Amazon CloudWatch Logs

User Guide

Amazon CloudWatch Logs: User Guide

Table of Contents

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, and you can query them and sort them based on other dimensions, group them by specific fields, create custom computations with a powerful query language, and visualize log data in dashboards.

Features

- **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 (p. 34).
- **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 "NullReferenceException") 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 (p. 5).
- **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*.
- **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.
- **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*.

Related AWSservices

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 Amazon CloudWatch Logs API calls in AWS CloudTrail (p. 186).

- **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, see Amazon CloudWatch Pricing.

Amazon CloudWatch Logs concepts

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

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.

Getting set up

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 Amazon Web Services

When you create an AWS account, we automatically sign up your account for all AWS services. You pay only for the services that you use.

If you have an AWS account already, skip to the next step. If you don't have an AWS account, use the following procedure 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 and entering a verification code on the phone keypad.

Sign in to the Amazon CloudWatch console

To sign in to the Amazon CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the Region. From the navigation bar, choose the Region where you have your AWS resources.
3. In the navigation pane, choose **Logs**.

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*.

Getting started with CloudWatch Logs

To collect logs from your Amazon EC2 instances and on-premises servers into CloudWatch Logs, AWS offers two options:

- **Recommended** – 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 your server uses Instance Metadata Service Version 2 (IMDSv2), you must use the newer unified agent instead of the older CloudWatch Logs agent.

  If you're using the unified 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*.
- **Supported, but on the path to deprecation** – The older CloudWatch Logs agent, which supports the collection of logs from only servers running Linux. If you're already using that agent, you may continue to do so. However, the older agent requires Python 2.7, 3.0, and 3.3. Because current EC2 instances do not use those versions of Python and those versions are deprecated and are no longer being patched, **we strongly recommend that you migrate to the unified CloudWatch agent.**

  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.

  When you migrate from the CloudWatch Logs agent to the unified CloudWatch agent, the unified agent's configuration wizard can read your current CloudWatch Logs agent configuration file and set up the new 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*.

Contents

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 Logs agent to get started with CloudWatch Logs

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*. Alternatively, you can continue using the previous CloudWatch Logs agent.

Contents

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

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 (p. 5).

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 Linux Instances*.

   For more information about connection issues, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
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 (p. 189).
5. By default, the `/etc/awslogs/awscli.conf` points to the us-east-1 Region. To push your logs to a different Region, edit the `awscli.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 (p. 61).

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 Linux Instances*.

   For more information about connection issues, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
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.
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 (p. 189).

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 (p. 61).

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 (p. 189).

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 for Linux Instances*.
4. On the **Review Policy** page, for **Policy Name**, type a name for the policy.
5. For **Policy Document**, paste in the following policy:

```
{
    "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:::myawsbucket/*"
            ]
        }
    ]
}
```

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 IAM 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 for Linux Instances*.
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://DOC-EXAMPLE-BUCKET1/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 (p. 61).

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 (p. 5).
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

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

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\W3SVC`*n*).

   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 user and 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. You can optionally create a unique IAM user account for configuring and running Systems Manager. 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 for Windows Instances*.

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 for Windows Instances*.
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 (p. 5).
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

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\W3SVC`*n*).

   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. For more information about using an earlier version of EC2Config, see Use EC2Config 3.x or Earlier to Configure CloudWatch in the *Amazon EC2 User Guide for Windows Instances*

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.

Quick Start: Install the CloudWatch Logs agent using AWS OpsWorks and Chef

You can install the CloudWatch Logs agent and create log streams using AWS OpsWorks and Chef, which is a third-party systems and cloud infrastructure automation tool. Chef uses "recipes," which you write to install and configure software on your computer, and "cookbooks," which are collections of recipes, to perform its configuration and policy distribution tasks. For more information, see Chef.

The Chef recipes examples below show how to monitor one log file on each EC2 instance. The recipes use the stack name as the log group and the instance's hostname as the log stream name. To monitor multiple log files, you need to extend the recipes to create multiple log groups and log streams.

Step 1: Create custom recipes

Create a repository to store your recipes. AWS OpsWorks supports Git and Subversion, or you can store an archive in Amazon S3. The structure of your cookbook repository is described in Cookbook Repositories in the *AWS OpsWorks User Guide*. The examples below assume that the cookbook is named `logs`. The install.rb recipe installs the CloudWatch Logs agent. You can also download the cookbook example (CloudWatchLogs-Cookbooks.zip).

Create a file named metadata.rb that contains the following code:

```
#metadata.rb

name              'logs'
version           '0.0.1'
```

Create the CloudWatch Logs configuration file:

```
#config.rb

template "/tmp/cwlogs.cfg" do
  cookbook "logs"
  source "cwlogs.cfg.erb"
  owner "root"
  group "root"
  mode 0644
end
```

Download and install the CloudWatch Logs agent:

```
# install.rb

directory "/opt/aws/cloudwatch" do
  recursive true
end

remote_file "/opt/aws/cloudwatch/awslogs-agent-setup.py" do
  source "https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py"
  mode "0755"
end
```

```
  execute "Install CloudWatch Logs agent" do
  command "/opt/aws/cloudwatch/awslogs-agent-setup.py -n -r region -c /tmp/cwlogs.cfg"
  not_if { system "pgrep -f aws-logs-agent-setup" }
end
```

> **Note**
> In the above example, replace *region* with one of the following: 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.

If the installation of the agent fails, check to make sure that the `python-dev` package is installed. If it isn't, use the following command, and then retry the agent installation:

```
sudo apt-get -y install python-dev
```

This recipe uses a cwlogs.cfg.erb template file that you can modify to specify various attributes such as what files to log. For more information about these attributes, see CloudWatch Logs agent reference (p. 189).

```
[general]
# Path to the AWSLogs agent's state file. Agent uses this file to maintain
# client side state across its executions.
state_file = /var/awslogs/state/agent-state


## Each log file is defined in its own section. The section name doesn't
## matter as long as its unique within this file.
#
#[kern.log]
#
## Path of log file for the agent to monitor and upload.
#
#file = /var/log/kern.log
#
## Name of the destination log group.
#
#log_group_name = kern.log
#
## Name of the destination log stream.
#
#log_stream_name = {instance_id}
#
## Format specifier for timestamp parsing.
#
#datetime_format = %b %d %H:%M:%S
#
#

[<%= node[:opsworks][:stack][:name] %>]
datetime_format = [%Y-%m-%d %H:%M:%S]
log_group_name = <%= node[:opsworks][:stack][:name].gsub(' ','_') %>
file = <%= node[:cwlogs][:logfile] %>
log_stream_name = <%= node[:opsworks][:instance][:hostname] %>
```

The template gets the stack name and host name by referencing the corresponding attributes in the stack configuration and deployment JSON. The attribute that specifies the file to log is defined in the cwlogs cookbook's default.rb attributes file (logs/attributes/default.rb).

```
default[:cwlogs][:logfile] = '/var/log/aws/opsworks/opsworks-agent.statistics.log'
```

Step 2: Create an AWS OpsWorks stack

1. Open the AWS OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the **OpsWorks Dashboard**, choose **Add stack** to create an AWS OpsWorks stack.
3. On the **Add stack** screen, choose **Chef 11 stack**.
4. For **Stack name**, enter a name.
5. For **Use custom Chef Cookbooks**, choose **Yes**.
6. For **Repository type**, select the repository type that you use. If you're using the above example, choose **Http Archive**.
7. For **Repository URL**, enter the repository where you stored the cookbook that you created in the previous step. If you're using the above example, enter **`https://s3.amazonaws.com/aws-cloudwatch/downloads/CloudWatchLogs-Cookbooks.zip`**.
8. Choose **Add Stack** to create the stack.

Step 3: Extend your IAM role

To use CloudWatch Logs with your AWS OpsWorks instances, you need to extend the IAM role used by your instances.

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, under **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 for Linux Instances*.
4. On the **Review Policy** page, for **Policy Name**, type a name for the policy.
5. For **Policy Document**, paste in the following policy:

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

6. Choose **Create Policy**.
7. In the navigation pane, choose **Roles**, and then in the contents pane, for **Role Name**, select the name of the instance role used by your AWS OpsWorks stack. You can find the one used by your stack in the stack settings (the default is `aws-opsworks-ec2-role`).

    Note
    Choose the role name, not the check box.

8. On the **Permissions** tab, under **Managed Policies**, choose **Attach Policy**.

9. On the **Attach Policy** page, in the table header (next to **Filter** and **Search**), choose **Policy Type**, **Customer Managed Policies**.
10. For **Customer Managed Policies**, select the IAM policy that you created above and choose **Attach Policy**.

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

Step 4: Add a layer

1. Open the AWS OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. In the navigation pane, choose **Layers**.
3. In the contents pane, select a layer and choose **Add layer**.
4. On the **OpsWorks** tab, for **Layer type**, choose **Custom**.
5. For the **Name** and **Short name** fields, enter the long and short name for the layer, and then choose **Add layer**.
6. On the **Recipes** tab, under **Custom Chef Recipes**, there are several headings—*Setup*, *Configure*, *Deploy*, *Undeploy*, and *Shutdown*—that correspond to AWS OpsWorks lifecycle events. AWS OpsWorks triggers these events at these key points in instance's lifecycle, which runs the associated recipes.

    Note
    If the above headings aren't visible, under **Custom Chef Recipes**, choose **edit**.
7. Enter *logs::config, logs::install* next to **Setup**, choose **+** to add it to the list, and then choose **Save**.

    AWS OpsWorks runs this recipe on each of the new instances in this layer, right after the instance boots.

Step 5: Add an instance

The layer only controls how to configure instances. You now need to add some instances to the layer and start them.

1. Open the AWS OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. In the navigation pane, choose **Instances** and then under your layer, choose **+ Instance**.
3. Accept the default settings and choose **Add Instance** to add the instance to the layer.
4. In the row's **Actions** column, click **start** to start the instance.

    AWS OpsWorks launches a new EC2 instance and configures CloudWatch Logs. The instance's status changes to online when it's ready.

Step 6: View your logs

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 (p. 61).

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 Linux Instances*.

   For more information about connection issues, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*
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 Linux Instances*.

   For more information about connection issues, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
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 Linux Instances*.

   For more information about connection issues, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
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
```

Quick Start: Use AWS 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 AWS 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 AWS 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 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 .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples

For examples specific to CloudWatch Logs, see Code examples for CloudWatch Logs using AWS SDKs (p. 147).

Example availability
Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Analyzing log data with CloudWatch Logs Insights

CloudWatch Logs Insights enables you to 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. If an issue occurs, you can use CloudWatch Logs Insights to identify potential causes and validate deployed fixes.

CloudWatch Logs Insights includes a purpose-built query language with a few simple but powerful commands. CloudWatch Logs Insights provides 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.

CloudWatch Logs Insights automatically discovers 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.

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

A single request can query up to 20 log groups. Queries time out after 15 minutes, if they have not completed. Query results are available for 7 days.

You can save queries that you have created. This can help you run complex queries when you need, without having to re-create them each time that you want to run them.

CloudWatch Logs Insights queries incur charges based on the amount of data that is queried. For more information, see Amazon CloudWatch Pricing.

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 logs and discovered fields

CloudWatch Logs Insights supports different log types. For every log that's sent to 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.

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 ephemeral fields for use in that query. For more information, see CloudWatch Logs Insights query syntax (p. 40).

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`.

Log type	Discovered log fields
Amazon VPC flow logs	`@timestamp`, `@logStream`, `@message`, `accountId`, `endTime`, `interfaceId`, `logStatus`, `startTime`, `version`, `action`, `bytes`, `dstAddr`, `dstPort`, `packets`, `protocol`, `srcAddr`, `srcPort`
Route 53 logs	`@timestamp`, `@logStream`, `@message`, `edgeLocation`, `hostZoneId`, `protocol`, `queryName`, `queryTimestamp`, `queryType`, `resolverIp`, `responseCode`, `version`
Lambda logs	`@timestamp`, `@logStream`, `@message`, `@requestId`, `@duration`, `@billedDuration`, `@type`, `@maxMemoryUsed`, `@memorySize` If a Lambda log line contains an X-Ray trace ID, it also includes the following fields: `@xrayTraceId` and `@xraySegmentId`. 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 extract the log fields by using the **`parse`** command. For more information, see Fields in JSON logs (p. 36).
CloudTrail logs Logs in JSON format	For more information, see Fields in JSON logs (p. 36).

Log type	Discovered log fields
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": "205.251.233.176",
    "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.

Tutorial: Run and modify a sample query

The following tutorial helps you get started with CloudWatch Logs Insights. You run a sample query, 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 (p. 5).

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 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.

   You can type the name of log groups you want to query in the search bar.

   When you select a log group, 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.
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 Saving and re-running CloudWatch Logs Insights queries (p. 54).

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 (").

CloudWatch Logs Insights includes powerful query abilities, including several commands and support for regular expressions, mathematical, and statistical operations. For more information, see CloudWatch Logs Insights query syntax (p. 40).

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

   You can enter the name of log groups that you want to query in the search bar. When you select a log group, CloudWatch Logs Insights automatically detects the data fields in the log group.
4. Delete the default query in the query editor, and enter the following command:

```
stats count(*) by fieldName
```

5. 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.
6. 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. Select one or more log groups to query.
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. Select one or more log groups to query.
4. 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.
5. 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.

CloudWatch Logs Insights query syntax

CloudWatch Logs Insights supports a query language that you can use to query your log groups. The query syntax supports different functions and operations that include but aren't limited to general functions, arithmetic and comparison operations, and regular expressions. You can create queries that contain multiple query commands. Separate query commands in your queries with Unix-style pipe characters (|). For more information about the query syntax, see Supported operations and functions.

CloudWatch Logs Insights supports the use of comments in queries. CloudWatch Logs Insights ignores lines that start with the hash symbol (#). CloudWatch Logs Insights automatically discovers fields for many log types and generates fields that start with the @ symbol. For more information about the fields that CloudWatch Logs automatically generates, see Supported logs and discovered fields in the *Amazon CloudWatch User Guide*.

CloudWatch Logs Insights query commands

The following table lists the supported query commands for CloudWatch Logs Insights and includes basic examples. For examples of general queries and queries for other log types, see Sample queries (p. 55) in the *Amazon CloudWatch Logs User Guide*.

Command	Description	Example(s)

`display`	Specifies which fields to display in the query results. If you use this command more than once in your query, the query results show only the fields that you specified in the last occurrence of the `display` command being used.	The following example query contains the field `@message`. The `parse` command extracts values from the message log format and creates the ephemeral fields `loggingType` and `loggingMessage`. The query filters log events to only those with `ERROR` as the value for `loggingType` and displays only the `loggingMessage` of those events in the results.
		`fields @message`
		`    | parse @message "[*] *" as loggingType, loggingMessage`
		`    | filter loggingType = "ERROR"`
		`    | display loggingMessage`

`fields`	Retrieves and displays specified fields from log events. If your query doesn't contain the `display` command, the query results display the fields that are specified for the `fields` command.
	
	You can use functions and operations in the `fields` command to modify field values for display and to create new fields for use in the rest of the query.
		The following example displays the fields `foo-bar`, `action`, and the absolute value of the difference between `f3` and `f4` for all log events in the log group.
		``fields `foo-bar`, action, abs(f3-f4)``
		
		The following example creates and displays an ephemeral field `opStatus`. The value of `opStatus` for each log entry is the concatenation of the values of the `Operation` and `StatusCode` fields, with a hyphen in between those values.
		`fields concat(Operation, '-', StatusCode) as opStatus`

`filter`	Filters the results of a query that's based on one or more conditions. The **`filter`** command supports a variety of operators and expressions. For more information, see the section called “Matches and regular expressions in the filter command” (p. 45).	The following example retrieves the `f1`, `f2`, and `f3` fields for all log events with a value greater than 2000 in the `duration` field. `fields f1, f2, f3 | filter (duration>2000)` The following example shows a similar query, but the query results don't display

Command	Description	Example(s)

		separate fields. Instead, the query results display the `@timestamp` field and all log data in the `@message` field for all log events where duration is greater than 2000.
		
		`filter (duration>2000)`
		
		The following example retrieves `f1` and `f2` fields for all log events where `f1` equals 10 or `f3` is greater than 25.
		
		`fields f1, f2 | filter (f1=10 or`
		` f3>25)`
		
		The following example returns log events where the field `statusCode` contains a value between 200 and 299.
		
		`fields f1 | filter statusCode`
		` like /2\d\d/`
		
		The next example returns log events where the field `statusCode` doesn't contain a value between 200 and 299.
		
		`fields f1 | filter statusCode not`
		` like /2\d\d/`
		
		The following example returns log events that have a `statusCode` of "300", "400", or "500".
		
		`fields @timestamp, @message`
		`    | filter statusCode in`
		` [300,400,500]`
		
		This final example returns log events that do not have `Type` fields with values of "foo", "bar", or "1".
		
		`fields @timestamp, @message`
		`    | filter Type not in`
		` ["foo","bar",1]`

Command	Description	Example(s)

`stats`	Uses log field values to calculate aggregate statistics. You can use `by` with the `stats` command to specify one or more criteria. You can use the criteria to group statistical data.
	
	The `stats` command supports the following operators: `sum()`, `avg()`, `count()`, `min()`, and `max()`.
		The following example calculates the average value of `f1` for each unique value of `f2`.
		
		`stats avg (f1) by f2`
		
		The following example counts the number of exceptions per hour.
		
		`filter @message like /Exception/`
		`| stats count(*) as exceptionCount by bin(1h)`
		`| sort exceptionCount desc`
		
		In the example, `as` groups the total count for the number of times the word ***"Exception"*** occurs in messages under the the ephemeral field **exceptionCount**, and `by` groups the statistical data under the field **bin(1h)**.

`sort`	Sorts the retrieved log events. Both ascending (`asc`) and descending (`desc`) order are supported.	The following example sorts the returned events in descending order based on the value of `f1`, and displays the fields `f1`, `f2`, and `f3`. `fields f1, f2, f3 | sort f1 desc`
`limit`	Specifies the number of log events returned by the query. You can use this to limit the results to a small number to see a small set of relevant results. You can also use `limit` with a number between 1000 and 10,000 to increase the number of query result rows displayed in the console to an amount greater than the default of 1000 rows. If you don't specify a limit, the query defaults to displaying a maximum of 1000 rows.	The following example sorts the events in descending order based on the value of `@timestamp`, and displays the fields `f1` and `f2` for the first 25 events by sort order. In this case, the sort order is by timestamp starting with the most recent, so the most recent 25 events are returned. `sort @timestamp desc | limit 25 | display f1, f2`

Command	Description	Example(s)

`parse`	Extracts data from a log field and creates one or more ephemeral fields that you can process further in the query.
	
	The `parse` command supports glob expressions and regular expressions.
	
	For a glob expression, use a constant string (characters enclosed in single or double quotation marks) with the `parse` command. Replace variable text with an asterisk (*). The asterisk functions as a wild card. Use the keyword `as` following the asterisk to extract wild card values into ephemeral fields and give them an alias.
	
	Enclose regular expressions in forward slashes (/). In the expression, each part of the matched string that is to be extracted is enclosed in a named capturing group. An example of a named capturing group is `(?<name>.*)`, where `name` is the name and `.*` is the pattern.
		Using this single log line as an example:
		
		`25 May 2019 10:24:39,474 [ERROR] {foo=2, bar=data} The error was: DataIntegrityException`
		
		The following two `parse` expressions each do the following: the ephemeral fields `level`, `config`, and `exception` are created. `level` has a value of `ERROR`, `config` has a value of `{foo=2, bar=data}`, and `exception` has a value of `DataIntegrityException`. The first example uses a glob expression, and the second uses a regular expression.
		
		`parse @message "[*] * The error was: *" as level, config, exception`
		
		`parse @message /\[(?<level>\S+)\]\s+(?<config>\{.*\})\s+The error was: (?<exception>\S+)/`
		
		Using this single log line as an example:
		
		`25 May 2019 10:24:39,474 user=user1234, method=sampleMethod, latency := 216`
		
		The following example uses a regular expression to extract the ephemeral fields `user2`, `method2`, and `latency2` from the log field `@message` and returns 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`

Guidelines for working with query commands

You must surround log fields named in queries that include characters other than the `@` symbol, period (`.`), and non-alphanumeric characters in backtick keys (`` ` ``). For example, the log field `foo-bar` must be enclosed in backtick kets (`` `foo-bar` ``) because it contains a non-alphanumeric character, the hyphen (`-`).

Use the `display` command to show the field or fields that you want to see in your query results. The `display` command only shows 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.

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

The value of `isRes` equals 0 or 1, depending on whether `resolverArn` is a discovered field . If your query contains multiple `fields` commands and doesn't include a `display` command, you'll display all of the fields that are specified in the `fields` commands.

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`).

> **Note**
> We assume that you're familiar with regular expressions. CloudWatch Logs Insights supports Hyperscan, a mutiple regular expression matching library. For more information about Hyperscan, see the Hyperscan website.

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 last example matches a substring with a regular expression.

```
fields f1, f2, f3
| filter f1 =~ /Exception/
```

Example: Match substrings with wild cards

You can use the asterisk symbol (`*`) as a wild card in regular expressions to match substrings. The following example returns log events where `f1` contains words that begin with the letter ***E***. The example is case sensitive.

```
fields f1, f2, f3
| filter f3 like /E*/
```

> **Note**
> You can place a period before the asterisk symbol (**.***) to create a greedy quantifier that returns as many matches as possible.

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 senstive.

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

Example: Match substrings with case insensitve patterns

You can match substrings that are case insensitve 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/
```

Using aliases in queries

You can use `as` to create one or more aliases in a query. Aliases are supported in the `fields`, `stats`, and `sort` commands.

You can create aliases for log fields and for the results of operations and functions.

Examples

The following examples show the use of aliases in query commands.

```
fields abs(myField) as AbsoluteValuemyField, myField2
```

Returns the absolute value of `myField` as `AbsoluteValuemyField` and also returns the field `myField2`.

```
stats avg(f1) as myAvgF1 | sort myAvgF1 desc
```

Calculates the average of the values of the `f1` as `myAvgF1` and returns them in descending order by that value.

Using comments in queries

You can comment out lines in a query by using the # character. Lines that start with the # character are ignored. This can be useful to document your query or to temporarily ignore part of a complex query for one call, without deleting that line.

In the following example, the second line of the query is ignored.

```
fields @timestamp, @message
    # | filter @message like /delay/
    | limit 20
```

Supported operations and functions

The query language supports many types of operations and functions, as shown in the following tables.

Comparison operations

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

```
= != < <= > >=
```

Boolean operators

You can use the Boolean operators **`and`**, **`or`**, and **`not`**. You can use these Boolean operators only in functions that return a Boolean value.

Arithmetic operations

You can use arithmetic operations in the `filter` and `fields` commands and as arguments for other functions. Arithmetic operations accept numeric data types as arguments and return numeric results.

Operation	Description
`a + b`	Addition
`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`

Numeric operations

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

Operation	Result type	Description
`abs(a: number)`	number	Absolute value.
`ceil(a: number)`	number	Round to ceiling (the smallest integer that is greater than the value of `a`).

Operation	Result type	Description
`floor(a: number)`	number	Round to floor (the largest integer that is smaller than the value of a).
`greatest(a: number, ...numbers: number[])`	number	Returns the largest value.
`least(a: number, ...numbers: number[])`	number	Returns the smallest value.
`log(a: number)`	number	Natural log.
`sqrt(a: number)`	number	Square root.

General functions

You can use general functions in the `filter` and `fields` commands and as arguments for other functions.

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

String functions

You can use string functions in the `filter` and `fields` commands and as arguments for other functions.

Function	Result type	Description
`isempty(fieldName: string)`	Number	Returns `1` if the field is missing or is an empty string.
`isblank(fieldName: string)`	Number	Returns `1` if the field is missing, an empty string, or contains only white space.
`concat(str: string, ...strings: string[])`	string	Concatenates the strings.

`ltrim(str: string)`
 	
`ltrim(str: string, trimChars: string)`
	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 `trimChars` from the left of `str`. For example,

Function	Result type	Description
		`ltrim("xyZxyfooxyZ","xyZ")` returns `"fooxyZ"`.

`rtrim(str: string)`
 	
`rtrim(str: string, trimChars: string)`
	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. Instead, it removes the characters of `trimChars` from the right of `str`. For example, `rtrim("xyZfooxyxyZ","xyZ")` returns `"xyZfoo"`.

`trim(str: string)`
 	
`trim(str: string, trimChars: string)`
	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 `trimChars` from both sides of `str`. For example, `trim("xyZxyfooxyxyZ","xyZ")` returns `"foo"`.

`strlen(str: string)`	number	Returns the length of the string in Unicode code points.
`toupper(str: string)`	string	Converts the string to uppercase.
`tolower(str: string)`	string	Converts the string to lowercase.

`substr(str: string, startIndex: number)`
 	
`substr(str: string, startIndex: number, length: number)`
	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, `substr("xyZfooxyZ",3, 3)` returns `"foo"`.

Function	Result type	Description
`replace(fieldName: string, searchValue: string, replaceValue: string)`	string	Replaces all instances of `searchValue` in `fieldName: string` with `replaceValue`. For example, the function `replace(logGroup,"smoke_test","Smoke` searches for log events where the field `logGroup` contains the string value `smoke_test` and replaces the value with the string `Smoke`.
`strcontains(str: string, searchValue: string)`	number	Returns 1 if `str` contains `searchValue` and 0 otherwise.

Datetime functions

You can use datetime functions in the `filter` and `fields` commands and as arguments for other functions. You can use these functions to create time buckets for queries with aggregate functions. You also can use time periods that consist of a number and either `m` for minutes or `h` for hours. For example, `10m` is 10 minutes, and `1h` is 1 hour. The following table contains a list of the different date time functions that you can use in your query commands. The table lists each function's result type and contains a description of each function.

> **Note**
> 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. For information about how to run a query command, see Tutorial: Run and modify a sample query in the Amazon CloudWatch Logs *User Guide*.

Function	Result type	Description
`bin(period: Period)`	Timestamp	Rounds the value of `@timestamp` to the given time period and then truncates. For example, `bin(5m)` rounds the value of `@timestamp` to 5 minutes before it truncates.
`datefloor(timestamp: Timestamp, period: Period)`	Timestamp	Truncates the timestamp to the given period. For example, `datefloor(@timestamp, 1h)` truncates all values of `@timestamp` to the bottom of the hour.
`dateceil(timestamp: Timestamp, period: Period)`	Timestamp	Rounds up the timestamp to the given period and then truncates. For example, `dateceil(@timestamp, 1h)` truncates all values of `@timestamp` to the top of the hour.
`fromMillis(fieldName: number)`	Timestamp	Interprets the input field as the number of milliseconds since the Unix epoch and converts it to a timestamp.
`toMillis(fieldName: Timestamp)`	number	Converts the timestamp found in the named field into a number representing the milliseconds since the Unix epoch.

Function	Result type	Description
		For example, `toMillis(@timestamp)` converts the timestamp `2022-01-14T13:18:031.000-08:00` to `1642195111000`.

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

IP address functions

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

Function	Result type	Description
`isValidIp(fieldName: string)`	boolean	Returns `true` if the field is a valid IPv4 or IPv6 address.
`isValidIpV4(fieldName: string)`	boolean	Returns `true` if the field is a valid IPv4 address.
`isValidIpV6(fieldName: string)`	boolean	Returns `true` if the field is a valid IPv6 address.
`isIpInSubnet(fieldName: string, subnet: string)`	boolean	Returns `true` 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 `192.0.2.0/24` or `2001:db8::/32`.
`isIpv4InSubnet(fieldName: string, subnet: string)`	boolean	Returns `true` if the field is a valid IPv4 address within the specified v4 subnet. When you specify the subnet, use CIDR notation such as `192.0.2.0/24`.
`isIpv6InSubnet(fieldName: string, subnet: string)`	boolean	Returns `true` if the field is a valid IPv6 address within the specified v6 subnet. When you specify the subnet, use CIDR notation such as `2001:db8::/32`.

Stats aggregation functions

You can use aggregation functions in the `stats` command and as arguments for other functions.

Function	Result type	Description
`avg(fieldName: NumericLogField)`	number	The average of the values in the specified field.

`count()`
 	
`count(fieldName: LogField)`
	number	Counts the log events. `count()` (or `count(*)`) counts all events returned by the query, while `count(fieldName)` counts all records that include the specified field name.

Function	Result type	Description
`count_distinct(fieldName: LogField)`	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 `count_distinct` is just an approximation.
`max(fieldName: LogField)`	LogFieldValue	The maximum of the values for this log field in the queried logs.
`min(fieldName: LogField)`	LogFieldValue	The minimum of the values for this log field in the queried logs.
`pct(fieldName: LogFieldValue, percent: number)`	LogFieldValue	A percentile indicates the relative standing of a value in a dataset. For example, `pct(@duration, 95)` returns the `@duration` value at which 95 percent of the values of `@duration` are lower than this value, and 5 percent are higher than this value.
`stddev(fieldName: NumericLogField)`	number	The standard deviation of the values in the specified field.
`sum(fieldName: NumericLogField)`	number	The sum of the values in the specified field.

Stats non-aggregation functions

You can use non-aggregation functions in the `stats` command and as arguments for other functions.

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

Visualizing 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 Aggregation Functions in the Stats Command (p. 51).

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.

Topics

Visualizing 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 (p. 51).
- 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" (p. 40).

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 "Visualizing log data grouped by fields" (p. 53).

Visualizing 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 (p. 51).

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" (p. 39). 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
```

Saving and re-running 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 organized them. You can save as many as 1000 queries per region and per account.

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**.

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**.

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*.

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/
```

Queries for Lambda logs

Determine the amount of overprovisioned memory.

```
filter @type = "REPORT"
    | stats max(@memorySize / 1000 / 1000) as provisonedMemoryMB,
        min(@maxMemoryUsed / 1000 / 1000) as smallestMemoryRequestMB,
        avg(@maxMemoryUsed / 1000 / 1000) as avgMemoryUsedMB,
        max(@maxMemoryUsed / 1000 / 1000) as maxMemoryUsedMB,
        provisonedMemoryMB - maxMemoryUsedMB as overProvisionedMB
```

Create a latency report.

```
filter @type = "REPORT" |
    stats avg(@duration), max(@duration), min(@duration) by bin(5m)
```

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
```

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
```

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.

Find the instances that are sending the most traffic through you NAT gateway.

> **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.

```
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
```

Examples of the parse command

Use a glob expression to extract the ephemeral 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 ephemeral 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 ephemeral 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
```

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 10 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 10 concurrent CloudWatch Logs Insights queries per account, including queries added to dashboards.

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.

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.

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 groups**.
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 groups**.
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 (p. 73).
    - 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 and pattern syntax (p. 74). 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.

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 groups**.
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 groups**.
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 1482217558365]
 [--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 automatically. You can change the log retention for each log group at any time.

To change the logs retention setting

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Find the log group to update.
4. In the **Expire Events After** column for that log group, choose the current retention setting, such as **Never Expire**.
5. In **Edit Retention**, for **Retention**, choose a log retention value, and then choose **Ok**.

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
CloudWatch Logs doesn’t support IAM policies that prevent users from assigning specified tags to log groups using the `aws:Resource/key-name` or `aws:TagKeys` condition keys. For more information about using tags to control access, see Controlling access to Amazon Web Services resources using tags.

Contents

Tag basics

You use 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-log-group

Adds or updates tags for the specified log group.

list-tags-log-group

Lists the tags for the specified log group.

untag-log-group

Removes tags from the specified log group.

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.

TagLogGroup

Adds or updates tags for the specified log group.

ListTagsLogGroup

Lists the tags for the specified log group.

UntagLogGroup

Removes tags from the specified log group.

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 for the 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 customer managed key. Encryption using AWS KMS is enabled at the log group level, by associating a 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 customer managed key, and you would like to restrict the key to be used with a single account and log group, you should assign a new customer managed key that includes a condition in the IAM policy. For more information, see AWS KMS keys and encryption context (p. 70).

After you associate a customer managed 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 customer managed key whenever encrypted data is requested.

If you later disassociate a customer managed 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 customer managed key remains encrypted with the default CloudWatch Logs server-side encryption.

> **Important**
> CloudWatch Logs supports only symmetric customer managed 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 customer managed key, your encrypted data in CloudWatch Logs can no longer be retrieved.
- You cannot associate a customer managed key with a log group using the CloudWatch console.

Step 1: Create an AWS KMS customer managed key

To create an AWS KMS customer managed 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 customer managed key

By default, all AWS KMS customer managed 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 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 or log groups you specify.

First, save the default policy for your customer managed 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 (p. 70).

The `Condition` section in this example restricts the key to a single log group ARN.

```
{
 "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Your_account_ID: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": {
                    "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:region:account-
id: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.

```
{
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Your_account_ID: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": {
                "ArnLike": {
                    "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:region:account-
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 log group with a customer managed key

You can associate a customer managed key with a log group when you create it or after it exists.

To find whether a log group already has a customer managed 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 customer managed 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 customer managed 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 a log group from a CMK

To disassociate the customer managed 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 customer managed 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 customer managed key that works only for that log group, follow these steps.

To convert an encrypted log group to use a customer managed 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 customer managed 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:

   ```
   {
       "Version": "2012-10-17",
       "Id": "key-default-1",
       "Statement": [
           {
               "Sid": "Enable IAM User Permissions",
               "Effect": "Allow",
               "Principal": {
                   "AWS": "arn:aws:iam::ACCOUNT-ID: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": {
                "ArnLike": {
                    "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:REGION:ACCOUNT-
ID:log-
group:LOG-GROUP-NAME"
                }
            }
        }
    ]
}
```

5. Enter the following command to add the updated policy to the new customer managed 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*`

```
{
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Your_account_ID: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
```

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.

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. Filtered results return the first 50 lines, which will not be displayed if the timestamp on the filtered results is earlier than the metric creation time.

Contents

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 and pattern syntax

You can create metric filters to match terms in your log events and convert log data into metrics. When a metric filter matches a term, it increments the metric's count. 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 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.

Note
When you assign a unit to a metric, make sure to specify the correct one. If you change the unit later, your change might not take effect.

Topics

Using filter patterns to match terms in log events

Filter patterns make up the syntax that metric filters use to match terms in log events. Terms can be words, exact phrases, or numeric values. 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. The following examples contain code snippets that show how you can use filter patterns to match terms in your 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
```

The 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`

The 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 single and multiple terms

You can use pattern matching to create filter patterns that return log events containing single and multiple 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 ***ARGUMENTS*** and the words ***ERROR*** and ***ARGUMENTS***.

```
?ERROR ?ARGUMENTS
```

The 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 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"
```

The 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
```

The filter pattern returns log event messages, such as the following:

- `[ERROR 400] BAD REQUEST`
- `[ERROR 401] UNAUTHORIZED REQUEST`

The 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 metric filters to match terms and extract values from JSON log events

Metric filters are configurations that include filter patterns. You can create metric filters to match terms in your log events and convert log data into metrics. When your metric filter matches a term, you can increment the metric's count. Metric filters only match the terms that you define in your filter pattern. You can test metric filters in the CloudWatch console. You also can create metric filters to match terms and extract values from JSON log events. The following examples describe the syntax for metric filters that match JSON terms containing strings and numeric values.

Example: Metric filters that match strings

You can create metric filters to match strings in JSON log events. The following code snippet shows an example of the syntax for string-based metric filters.

```
{ PropertySelector EqualityOperator String }
```

Enclose metric filters in curly braces ("{}"). String-based metric filters 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. Arrays contain elements that follow a zero-based ordering system (0 = 1, 1 = 2, and so on). Enclose elements in brackets ("[]"). If a property selector points to an array or object, the metric filter won't match the log format.
- **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.

The following code snippet contains an example of a metric filter showing how you can format a metric filter to match a JSON term with a string.

```
{ $.eventType = "UpdateTrail" }
```

Example: Metric filters that match numeric values

You can create metric filters to match numeric values in JSON log events. The following code snippet shows an example of the syntax for metric filters that match numeric values.

```
{ PropertySelector NumericOperator Number }
```

Enclose metric filters in curly braces ("{}"). Metric filters 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. Arrays contain elements that follow a zero-based ordering system (0 = 1, 1 = 2, and so on). Enclose elements in brackets ("[]"). If a property selector points to an array or object, the metric filter won't match the log format.
- **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 metric filters to match JSON terms with numeric values.

```
// Metric filter with greater than symbol
{ $.bandwidth > 75 }
```

```
// Metric filter with less than symbol
{ $.latency < 50 }
// Metric filter with greater than or equal to symbol
{ $.refreshRate >= 60 }
// Metric filter with less than or equal to symbol
{ $.responseTime <= 5 }
// Metric filter with equal sign
{ $.errorCode = 400}
// Metric filter with not equal sign and scientific notation
{ $.errorCode != 500 }
// Metric filter with scientific notation and plus symbol
{ $.number[0] = 1e-3 }
// Metric filter with scientific notation and minus symbol
{ $.number[0] != 1e+3 }
```

Matching terms in JSON log events

The following examples contain code snippets that show how metric filters can match terms in a JSON log event.

Example: 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
}
```

Note
If you test the example metric filters with the example JSON log event, you must enter the example JSON log on a single line.

Example: Metric filter that matches string

The metric filter matches the string `"UpdateTrail"` in the property `"eventType"`.

```
{ $.eventType = "UpdateTrail" }
```

Example: Metric filter that matches number

The metric filter 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: Metric filter that matches element in array

The metric filter matches the element `"value"` in the array `"arrayKey"`.

```
{ $.arrayKey[0] = "value" }
```

Example: Metric filter that matches an object in array

The metric filter matches the object `"id":2` in the array `"objectList"`.

```
{ $.objectList[1].id = 2 }
```

Example: Metric filter that matches JSON logs using `IS`

You can create metric filters 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 metric filter returns JSON logs where the value of `SomeObject` is `NULL`.

```
{ $.SomeObject IS NULL }
```

Example: Metric filter that matches JSON logs using `NOT EXISTS`

You can create metric filters with the `NOT EXISTS` variable to return JSON logs that don't contain specific fields in the log data. The following metric filter 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.

Using compound expressions to match terms in JSON objects

You can use the logical operators AND ("&&") and OR ("||") in metric filters 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 metric filters with compound expressions to match terms in a JSON object.

Example: 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 (&&)

The metric filter contains a compound expression that matches `"id"` in `"user"` with a numeric value of `1` and `"users"` in `"email"` with the string `"John.Doe@example.com"`.

```
{ ($.user.id = 1) && ($.users[0].email = "John.Doe@example.com") }
```

Example: Expression that matches using OR (||)

The metric filter 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 (&&)

The metric filter contains a compound expression that doesn't find a match because the expression doesn't match the first and second coordinates in `"coordinates"` and the third action in `"actions"`.

```
{ ($.user.email = "John.Stiles@example.com" || $.coordinates[0][1] = "nonmatch") &&
 $.actions[2] = "nonmatch" }
```

Example: Expression that doesn't match using OR (||)

The metric filter 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 metric filters to extract values from space-delimited log events

You can create metric filters that map to and extract values from fields in space-delimited log events. The following examples contain code snippets that show a space-delimited log event, a metric filter that maps to the fields in the space-delimited log event, and the values that the metric filter extracts from the fields in the space-delimited log event.

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.

Example: Metric filter

To create a metric filter that maps to and extracts values from fields in a space-delimited log event, enclose the metric filter in brackets ("[]"), and specify fields with names that are separated by commas (","). The following metric filter parses seven fields.

```
[ip, user, username, timestamp, request =*.html*, status_code = 4*, bytes]
```

You can use numeric operators (>, <, =, !=, >>=, or <=) and the asterisk (*) as a wild card to give your metric filter conditions. In the example metric filter, `request` contains a wild card that states it must extract a value with `.html`, and `status_code` contains a wild card 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. Elipsis can reference as many fields as needed. The following example shows a metric filter with ellipsis that represent the first four unnamed fields shown in the previous example metric filter.

```
[..., request =*.html*, status_code = 4*, bytes]
```

You also can use the logical operators AND (&&) and OR (||) to create compound expressions. The following metric filter 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]
```

Example: Extracted fields and values

The following code snippet shows the values that the metric filter extracts from the fields in the space-delimited log event.

```
{
   "$bytes": "1534",
   "$status_code": "404",
   "$request": "GET /index.html HTTP/1.0",
   "$timestamp": "10/Oct/2000:13:25:15 -0700",
   "$username": "frank",
   "$user": "Prod",
   "$ip": "127.0.0.1"
}
```

Using pattern matching to match terms in space-delimited log events

You can use pattern matching to create space-delimited metric filters 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 metric filters.

Example: Match terms in order

The following space-delimited metric filter returns log events where the first word in the log events is ***ERROR***.

```
[w1=ERROR, w2]
```

Note

When you create space-delimited metric filters that use pattern matching, you must include a blank indicator after you specify the order of your terms. For example, if you create a metric filter 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 metric filters that contain conditions. The following metric filter 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 metric filters 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 metric filter that returns log events where the first words aren't ***ERROR*** and ***WARNING***.

```
[w1!=ERROR && w1!=WARNING, w2]
```

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 (p. 84).

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"`.

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 (p. 85)
- Example: Count log events (p. 86)
- Example: Count occurrences of a term (p. 87)
- Example: Count HTTP 404 codes (p. 88)
- Example: Count HTTP 4xx codes (p. 90)
- Example: Extract fields from an Apache log and assign dimensions (p. 91)

Create a metric filter for a log group

To create a metric filter for a log group, follow these steps.

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 the filter pattern to use. For more information, see Filter and pattern syntax (p. 74).

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 enter a name for the filter.
8. Under **Metric details**, for **Metric namespace**, enter a name for the CloudWatch namespace where the metric will be published. If this namespace doesn't already exist, be sure that **Create new** is selected.
9. For **Metric name**, enter a name for the new metric.
10. 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.
11. (Optional) For **Unit**, select a unit to assign to the metric. If you do not specify a unit, the unit is set as `None`.
12. (Optional) Enter the names and tokens for as many as three dimensions for the metric.

    If you assign dimensions to a metric created by a metric filter, you can't assign a default value for that metric.
13. Choose **Create metric filter**.

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 **`ApacheNotFoundErrorCount`**.
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=ApacheNotFoundErrorCount,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": "ApacheNotFoundErrorCount"
                }
            ],
            "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 ApacheNotFoundErrorCount.

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 (p. 88) 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 extraction rule matches the seven fields of the log event. The metric value is the value of the seventh matched token. You can see the reference to the token as "$7" in the `metricValue` field of the extraction rule.

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-transformation  \
 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-transformation  \
metricName=BytesTransferred,metricNamespace=MyNamespace,metricValue=
$size,unit=Bytes,dimensions='{EventType=$eventtype}'
```

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-transformation  \
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 Kinesis 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.

To begin subscribing to log events, create the receiving resource, such as a Kinesis 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.

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 does not try to re-deliver if the error is a non-retryable error, such as AccessDeniedException or ResourceNotFoundException.

CloudWatch Logs also produces CloudWatch metrics about the forwarding of log events to subscriptions. For more information, see Amazon CloudWatch Logs Metrics and Dimensions.

Contents

Concepts

Each subscription filter is made up of the following key elements:

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.

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 and pattern syntax (p. 74).

destination arn

The Amazon Resource Name (ARN) of the Kinesis stream, Kinesis Data 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 an Amazon Kinesis stream. By default, log data is grouped by log stream. For a more even distribution, you can group log data randomly.

Using CloudWatch Logs subscription filters

You can use a subscription filter with Kinesis, Lambda, or Kinesis Data Firehose. Logs that are sent to a receiving service through a subscription filter are base64 encoded and compressed with the gzip format.

Examples

Example 1: Subscription filters with Kinesis

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 Kinesis stream 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 Kinesis stream, calculate the volume of log data that will be generated. Be sure to create a Kinesis 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 Kinesis stream volume limits, see Amazon Kinesis Data Streams Limits.

To create a subscription filter for Kinesis

1. Create a destination Kinesis stream using the following command:

```
$ C:\>  aws kinesis create-stream --stream-name "RootAccess" --shard-count 1
```

2. Wait until the Kinesis stream becomes Active (this might take a minute or two). You can use the following Kinesis 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 Kinesis 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 (p. 125).

   ```
   {
     "Statement": {
       "Effect": "Allow",
       "Principal": { "Service": "logs.region.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.region.amazonaws.com"
                   },
                   "Condition": {
                       "StringLike": {
                           "aws:SourceArn": { "arn:aws:logs:region:123456789012:*" }
                       }
                   }
               }
           },
           "RoleId": "AAOIIAH450GAB4HC5F431",
           "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 Kinesis 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 Kinesis 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 Kinesis stream. You can verify that this is happening by grabbing a Kinesis shard iterator and using the Kinesis get-records command to fetch some Kinesis records:

```
aws kinesis get-shard-iterator --stream-name RootAccess --shard-id shardId-000000000000
 --shard-iterator-type TRIM_HORIZON
```

```
{
    "ShardIterator":
    "AAAAAAAAAAFGU/
kLvNggvndHq2UIFOw5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3IgvqOb5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f
+OIK8zM5My8ID+g6rMo7UKWeI4+IWiK2OSh0uP"
}
```

```
aws kinesis get-records --limit 10 --shard-iterator "AAAAAAAAAAFGU/
kLvNggvndHq2UIFOw5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3IgvqOb5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f
+OIK8zM5My8ID+g6rMo7UKWeI4+IWiK2OSh0uP"
```

Note that you might need to make this call a few times before Kinesis starts to return data.

You should expect to see a response with an array of records. The **Data** attribute in a Kinesis 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\":\"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 Kinesis 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.region.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\":\"Root
   \"}"
           },
           {
               "id": "31953106606966983378809025079804211143289615424298221569",
               "timestamp": 1432826855000,
               "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
   \"}"
           },
           {
               "id": "31953106606966983378809025079804211143289615424298221570",
               "timestamp": 1432826855000,
               "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"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 Kinesis 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 Kinesis Data Firehose delivery stream. Data sent from CloudWatch Logs to Amazon Kinesis Data Firehose is already compressed with gzip level 6 compression, so you do not need to use compression within your Kinesis Data Firehose delivery stream.

Note
Before you create the Kinesis Data Firehose stream, calculate the volume of log data that will be generated. Be sure to create a Kinesis Data Firehose stream that can handle this volume. If the stream cannot handle the volume, the log stream will be throttled. For more information about Kinesis Data Firehose stream volume limits, see Amazon Kinesis Data Firehose Data Limits.

To create a subscription filter for Kinesis Data 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 my-bucket --create-bucket-configuration
 LocationConstraint=region
```

   The following is example output:

```
{
    "Location": "/my-bucket"
}
```

2. Create the IAM role that will grant Amazon Kinesis Data Firehose permission to put data into your Amazon S3 bucket.

   For more information, see Controlling Access with Amazon Kinesis Data Firehose in the *Amazon Kinesis 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": "AAOIIAH450GAB4HC5F431",
        "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 Kinesis Data 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:::my-bucket",
          "arn:aws:s3:::my-bucket/" ]
    }
  ]
}
```

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 Kinesis Data 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:::my-bucket"}'
```

Note that Kinesis Data 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 Kinesis Data 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:::my-bucket",
                    "BufferingHints": {
                        "IntervalInSeconds": 300,
                        "SizeInMBs": 5
                    }
                }
            }
        ]
    }
}
```

8. Create the IAM role that will grant CloudWatch Logs permission to put data into your Kinesis Data 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 (p. 125).

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.region.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.region.amazonaws.com"
                },
                "Condition": {
                     "StringLike": {
                         "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
                     }
                 }
            }
        },
        "RoleId": "AAOIIAH450GAB4HC5F431",
        "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:*"],
        "Resource":["arn:aws:firehose:region:123456789012:*"]
      }
    ]
}
```

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 Kinesis 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 Kinesis 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 Kinesis Data Firehose delivery stream. Your data will start appearing in your Amazon S3 based on the time buffer interval set on your Amazon Kinesis 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 'my-bucket' --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 'my-bucket' --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
```

Cross-account log data sharing with 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 Kinesis Data Firehose stream (this is known as cross-account data sharing). For example, this log event data can be read from a centralized Kinesis or Kinesis Data 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 Kinesis to attach the data to their existing security analytic systems.

Topics

Cross-account log data sharing using Kinesis

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 Kinesis 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 Kinesis 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.

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

Follow the steps in these sections to set up a new cross-account log subscription.

Topics

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 Kinesis 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 Kinesis. At a command prompt, type:

```
aws kinesis create-stream --stream-name "RecipientStream" --shard-count 1
```

2. Wait until the Kinesis 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":
"49551135218688818456679503831981458784591352702l8EXAMPLE"
        }
      }
    ]
  }
}
```

It might take a minute or two for your stream to show up in the active state.

3. Create the IAM role that will grant CloudWatch Logs the permission to put data into your Kinesis 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 (p. 125).

```
{
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.region.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.region.amazonaws.com"
                }
            }
        },
        "RoleId": "AAOIIAH450GAB4HC5F431",
        "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 Kinesis 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.

```
{
    "Version" : "2012-10-17",
    "Statement" : [
        {
            "Sid" : "",
            "Effect" : "Allow",
            "Principal" : "*",
            "Action" : "logs:PutSubscriptionFilter",
            "Resource" :
 "arn:aws:logs:region: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.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "111111111111"
      },
      "Action" : "logs:PutSubscriptionFilter",
      "Resource" : "arn:aws:logs:region: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 (p. 111). If you're granting permissions directly to the other account instead of using Organizations, you can skip that step and proceed to Create a subscription filter (p. 112).

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 Create a subscription filter (p. 112).

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.

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.region.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://~/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
```

When you have finished, you can proceed to Create a subscription filter (p. 112).

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.

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 Kinesis 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`.

```
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 Kinesis 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 Kinesis 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 Kinesis shard, and using the **aws kinesis get-records** command to fetch some Kinesis records:

```
aws kinesis get-shard-iterator \
      --stream-name RecipientStream \
      --shard-id shardId-000000000000 \
      --shard-iterator-type TRIM_HORIZON

{
    "ShardIterator":
    "AAAAAAAAAAFGU/
kLvNggvndHq2UIFOw5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3IgvqOb5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f
+OIK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
}

aws kinesis get-records \
      --limit 10 \
      --shard-iterator
      "AAAAAAAAAAFGU/
kLvNggvndHq2UIFOw5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3IgvqOb5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f
+OIK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
```

Note

You might need to rerun the get-records command a few times before Kinesis starts to return data.

You should see a response with an array of Kinesis records. The data attribute in the Kinesis 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 Kinesis 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.

Modifying 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": "{\"Version\": \"2012-10-17\", \"Statement\":
 [{\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\"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:

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "222222222222"
      },
      "Action" : "logs:PutSubscriptionFilter",
      "Resource" : "arn:aws:logs:region: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

> **Note**
> This step is needed only for cross-account subscriptions for logs that are created by the services listed in Enabling logging from certain AWS services (p. 128). 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 (p. 116).

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.region.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.

```
{
    "Version" : "2012-10-17",
    "Statement" : [
        {
            "Sid" : "",
            "Effect" : "Allow",
            "Principal" : "*",
            "Action" : "logs:PutSubscriptionFilter",
            "Resource" :
 "arn:aws:logs:region: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 Enabling logging from certain AWS services (p. 128), 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 (p. 115).

```
aws logs put-destination-policy
    \ --destination-name "testDestination"
    \ --access-policy file://~/AccessPolicy.json
    \ --force
```

Cross-account log data sharing using Kinesis Data 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 Kinesis 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 Kinesis Data Firehose delivery stream with Amazon S3 storage. You can also set up Kinesis Data Firehose delivery streams with different settings. For more information, see Creating a Kinesis Data Firehose Delivery Stream.

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.

Topics

Step 1: Create a Kinesis Data Firehose delivery stream

Important
Before you complete the following steps, you must use an access policy, so Kinesis Data Firehose can access your Amazon S3 bucket. For more information, see Controlling Access in the *Amazon Kinesis 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 Kinesis Data Firehose delivery stream to be used as the destination

1. Create an Amazon S3 bucket:

```
aws s3api create-bucket --bucket firehose-test-bucket1 --create-bucket-configuration
 LocationConstraint=us-east-1
```

2. Create the IAM role that grants Kinesis Data 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. Enter the following command to create the Kinesis Data Firehose delivery stream. Replace *my-role-arn* and *my-bucket-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:::firehose-test-bucket1"}'
```

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 Kinesis Data Firehose stream that you created in Step 1: Create a Kinesis Data Firehose delivery stream (p. 118) 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:::firehose-test-bucket1",
                    "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:::firehose-test-bucket1",
                    "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 Kinesis Data 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 (p. 125).

```
{
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.us-east-1.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": "2021-02-02T08:10:43+00:00",
        "AssumeRolePolicyDocument": {
            "Statement": {
                "Effect": "Allow",
                "Principal": {
                    "Service": "logs.us-east-1.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 Kinesis Data 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 **targetAarn** 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 the log data recipient account (222222222222). You can use a text editor to put this policy in the **~/AccessPolicy.json** file:

   ```
   {
     "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: 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*.

```
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 Kinesis Data 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 Kinesis Data Firehose delivery stream. The data starts appearing in your Amazon S3 bucket based on the time buffer interval that is set on the Kinesis Data 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 'firehose-test-bucket1'
```

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 'firehose-test-bucket1' --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:

```
{
  "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.

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` or `aws:SourceAccount` global condition context keys in resource policies to limit the scope of the permissions that you grant to CloudWatch Logs to write data to Kinesis and Kinesis Data Firehose.

The value of `aws:SourceArn` must limit the permissions to only the accounts that are writing and receiving data.

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 wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:servicename::123456789012:*`.

The policies documented for granting access to CloudWatch Logs to write data to Kinesis and Kinesis Data Firehose in Create a destination (p. 107) and Step 2: Create a destination (p. 119) show how you can use the aws:SourceArn global condition context key to help prevent the confused deputy problem.

AWS services that publish logs to CloudWatch Logs

The following AWS services publish logs to CloudWatch Logs. For information about the logs that these services send, see the linked documentation.

Service	Documentation
Amazon API Gateway	Setting Up CloudWatch API Logging in API Gateway
Amazon Aurora MySQL	Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs
Amazon Chime	CloudWatch Logs for Amazon Chime
AWS CloudHSM	Monitoring AWS CloudHSM Audit Logs in Amazon CloudWatch Logs
AWS CloudTrail	Monitoring CloudTrail Log Files with Amazon CloudWatch Logs
AWS CodeBuild	Step 8: View detailed build information
Amazon Cognito	Creating the CloudWatch Logs IAM Role
Amazon Connect	Logging and Monitoring Amazon Connect
AWS DataSync	Allowing DataSync to Upload Logs to Amazon CloudWatch Log Groups
AWS Elastic Beanstalk	Using Elastic Beanstalk with Amazon CloudWatch Logs
Amazon Elastic Container Service	Using CloudWatch Logs with Container Instances
Amazon Elastic Kubernetes Service	Amazon Amazon Elastic Kubernetes Service Control Plane Logging
Amazon ElastiCache for Redis	Log delivery
AWS Fargate	Using the awslogs Log Driver
AWS Glue	Continuous Logging for AWS Glue Jobs
AWS IoT	Monitoring with CloudWatch Logs
AWS Lambda	Accessing Amazon CloudWatch Logs for AWS Lambda
Amazon Macie	Monitoring sensitive data discovery jobs with Amazon CloudWatch Logs
Amazon Managed Streaming for Apache Kafka	Logging

Service	Documentation
Amazon MQ	Configuring Amazon MQ to Publish General and Audit Logs to Amazon CloudWatch Logs
AWS Network Firewall	Logging network traffic from AWS Network Firewall
AWS OpsWorks	Using Amazon CloudWatch Logs with AWS OpsWorks Stacks
Amazon Relational Database Service	Publishing PostgreSQL Logs to CloudWatch Logs
AWS Robomaker	AWS RoboMaker's CloudWatch ROS nodes with offline support
Amazon Route 53	Logging and Monitoring in Amazon Route 53
Amazon SageMaker	Log Amazon SageMaker Events with Amazon CloudWatch
Amazon Simple Notification Service	Viewing CloudWatch Logs
AWS Step Functions	Logging using CloudWatch Logs
AWS Storage Gateway	Monitoring your file gateway
Amazon VPC	VPC Flow Logs

Enabling logging from certain 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 Kinesis 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 Kinesis Data Firehose without additional setup.

Even when logs are published directly to Amazon S3 or Kinesis Data Firehose, charges apply. For more information, see *Vended Logs* on the **Logs** tab at Amazon CloudWatch Pricing.

Permissions

Some of these AWS services use a common infrastructure to send their logs to CloudWatch Logs, Amazon S3, or Kinesis Data Firehose. To enable the AWS services listed in the following 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.

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 table summarizes which types of logs and which log destinations that the information in this section applies to.

Log type	CloudWatch Logs (p. 129)	Amazon S3 (p. 130)	Kinesis Data Firehose (p. 133)
Amazon API Gateway access logs	✓		
Amazon Chime media quality metric logs and SIP message logs	✓		
CloudFront: access logs		✓	
CloudWatch Evidently evaluation event logs	✓	✓	
Amazon ElastiCache for Redis logs	✓		✓
AWS Global Accelerator flow logs		✓	
Amazon MSK broker logs	✓	✓	✓
Amazon MSK Connect logs	✓	✓	✓
AWS Network Firewall logs	✓	✓	✓
Network Load Balancer access logs		✓	

Log type	CloudWatch Logs (p. 129)	Amazon S3 (p. 130)	Kinesis Data Firehose (p. 133)
Amazon Route 53 resolver query logs	✓	✓	✓
Amazon SageMaker events	✓		
Amazon SageMaker worker events	✓		
EC2 Spot Instance data feed files		✓	
AWS Step Functions Express Workflow and Standard Workflow logs	✓		
Storage Gateway audit logs and health logs	✓		
Amazon Virtual Private Cloud flow logs		✓	
AWS WAF logs	✓	✓	✓

The following sections provide more details for each of these destinations.

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`

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.

```
{
  "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-1:0123456789:log-group:my-log-group:log-stream:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": ["0123456789"]
        },
        "ArnLike": {
          "aws:SourceArn": ["arn:aws:logs:us-east-1:0123456789:*"]
        }
      }
    }
  ]
}
```

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.

Log group resource policy size limit considerations

These services must list each log group that they're sending logs to in the resource policy, and CloudWatch Logs resource policies are limited to 5120 characters. A service that sends logs to a large number of log groups may run into this limit.

To mitigate this, CloudWatch Logs monitors the size of resource policies used by the service that is sending logs, and when it detects that a policy approaches the size limit of 5120 characters, CloudWatch Logs automatically enables `/aws/vendedlogs/*` in the resource policy for that service. You can then start using log groups with names that start with `/aws/vendedlogs/` as the destinations for logs from these services.

Logs sent to Amazon S3

Important

When you set up the log types in the following list 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. Continue reading this section to see the details.

This section applies when the following types of logs are sent to Amazon S3:

- CloudFront access logs and streaming access logs. CloudFront uses a different permissions model than the other services in this list. For more information, see Permissions required to configure standard logging and to access your log files.
- Amazon EC2 Spot Instance data feed
- AWS Global Accelerator flow logs

- Amazon Managed Streaming for Apache Kafka broker logs
- Network Load Balancer access logs
- AWS Network Firewall logs
- Amazon Virtual Private Cloud flow logs

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.

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.

```
{
    "Version": "2012-10-17",
    "Id": "AWSLogDeliveryWrite20150319",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryAclCheck",
            "Effect": "Allow",
            "Principal": {
                "Service": "delivery.logs.amazonaws.com"
                },
            "Action": "s3:GetBucketAcl",
            "Resource": "arn:aws:s3:::my-bucket",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": ["0123456789"]
                },
                "ArnLike": {
                "aws:SourceArn": ["arn:aws:logs:us-east-1:0123456789:*"]
                }
            }
        },
        {
            "Sid": "AWSLogDeliveryWrite",
            "Effect": "Allow",
            "Principal": {
                "Service": "delivery.logs.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::my-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-1:0123456789:*"]
                }
            }
        }
    ]
}
```

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 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 `S3:GetBucketPolicy` and `S3:PutBucketPolicy` permissions for the bucket, that statement is appended to the bucket's resource policy.

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 Kinesis Data Firehose

This section applies when the types of logs listed in the table in the preceding section are sent to Kinesis Data Firehose:

User permissions

To be able to set up sending any of these types of logs to Kinesis Data 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 Kinesis Data Firehose, then to set up the sending of another one of these types of logs to Kinesis Data Firehose you need to have only the `logs:CreateLogDelivery` and `firehose:TagDeliveryStream` permissions.

IAM roles used for permissions

Because Kinesis Data Firehose does not use resource policies, AWS uses IAM roles when setting up these logs to be sent to Kinesis Data Firehose. AWS creates a service-linked role named **AWSServiceRoleForLogDelivery**. This service-linked role includes the following permissions.

```
{
    "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 Kinesis Data 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:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

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` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that CloudWatch Logs and Amazon S3 give to the services that are generating logs. If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The values of `aws:SourceArn` must be the ARNs of the AWS resources that are generating logs.

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 wildcards (`*`) for the unknown portions of the ARN.

The policies in the previous sections of this page show how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to prevent the confused deputy problem.

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
AWSServiceRoleForLogDelivery service-linked role policy (p. 133) – 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

Exporting log data to Amazon S3

You can export log data from your log groups to an Amazon S3 bucket and use this data in custom processing and analysis, or to load onto other systems.

Important
Exporting log data to Amazon S3 buckets that are encrypted by AWS KMS is not supported. Exporting log data to Amazon S3 buckets that have S3 Object Lock enabled with a retention period is not supported.

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 Amazon S3 bucket and define Amazon S3 lifecycle rules to archive or delete exported files automatically.

Exporting to S3 buckets that are encrypted with AES-256 is supported. Exporting to S3 buckets encrypted with SSE-KMS is not supported. For more information, see How Do I Enable Default Encryption for an S3 Bucket?.

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 fild data by using Linux utilities.

Log data can take up to 12 hours to become available for export. For near real-time analysis of log data, see Analyzing log data with CloudWatch Logs Insights (p. 34) or Real-time processing of log data with subscriptions (p. 94) instead.

Note
Starting on February 15, 2019, the export to Amazon S3 feature requires callers to have `s3:PutObject` access to the destination bucket.

Contents

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 Amazon 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 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 Amazon 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 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 example, 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 Amazon S3 buckets that are encrypted by AWS KMS is not supported.

Step 1: Create an 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, 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**.

Step 2: Create an IAM user with full access to Amazon S3 and CloudWatch Logs

In the following steps, you create the IAM user with necessary permissions.

To create the necessary IAM user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Users**, **Add user**.
3. Enter a user name, such as *`CWLExportUser`*.
4. Select both **Programmatic access** and **AWS Management Console access**.
5. Choose either **Autogenerated password** or **Custom password**.
6. Choose **Next: Permissions**.
7. Choose **Attach existing policies directly**, and attach the `AmazonS3FullAccess` and `CloudWatchLogsFullAccess` policies to the user. You can use the search box to find the policies.

8. Choose **Next: Tags**, **Next: Review**, and then **Create user**.

Step 3: Set permissions on an Amazon S3 bucket

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.

To set permissions on an Amazon S3 bucket

1. In the Amazon S3 console, choose the bucket that you created in step 1.
2. Choose **Permissions**, **Bucket policy**.
3. In the **Bucket Policy Editor**, add one of the following policies. Change `my-exported-logs` to the name of your S3 bucket and `random-string` to a randomly generated string of characters. Be sure to specify the correct Region endpoint for **Principal**.

   - If the bucket is in your account, add the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
          "Action": "s3:GetBucketAcl",
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs",
          "Principal": { "Service": "logs.us-west-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "Service": "logs.us-west-2.amazonaws.com" }
      }
    ]
}
```

   - If the bucket is in a different account, use the following policy instead. It includes an additional statement using the IAM user you created in the previous step.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
          "Action": "s3:GetBucketAcl",
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs",
          "Principal": { "Service": "logs.us-west-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "Service": "logs.us-west-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "AWS": "arn:aws:iam::SendingAccountID:user/CWLExportUser" }
      }
    ]
}
```

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.

Step 4: Create an export task

In this step, 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 as the IAM user that you created in **Step 2: Create an IAM user with full access to Amazon S3 and CloudWatch Logs**.
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 Amazon S3 bucket.
9. For **S3 bucket name**, choose an Amazon 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 Amazon S3 buckets that are encrypted by AWS KMS is not supported.

Step 1: Create an 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, 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 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"
}
```

Step 2: Create an IAM user with full access to Amazon S3 and CloudWatch Logs

In the following steps, you create the IAM user with necessary permissions.

To create the user and assign permissions

1. Create the IAM user by entering the following command.

   ```
   aws iam create-user --user-name CWLExportUser
   ```

2. Attach the IAM managed policies to the IAM user that you just created.

   ```
   export S3POLICYARN=$(aws iam list-policies --query 'Policies[?
   PolicyName==`AmazonS3FullAccess`].{ARN:Arn}' --output text)
   ```

   ```
   export CWLPOLICYARN=$( aws iam list-policies --query 'Policies[?
   PolicyName==`CloudWatchLogsFullAccess`].{ARN:Arn}' --output text)
   ```

   ```
   aws iam attach-user-policy --user-name CWLExportUser --policy-arn $S3POLICYARN
   ```

   ```
   aws iam attach-user-policy --user-name CWLExportUser --policy-arn $CWLPOLICYARN
   ```

3. Confirm that the two managed policies are attached.

   ```
   aws iam list-attached-user-policies --user-name CWLExportUser
   ```

4. Configure your AWS CLI to include the IAM credentials of the `CWLExportUser` IAM user. For more information, see Configuring the AWS CLI.

Step 3: Set permissions on an Amazon S3 bucket

By default, all Amazon 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.

To set permissions on an Amazon S3 bucket

1. Create a file named `policy.json` and add the following access policy, changing `Resource` to the name of your S3 bucket and `Principal` to the endpoint of the Region where you are exporting log data. Use a text editor to create this policy file. Don't use the IAM console.

   - If the bucket is in your account, use the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "s3:GetBucketAcl",
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::my-exported-logs",
            "Principal": { "Service": "logs.us-east-2.amazonaws.com" }
        },
        {
            "Action": "s3:PutObject" ,
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::my-exported-logs/*",
            "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
            "Principal": { "Service": "logs.us-east-2.amazonaws.com" }
        }
    ]
}
```

   - If the bucket is in a different account, use the following policy instead. It includes an additional statement using the IAM user you created in the previous step.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
          "Action": "s3:GetBucketAcl",
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs",
          "Principal": { "Service": "logs.us-east-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "Service": "logs.us-east-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "AWS": "arn:aws:iam::SendingAccountID:user/CWLExportUser" }
      }
    ]
}
```

- If the bucket is in a different account, and you are using an IAM role instead of an IAM user, use the following policy instead.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
          "Action": "s3:GetBucketAcl",
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs",
          "Principal": { "Service": "logs.us-east-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "Service": "logs.us-east-2.amazonaws.com" }
      },
      {
          "Action": "s3:PutObject" ,
          "Effect": "Allow",
          "Resource": "arn:aws:s3:::my-exported-logs/random-string/*",
          "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
          "Principal": { "AWS": "arn:aws:iam::SendingAccountID:role/CWLExportUser" }
      }
    ]
}
```

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 Amazon 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.

Step 4: Create an export task

After you create the export task for exporting logs from a log group, the export task might take anywhere from a few seconds to a few hours, depending on the size of the data to export.

To create an export task using the AWS CLI

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"
}
```

Step 5: Describe export tasks

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 CWLExportUser 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
    }]
}
```

Step 6: Cancel an export task

You can cancel an export task if it is in either the PENDING or the 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 CloudWatch Logs log group to stream data it receives to your Amazon OpenSearch Service cluster in near real-time through a CloudWatch Logs subscription. For more information, see Real-time processing of log data with subscriptions (p. 94).

Depending on the amount of log data being streamed, you might want to set a function-level concurrent execution limit on the function. For more information, see Lambda function scaling.

Note
Streaming large amounts of CloudWatch Logs data to OpenSearch might result in high usage charges. We recommend that you create a Budget in the Billing and Cost Management console. For more information, see Managing your costs with AWS Budgets.

Prerequisites

Before you begin, create an OpenSearch 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 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, see the Amazon OpenSearch Service Developer Guide.

Subscribe a log group to OpenSearch

You can use the CloudWatch console to subscribe a log group to OpenSearch.

To subscribe a log group to OpenSearch

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**, **Create OpenSearch 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:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "es:*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:es:region:account-id:domain/target-domain-name/*"
        }
    ]
}
```

- If the target OpenSearch 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 (p. 73).
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).

The examples are divided into the following categories:

Actions

Code excerpts that show you how to call individual service functions.

Cross-service examples

Sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch Logs with an AWS SDK (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Code examples

Actions for CloudWatch Logs using AWS SDKs

The following code examples demonstrate how to perform individual CloudWatch Logs actions with AWS SDKs. These excerpts call the CloudWatch Logs API and are not intended to be run in isolation. Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

The following examples include only the most commonly used actions. For a complete list, see the *CloudWatch Logs API Reference*.

Examples

- Cancel a CloudWatch Logs export task using an AWS SDK (p. 149)
- Create a CloudWatch Logs log group using an AWS SDK (p. 149)
- Create a CloudWatch Logs log stream using an AWS SDK (p. 150)
- Create a CloudWatch Logs subscription filter using an AWS SDK (p. 151)
- Delete a CloudWatch Logs log group using an AWS SDK (p. 154)
- Delete a CloudWatch Logs subscription filter using an AWS SDK (p. 154)
- Describe CloudWatch Logs subscription filters using an AWS SDK (p. 157)
- Describe CloudWatch Logs export tasks using an AWS SDK (p. 161)
- Describe CloudWatch Logs log groups using an AWS SDK (p. 162)

Associate an AWS KMS key to a CloudWatch Logs log group using an AWS SDK

The following code example shows how to associate an AWS KMS key with an existing CloudWatch Logs log group.

.NET

AWS SDK for .NET

```
        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/7c9eccc2-38cb-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}.");
            }
        }
```

- Find instructions and more code on GitHub.
- 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Cancel a CloudWatch Logs export task using an AWS SDK

The following code example shows how to cancel an existing CloudWatch Logs export task.

.NET

AWS SDK for .NET

```
        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.");
            }
        }
```

- Find instructions and more code on GitHub.
- 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Create a CloudWatch Logs log group using an AWS SDK

The following code example shows how to create a new CloudWatch Logs log group.

.NET

AWS SDK for .NET

```
        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.");
            }
        }
```

- Find instructions and more code on GitHub.
- For API details, see CreateLogGroup 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Create a CloudWatch Logs log stream using an AWS SDK

The following code example shows how to create a new CloudWatch Logs log stream.

.NET

AWS SDK for .NET

```
        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.");
            }
        }
```

- Find instructions and more code on GitHub.
- For API details, see CreateLogStream 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Create a CloudWatch Logs subscription filter using an AWS SDK

The following code examples show how to create an Amazon CloudWatch Logs subscription filter.

C++

SDK for C++

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;
        }
```

- Find instructions and more code on GitHub.
- For API details, see PutSubscriptionFilter in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
    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);
        }
    }
```

- Find instructions and more code on GitHub.
- For API details, see PutSubscriptionFilter in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch Logs service client object.
export const cwlClient = new CloudWatchLogsClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {
  PutSubscriptionFilterCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { cwlClient } from "./libs/cloudWatchLogsClient.js";

// Set the parameters
export const params = {
  destinationArn: "LAMBDA_FUNCTION_ARN", //LAMBDA_FUNCTION_ARN
  filterName: "FILTER_NAME", //FILTER_NAME
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP", //LOG_GROUP
};

export const run = async () => {
  try {
    const data = await cwlClient.send(new PutSubscriptionFilterCommand(params));
    console.log("Success", data.subscriptionFilters);
    return data; //For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on GitHub.
- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see PutSubscriptionFilter in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// 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);
  }
});
```

- Find instructions and more code on GitHub.
- 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Delete a CloudWatch Logs log group using an AWS SDK

The following code example shows how to delete an existing CloudWatch Logs log group.

.NET

AWS SDK for .NET

```
        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}.");
            }
        }
```

- Find instructions and more code on GitHub.
- For API details, see DeleteLogGroup 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Delete a CloudWatch Logs subscription filter using an AWS SDK

The following code examples show how to delete an Amazon CloudWatch Logs subscription filter.

C++

SDK for C++

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;
        }
```

- Find instructions and more code on GitHub.
- For API details, see DeleteSubscriptionFilter in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
    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);
        }
    }
```

- Find instructions and more code on GitHub.
- For API details, see DeleteSubscriptionFilter in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch Logs service client object.
export const cwlClient = new CloudWatchLogsClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {
  DeleteSubscriptionFilterCommand
} from "@aws-sdk/client-cloudwatch-logs";
import { cwlClient } from "./libs/cloudWatchLogsClient.js";

// Set the parameters
export const params = {
  filterName: "FILTER", //FILTER
  logGroupName: "LOG_GROUP", //LOG_GROUP
};

export const run = async () => {
  try {
    const data = await cwlClient.send(
      new DeleteSubscriptionFilterCommand(params)
    );
    console.log(
      "Success, subscription filter deleted",
      data
    );
    return data; //For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on GitHub.
- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteSubscriptionFilter in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// 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);
  }
});
```

- Find instructions and more code on GitHub.
- 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
This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun deleteSubFilter( filter: String?, logGroup: String?) {

     val request = DeleteSubscriptionFilterRequest {
         filterName = filter
         logGroupName = logGroup
     }

     CloudWatchLogsClient { region = "us-west-2" }.use { logs ->
        logs.deleteSubscriptionFilter(request)
        println( "Successfully deleted CloudWatch logs subscription filter named
 $filter")
    }
}
```

- Find instructions and more code on GitHub.
- 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Describe CloudWatch Logs subscription filters using an AWS SDK

The following code examples show how to describe Amazon CloudWatch Logs existing subscription filters.

C++

SDK for C++

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();
        }
```

- Find instructions and more code on GitHub.
- For API details, see DescribeSubscriptionFilters in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
   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");
    }
```

- Find instructions and more code on GitHub.
- For API details, see DescribeSubscriptionFilters in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch Logs service client object.
export const cwlClient = new CloudWatchLogsClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-
logs";
import { cwlClient } from "./libs/cloudWatchLogsClient.js";

// Set the parameters
export const params = {
  logGroupName: "GROUP_NAME", //GROUP_NAME
  limit: 5
};

export const run = async () => {
  try {
    const data = await cwlClient.send(
      new DescribeSubscriptionFiltersCommand(params)
    );
    console.log("Success", data.subscriptionFilters);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on GitHub.
- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DescribeSubscriptionFilters in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// 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 = {
  logGroupName: 'GROUP_NAME',
  limit: 5
};

cwl.describeSubscriptionFilters(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- Find instructions and more code on GitHub.
- 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
This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun describeFilters(logGroup: String) {

        val request =  DescribeSubscriptionFiltersRequest {
          logGroupName = logGroup
          limit = 1
        }

        CloudWatchLogsClient { 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}" )
          }
        }
 }
```

- Find instructions and more code on GitHub.
- 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Describe CloudWatch Logs export tasks using an AWS SDK

The following code example shows how to describe CloudWatch Logs export tasks.

.NET

AWS SDK for .NET

```
        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);
        }
```

- Find instructions and more code on GitHub.
- 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Describe CloudWatch Logs log groups using an AWS SDK

The following code example shows how to describe CloudWatch Logs log groups.

.NET

AWS SDK for .NET

```
        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();

            var request = new DescribeLogGroupsRequest
            {
                Limit = 5,
            };

            var response = await client.DescribeLogGroupsAsync(request);

            if (response.LogGroups.Count > 0)
            {
                do
                {
                    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");
                    });
                }
                while (response.NextToken is not null);
            }
        }
```

- Find instructions and more code on GitHub.
- For API details, see DescribeLogGroups 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 (p. 33). This topic also includes information about getting started and details about previous SDK versions.

Cross-service examples for CloudWatch Logs using AWS SDKs

The following sample applications use AWS SDKs to combine CloudWatch Logs with other AWS services. Each example includes a link to GitHub, where you can find instructions on how to set up and run the application.

Examples

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.

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
- EventBridge
- Lambda

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch Logs with an AWS SDK (p. 33). 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

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. This content includes 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 user accounts with 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 recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- 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 personal data that is stored in Amazon S3.
- If you require FIPS 140-2 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) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form 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 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 keys.

If you want to manage the keys used for encrypting and decrypting your logs, use customer master keys (CMK) from AWS Key Management Service. For more information, see Encrypt log data in CloudWatch Logs using AWS Key Management Service (p. 66).

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 (p. 165)
- Access control (p. 167)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials* and they provide complete access to all of your AWS resources.

Important
For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the *IAM User Guide*.

- **IAM user** – An IAM user is simply an identity within your AWS account that has specific custom permissions (for example, permissions to view metrics in CloudWatch Logs). You can use an IAM user name and password to sign in to secure AWS webpages such as the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (AWS CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. CloudWatch Logs supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is another IAM identity you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **Applications running on Amazon EC2** – Instead of storing access keys within the EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 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

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

Note
An *account administrator* (or administrator IAM user) is a user with administrator privileges. For more information, see IAM best practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

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 `*` at the end, is what is returned by the `describe-log-groups` CLI command and the **DescribeLogGroups** API. arn:aws:logs:*region*:*account-id*:log-group:*log_group_name*

Resource type	ARN format
	arn:aws:logs:*region*:*account-id*:log-group:*log_group_name*:*
Log stream	arn:aws:logs:*region*:*account-id*:log-group:*log_group_name*:log-stream:*log-stream-name*
Destination	arn:aws:logs:*region*:*account-id*:destination:*destination_name*

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 (p. 171).

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 (p. 176).

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, an IAM 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 an IAM 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 polices) 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 log data sharing with subscriptions (p. 106).

Topics

- Resource-based policies (p. 170)

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.

Identity-based policies (IAM policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to view logs in the CloudWatch Logs console, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. The principal in the trust policy an also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following is an example policy that grants permissions for the `logs:PutLogEvents`, `logs:CreateLogGroup`, and `logs:CreateLogStream` actions on all resources in us-east-1. For log groups, CloudWatch Logs supports identifying specific resources using the resource ARNs (also referred to as resource-level permissions) for some of the API actions. If you want to include all log groups, you must specify the wildcard character (*).

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Action":[
            "logs:PutLogEvents",
            "logs:CreateLogGroup",
            "logs:CreateLogStream"
         ],
         "Resource":"arn:aws:logs:us-east-1:*:*"
      }
   ]
}
```

For more information about using identity-based policies with CloudWatch Logs, see Using identity-based policies (IAM policies) for CloudWatch Logs (p. 171). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-based policies

CloudWatch Logs supports resource-based policies for destinations, which you can use to enable cross account subscriptions. For more information, see Create a destination (p. 107). Destinations can be created using the PutDestination API, and you can add a resource policy to the destination using the PutDestination 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`.

```
{
  "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 (p. 167) and CloudWatch Logs permissions reference (p. 176).

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 (p. 167).
- **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 (p. 176).

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**
> CloudWatch Logs doesn't support IAM policies that prevent users from assigning specified tags to log groups using the `aws:Resource/key-name` or `aws:TagKeys` condition keys. Additionally, you can't control access to the `DescribeLogGroups` action by using the `aws:ResourceTag/key-name` condition key. Other CloudWatch Logs actions do support the use of the `aws:ResourceTag/key-name` condition key to control access. 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 (p. 167).

> **Note**
> CloudWatch Logs doesn't support IAM policies that prevent users from assigning specified tags to log groups using the `aws:Resource/key-name` or `aws:TagKeys` condition keys. Additionally, you can't control access to the `DescribeLogGroups` action by using the `aws:ResourceTag/key-name` condition key. Other CloudWatch Logs actions do support the use of the `aws:ResourceTag/key-name` condition key to control access. For more information about using tags to control access, see Controlling access to Amazon Web Services resources using tags.

This topic covers the following:

- Permissions required to use the CloudWatch console (p. 172)
- AWS managed (predefined) policies for CloudWatch Logs (p. 174)
- Customer managed policy examples (p. 175)

The following is an example of a permissions policy:

```
{
  "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 (p. 176).

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 (p. 174).

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:describeSubscriptionFilters
- logs:filterLogEvents
- logs:getLogEvents
- logs:putMetricFilter
- logs:putQueryDefinition
- logs:putRetentionPolicy
- 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 contents are as follows:

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

CloudWatchLogsReadOnlyAccess

The **CloudWatchLogsReadOnlylAccess** policy grants read-only access to CloudWatch Logs. The contents are as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

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 IAM 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

The following policy allows a user to access all CloudWatch Logs actions.

```
{
  "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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "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.

```
{
   "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.

For more information about tagging log groups, see Tag log groups in Amazon CloudWatch Logs (p. 64).

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`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:*"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/Team": "Green"
                }
            }
        }
    ]
}
```

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 (p. 167) 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	`logs:CancelExportTask` Required to cancel a pending or running export task.
CreateExportTask	`logs:CreateExportTask` Required to export data from a log group to an Amazon S3 bucket.
CreateLogGroup	`logs:CreateLogGroup` Required to create a new log group.
CreateLogStream	`logs:CreateLogStream` Required to create a new log stream in a log group.
DeleteDestination	`logs:DeleteDestination` Required to delete a log destination and disables any subscription filters to it.
DeleteLogGroup	`logs:DeleteLogGroup` Required to delete a log group and any associated archived log events.
DeleteLogStream	`logs:DeleteLogStream` Required to delete a log stream and any associated archived log events.
DeleteMetricFilter	`logs:DeleteMetricFilter` Required to delete a metric filter associated with a log group.
DeleteQueryDefinition	`logs:DeleteQueryDefinition` Required to delete a saved query definition in CloudWatch Logs Insights.
DeleteResourcePolicy	`logs:DeleteResourcePolicy` Required to delete a CloudWatch Logs resource policy.
DeleteRetentionPolicy	`logs:DeleteRetentionPolicy` Required to delete a log group's retention policy.

CloudWatch Logs API operations	Required permissions (API actions)
DeleteSubscriptionFilter	`logs:DeleteSubscriptionFilter` Required to delete the subscription filter associated with a log group.
DescribeDestinations	`logs:DescribeDestinations` Required to view all destinations associated with the account.
DescribeExportTasks	`logs:DescribeExportTasks` Required to view all export tasks associated with the account.
DescribeLogGroups	`logs:DescribeLogGroups` Required to view all log groups associated with the account.
DescribeLogStreams	`logs:DescribeLogStreams` Required to view all log streams associated with a log group.
DescribeMetricFilters	`logs:DescribeMetricFilters` Required to view all metrics associated with a log group.
DescribeQueryDefinitions	`logs:DescribeQueryDefinitions` Required to see the list of saved query definitions in CloudWatch Logs Insights.
DescribeQueries	`logs:DescribeQueries` Required to see the list of CloudWatch Logs Insights queries that are scheduled, executing, or have recently excecuted.
DescribeResourcePolicies	`logs:DescribeResourcePolicies` Required to view a list of CloudWatch Logs resource policies.
DescribeSubscriptionFilters	`logs:DescribeSubscriptionFilters` Required to view all subscription filters associated with a log group.
FilterLogEvents	`logs:FilterLogEvents` Required to sort log events by log group filter pattern.
GetLogEvents	`logs:GetLogEvents` Required to retrieve log events from a log stream.

CloudWatch Logs API operations	Required permissions (API actions)
GetLogGroupFields	`logs:GetLogGroupFields` Required to retrieve the list of fields that are included in the log events in a log group.
GetLogRecord	`logs:GetLogRecord` Required to retrieve the details from a single log event.
GetQueryResults	`logs:GetQueryResults` Required to retrieve the results of CloudWatch Logs Insights queries.
ListTagsLogGroup	`logs:ListTagsLogGroup` Required to list the tags associated with a log group.
PutDestination	`logs:PutDestination` Required to create or update a destination log stream (such as an Kinesis stream).
PutDestinationPolicy	`logs:PutDestinationPolicy` Required to create or update an access policy associated with an existing log destination.
PutLogEvents	`logs:PutLogEvents` Required to upload a batch of log events to a log stream.
PutMetricFilter	`logs:PutMetricFilter` Required to create or update a metric filter and associate it with a log group.
PutQueryDefinition	`logs:PutQueryDefinition` Required to save a query in CloudWatch Logs Insights.
PutResourcePolicy	`logs:PutResourcePolicy` Required to create a CloudWatch Logs resource policy.
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.

CloudWatch Logs API operations	Required permissions (API actions)
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 Kinesis Data Firehose. For more information, see Enabling logging from certain AWS services (p. 128).

The **AWSServiceRoleForLogDelivery** service-linked role trusts the following services to assume the role:

- `CloudWatch Logs`

The role permissions policy allows CloudWatch Logs to complete the following actions on the specified resources:

- Action: `firehose:PutRecord` and `firehose:PutRecordBatch` on all Kinesis Data Firehose streams that have a tag with a `LogDeliveryEnabled` key with a value of `True`. This tag is automatically attached to an Kinesis Data Firehose stream when you create a subscription to deliver the logs to Kinesis Data 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 Kinesis Data 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 Kinesis Data 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 Kinesis Data 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.

Compliance validation for Amazon CloudWatch Logs

Third-party auditors assess the security and compliance of Amazon CloudWatch Logs as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. 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 Amazon CloudWatch Logs is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

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 the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon CloudWatch Logs through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

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.

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 the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)
- AWS GovCloud (US-East)

- AWS GovCloud (US-West)

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**. 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*.

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 user 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 Amazon CloudWatch Logs API calls 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 in CloudWatch Logs. CloudTrail captures API calls made by or on behalf of your AWS account. The calls captured include calls from the CloudWatch console and code calls to the CloudWatch Logs API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for CloudWatch Logs. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. 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, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Topics

CloudWatch Logs information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in CloudWatch Logs, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for CloudWatch Logs, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

CloudWatch Logs supports logging the following actions as events in CloudTrail log files:

- CancelExportTask
- CreateExportTask
- CreateLogGroup
- CreateLogStream
- DeleteDestination

- DeleteLogGroup
- DeleteLogStream
- DeleteMetricFilter
- DeleteRetentionPolicy
- DeleteSubscriptionFilter
- PutDestination
- PutDestinationPolicy
- PutMetricFilter
- PutResourcePolicy
- PutRetentionPolicy
- PutSubscriptionFilter
- StartQuery
- StopQuery
- TestMetricFilter

Only request elements are logged in CloudTrail for these CloudWatch Logs API actions:

- DescribeDestinations
- DescribeExportTasks
- DescribeLogGroups
- DescribeLogStreams
- DescribeMetricFilters
- DescribeQueries
- DescribeResourcePolicies
- DescribeSubscriptionFilters
- GetLogGroupFields
- GetLogRecord

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 AWS Identity and Access Management (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.

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"
}
```

CloudWatch Logs agent reference

Important
This reference is for the older deprecated CloudWatch Logs agent. If you use Instance Metadata Service Version 2 (IMDSv2), you must use the new unified CloudWatch agent. Even if you are not using IMDSv2, we strongly recommend that you use the newer unified CloudWatch agent instead of the older logs agent. For more information about the newer unified agent, see Collecting metrics and logs from Amazon EC2 instance and on-premises servers with the CloudWatch agent.
For information about migrating from the older CloudWatch Logs agent to the unified agent, see Create the CloudWatch agent configuration file wtih 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 for Linux Instances*.

   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 the `CreateLogGroup`, `CreateLogStream`, `DescribeLogStreams`, and `PutLogEvents` operations. If you're using the latest agent, `DescribeLogStreams` is not needed. See the sample IAM policy below.

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

Monitoring with CloudWatch metrics

CloudWatch Logs sends metrics to Amazon CloudWatch every minute.

CloudWatch Logs metrics

The `AWS/Logs` namespace includes the following metrics.

Metric	Description
`CallCount`	The number of specified API operations performed in your account. `CallCount` is a CloudWatch Logs service usage metric. For more information, see CloudWatch Logs service usage metrics (p. 199). 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. Valid Dimensions: LogGroupName, DestinationType, FilterName Valid Statistic: Sum Units: None
`DeliveryThrottling`	The number of log events for which CloudWatch Logs was throttled when forwarding data to the subscription destination. Valid Dimensions: LogGroupName, DestinationType, FilterName Valid Statistic: Sum Units: None
`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 (p. 199). 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

Metric	Description
	Valid Statistic: Sum Units: Bytes
`ForwardedLogEvents`	The number of log events forwarded to the subscription destination. Valid Dimensions: LogGroupName, DestinationType, FilterName Valid Statistic: Sum Units: None
`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
`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 (p. 199). Valid Dimensions: Class, Resource, Service, Type Valid Statistic: Sum Units: None

Dimensions for CloudWatch Logs metrics

The dimensions that you can use with CloudWatch Logs metrics are listed below.

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 Kinesis Data Firehose.

Dimension	Description
`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 (?).

CloudWatch Logs service usage metrics

CloudWatch Logs sends metrics to CloudWatch that track the usage 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 (p. 201).

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.

Dimension	Description
	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`.
`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

The following tables provide the default service quotas, also referred to as limits, for CloudWatch Logs for an AWS account. Most of these service quotas, but not all, are listed under the Amazon CloudWatch Logs namespace in the Service Quotas console. To request a quota increase for those quotas, see the procedure later in this section.

Resource	Default quota
Batch size	1 MB (maximum). This quota can't be changed.
Data archiving	Up to 5 GB of data archiving for free. This quota can't be changed.
CreateLogGroup	5 transactions per second (TPS/account/Region), after which transactions are throttled. You can request a quota increase.
CreateLogStream	50 transactions per second (TPS/account/Region), after which transactions are throttled. You can request a quota increase.
DeleteLogGroup	5 transactions per second (TPS/account/Region), after which transactions are throttled. You can request a quota increase.
DescribeLogGroups	5 transactions per second (TPS/account/Region). You can request a quota increase.
DescribeLogStreams	5 transactions per second (TPS/account/Region). You can request a quota increase.
Discovered log fields	CloudWatch Logs Insights can discover a maximum of 1000 log event fields in a log group. This quota can't be changed. For more information, see Supported logs and discovered fields (p. 35).
Extracted log fields in JSON logs	CloudWatch Logs Insights can extract a maximum of 200 log event fields from a JSON log. This quota can't be changed. For more information, see Supported logs and discovered fields (p. 35).
Event size	256 KB (maximum). This quota can't be changed.
Export task	One active (running or pending) export task at a time, per account. This quota can't be changed.
FilterLogEvents	25 requests per second in US East (N. Virginia). 10 requests per second in the following Regions: • US East (Ohio) • US West (N. California) • US West (Oregon) • Africa (Cape Town) • Asia Pacific (Hong Kong)

Resource	Default quota
	• Asia Pacific (Mumbai) • Asia Pacific (Seoul) • Asia Pacific (Singapore) • Asia Pacific (Tokyo) • Asia Pacific (Sydney) • Canada (Central) • Europe (Ireland) • Europe (London) • Europe (Milan) • Europe (Paris) • Europe (Stockholm) • Middle East (Bahrain) • South America (São Paulo) • AWS GovCloud (US-East) • AWS GovCloud (US-West) 5 requests per second in all other Regions. This quota can't be changed.
GetLogEvents	30 requests per second in Europe (Paris). 25 requests per second in the following Regions: • US East (N. Virginia) • US East (Ohio) • US West (N. California) • Africa (Cape Town) • Asia Pacific (Hong Kong) • Asia Pacific (Mumbai) • Asia Pacific (Seoul) • Asia Pacific (Singapore) • Asia Pacific (Tokyo) • Asia Pacific (Sydney) • Canada (Central) • Europe (London) • Europe (Milan) • Europe (Stockholm) • Middle East (Bahrain) • South America (São Paulo) • AWS GovCloud (US-East) • AWS GovCloud (US-West) 10 requests per second in all other Regions. This quota can't be changed.

Resource	Default quota
GetLogEvents	25 requests per second per account per Region in most Regions. The exceptions are 30 requests per second in Europe (Paris), and 10 requests per second in US West (Oregon), Europe (Ireland), Europe (Frankfurt), and Asia Pacific (Osaka). This quota can't be changed. We recommend subscriptions if you are continuously processing new data. If you need historical data, we recommend exporting your data to Amazon S3.
Incoming data	Up to 5 GB of incoming data for free. This quota can't be changed.
Log groups	1,000,000 log groups per account per Region. You can request a quota increase. There is no quota on the number of log streams that can belong to one log group.
Metrics filters	100 per log group. This quota can't be changed.
Embedded metric format metrics	100 metrics per log event and 9 dimensions per metric. For more information about the embedded metric format, see Specification: Embedded Metric Format in the Amazon CloudWatch User Guide.
PutLogEvents	5 requests per second per log stream. Additional requests are throttled. This quota can't be changed. The maximum batch size of a PutLogEvents request is 1MB. 800 transactions per second per account per Region, except for the following Regions where the quota is 1500 transactions per second per account per Region: US East (N. Virginia), US West (Oregon), and Europe (Ireland). You can request a quota increase.
Query execution timeout	Queries in CloudWatch Logs Insights time out after 15 minutes. This time limit can't be changed.
Queried log groups	A maximum of 20 log groups can be queried in a single CloudWatch Logs Insights query. This quota can't be changed.
Query concurrency	A maximum of 10 concurrent CloudWatch Logs Insights queries, including queries that have been added to dashboards. This quota is not managed through Service Quotas. You can request a quota increase by creating a support case.
Query results availability	Results from a query are retrievable for 7 days. This availability time can't be changed.

Resource	Default quota
Query results displayed in console	By default, up to 1000 rows of query results are displayed on the console. You can use the **`limit`** command in a query to increase this to as many as 10,000 rows. For more information, see CloudWatch Logs Insights query syntax (p. 40).
Resource policies	Up to 10 CloudWatch Logs resource policies per Region per account. This quota can't be changed.
Saved queries	You can save as many as 1000 CloudWatch Logs Insights queries, per Region per account. This quota can't be changed.
Subscription filters	2 per log group. This quota can't be changed.

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.

update-history-change	update-history-description	update-history-date
CloudWatch Logs Insights released (p. 206)	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 *Amazon CloudWatch Logs User Guide*	November 27, 2018
Support for Amazon VPC endpoints (p. 206)	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 *Amazon CloudWatch Logs User Guide*.	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 (p. 183).	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? (p. 1) 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 (p. 64).	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 (p. 63).	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 (p. 61).	August 29, 2016

Change	Description	Release date
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 Amazon CloudWatch Logs API calls in AWS CloudTrail (p. 186).	March 10, 2016
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 (p. 136).	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? (p. 1).	July 10, 2014

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.