770b81c8f990d\"",

```

```
    "Size": 198,
    "StorageClass": "STANDARD",
    "Owner": {
      "DisplayName": "firehose+2test",
      "ID": "EXAMPLE27fd05889c665d2636218451970ef79400e3d2aecca3adb1930042e0"
    }
  ]
}
```

You can then retrieve a specific object from the bucket by entering the following command. Replace the value of `key` with the value you found in the previous command.

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket' --key '2021/02/02/08/my-delivery-stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba' testfile.gz
```

The data in the Amazon S3 object is compressed with the gzip format. You can examine the raw data from the command line using one of the following commands:

Linux:

```
zcat testfile.gz
```

macOS:

```
zcat <testfile.gz
```

Modifying destination membership at runtime

You might encounter situations where you have to add or remove log senders from a destination that you own. You can use the **PutDestinationPolicy** and **PutAccountPolicy** actions on your destination with the new access policy. In the following example, a previously added account **111111111111** is stopped from sending any more log data, and account **333333333333** is enabled.

1. Fetch the policy that is currently associated with the destination **testDestination** and make a note of the **AccessPolicy**:

```
aws logs describe-destinations \
```

```
--destination-name-prefix "testFirehoseDestination"
```

The returned data might look like this.

```
{
  "destinations": [
    {
      "destinationName": "testFirehoseDestination",
      "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
      "roleArn": "arn:aws:iam:: 222222222222:role/CWLtoKinesisFirehoseRole",
      "accessPolicy": "{\n  \"Version\" : \"2012-10-17\",\n  \"Statement
\" : [\n    {\n      \"Sid\" : \"\",\n      \"Effect\" : \"Allow\",\n
      \"Principal\" : {\n        \"AWS\" : \"111111111111 \"\n      },\n      \"Action
\" : \"logs:PutSubscriptionFilter\",\n      \"Resource\" : \"arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination\"\n    }\n  ]\n}\n\n",
      "arn": "arn:aws:logs:us-east-1:
222222222222:destination:testFirehoseDestination",
      "creationTime": 1612256124430
    }
  ]
}
```

2. Update the policy to reflect that account **111111111111** is stopped, and that account **333333333333** is enabled. Put this policy in the `~/NewAccessPolicy.json` file:

JSON

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "333333333333 "
      },
      "Action" : ["logs:PutSubscriptionFilter", "logs:PutAccountPolicy"],
      "Resource" : "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"
    }
  ]
}
```

```
}
```

3. Use the following command to associate the policy defined in the **NewAccessPolicy.json** file with the destination:

```
aws logs put-destination-policy \  
  --destination-name "testFirehoseDestination" \  
  
  --access-policy file://~/NewAccessPolicy.json
```

This eventually disables the log events from account ID **111111111111**. Log events from account ID **333333333333** start flowing to the destination as soon as the owner of account **333333333333** creates a subscription filter.

Confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the calling service) calls another service (the called service). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#), [aws:SourceAccount](#), [aws:SourceOrgID](#), and [aws:SourceOrgPaths](#) global condition context keys in resource policies to limit the permissions that gives another service to the resource. Use `aws:SourceArn` to associate only one resource with cross-service access. Use `aws:SourceAccount` to let any resource in that account be associated with the cross-service use. Use `aws:SourceOrgID` to allow any resource from any account within an organization be associated with the cross-service use. Use `aws:SourceOrgPaths` to associate any resource from accounts within an AWS Organizations path with the cross-service use. For more information about using and understanding paths, see [Understand the AWS Organizations entity path](#).

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn`

global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:servicename*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both `aws:SourceAccount` and `aws:SourceArn` to limit permissions.

To protect against the confused deputy problem at scale, use the `aws:SourceOrgID` or `aws:SourceOrgPaths` global condition context key with the organization ID or organization path of the resource in your resource-based policies. Policies that include the `aws:SourceOrgID` or `aws:SourceOrgPaths` key will automatically include the correct accounts and you don't have to manually update the policies when you add, remove, or move accounts in your organization.

The policies documented for granting access to CloudWatch Logs to write data to Amazon Kinesis Data Streams and Firehose in [Step 1: Create a destination](#) and [Step 2: Create a destination](#) show how you can use the `aws:SourceArn` global condition context key to help prevent the confused deputy problem.

Log recursion prevention

There is a risk of causing an infinite log recursion with subscription filters that can lead to a large increase in ingestion billing in both CloudWatch Logs and your destination, if not prevented. This can occur when a subscription filter is associated with a log group that receives log events as a result of your subscription delivery workflow. The logs ingested into the log group will be delivered to the destination, causing the log group to ingest more logs which will then be forwarded again to the destination, creating a recursion loop.

For example, consider a subscription filter with the destination as Firehose, which delivers log events to Amazon S3. Additionally, there is also a Lambda function that processes new events delivered to Amazon S3 and produces some logs itself. If the subscription filter is applied to the Lambda function's log group, then the log events produced by the function will get forwarded to Firehose and Amazon S3 at the destination, which will then invoke the function again, causing more logs to be produced and forwarded to Firehose and Amazon S3, causing another invocation of the function and so on. This will occur in an infinite loop, leading to an unexpected billing increase on log ingestion, Firehose, and Amazon S3.

If the Lambda function is attached to a VPC with flow logs enabled for CloudWatch Logs, then the VPC's log group can cause a log recursion as well.

We recommend that you don't apply subscription filters to log groups that are a part of your subscription delivery workflow. For account-level subscription filters, use the `selectionCriteria` parameter in the `PutAccountPolicy` API to exclude these log groups from the policy.

When excluding log groups, consider the following AWS services that produce logs and may be a part of your subscription delivery workflows:

- Amazon EC2 with Fargate
- Lambda
- AWS Step Functions
- Amazon VPC flow logs that are enabled for CloudWatch Logs

 Note

Log events produced by a Lambda destination's log group will not be forwarded back to the Lambda function for an account-level subscription filter policy. In this case, excluding the destination Lambda function's log group using `selectionCriteria` is not required for account subscription policies.

Filter pattern syntax for metric filters, subscription filters, filter log events, and Live Tail

Note

For information about how to query your log groups with the Amazon CloudWatch Logs Insights query language, see [CloudWatch Logs Insights language query syntax](#).

With CloudWatch Logs, you can use [metric filters](#) to transform log data into actionable metrics, [subscription filters](#) to route log events to other AWS services, [filter log events](#) to search for log events, and [Live Tail](#) to interactively view your logs in real-time as they are ingested.

Filter patterns make up the syntax that metric filters, subscription filters, log events, and Live Tail use to match terms in log events. Terms can be words, exact phrases, or numeric values. Regular expressions (regex) can be used to create standalone filter patterns, or can be incorporated with JSON and space-delimited filter patterns.

Create filter patterns with the terms that you want to match. Filter patterns only return the log events that contain the terms you define. You can test filter patterns in the CloudWatch console.

Topics

- [Supported regular expressions \(regex\) syntax](#)
- [Using filter patterns to match terms with a regular expression \(regex\)](#)
- [Using filter patterns to match terms in unstructured log events](#)
- [Using filter patterns to match terms in JSON log events](#)
- [Using filter patterns to match terms in space-delimited log events](#)

Supported regular expressions (regex) syntax

Supported regex syntax

When using regex to search and filter log data, you must surround your expressions with %.

Filter patterns with regex can only include the following:

- **Alphanumeric characters** – An alphanumeric character is a character that is either a letter (from A to Z or a to z) or a digit (from 0 to 9).
- **Supported symbol characters** – These include: ':', '_', '#', '=', '@', '/', ';', ',', and '-'. For example, `%something!` would be rejected since `!` is not supported.
- **Supported operators** – These include: '^', '\$', '?', '[', ']', '{', '}', '|', '\\', '*', '+', and '.'.

The (and) operators are not supported. You cannot use parentheses to define a subpattern.

Multi-byte characters are not supported.

Note

Quotas

There is a maximum of 5 filter patterns containing regex for each log group when creating metric filters or subscription filters.

There is a limit of 2 regex for each filter pattern when creating a delimited or JSON filter pattern for metric filters and subscription filters or when filtering log events or Live Tail.

Usage of supported operators

- **^**: Anchors the match to the beginning of a string. For example, `^[hc]at%` matches "hat" and "cat", but only at the beginning of a string.
- **\$**: Anchors the match to the end of a string. For example, `%[hc]at$%` matches "hat" and "cat", but only at the end of a string.
- **?**: Matches zero or one occurrence of the preceding term. For example, `%colou?r%` can match both "color" and "colour".
- **[]**: Defines a character class. Matches the character list or character range contained within the brackets. For example, `%[abc]%` matches "a", "b", or "c"; `%[a-z]%` matches any lowercase letter from "a" to "z"; and `%[abcx-z]%` matches "a", "b", "c", "x", "y", or "z".
- **{m, n}**: Matches the preceding term at least *m* and not more than *n* times. For example, `%a{3,5}%` matches only "aaa", "aaaa", and "aaaaa".

Note

Either *m* or *n* can be omitted if you chose not to define a minimum or maximum.

- `|`: Boolean "Or", which matches the term on either side of the vertical bar. For example:
 - `%gra|ey%` can match "gray" or "grey"
 - `%^starting|^initializing|^shutting down%` can match "starting ...", or "initializing ...", or "shutting down", but won't match "skipping initializing ..."
 - `%abcc|ab[^c]%` can match "abcc ..." and "aba ..." but won't match "aac ..."
- `\`: Escape character, which allows you to use the literal meaning of an operator instead of its special meaning. For example, `%\[.\]%` matches any single character surrounded by "[" and "]" since the brackets are escaped, such as "[a]", "[b]", "[7]", "[@]", "[]", and "[]".

Note

`%10\.10\.0\.1%` is the correct way to create a regex to match the IP address 10.10.0.1.

- `*`: Matches zero or more instances of the preceding term. For example, `%ab*c%` can match "ac", "abc", and "abbbc"; `%ab[0-9]*%` can match "ab", "ab0", and "ab129".
- `+`: Matches one or more instances of the preceding term. For example, `%ab+c%` can match "abc", "abbc", and "abbbc", but not "ac".
- `.`: Matches any single character. For example, `%.at%` matches any three character string ending with "at", including "hat", "cat", "bat", "4at", "#at" and " at" (starting with a space).

Note

When creating a regex to match IP addresses, it is important to escape the `.` operator. For example, `%10.10.0.1%` can match "10010,051" which might not be the actual intended purpose of the expression.

- `\d`, `\D`: Matches a digit/non-digit character. For example, `%\d%` is equivalent to `%[0-9]%` and `%\D%` is equivalent to `%[^0-9]%`.

Note

The uppercase operator denotes the inverse of its lowercase counterpart.

- `\s`, `\S`: Matches a whitespace character/non-whitespace character.

Note

The uppercase operator denotes the inverse of its lowercase counterpart. Whitespace characters include the tab (`\t`), space (), and newline (`\n`) characters.

- `\w`, `\W`: Matches an alphanumeric character/non-alphanumeric character. For example, `%\w%` is equivalent to `%[a-zA-Z_0-9] %` and `%\W%` is equivalent to `%[^\a-zA-Z_0-9] %`.

Note

The uppercase operator denotes the inverse of its lowercase counterpart.

- `\xhh`: Matches the ASCII mapping for a two-digit hexadecimal character. `\x` is the escape sequence which indicates that the following characters represent the hexadecimal value for ASCII. `hh` specifies the two hexadecimal digits (0-9 and A-F) which point to a character in the ASCII table.

Note

You can use `\xhh` to match symbol characters that are not supported by the filter pattern. For example, `%\x3A%` matches `;` and `%\x28%` matches `(`.

Using filter patterns to match terms with a regular expression (regex)

Match terms using regex

You can match terms in your log events using a regex pattern surrounded with `%` (percentage signs before and after the regex pattern). The following code snippet shows an example of a filter pattern that returns all log events consisting of the **AUTHORIZED** keyword.

For a list of supported regular expressions, see [Supported regular expressions](#).

```
%AUTHORIZED%
```

This filter pattern returns log event messages, such as the following:

- [ERROR 401] UNAUTHORIZED REQUEST
- [SUCCESS 200] AUTHORIZED REQUEST

Using filter patterns to match terms in unstructured log events

Match terms in unstructured log events

The following examples contain code snippets that show how you can use filter patterns to match terms in unstructured log events.

Note

Filter patterns are case sensitive. Enclose exact phrases and terms that include non-alphanumeric characters in double quotation marks ("").

Example: Match a single term

The following code snippet shows an example of a single-term filter pattern that returns all log events where messages contain the word **ERROR**.

```
ERROR
```

This filter pattern matches log event messages, such as the following:

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST
- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

Example: Match multiple terms

The following code snippet shows an example of a multiple-term filter pattern that returns all log events where messages contain the words **ERROR** and **ARGUMENTS**.

```
ERROR ARGUMENTS
```

The filter returns log event messages, such as the following:

- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

This filter pattern doesn't return the following log event messages because they don't contain both of the terms specified in the filter pattern.

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST

Example: Match optional terms

You can use pattern matching to create filter patterns that return log events containing optional terms. Place a question mark ("?") before the terms that you want to match. The following code snippet shows an example of a filter pattern that returns all log events where messages contain the word **ERROR** or the word **ARGUMENTS**.

```
?ERROR ?ARGUMENTS
```

This filter pattern matches log event messages, such as the following:

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST
- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

Note

You can't combine the question mark ("?",) with other filter patterns, such as include and exclude terms. If you combine "?" with other filter patterns, all question mark terms will be ignored.

For example, the following filter pattern matches all events containing the word **REQUEST**, but the question mark ("?",) filter terms are ignored and have no effect.

```
?ERROR ?ARGUMENTS REQUEST
```

Log event matches

- [INFO] REQUEST FAILED
- [WARN] UNAUTHORIZED REQUEST
- [ERROR] 400 BAD REQUEST

Example: Match exact phrases

The following code snippet shows an example of a filter pattern that returns log events where messages contain the exact phrase **INTERNAL SERVER ERROR**.

```
"INTERNAL SERVER ERROR"
```

This filter pattern returns the following log event message:

- [ERROR 500] INTERNAL SERVER ERROR

Example: Include and exclude terms

You can create filter patterns that return log events where messages include some terms and exclude other terms. Place a minus symbol ("-") before the terms that you want to exclude. The following code snippet shows an example of a filter pattern that returns log events where messages include the term **ERROR** and exclude the term **ARGUMENTS**.

```
ERROR -ARGUMENTS
```

This filter pattern returns log event messages, such as the following:

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST

This filter pattern doesn't return the following log event messages because they contain the word **ARGUMENTS**.

- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

Example: Match everything

You can match everything in your log events with double quotation marks. The following code snippet shows an example of a filter pattern that returns all log events.

```
" "
```

Using filter patterns to match terms in JSON log events

Writing filter patterns for JSON log events

The following describes how to write the syntax for filter patterns that match JSON terms containing strings and numeric values.

Writing filter patterns that match strings

You can create filter patterns to match strings in JSON log events. The following code snippet shows an example of the syntax for string-based filter pattern.


```
{ PropertySelector EqualityOperator String }
```

Enclose filter patterns in curly braces ("{}"). String-based filter patterns must contain the following parts:

- **Property selector**

Set off property selectors with a dollar sign followed by a period ("\$."). Property selectors are alphanumeric strings that support hyphen ("-") and underscore "_" characters. Strings don't support scientific notation. Property selectors point to value nodes in JSON log events. Value nodes can be strings or numbers. Place arrays after property selectors. The elements in arrays follow a zero-based numbering system, meaning that the first element in the array is element 0, the second element is element 1, and so on. Enclose elements in brackets ("[]"). If a property selector points to an array or object, the filter pattern won't match the log format. If the JSON property contains a period ("."), then the bracket notation may be used to select that property.

 Note

Wildcard selector

You can use the JSON wildcard to select any array element or any JSON object field.

Quotas

You can only use up to one wildcard selector in a property selector.

- **Equality operator**

Set off equality operators with one of the following symbols: equal ("=") or not equal ("!="). Equality operators return a Boolean value (true or false).

- **String**

You can enclose strings in double quotation marks (""). Strings that contain types other than alphanumeric characters and the underscore symbol must be placed in double quotation marks. Use the asterisk ("*") as a wild card to match text.

Note

You can use any conditional regular expression when creating filter patterns to match terms in JSON log events. For a list of supported regular expressions, see [Supported regular expressions](#).

The following code snippet contains an example of a filter pattern showing how you can format a filter pattern to match a JSON term with a string.

```
{ $.eventType = "UpdateTrail" }
```

Writing filter patterns that match numeric values

You can create filter patterns to match numeric values in JSON log events. The following code snippet shows an example of the syntax for filter patterns that match numeric values.

```
{ PropertySelector NumericOperator Number }
```

Enclose filter patterns in curly braces ("{}"). Filter patterns that match numeric values must have the following parts:

- **Property selector**

Set off property selectors with a dollar sign followed by a period ("\$."). Property selectors are alphanumeric strings that support hyphen ("-") and underscore "_" characters. Strings don't support scientific notation. Property selectors point to value nodes in JSON log events. Value nodes can be strings or numbers. Place arrays after property selectors. The elements in arrays follow a zero-based numbering system, meaning that the first element in the array is element 0, the second element is element 1, and so on. Enclose elements in brackets ("[]"). If a property selector points to an array or object, the filter pattern won't match the log format. If the JSON property contains a period ("."), then the bracket notation may be used to select that property.

Note**Wildcard selector**

You can use the JSON wildcard to select any array element or any JSON object field.

Quotas

You can only use up to one wildcard selector in a property selector.

- **Numeric operator**

Set off numeric operators with one of the following symbols: greater than ("**>**"), less than ("**<**"), equal ("**=**"), not equal ("**!=**"), greater than or equal to ("**>=**"), or less than or equal to ("**<=**").

- **Number**

You can use integers that contain plus ("**+**") or minus ("**-**") symbols and follow scientific notation. Use the asterisk ("*****") as a wild card to match numbers.

The following code snippet contains examples showing how you can format filter patterns to match JSON terms with numeric values.

```
// Filter pattern with greater than symbol
{ $.bandwidth > 75 }
// Filter pattern with less than symbol
{ $.latency < 50 }
// Filter pattern with greater than or equal to symbol
{ $.refreshRate >= 60 }
// Filter pattern with less than or equal to symbol
{ $.responseTime <= 5 }
// Filter pattern with equal sign
{ $.errorCode = 400 }
// Filter pattern with not equal sign
{ $.errorCode != 500 }
// Filter pattern with scientific notation and plus symbol
{ $.number[0] = 1e+3 }
// Filter pattern with scientific notation and minus symbol
{ $.number[0] != 1e-3 }
```

Match terms in JSON log events using simple expressions

The following examples contain code snippets that show how filter patterns can match terms in a JSON log event.

Note

If you test an example filter pattern with the example JSON log event, you must enter the example JSON log on a single line.

JSON log event

```
{
  "eventType": "UpdateTrail",
  "sourceIPAddress": "111.111.111.111",
  "arrayKey": [
    "value",
    "another value"
  ],
  "objectList": [
    {
      "name": "a",
      "id": 1
    },
    {
      "name": "b",
      "id": 2
    }
  ],
  "SomeObject": null,
  "cluster.name": "c"
}
```

Example: Filter pattern that matches string values

This filter pattern matches the string "UpdateTrail" in the property "eventType".

```
{ $.eventType = "UpdateTrail" }
```

Example: Filter pattern that matches string values (IP address)

This filter pattern contains a wild card and matches the property "sourceIPAddress" because it doesn't contain a number with the prefix "123.123. ".

```
{ $.sourceIPAddress != 123.123.* }
```

Example: Filter pattern that matches a specific array element with a string value

This filter pattern matches the element "value" in the array "arrayKey".

```
{ $.arrayKey[0] = "value" }
```

Example: Filter pattern that matches a string using regex

This filter pattern matches the string "Trail" in the property "eventType".

```
{ $.eventType = %Trail% }
```

Example: Filter pattern that uses a wildcard to match values of any element in the array using regex

The filter pattern contain regex which matches the element "value" in the array "arrayKey".

```
{ $.arrayKey[*] = %val.{2}% }
```

Example: Filter pattern that uses a wildcard to match values of any element with a specific prefix and subnet using regex (IP address)

This filter pattern contains regex which matches the element "111.111.111.111" in the property "sourceIPAddress".

```
{ $.* = %111\.111\.111\.1[0-9]{1,2}% }
```

Note**Quotas**

You can only use up to one wildcard selector in a property selector.

Example: Filter pattern that matches a JSON property with a period (.) in the key

```
{ $.['cluster.name'] = "c" }
```

Example: Filter pattern that matches JSON logs using IS

You can create filter patterns that match fields in JSON logs with the IS variable. The IS variable can match fields that contain the values NULL, TRUE, or FALSE. The following filter pattern returns JSON logs where the value of `SomeObject` is NULL.

```
{ $.SomeObject IS NULL }
```

Example: Filter pattern that matches JSON logs using NOT EXISTS

You can create filter patterns with the NOT EXISTS variable to return JSON logs that don't contain specific fields in the log data. The following filter pattern uses NOT EXISTS to return JSON logs that don't contain the field `SomeOtherObject`.

```
{ $.SomeOtherObject NOT EXISTS }
```

Note

The variables IS NOT and EXISTS currently aren't supported.

Match terms in JSON objects using compound expressions

You can use the logical operators AND ("&&") and OR ("||") in filter patterns to create compound expressions that match log events where two or more conditions are true. Compound expressions support the use of parentheses ("()") and the following standard order of operations: () > && > ||. The following examples contain code snippets that show how you can use filter patterns with compound expressions to match terms in a JSON object.

JSON object

```
{
  "user": {
    "id": 1,
    "email": "John.Stiles@example.com"
  },
  "users": [
    {
      "id": 2,
      "email": "John.Doe@example.com"
    },
    {
      "id": 3,
      "email": "Jane.Doe@example.com"
    }
  ],
  "actions": [
    "GET",
    "PUT",
    "DELETE"
  ],
  "coordinates": [
    [0, 1, 2],
    [4, 5, 6],
    [7, 8, 9]
  ]
}
```

Example: Expression that matches using AND (&&)

This filter pattern contains a compound expression that matches "id" in "user" with a numeric value of 1 and "email" in the first element of the "users" array with the string "John.Doe@example.com".

```
{ ($.user.id = 1) && ($.users[0].email = "John.Doe@example.com") }
```

Example: Expression that matches using OR (||)

This filter pattern contains a compound expression that matches "email" in "user" with the string "John.Stiles@example.com".

```
{ $.user.email = "John.Stiles@example.com" || $.coordinates[0][1] = "nonmatch" && $.actions[2] = "nonmatch" }
```

Example: Expression that doesn't match using AND (&&)

This filter pattern contains a compound expression that doesn't find a match because the expression doesn't match the third action in "actions".

```
{ ($.user.email = "John.Stiles@example.com" || $.coordinates[0][1] = "nonmatch") && $.actions[2] = "nonmatch" }
```

Note

Quotas

You can only use up to one wildcard selector in a property selector, and up to three wildcard selectors in a filter pattern with compound expressions.

Example: Expression that doesn't match using OR (||)

This filter pattern contains a compound expression that doesn't find a match because the expression doesn't match the first property in "users" or the third action in "actions".


```
{ ($.user.id = 2 && $.users[0].email = "nonmatch") || $.actions[2] = "GET" }
```

Using filter patterns to match terms in space-delimited log events

Writing filter patterns for space-delimited log events

You can create filter patterns to match terms in space-delimited log events. The following provides an example space-delimited log event and describes how to write the syntax for filter patterns that match terms in the space-delimited log event.

Note

You can use any conditional regular expression when creating filter patterns to match terms in space-delimited log events. For a list of supported regular expressions, see [Supported regular expressions](#).

Example: Space-delimited log event

The following code snippet shows a space-delimited log event that contains seven fields: ip, user, username, timestamp, request, status_code, and bytes.

```
127.0.0.1 Prod frank [10/Oct/2000:13:25:15 -0700] "GET /index.html HTTP/1.0" 404  
1534
```

Note

Characters between brackets ("[]") and double quotation marks (""") are considered single fields.

Writing filter patterns that match terms in a space-delimited log event

To create a filter pattern that matches terms in a space-delimited log event, enclose the filter pattern in brackets ("[]"), and specify fields with names that are separated by commas (","),. The following filter pattern parses seven fields.

```
[ip=%127\.\0\.\0\.[1-9]%, user, username, timestamp, request =*.html*, status_code = 4*, bytes]
```

You can use numeric operators (>, <, =, !=, >=, or <=) and the asterisk (*) as a wild card or regex to give your filter pattern conditions. In the example filter pattern, `ip` uses regex that matches IP address range 127.0.0.1 - 127.0.0.9, `request` contains a wildcard that states it must extract a value with `.html`, and `status_code` contains a wildcard that states it must extract a value beginning with 4.

If you don't know the number of fields that you're parsing in a space-delimited log event, you can use ellipsis (...) to reference any unnamed field. Ellipsis can reference as many fields as needed. The following example shows a filter pattern with ellipsis that represent the first four unnamed fields shown in the previous example filter pattern.

```
[..., request =*.html*, status_code = 4*, bytes]
```

You also can use the logical operators AND (&&) and OR (||) to create compound expressions. The following filter pattern contains a compound expression that states the value of `status_code` must be 404 or 410.

```
[ip, user, username, timestamp, request =*.html*, status_code = 404 || status_code = 410, bytes]
```

Match terms in space-delimited log events using pattern matching

You can use pattern matching to create space-delimited filter patterns that match terms in a specific order. Specify the order of your terms with indicators. Use **w1** to represent your first term and **w2** and so on to represent the order of your subsequent terms. Place commas (",") between your terms. The following examples contain code snippets that show how you can use pattern matching with space-delimited filter patterns.

Note

You can use any conditional regular expression when creating filter patterns to match terms in space-delimited log events. For a list of supported regular expressions, see [Supported regular expressions](#).

Space-delimited log event

```
INFO 09/25/2014 12:00:00 GET /service/resource/67 1200
INFO 09/25/2014 12:00:01 POST /service/resource/67/part/111 1310
WARNING 09/25/2014 12:00:02 Invalid user request
ERROR 09/25/2014 12:00:02 Failed to process request
```

Example: Match terms in order

The following space-delimited filter pattern returns log events where the first word in the log events is **ERROR**.

```
[w1=ERROR, w2]
```

Note

When you create space-delimited filter patterns that use pattern matching, you must include a blank indicator after you specify the order of your terms. For example, if you create a filter pattern that returns log events where the first word is **ERROR**, include a blank **w2** indicator after the **w1** term.

Example: Match terms with AND (&&) and OR (||)

You can use the logical operators AND ("&&") and OR ("||") to create space-delimited filter patterns that contain conditions. The following filter pattern returns log events where the first word in the events is **ERROR** or **WARNING**.

```
[w1=ERROR || w1=WARNING, w2]
```

Example: Exclude terms from matches

You can create space-delimited filter patterns that return log events excluding one or more terms. Place a not equal symbol ("!=") before the term or terms that you want to exclude. The following code snippet shows an example of a filter pattern that returns log events where the first words aren't **ERROR** and **WARNING**.

```
[w1!=ERROR && w1!=WARNING, w2]
```

Example: Match the top level item in a resource URI

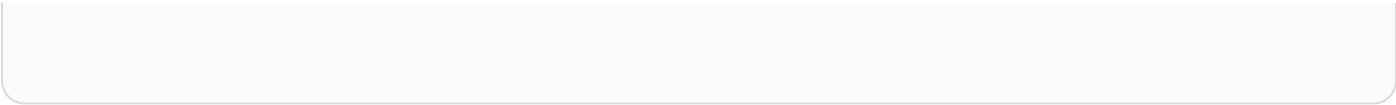
The following code snippet shows an example of a filter pattern that matches the top level item in a resource URI using regex.

```
[logLevel, date, time, method, url=%/service/resource/[0-9]+$, response_time]
```

Example: Match the child level item in a resource URI

The following code snippet shows an example of a filter pattern that matches the child level item in a resource URI using regex.

```
[logLevel, date, time, method, url=%/service/resource/[0-9]+/part/[0-9]+$,  
response_time]
```



Enable logging from AWS services

While many services publish logs only to CloudWatch Logs, some AWS services can publish logs directly to Amazon Simple Storage Service or Amazon Data Firehose. If your main requirement for logs is storage or processing in one of these services, you can easily have the service that produces the logs send them directly to Amazon S3 or Firehose without additional setup.

Even when logs are published directly to Amazon S3 or Firehose, charges apply. For more information, see *Vended Logs* on the **Logs** tab at [Amazon CloudWatch Pricing](#).

Some AWS services use a common infrastructure to send their logs. To enable logging from these services, you must be logged in as a user that has certain permissions. Additionally, you must grant permissions to AWS to enable the logs to be sent.

For services that require these permissions, there are two versions of the permissions needed. The services that require these extra permissions are noted as **Supported [V1 Permissions]** and **Supported [V2 Permissions]** in the table. For information about these required permissions, see the sections after the table.

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon API Gateway access logs	Vended logs	Supported [V1 Permissions]			
AWS AppSync logs	Custom logs	Supported			
Amazon Aurora MySQL logs	Custom logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon Bedrock Knowledge bases logging	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Bedrock Agent logging	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Bedrock AgentCore Runtime	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]
Amazon Bedrock AgentCore Gateway	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]
Amazon Bedrock AgentCore Identity	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]
Amazon Bedrock AgentCore Tools	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon Chime media quality metric logs and SIP message logs	Vended logs	Supported [V1 Permissions]			
CloudFront: access logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
AWS CloudHSM audit logs	Custom logs	Supported			
CloudWatch Evidently evaluation event logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]		
CloudWatch Internet Monitor logs	Vended logs		Supported [V1 Permissions]		
CloudTrail logs	Custom logs	Supported			
AWS CodeBuild logs	Custom logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon CodeWhisperer event logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Cognito logs	Vended logs	Supported [V1 Permissions]			
Amazon Connect logs	Custom logs	Supported			
AWS DataSync logs	Custom logs	Supported			
Amazon ElastiCache (Redis OSS) logs	Vended logs	Supported [V1 Permissions]		Supported [V1 Permissions]	
AWS Elastic Beanstalk logs	Custom logs	Supported			
Amazon Elastic Container Service logs	Custom logs	Supported			
Amazon Elastic Kubernetes Service control plane logs	Vended logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
AWS Elemental MediaPackage access logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
AWS Elemental MediaTailor logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
AWS Entity Resolution logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon EventBridge Pipes logging	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
Amazon EventBridge event buses	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
AWS Fargate logs	Custom logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
AWS Fault Injection Service experiment logs	Vended logs		Supported [V1 Permissions]		
Amazon FinSpace	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
AWS Global Accelerator flow logs	Vended logs		Supported [V1 Permissions]		
AWS Glue job logs	Custom logs	Supported			
IAM Identity Center error logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Interactive Video Service chat logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
AWS IoT logs	Custom logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
AWS IoT FleetWise logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
AWS Lambda logs	Vended logs	Supported	Supported	Supported	
Amazon Macie logs	Custom logs	Supported			
Amazon SES logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
AWS Mainframe Modernization	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
Amazon Managed Service for Prometheus logs	Vended logs	Supported [V1 Permissions]			
Amazon MSK broker logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon MSK Connect logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
Amazon MQ logs	Custom logs	Supported			
AWS Network Firewall logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
AWS Network Firewall Proxy logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Network Load Balancer access logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
OpenSearch logs	Custom logs	Supported			
Amazon OpenSearch Service ingestion logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
AWS PCS logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Q Business connector logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Q Business conversation logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Quick Suite chat and feedback logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Relational Database Service PostgreSQL logs	Custom logs	Supported			
AWS RTB Fabric logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Route 53 public DNS query logs	Vended logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon Route 53 resolver query logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]		
Amazon SageMaker AI events	Vended logs	Supported [V1 Permissions]			
Amazon SageMaker AI worker events	Vended logs	Supported [V1 Permissions]			
AWS Site-to-Site VPN logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
Amazon Simple Email Service logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
Amazon Simple Notification Service logs	Custom logs	Supported			

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon Simple Notification Service data protection policy logs	Custom logs	Supported			
EC2 Spot Instance data feed files	Vended logs		Supported [V1 Permissions]		
AWS Step Functions Express Workflow and Standard Workflow logs	Vended logs	Supported [V1 Permissions]			
Storage Gateway audit logs and health logs	Vended logs	Supported [V1 Permissions]			
AWS Transfer Family logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
AWS Verified Access logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	

Log source	Log type	the section called “Logs sent to CloudWatch Logs”	the section called “Logs sent to Amazon S3”	the section called “Logs sent to Firehose”	the section called “Traces sent to X-Ray”
Amazon Virtual Private Cloud flow logs	Vended logs	Supported	Supported [V1 Permissions]	Supported [V1 Permissions]	
Amazon VPC Lattice access logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported [V1 Permissions]	
Amazon VPC Route Server	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	
AWS WAF logs	Vended logs	Supported [V1 Permissions]	Supported [V1 Permissions]	Supported	
Amazon WorkMail audit logs	Vended logs	Supported [V2 Permissions]	Supported [V2 Permissions]	Supported [V2 Permissions]	

Logging that requires additional permissions [V1]

Some AWS services use a common infrastructure to send their logs to CloudWatch Logs, Amazon S3, or Firehose. To enable the AWS services listed in the preceding table to send their logs to these destinations, you must be logged in as a user that has certain permissions.

Additionally, permissions must be granted to AWS to enable the logs to be sent. AWS can automatically create those permissions when the logs are set up, or you can create them yourself first before you set up the logging. For cross-account delivery, you must manually create the permission policies yourself.

If you choose to have AWS automatically set up the necessary permissions and resource policies when you or someone in your organization first sets up the sending of logs, then the user who is setting up the sending of logs must have certain permissions, as explained later in this section. Alternatively, you can create the resource policies yourself, and then the users who set up the sending of logs do not need as many permissions.

The following topics provide more details for each of these destinations.

Topics

- [Logs sent to CloudWatch Logs](#)
- [Logs sent to Amazon S3](#)
- [Logs sent to Firehose](#)

Logs sent to CloudWatch Logs

Important

When you set up the log types in the following list to be sent to CloudWatch Logs, AWS creates or changes the resource policies associated with the log group receiving the logs, if needed. Continue reading this section to see the details.

This section applies when the types of logs listed in the table in the preceding section are sent to CloudWatch Logs:

User permissions

To be able to set up sending any of these types of logs to CloudWatch Logs for the first time, you must be logged into an account with the following permissions.

- `logs:CreateLogDelivery`
- `logs:PutResourcePolicy`
- `logs:DescribeResourcePolicies`
- `logs:DescribeLogGroups`

 Note

When you specify the `logs:DescribeLogGroups`, `logs:DescribeResourcePolicies`, or `logs:PutResourcePolicy` permission, be sure to set the ARN of its Resource line to use a `*` wildcard, instead of specifying only a single log group name. For example, `"Resource": "arn:aws:logs:us-east-1:111122223333:log-group:*"`

If any of these types of logs is already being sent to a log group in CloudWatch Logs, then to set up the sending of another one of these types of logs to that same log group, you only need the `logs:CreateLogDelivery` permission.

Log group resource policy

The log group where the logs are being sent must have a resource policy that includes certain permissions. If the log group currently does not have a resource policy, and the user setting up the logging has the `logs:PutResourcePolicy`, `logs:DescribeResourcePolicies`, and `logs:DescribeLogGroups` permissions for the log group, then AWS automatically creates the following policy for it when you begin sending the logs to CloudWatch Logs.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite20150319",
      "Effect": "Allow",
      "Principal": {
```

```

        "Service": [
            "delivery.logs.amazonaws.com"
        ],
    },
    "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group:log-
stream:*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": [
                "0123456789"
            ]
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:logs:us-east-1:111122223333:*"
            ]
        }
    }
}

```

If the log group does have a resource policy but that policy doesn't contain the statement shown in the previous policy, and the user setting up the logging has the `logs:PutResourcePolicy`, `logs:DescribeResourcePolicies`, and `logs:DescribeLogGroups` permissions for the log group, that statement is appended to the log group's resource policy.

Logs sent to Amazon S3

When you set logs to be sent to Amazon S3, AWS creates or changes the resource policies associated with the S3 bucket that is receiving the logs, if needed.

Logs published directly to Amazon S3 are published to an existing bucket that you specify. One or more log files are created every five minutes in the specified bucket.

When you deliver logs for the first time to an Amazon S3 bucket, the service that delivers logs records the owner of the bucket to ensure that the logs are delivered only to a bucket belonging to this account. As a result, to change the Amazon S3 bucket owner, you must re-create or update the log subscription in the originating service.

Note

CloudFront uses a different permissions model than the other services that send vended logs to S3. For more information, see [Permissions required to configure standard logging and to access your log files](#).

Additionally, if you use the same S3 bucket for CloudFront access logs and another log source, enabling ACL on the bucket for CloudFront also grants permission to all other log sources that use this bucket.

Important

If you're sending logs to an Amazon S3 bucket and the bucket policy contains a `NotAction` or `NotPrincipal` element, adding log delivery permissions to the bucket automatically and creating a log subscription will fail. To create a log subscription successfully, you need to manually add the log delivery permissions to the bucket policy, then create the log subscription. For more information, see the instructions in this section.

If the bucket has server-side encryption using a customer managed AWS KMS key, you must also add the key policy for your customer managed key. For more information, see [Amazon S3](#).

If the destination bucket has SSE-KMS and a Bucket Key enabled, the attached customer managed KMS key policy no longer works as expected for all requests. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys](#).

If you're using vended logs and S3 encryption with a customer managed AWS KMS key, you must use a fully qualified AWS KMS key ARN instead of a key ID when you configure the bucket. For more information, see [put-bucket-encryption](#).

User permissions

To be able to set up sending any of these types of logs to Amazon S3 for the first time, you must be logged into an account with the following permissions.

- logs:CreateLogDelivery
- S3:GetBucketPolicy
- S3:PutBucketPolicy

If any of these types of logs is already being sent to an Amazon S3 bucket, then to set up the sending of another one of these types of logs to the same bucket you only need to have the logs:CreateLogDelivery permission.

S3 bucket resource policy

The S3 bucket where the logs are being sent must have a resource policy that includes certain permissions. If the bucket currently does not have a resource policy and the user setting up the logging has the S3:GetBucketPolicy and S3:PutBucketPolicy permissions for the bucket, then AWS automatically creates the following policy for it when you begin sending the logs to Amazon S3.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "AWSLogDeliveryWrite20150319",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryAclCheck",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "s3:GetBucketAcl",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "0123456789"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:111122223333:*"
          ]
        }
      }
    }
  ]
}
```

```

        }
    },
    {
        "Sid": "AWSLogDeliveryWrite",
        "Effect": "Allow",
        "Principal": {
            "Service": "delivery.logs.amazonaws.com"
        },
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/AWSLogs/account-ID/*",
        "Condition": {
            "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control",
                "aws:SourceAccount": [
                    "0123456789"
                ]
            },
            "ArnLike": {
                "aws:SourceArn": [
                    "arn:aws:logs:us-east-1:111122223333:"
                ]
            }
        }
    }
]
}

```

In the previous policy, for `aws:SourceAccount`, specify the list of account IDs for which logs are being delivered to this bucket. For `aws:SourceArn`, specify the list of ARNs of the resource that generates the logs, in the form `arn:aws:logs:source-region:source-account-id:*`.

If the bucket has a resource policy but that policy doesn't contain the statement shown in the previous policy, and the user setting up the logging has the `S3:GetBucketPolicy` and `S3:PutBucketPolicy` permissions for the bucket, that statement is appended to the bucket's resource policy.

Note

In some cases, you may see `AccessDenied` errors in AWS CloudTrail if the `s3:ListBucket` permission has not been granted to `delivery.logs.amazonaws.com`.

To avoid these errors in your CloudTrail logs, you must grant the `s3:ListBucket` permission to `delivery.logs.amazonaws.com` and you must include the `Condition` parameters shown with the `s3:GetBucketAcl` permission set in the preceding bucket policy. To make this simpler, instead of creating a new Statement, you can directly update the `AWSLogDeliveryAclCheck` to be `"Action": ["s3:GetBucketAcl", "s3:ListBucket"]`

Amazon S3 bucket server-side encryption

You can protect the data in your Amazon S3 bucket by enabling either server-side Encryption with Amazon S3-managed keys (SSE-S3) or server-side encryption with a AWS KMS key stored in AWS Key Management Service (SSE-KMS). For more information, see [Protecting data using server-side encryption](#).

If you choose SSE-S3, no additional configuration is required. Amazon S3 handles the encryption key.

Warning

If you choose SSE-KMS, you must use a customer managed key, because using an AWS managed key is not supported for this scenario. If you set up encryption using an AWS managed key, the logs will be delivered in an unreadable format.

When you use a customer managed AWS KMS key, you can specify the Amazon Resource Name (ARN) of the customer managed key when you enable bucket encryption. You must add the following to the key policy for your customer managed key (not to the bucket policy for your S3 bucket), so that the log delivery account can write to your S3 bucket.

If you choose SSE-KMS, you must use a customer managed key, because using an AWS managed key is not supported for this scenario. When you use a customer managed AWS KMS key, you can specify the Amazon Resource Name (ARN) of the customer managed key when you enable bucket encryption. You must add the following to the key policy for your customer managed key (not to the bucket policy for your S3 bucket), so that the log delivery account can write to your S3 bucket.

```
{  
  "Sid": "Allow Logs Delivery to use the key",
```



```
"Effect": "Allow",
"Principal": {
  "Service": [ "delivery.logs.amazonaws.com" ]
},
"Action": [
  "kms:Encrypt",
  "kms:Decrypt",
  "kms:ReEncrypt*",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
],
"Resource": "*",
"Condition": {
  "StringEquals": {
    "aws:SourceAccount": ["0123456789"]
  },
  "ArnLike": {
    "aws:SourceArn": ["arn:aws:logs:us-east-1:0123456789:*"]
  }
}
```

For `aws:SourceAccount`, specify the list of account IDs for which logs are being delivered to this bucket. For `aws:SourceArn`, specify the list of ARNs of the resource that generates the logs, in the form `arn:aws:logs:source-region:source-account-id:*`.

Logs sent to Firehose

This section applies when the types of logs listed in the table in the preceding section are sent to Firehose:

User permissions

To be able to set up sending any of these types of logs to Firehose for the first time, you must be logged into an account with the following permissions.

- `logs:CreateLogDelivery`
- `firehose:TagDeliveryStream`
- `iam:CreateServiceLinkedRole`

If any of these types of logs is already being sent to Firehose, then to set up the sending of another one of these types of logs to Firehose you need to have only the `logs:CreateLogDelivery` and `firehose:TagDeliveryStream` permissions.

IAM roles used for permissions

Because Firehose does not use resource policies, AWS uses IAM roles when setting up these logs to be sent to Firehose. AWS creates a service-linked role named **AWSServiceRoleForLogDelivery**. This service-linked role includes the following permissions.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch",
        "firehose:ListTagsForDeliveryStream"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/LogDeliveryEnabled": "true"
        }
      },
      "Effect": "Allow"
    }
  ]
}
```

This service-linked role grants permission for all Firehose delivery streams that have the `LogDeliveryEnabled` tag set to `true`. AWS gives this tag to the destination delivery stream when you set up the logging.

This service-linked role also has a trust policy that allows the `delivery.logs.amazonaws.com` service principal to assume the needed service-linked role. That trust policy is as follows:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Logging that requires additional permissions [V2]

Some AWS services use a new method to send their logs. This is a flexible method that enables you to set up log delivery from these services to one or more of the following destinations: CloudWatch Logs, Amazon S3, or Firehose and X-Ray for trace delivery.

A working log delivery consists of three elements:

- A `DeliverySource`, which is a logical object that represents the resource(s) that actually send the logs.
- A `DeliveryDestination`, which is a logical object that represents the actual delivery destination.
- A `Delivery`, which connects a delivery source to delivery destination

To configure logs delivery between a supported AWS service and a destination, you must do the following:

- Create a delivery source with [PutDeliverySource](#).
- Create a delivery destination with [PutDeliveryDestination](#).
- If you are delivering logs cross-account, you must use [PutDeliveryDestinationPolicy](#) in the destination account to assign an IAM policy to the destination. This policy authorizes creating a

delivery from the delivery source in account A to the delivery destination in account B. For cross-account delivery, you must manually create the permission policies yourself.

- Create a delivery by pairing exactly one delivery source and one delivery destination, by using [CreateDelivery](#).

The following sections provide the details of the permissions you need to have when you are signed in to set up log delivery to each type of destination, using the V2 process. These permissions can be granted to an IAM role that you are signed in with.

Important

It is your responsibility to remove log delivery resources after deleting the log-generating resource. To do so, follow these steps.

1. Delete the Delivery by using the [DeleteDelivery](#) operation.
2. Delete the DeliverySource by using the [DeleteDeliverySource](#) operation.
3. If the DeliveryDestination associated with the DeliverySource that you just deleted is used only for this specific DeliverySource, then you can remove it by using the [DeleteDeliveryDestinations](#) operation.

Contents

- [Logs sent to CloudWatch Logs](#)
- [Logs sent to Amazon S3](#)
 - [Amazon S3 bucket server-side encryption](#)
- [Logs sent to Firehose](#)
- [Traces sent to X-Ray](#)

Logs sent to CloudWatch Logs

User permissions

To enable sending logs to CloudWatch Logs, you must be signed in with the following permissions.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadWriteAccessForLogDeliveryActions",
      "Effect": "Allow",
      "Action": [
        "logs:GetDelivery",
        "logs:GetDeliverySource",
        "logs:PutDeliveryDestination",
        "logs:GetDeliveryDestinationPolicy",
        "logs>DeleteDeliverySource",
        "logs:PutDeliveryDestinationPolicy",
        "logs>CreateDelivery",
        "logs:GetDeliveryDestination",
        "logs:PutDeliverySource",
        "logs>DeleteDeliveryDestination",
        "logs>DeleteDeliveryDestinationPolicy",
        "logs>DeleteDelivery",
        "logs:UpdateDeliveryConfiguration"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:delivery:*",
        "arn:aws:logs:us-east-1:444455556666:delivery-source:*",
        "arn:aws:logs:us-east-1:777788889999:delivery-destination:*"
      ]
    },
    {
      "Sid": "ListAccessForLogDeliveryActions",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeDeliveryDestinations",
        "logs:DescribeDeliverySources",
        "logs:DescribeDeliveries",
        "logs:DescribeConfigurationTemplates"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowUpdatesToResourcePolicyCWL",
      "Effect": "Allow",

```

```

        "Action": [
            "logs:PutResourcePolicy",
            "logs:DescribeResourcePolicies",
            "logs:DescribeLogGroups"
        ],
        "Resource": [
            "arn:aws:logs:us-east-1:123456789012:*"
        ]
    }
]
}

```

Log group resource policy

The log group where the logs are being sent must have a resource policy that includes certain permissions. If the log group currently does not have a resource policy, and the user setting up the logging has the `logs:PutResourcePolicy`, `logs:DescribeResourcePolicies`, and `logs:DescribeLogGroups` permissions for the log group, then AWS automatically creates the following policy for it when you begin sending the logs to CloudWatch Logs.

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryWrite20150319",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "delivery.logs.amazonaws.com"
                ]
            },
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group:log-  
stream:*"
            ]
        }
    ]
}

```

```

        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": [
                    "0123456789"
                ]
            },
            "ArnLike": {
                "aws:SourceArn": [
                    "arn:aws:logs:us-east-1:111122223333:*"
                ]
            }
        }
    }
}

```

Logs sent to Amazon S3

User permissions

To enable sending logs to Amazon S3, you must be signed in with the following permissions.

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadWriteAccessForLogDeliveryActions",
            "Effect": "Allow",
            "Action": [
                "logs:GetDelivery",
                "logs:GetDeliverySource",
                "logs:PutDeliveryDestination",
                "logs:GetDeliveryDestinationPolicy",
                "logs>DeleteDeliverySource",
                "logs:PutDeliveryDestinationPolicy",
                "logs>CreateDelivery",
                "logs:GetDeliveryDestination",
                "logs:PutDeliverySource",
                "logs>DeleteDeliveryDestination",
                "logs>DeleteDeliveryDestinationPolicy",
            ]
        }
    ]
}

```

```

        "logs:DeleteDelivery",
        "logs:UpdateDeliveryConfiguration"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:111122223333:delivery:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-source:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    ],
    },
    {
        "Sid": "ListAccessForLogDeliveryActions",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeDeliveryDestinations",
            "logs:DescribeDeliverySources",
            "logs:DescribeDeliveries",
            "logs:DescribeConfigurationTemplates"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AllowUpdatesToResourcePolicyS3",
        "Effect": "Allow",
        "Action": [
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy"
        ],
        "Resource": "arn:aws:s3:::bucket-name"
    }
]
}

```

The S3 bucket where the logs are being sent must have a resource policy that includes certain permissions. If the bucket currently does not have a resource policy and the user setting up the logging has the `S3:GetBucketPolicy` and `S3:PutBucketPolicy` permissions for the bucket, then AWS automatically creates the following policy for it when you begin sending the logs to Amazon S3.

JSON

```
{
```



```

"Version": "2012-10-17",
"Id": "AWSLogDeliveryWrite20150319",
"Statement": [
  {
    "Sid": "AWSLogDeliveryWrite",
    "Effect": "Allow",
    "Principal": {
      "Service": "delivery.logs.amazonaws.com"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/AWSLogs/account-ID/*",
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control",
        "aws:SourceAccount": [
          "0123456789"
        ]
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:111122223333:delivery-source:*"
        ]
      }
    }
  }
]
}

```

In the previous policy, for `aws:SourceAccount`, specify the list of account IDs for which logs are being delivered to this bucket. For `aws:SourceArn`, specify the list of ARNs of the resource that generates the logs, in the form `arn:aws:logs:source-region:source-account-id:*`.

If the bucket has a resource policy but that policy doesn't contain the statement shown in the previous policy, and the user setting up the logging has the `S3:GetBucketPolicy` and `S3:PutBucketPolicy` permissions for the bucket, that statement is appended to the bucket's resource policy.

Note

In some cases, you may see `AccessDenied` errors in AWS CloudTrail if the `s3:ListBucket` permission has not been granted to `delivery.logs.amazonaws.com`.

To avoid these errors in your CloudTrail logs, you must grant the `s3:ListBucket` permission to `delivery.logs.amazonaws.com` and you must include the `Condition` parameters shown with the `s3:GetBucketAcl` permission set in the preceding bucket policy. To make this simpler, instead of creating a new Statement, you can directly update the `AWSLogDeliveryAclCheck` to be `"Action": ["s3:GetBucketAcl", "s3:ListBucket"]`

Amazon S3 bucket server-side encryption

You can protect the data in your Amazon S3 bucket by enabling either server-side Encryption with Amazon S3-managed keys (SSE-S3) or server-side encryption with a AWS KMS key stored in AWS Key Management Service (SSE-KMS). For more information, see [Protecting data using server-side encryption](#).

If you choose SSE-S3, no additional configuration is required. Amazon S3 handles the encryption key.

Warning

If you choose SSE-KMS, you must use a customer managed key, because using an AWS managed key is not supported for this scenario. If you set up encryption using an AWS managed key, the logs will be delivered in an unreadable format.

When you use a customer managed AWS KMS key, you can specify the Amazon Resource Name (ARN) of the customer managed key when you enable bucket encryption. You must add the following to the key policy for your customer managed key (not to the bucket policy for your S3 bucket), so that the log delivery account can write to your S3 bucket.

If you choose SSE-KMS, you must use a customer managed key, because using an AWS managed key is not supported for this scenario. When you use a customer managed AWS KMS key, you can specify the Amazon Resource Name (ARN) of the customer managed key when you enable bucket encryption. You must add the following to the key policy for your customer managed key (not to the bucket policy for your S3 bucket), so that the log delivery account can write to your S3 bucket.

```
{  
  "Sid": "Allow Logs Delivery to use the key",  
  "Effect": "Allow",
```

```

"Principal": {
  "Service": [ "delivery.logs.amazonaws.com" ]
},
"Action": [
  "kms:Encrypt",
  "kms:Decrypt",
  "kms:ReEncrypt*",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
],
"Resource": "*",
"Condition": {
  "StringEquals": {
    "aws:SourceAccount": ["0123456789"]
  },
  "ArnLike": {
    "aws:SourceArn": ["arn:aws:logs:us-east-1:0123456789:delivery-source:*"]
  }
}
}

```

For `aws:SourceAccount`, specify the list of account IDs for which logs are being delivered to this bucket. For `aws:SourceArn`, specify the list of ARNs of the resource that generates the logs, in the form `arn:aws:logs:source-region:source-account-id:*`.

Logs sent to Firehose

User permissions

To enable sending logs to Firehose, you must be signed in with the following permissions.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadWriteAccessForLogDeliveryActions",
      "Effect": "Allow",
      "Action": [
        "logs:GetDelivery",
        "logs:GetDeliverySource",

```

```

        "logs:PutDeliveryDestination",
        "logs:GetDeliveryDestinationPolicy",
        "logs>DeleteDeliverySource",
        "logs:PutDeliveryDestinationPolicy",
        "logs>CreateDelivery",
        "logs:GetDeliveryDestination",
        "logs:PutDeliverySource",
        "logs>DeleteDeliveryDestination",
        "logs>DeleteDeliveryDestinationPolicy",
        "logs>DeleteDelivery",
        "logs:UpdateDeliveryConfiguration"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:111122223333:delivery:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-source:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    ]
},
{
    "Sid": "ListAccessForLogDeliveryActions",
    "Effect": "Allow",
    "Action": [
        "logs:DescribeDeliveryDestinations",
        "logs:DescribeDeliverySources",
        "logs:DescribeDeliveries",
        "logs:DescribeConfigurationTemplates"
    ],
    "Resource": "*"
},
{
    "Sid": "AllowUpdatesToResourcePolicyFH",
    "Effect": "Allow",
    "Action": [
        "firehose:TagDeliveryStream"
    ],
    "Resource": [
        "arn:aws:firehose:us-east-1:111122223333:deliverystream/*"
    ]
},
{
    "Sid": "CreateServiceLinkedRole",
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ]
}

```

```

    ],
    "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
delivery.logs.amazonaws.com/AWSServiceRoleForLogDelivery"
  }
]
}

```

IAM roles used for resource permissions

Because Firehose does not use resource policies, AWS uses IAM roles when setting up these logs to be sent to Firehose. AWS creates a service-linked role named **AWSServiceRoleForLogDelivery**. This service-linked role includes the following permissions.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch",
        "firehose:ListTagsForDeliveryStream"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/LogDeliveryEnabled": "true"
        }
      },
      "Effect": "Allow"
    }
  ]
}

```

This service-linked role grants permission for all Firehose delivery streams that have the `LogDeliveryEnabled` tag set to `true`. AWS gives this tag to the destination delivery stream when you set up the logging.

This service-linked role also has a trust policy that allows the `delivery.logs.amazonaws.com` service principal to assume the needed service-linked role. That trust policy is as follows:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Traces sent to X-Ray

User permissions

To enable sending traces to AWS X-Ray, you must be signed in with the following permissions.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadWriteAccessForLogDeliveryActions",
      "Effect": "Allow",
      "Action": [
        "logs:GetDelivery",
        "logs:GetDeliverySource",
        "logs:PutDeliveryDestination",
        "logs:GetDeliveryDestinationPolicy",
        "logs>DeleteDeliverySource",
        "logs:PutDeliveryDestinationPolicy",
        "logs>CreateDelivery",

```

```

        "logs:GetDeliveryDestination",
        "logs:PutDeliverySource",
        "logs:DeleteDeliveryDestination",
        "logs:DeleteDeliveryDestinationPolicy",
        "logs:DeleteDelivery",
        "logs:UpdateDeliveryConfiguration"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:111122223333:delivery:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-source:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    ]
},
{
    "Sid": "ListAccessForLogDeliveryActions",
    "Effect": "Allow",
    "Action": [
        "logs:DescribeDeliveryDestinations",
        "logs:DescribeDeliverySources",
        "logs:DescribeDeliveries",
        "logs:DescribeConfigurationTemplates"
    ],
    "Resource": "*"
},
{
    "Sid": "AllowUpdatesToResourcePolicyXRay",
    "Effect": "Allow",
    "Action": [
        "xray:PutResourcePolicy",
        "xray:ListResourcePolicies"
    ],
    "Resource": "*"
}
]
}

```

X-Ray resource policy

The destination account where the traces are being sent must have a resource policy that includes certain permissions. When X-Ray does not currently have a resource policy, and the user setting up the tracing has `xray:PutResourcePolicy` and `xray:ListResourcePolicies` permissions in

the account, AWS will automatically create the following policy when you begin sending traces to X-Ray.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "xray:PutTraceSegments",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ForAllValues:ArnLike": {
          "logs:LogGeneratingResourceArns": "arn:aws:bedrock-agentcore:us-east-1:123456789012:memory/MemoryId"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:logs:us-east-1:123456789012:delivery-source:xray-test"
        }
      }
    }
  ]
}
```

Enable transaction search

To enable sending traces to X-Ray, you must enable [transaction search](#).

Service-specific permissions

In addition to the destination-specific permissions listed in the previous sections, some services require explicit authorization that customers are allowed to send logs from their resources, as an

additional layer of security. It authorizes the `AllowVendedLogDeliveryForResource` action for resources that vend logs within that service. For these services, use the following policy and replace *service* and *resource-type* with the appropriate values. For the service-specific values for these fields, see those services' documentation page for vended logs. In the following example, the policy has been updated to enable vended logs from Amazon SES.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ServiceLevelAccessForLogDelivery",
      "Effect": "Allow",
      "Action": [
        "ses:AllowVendedLogDeliveryForResource"
      ],
      "Resource": "arn:aws:ses:us-east-1:123456789012:resource-type/*"
    }
  ]
}
```

Console-specific permissions

In addition to the permissions listed in the previous sections, if you are setting up log delivery using the console instead of the APIs, you also need the following additional permissions:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLogDeliveryActionsConsoleCWL",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups",
        "logs:CreateLogGroup"
      ],
    }
  ]
}
```

```

        "Resource": [
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ],
    },
    {
        "Sid": "AllowLogDeliveryActionsConsoleS3",
        "Effect": "Allow",
        "Action": [
            "s3:ListAllMyBuckets",
            "s3:ListBucket",
            "s3:GetBucketLocation"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Sid": "AllowLogDeliveryActionsConsoleFH",
        "Effect": "Allow",
        "Action": [
            "firehose:ListDeliveryStreams",
            "firehose:DescribeDeliveryStream"
        ],
        "Resource": [
            "*"
        ]
    }
}

```

Cross-account delivery example

In this example, two accounts are involved. The account with the log-generating resource is Account A, ID: **123456789012**, and the account with the log-consuming resource is Account B, ID: **111122223333**.

Account A wants to deliver logs from the Amazon Bedrock knowledge base in their account with the ARN `arn:aws:bedrock:us-east-1:123456789012:knowledge-base/kb-12345678`.

For this example, account A needs the following permissions:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowVendedLogDeliveryForKnowledgeBase",
      "Effect": "Allow",
      "Action": [
        "bedrock:AllowVendedLogDeliveryForResource"
      ],
      "Resource": "arn:aws:bedrock:us-east-1:123456789012:knowledge-
base/XXXXXXXXXX"
    },
    {
      "Sid": "CreateLogDeliveryPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:PutDeliverySource",
        "logs:CreateDelivery"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:123456789012:delivery-source:*",
        "arn:aws:logs:us-east-1:123456789012:delivery:*",
        "arn:aws:logs:us-east-1:444455556666:delivery-destination:*"
      ]
    }
  ]
}
```

Create delivery source

To begin, account A creates a delivery source with their bedrock knowledge base:

```
aws logs put-delivery-source --name my-delivery-source --log-type APPLICATION_LOGS --
resource-arn arn:aws:bedrock:region:AAAAAAAAAAAA:knowledge-base/XXXXXXXXXX
```

Next, account B must create the delivery destination using one of the flows below:

- [Configure delivery to an Amazon S3 bucket](#)

- [Configure delivery to a Firehose stream](#)

Configure delivery to an Amazon S3 bucket

Account B wants to receive the logs into their S3 bucket with the ARN `arn:aws:s3:::amzn-s3-demo-bucket`. For this example, account B will need the following permissions:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PutLogDestinationPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:PutDeliveryDestination",
        "logs:PutDeliveryDestinationPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    }
  ]
}
```

The bucket will need the following permissions in its bucket policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogsDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": [
```

```

        "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/AWSLogs/123456789012/
*",
    "Condition": {
        "StringEquals": {
            "s3:x-amz-acl": "bucket-owner-full-control",
            "aws:SourceAccount": [
                "123456789012"
            ]
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:logs:us-east-1:123456789012:delivery-source:my-
delivery-source"
            ]
        }
    }
}
]
}

```

If the bucket is encrypted with SSE-KMS, ensure the AWS KMS key policy has the appropriate permissions. For example, if the KMS key is `arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`, use the following:

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowLogsGenerateDataKey",
            "Effect": "Allow",
            "Principal": {
                "Service": "delivery.logs.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey"
            ],

```

```

    "Resource": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": [
          "123456789012"
        ]
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:123456789012:delivery-source:my-
delivery-source"
        ]
      }
    }
  }
]
}

```

Account B can then create a delivery destination with the S3 bucket as the destination resource:

```
aws logs put-delivery-destination --name my-s3-delivery-destination --delivery-
destination-configuration "destinationResourceArn=arn:aws:s3:::amzn-s3-demo-bucket"
```

Next, Account B creates a delivery destination policy on their newly created delivery destination, which will give permission for Account A to create a log delivery. The policy that will be added to the newly created delivery destination is the following:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateDelivery",
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": [

```

```

        "logs:CreateDelivery"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-
destination:amzn-s3-demo-bucket"
    }
  ]
}

```

This policy will be saved in Account B's computer as `destination-policy-s3.json`. To attach this resource, Account B will run the following command:

```
aws logs put-delivery-destination-policy --delivery-destination-name my-s3-delivery-
destination --delivery-destination-policy file://destination-policy-s3.json
```

Lastly, Account A creates the delivery, which links the delivery source in Account A to the delivery destination in Account B.

```
aws logs create-delivery --delivery-source-name my-delivery-source --delivery-
destination-arn arn:aws:logs:region:BBBBBBBBBBBB:delivery-destination:my-s3-delivery-
destination
```

Configure delivery to a Firehose stream

In this example, Account B wants to receive logs into their Firehose stream. The Firehose stream has the following ARN and is configured to use the DirectPut delivery stream type:

`arn:aws:firehose:us-east-1:111122223333:deliverystream/log-delivery-stream`

For this example, Account B needs the following permissions:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowFirehoseCreateSLR",
      "Effect": "Allow",
      "Action": [

```

```

        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
delivery.logs.amazonaws.com/AWSServiceRoleForLogDelivery"
  },
  {
    "Sid": "AllowFirehoseTagging",
    "Effect": "Allow",
    "Action": [
      "firehose:TagDeliveryStream"
    ],
    "Resource": "arn:aws:firehose:us-
east-1:111122223333:deliverystream/X"
  },
  {
    "Sid": "AllowFirehoseDeliveryDestination",
    "Effect": "Allow",
    "Action": [
      "logs:PutDeliveryDestination",
      "logs:PutDeliveryDestinationPolicy"
    ],
    "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-
destination:*"
  }
]
}

```

The Firehose stream must have the tag `LogDeliveryEnabled` set to `true`.

Account B will then create a delivery destination with the Firehose stream as the destination resource:

```

aws logs put-delivery-destination --name my-fh-delivery-destination --delivery-
destination-configuration
  "destinationResourceArn=arn:aws:firehose:region:BBBBBBBBBBBBB:deliverystream/X"

```

Next, Account B creates a delivery destination policy on their newly created delivery destination, which will give permission for Account A to create a log delivery. The policy to be added to the newly created delivery destination is the following:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateDelivery",
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": [
        "logs:CreateDelivery"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-destination:amzn-s3-demo-bucket"
    }
  ]
}
```

This policy will be saved in Account B's computer as `destination-policy-fh.json`. To attach this resource, Account B runs the following command:

```
aws logs put-delivery-destination-policy --delivery-destination-name my-fh-delivery-destination --delivery-destination-policy file://destination-policy-fh.json
```

Lastly, Account A creates the delivery, which links the delivery source in Account A to the delivery destination in Account B.

```
aws logs create-delivery --delivery-source-name my-delivery-source --delivery-destination-arn arn:aws:logs:region:BBBBBBBBBBBBB:delivery-destination:my-fh-delivery-destination
```

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur

when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#), [aws:SourceAccount](#), [aws:SourceOrgID](#), and [aws:SourceOrgPaths](#) global condition context keys in resource policies to limit the permissions that CloudWatch Logs gives another service to the resource. Use `aws:SourceArn` to associate only one resource with cross-service access. Use `aws:SourceAccount` to let any resource in that account be associated with the cross-service use. Use `aws:SourceOrgID` to allow any resource from any account within an organization be associated with the cross-service use. Use `aws:SourceOrgPaths` to associate any resource from accounts within an AWS Organizations path with the cross-service use. For more information about using and understanding paths, see [Understand the AWS Organizations entity path](#).

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global condition context key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:servicename:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both `aws:SourceAccount` and `aws:SourceArn` to limit permissions.

To protect against the confused deputy problem at scale, use the `aws:SourceOrgID` or `aws:SourceOrgPaths` global condition context key with the organization ID or organization path of the resource in your resource-based policies. Policies that include the `aws:SourceOrgID` or `aws:SourceOrgPaths` key will automatically include the correct accounts and you don't have to manually update the policies when you add, remove, or move accounts in your organization.

The policies documented for granting access to CloudWatch Logs to write data to Kinesis Data Streams and Firehose in [Step 1: Create a destination](#) and [Step 2: Create a destination](#) show how you can use the `aws:SourceArn` global condition context key to help prevent the confused deputy problem.

Exporting log data to Amazon S3

This chapter provides you with information, so you can export log data from your log groups to an Amazon S3 bucket for custom processing and analysis, or to load onto other systems. You can export to a bucket in the same account or a different account.

You can do the following:

- Export log data to S3 buckets that are encrypted by SSE-KMS in AWS Key Management Service (AWS KMS)
- Export log data to S3 buckets that have S3 Object Lock enabled with a retention period

We recommend that you don't regularly export to Amazon S3 as a way to continuously archive your logs. For that use case, we instead recommend that you use subscriptions. For more information about subscriptions, see [Real-time processing of log data with subscriptions](#).

To begin the export process, you must create an S3 bucket to store the exported log data. You can store the exported files in your S3 bucket and define Amazon S3 lifecycle rules to archive or delete exported files automatically.

You can export to S3 buckets that are encrypted with AES-256 or with SSE-KMS. Exporting to buckets encrypted with DSSE-KMS is not supported.

You can export logs from multiple log groups or multiple time ranges to the same S3 bucket. To separate log data for each export task, you can specify a prefix that will be used as the Amazon S3 key prefix for all exported objects.

Note

Time-based sorting on chunks of log data inside an exported file is not guaranteed. You can sort the exported log field data by using Linux utilities. For example, the following utility command sorts the events in all `.gz` files in a single folder.

```
find . -exec zcat {} + | sed -r 's/^[0-9]+/\x0&/' | sort -z
```

The following utility command sorts `.gz` files from multiple subfolders.

```
find ./ */ -type f -exec zcat {} + | sed -r 's/^[0-9]+/\x0&/' | sort -z
```

Additionally, you can use another `stdout` command to pipe the sorted output to another file to save it.

Log data can take up to 12 hours to become available for export. Export tasks time out after 24 hours. If your export tasks are timing out, reduce the time range when you create the export task.

For near real-time analysis of log data, see [Analyzing log data with CloudWatch Logs Insights](#) or [Real-time processing of log data with subscriptions](#) instead.

Contents

- [Concepts](#)
- [Export log data to Amazon S3 using the console](#)
- [Export log data to Amazon S3 using the AWS CLI](#)
- [Describe export tasks \(CLI\)](#)
- [Cancel an export task \(CLI\)](#)

Concepts

Before you begin, become familiar with the following export concepts:

log group name

The name of the log group associated with an export task. The log data in this log group will be exported to the specified S3 bucket.

from (timestamp)

A required timestamp expressed as the number of milliseconds since Jan 1, 1970 00:00:00 UTC. All log events in the log group that were ingested on or after this time will be exported.

to (timestamp)

A required timestamp expressed as the number of milliseconds since Jan 1, 1970 00:00:00 UTC. All log events in the log group that were ingested before this time will be exported.

destination bucket

The name of the S3 bucket associated with an export task. This bucket is used to export the log data from the specified log group.

destination prefix

An optional attribute that is used as the Amazon S3 key prefix for all exported objects. This helps create a folder-like organization in your bucket.

Export log data to Amazon S3 using the console

In the following examples, you use the Amazon CloudWatch console to export all data from an Amazon CloudWatch Logs log group named `my-log-group` to an Amazon S3 bucket named `my-exported-logs`.

Exporting log data to S3 buckets that are encrypted by SSE-KMS is supported. Exporting to buckets encrypted with DSSE-KMS is not supported.

The details of how you set up the export depends on whether the Amazon S3 bucket that you want to export to is in the same account as your logs that are being exported, or in a different account.

Topics

- [Same-account export \(console\)](#)
- [Cross-account export \(console\)](#)

Same-account export (console)

If the Amazon S3 bucket is in the same account as the logs that are being exported, use the instructions in this section.

Topics

- [Create an Amazon S3 bucket \(console\)](#)
- [Set up access permissions \(console\)](#)
- [Set permissions on an Amazon S3 bucket \(console\)](#)
- [\(Optional\) Exporting to a bucket encrypted with SSE-KMS \(console\)](#)
- [Create an export task \(console\)](#)

Create an Amazon S3 bucket (console)

We recommend that you use a bucket that was created specifically for CloudWatch Logs. However, if you want to use an existing bucket, you can skip to step 2.

Note

The Amazon S3 bucket must reside in the same Region as the log data to export. CloudWatch Logs doesn't support exporting data to Amazon S3 buckets in a different Region.

To create an Amazon S3 bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. If necessary, change the Region. From the navigation bar, choose the Region where your CloudWatch Logs reside.
3. Choose **Create Bucket**.
4. For **Bucket Name**, enter a name for the bucket.
5. For **Region**, select the Region where your CloudWatch Logs data resides.
6. Choose **Create**.

Set up access permissions (console)

To create the export task, you'll need to be signed on with the AmazonS3ReadOnlyAccess IAM role and with the following permissions:

- logs:CreateExportTask
- logs:CancelExportTask
- logs:DescribeExportTasks
- logs:DescribeLogStreams
- logs:DescribeLogGroups

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Set permissions on an Amazon S3 bucket (console)

By default, all Amazon S3 buckets and objects are private. Only the resource owner, the AWS account that created the bucket, can access the bucket and any objects that it contains. However, the resource owner can choose to grant access permissions to other resources and users by writing an access policy.

When you set the policy, we recommend that you include a randomly generated string as the prefix for the bucket, so that only intended log streams are exported to the bucket.

Important

To make exports to Amazon S3 buckets more secure, we now require you to specify the list of source accounts that are allowed to export log data to your S3 bucket.

In the following example, the list of account IDs in the `aws:SourceAccount` key would be the accounts from which a user can export log data to your Amazon S3 bucket. The `aws:SourceArn` key would be the resource for which the action is being taken. You may restrict this to a specific log group, or use a wildcard as shown in this example.

We recommend that you also include the account ID of the account where the S3 bucket is created, to allow export within the same account.

To set permissions on an Amazon S3 bucket

1. In the Amazon S3 console, choose the bucket that you created.
2. Choose **Permissions, Bucket policy**.

3. In the **Bucket Policy Editor**, add the following policy. Change my-exported-logs to the name of your S3 bucket. Be sure to specify the correct Region endpoint, such as us-west-1, for **Principal**.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudWatchLogsGetBucketAcl",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:log-group:*",
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
          ]
        }
      }
    },
    {
      "Sid": "AllowCloudWatchLogsPutObject",
      "Action": "s3:PutObject",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control",
          "aws:SourceAccount": [
            "123456789012",

```



```

        "111122223333"
      ],
    },
    "ArnLike": {
      "aws:SourceArn": [
        "arn:aws:logs:us-east-1:123456789012:log-group:*",
        "arn:aws:logs:us-east-1:111122223333:log-group:*"
      ]
    }
  }
}
]
}

```

4. Choose **Save** to set the policy that you just added as the access policy on your bucket. This policy enables CloudWatch Logs to export log data to your Amazon S3 bucket. The bucket owner has full permissions on all of the exported objects.

Warning

If the existing bucket already has one or more policies attached to it, add the statements for CloudWatch Logs access to that policy or policies. We recommend that you evaluate the resulting set of permissions to be sure that they're appropriate for the users who will access the bucket.

(Optional) Exporting to a bucket encrypted with SSE-KMS (console)

This step is necessary only if you are exporting to an Amazon S3 bucket that uses server-side encryption with AWS KMS keys. This encryption is known as SSE-KMS.

To export to a bucket encrypted with SSE-KMS

1. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the left navigation bar, choose **Customer managed keys**.

Choose **Create Key**.

4. For **Key type**, choose **Symmetric**.

5. For **Key usage**, choose **Encrypt and decrypt** and then choose **Next**.
6. Under **Add labels**, enter an alias for the key and optionally add a description or tags. Then choose **Next**.
7. Under **Key administrators**, select who can administer this key, and then choose **Next**.
8. Under **Define key usage permissions**, make no changes and choose **Next**.
9. Review the settings and choose **Finish**.
10. Back at the **Customer managed keys** page, choose the name of the key that you just created.
11. Choose the **Key policy** tab and choose **Switch to policy view**.
12. In the **Key policy** section, choose **Edit**.
13. Add the following statement to the key policy statement list. When you do, replace *Region* with the Region of your logs and replace *account-ARN* with the ARN of the account that owns the KMS key.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CWL Service Principal usage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.Region.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",

```

```
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
]
```

14. Choose **Save changes**.
15. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
16. Find the bucket that you created in [Create an S3 bucket \(CLI\)](#) and choose the bucket name.
17. Choose the **Properties** tab. Then, under **Default Encryption**, choose **Edit**.
18. Under **Server-side Encryption**, choose **Enable**.
19. Under **Encryption type**, choose **AWS Key Management Service key (SSE-KMS)**.
20. Choose **Choose from your AWS KMS keys** and find the key that you created.
21. For **Bucket key**, choose **Enable**.
22. Choose **Save changes**.

Create an export task (console)

In this procedure, you create the export task for exporting logs from a log group.

To export data to Amazon S3 using the CloudWatch console

1. Sign in with sufficient permissions as documented in [Set up access permissions \(console\)](#).
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Log groups**.
4. On the **Log Groups** screen, choose the name of the log group.
5. Choose **Actions, Export data to Amazon S3**.
6. On the **Export data to Amazon S3** screen, under **Define data export**, set the time range for the data to export using **From** and **To**.

7. If your log group has multiple log streams, you can provide a log stream prefix to limit the log group data to a specific stream. Choose **Advanced**, and then for **Stream prefix**, enter the log stream prefix.
8. Under **Choose S3 bucket**, choose the account associated with the S3 bucket.
9. For **S3 bucket name**, choose an S3 bucket.
10. For **S3 Bucket prefix**, enter the randomly generated string that you specified in the bucket policy.
11. Choose **Export** to export your log data to Amazon S3.
12. To view the status of the log data that you exported to Amazon S3, choose **Actions** and then **View all exports to Amazon S3**.

Cross-account export (console)

If the Amazon S3 bucket is in a different account than the logs that are being exported, use the instructions in this section.

Topics

- [Create an Amazon S3 bucket for cross-account export \(console\)](#)
- [Set up access permissions for cross-account export \(console\)](#)
- [Set permissions on an S3 bucket for cross-account export \(console\)](#)
- [\(Optional\) Exporting to a bucket encrypted with SSE-KMS for cross-account export \(console\)](#)
- [Create an export task for cross-account export \(console\)](#)

Create an Amazon S3 bucket for cross-account export (console)

We recommend that you use a bucket that was created specifically for CloudWatch Logs. However, if you want to use an existing bucket, you can skip this procedure.

Note

The Amazon S3 bucket must reside in the same Region as the log data to export. CloudWatch Logs doesn't support exporting data to Amazon S3 buckets in a different Region.

To create an Amazon S3 bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. If necessary, change the Region. From the navigation bar, choose the Region where your CloudWatch Logs reside.
3. Choose **Create Bucket**.
4. For **Bucket Name**, enter a name for the bucket.
5. For **Region**, select the Region where your CloudWatch Logs data resides.
6. Choose **Create**.

Set up access permissions for cross-account export (console)

First, you must create a new IAM policy to enable CloudWatch Logs to have the `s3:PutObject` action for the destination Amazon S3 bucket in the destination account.

The additional actions included in the policy depend on whether the destination bucket uses AWS KMS encryption or has ACLs enabled using the [S3 Object Ownership](#) setting.

- If using KMS encryption, add the `GenerateDataKey` and `Decrypt` actions for the key resource
- If ACLs are enabled on the bucket add the `s3:PutObjectAcl` action for the bucket resource

To create an IAM policy to export logs to an Amazon S3 bucket

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose **Policies**.
3. Choose **Create policy**.
4. In the **Policy editor** section, choose **JSON**.
5. If the destination bucket does not use AWS KMS encryption, paste the following policy into the editor.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Effect": "Allow",
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::my-exported-logs/*"
    }
]
}

```

If the destination bucket does use AWS KMS encryption, paste the following policy into the editor.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}

```

6. Choose **Next**.
7. Enter a policy name. You will use this name to attach the policy to your IAM role.
8. Choose **Create policy** to save the new policy.

To create the export task, you'll need to be signed in with the AmazonS3ReadOnlyAccess IAM role. You must also be signed in with the IAM policy that you just created, and also with the following permissions:

- `logs:CreateExportTask`
- `logs:CancelExportTask`
- `logs:DescribeExportTasks`
- `logs:DescribeLogStreams`
- `logs:DescribeLogGroups`

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:

- Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
- (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Set permissions on an S3 bucket for cross-account export (console)

By default, all S3 buckets and objects are private. Only the resource owner, the AWS account that created the bucket, can access the bucket and any objects that it contains. However, the resource owner can choose to grant access permissions to other resources and users by writing an access policy.

When you set the policy, we recommend that you include a randomly generated string as the prefix for the bucket, so that only intended log streams are exported to the bucket.

Important

To make exports to S3 buckets more secure, we now require you to specify the list of source accounts that are allowed to export log data to your S3 bucket.

In the following example, the list of account IDs in the `aws:SourceAccount` key would be the accounts from which a user can export log data to your S3 bucket. The `aws:SourceArn` key would be the resource for which the action is being taken. You may restrict this to a specific log group, or use a wildcard as shown in this example. We recommend that you also include the account ID of the account where the S3 bucket is created, to allow export within the same account.

To set permissions on an Amazon S3 bucket

1. In the Amazon S3 console, choose the bucket that you created.
2. Choose **Permissions, Bucket policy**.
3. In the **Bucket Policy Editor**, add the following policy. Change `my-exported-logs` to the name of your S3 bucket. Be sure to specify the correct Region endpoint, such as `us-west-1`, for **Principal**.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:log-group:*",
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
          ]
        }
      }
    }
  ]
}
```



```

    }
  },
  {
    "Action": "s3:PutObject",
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control",
        "aws:SourceAccount": [
          "123456789012",
          "111122223333"
        ]
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:123456789012:log-group:*",
          "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/role_name"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control"
      }
    }
  }
]
}

```

4. Choose **Save** to set the policy that you just added as the access policy on your bucket. This policy enables CloudWatch Logs to export log data to your S3 bucket. The bucket owner has full permissions on all of the exported objects.

⚠ Warning

If the existing bucket already has one or more policies attached to it, add the statements for CloudWatch Logs access to that policy or policies. We recommend that you evaluate the resulting set of permissions to be sure that they're appropriate for the users who will access the bucket.

(Optional) Exporting to a bucket encrypted with SSE-KMS for cross-account export (console)

This procedure is necessary only if you are exporting to an S3 bucket that uses server-side encryption with AWS KMS keys. This encryption is known as SSE-KMS.

To export to a bucket encrypted with SSE-KMS

1. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the left navigation bar, choose **Customer managed keys**.

Choose **Create Key**.
4. For **Key type**, choose **Symmetric**.
5. For **Key usage**, choose **Encrypt and decrypt** and then choose **Next**.
6. Under **Add labels**, enter an alias for the key and optionally add a description or tags. Then choose **Next**.
7. Under **Key administrators**, select who can administer this key, and then choose **Next**.
8. Under **Define key usage permissions**, make no changes and choose **Next**.
9. Review the settings and choose **Finish**.
10. Back at the **Customer managed keys** page, choose the name of the key that you just created.
11. Choose the **Key policy** tab and choose **Switch to policy view**.
12. In the **Key policy** section, choose **Edit**.
13. Add the following statement to the key policy statement list. When you do, replace *Region* with the Region of your logs and replace *account-ARN* with the ARN of the account that owns the KMS key.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CWL Service Principal usage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM Role Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role_name"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ]
    }
  ]
}
```

```
    ],  
    "Resource": "arn:aws:kms:us-east-1:111122223333:key/key-id"  
  }  
]  
}
```

14. Choose **Save changes**.
15. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
16. Find the bucket that you created in [Create an S3 bucket \(CLI\)](#) and choose the bucket name.
17. Choose the **Properties** tab. Then, under **Default Encryption**, choose **Edit**.
18. Under **Server-side Encryption**, choose **Enable**.
19. Under **Encryption type**, choose **AWS Key Management Service key (SSE-KMS)**.
20. Choose **Choose from your AWS KMS keys** and find the key that you created.
21. For **Bucket key**, choose **Enable**.
22. Choose **Save changes**.

Create an export task for cross-account export (console)

In this procedure, you create the export task for exporting logs from a log group.

To export data to Amazon S3 using the CloudWatch console

1. Sign in with sufficient permissions as documented in [Set up access permissions \(console\)](#).
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Log groups**.
4. On the **Log Groups** screen, choose the name of the log group.
5. Choose **Actions, Export data to Amazon S3**.
6. On the **Export data to Amazon S3** screen, under **Define data export**, set the time range for the data to export using **From** and **To**.
7. If your log group has multiple log streams, you can provide a log stream prefix to limit the log group data to a specific stream. Choose **Advanced**, and then for **Stream prefix**, enter the log stream prefix.
8. Under **Choose S3 bucket**, choose the account associated with the S3 bucket.
9. For **S3 bucket name**, choose an S3 bucket.

10. For **S3 Bucket prefix**, enter the randomly generated string that you specified in the bucket policy.
11. Choose **Export** to export your log data to Amazon S3.
12. To view the status of the log data that you exported to Amazon S3, choose **Actions** and then **View all exports to Amazon S3**.

Export log data to Amazon S3 using the AWS CLI

In the following example, you use an export task to export all data from a CloudWatch Logs log group named `my-log-group` to an Amazon S3 bucket named `my-exported-logs`. This example assumes that you have already created a log group called `my-log-group`.

Exporting log data to S3 buckets that are encrypted by AWS KMS is supported. Exporting to buckets encrypted with DSSE-KMS is not supported.

The details of how you set up the export depends on whether the Amazon S3 bucket that you want to export to is in the same account as your logs that are being exported, or in a different account.

Topics

- [Same-account export \(CLI\)](#)
- [Cross-account export \(CLI\)](#)

Same-account export (CLI)

If the Amazon S3 bucket is in the same account as the logs that are being exported, use the instructions in this section.

Topics

- [Create an S3 bucket \(CLI\)](#)
- [Set up access permissions \(CLI\)](#)
- [Set permissions on an S3 bucket \(CLI\)](#)
- [\(Optional\) Exporting to a bucket encrypted with SSE-KMS \(CLI\)](#)
- [Create an export task \(CLI\)](#)

Create an S3 bucket (CLI)

We recommend that you use a bucket that was created specifically for CloudWatch Logs. However, if you want to use an existing bucket, you can skip this procedure.

Note

The S3 bucket must reside in the same Region as the log data to export. CloudWatch Logs doesn't support exporting data to S3 buckets in a different Region.

To create an S3 bucket using the AWS CLI

At a command prompt, run the following [create-bucket](#) command, where `LocationConstraint` is the Region where you are exporting log data.

```
aws s3api create-bucket --bucket my-exported-logs --create-bucket-configuration  
LocationConstraint=us-east-2
```

The following is example output.

```
{  
  "Location": "/my-exported-logs"  
}
```

Set up access permissions (CLI)

To create the export task later, you'll need to be signed on with the `AmazonS3ReadOnlyAccess` IAM role and with the following permissions:

- `logs:CreateExportTask`
- `logs:CancelExportTask`
- `logs:DescribeExportTasks`
- `logs:DescribeLogStreams`
- `logs:DescribeLogGroups`

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:

- Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
- (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Set permissions on an S3 bucket (CLI)

By default, all S3 buckets and objects are private. Only the resource owner, the account that created the bucket, can access the bucket and any objects that it contains. However, the resource owner can choose to grant access permissions to other resources and users by writing an access policy.

Important

To make exports to S3 buckets more secure, we now require you to specify the list of source accounts that are allowed to export log data to your S3 bucket.

In the following example, the list of account IDs in the `aws:SourceAccount` key would be the accounts from which a user can export log data to your S3 bucket. The `aws:SourceArn` key would be the resource for which the action is being taken. You may restrict this to a specific log group, or use a wildcard as shown in this example.

We recommend that you also include the account ID of the account where the S3 bucket is created, to allow export within the same account.

To set permissions on an S3 bucket

1. Create a file named `policy.json` and add the following access policy, changing `my-exported-logs` to the name of your S3 bucket and `Principal` to the endpoint of the

Region where you are exporting log data, such as `us-west-1`. Use a text editor to create this policy file. Don't use the IAM console.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGetBucketAcl",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        }
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:123456789012:log-group:*",
          "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
      }
    },
    {
      "Sid": "AllowPutObject",
      "Action": "s3:PutObject",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control",
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        }
      }
    }
  ]
}
```



```

    ],
    },
    "ArnLike": {
      "aws:SourceArn": [
        "arn:aws:logs:us-east-1:123456789012:log-group:*",
        "arn:aws:logs:us-east-1:111122223333:log-group:*"
      ]
    }
  }
}
]
}

```

2. Set the policy that you just added as the access policy on your bucket by using the [put-bucket-policy](#) command. This policy enables CloudWatch Logs to export log data to your S3 bucket. The bucket owner will have full permissions on all of the exported objects.

```
aws s3api put-bucket-policy --bucket my-exported-logs --policy file://policy.json
```

Warning

If the existing bucket already has one or more policies attached to it, add the statements for CloudWatch Logs access to that policy or policies. We recommend that you evaluate the resulting set of permissions to be sure that they're appropriate for the users who will access the bucket.

(Optional) Exporting to a bucket encrypted with SSE-KMS (CLI)

This procedure is necessary only if you are exporting to an S3 bucket that uses server-side encryption with AWS KMS keys. This encryption is known as SSE-KMS.

To export to a bucket encrypted with SSE-KMS

1. Use a text editor to create a file named `key_policy.json` and add the following access policy. When you add the policy, make the following changes:
 - Replace *Region* with the Region of your logs.
 - Replace *account-ARN* with the ARN of the account that owns the KMS key.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CWL Service Principal usage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.Region.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

2. Enter the following command:

```
aws kms create-key --policy file://key_policy.json
```

The following is example output from this command:

```
{
  "KeyMetadata": {
    "AWSAccountId": "account_id",
    "KeyId": "key_id",
    "Arn": "arn:aws:kms:us-east-2:account_id:key/key_id",
    "CreationDate": "time",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "MultiRegion": false
  }
}
```

3. Use a text editor to create a file called `bucketencryption.json` with the following contents.

```
{
  "Rules": [
    {
      "ApplyServerSideEncryptionByDefault": {
        "SSEAlgorithm": "aws:kms",
        "KMSMasterKeyID": "{KMS Key ARN}"
      },
      "BucketKeyEnabled": true
    }
  ]
}
```

4. Enter the following command, replacing *bucket-name* with the name of the bucket that you are exporting logs to.

```
aws s3api put-bucket-encryption --bucket bucket-name --server-side-encryption-configuration file://bucketencryption.json
```

If the command doesn't return an error, the process is successful.

Create an export task (CLI)

Use the following command to create the export task. After you create it, the export task might take anywhere from a few seconds to a few hours, depending on the size of the data to export.

To export data to Amazon S3 using the AWS CLI

1. Sign in with sufficient permissions as documented in [Set up access permissions \(CLI\)](#).
2. At a command prompt, use the following [create-export-task](#) command to create the export task.

```
aws logs create-export-task --profile CWLExportUser --task-name "my-log-group-09-10-2015" --log-group-name "my-log-group" --from 1441490400000 --to 1441494000000 --destination "my-exported-logs" --destination-prefix "export-task-output"
```

The following is example output.

```
{
  "taskId": "cda45419-90ea-4db5-9833-aade86253e66"
}
```

Cross-account export (CLI)

If the Amazon S3 bucket is in a different account than the logs that are being exported, use the instructions in this section.

Topics

- [Create an S3 bucket for cross-account export \(CLI\)](#)
- [Set up access permissions for cross-account export \(CLI\)](#)
- [Set permissions on an S3 bucket for cross-account export \(CLI\)](#)

- [\(Optional\) Exporting to a bucket encrypted with SSE-KMS for cross-account export \(CLI\)](#)
- [Create an export task for cross-account export \(CLI\)](#)

Create an S3 bucket for cross-account export (CLI)

We recommend that you use a bucket that was created specifically for CloudWatch Logs. However, if you want to use an existing bucket, you can skip to step 2.

Note

The S3 bucket must reside in the same Region as the log data to export. CloudWatch Logs doesn't support exporting data to S3 buckets in a different Region.

To create an S3 bucket using the AWS CLI

At a command prompt, run the following [create-bucket](#) command, where `LocationConstraint` is the Region where you are exporting log data.

```
aws s3api create-bucket --bucket my-exported-logs --create-bucket-configuration  
LocationConstraint=us-east-2
```

The following is example output.

```
{  
  "Location": "/my-exported-logs"  
}
```

Set up access permissions for cross-account export (CLI)

First, you must create a new IAM policy to enable CloudWatch Logs to have the `s3:PutObject` permission for the destination Amazon S3 bucket.

To create the export task later, you'll need to be signed on with the `AmazonS3ReadOnlyAccess` IAM role and with certain other permissions. You can create a policy that contains some of these other necessary permissions.

The policy that you create depends on whether the destination bucket uses AWS KMS encryption. If it does not use AWS KMS encryption, create a policy with the following contents.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::my-exported-logs/*"
    }
  ]
}
```

If the destination bucket uses AWS KMS encryption, create a policy with the following contents.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::>amzn-s3-demo-bucket/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:us-east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }
]
```

To create the export task later, you must be signed on with the AmazonS3ReadOnlyAccess IAM role, the IAM policy that you just created, and also with the following permissions:

- `logs:CreateExportTask`
- `logs:CancelExportTask`
- `logs:DescribeExportTasks`
- `logs:DescribeLogStreams`
- `logs:DescribeLogGroups`

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Set permissions on an S3 bucket for cross-account export (CLI)

By default, all S3 buckets and objects are private. Only the resource owner, the account that created the bucket, can access the bucket and any objects that it contains. However, the resource owner can choose to grant access permissions to other resources and users by writing an access policy.

Important

To make exports to S3 buckets more secure, we now require you to specify the list of source accounts that are allowed to export log data to your S3 bucket.

In the following example, the list of account IDs in the `aws:SourceAccount` key would be the accounts from which a user can export log data to your S3 bucket. The

`aws:SourceArn` key would be the resource for which the action is being taken. You may restrict this to a specific log group, or use a wildcard as shown in this example. We recommend that you also include the account ID of the account where the S3 bucket is created, to allow export within the same account.

To set permissions on an S3 bucket

1. Create a file named `policy.json` and add the following access policy, changing `my-exported-logs` to the name of your S3 bucket and `Principal` to the endpoint of the Region where you are exporting log data, such as `us-west-1`. Use a text editor to create this policy file. Don't use the IAM console.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:log-group:*",
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
          ]
        }
      }
    },
    {
      "Action": "s3:PutObject" ,
```



```

    "Effect": "Allow",
    "Resource": "arn:aws:s3:::my-exported-logs/*",
    "Principal": { "Service": "logs.Region.amazonaws.com" },
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control",
        "aws:SourceAccount": [
          "123456789012",
          "111122223333"
        ]
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:123456789012:log-group:*",
          "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/role_name"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::>amzn-s3-demo-bucket/*",
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control"
      }
    }
  }
]
}

```

2. Set the policy that you just added as the access policy on your bucket by using the [put-bucket-policy](#) command. This policy enables CloudWatch Logs to export log data to your S3 bucket. The bucket owner will have full permissions on all of the exported objects.

```
aws s3api put-bucket-policy --bucket my-exported-logs --policy file://policy.json
```

⚠ Warning

If the existing bucket already has one or more policies attached to it, add the statements for CloudWatch Logs access to that policy or policies. We recommend that you evaluate the resulting set of permissions to be sure that they're appropriate for the users who will access the bucket.

(Optional) Exporting to a bucket encrypted with SSE-KMS for cross-account export (CLI)

This procedure is necessary only if you are exporting to an S3 bucket that uses server-side encryption with AWS KMS keys. This encryption is known as SSE-KMS.

To export to a bucket encrypted with SSE-KMS

1. Use a text editor to create a file named `key_policy.json` and add the following access policy. When you add the policy, make the following changes:
 - Replace *Region* with the Region of your logs.
 - Replace *account-ARN* with the ARN of the account that owns the KMS key.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCWLServicePrincipalUsage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
    }
  ],
}
```

```

        "Resource": "*"
    },
    {
        "Sid": "EnableIAMUserPermissions",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::123456789012:root"
        },
        "Action": [
            "kms:GetKeyPolicy*",
            "kms:PutKeyPolicy*",
            "kms:DescribeKey*",
            "kms:CreateAlias*",
            "kms:ScheduleKeyDeletion*",
            "kms:Decrypt"
        ],
        "Resource": "*"
    },
    {
        "Sid": "EnableIAMRolePermissions",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/role_name"
        },
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "arn:aws:kms:us-east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
]
}

```

2. Enter the following command:

```
aws kms create-key --policy file:///key_policy.json
```

The following is example output from this command:

```

{
  "KeyMetadata": {
    "AWSAccountId": "account_id",

```

```

    "KeyId": "key_id",
    "Arn": "arn:aws:kms:us-east-2:account_id:key/key_id",
    "CreationDate": "time",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ],
    "MultiRegion": false
}

```

3. Use a text editor to create a file called `bucketencryption.json` with the following contents.

```

{
  "Rules": [
    {
      "ApplyServerSideEncryptionByDefault": {
        "SSEAlgorithm": "aws:kms",
        "KMSEMasterKeyID": "{KMS Key ARN}"
      },
      "BucketKeyEnabled": true
    }
  ]
}

```

4. Enter the following command, replacing *bucket-name* with the name of the bucket that you are exporting logs to.

```
aws s3api put-bucket-encryption --bucket bucket-name --server-side-encryption-configuration file://bucketencryption.json
```

If the command doesn't return an error, the process is successful.

Create an export task for cross-account export (CLI)

Use the following command to create the export task. After you create it, the export task might take anywhere from a few seconds to a few hours, depending on the size of the data to export.

To export data to Amazon S3 using the AWS CLI

1. Sign in with sufficient permissions as documented in [Set up access permissions \(CLI\)](#).
2. At a command prompt, use the following [create-export-task](#) command to create the export task.

```
aws logs create-export-task --profile CWLExportUser --task-name "my-log-group-09-10-2015" --log-group-name "my-log-group" --from 1441490400000 --to 1441494000000 --destination "my-exported-logs" --destination-prefix "export-task-output"
```

The following is example output.

```
{
  "taskId": "cda45419-90ea-4db5-9833-aade86253e66"
}
```

Describe export tasks (CLI)

After you create an export task, you can get the current status of the task.

To describe export tasks using the AWS CLI

At a command prompt, use the following [describe-export-tasks](#) command.

```
aws logs --profile CWLExportUser describe-export-tasks --task-id "cda45419-90ea-4db5-9833-aade86253e66"
```

The following is example output.

```
{
  "exportTasks": [
    {
      "destination": "my-exported-logs",
```

```

    "destinationPrefix": "export-task-output",
    "executionInfo": {
      "creationTime": 1441495400000
    },
    "from": 1441490400000,
    "logGroupName": "my-log-group",
    "status": {
      "code": "RUNNING",
      "message": "Started Successfully"
    },
    "taskId": "cda45419-90ea-4db5-9833-aade86253e66",
    "taskName": "my-log-group-09-10-2015",
    "tTo": 1441494000000
  }]
}
```

You can use the `describe-export-tasks` command in three different ways:

- **Without any filters** – Lists all of your export tasks, in reverse order of creation.
- **Filter on task ID** – Lists the export task, if one exists, with the specified ID.
- **Filter on task status** – Lists the export tasks with the specified status.

For example, use the following command to filter on the `FAILED` status.

```
aws logs --profile CWLEXPOTUser describe-export-tasks --status-code "FAILED"
```

The following is example output.

```

{
  "exportTasks": [
    {
      "destination": "my-exported-logs",
      "destinationPrefix": "export-task-output",
      "executionInfo": {
        "completionTime": 1441498600000
        "creationTime": 1441495400000
      },
      "from": 1441490400000,
      "logGroupName": "my-log-group",
      "status": {
        "code": "FAILED",

```

```
    "message": "FAILED"
  },
  "taskId": "cda45419-90ea-4db5-9833-aade86253e66",
  "taskName": "my-log-group-09-10-2015",
  "to": 1441494000000
}]
}
```

Cancel an export task (CLI)

You can cancel an export task if it's in a PENDING or RUNNING state.

To cancel an export task using the AWS CLI

At a command prompt, use the following [cancel-export-task](#) command:

```
aws logs --profile CWLExportUser cancel-export-task --task-id "cda45419-90ea-4db5-9833-aade86253e66"
```

You can use the [describe-export-tasks](#) command to verify that the task was canceled successfully.

Streaming CloudWatch Logs data to Amazon OpenSearch Service

You can configure a log group in Amazon CloudWatch Logs, so you can stream data to your Amazon OpenSearch Service cluster in near real-time. For more information, see [Real-time processing of log data with subscriptions](#).

Note

Streaming to OpenSearch Service is supported only for log groups in the Standard log class. For more information about log classes, see [Log classes](#).

Depending on the amount of log data that's streamed, consider setting a function-level concurrency limit. For more information, see [Lambda function scaling](#).

Note

Because streaming large amounts of CloudWatch Logs data to OpenSearch Service might result in high usage charges, we recommend that you create a budget in the AWS Billing and Cost Management console. For more information, see [Managing your costs with AWS Budgets](#).

This section describes the prerequisites you must complete before subscribing a log group to OpenSearch Service. It also describes how to subscribe a log group to OpenSearch Service.

Prerequisites

Before you begin, create an OpenSearch Service domain. The domain can have either public access or VPC access, but you can't then modify the type of access after the domain is created. You might want to review your OpenSearch Service domain settings later, and modify your cluster configuration based on the amount of data your cluster will be processing. For instructions to create a domain, see [Creating OpenSearch Service domains](#).

For more information about OpenSearch Service, see the [Amazon OpenSearch Service Developer Guide](#).

Subscribe a log group to OpenSearch Service

You can use the CloudWatch console to subscribe a log group to OpenSearch Service.

To subscribe a log group to OpenSearch Service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Select the name of the log group.
4. Choose **Actions, Subscription filters, Create Amazon OpenSearch Service subscription filter**.
5. Choose whether you want to stream to a cluster in this account or another account.
 - If you chose this account, select the domain you created in the previous step.
 - If you chose another account, provide the domain ARN and endpoint.
6. For **Lambda IAM Execution Role**, choose the IAM role that Lambda should use when executing calls to OpenSearch.

The IAM role you choose must fulfill these requirements:

- It must have `lambda.amazonaws.com` in the trust relationship.
- It must include the following policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowOpenSearchStreamingAccess",
      "Action": [
        "es:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:es:us-east-1:123456789012:domain/cloudwatch-logs/*"
    }
  ]
}
```

- If the target OpenSearch Service domain uses VPC access, the role must have the **AWSLambdaVPCAccessExecutionRole** policy attached. This Amazon-managed policy grants Lambda access to the customer's VPC, enabling Lambda to write to the OpenSearch endpoint in the VPC.
7. For **Log format**, choose a log format.
 8. For **Subscription filter pattern**, type the terms or pattern to find in your log events. This ensures that you send only the data you're interested in to your OpenSearch cluster. For more information, see [Creating metrics from log events using filters](#).
 9. (Optional) For **Select log data to test**, select a log stream and then choose **Test pattern** to verify that your search filter is returning the results you expect.
 10. Choose **Start streaming**.

Code examples for CloudWatch Logs using AWS SDKs

The following code examples show how to use CloudWatch Logs 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.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Basic examples for CloudWatch Logs using AWS SDKs](#)
 - [Actions for CloudWatch Logs using AWS SDKs](#)
 - [Use AssociateKmsKey with an AWS SDK](#)
 - [Use CancelExportTask with an AWS SDK](#)
 - [Use CreateExportTask with an AWS SDK](#)
 - [Use CreateLogGroup with an AWS SDK or CLI](#)
 - [Use CreateLogStream with an AWS SDK or CLI](#)
 - [Use DeleteLogGroup with an AWS SDK or CLI](#)
 - [Use DeleteSubscriptionFilter with an AWS SDK](#)
 - [Use DescribeExportTasks with an AWS SDK](#)
 - [Use DescribeLogGroups with an AWS SDK or CLI](#)
 - [Use DescribeLogStreams with an AWS SDK or CLI](#)
 - [Use DescribeSubscriptionFilters with an AWS SDK](#)
 - [Use GetLogEvents with an AWS SDK or CLI](#)
 - [Use GetQueryResults with an AWS SDK](#)
 - [Use PutSubscriptionFilter with an AWS SDK](#)
 - [Use StartLiveTail with an AWS SDK](#)
 - [Use StartQuery with an AWS SDK](#)

- [Scenarios for CloudWatch Logs using AWS SDKs](#)
 - [Use CloudWatch Logs to run a large query](#)
 - [Use scheduled events to invoke a Lambda function](#)

Basic examples for CloudWatch Logs using AWS SDKs

The following code examples show how to use the basics of Amazon CloudWatch Logs with AWS SDKs.

Examples

- [Actions for CloudWatch Logs using AWS SDKs](#)
 - [Use AssociateKmsKey with an AWS SDK](#)
 - [Use CancelExportTask with an AWS SDK](#)
 - [Use CreateExportTask with an AWS SDK](#)
 - [Use CreateLogGroup with an AWS SDK or CLI](#)
 - [Use CreateLogStream with an AWS SDK or CLI](#)
 - [Use DeleteLogGroup with an AWS SDK or CLI](#)
 - [Use DeleteSubscriptionFilter with an AWS SDK](#)
 - [Use DescribeExportTasks with an AWS SDK](#)
 - [Use DescribeLogGroups with an AWS SDK or CLI](#)
 - [Use DescribeLogStreams with an AWS SDK or CLI](#)
 - [Use DescribeSubscriptionFilters with an AWS SDK](#)
 - [Use GetLogEvents with an AWS SDK or CLI](#)
 - [Use GetQueryResults with an AWS SDK](#)
 - [Use PutSubscriptionFilter with an AWS SDK](#)
 - [Use StartLiveTail with an AWS SDK](#)
 - [Use StartQuery with an AWS SDK](#)

Actions for CloudWatch Logs using AWS SDKs

The following code examples demonstrate how to perform individual CloudWatch Logs actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the CloudWatch Logs API and are code excerpts from larger programs that must be run in context. You can see actions in context in [Scenarios for CloudWatch Logs using AWS SDKs](#).

The following examples include only the most commonly used actions. For a complete list, see the [Amazon CloudWatch Logs API Reference](#).

Examples

- [Use AssociateKmsKey with an AWS SDK](#)
- [Use CancelExportTask with an AWS SDK](#)
- [Use CreateExportTask with an AWS SDK](#)
- [Use CreateLogGroup with an AWS SDK or CLI](#)
- [Use CreateLogStream with an AWS SDK or CLI](#)
- [Use DeleteLogGroup with an AWS SDK or CLI](#)
- [Use DeleteSubscriptionFilter with an AWS SDK](#)
- [Use DescribeExportTasks with an AWS SDK](#)
- [Use DescribeLogGroups with an AWS SDK or CLI](#)
- [Use DescribeLogStreams with an AWS SDK or CLI](#)
- [Use DescribeSubscriptionFilters with an AWS SDK](#)
- [Use GetLogEvents with an AWS SDK or CLI](#)
- [Use GetQueryResults with an AWS SDK](#)
- [Use PutSubscriptionFilter with an AWS SDK](#)
- [Use StartLiveTail with an AWS SDK](#)
- [Use StartQuery with an AWS SDK](#)

Use AssociateKmsKey with an AWS SDK

The following code example shows how to use AssociateKmsKey.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to associate an AWS Key Management Service (AWS KMS) key with
/// an Amazon CloudWatch Logs log group.
/// </summary>
public class AssociateKmsKey
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        string kmsKeyId = "arn:aws:kms:us-west-2:<account-
number>:key/7c9ecce2-38cb-4c4f-9db3-766ee8dd3ad4";
        string groupName = "cloudwatchlogs-example-loggroup";

        var request = new AssociateKmsKeyRequest
        {
            KmsKeyId = kmsKeyId,
            LogGroupName = groupName,
        };

        var response = await client.AssociateKmsKeyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
        {
            Console.WriteLine($"Successfully associated KMS key ID:
{kmsKeyId} with log group: {groupName}.");
        }
        else
        {
            Console.WriteLine("Could not make the association between:
{kmsKeyId} and {groupName}.");
        }
    }
}
```

- For API details, see [AssociateKmsKey](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use CancelExportTask with an AWS SDK

The following code example shows how to use CancelExportTask.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to cancel an Amazon CloudWatch Logs export task.
```

```
/// </summary>
public class CancelExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskId = "exampleTaskId";

        var request = new CancelExportTaskRequest
        {
            TaskId = taskId,
        };

        var response = await client.CancelExportTaskAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{taskId} successfully canceled.");
        }
        else
        {
            Console.WriteLine($"{taskId} could not be canceled.");
        }
    }
}
```

- For API details, see [CancelExportTask](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use CreateExportTask with an AWS SDK

The following code example shows how to use CreateExportTask.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to create an Export Task to export the contents of the Amazon
/// CloudWatch Logs to the specified Amazon Simple Storage Service (Amazon
S3)
/// bucket.
/// </summary>
public class CreateExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskName = "export-task-example";
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string destination = "amzn-s3-demo-bucket";
        var fromTime = 1437584472382;
        var toTime = 1437584472833;

        var request = new CreateExportTaskRequest
        {
            From = fromTime,
            To = toTime,
            TaskName = taskName,
            LogGroupName = logGroupName,
```

```
        Destination = destination,
    };

    var response = await client.CreateExportTaskAsync(request);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"The task, {taskName} with ID: " +
                           $"{response.TaskId} has been created
successfully.");
    }
}
}
```

- For API details, see [CreateExportTask](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use CreateLogGroup with an AWS SDK or CLI

The following code examples show how to use CreateLogGroup.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

```
/// <summary>
/// Shows how to create an Amazon CloudWatch Logs log group.
/// </summary>
public class CreateLogGroup
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new CreateLogGroupRequest
        {
            LogGroupName = logGroupName,
        };

        var response = await client.CreateLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully create log group with ID:
{logGroupName}.");
        }
        else
        {
            Console.WriteLine("Could not create log group.");
        }
    }
}
```

- For API details, see [CreateLogGroup](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

The following command creates a log group named my-logs:

```
aws logs create-log-group --log-group-name my-logs
```

- For API details, see [CreateLogGroup](#) in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

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 { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see [CreateLogGroup](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use CreateLogStream with an AWS SDK or CLI

The following code examples show how to use CreateLogStream.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to create an Amazon CloudWatch Logs stream for a CloudWatch
/// log group.
/// </summary>
public class CreateLogStream
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string logStreamName = "cloudwatchlogs-example-logstream";

        var request = new CreateLogStreamRequest
        {
            LogGroupName = logGroupName,
            LogStreamName = logStreamName,
        };

        var response = await client.CreateLogStreamAsync(request);
    }
}
```

```
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{logStreamName} successfully created for
{logGroupName}.");
        }
        else
        {
            Console.WriteLine("Could not create stream.");
        }
    }
}
```

- For API details, see [CreateLogStream](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

The following command creates a log stream named 20150601 in the log group my-logs:

```
aws logs create-log-stream --log-group-name my-logs --log-stream-name 20150601
```

- For API details, see [CreateLogStream](#) in *AWS CLI Command Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DeleteLogGroup with an AWS SDK or CLI

The following code examples show how to use DeleteLogGroup.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Uses the Amazon CloudWatch Logs Service to delete an existing
/// CloudWatch Logs log group.
/// </summary>
public class DeleteLogGroup
{
    public static async Task Main()
    {
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new DeleteLogGroupRequest
        {
            LogGroupName = logGroupName,
        };

        var response = await client.DeleteLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted CloudWatch log group,
{logGroupName}.");
        }
    }
}
```

- For API details, see [DeleteLogGroup](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

The following command deletes a log group named `my-logs`:

```
aws logs delete-log-group --log-group-name my-logs
```

- For API details, see [DeleteLogGroup](#) in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

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 { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```


- For API details, see [DeleteLogGroup](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DeleteSubscriptionFilter with an AWS SDK

The following code examples show how to use DeleteSubscriptionFilter.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/logs/CloudWatchLogsClient.h>
#include <aws/logs/model/DeleteSubscriptionFilterRequest.h>
#include <iostream>
```

Delete the subscription filter.

```
Aws::CloudWatchLogs::CloudWatchLogsClient cwl;
Aws::CloudWatchLogs::Model::DeleteSubscriptionFilterRequest request;
request.SetFilterName(filter_name);
request.SetLogGroupName(log_group);

auto outcome = cwl.DeleteSubscriptionFilter(request);
if (!outcome.IsSuccess()) {
```

```

        std::cout << "Failed to delete CloudWatch log subscription filter "
        << filter_name << ": " << outcome.GetError().GetMessage() <<
        std::endl;
    } else {
        std::cout << "Successfully deleted CloudWatch logs subscription " <<
        "filter " << filter_name << std::endl;
    }
}

```

- For API details, see [DeleteSubscriptionFilter](#) in *AWS SDK for C++ API 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.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.DeleteSubscriptionFilterRequest;

/**
 * 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 DeleteSubscriptionFilter {
    public static void main(String[] args) {
        final String usage = ""

        Usage:
        <filter> <logGroup>
    }
}

```

```

        Where:
            filter - The name of the subscription filter (for example,
MyFilter).
            logGroup - The name of the log group. (for example, testgroup).
        """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String filter = args[0];
        String logGroup = args[1];
        CloudWatchLogsClient logs = CloudWatchLogsClient.builder()
            .build();

        deleteSubFilter(logs, filter, logGroup);
        logs.close();
    }

    public static void deleteSubFilter(CloudWatchLogsClient logs, String filter,
String logGroup) {
        try {
            DeleteSubscriptionFilterRequest request =
DeleteSubscriptionFilterRequest.builder()
                .filterName(filter)
                .logGroupName(logGroup)
                .build();

            logs.deleteSubscriptionFilter(request);
            System.out.printf("Successfully deleted CloudWatch logs subscription
filter %s", filter);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}

```

- For API details, see [DeleteSubscriptionFilter](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

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 { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see [DeleteSubscriptionFilter](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteSubscriptionFilter](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteSubFilter(
    filter: String?,
    logGroup: String?,
) {
    val request =
```

```
        DeleteSubscriptionFilterRequest {
            filterName = filter
            logGroupName = logGroup
        }

        CloudWatchLogsClient.fromEnvironment { region = "us-west-2" }.use { logs ->
            logs.deleteSubscriptionFilter(request)
            println("Successfully deleted CloudWatch logs subscription filter named
$filter")
        }
    }
}
```

- For API details, see [DeleteSubscriptionFilter](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeExportTasks with an AWS SDK

The following code example shows how to use DescribeExportTasks.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to retrieve a list of information about Amazon CloudWatch
/// Logs export tasks.
```

```
/// </summary>
public class DescribeExportTasks
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        var request = new DescribeExportTasksRequest
        {
            Limit = 5,
        };

        var response = new DescribeExportTasksResponse();

        do
        {
            response = await client.DescribeExportTasksAsync(request);
            response.ExportTasks.ForEach(t =>
            {
                Console.WriteLine($"{t.TaskName} with ID: {t.TaskId} has
status: {t.Status}");
            });
            while (response.NextToken is not null);
        }
    }
}
```

- For API details, see [DescribeExportTasks](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeLogGroups with an AWS SDK or CLI

The following code examples show how to use DescribeLogGroups.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Retrieves information about existing Amazon CloudWatch Logs log groups
/// and displays the information on the console.
/// </summary>
public class DescribeLogGroups
{
    public static async Task Main()
    {
        // Creates a CloudWatch Logs client using the default
        // user. If you need to work with resources in another
        // AWS Region than the one defined for the default user,
        // pass the AWS Region as a parameter to the client constructor.
        var client = new AmazonCloudWatchLogsClient();

        bool done = false;
        string newToken = null;

        var request = new DescribeLogGroupsRequest
        {
            Limit = 5,
        };

        DescribeLogGroupsResponse response;

        do
        {
            if (newToken is not null)
```



```
        {
            request.NextToken = newToken;
        }

        response = await client.DescribeLogGroupsAsync(request);

        response.LogGroups.ForEach(lg =>
        {
            Console.WriteLine($"{lg.LogGroupName} is associated with the
key: {lg.KmsKeyId}.");
            Console.WriteLine($"Created on:
{lg.CreationTime.Date.Date}");
            Console.WriteLine($"Date for this group will be stored for:
{lg.RetentionInDays} days.\n");
        });

        if (response.NextToken is null)
        {
            done = true;
        }
        else
        {
            newToken = response.NextToken;
        }
    }
    while (!done);
}
```

- For API details, see [DescribeLogGroups](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

The following command describes a log group named `my-logs`:

```
aws logs describe-log-groups --log-group-name-prefix my-logs
```

Output:

```
{
  "logGroups": [
    {
      "storedBytes": 0,
      "metricFilterCount": 0,
      "creationTime": 1433189500783,
      "logGroupName": "my-logs",
      "retentionInDays": 5,
      "arn": "arn:aws:logs:us-west-2:0123456789012:log-group:my-logs:*"
    }
  ]
}
```

- For API details, see [DescribeLogGroups](#) in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

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 {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups?.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }
}
```

```
console.log(logGroups);  
return logGroups;  
};
```

- For API details, see [DescribeLogGroups](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeLogStreams with an AWS SDK or CLI

The following code examples show how to use DescribeLogStreams.

CLI

AWS CLI

The following command shows all log streams starting with the prefix 2015 in the log group my-logs:

```
aws logs describe-log-streams --log-group-name my-logs --log-stream-name-  
prefix 2015
```

Output:

```
{  
  "logStreams": [  
    {  
      "creationTime": 1433189871774,  
      "arn": "arn:aws:logs:us-west-2:0123456789012:log-group:my-logs:log-  
stream:20150531",  
      "logStreamName": "20150531",  
      "storedBytes": 0  
    },  
    {  
      "creationTime": 1433189873898,  
      "arn": "arn:aws:logs:us-west-2:0123456789012:log-group:my-logs:log-  
stream:20150601",  
      "logStreamName": "20150601",
```

```
        "storedBytes": 0
    }
]
}
```

- For API details, see [DescribeLogStreams](#) 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](#).

Searches for log streams within a specified log group that match a given prefix.

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class CloudWatchLogsSearch {

    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <logGroupName> <logStreamName>

            Where:
                logGroupName - The name of the log group (for example,
                WeathertopJavaContainerLogs).
                logStreamName - The name of the log stream (for example,
                weathertop-java-stream).
```

```

        pattern - the pattern to use (for example, INFO)

        """;

    if (args.length != 3) {
        System.out.print(usage);
        System.exit(1);
    }

    String logGroupName = args[0] ;
    String logStreamName = args[1] ;
    String pattern = args[2] ;

    CloudWatchLogsClient cwlClient = CloudWatchLogsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    searchLogStreamsAndFilterEvents(cwlClient, logGroupName, logStreamName,
    pattern);
}

/**
 * Searches for log streams with a specific prefix within a log group and
 * filters log events based on a specified pattern.
 *
 * @param cwlClient      the CloudWatchLogsClient used to interact with AWS
 *                        CloudWatch Logs
 * @param logGroupName  the name of the log group to search within
 * @param logStreamPrefix the prefix of the log streams to search for
 * @param pattern        the pattern to filter log events by
 */
public static void searchLogStreamsAndFilterEvents(CloudWatchLogsClient
cwlClient, String logGroupName, String logStreamPrefix, String pattern) {
    DescribeLogStreamsRequest describeLogStreamsRequest =
    DescribeLogStreamsRequest.builder()
        .logGroupName(logGroupName)
        .logStreamNamePrefix(logStreamPrefix)
        .build();

    DescribeLogStreamsResponse describeLogStreamsResponse =
    cwlClient.describeLogStreams(describeLogStreamsRequest);
    List<LogStream> logStreams = describeLogStreamsResponse.logStreams();

    for (LogStream logStream : logStreams) {

```

```

        String logStreamName = logStream.logStreamName();
        System.out.println("Searching in log stream: " + logStreamName);

        FilterLogEventsRequest filterLogEventsRequest =
        FilterLogEventsRequest.builder()
            .logGroupName(logGroupName)
            .logStreamNames(logStreamName)
            .filterPattern(pattern)
            .build();

        FilterLogEventsResponse filterLogEventsResponse =
        cwlClient.filterLogEvents(filterLogEventsRequest);

        for (FilteredLogEvent event : filterLogEventsResponse.events()) {
            System.out.println(event.message());
        }

        System.out.println("-----"); //
        Separator for better readability
    }
}
}

```

Prints metadata about the most recent log stream in a specified log group.

```

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CloudWatchLogQuery {
    public static void main(final String[] args) {
        final String usage = ""
            Usage:
            <logGroupName>

```

```

        Where:
            logGroupName - The name of the log group (for example, /aws/
lambda/ChatAIHandler).
        """;

        if (args.length != 1) {
            System.out.print(usage);
            System.exit(1);
        }

        String logGroupName = "/aws/lambda/ChatAIHandler" ; //args[0];
        CloudWatchLogsClient logsClient = CloudWatchLogsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        describeMostRecentLogStream(logsClient, logGroupName);
    }

    /**
     * Describes and prints metadata about the most recent log stream in the
     specified log group.
     *
     * @param logsClient    the CloudWatchLogsClient used to interact with AWS
     CloudWatch Logs
     * @param logGroupName the name of the log group
     */
    public static void describeMostRecentLogStream(CloudWatchLogsClient
logsClient, String logGroupName) {
        DescribeLogStreamsRequest streamsRequest =
DescribeLogStreamsRequest.builder()
            .logGroupName(logGroupName)
            .orderBy(OrderBy.LAST_EVENT_TIME)
            .descending(true)
            .limit(1)
            .build();

        try {
            DescribeLogStreamsResponse streamsResponse =
logsClient.describeLogStreams(streamsRequest);
            List<LogStream> logStreams = streamsResponse.logStreams();

            if (logStreams.isEmpty()) {
                System.out.println("No log streams found for log group: " +
logGroupName);
            }
        }
    }

```

```
        return;
    }

    LogStream stream = logStreams.get(0);
    System.out.println("Most Recent Log Stream:");
    System.out.println("  Name: " + stream.logStreamName());
    System.out.println("  ARN: " + stream.arn());
    System.out.println("  Creation Time: " + stream.creationTime());
    System.out.println("  First Event Time: " +
stream.firstEventTimestamp());
    System.out.println("  Last Event Time: " +
stream.lastEventTimestamp());
    System.out.println("  Stored Bytes: " + stream.storedBytes());
    System.out.println("  Upload Sequence Token: " +
stream.uploadSequenceToken());

    } catch (CloudWatchLogsException e) {
        System.err.println("Failed to describe log stream: " +
e.awsErrorDetails().errorMessage());
    }
}
}
```

- For API details, see [DescribeLogStreams](#) in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeSubscriptionFilters with an AWS SDK

The following code examples show how to use DescribeSubscriptionFilters.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/logs/CloudWatchLogsClient.h>
#include <aws/logs/model/DescribeSubscriptionFiltersRequest.h>
#include <aws/logs/model/DescribeSubscriptionFiltersResult.h>
#include <iostream>
#include <iomanip>
```

List the subscription filters.

```
Aws::CloudWatchLogs::CloudWatchLogsClient cwl;
Aws::CloudWatchLogs::Model::DescribeSubscriptionFiltersRequest request;
request.SetLogGroupName(log_group);
request.SetLimit(1);

bool done = false;
bool header = false;
while (!done) {
    auto outcome = cwl.DescribeSubscriptionFilters(
        request);
    if (!outcome.IsSuccess()) {
        std::cout << "Failed to describe CloudWatch subscription filters
"
        << "for log group " << log_group << ": " <<
        outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header) {
```

```

        std::cout << std::left << std::setw(32) << "Name" <<
            std::setw(64) << "FilterPattern" << std::setw(64) <<
            "DestinationArn" << std::endl;
        header = true;
    }

    const auto &filters = outcome.GetResult().GetSubscriptionFilters();
    for (const auto &filter : filters) {
        std::cout << std::left << std::setw(32) <<
            filter.GetFilterName() << std::setw(64) <<
            filter.GetFilterPattern() << std::setw(64) <<
            filter.GetDestinationArn() << std::endl;
    }

    const auto &next_token = outcome.GetResult().GetNextToken();
    request.SetNextToken(next_token);
    done = next_token.empty();
}

```

- For API details, see [DescribeSubscriptionFilters](#) in *AWS SDK for C++ API 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.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.DescribeSubscriptionFiltersRequest;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.DescribeSubscriptionFiltersResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.SubscriptionFilter;

/**

```

```
* 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 DescribeSubscriptionFilters {
    public static void main(String[] args) {

        final String usage = ""

            Usage:
            <logGroup>

            Where:
            logGroup - A log group name (for example, myloggroup).
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String logGroup = args[0];
        CloudWatchLogsClient logs = CloudWatchLogsClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        describeFilters(logs, logGroup);
        logs.close();
    }

    public static void describeFilters(CloudWatchLogsClient logs, String
logGroup) {
        try {
            boolean done = false;
            String newToken = null;

            while (!done) {
                DescribeSubscriptionFiltersResponse response;
                if (newToken == null) {
```

```

        DescribeSubscriptionFiltersRequest request =
DescribeSubscriptionFiltersRequest.builder()
            .logGroupName(logGroup)
            .limit(1).build();

        response = logs.describeSubscriptionFilters(request);
    } else {
        DescribeSubscriptionFiltersRequest request =
DescribeSubscriptionFiltersRequest.builder()
            .nextToken(newToken)
            .logGroupName(logGroup)
            .limit(1).build();
        response = logs.describeSubscriptionFilters(request);
    }

    for (SubscriptionFilter filter : response.subscriptionFilters())
    {
        System.out.printf("Retrieved filter with name %s, " +
"pattern %s " + "and destination arn %s",
            filter.filterName(),
            filter.filterPattern(),
            filter.destinationArn());
    }

    if (response.nextToken() == null) {
        done = true;
    } else {
        newToken = response.nextToken();
    }
}

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
System.out.printf("Done");
}
}

```

- For API details, see [DescribeSubscriptionFilters](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

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 { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see [DescribeSubscriptionFilters](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cw1.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DescribeSubscriptionFilters](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun describeFilters(logGroup: String) {
    val request =
        DescribeSubscriptionFiltersRequest {
            logGroupName = logGroup
            limit = 1
        }
}
```

```
CloudWatchLogsClient.fromEnvironment { region = "us-west-2" }.use { cwlClient
->
    val response = cwlClient.describeSubscriptionFilters(request)
    response.subscriptionFilters?.forEach { filter ->
        println("Retrieved filter with name ${filter.filterName} pattern
${filter.filterPattern} and destination ${filter.destinationArn}")
    }
}
```

- For API details, see [DescribeSubscriptionFilters](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetLogEvents with an AWS SDK or CLI

The following code examples show how to use GetLogEvents.

CLI

AWS CLI

The following command retrieves log events from a log stream named 20150601 in the log group my-logs:

```
aws logs get-log-events --log-group-name my-logs --log-stream-name 20150601
```

Output:

```
{
  "nextForwardToken":
  "f/31961209122447488583055879464742346735121166569214640130",
  "events": [
    {
      "ingestionTime": 1433190494190,
      "timestamp": 1433190184356,
      "message": "Example Event 1"
    },
  ],
}
```

```
{
  "ingestionTime": 1433190516679,
  "timestamp": 1433190184356,
  "message": "Example Event 1"
},
{
  "ingestionTime": 1433190494190,
  "timestamp": 1433190184358,
  "message": "Example Event 2"
}
],
"nextBackwardToken":
"b/31961209122358285602261756944988674324553373268216709120"
}
```

- For API details, see [GetLogEvents](#) 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.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import
  software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import
  software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.GetLogEventsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.GetLogEventsResponse;

import java.time.Instant;
import java.time.temporal.ChronoUnit;

/**
 * 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 GetLogEvents {

    public static void main(String[] args) {

        final String usage = ""

            Usage:
                <logGroupName> <logStreamName>

            Where:
                logGroupName - The name of the log group (for example,
myloggroup).
                logStreamName - The name of the log stream (for example,
mystream).

            "";

        // if (args.length != 2) {
        //     System.out.print(usage);
        //     System.exit(1);
//     }

        String logGroupName = "WeathertopJavaContainerLogs" ; //args[0];
        String logStreamName = "weathertop-java-stream" ; //args[1];

        Region region = Region.US_EAST_1 ;
        CloudWatchLogsClient cloudWatchLogsClient =
CloudWatchLogsClient.builder()
            .region(region)
            .build();

        getCWLogEvents(cloudWatchLogsClient, logGroupName, logStreamName);
        cloudWatchLogsClient.close();
    }

    public static void getCWLogEvents(CloudWatchLogsClient cloudWatchLogsClient,
        String logGroupName,

```

```

        String logStreamPrefix) {

    try {
        // First, find the exact log stream name
        DescribeLogStreamsRequest describeRequest =
DescribeLogStreamsRequest.builder()
            .logGroupName(logGroupName)
            .logStreamNamePrefix(logStreamPrefix)
            .limit(1) // get the first matching stream
            .build();

        DescribeLogStreamsResponse describeResponse =
cloudWatchLogsClient.describeLogStreams(describeRequest);

        if (describeResponse.getLogStreams().isEmpty()) {
            System.out.println("No matching log streams found for prefix: " +
logStreamPrefix);
            return;
        }

        String exactLogStreamName =
describeResponse.getLogStreams().get(0).getLogStreamName();
        System.out.println("Using exact log stream: " + exactLogStreamName);

        long startTime = Instant.now().minus(7,
ChronoUnit.DAYS).toEpochMilli();
        long endTime = Instant.now().toEpochMilli();

        GetLogEventsRequest getLogEventsRequest =
GetLogEventsRequest.builder()
            .logGroupName(logGroupName)
            .logStreamName(exactLogStreamName) // <-- exact name, not
prefix
            .startTime(startTime)
            .endTime(endTime)
            .startFromHead(true)
            .build();

        GetLogEventsResponse response =
cloudWatchLogsClient.getLogEvents(getLogEventsRequest);

        if (response.getEvents().isEmpty()) {
            System.out.println("No log events found in the past 7 days.");
        } else {
            response.getEvents().forEach(e -> System.out.println(e.getMessage()));
        }
    }
}

```

```
        }

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see [GetLogEvents](#) in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetQueryResults with an AWS SDK

The following code examples show how to use `GetQueryResults`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Run a large query](#)

.NET

SDK for .NET (v4)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Gets the results of a CloudWatch Logs Insights query.
/// </summary>
/// <param name="queryId">The ID of the query.</param>
/// <returns>The query results response.</returns>
```

```
public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string
queryId)
{
    try
    {
        var request = new GetQueryResultsRequest
        {
            QueryId = queryId
        };

        var response = await
        _amazonCloudWatchLogs.GetQueryResultsAsync(request);
        return response;
    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError($"Query not found: {ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while getting query results:
{ex.Message}");
        return null;
    }
}
```

- For API details, see [GetQueryResults](#) in *AWS SDK for .NET API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
```

```

    * @param {string} queryId
    */
    _getQueryResults(queryId) {
        return this.client.send(new GetQueryResultsCommand({ queryId }));
    }

```

- For API details, see [GetQueryResults](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

def _wait_for_query_results(self, client, query_id):
    """
    Waits for the query to complete and retrieves the results.

    :param query_id: The ID of the initiated query.
    :type query_id: str
    :return: A list containing the results of the query.
    :rtype: list
    """
    while True:
        time.sleep(1)
        results = client.get_query_results(queryId=query_id)
        if results["status"] in [
            "Complete",
            "Failed",
            "Cancelled",
            "Timeout",
            "Unknown",
        ]:
            return results.get("results", [])

```

- For API details, see [GetQueryResults](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use PutSubscriptionFilter with an AWS SDK

The following code examples show how to use PutSubscriptionFilter.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/logs/CloudWatchLogsClient.h>
#include <aws/logs/model/PutSubscriptionFilterRequest.h>
#include <aws/core/utils/Outcome.h>
#include <iostream>
```

Create the subscription filter.

```
Aws::CloudWatchLogs::CloudWatchLogsClient cwl;
Aws::CloudWatchLogs::Model::PutSubscriptionFilterRequest request;
request.SetFilterName(filter_name);
request.SetFilterPattern(filter_pattern);
request.SetLogGroupName(log_group);
request.SetDestinationArn(dest_arn);
auto outcome = cwl.PutSubscriptionFilter(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch logs subscription filter "
```

```

        << filter_name << ": " << outcome.GetError().GetMessage() <<
        std::endl;
    }
    else
    {
        std::cout << "Successfully created CloudWatch logs subscription " <<
        "filter " << filter_name << std::endl;
    }

```

- For API details, see [PutSubscriptionFilter](#) in *AWS SDK for C++ API 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.cloudwatchlogs.CloudWatchLogsClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.CloudWatchLogsException;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.PutSubscriptionFilterRequest;

/**
 * Before running this code example, you need to grant permission to CloudWatch
 * Logs the right to execute your Lambda function.
 * To perform this task, you can use this CLI command:
 *
 * aws lambda add-permission --function-name "lamda1" --statement-id "lamda1"
 * --principal "logs.us-west-2.amazonaws.com" --action "lambda:InvokeFunction"
 * --source-arn "arn:aws:logs:us-west-2:111111111111:log-group:testgroup:*"
 * --source-account "111111111111"
 *
 * Make sure you replace the function name with your function name and replace
 * '111111111111' with your account details.
 * For more information, see "Subscription Filters with AWS Lambda" in the

```

```

* Amazon CloudWatch Logs Guide.
*
*
* Also, 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 PutSubscriptionFilter {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <filter> <pattern> <logGroup> <functionArn> \s

            Where:
                filter - A filter name (for example, myfilter).
                pattern - A filter pattern (for example, ERROR).
                logGroup - A log group name (testgroup).
                functionArn - An AWS Lambda function ARN (for example,
arn:aws:lambda:us-west-2:111111111111:function:lambda1) .
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String filter = args[0];
        String pattern = args[1];
        String logGroup = args[2];
        String functionArn = args[3];
        Region region = Region.US_WEST_2;
        CloudWatchLogsClient cw1 = CloudWatchLogsClient.builder()
            .region(region)
            .build();

        putSubFilters(cw1, filter, pattern, logGroup, functionArn);
        cw1.close();
    }
}

```



```
}

public static void putSubFilters(CloudWatchLogsClient cwl,
    String filter,
    String pattern,
    String logGroup,
    String functionArn) {

    try {
        PutSubscriptionFilterRequest request =
        PutSubscriptionFilterRequest.builder()
            .filterName(filter)
            .filterPattern(pattern)
            .logGroupName(logGroup)
            .destinationArn(functionArn)
            .build();

        cwl.putSubscriptionFilter(request);
        System.out.printf(
            "Successfully created CloudWatch logs subscription filter
%s",
            filter);

    } catch (CloudWatchLogsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [PutSubscriptionFilter](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

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 { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
    SubscriptionFilters.html
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
    FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

    // The name of the log group. Messages in this group matching the filter
    pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see [PutSubscriptionFilter](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutSubscriptionFilter](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use StartLiveTail with an AWS SDK

The following code examples show how to use StartLiveTail.

.NET

SDK for .NET

Include the required files.

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

Start the Live Tail session.

```
var client = new AmazonCloudWatchLogsClient();
var request = new StartLiveTailRequest
{
    LogGroupIdentifiers = logGroupIdentifiers,
    LogStreamNames = logStreamNames,
    LogEventFilterPattern = filterPattern,
};

var response = await client.StartLiveTailAsync(request);

// Catch if request fails
if (response.HttpStatusCode != System.Net.HttpStatusCode.OK)
{
    Console.WriteLine("Failed to start live tail session");
    return;
}
```

You can handle the events from the Live Tail session in two ways:

```
/* Method 1
 * 1). Asynchronously loop through the event stream
 * 2). Set a timer to dispose the stream and stop the Live Tail
session at the end.
*/
```

```

var eventStream = response.ResponseStream;
var task = Task.Run(() =>
{
    foreach (var item in eventStream)
    {
        if (item is LiveTailSessionUpdate liveTailSessionUpdate)
        {
            foreach (var sessionResult in
liveTailSessionUpdate.SessionResults)
            {
                Console.WriteLine("Message : {0}",
sessionResult.Message);
            }
        }
        if (item is LiveTailSessionStart)
        {
            Console.WriteLine("Live Tail session started");
        }
        // On-stream exceptions are processed here
        if (item is CloudWatchLogsEventStreamException)
        {
            Console.WriteLine($"ERROR: {item}");
        }
    }
});
// Close the stream to stop the session after a timeout
if (!task.Wait(TimeSpan.FromSeconds(10))){
    eventStream.Dispose();
    Console.WriteLine("End of line");
}

```

```

/* Method 2
* 1). Add event handlers to each event variable
* 2). Start processing the stream and wait for a timeout using
AutoResetEvent
*/
AutoResetEvent endEvent = new AutoResetEvent(false);
var eventStream = response.ResponseStream;
using (eventStream) // automatically disposes the stream to stop the
session after execution finishes
{
    eventStream.SessionStartReceived += (sender, e) =>

```

```
        {
            Console.WriteLine("LiveTail session started");
        };
        eventStream.SessionUpdateReceived += (sender, e) =>
        {
            foreach (LiveTailSessionLogEvent logEvent in
e.EventStreamEvent.SessionResults){
                Console.WriteLine("Message: {0}", logEvent.Message);
            }
        };
        // On-stream exceptions are captured here
        eventStream.ExceptionReceived += (sender, e) =>
        {
            Console.WriteLine($"ERROR:
{e.EventStreamException.Message}");
        };

        eventStream.StartProcessing();
        // Stream events for this amount of time.
        endEvent.WaitOne(TimeSpan.FromSeconds(10));
        Console.WriteLine("End of line");
    }
}
```

- For API details, see [StartLiveTail](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

Include the required files.

```
import (
    "context"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)
```

Handle the events from the Live Tail session.

```
func handleEventStreamAsync(stream *cloudwatchlogs.StartLiveTailEventStream) {
    eventsChan := stream.Events()
    for {
        event := <-eventsChan
        switch e := event.(type) {
        case *types.StartLiveTailResponseStreamMemberSessionStart:
            log.Println("Received SessionStart event")
        case *types.StartLiveTailResponseStreamMemberSessionUpdate:
            for _, logEvent := range e.Value.SessionResults {
                log.Println(*logEvent.Message)
            }
        default:
            // Handle on-stream exceptions
            if err := stream.Err(); err != nil {
                log.Fatalf("Error occurred during streaming: %v", err)
            } else if event == nil {
                log.Println("Stream is Closed")
                return
            } else {
                log.Fatalf("Unknown event type: %T", e)
            }
        }
    }
}
```

Start the Live Tail session.

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic("configuration error, " + err.Error())
}
client := cloudwatchlogs.NewFromConfig(cfg)

request := &cloudwatchlogs.StartLiveTailInput{
    LogGroupIdentifiers: logGroupIdentifiers,
    LogStreamNames:      logStreamNames,
    LogEventFilterPattern: logEventFilterPattern,
}

response, err := client.StartLiveTail(context.TODO(), request)
```

```
// Handle pre-stream Exceptions
if err != nil {
    log.Fatalf("Failed to start streaming: %v", err)
}

// Start a Goroutine to handle events over stream
stream := response.GetStream()
go handleEventStreamAsync(stream)
```

Stop the Live Tail session after a period of time has elapsed.

```
// Close the stream (which ends the session) after a timeout
time.Sleep(10 * time.Second)
stream.Close()
log.Println("Event stream closed")
```

- For API details, see [StartLiveTail](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Include the required files.

```
import io.reactivex.FlowableSubscriber;
import io.reactivex.annotations.NonNull;
import org.reactivestreams.Subscription;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsAsyncClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.LiveTailSessionLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.LiveTailSessionStart;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.LiveTailSessionUpdate;
import software.amazon.awssdk.services.cloudwatchlogs.model.StartLiveTailRequest;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.StartLiveTailResponseHandler;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.CloudWatchLogsException;
```



```
import
    software.amazon.awssdk.services.cloudwatchlogs.model.StartLiveTailResponseStream;

import java.util.Date;
import java.util.List;
import java.util.concurrent.atomic.AtomicReference;
```

Handle the events from the Live Tail session.

```
private static StartLiveTailResponseHandler
getStartLiveTailResponseStreamHandler(
    AtomicReference<Subscription> subscriptionAtomicReference) {
    return StartLiveTailResponseHandler.builder()
        .onResponse(r -> System.out.println("Received initial response"))
        .onError(throwable -> {
            CloudWatchLogsException e = (CloudWatchLogsException)
throwable.getCause();
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        })
        .subscriber(() -> new FlowableSubscriber<>() {
            @Override
            public void onSubscribe(@NonNull Subscription s) {
                subscriptionAtomicReference.set(s);
                s.request(Long.MAX_VALUE);
            }

            @Override
            public void onNext(StartLiveTailResponseStream event) {
                if (event instanceof LiveTailSessionStart) {
                    LiveTailSessionStart sessionStart =
(LiveTailSessionStart) event;
                    System.out.println(sessionStart);
                } else if (event instanceof LiveTailSessionUpdate) {
                    LiveTailSessionUpdate sessionUpdate =
(LiveTailSessionUpdate) event;
                    List<LiveTailSessionLogEvent> logEvents =
sessionUpdate.sessionResults();
                    logEvents.forEach(e -> {
                        long timestamp = e.timestamp();
                        Date date = new Date(timestamp);
                        System.out.println "[" + date + " ] " + e.message());
                    });
                }
            }
        })
    }
```

```

        });
    } else {
        throw CloudWatchLogsException.builder().message("Unknown
event type").build();
    }
}

@Override
public void onError(Throwable throwable) {
    System.out.println(throwable.getMessage());
    System.exit(1);
}

@Override
public void onComplete() {
    System.out.println("Completed Streaming Session");
}
})
.build();
}

```

Start the Live Tail session.

```

CloudWatchLogsAsyncClient cloudWatchLogsAsyncClient =
    CloudWatchLogsAsyncClient.builder()
        .credentialsProvider(ProfileCredentialsProvider.create())
        .build();

StartLiveTailRequest request =
    StartLiveTailRequest.builder()
        .logGroupIdentifiers(logGroupIdentifiers)
        .logStreamNames(logStreamNames)
        .logEventFilterPattern(logEventFilterPattern)
        .build();

/* Create a reference to store the subscription */
final AtomicReference<Subscription> subscriptionAtomicReference = new
AtomicReference<>(null);

cloudWatchLogsAsyncClient.startLiveTail(request,
getStartLiveTailResponseStreamHandler(subscriptionAtomicReference));

```

Stop the Live Tail session after a period of time has elapsed.

```
/* Set a timeout for the session and cancel the subscription. This will:  
 * 1). Close the stream  
 * 2). Stop the Live Tail session  
 */  
try {  
    Thread.sleep(10000);  
} catch (InterruptedException e) {  
    throw new RuntimeException(e);  
}  
if (subscriptionAtomicReference.get() != null) {  
    subscriptionAtomicReference.get().cancel();  
    System.out.println("Subscription to stream closed");  
}
```

- For API details, see [StartLiveTail](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Include the required files.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-cloudwatch-logs";
```

Handle the events from the Live Tail session.

```
async function handleResponseAsync(response) {  
    try {  
        for await (const event of response.responseStream) {  
            if (event.sessionStart !== undefined) {  
                console.log(event.sessionStart);  
            } else if (event.sessionUpdate !== undefined) {  
                for (const logEvent of event.sessionUpdate.sessionResults) {  
                    const timestamp = logEvent.timestamp;  
                    const date = new Date(timestamp);  
                    console.log "[" + date + "]" + logEvent.message);  
                }  
            }  
        }  
    }  
}
```

```
    }
  } else {
    console.error("Unknown event type");
  }
}
} catch (err) {
  // On-stream exceptions are captured here
  console.error(err)
}
}
```

Start the Live Tail session.

```
const client = new CloudWatchLogsClient();

const command = new StartLiveTailCommand({
  logGroupIdentifiers: logGroupIdentifiers,
  logStreamNames: logStreamNames,
  logEventFilterPattern: filterPattern
});
try{
  const response = await client.send(command);
  handleResponseAsync(response);
} catch (err){
  // Pre-stream exceptions are captured here
  console.log(err);
}
```

Stop the Live Tail session after a period of time has elapsed.

```
/* Set a timeout to close the client. This will stop the Live Tail session.
*/
setTimeout(function() {
  console.log("Client timeout");
  client.destroy();
}, 10000);
```

- For API details, see [StartLiveTail](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Include the required files.

```
import aws.sdk.kotlin.services.cloudwatchlogs.CloudWatchLogsClient
import aws.sdk.kotlin.services.cloudwatchlogs.model.StartLiveTailRequest
import aws.sdk.kotlin.services.cloudwatchlogs.model.StartLiveTailResponseStream
import kotlinx.coroutines.flow.takeWhile
```

Start the Live Tail session.

```
val client = CloudWatchLogsClient.fromEnvironment()

val request = StartLiveTailRequest {
    logGroupIdentifiers = logGroupIdentifiersVal
    logStreamNames = logStreamNamesVal
    logEventFilterPattern = logEventFilterPatternVal
}

val startTime = System.currentTimeMillis()

try {
    client.startLiveTail(request) { response ->
        val stream = response.responseStream
        if (stream != null) {
            /* Set a timeout to unsubscribe from the flow. This will:
             * 1). Close the stream
             * 2). Stop the Live Tail session
             */
            stream.takeWhile { System.currentTimeMillis() - startTime <
10000 }.collect { value ->
                if (value is StartLiveTailResponseStream.SessionStart) {
                    println(value.asSessionStart())
                } else if (value is
StartLiveTailResponseStream.SessionUpdate) {
                    for (e in value.asSessionUpdate().sessionResults!!) {
                        println(e)
                    }
                } else {
                    throw IllegalArgumentException("Unknown event type")
                }
            }
        }
    }
```

```

        }
    }
    } else {
        throw IllegalArgumentException("No response stream")
    }
}
} catch (e: Exception) {
    println("Exception occurred during StartLiveTail: $e")
    System.exit(1)
}

```

- For API details, see [StartLiveTail](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Include the required files.

```

import boto3
import time
from datetime import datetime

```

Start the Live Tail session.

```

# Initialize the client
client = boto3.client('logs')

start_time = time.time()

try:
    response = client.start_live_tail(
        logGroupIdentifiers=log_group_identifiers,
        logStreamNames=log_streams,
        logEventFilterPattern=filter_pattern
    )
    event_stream = response['responseStream']
    # Handle the events streamed back in the response
    for event in event_stream:
        # Set a timeout to close the stream.

```

```

        # This will end the Live Tail session.
        if (time.time() - start_time >= 10):
            event_stream.close()
            break
        # Handle when session is started
        if 'sessionStart' in event:
            session_start_event = event['sessionStart']
            print(session_start_event)
        # Handle when log event is given in a session update
        elif 'sessionUpdate' in event:
            log_events = event['sessionUpdate']['sessionResults']
            for log_event in log_events:
                print('[{date}]'
{log}'.format(date=datetime.fromtimestamp(log_event['timestamp']/1000),log=log_event['me
            else:
                # On-stream exceptions are captured here
                raise RuntimeError(str(event))
    except Exception as e:
        print(e)

```

- For API details, see [StartLiveTail](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use StartQuery with an AWS SDK

The following code examples show how to use StartQuery.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Run a large query](#)

.NET

SDK for .NET (v4)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Starts a CloudWatch Logs Insights query.
/// </summary>
/// <param name="logGroupName">The name of the log group to query.</param>
/// <param name="queryString">The CloudWatch Logs Insights query string.</
param>
/// <param name="startTime">The start time for the query (seconds since
epoch).</param>
/// <param name="endTime">The end time for the query (seconds since epoch).</
param>
/// <param name="limit">The maximum number of results to return.</param>
/// <returns>The query ID if successful, null otherwise.</returns>
public async Task<string?> StartQueryAsync(
    string logGroupName,
    string queryString,
    long startTime,
    long endTime,
    int limit = 10000)
{
    try
    {
        var request = new StartQueryRequest
        {
            LogGroupName = logGroupName,
            QueryString = queryString,
            StartTime = startTime,
            EndTime = endTime,
            Limit = limit
        };

        var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
        return response.QueryId;
    }
}
```



```

    }
    catch (InvalidParameterException ex)
    {
        _logger.LogError($"Invalid parameter for query: {ex.Message}");
        return null;
    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError($"Log group not found: {ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while starting query:
{ex.Message}");
        return null;
    }
}

```

- For API details, see [StartQuery](#) in *AWS SDK for .NET API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
        return await this.client.send(

```

```

        new StartQueryCommand({
            logGroupNames: this.logGroupNames,
            queryString: "fields @timestamp, @message | sort @timestamp asc",
            startTime: startDate.valueOf(),
            endTime: endDate.valueOf(),
            limit: maxLogs,
        }),
    );
} catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
    }

    throw err;
}
}

```

- For API details, see [StartQuery](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

def perform_query(self, date_range):
    """
    Performs the actual CloudWatch log query.

    :param date_range: A tuple representing the start and end datetime for
    the query.
    :type date_range: tuple
    :return: A list containing the query results.
    """

```

```

        :rtype: list
        """
        client = boto3.client("logs")
        try:
            try:
                start_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])
                )
                end_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
                )
                response = client.start_query(
                    logGroupName=self.log_group,
                    startTime=start_time,
                    endTime=end_time,
                    queryString=self.query_string,
                    limit=self.limit,
                )
                query_id = response["queryId"]
            except client.exceptions.ResourceNotFoundException as e:
                raise DateOutOfBoundsError(f"Resource not found: {e}")
            while True:
                time.sleep(1)
                results = client.get_query_results(queryId=query_id)
                if results["status"] in [
                    "Complete",
                    "Failed",
                    "Cancelled",
                    "Timeout",
                    "Unknown",
                ]:
                    return results.get("results", [])
            except DateOutOfBoundsError:
                return []

def _initiate_query(self, client, date_range, max_logs):
    """
    Initiates the CloudWatch logs query.

    :param date_range: A tuple representing the start and end datetime for
    the query.
    :type date_range: tuple

```

```

:param max_logs: The maximum number of logs to retrieve.
:type max_logs: int
:return: The query ID as a string.
:rtype: str
"""
try:
    start_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])
    )
    end_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
    )
    response = client.start_query(
        logGroupName=self.log_group,
        startTime=start_time,
        endTime=end_time,
        queryString=self.query_string,
        limit=max_logs,
    )
    return response["queryId"]
except client.exceptions.ResourceNotFoundException as e:
    raise DateOutOfBoundsError(f"Resource not found: {e}")

```

- For API details, see [StartQuery](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for CloudWatch Logs using AWS SDKs

The following code examples show you how to implement common scenarios in CloudWatch Logs with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within CloudWatch Logs or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- [Use CloudWatch Logs to run a large query](#)
- [Use scheduled events to invoke a Lambda function](#)

Use CloudWatch Logs to run a large query

The following code examples show how to use CloudWatch Logs to query more than 10,000 records.

.NET

SDK for .NET (v4)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

This is the main workflow that demonstrates the large query scenario.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
```

Before running this .NET code example, set up your development environment, including your credentials.

This .NET code example performs the following tasks for the CloudWatch Logs Large Query workflow:

1. Prepare the Application:
 - Prompt the user to deploy CloudFormation stack and generate sample logs.
 - Deploy the CloudFormation template for resource creation.
 - Generate 50,000 sample log entries using CloudWatch Logs API.
 - Wait 5 minutes for logs to be fully ingested.
2. Execute Large Query:
 - Perform recursive queries to retrieve all logs using binary search.
 - Display progress for each query executed.
 - Show total execution time and logs found.
3. Clean up:
 - Prompt the user to delete the CloudFormation stack and all resources.
 - Destroy the CloudFormation stack and wait until removed.

```
*/

public static ILogger<LargeQueryWorkflow> _logger = null!;
public static CloudWatchLogsWrapper _wrapper = null!;
public static IAmazonCloudFormation _amazonCloudFormation = null!;

private static string _logGroupName = "/workflows/cloudwatch-logs/large-
query";
private static string _logStreamName = "stream1";
private static long _queryStartDate;
private static long _queryEndDate;

public static bool _interactive = true;
public static string _stackName = "CloudWatchLargeQueryStack";
private static string _stackResourcePath = "../..../..../..../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

public static async Task Main(string[] args)
{
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter("Microsoft", LogLevel.Information))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatchLogs>())
```

```

        .AddAWSService<IAmazonCloudFormation>()
        .AddTransient<CloudWatchLogsWrapper>()
    )
    .Build();

if (!_interactive)
{
    _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<LargeQueryWorkflow>();

    _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
    _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
}

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the CloudWatch Logs Large Query
Scenario.");
Console.WriteLine(new string('-', 80));
Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
Console.WriteLine("CloudWatch Logs using recursive binary search to
retrieve more than");
Console.WriteLine("the 10,000 result limit.");
Console.WriteLine();

try
{
    Console.WriteLine(new string('-', 80));
    var prepareSuccess = await PrepareApplication();
    Console.WriteLine(new string('-', 80));

    if (prepareSuccess)
    {
        Console.WriteLine(new string('-', 80));
        await ExecuteLargeQuery();
        Console.WriteLine(new string('-', 80));
    }

    Console.WriteLine(new string('-', 80));
    await Cleanup();
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)

```

```
        {
            _logger.LogError(ex, "There was a problem with the scenario,
initiating cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query
Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to
retrieve more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
```



```

        {
            _logger.LogError(ex, "There was a problem with the scenario,
initiating cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();

        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack
resource .yaml file (or press Enter for default '{_stackResourcePath}') ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)

```

```

        {
            Console.WriteLine();
            Console.WriteLine("Generating 50,000 sample log entries...");
            var generateSuccess = await GenerateSampleLogs();

            if (generateSuccess)
            {
                Console.WriteLine();
                Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                await WaitWithCountdown(300);

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
    }
else
{
    _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
    _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

    var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
    var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

    _queryStartDate = startDateMs / 1000;
    _queryEndDate = endDateMs / 1000;

    Console.WriteLine("Application preparation complete.");
    return true;
}
}
catch (Exception ex)
{
    _logger.LogError(ex, "An error occurred while preparing the
application.");
}

Console.WriteLine("Application preparation failed.");
return false;

```

```
}

/// <summary>
/// Deploys the CloudFormation stack with the necessary resources.
/// </summary>
/// <param name="stackName">The name of the CloudFormation stack.</param>
/// <returns>True if the stack was deployed successfully.</returns>
private static async Task<bool> DeployCloudFormationStack(string stackName)
{
    Console.WriteLine($"\\nDeploying CloudFormation stack: {stackName}");

    try
    {
        var request = new CreateStackRequest
        {
            StackName = stackName,
            TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
        };

        var response = await _amazonCloudFormation.CreateStackAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

            bool stackCreated = await
WaitForStackCompletion(response.StackId);

            if (stackCreated)
            {
                Console.WriteLine("CloudFormation stack created
successfully.");
                return true;
            }
            else
            {
                _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                return false;
            }
        }
        else
        {

```

```

        _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
        return false;
    }
}
catch (AlreadyExistsException)
{
    _logger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
    var newStackName = PromptUserForStackName();
    return await DeployCloudFormationStack(newStackName);
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
    return false;
}
}

/// <summary>
/// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
/// </summary>
/// <param name="stackId">The ID of the CloudFormation stack.</param>
/// <returns>True if the stack was created successfully.</returns>
private static async Task<bool> WaitForStackCompletion(string stackId)
{
    int retryCount = 0;
    const int maxRetries = 30;
    const int retryDelay = 10000;

    while (retryCount < maxRetries)
    {
        var describeStacksRequest = new DescribeStacksRequest
        {
            StackName = stackId
        };

        var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

        if (describeStacksResponse.Stacks.Count > 0)
        {

```

```

        if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
        {
            return true;
        }
        if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
            describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
        {
            return false;
        }
    }

    Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
    await Task.Delay(retryDelay);
    retryCount++;
}

_logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
return false;
}

/// <summary>
/// Generates sample logs directly using CloudWatch Logs API.
/// Creates 50,000 log entries spanning 5 minutes.
/// </summary>
/// <returns>True if logs were generated successfully.</returns>
private static async Task<bool> GenerateSampleLogs()
{
    const int totalEntries = 50000;
    const int entriesPerBatch = 10000;
    const int fiveMinutesMs = 5 * 60 * 1000;

    try
    {
        // Calculate timestamps
        var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
        var timestampIncrement = fiveMinutesMs / totalEntries;

        Console.WriteLine($"Generating {totalEntries} log entries...");
    }
}

```

```

        var entryCount = 0;
        var currentTimestamp = startTimeMs;
        var numBatches = totalEntries / entriesPerBatch;

        // Generate and upload logs in batches
        for (int batchNum = 0; batchNum < numBatches; batchNum++)
        {
            var logEvents = new List<InputLogEvent>();

            for (int i = 0; i < entriesPerBatch; i++)
            {
                logEvents.Add(new InputLogEvent
                {
                    Timestamp =
                        DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                    Message = $"Entry {entryCount}"
                });

                entryCount++;
                currentTimestamp += timestampIncrement;
            }

            // Upload batch
            var success = await _wrapper.PutLogEventsAsync(_logGroupName,
                _logStreamName, logEvents);
            if (!success)
            {
                _logger.LogError($"Failed to upload batch {batchNum + 1}/{
numBatches}");
                return false;
            }

            Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
        }

        // Set query date range (convert milliseconds to seconds for query
API)
        _queryStartDate = startTimeMs / 1000;
        _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

        Console.WriteLine($"Query start date:
{DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-
ddTHH:mm:ss.fffZ}");

```

```

        Console.WriteLine($"Query end date:
{DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
        Console.WriteLine($"Successfully uploaded {totalEntries} log
entries");

        return true;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while generating sample
logs.");
        return false;
    }
}

/// <summary>
/// Executes the large query workflow.
/// </summary>
public static async Task ExecuteLargeQuery()
{
    Console.WriteLine("Starting recursive query to retrieve all logs...");
    Console.WriteLine();

    var queryLimit = PromptUserForInteger("Enter the query limit (max 10000)
", 10000);
    if (queryLimit > 10000) queryLimit = 10000;

    var queryString = "fields @timestamp, @message | sort @timestamp asc";

    var stopwatch = Stopwatch.StartNew();
    var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
    stopwatch.Stop();

    Console.WriteLine();
    Console.WriteLine($"Queries finished in
{stopwatch.Elapsed.TotalSeconds:F3} seconds.");
    Console.WriteLine($"Total logs found: {allResults.Count}");

    // Check for duplicates
    Console.WriteLine();
    Console.WriteLine("Checking for duplicate logs...");
    var duplicates = FindDuplicateLogs(allResults);
    if (duplicates.Count > 0)

```

```

    {
        Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
        Console.WriteLine("Duplicate entries (showing first 10):");
        foreach (var dup in duplicates.Take(10))
        {
            Console.WriteLine($" [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
        }

        var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count -
1);

        Console.WriteLine($"Unique logs: {uniqueCount}");
    }
    else
    {
        Console.WriteLine("No duplicates found. All logs are unique.");
    }
    Console.WriteLine();

    var viewSample = !_interactive || GetYesNoResponse("Would you like to see
a sample of the logs? (y/n) ");
    if (viewSample)
    {
        Console.WriteLine();
        Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
        for (int i = 0; i < Math.Min(10, allResults.Count); i++)
        {
            var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
            var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
            Console.WriteLine($"[{timestamp}] {message}");
        }
    }
}

/// <summary>
/// Performs a large query using recursive binary search.
/// </summary>
private static async Task<List<List<ResultField>>> PerformLargeQuery(
    string logGroupName,
    string queryString,
    long startTime,

```



```
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }

        var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-
MM-ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
{endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($" -> Returning {results.Count} logs (less than
limit of {limit})");
            return results;
        }

        Console.WriteLine($" -> Hit limit of {limit}. Need to split and
recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($" -> No timestamp found in results. Returning
{results.Count} logs.");
            return results;
        }

        Console.WriteLine($" -> Last log timestamp: {lastLogTimestamp}");
    }
}
```

```

        // Parse the timestamp and add 1 millisecond to avoid querying the same
        log again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($" -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($" -> Offset timestamp (last
+ 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid
        overlapping with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
        last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second
            Console.WriteLine($" -> Adjusted to next full second:
{offsetLastLogTime}s
({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($" -> Comparing:
offsetLastLogTime={offsetLastLogTime}s vs endTime={endTime}s");
        Console.WriteLine($" -> End time as date:
{DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($" -> No time range left to query. Offset time
({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
SplitDateRange(offsetLastLogTime, endTime);

```

```

        var range1StartDate =
            DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
            ddTHH:mm:ss.fffZ");
        var range1EndDate =
            DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-
            ddTHH:mm:ss.fffZ");
        var range2StartDate =
            DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
            ddTHH:mm:ss.fffZ");
        var range2EndDate =
            DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-
            ddTHH:mm:ss.fffZ");

        Console.WriteLine($" -> Splitting remaining range:");
        Console.WriteLine($"      Range 1: {range1StartDate} ({range1Start}s) to
        {range1EndDate} ({range1End}s)");
        Console.WriteLine($"      Range 2: {range2StartDate} ({range2Start}s) to
        {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($" -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
        range1Start, range1End, limit);
        Console.WriteLine($" -> Range 1 returned {results1.Count} logs");

        Console.WriteLine($" -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
        range2Start, range2End, limit);
        Console.WriteLine($" -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($" -> Combined total: {allResults.Count} logs
        ({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.

```

```

    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {

```

```

        return null;
    }

    if (response.Status == QueryStatus.Complete)
    {
        return response.Results;
    }

    if (response.Status == QueryStatus.Failed ||
        response.Status == QueryStatus.Cancelled ||
        response.Status == QueryStatus.Timeout ||
        response.Status == QueryStatus.Unknown)
    {
        _logger.LogError($"Query failed with status: {response.Status}");
        return null;
    }

    await Task.Delay(retryDelay);
    retryCount++;
}

_logger.LogError("Timed out waiting for query results.");
return null;
}

/// <summary>
/// Cleans up the resources created during the scenario.
/// </summary>
public static async Task<bool> Cleanup()
{
    var cleanup = !_interactive || GetYesNoResponse(
        "Do you want to delete the CloudFormation stack and all resources?
(y/n) ");

    if (cleanup)
    {
        try
        {
            var stackDeleteSuccess = await
DeleteCloudFormationStack(_stackName, false);
            return stackDeleteSuccess;
        }
        catch (Exception ex)
        {

```

```

        _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
        return false;
    }
}

Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
_logger.LogInformation("CloudWatch Logs Large Query scenario is
complete.");
return true;
}

/// <summary>
/// Deletes the CloudFormation stack and waits for confirmation.
/// </summary>
private static async Task<bool> DeleteCloudFormationStack(string stackName,
bool forceDelete)
{
    var request = new DeleteStackRequest
    {
        StackName = stackName,
    };

    if (forceDelete)
    {
        request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
    }

    await _amazonCloudFormation.DeleteStackAsync(request);
    Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

    bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

    if (stackDeleted)
    {
        Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
        return true;
    }
    else
    {

```

```

        _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
        return false;
    }
}

/// <summary>
/// Waits for the stack to be deleted.
/// </summary>
private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
{
    int retryCount = 0;
    const int maxRetries = 30;
    const int retryDelay = 10000;

    while (retryCount < maxRetries)
    {
        var describeStacksRequest = new DescribeStacksRequest
        {
            StackName = stackName
        };

        try
        {
            var describeStacksResponse = await
            _amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count == 0 ||
                describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
            {
                return true;
            }

            if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus
== StackStatus.DELETE_FAILED)
            {
                return await DeleteCloudFormationStack(stackName, true);
            }
        }
        catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
        {

```

```

        return true;
    }

    Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to
be deleted...");
    await Task.Delay(retryDelay);
    retryCount++;
}

_logger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
return false;
}

/// <summary>
/// Waits with a countdown display.
/// </summary>
private static async Task WaitWithCountdown(int seconds)
{
    for (int i = seconds; i > 0; i--)
    {
        Console.Write($"\\rWaiting: {i} seconds remaining... ");
        await Task.Delay(1000);
    }
    Console.WriteLine("\\rWait complete.                ");
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
    return response;
}

/// <summary>
/// Prompts the user for a stack name.
/// </summary>
private static string PromptUserForStackName()
{

```



```

        if (_interactive)
        {
            Console.WriteLine($"Enter a name for the CloudFormation stack (press
Enter for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrEmpty(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
            return _stackName;
        }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
        if (_interactive)
        {
            Console.WriteLine($"{prompt}(press Enter for default '{defaultValue}'):
");
            string? input = Console.ReadLine();
            return string.IsNullOrEmpty(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.WriteLine($"{prompt}(press Enter for default '{defaultValue}'):
");

```

```

        string? input = Console.ReadLine();
        if (string.IsNullOrEmpty(input) || !int.TryParse(input, out var
result))
        {
            return defaultValue;
        }
        return result;
    }
    return defaultValue;
}

/// <summary>
/// Prompts the user for a long value.
/// </summary>
private static long PromptUserForLong(string prompt)
{
    if (_interactive)
    {
        Console.Write(prompt);
        string? input = Console.ReadLine();
        if (long.TryParse(input, out var result))
        {
            return result;
        }
    }
    return 0;
}

/// <summary>
/// Finds duplicate log entries based on timestamp and message.
/// </summary>
private static List<(string Timestamp, string Message, int Count)>
FindDuplicateLogs(List<List<ResultField>> logs)
{
    var logSignatures = new Dictionary<string, int>();

    foreach (var log in logs)
    {
        var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
        var message = log.Find(f => f.Field == "@message")?.Value ?? "";
        var signature = $"{timestamp}|{message}";

        if (logSignatures.ContainsKey(signature))
        {

```

```

        logSignatures[signature]++;
    }
    else
    {
        logSignatures[signature] = 1;
    }
}

return logSignatures
    .Where(kvp => kvp.Value > 1)
    .Select(kvp =>
    {
        var parts = kvp.Key.Split('|');
        return (Timestamp: parts[0], Message: parts[1], Count:
kvp.Value);
    })
    .OrderByDescending(x => x.Count)
    .ToList();
}
}

```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
 - [GetQueryResults](#)
 - [StartQuery](#)

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

This is the entry point.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";

```

```
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(Number.parseInt(process.env.QUERY_START_DATE)),
    new Date(Number.parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found: ${cloudWatchQuery.results.length}`,
);
```

This is a class that splits queries into multiple steps if necessary.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
```

```

    * performs a binary search across all of the logs in the provided
    * date range if a query returns the maximum number of results.
    *
    * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient}
client
    * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
{ limit: number } }} config
    */
    constructor(client, { logGroupNames, dateRange, queryConfig }) {
        this.client = client;
        /**
         * All log groups are queried.
         */
        this.logGroupNames = logGroupNames;

        /**
         * The inclusive date range that is queried.
         */
        this.dateRange = dateRange;

        /**
         * CloudWatch Logs never returns more than 10,000 logs.
         */
        this.limit = queryConfig?.limit ?? 10000;

        /**
         * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
         */
        this.results = [];
    }

    /**
     * Run the query.
     */
    async run() {
        this.secondsElapsed = 0;
        const start = new Date();
        this.results = await this._largeQuery(this.dateRange);
        const end = new Date();
        this.secondsElapsed = (end - start) / 1000;
        return this.results;
    }

    /**

```

```

    * Recursively query for logs.
    * @param {[Date, Date]} dateRange
    * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[
[]>}}
    */
    async _largeQuery(dateRange) {
        const logs = await this._query(dateRange, this.limit);

        console.log(
            `Query date range: ${dateRange
                .map((d) => d.toISOString())
                .join(" to ")}. Found ${logs.length} logs.`
        );

        if (logs.length < this.limit) {
            return logs;
        }

        const lastLogDate = this._getLastLogDate(logs);
        const offsetLastLogDate = new Date(lastLogDate);
        offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
        const subDateRange = [offsetLastLogDate, dateRange[1]];
        const [r1, r2] = splitDateRange(subDateRange);
        const results = await Promise.all([
            this._largeQuery(r1),
            this._largeQuery(r2),
        ]);
        return [logs, ...results].flat();
    }

    /**
     * Find the most recent log in a list of logs.
     * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
     */
    _getLastLogDate(logs) {
        const timestamps = logs
            .map(
                (log) =>
                    log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
            )
            .filter((t) => !!t)
            .map((t) => `${t}Z`)
            .sort();
    }

```

```

    if (!timestamps.length) {
      throw new Error("No timestamp found in logs.");
    }

    return new Date(timestamps[timestamps.length - 1]);
  }

  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }

  /**
   * Starts a query and waits for it to complete.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   */
  async _query(dateRange, maxLogs) {
    try {
      const { queryId } = await this._startQuery(dateRange, maxLogs);
      const { results } = await this._waitUntilQueryDone(queryId);
      return results ?? [];
    } catch (err) {
      /**
       * This error is thrown when StartQuery returns an error indicating
       * that the query's start or end date occur before the log group was
       * created.
       */
      if (err instanceof DateOutOfBoundsError) {
        return [];
      }
      throw err;
    }
  }

  /**
   * Wrapper for the StartQueryCommand. Uses a static query string
   * for consistency.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   * @returns {Promise<{ queryId: string }>}

```

```

    */
    async _startQuery([startDate, endDate], maxLogs = 10000) {
      try {
        return await this.client.send(
          new StartQueryCommand({
            logGroupNames: this.logGroupNames,
            queryString: "fields @timestamp, @message | sort @timestamp asc",
            startTime: startDate.valueOf(),
            endTime: endDate.valueOf(),
            limit: maxLogs,
          }),
        );
      } catch (err) {
        /** @type {string} */
        const message = err.message;
        if (message.startsWith("Query's end date and time")) {
          // This error indicates that the query's start or end date occur
          // before the log group was created.
          throw new DateOutOfBoundsError(message);
        }

        throw err;
      }
    }

    /**
     * Call GetQueryResultsCommand until the query is done.
     * @param {string} queryId
     */
    _waitUntilQueryDone(queryId) {
      const getResults = async () => {
        const results = await this._getQueryResults(queryId);
        const queryDone = [
          "Complete",
          "Failed",
          "Cancelled",
          "Timeout",
          "Unknown",
        ].includes(results.status);

        return { queryDone, results };
      };

      return retry(

```



```
    { intervalInMs: 1000, maxRetries: 60, quiet: true },
    async () => {
      const { queryDone, results } = await getResults();
      if (!queryDone) {
        throw new Error("Query not done.");
      }

      return results;
    },
  );
}
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [GetQueryResults](#)
 - [StartQuery](#)

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

This file invokes an example module for managing CloudWatch queries exceeding 10,000 results.

```
import logging
import os
import sys

import boto3
from botocore.config import Config

from cloudwatch_query import CloudWatchQuery
from date_utilities import DateUtilities
```

```

# Configure logging at the module level.
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(filename)s:%(lineno)d - %(message)s",
)

DEFAULT_QUERY_LOG_GROUP = "/workflows/cloudwatch-logs/large-query"

class CloudWatchLogsQueryRunner:
    def __init__(self):
        """
        Initializes the CloudWatchLogsQueryRunner class by setting up date
        utilities
        and creating a CloudWatch Logs client with retry configuration.
        """
        self.date_utilities = DateUtilities()
        self.cloudwatch_logs_client = self.create_cloudwatch_logs_client()

    def create_cloudwatch_logs_client(self):
        """
        Creates and returns a CloudWatch Logs client with a specified retry
        configuration.

        :return: A CloudWatch Logs client instance.
        :rtype: boto3.client
        """
        try:
            return boto3.client("logs", config=Config(retries={"max_attempts":
10}))
        except Exception as e:
            logging.error(f"Failed to create CloudWatch Logs client: {e}")
            sys.exit(1)

    def fetch_environment_variables(self):
        """
        Fetches and validates required environment variables for query start and
        end dates.
        Fetches the environment variable for log group, returning the default
        value if it
        does not exist.

        :return: Tuple of query start date and end date as integers and the log
        group.

```

```

        :rtype: tuple
        :raises SystemExit: If required environment variables are missing or
invalid.
        """
        try:
            query_start_date = int(os.environ["QUERY_START_DATE"])
            query_end_date = int(os.environ["QUERY_END_DATE"])
        except KeyError:
            logging.error(
                "Both QUERY_START_DATE and QUERY_END_DATE environment variables
are required."
            )
            sys.exit(1)
        except ValueError as e:
            logging.error(f"Error parsing date environment variables: {e}")
            sys.exit(1)

        try:
            log_group = os.environ["QUERY_LOG_GROUP"]
        except KeyError:
            logging.warning("No QUERY_LOG_GROUP environment variable, using
default value")
            log_group = DEFAULT_QUERY_LOG_GROUP

        return query_start_date, query_end_date, log_group

def convert_dates_to_iso8601(self, start_date, end_date):
    """
    Converts UNIX timestamp dates to ISO 8601 format using DateUtilities.

    :param start_date: The start date in UNIX timestamp.
    :type start_date: int
    :param end_date: The end date in UNIX timestamp.
    :type end_date: int
    :return: Start and end dates in ISO 8601 format.
    :rtype: tuple
    """
    start_date_iso8601 =
self.date_utilities.convert_unix_timestamp_to_iso8601(
    start_date
)
    end_date_iso8601 = self.date_utilities.convert_unix_timestamp_to_iso8601(
    end_date
)

```

```

        return start_date_iso8601, end_date_iso8601

    def execute_query(
        self,
        start_date_iso8601,
        end_date_iso8601,
        log_group="/workflows/cloudwatch-logs/large-query",
        query="fields @timestamp, @message | sort @timestamp asc"
    ):
        """
        Creates a CloudWatchQuery instance and executes the query with provided
        date range.

        :param start_date_iso8601: The start date in ISO 8601 format.
        :type start_date_iso8601: str
        :param end_date_iso8601: The end date in ISO 8601 format.
        :type end_date_iso8601: str
        :param log_group: Log group to search: "/workflows/cloudwatch-logs/large-
query"
        :type log_group: str
        :param query: Query string to pass to the CloudWatchQuery instance
        :type query: str
        """
        cloudwatch_query = CloudWatchQuery(
            log_group=log_group,
            query_string=query
        )
        cloudwatch_query.query_logs((start_date_iso8601, end_date_iso8601))
        logging.info("Query executed successfully.")
        logging.info(
            f"Queries completed in {cloudwatch_query.query_duration} seconds.
Total logs found: {len(cloudwatch_query.query_results)}"
        )

def main():
    """
    Main function to start a recursive CloudWatch logs query.
    Fetches required environment variables, converts dates, and executes the
    query.
    """
    logging.info("Starting a recursive CloudWatch logs query...")
    runner = CloudWatchLogsQueryRunner()

```

```

    query_start_date, query_end_date, log_group =
runner.fetch_environment_variables()
    start_date_iso8601 = DateUtilities.convert_unix_timestamp_to_iso8601(
        query_start_date
    )
    end_date_iso8601 =
DateUtilities.convert_unix_timestamp_to_iso8601(query_end_date)
    runner.execute_query(start_date_iso8601, end_date_iso8601,
log_group=log_group)

if __name__ == "__main__":
    main()

```

This module processes CloudWatch queries exceeding 10,000 results.

```

import logging
import time
from datetime import datetime
import threading
import boto3

from date_utilities import DateUtilities

DEFAULT_QUERY = "fields @timestamp, @message | sort @timestamp asc"
DEFAULT_LOG_GROUP = "/workflows/cloudwatch-logs/large-query"

class DateOutOfBoundsError(Exception):
    """Exception raised when the date range for a query is out of bounds."""

    pass

class CloudWatchQuery:
    """
    A class to query AWS CloudWatch logs within a specified date range.

    :vartype date_range: tuple
    :ivar limit: Maximum number of log entries to return.
    :vartype limit: int
    :log_group str: Name of the log group to query
    :query_string str: query
    """

```

```

"""

def __init__(self, log_group: str = DEFAULT_LOG_GROUP, query_string:
str=DEFAULT_QUERY) -> None:
    self.lock = threading.Lock()
    self.log_group = log_group
    self.query_string = query_string
    self.query_results = []
    self.query_duration = None
    self.datetime_format = "%Y-%m-%d %H:%M:%S.%f"
    self.date_utilities = DateUtilities()
    self.limit = 10000

def query_logs(self, date_range):
    """
    Executes a CloudWatch logs query for a specified date range and
    calculates the execution time of the query.

    :return: A batch of logs retrieved from the CloudWatch logs query.
    :rtype: list
    """
    start_time = datetime.now()

    start_date, end_date = self.date_utilities.normalize_date_range_format(
        date_range, from_format="unix_timestamp", to_format="datetime"
    )

    logging.info(
        f"Original query:"
        f"\n      START:      {start_date}"
        f"\n      END:        {end_date}"
        f"\n      LOG GROUP: {self.log_group}"
    )
    self.recursive_query((start_date, end_date))
    end_time = datetime.now()
    self.query_duration = (end_time - start_time).total_seconds()

def recursive_query(self, date_range):
    """
    Processes logs within a given date range, fetching batches of logs
    recursively if necessary.

    :param date_range: The date range to fetch logs for, specified as a tuple
    (start_timestamp, end_timestamp).

```

```

        :type date_range: tuple
        :return: None if the recursive fetching is continued or stops when the
final batch of logs is processed.
            Although it doesn't explicitly return the query results, this
method accumulates all fetched logs
            in the `self.query_results` attribute.
        :rtype: None
        """
        batch_of_logs = self.perform_query(date_range)
        # Add the batch to the accumulated logs
        with self.lock:
            self.query_results.extend(batch_of_logs)
        if len(batch_of_logs) == self.limit:
            logging.info(f"Fetched {self.limit}, checking for more...")
            most_recent_log = self.find_most_recent_log(batch_of_logs)
            most_recent_log_timestamp = next(
                item["value"]
                for item in most_recent_log
                if item["field"] == "@timestamp"
            )
            new_range = (most_recent_log_timestamp, date_range[1])
            midpoint = self.date_utilities.find_middle_time(new_range)

            first_half_thread = threading.Thread(
                target=self.recursive_query,
                args=((most_recent_log_timestamp, midpoint)),
            )
            second_half_thread = threading.Thread(
                target=self.recursive_query, args=((midpoint, date_range[1])),
            )

            first_half_thread.start()
            second_half_thread.start()

            first_half_thread.join()
            second_half_thread.join()

    def find_most_recent_log(self, logs):
        """
        Search a list of log items and return most recent log entry.
        :param logs: A list of logs to analyze.
        :return: log
        :type :return List containing log item details
        """

```

```

        most_recent_log = None
        most_recent_date = "1970-01-01 00:00:00.000"

        for log in logs:
            for item in log:
                if item["field"] == "@timestamp":
                    logging.debug(f"Compared: {item['value']} to
{most_recent_date}")
                    if (
                        self.date_utilities.compare_dates(
                            item["value"], most_recent_date
                        )
                        == item["value"]
                    ):
                        logging.debug(f"New most recent: {item['value']}")
                        most_recent_date = item["value"]
                        most_recent_log = log
            logging.info(f"Most recent log date of batch: {most_recent_date}")
        return most_recent_log

    def perform_query(self, date_range):
        """
        Performs the actual CloudWatch log query.

        :param date_range: A tuple representing the start and end datetime for
the query.
        :type date_range: tuple
        :return: A list containing the query results.
        :rtype: list
        """
        client = boto3.client("logs")
        try:
            try:
                start_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])
                )
                end_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
                )
                response = client.start_query(
                    logGroupName=self.log_group,
                    startTime=start_time,

```



```

        endTime=end_time,
        queryString=self.query_string,
        limit=self.limit,
    )
    query_id = response["queryId"]
except client.exceptions.ResourceNotFoundException as e:
    raise DateOutOfBoundsError(f"Resource not found: {e}")
while True:
    time.sleep(1)
    results = client.get_query_results(queryId=query_id)
    if results["status"] in [
        "Complete",
        "Failed",
        "Cancelled",
        "Timeout",
        "Unknown",
    ]:
        return results.get("results", [])
except DateOutOfBoundsError:
    return []

def _initiate_query(self, client, date_range, max_logs):
    """
    Initiates the CloudWatch logs query.

    :param date_range: A tuple representing the start and end datetime for
    the query.
    :type date_range: tuple
    :param max_logs: The maximum number of logs to retrieve.
    :type max_logs: int
    :return: The query ID as a string.
    :rtype: str
    """
    try:
        start_time = round(
self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])
        )
        end_time = round(
self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
        )
        response = client.start_query(
            logGroupName=self.log_group,

```

```

        startTime=start_time,
        endTime=end_time,
        queryString=self.query_string,
        limit=max_logs,
    )
    return response["queryId"]
except client.exceptions.ResourceNotFoundException as e:
    raise DateOutOfBoundsError(f"Resource not found: {e}")

def _wait_for_query_results(self, client, query_id):
    """
    Waits for the query to complete and retrieves the results.

    :param query_id: The ID of the initiated query.
    :type query_id: str
    :return: A list containing the results of the query.
    :rtype: list
    """
    while True:
        time.sleep(1)
        results = client.get_query_results(queryId=query_id)
        if results["status"] in [
            "Complete",
            "Failed",
            "Cancelled",
            "Timeout",
            "Unknown",
        ]:
            return results.get("results", [])

```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [GetQueryResults](#)
 - [StartQuery](#)

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use scheduled events to invoke a Lambda function

The following code examples show how to create an AWS Lambda function invoked by an Amazon EventBridge scheduled event.

Java

SDK for Java 2.x

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda Java runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda JavaScript runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](#).

Services used in this example

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

This example shows how to register an AWS Lambda function as the target of a scheduled Amazon EventBridge event. The Lambda handler writes a friendly message and the full event data to Amazon CloudWatch Logs for later retrieval.

- Deploys a Lambda function.
- Creates an EventBridge scheduled event and makes the Lambda function the target.
- Grants permission to let EventBridge invoke the Lambda function.
- Prints the latest data from CloudWatch Logs to show the result of the scheduled invocations.
- Cleans up all resources created during the demo.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Logs with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon CloudWatch Logs

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 WorkSpaces, 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 Amazon CloudWatch Logs. It shows you how to configure Amazon CloudWatch Logs to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CloudWatch Logs resources.

Contents

- [Data protection in Amazon CloudWatch Logs](#)
- [Identity and access management for Amazon CloudWatch Logs](#)
- [Compliance validation for Amazon CloudWatch Logs](#)
- [Resilience in Amazon CloudWatch Logs](#)
- [Infrastructure security in Amazon CloudWatch Logs](#)
- [Using CloudWatch Logs with interface VPC endpoints](#)

Data protection in Amazon CloudWatch Logs

Note

In addition to the following information about general data protection in AWS, CloudWatch Logs also enables you to protect sensitive data in log events by masking it. For more information, see [Help protect sensitive log data with masking](#).

The AWS [shared responsibility model](#) applies to data protection in Amazon CloudWatch Logs. 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 CloudWatch Logs 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.

Encryption at rest

CloudWatch Logs protects data at rest using encryption. All log groups are encrypted. By default, the CloudWatch Logs service manages the server-side encryption and uses server-side encryption with 256-bit Advanced Encryption Standard Galois/Counter Mode (AES-GCM) to encrypt log data at rest.

If you want to manage the keys used for encrypting and decrypting your logs, use AWS KMS keys. For more information, see [Encrypt log data in CloudWatch Logs using AWS Key Management Service](#).

Encryption in transit

CloudWatch Logs uses end-to-end encryption of data in transit. The CloudWatch Logs service manages the server-side encryption keys.

Identity and access management for Amazon CloudWatch Logs

Access to Amazon CloudWatch Logs requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources, such as to retrieve CloudWatch Logs data about your cloud resources. The following sections provide details on how you can use [AWS Identity and Access Management \(IAM\)](#) and CloudWatch Logs to help secure your resources by controlling who can access them:

- [Authentication](#)
- [Access control](#)

Authentication

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in 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 CloudWatch Logs resources. For example, you must have permissions to create log streams, create log groups, and so on.

The following sections describe how to manage permissions for CloudWatch Logs. We recommend that you read the overview first.

- [Overview of managing access permissions to your CloudWatch Logs resources](#)
- [Using identity-based policies \(IAM policies\) for CloudWatch Logs](#)
- [CloudWatch Logs permissions reference](#)

Overview of managing access permissions to your CloudWatch Logs resources

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Topics

- [CloudWatch Logs resources and operations](#)
- [Understanding resource ownership](#)
- [Managing access to resources](#)
- [Specifying policy elements: Actions, effects, and principals](#)
- [Specifying conditions in a policy](#)

CloudWatch Logs resources and operations

In CloudWatch Logs the primary resources are log groups, log streams and destinations. CloudWatch Logs does not support subresources (other resources for use with the primary resource).

These resources and subresources have unique Amazon Resource Names (ARNs) associated with them as shown in the following table.

Resource type	ARN format
Log group	Both of the following are used. The second one, with the <code>:*</code> at the end, is what is returned by the <code>describe-log-groups</code> CLI command and the DescribeLogGroups API. arn:aws:logs:<i>region</i>:<i>account-id</i> :log-group:<i>log_group_name</i>

Resource type	ARN format
	arn:aws:logs:<i>region</i>:<i>account-id</i> :log-group:<i>log_group_name</i> :* Use the first version, without the trailing : *, in the following situations: • In the <code>logGroupIdentifier</code> input field in many CloudWatch Logs APIs. • In the <code>resourceArn</code> field in tagging APIs • In IAM policies, when specifying permissions for TagResource, UntagResource, and ListTagsForResource. Use the second version, with the trailing : *, to refer to the ARN when specifying permissions in IAM policies for all other API actions.
Log stream	arn:aws:logs: <i>region</i> : <i>account-id</i> :log-group: <i>log_group_name</i> :log-stream: <i>log-stream-name</i>
Destination	arn:aws:logs: <i>region</i> : <i>account-id</i> :destination: <i>destination_name</i>

For more information about ARNs, see [ARNs](#) in *IAM User Guide*. For information about CloudWatch Logs ARNs, see [Amazon Resource Names \(ARNs\)](#) in *Amazon Web Services General Reference*. For an example of a policy that covers CloudWatch Logs, see [Using identity-based policies \(IAM policies\) for CloudWatch Logs](#).

CloudWatch Logs provides a set of operations to work with the CloudWatch Logs resources. For a list of available operations, see [CloudWatch Logs permissions reference](#).

Understanding resource ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the [principal entity](#) (that is, the root account, a user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a log group, your AWS account is the owner of the CloudWatch Logs resource.
- If you create a user in your AWS account and grant permissions to create CloudWatch Logs resources to that user, the user can create CloudWatch Logs resources. However, your AWS account, to which the user belongs, owns the CloudWatch Logs resources.
- If you create an IAM role in your AWS account with permissions to create CloudWatch Logs resources, anyone who can assume the role can create CloudWatch Logs resources. Your AWS account, to which the role belongs, owns the CloudWatch Logs resources.

Managing access to resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

Note

This section discusses using IAM in the context of CloudWatch Logs. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see [What is IAM?](#) in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see [IAM policy reference](#) in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as identity-based policies (IAM policies) and policies attached to a resource are referred to as resource-based policies. CloudWatch Logs supports identity-based policies, and resource-based policies for destinations, which are used to enable cross account subscriptions. For more information, see [Cross-account cross-Region subscriptions](#).

Topics

- [Log group permissions and Contributor Insights](#)

- [Resource-based policies](#)

Log group permissions and Contributor Insights

Contributor Insights is a feature of CloudWatch that enables you to analyze data from log groups and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see [Using Contributor Insights to Analyze High-Cardinality Data](#).

When you grant a user the `cloudwatch:PutInsightRule` and `cloudwatch:GetInsightRuleReport` permissions, that user can create a rule that evaluates any log group in CloudWatch Logs and then see the results. The results can contain contributor data for those log groups. Be sure to grant these permissions only to users who should be able to view this data.

Resource-based policies

CloudWatch Logs supports resource-based policies for destinations, which you can use to enable cross account subscriptions. For more information, see [Step 1: Create a destination](#). Destinations can be created using the [PutDestination](#) API, and you can add a resource policy to the destination using the [PutDestinationPolicy](#) API. The following example allows another AWS account with the account ID 111122223333 to subscribe their log groups to the destination `arn:aws:logs:us-east-1:123456789012:destination:testDestination`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "111122223333"
      },
      "Action" : "logs:PutSubscriptionFilter",
      "Resource" : "arn:aws:logs:us-east-1:123456789012:destination:testDestination"
    }
  ]
}
```

```
}
```

Specifying policy elements: Actions, effects, and principals

For each CloudWatch Logs resource, the service defines a set of API operations. To grant permissions for these API operations, CloudWatch Logs defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see [CloudWatch Logs resources and operations](#) and [CloudWatch Logs permissions reference](#).

The following are the basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information, see [CloudWatch Logs resources and operations](#).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, the `logs.DescribeLogGroups` permission allows the user permissions to perform the `DescribeLogGroups` operation.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). CloudWatch Logs supports resource-based policies for destinations.

To learn more about IAM policy syntax and descriptions, see [AWS IAM Policy Reference](#) in the *IAM User Guide*.

For a table showing all of the CloudWatch Logs API actions and the resources that they apply to, see [CloudWatch Logs permissions reference](#).

Specifying conditions in a policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a

specific date. For more information about specifying conditions in a policy language, see [Condition](#) in the *IAM User Guide*.

To express conditions, you use predefined condition keys. For a list of context keys supported by each AWS service and a list of AWS-wide policy keys, see [Actions, resources, and condition keys for AWS services](#) and [AWS global condition context keys](#).

Note

You can use tags to control access to CloudWatch Logs resources, including log groups and destinations. Access to log streams is controlled at the log group level, because of the hierarchical relation between log groups and log streams. For more information about using tags to control access, see [Controlling access to Amazon Web Services resources using tags](#).

Using identity-based policies (IAM policies) for CloudWatch Logs

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

Important

We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your CloudWatch Logs resources. For more information, see [Overview of managing access permissions to your CloudWatch Logs resources](#).

This topic covers the following:

- [Permissions required to use the CloudWatch console](#)
- [AWS managed \(predefined\) policies for CloudWatch Logs](#)
- [Customer managed policy examples](#)

The following is an example of a permissions policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    }
  ]
}
```

This policy has one statement that grants permissions to create log groups and log streams, to upload log events to log streams, and to list details about log streams.

The wildcard character (*) at the end of the Resource value means that the statement allows permission for the `logs:CreateLogGroup`, `logs:CreateLogStream`, `logs:PutLogEvents`, and `logs:DescribeLogStreams` actions on any log group. To limit this permission to a specific log group, replace the wildcard character (*) in the resource ARN with the specific log group ARN. For more information about the sections within an IAM policy statement, see [IAM Policy Elements Reference](#) in *IAM User Guide*. For a list showing all of the CloudWatch Logs actions, see [CloudWatch Logs permissions reference](#).

Permissions required to use the CloudWatch console

For a user to work with CloudWatch Logs in the CloudWatch console, that user must have a minimum set of permissions that allows the user to describe other AWS resources in their AWS account. In order to use CloudWatch Logs in the CloudWatch console, you must have permissions from the following services:

- CloudWatch

- CloudWatch Logs
- OpenSearch Service
- IAM
- Kinesis
- Lambda
- Amazon S3

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy. To ensure that those users can still use the CloudWatch console, also attach the `CloudWatchReadOnlyAccess` managed policy to the user, as described in [AWS managed \(predefined\) policies for CloudWatch Logs](#).

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the CloudWatch Logs API.

The full set of permissions required to work with the CloudWatch console for a user who is not using the console to manage log subscriptions are:

- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `logs:CancelExportTask`
- `logs>CreateExportTask`
- `logs>CreateLogGroup`
- `logs>CreateLogStream`
- `logs>DeleteLogGroup`
- `logs>DeleteLogStream`
- `logs>DeleteMetricFilter`
- `logs>DeleteQueryDefinition`
- `logs>DeleteRetentionPolicy`
- `logs>DeleteSubscriptionFilter`
- `logs:DescribeExportTasks`
- `logs:DescribeLogGroups`
- `logs:DescribeLogStreams`

- logs:DescribeMetricFilters
- logs:DescribeQueryDefinitions
- logs:DescribeQueries
- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutQueryDefinition
- logs:PutRetentionPolicy
- logs:StartQuery
- logs:StopQuery
- logs:PutSubscriptionFilter
- logs:TestMetricFilter

For a user who will also be using the console to manage log subscriptions, the following permissions are also required:

- es:DescribeElasticsearchDomain
- es:ListDomainNames
- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStreams
- kinesis:ListStreams

- `lambda:AddPermission`
- `lambda:CreateFunction`
- `lambda:GetFunctionConfiguration`
- `lambda:ListAliases`
- `lambda:ListFunctions`
- `lambda:ListVersionsByFunction`
- `lambda:RemovePermission`
- `s3:ListBuckets`

AWS managed (predefined) policies for CloudWatch Logs

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. Managed policies grant necessary permissions for common use cases so you can avoid having to investigate what permissions are needed. For more information, see [AWS Managed Policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users and roles in your account, are specific to CloudWatch Logs:

- **CloudWatchLogsFullAccess** – Grants full access to CloudWatch Logs.
- **CloudWatchLogsReadOnlyAccess** – Grants read-only access to CloudWatch Logs.

CloudWatchLogsFullAccess

The **CloudWatchLogsFullAccess** policy grants full access to CloudWatch Logs. The policy includes the `cloudwatch:GenerateQuery` and `cloudwatch:GenerateQueryResultsSummary` permissions, so that users with this policy can generate a [CloudWatch Logs Insights](#) query string from a natural language prompt. To see the full contents of the policy, see [CloudWatchLogsFullAccess](#) in the *AWS Managed Policy Reference Guide*.

CloudWatchLogsReadOnlyAccess

The **CloudWatchLogsReadOnlyAccess** policy grants read-only access to CloudWatch Logs. It includes the `cloudwatch:GenerateQuery` and `cloudwatch:GenerateQueryResultsSummary` permissions, so that users with this policy can generate a [CloudWatch Logs Insights](#) query string from a natural language prompt. To see the full

contents of the policy, see [CloudWatchLogsReadOnlyAccess](#) in the *AWS Managed Policy Reference Guide*.

CloudWatchOpenSearchDashboardsFullAccess

The **CloudWatchOpenSearchDashboardsFullAccess** policy grants access to create, manage, and delete integrations with OpenSearch Service, and to create delete and manage vended log dashboards in those integrations. For more information, see [Analyze with Amazon OpenSearch Service](#).

To see the full contents of the policy, see [CloudWatchOpenSearchDashboardsFullAccess](#) in the *AWS Managed Policy Reference Guide*.

CloudWatchOpenSearchDashboardAccess

The **CloudWatchOpenSearchDashboardAccess** policy grants access to view vended logs dashboards that are created with Amazon OpenSearch Service analytics. For more information, see [Analyze with Amazon OpenSearch Service](#).

Important

In addition to granting this policy, to enable a role or user to be able to view vended log dashboards, you must also specify them when you create the integration with OpenSearch Service. For more information, see [Step 1: Create the integration with OpenSearch Service](#).

To see the full contents of the policy, see [CloudWatchOpenSearchDashboardAccess](#) in the *AWS Managed Policy Reference Guide*.

CloudWatchLogsCrossAccountSharingConfiguration

The **CloudWatchLogsCrossAccountSharingConfiguration** policy grants access to create, manage, and view Observability Access Manager links for sharing CloudWatch Logs resources between accounts. For more information, see [CloudWatch cross-account observability](#).

To see the full contents of the policy, see [CloudWatchLogsCrossAccountSharingConfiguration](#) in the *AWS Managed Policy Reference Guide*.

CloudWatch Logs updates to AWS managed policies

View details about updates to AWS managed policies for CloudWatch Logs since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Logs Document history page.

Change	Description	Date
CloudWatchLogsFullAccess – Update to an existing policy.	CloudWatch Logs added permissions to CloudWatchLogsFullAccess. Permissions for observability administration actions were added to allow read-only access to telemetry pipelines and S3 table integrations.	December 02, 2025
CloudWatchLogsReadOnlyAccess – Update to an existing policy.	CloudWatch Logs added permissions to CloudWatchLogsReadOnlyAccess. Permissions for observability administration actions were added to allow read-only access to telemetry pipelines and S3 table integrations.	December 02, 2025
CloudWatchLogsFullAccess – Update to an existing policy.	CloudWatch Logs added permissions to CloudWatchLogsFullAccess. Permissions for <code>cloudwatch:GenerateQueryResultsSummary</code> were added to allow for generation of a natural language summary of the query results.	May 20, 2025

Change	Description	Date
CloudWatchLogsReadOnlyAccess – Update to an existing policy.	CloudWatch Logs added permissions to CloudWatchLogsReadOnlyAccess. Permissions for <code>cloudwatch:GenerateQueryResultsSummary</code> were added to allow for generation of a natural language summary of the query results.	May 20, 2025
CloudWatchLogsFullAccess – Update to an existing policy.	CloudWatch Logs added permissions to CloudWatchLogsFullAccess. Permissions for Amazon OpenSearch Service and IAM were added, to enable CloudWatch Logs integration with OpenSearch Service for some features.	December 1, 2024

Change	Description	Date
CloudWatchOpenSearchDashboardsFullAccess – New IAM policy.	CloudWatch Logs added a new IAM policy, CloudWatchOpenSearchDashboardsFullAccess .- This policy grants access to create, manage, and delete integrations with OpenSearch Service, and to create, manage, and delete vendored log dashboards in those integrations. For more information, see Analyze with Amazon OpenSearch Service .	December 1, 2024
CloudWatchOpenSearchDashboardAccess – New IAM policy.	CloudWatch Logs added a new IAM policy, CloudWatchOpenSearchDashboardAccess .- This policy grants access to view vendored logs dashboards powered by Amazon OpenSearch Service. For more information, see Analyze with Amazon OpenSearch Service .	December 1, 2024

Change	Description	Date
CloudWatchLogsFullAccess – Update to an existing policy.	CloudWatch Logs added a permission to CloudWatchLogsFullAccess. The <code>cloudwatch:GenerateQuery</code> permission was added, so that users with this policy can generate a CloudWatch Logs Insights query string from a natural language prompt.	November 27, 2023
CloudWatchLogsReadOnlyAccess – Update to an existing policy.	CloudWatch added a permission to CloudWatchLogsReadOnlyAccess. The <code>cloudwatch:GenerateQuery</code> permission was added, so that users with this policy can generate a CloudWatch Logs Insights query string from a natural language prompt.	November 27, 2023

Change	Description	Date
CloudWatchLogsReadOnlyAccess – Update to an existing policy	CloudWatch Logs added permissions to CloudWatchLogsReadOnlyAccess. The <code>logs:StartLiveTail</code> and <code>logs:StopLiveTail</code> permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time.	June 6, 2023
CloudWatchLogsCrossAccountSharingConfiguration – New policy	CloudWatch Logs added a new policy to enable you to manage CloudWatch cross-account observability links that share CloudWatch Logs log groups. For more information, see CloudWatch cross-account observability	November 27, 2022

Change	Description	Date
CloudWatchLogsReadOnlyAccess – Update to an existing policy	CloudWatch Logs added permissions to CloudWatchLogsReadOnlyAccess. The <code>oam:ListSinks</code> and <code>oam:ListAttachedLinks</code> permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022

Customer managed policy examples

You can create your own custom IAM policies to allow permissions for CloudWatch Logs actions and resources. You can attach these custom policies to the users or groups that require those permissions.

In this section, you can find example user policies that grant permissions for various CloudWatch Logs actions. These policies work when you are using the CloudWatch Logs API, AWS SDKs, or the AWS CLI.

Examples

- [Example 1: Allow full access to CloudWatch Logs](#)
- [Example 2: Allow read-only access to CloudWatch Logs](#)
- [Example 3: Allow access to one log group](#)

Example 1: Allow full access to CloudWatch Logs

The following policy allows a user to access all CloudWatch Logs actions.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Example 2: Allow read-only access to CloudWatch Logs

AWS provides a **CloudWatchLogsReadOnlyAccess** policy that enables read-only access to CloudWatch Logs data. This policy includes the following permissions.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:Describe*",
        "logs:Get*",
        "logs:List*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "cloudwatch:GenerateQuery"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

Example 3: Allow access to one log group

The following policy allows a user to read and write log events in one specified log group.

Important

The `:*` at the end of the log group name in the `Resource` line is required to indicate that the policy applies to all log streams in this log group. If you omit `:*`, the policy will not be enforced.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:GetLogEvents"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:us-west-2:123456789012:log-
group:SampleLogGroupName:*"
    }
  ]
}
```

Use tagging and IAM policies for control at the log group level

You can grant users access to certain log groups while preventing them from accessing other log groups. To do so, tag your log groups and use IAM policies that refer to those tags. To apply tags to a log group, you need to have either the `logs:TagResource` or `logs:TagLogGroup` permission.

This applies both if you are assigning tags to the log group when you create it, or assigning them later.

For more information about tagging log groups, see [Tag log groups in Amazon CloudWatch Logs](#).

When you tag log groups, you can then grant an IAM policy to a user to allow access to only the log groups with a particular tag. For example, the following policy statement grants access to only log groups with the value of Green for the tag key Team.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:ResourceTag/Team": "Green"
        }
      }
    }
  ]
}
```

The **StopQuery** and **StopLiveTail** API operations don't interact with AWS resources in the traditional sense. They don't return any data, put any data, or modify a resource in any way. Instead, they operate only on a given live tail session or a given CloudWatch Logs Insights query, which are not categorized as resources. As a result, when you specify the Resource field in IAM policies for these operations, you must set the value of the Resource field as *, as in the following example.

JSON

```
{
```

```
"Version": "2012-10-17",
"Statement":
  [ {
    "Effect": "Allow",
    "Action": [
      "logs:StopQuery",
      "logs:StopLiveTail"
    ],
    "Resource": "*"
  }
]
```

For more information about using IAM policy statements, see [Controlling Access Using Policies](#) in the *IAM User Guide*.

CloudWatch Logs permissions reference

When you are setting up [Access control](#) and writing permissions policies that you can attach to an IAM identity (identity-based policies), you can use the following table as a reference. The table lists each CloudWatch Logs API operation and the corresponding actions for which you can grant permissions to perform the action. You specify the actions in the policy's `Action` field. For the `Resource` field, you can specify the ARN of a log group or log stream, or specify `*` to represent all CloudWatch Logs resources.

You can use AWS-wide condition keys in your CloudWatch Logs policies to express conditions. For a complete list of AWS-wide keys, see [AWS Global and IAM Condition Context Keys](#) in the *IAM User Guide*.

Note

To specify an action, use the `logs:` prefix followed by the API operation name. For example: `logs:CreateLogGroup`, `logs:CreateLogStream`, or `logs:*` (for all CloudWatch Logs actions).

CloudWatch Logs API operations and required permissions for actions

CloudWatch Logs API operations	Required permissions (API actions)
CancelExportTask	<code>logs:CancelExportTask</code> Required to cancel a pending or running export task.
CreateExportTask	<code>logs:CreateExportTask</code> Required to export data from a log group to an Amazon S3 bucket.
CreateLogGroup	<code>logs:CreateLogGroup</code> Required to create a new log group.
CreateLogStream	<code>logs:CreateLogStream</code> Required to create a new log stream in a log group.
DeleteDestination	<code>logs:DeleteDestination</code> Required to delete a log destination and disables any subscription filters to it.
DeleteLogGroup	<code>logs:DeleteLogGroup</code> Required to delete a log group and any associated archived log events.
DeleteLogStream	<code>logs:DeleteLogStream</code>

CloudWatch Logs API operations	Required permissions (API actions)
	Required to delete a log stream and any associated archived log events.
DeleteMetricFilter	<code>logs:DeleteMetricFilter</code> Required to delete a metric filter associated with a log group.
DeleteQueryDefinition	<code>logs:DeleteQueryDefinition</code> Required to delete a saved query definition in CloudWatch Logs Insights.
DeleteResourcePolicy	<code>logs:DeleteResourcePolicy</code> Required to delete a CloudWatch Logs resource policy.
DeleteRetentionPolicy	<code>logs:DeleteRetentionPolicy</code> Required to delete a log group's retention policy.
DeleteSubscriptionFilter	<code>logs:DeleteSubscriptionFilter</code> Required to delete the subscription filter associated with a log group.
DescribeDestinations	<code>logs:DescribeDestinations</code> Required to view all destinations associated with the account.

CloudWatch Logs API operations	Required permissions (API actions)
DescribeExportTasks	<code>logs:DescribeExportTasks</code> Required to view all export tasks associated with the account.
DescribeLogGroups	<code>logs:DescribeLogGroups</code> Required to view all log groups associated with the account.
DescribeLogStreams	<code>logs:DescribeLogStreams</code> Required to view all log streams associated with a log group.
DescribeMetricFilters	<code>logs:DescribeMetricFilters</code> Required to view all metrics associated with a log group.
DescribeQueryDefinitions	<code>logs:DescribeQueryDefinitions</code> Required to see the list of saved query definitions in CloudWatch Logs Insights.
DescribeQueries	<code>logs:DescribeQueries</code> Required to see the list of CloudWatch Logs Insights queries that are scheduled, executing, or have recently executed.

CloudWatch Logs API operations	Required permissions (API actions)
DescribeResourcePolicies	<code>logs:DescribeResourcePolicies</code> Required to view a list of CloudWatch Logs resource policies.
DescribeSubscriptionFilters	<code>logs:DescribeSubscriptionFilters</code> Required to view all subscription filters associated with a log group.
FilterLogEvents	<code>logs:FilterLogEvents</code> Required to sort log events by log group filter pattern.
GetLogEvents	<code>logs:GetLogEvents</code> Required to retrieve log events from a log stream.
GetLogGroupFields	<code>logs:GetLogGroupFields</code> Required to retrieve the list of fields that are included in the log events in a log group.
GetLogRecord	<code>logs:GetLogRecord</code> Required to retrieve the details from a single log event.

CloudWatch Logs API operations	Required permissions (API actions)
GetLogObject	<code>logs:GetLogRecord</code> Required to fetch the content of large portions of log events that have been ingested through the PutOpenTelemetryLogs API.
GetQueryResults	<code>logs:GetQueryResults</code> Required to retrieve the results of CloudWatch Logs Insights queries.
ListEntitiesForLogGroup (CloudWatch console-only permission)	<code>logs:ListEntitiesForLogGroup</code> Required to find the entities associated with a log group. Required to explore related logs within the CloudWatch console.
ListLogGroupsForEntity (CloudWatch console-only permission)	<code>logs:ListLogGroupsForEntity</code> Required to find the log groups associated with an entity. Required to explore related logs within the CloudWatch console.
ListTagsLogGroup	<code>logs:ListTagsLogGroup</code> Required to list the tags associated with a log group.
ListLogGroups	<code>logs:DescribeLogGroups</code> Required to view all log groups associated with the account.

CloudWatch Logs API operations	Required permissions (API actions)
PutDestination	<code>logs:PutDestination</code> Required to create or update a destination log stream (such as an Kinesis stream).
PutDestinationPolicy	<code>logs:PutDestinationPolicy</code> Required to create or update an access policy associated with an existing log destination.
PutLogEvents	<code>logs:PutLogEvents</code> Required to upload a batch of log events to a log stream.
PutMetricFilter	<code>logs:PutMetricFilter</code> Required to create or update a metric filter and associate it with a log group.
PutQueryDefinition	<code>logs:PutQueryDefinition</code> Required to save a query in CloudWatch Logs Insights.
PutResourcePolicy	<code>logs:PutResourcePolicy</code> Required to create a CloudWatch Logs resource policy.

CloudWatch Logs API operations	Required permissions (API actions)
PutRetentionPolicy	logs:PutRetentionPolicy Required to set the number of days to keep log events (retention) in a log group.
PutSubscriptionFilter	logs:PutSubscriptionFilter Required to create or update a subscription filter and associate it with a log group.
StartQuery	logs:StartQuery Required to start CloudWatch Logs Insights queries.
StopQuery	logs:StopQuery Required to stop a CloudWatch Logs Insights query that is in progress.
TagLogGroup	logs:TagLogGroup Required to add or update log group tags.
TestMetricFilter	logs:TestMetricFilter Required to test a filter pattern against a sampling of log event messages.

Using service-linked roles for CloudWatch Logs

Amazon CloudWatch Logs uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to CloudWatch Logs. Service-

linked roles are predefined by CloudWatch Logs and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CloudWatch Logs more efficient because you aren't required to manually add the necessary permissions. CloudWatch Logs defines the permissions of its service-linked roles, and unless defined otherwise, only CloudWatch Logs can assume those roles. The defined permissions include the trust policy and the permissions policy. That permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#). Look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch Logs

CloudWatch Logs uses the service-linked role named **AWSServiceRoleForLogDelivery**. CloudWatch Logs uses this service-linked role to write logs directly to Firehose. For more information, see [Enable logging from AWS services](#).

The **AWSServiceRoleForLogDelivery** service-linked role trusts the following services to assume the role:

- `logs.amazonaws.com`

The role permissions policy allows CloudWatch Logs to complete the following actions on the specified resources:

- Action: `firehose:PutRecord` and `firehose:PutRecordBatch` on all Firehose streams that have a tag with a `LogDeliveryEnabled` key with a value of `True`. This tag is automatically attached to an Firehose stream when you create a subscription to deliver the logs to Firehose.

You must configure permissions to allow an IAM entity to create, edit, or delete a service-linked role. This entity could be a user, group, or role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a service-linked role for CloudWatch Logs

You aren't required to manually create a service-linked role. When you set up logs to be sent directly to a Firehose stream in the AWS Management Console, the AWS CLI, or the AWS API, CloudWatch Logs creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you again set up logs to be sent directly to a Firehose stream, CloudWatch Logs creates the service-linked role for you again.

Editing a service-linked role for CloudWatch Logs

CloudWatch Logs does not allow you to edit **AWSServiceRoleForLogDelivery**, or any other service-linked role, after you create it. You cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for CloudWatch Logs

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note

If the CloudWatch Logs service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CloudWatch Logs resources used by the **AWSServiceRoleForLogDelivery** service-linked role

- Stop sending logs directly to Firehose streams.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForLogDelivery** service-linked role. For more information, see [Deleting a Service-Linked Role](#)

Supported Regions for CloudWatch Logs service-linked roles

CloudWatch Logs supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see [CloudWatch Logs Regions and Endpoints](#).

CloudWatch Logs updates to AWS service linked roles

View details about updates to AWS service linked role for CloudWatch Logs since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Logs Document history page.

Change	Description	Date
AWSServiceRoleForLogDelivery service-linked role policy – Update to an existing policy	CloudWatch Logs changed the permissions in the IAM policy associated with the AWSServiceRoleForLogDelivery service-linked role. The following change was made: The firehose: ResourceTag/LogDeliveryEnabled": "true" condition key was changed to aws:ResourceTag/LogDeliveryEnabled": "true" .	July 15, 2021
CloudWatch Logs started tracking changes	CloudWatch Logs started tracking changes for its AWS managed policies.	June 10, 2021

Compliance validation for Amazon CloudWatch Logs

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](#).

Resilience in Amazon CloudWatch Logs

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through 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 Amazon CloudWatch Logs

As a managed service, Amazon CloudWatch Logs 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 CloudWatch Logs 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.

Using CloudWatch Logs with interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and CloudWatch Logs. You can use this

connection to send logs to CloudWatch Logs without sending them through the internet. CloudWatch Logs supports IPv4 VPC endpoints in all Regions, and supports IPv6 endpoints in all Regions.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. To connect your VPC to CloudWatch Logs, you define an *interface VPC endpoint* for CloudWatch Logs. This type of endpoint enables you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to CloudWatch Logs without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see [What is Amazon VPC](#) in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see [New – AWS PrivateLink for AWS Services](#).

The following steps are for users of Amazon VPC. For more information, see [Getting Started](#) in the *Amazon VPC User Guide*.

Availability

CloudWatch Logs currently supports VPC endpoints in all AWS Regions, including the AWS GovCloud (US) Regions.

Creating a VPC endpoint for CloudWatch Logs

To start using CloudWatch Logs with your VPC, create an interface VPC endpoint for CloudWatch Logs. The service to choose is **com.amazonaws.*Region*.logs**. To connect with a FIPS endpoint, the service to choose is `com.amazonaws.Region.logs-fips`. You do not need to change any settings for CloudWatch Logs. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

Some CloudWatch Logs APIs, such as `StartLiveTail` and `GetLogObject`, are hosted under a different endpoint and VPC endpoint: `stream-logs.Region.amazonaws.com`. To create an interface VPC endpoint for these APIs, the service to choose is **com.amazonaws.*Region*.stream-logs**. To connect with a FIPS endpoint, the service to choose is `com.amazonaws.Region.stream-logs-fips`.

Testing the connection between your VPC and CloudWatch Logs

After you create the endpoint, you can test the connection.

To test the connection between your VPC and your CloudWatch Logs endpoint

1. Connect to an Amazon EC2 instance that resides in your VPC. For information about connecting, see [Connect to Your Linux Instance](#) or [Connecting to Your Windows Instance](#) in the Amazon EC2 documentation.
2. From the instance, use the AWS CLI to create a log entry in one of your existing log groups.

First, create a JSON file with a log event. The timestamp must be specified as the number of milliseconds after Jan 1, 1970 00:00:00 UTC.

```
[
  {
    "timestamp": 1533854071310,
    "message": "VPC Connection Test"
  }
]
```

Then, use the `put-log-events` command to create the log entry:

```
aws logs put-log-events --log-group-name LogGroupName --log-stream-
name LogStreamName --log-events file://JSONFileName
```

If the response to the command includes `nextSequenceToken`, the command has succeeded and your VPC endpoint is working.

Controlling access to your CloudWatch Logs VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, we attach a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch Logs. This policy enables users connecting to CloudWatch Logs through the VPC to create log streams and send logs to CloudWatch Logs, and prevents them from performing other CloudWatch Logs actions.

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To modify the VPC endpoint policy for CloudWatch Logs

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch Logs, choose **Create Endpoint**. Then select **com.amazonaws.*Region*.logs** and choose **Create endpoint**.
4. Select the **com.amazonaws.*Region*.logs** endpoint, and choose the **Policy** tab in the lower half of the screen.
5. Choose **Edit Policy** and make the changes to the policy.

Support for VPC context keys

CloudWatch Logs supports the `aws:SourceVpc` and `aws:SourceVpce` context keys that can limit access to specific VPCs or specific VPC endpoints. These keys work only when the user is using VPC endpoints. For more information, see [Keys Available for Some Services](#) in the *IAM User Guide*.

Logging CloudWatch Logs API and console operations in AWS CloudTrail

Amazon CloudWatch Logs 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 CloudWatch Logs as events. The calls captured include calls from the CloudWatch Logs console and code calls to the CloudWatch Logs API operations. Using the information collected by CloudTrail, you can determine the request that was made to CloudWatch Logs, 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 or a [CloudTrail Lake](#) event data store.

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](#).

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to [Apache ORC](#) format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying [advanced event selectors](#). The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see [Working with AWS CloudTrail Lake](#) in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the [pricing option](#) you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see [AWS CloudTrail Pricing](#).

CloudWatch Logs supports logging the following actions as events in CloudTrail log files:

- [AssociateKmsKey](#)
- [CancelExportTask](#)
- [CreateDelivery](#)
- [CreateExportTask](#)
- [CreateLogAnomalyDetector](#)
- [CreateLogGroup](#)
- [CreateLogStream](#)
- [DeleteAccountPolicy](#)
- [DeleteDataProtectionPolicy](#)
- [DeleteDelivery](#)
- [DeleteDeliveryDestination](#)
- [DeleteDeliveryDestinationPolicy](#)

- [DeleteDeliverySource](#)
- [DeleteDestination](#)
- [DeleteIndexPolicy](#)
- [DeleteIntegration](#)
- [DeleteLogAnomalyDetector](#)
- [DeleteLogGroup](#)
- [DeleteLogStream](#)
- [DeleteMetricFilter](#)
- [DeleteQueryDefinition](#)
- [DeleteResourcePolicy](#)
- [DeleteRetentionPolicy](#)
- [DeleteSubscriptionFilter](#)
- [DeleteTransformer](#)
- [DescribeAccountPolicies](#)
- [DescribeConfigurationTemplates](#)
- [DescribeDeliveries](#)
- [DescribeDeliveryDestinations](#)
- [DescribeDeliverySources](#)
- [DescribeDestinations](#)
- [DescribeExportTasks](#)
- [DescribeFieldIndexes](#)
- [DescribeIndexPolicies](#)
- [DescribeLogGroups](#)
- [DescribeLogStreams](#)
- [DescribeMetricFilters](#)
- [DescribeQueries](#)
- [DescribeQueryDefinitions](#)
- [DescribeResourcePolicies](#)
- [DescribeSubscriptionFilters](#)

- [DisassociateKmsKey](#)
- [FilterLogEvents](#)
- [GetDataProtectionPolicy](#)
- [GetDelivery](#)
- [GetDeliveryDestination](#)
- [GetDeliveryDestinationPolicy](#)
- [GetDeliverySource](#)
- [GetIntegration](#)
- [GetLogAnomalyDetector](#)
- [GetLogEvents](#)
- [GetLogGroupFields](#)
- [GetLogRecord](#)
- [GetQueryResults](#)
- [GetTransformer](#)
- [ListAnomalies](#)
- [ListIntegrations](#)
- [ListLogAnomalyDetectors](#)
- [ListLogGroups](#)
- [ListLogGroupsForQuery](#)
- [ListTagsForResource](#)
- [ListTagsLogGroup](#)
- [PutAccountPolicy](#)
- [PutDataProtectionPolicy](#)
- [PutDeliveryDestination](#)
- [PutDeliveryDestinationPolicy](#)
- [PutDeliverySource](#)
- [PutDestination](#)
- [PutDestinationPolicy](#)
- [PutIndexPolicy](#)

- [PutIntegration](#)
- [PutMetricFilter](#)
- [PutQueryDefinition](#)
- [PutResourcePolicy](#)
- [PutRetentionPolicy](#)
- [PutSubscriptionFilter](#)
- [PutTransformer](#)
- [StartLiveTail](#)
- [StartQuery](#)
- [StopQuery](#)
- [TagResource](#)
- [TestMetricFilter](#)
- [TestTransformer](#)
- [UntagResource](#)
- [UpdateAnomaly](#)
- [UpdateDeliveryConfiguration](#)
- [UpdateLogAnomalyDetector](#)

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 or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Query generation information in CloudTrail

CloudTrail logging for Query generator console events is also supported. Query generator is currently supported for CloudWatch Logs Insights and CloudWatch Metric Insights. In these CloudTrail events, the `eventSource` is `monitoring.amazonaws.com`.

The following example shows a CloudTrail log entry that demonstrates the **GenerateQuery** action in CloudWatch Logs Insights.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111222333444:role/Administrator",
        "accountId": "123456789012",
        "userName": "SAMPLE_NAME"
      },
      "attributes": {
        "creationDate": "2020-04-08T21:43:24Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2020-04-08T23:06:30Z",
  "eventSource": "monitoring.amazonaws.com",
  "eventName": "GenerateQuery",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "exampleUserAgent",
  "requestParameters": {
    "query_ask": "****",
    "query_type": "LogsInsights",
    "logs_insights": {
      "fields": "****",
      "log_group_names": ["yourloggroup"]
    },
    "include_description": true
  },
  "responseElements": null,
  "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
  "eventID": "52723fd9-4a54-478c-ac55-1234567890",
}
```

```
"readOnly": true,  
"eventType": "AwsApiCall",  
"managementEvent": true,  
"recipientAccountId": "111122223333",  
"eventCategory": "Management"  
}
```

Understanding log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following log file entry shows that a user called the CloudWatch Logs **CreateExportTask** action.

```
{  
  "eventVersion": "1.03",  
  "userIdentity": {  
    "type": "IAMUser",  
    "principalId": "EX_PRINCIPAL_ID",  
    "arn": "arn:aws:iam::123456789012:user/someuser",  
    "accountId": "123456789012",  
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
    "userName": "someuser"  
  },  
  "eventTime": "2016-02-08T06:35:14Z",  
  "eventSource": "logs.amazonaws.com",  
  "eventName": "CreateExportTask",  
  "awsRegion": "us-east-1",  
  "sourceIPAddress": "127.0.0.1",  
  "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",  
  "requestParameters": {  
    "destination": "yourdestination",  
    "logGroupName": "yourloggroup",  
    "to": 123456789012,  
    "from": 0,  
    "taskName": "yourtask"  
  },  
  "responseElements": {
```

```
        "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
    },
    "requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
    "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
    "eventType": "AwsApiCall",
    "apiVersion": "20140328",
    "recipientAccountId": "123456789012"
}
```

Monitoring with CloudWatch metrics

You can use the tables in this section to review the metrics that Amazon CloudWatch Logs sends to Amazon CloudWatch every minute.

For centralized monitoring across multiple accounts and regions, you can use CloudWatch Logs Centralization to consolidate log data and metrics in a central location. For more information, see [Cross-account cross-Region log centralization](#).

CloudWatch Logs metrics

The AWS/Logs namespace includes the following metrics.

Metric	Description
CallCount	The number of specified API operations performed in your account. <code>CallCount</code> is a CloudWatch Logs service usage metric. For more information, see CloudWatch Logs service usage metrics. Valid Dimensions: Class, Resource, Service, Type Valid Statistic: Sum Units: None
DeliveryErrors	The number of log events for which CloudWatch Logs received an error when forwarding data to the subscription destination. If the destination service returns a retryable error such as a throttling exception or a retryable service exception (HTTP 5xx for example), CloudWatch Logs continues to retry delivery for up to 24 hours. CloudWatch Logs does not try to re-deliver if the error is a non-retryable error, such as <code>AccessDeniedException</code> or <code>ResourceNotFoundException</code>. Valid Dimensions: LogGroupName, DestinationType, FilterName, PolicyLevel Valid Statistic: Sum

Metric	Description
	Units: None
DeliveryThrottling	The number of log events for which CloudWatch Logs was throttled when forwarding data to the subscription destination. If the destination service returns a retryable error such as a throttling exception or a retryable service exception (HTTP 5xx for example), CloudWatch Logs continues to retry delivery for up to 24 hours. CloudWatch Logs does not try to re-deliver if the error is a non-retryable error, such as <code>AccessDeniedException</code> or <code>ResourceNotFoundException</code>. Valid Dimensions: LogGroupName, DestinationType, FilterName, PolicyLevel Valid Statistic: Sum Units: None
EMFDisabledErrors	The number of EMF-formatted log events that were ignored due to an active account policy of type <code>METRIC_EXTRACTION_POLICY</code> that matches the log group. For more information about the metric extraction policies, see PutAccountPolicy. Valid Dimensions: LogGroupName Valid Statistic: Sum Units: None

Metric	Description
EMFParsingErrors	The number of parsing errors encountered while processing embedded metric format logs. Such errors happen when logs are identified as embedded metric format but do not follow the correct format. For more information about the embedded metric format, see Specification: Embedded metric format. Valid Dimensions: LogGroupName Valid Statistic: Sum Units: None
EMFValidationErrors	The number of validation errors encountered while processing embedded metric format logs. These errors occur when metric definitions within embedded metric format logs do not adhere to the embedded metric format and <code>MetricDatum</code> specifications. For information about the CloudWatch embedded metric format, see Specification: Embedded metric format. For information about the data type <code>MetricDatum</code>, see MetricDatum in the <i>Amazon CloudWatch API Reference</i>.  Note Certain validation errors can lead to multiple metrics within an EMF log not being published. For example, all metrics set with an invalid namespace will be dropped. Valid Dimensions: LogGroupName Valid Statistic: Sum Units: None

Metric	Description
ErrorCount	The number of API operations performed in your account that resulted in errors. ErrorCount is a CloudWatch Logs service usage metric. For more information, see CloudWatch Logs service usage metrics. Valid Dimensions: Class, Resource, Service, Type Valid Statistic: Sum Units: None
ForwardedBytes	The volume of log events in compressed bytes forwarded to the subscription destination. Valid Dimensions: LogGroupName, DestinationType, FilterName Valid Statistic: Sum Units: Bytes
Forwarded LogEvents	The number of log events forwarded to the subscription destination. Valid Dimensions: LogGroupName, DestinationType, FilterName, PolicyLevel Valid Statistic: Sum Units: None

Metric	Description
IncomingBytes	The volume of log events in uncompressed bytes uploaded to CloudWatch Logs. When used with the LogGroupName dimension, this is the volume of log events in uncompressed bytes uploaded to the log group. Valid Dimensions: LogGroupName Valid Statistic: Sum Units: Bytes
IncomingLogEvents	The number of log events uploaded to CloudWatch Logs. When used with the LogGroupName dimension, this is the number of log events uploaded to the log group. Valid Dimensions: LogGroupName Valid Statistic: Sum Units: None
LogEventsWithFindings	The number of log events that matched a data string that you are auditing using the CloudWatch Logs data protection feature. For more information, see Help protect sensitive log data with masking. Valid Dimensions: None Valid Statistic: Sum Units: None

Metric	Description
ThrottleCount	The number of API operations performed in your account that were throttled because of usage quotas. ThrottleCount is a CloudWatch Logs service usage metric. For more information, see CloudWatch Logs service usage metrics. Valid Dimensions: Class, Resource, Service, Type Valid Statistic: Sum Units: None

Dimensions for CloudWatch Logs metrics

The dimensions that you can use with most CloudWatch Logs metrics are listed in the following table.

Dimension	Description
LogGroupName	The name of the CloudWatch Logs log group for which to display metrics.
DestinationType	The subscription destination for the CloudWatch Logs data, which can be AWS Lambda, Amazon Kinesis Data Streams, or Amazon Data Firehose.
FilterName	The name of the subscription filter that is forwarding data from the log group to the destination. The subscription filter name is automatically converted by CloudWatch to ASCII and any unsupported characters get replaced with a question mark (?).

Subscription filter metric dimensions

The dimensions for metrics related to account-level subscription filters are listed in the following table.

Dimension	Description
PolicyLevel	The level where the policy applies. Currently, the only valid value for this dimension is AccountPolicy
DestinationType	The subscription destination for the CloudWatch Logs data, which can be AWS Lambda, Amazon Kinesis Data Streams, or Amazon Data Firehose.
FilterName	The name of the subscription filter that is forwarding data from the log group to the destination. The subscription filter name is automatically converted by CloudWatch to ASCII and any unsupported characters get replaced with a question mark (?).

Log transformer metrics and dimensions

CloudWatch Logs publishes the following log transformer metrics to CloudWatch in the AWS/Logs namespace.

Metric	Description
TransformationErrors	The number of errors encountered while transforming log events with the specified transformer. Unit: None Valid statistic: Sum
TransformedBytes	The volume of the output of transformed log events, in uncompressed bytes. Unit: Bytes Valid statistic: Sum
TransformedLogEvents	The number of transformed log events.

Metric	Description
	Unit: None
	Valid statistic: Sum

The following dimensions are used by transformer metrics.

Dimension	Description
LogGroupName	This dimension is used only for log-group-level transformers.
PolicyLevel	This dimension is used only for account-level transformers. Currently the only valid value for this dimension is AccountPolicy

CloudWatch Logs service usage metrics

CloudWatch Logs sends metrics to CloudWatch that track the usage of CloudWatch Logs API operations. These metrics correspond to AWS service quotas. Tracking these metrics can help you proactively manage your quotas. For more information, see [Service Quotas Integration and Usage Metrics](#).

For example, you could track the `ThrottleCount` metric or set an alarm on that metric. If the value of this metric rises, you should consider requesting a quota increase for the API operation that is getting throttled. For more information about CloudWatch Logs service quotas, see [CloudWatch Logs quotas](#).

CloudWatch Logs publishes service quota usage metrics every minute in both the `AWS/Usage` and `AWS/Logs` namespaces.

The following table lists the service usage metrics published by CloudWatch Logs. These metrics do not have a specified unit. The most useful statistic for these metrics is `SUM`, which represents the total operation count for the 1-minute period.

Each of these metrics is published with values for all of the `Service`, `Class`, `Type`, and `Resource` dimensions. They are also published with a single dimension called `Account Metrics`. Use the

Account Metrics dimension to see the sum of metrics for all API operations in your account. Use the other dimensions and specify the name of an API operation for the **Resource** dimension to find the metrics for that particular API.

Metrics

Metric	Description
CallCount	The number of specified operations performed in your account. CallCount is published in both the AWS/Usage and AWS/Logs namespaces.
ErrorCount	The number of API operations performed in your account that resulted in errors. ErrorCount is published in only the AWS/Logs.
ThrottleCount	The number of API operations performed in your account that were throttled because of usage quotas. ThrottleCount is published in only the AWS/Logs.

Dimensions

Dimension	Description
Account metrics	Use this dimension to get a sum of the metric across all of the CloudWatch Logs APIs. If you want to see the metrics for one particular API, use the other dimensions listed in this table and specify the API name as the value of Resource.
Service	The name of the AWS service containing the resource. For CloudWatch Logs usage metrics, the value for this dimension is Logs.
Class	The class of resource being tracked. CloudWatch Logs API usage metrics use this dimension with a value of None.

Dimension	Description
Type	The type of resource being tracked. Currently, when the Service dimension is Logs, the only valid value for Type is API.
Resource	The name of the API operation. Valid values include all of the API operation names that are listed in Actions . For example, PutLogEvents

CloudWatch Logs quotas

You can use the table in this section to review the default service quotas, also referred to as limits, for an AWS account in Amazon CloudWatch Logs. Most of the service quotas, but not all, are listed under the Amazon CloudWatch Logs namespace in the Service Quotas console.

Note

If you want to request a quota increase for any of these quotas, see [the procedure](#) in this section.

Name	Default	Adjustable	Description
Active export task	Each supported Region: 1	No	The number of active (running or pending) export tasks per account
AssociateKmsKey throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of associate-kms-key calls per second per account/per region
Batch size	Each supported Region: 1 Megabytes	No	The maximum batch size in MB of a put-log-events request
CancelExportTask throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of cancel-export-task calls per second per account/per region
CreateExportTask throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of create-export-task calls per second per account/per region

Name	Default	Adjustable	Description
CreateLogGroup throttle limit in transactions per second	Each supported Region: 10 per second	Yes	The maximum number of create-log-group calls per second per account/per region
CreateLogStream throttle limit in transactions per second	Each supported Region: 50 per second	Yes	The maximum number of create-log-stream calls per second per account/per region
Data archiving	Each supported Region: 5 Gigabytes	No	The maximum size in GB of free data archiving
DeleteDestination throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of delete-destination calls per second per account/per region
DeleteLogGroup throttle limit in transactions per second	Each supported Region: 10 per second	Yes	The maximum number of delete-log-group calls per second per account/per region
DeleteLogStream throttle limit in transactions per second	Each supported Region: 15 per second	Yes	The maximum number of delete-log-stream calls per second per account/per region
DeleteMetricFilter throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of delete-metric-filter calls per second per account/per region

Name	Default	Adjustable	Description
DeleteRetentionPolicy throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of delete-retention-policy calls per second per account/per region
DeleteSubscriptionFilter throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of delete-subscription-filter calls per second per account/per region
DeleteTransformer throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of delete-transformer calls per second per account/per region
DescribeDestinations throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of describe-destinations calls per second per account/per region
DescribeExportTasks throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of describe-export-tasks calls per second per account/per region
DescribeLogGroups throttle limit in transactions per second	Each supported Region: 10 per second	Yes	The maximum number of describe-log-groups calls per second per account/per region
DescribeLogStreams throttle limit in transactions per second	Each supported Region: 25 per second	Yes	The maximum number of describe-log-streams calls per second per account/per region

Name	Default	Adjustable	Description
DescribeMetricFilters throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of describe-metric-filters calls per second per account/per region
DescribeSubscriptionFilters throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of describe-subscription-filters calls per second per account/per region
Event size	Each supported Region: 1,024 Kilobytes	No	The maximum log event size in KB
FilterLogEvents throttle limit in transactions per second	us-east-1: 25 per second ap-northeast-3: 5 per second ap-southeast-3: 5 per second eu-central-1: 5 per second Each of the other supported Regions: 10 per second	No	The maximum number of filter-log-events calls per second per account/per region

Name	Default	Adjustable	Description
GetLogEvents throttle limit in transactions per second	us-west-2: 10 per second ap-northeast-3: 10 per second ap-southeast-3: 10 per second eu-central-1: 10 per second eu-west-1: 10 per second eu-west-3: 30 per second Each of the other supported Regions: 25 per second	No	The maximum number of get-log-events calls per second per account/per region
GetQueryResults throttle limit in transactions per second	Each supported Region: 6	No	The maximum number of get-query-results calls per second per account/per region
GetTransformer throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of get-transformer calls per second per account/per region

Name	Default	Adjustable	Description
ListTagsForResource throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of list-tags-for-resource calls per second per account/per region
ListTagsLogGroup throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of list-tags-log-group calls per second per account/per region
Live Tail concurrent sessions limit	Each supported Region: 15	Yes	The maximum number of concurrent active Live Tail sessions per account
Log groups	Each supported Region: 1,000,000	Yes	The maximum number of log groups an account can have
Log groups scanned per Live Tail session	Each supported Region: 10	No	The maximum number of log groups that can be scanned per Live Tail session
Metrics filters per log group	Each supported Region: 100	No	The number of metric filters per log group
PutDestination throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of put-destination calls per second per account/per region
PutDestinationPolicy throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of put-destination-policy calls per second per account/per region

Name	Default	Adjustable	Description
PutLogEvents throttle limit in transactions per second	Each supported Region: 5,000 per second	Yes	The maximum number of put-log-events calls per second per account/per region
PutMetricFilter throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of put-metric-filter calls per second per account/per region
PutRetentionPolicy throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of put-retention-policy calls per second per account/per region
PutSubscriptionFilter throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of put-subscription-filter calls per second per account/per region
PutTransformer throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of put-transformer calls per second per account/per region
Resource policies	Each supported Region: 10	No	The maximum number of resource policies per account/per region
StartLiveTail throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of start-live-tail calls per second per account/per region. This limit applies independently to both console and API

Name	Default	Adjustable	Description
StartQuery throttle limit in transactions per second	Each supported Region: 6	No	The maximum number of start-query calls per second per account/per region
Subscription filters per log group	Each supported Region: 2	No	The number of subscription filters per log group
TagLogGroup throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of tag-log-group calls per second per account/per region
TagResource throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of tag-resource calls per second per account/per region
TestMetricFilter throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of test-metric-filter calls per second per account/per region
TestTransformer throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of test-transformer calls per second per account/per region
UntagLogGroup throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of untag-log-group calls per second per account/per region

Name	Default	Adjustable	Description
UntagResource throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of untag-resource calls per second per account/per region
logs anomaly detectors	Each supported Region: 500	Yes	The maximum number of active log anomaly detectors

Managing your CloudWatch Logs service quotas

CloudWatch Logs has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see [What Is Service Quotas?](#) in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of your CloudWatch Logs service quotas.

AWS Management Console

To view CloudWatch Logs service quotas using the console

1. Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon CloudWatch Logs**.

In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.

4. To view additional information about a service quota, such as the description, choose the quota name.
5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the console see the [Service Quotas User Guide](#). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

AWS CLI

To view CloudWatch Logs service quotas using the AWS CLI

Run the following command to view the default CloudWatch Logs quotas.

```
aws service-quotas list-aws-default-service-quotas \
  --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
  --service-code logs \
  --output table
```

To work more with service quotas using the AWS CLI, see the [Service Quotas AWS CLI Command Reference](#). To request a quota increase, see the [request-service-quota-increase](#) command in the [AWS CLI Command Reference](#).

Document history

The following table describes important changes in each release of the CloudWatch Logs User Guide, beginning in June 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
Updated CloudWatchLogsReadOnlyAccess policy	Permissions for observability administration actions were added to CloudWatchLogsReadOnlyAccess to allow read-only access to telemetry pipelines and S3 table integrations.	December 2, 2025
Updated CloudWatchLogsFullAccess policy	Permissions for observability administration actions were added to CloudWatchLogsFullAccess to allow read-only access to telemetry pipelines and S3 table integrations.	December 2, 2025
New log management feature in CloudWatch Logs	CloudWatch consolidates log management into a single service with built-in governance capabilities without storing and maintaining multiple copies of the same data across different tools and data stores. For more information, see Log management .	December 2, 2025
New data ingestion and normalization feature in CloudWatch Logs	CloudWatch automatically collects AWS vended logs across accounts and AWS	December 2, 2025

Regions through integrating with AWS Organizations and pre-built connectors for third-party sources. For more information, see [Transform logs during ingestion](#).

[New analytics and querying based on data sources feature in CloudWatch Logs Insights](#)

CloudWatch Insights introduces facets for queries. Facets are useful for analyzing logs as they allow you to interactively filter and group your data without running queries. A facet is a field in your logs (such as ServiceName or StatusCode) that enables filtering, aggregation and analysis across log groups. For more information, see [Use facets to group and explore logs](#).

December 2, 2025

[New S3 Tables Integration](#)

The S3 Tables Integration with CloudWatch allows you to access log data ingested into CloudWatch using analytics engines such as Amazon Athena, Amazon Redshift, and third-party tools that support connection to Apache Iceberg-compatible stores. For more information, see [Access logs with S3 Tables Integration](#).

December 2, 2025

[CloudWatch Logs added support for Cross-account cross-Region log centralization](#)

CloudWatch Logs added support for the **Cross-account cross-Region log centralization** that enables CloudWatch Logs to gather log data from AWS Organizations member accounts into a central account for analysis. For more information, see [Cross-account cross-Region log centralization](#).

September 17, 2025

[CloudWatch Logs anomaly detection updated example policy](#)

CloudWatch Logs updated the KMS example policy to reduce its scope and improve security to by adding conditions for `aws:SourceAccount` and `aws:SourceArn` . For more information, see [Encrypt an anomaly detector and its results with AWS KMS](#)

July 14, 2025

[CloudWatch Logs Insights adds support for Amazon VPC Route Server logs](#)

CloudWatch Logs Insights adds support for Amazon VPC Route Server logs. For more information, see [Enable logging from AWS services](#). The new log source for Amazon VPC Route Server, `EVENT_LOGS` is documented in [PutDeliverySource](#)

June 12, 2025

[CloudWatch Logs Insights adds support for AWS PCS logs](#)

CloudWatch Logs Insights adds support for AWS PCS logs. For more information, see [Enable logging from AWS services](#). The new log sources for AWS PCS, PCS_SCHEDULER_LOGS and PCS_JOBCOMP_LOGS are documented in [PutDeliverySource](#)

June 12, 2025

[CloudWatch Logs Insights adds support for AWS Entity Resolution logs](#)

CloudWatch Logs Insights adds support for AWS Entity Resolution logs. For more information, see [Enable logging from AWS services](#). The new log source for AWS Entity Resolution, WORKFLOW_LOGS is documented in [PutDeliverySource](#)

May 22, 2025

[CloudWatch Logs managed policy updates to support natural language summaries](#)

Permissions for `cloudwatch:GenerateQueryResultsSummary` were added to **CloudWatchLogsFullAccess** and **CloudWatchLogsReadOnlyAccess** allow for generation of a natural language summary of the query results. To see the contents of the policies, see [CloudWatchLogsFullAccess](#) and [CloudWatchLogsReadOnlyAccess](#) in the *AWS Managed Policy Reference Guide*.

May 20, 2025

[CloudWatch Logs Insights adds support for generating natural language summaries from CloudWatch Logs Insights query results](#)

Added support for natural language summaries in CloudWatch Logs Insights. This feature generates human-readable summaries of query results, currently available in US East (N. Virginia). For more information, see [Generate natural language summaries from CloudWatch Logs Insights query results](#).

May 20, 2025

[CloudWatchOpenSearchDashboardsAccess and CloudWatchOpenSearchDashboardsFullAccess are new IAM policies](#)

CloudWatch Logs added two new IAM policies, **CloudWatchOpenSearchDashboardsFullAccess** and **CloudWatchOpenSearchDashboardsAccess**. **CloudWatchOpenSearchDashboardsFullAccess** grants permission to create and manage integrations with OpenSearch Service. **CloudWatchOpenSearchDashboardsAccess** grants access to view vended logs dashboards that are created in these integrations. For more information, see [Vended log dashboards powered by Amazon OpenSearch Service](#).

December 1, 2024

[CloudWatchLogsFullAccess policy updated](#)

CloudWatch Logs added permissions for Amazon OpenSearch Service and IAM to the **CloudWatchLogsFullAccess** policy, to enable CloudWatch Logs integration with OpenSearch Service for some features.

December 1, 2024

[CloudWatch Logs Insights adds new structure types to the query syntax](#)

CloudWatch Logs Insights adds the unnest command and two JSON functions, which enable you to operate JSON strings as maps and lists. For more information, see [Structure types](#).

November 21, 2024

[CloudWatch Logs supports log transformation during log ingestion](#)

You can create log transformers which can modify log events at the time of ingestion, helping you normalize your logs in different formats and from different sources into consistent and context-rich formats. For more information, see [Transform logs during ingestion](#).

November 20, 2024

[CloudWatch Logs Insights adds field indexing](#)

CloudWatch Logs Insights has added support for field indexing of logs. When you then use a field index in a CloudWatch Logs Insights query, the query attempts to skip processing log events that are known to not include the indexed field. For more information, see [Create field indexes to improve query performance and reduce scan volume](#).

November 20, 2024

[CloudWatch Logs Insights support for natural language query generation is generally available](#)

CloudWatch Logs Insights supports natural language to generate and update queries. For more information, see [Use natural language to generate and update CloudWatch Logs Insights queries](#).

June 20, 2024

[CloudWatchLogsRead OnlyAccess policy updated](#)

CloudWatch Logs added the `cloudwatch:GenerateQuery` permission to **CloudWatchLogsRead OnlyAccess**, so that users with this policy can generate a [CloudWatch Logs Insights](#) query string from a natural language prompt.

November 26, 2023

[CloudWatchLogsFullAccess policy updated](#)

CloudWatch Logs added the `cloudwatch:GenerateQuery` permission to **CloudWatchLogsFullAccess**, so that users with this policy can generate a [CloudWatch Logs Insights](#) query string from a natural language prompt.

November 26, 2023

[CloudWatch Logs adds log pattern analysis](#)

CloudWatch Logs now scans for patterns in log events every time you perform a CloudWatch Logs Insights query. For more information, see [Pattern analysis](#).

November 26, 2023

[CloudWatch Logs adds log anomaly detection](#)

You can create a log anomaly detector for a log group. The anomaly detector scans the log events ingested into the log group and finds anomalies in the log data. For more information, see [Log anomaly detection](#).

November 26, 2023

[CloudWatch Logs adds compare feature](#)

You can now use CloudWatch Logs Insights to compare changes in your log events over time. . For more information, see [Compare \(diff\) with previous time ranges](#).

November 26, 2023

[CloudWatch Logs adds a new log class](#)

CloudWatch Logs supports two classes of log groups so that you can have a cost-effective option for logs that you access infrequently, and you also have a full-featured option for logs that require real-time monitoring or other features. For more information, see [Log classes](#).

November 26, 2023

[CloudWatch Logs Insights supports natural language query generation](#)

CloudWatch Logs Insights supports natural language to generate and update queries. For more information, see [Use natural language to generate and update CloudWatch Logs Insights queries](#).

November 26, 2023

[CloudWatch Logs adds regular expression filter pattern syntax support for Live Tail](#)

You can now further customize your search and match operations to meet your needs with flexible regular expressions within Live Tail filter patterns. For more information, see [Filter pattern syntax](#) in the *Amazon CloudWatch Logs User Guide*.

November 13, 2023

[CloudWatch Logs adds regular expression filter pattern syntax support for metric filters, subscription filters, and filter log events](#)

You can now further customize your search and match operations to meet your needs with flexible regular expressions within filter patterns. For more information, see [Filter pattern syntax](#) in the *Amazon CloudWatch Logs User Guide*.

September 5, 2023

[CloudWatch Logs Insights adds a pattern command](#)

You can now use **pattern** in your CloudWatch Logs Insights queries to automatically cluster your log data into patterns. A pattern is shared text structure that recurs among your log fields. For more information, see [pattern](#) in the *Amazon CloudWatch Logs User Guide*.

July 17, 2023

[CloudWatch Logs Insights adds a dedup command](#)

You can now use **dedup** in your CloudWatch Logs Insights queries to remove duplicate results based on specific values in fields that you specify. For more information, see [dedup](#) in the *Amazon CloudWatch Logs User Guide*.

June 20, 2023

[Account-level data protection policies](#)

You can now set data protection policies at the account level. These account-level policies can audit and mask sensitive information in log events in all log groups in the account. For more information, see [Help protect sensitive log data with masking](#) in the *Amazon CloudWatch Logs User Guide*.

June 8, 2023

[Live Tail feature added](#)

CloudWatch Logs added Live Tail ability, so you can scan logs as they are ingested to help with troubleshooting. You can optionally filter the displayed stream of log events based on specified terms, and also highlight log events that have specified terms. For more information, see [Use live tail to view logs in near real time](#).

June 6, 2023

CloudWatchLogsReadOnlyAccess policy updated	CloudWatch Logs added permissions to CloudWatchLogsReadOnlyAccess . The <code>logs:StartLiveTail</code> and <code>logs:StopLiveTail</code> permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time .	June 6, 2023
CloudWatch Logs Insights released	You can use CloudWatch Logs Insights to interactively search and analyze your log data. For more information see Analyze Log Data with CloudWatch Logs Insights in the <i>Amazon CloudWatch Logs User Guide</i>	November 27, 2018
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and CloudWatch Logs. For more information, see Using CloudWatch Logs with Interface VPC Endpoints in the <i>Amazon CloudWatch Logs User Guide</i> .	June 28, 2018

The following table describes the important changes to the Amazon CloudWatch Logs User's Guide.

Change	Description	Release date
Interface VPC endpoints	In some Regions, you can use an interface VPC endpoint to keep traffic between your Amazon VPC and CloudWatch Logs from leaving the Amazon network. For more information see Using CloudWatch Logs with interface VPC endpoints .	March 7, 2018
Route 53 DNS query logs	You can use CloudWatch Logs to store logs about the DNS queries received by Route 53. For more information see What is Amazon CloudWatch Logs? or Logging DNS Queries in the Amazon Route 53 Developer Guide.	September 7, 2017
Tag log groups	You can use tags to categorize your log groups. For more information, see Tag log groups in Amazon CloudWatch Logs .	December 13, 2016
Console improvements	You can navigate from metrics graphs to the associated log groups. For more information, see Pivot from metrics to logs .	November 7, 2016
Console usability improvements	Improved the experience to make it easier to search, filter, and troubleshoot. For example, you can now filter your log data to a date and time range. For more information, see View log data sent to CloudWatch Logs .	August 29, 2016
Added AWS CloudTrail support for Amazon CloudWatch Logs and new CloudWatch Logs metrics	Added AWS CloudTrail support for CloudWatch Logs. For more information, see Logging CloudWatch Logs API and console operations in AWS CloudTrail .	March 10, 2016

Change	Description	Release date
Added support for CloudWatch Logs export to Amazon S3	Added support for exporting CloudWatch Logs data to Amazon S3. For more information, see Exporting log data to Amazon S3 .	December 7, 2015
Added support for AWS CloudTrail logged events in Amazon CloudWatch Logs	You can create alarms in CloudWatch and receive notifications of particular API activity as captured by CloudTrail and use the notification to perform troubleshooting.	November 10, 2014
Added support for Amazon CloudWatch Logs	You can use Amazon CloudWatch Logs to monitor, store, and access your system, application, and custom log files from Amazon Elastic Compute Cloud (Amazon EC2) instances or other sources. You can then retrieve the associated log data from CloudWatch Logs using the Amazon CloudWatch console, the CloudWatch Logs commands in the AWS CLI, or the CloudWatch Logs SDK. For more information, see What is Amazon CloudWatch Logs? .	July 10, 2014

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.