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. You can then retrieve the associated log data from CloudWatch Logs.

Features

- **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 periods 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 AWS Services

The following services are used in conjunction with CloudWatch Logs:

- **AWS CloudTrail** is a web service that enables you to monitor the calls made to the CloudWatch Logs API for your account, including calls made by the AWS Management Console, command line interface (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. 114).
- **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. In addition, AWS may delete empty log streams that are over 2 months old.

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 (AWS)

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://aws.amazon.com/, and then choose **Create an AWS Account**.

   Note
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code using 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 both a new unified CloudWatch agent, and an older CloudWatch Logs agent. We recommend the unified CloudWatch agent. The new unified agent has the following advantages.

- You can collect both logs and advanced metrics with the installation and configuration of just one agent.
- The unified agent enables the collection of logs from servers running Windows Server.
- If you are using the agent to collect CloudWatch metrics, the unified agent also enables the collection of additional system metrics, for in-guest visibility.
- The unified agent provides better performance.

The older CloudWatch Logs agent is still supported. If you are already using that agent, you may continue to do so. If you would like to migrate from the CloudWatch Logs agent to the new unified CloudWatch agent, we also provide a migration path.

Contents

Differences between the two agents

The two agents provide similar functionality for CloudWatch Logs, with the unified CloudWatch agent adding the ability to collect logs from servers running Windows Server. The other difference is the symbols supported for log timestamp format, as shown in the following table.

Symbols Supported by Both Agents	Symbols Supported Only by Unified CloudWatch Agent	Symbols Supported Only by Older CloudWatch Logs Agent
%A, %a, %b, %B, %d, %H, %l, %m, %M, %p, %S, %y, %Y, %Z, %z	%-d, %-l, %-m, %-M, %-S	%c, %f, %j, %U, %W, %w

For more information about the meanings of the symbols supported by the new CloudWatch agent, see CloudWatch Agent Configuration File: Logs Section in the *Amazon CloudWatch User Guide*. For information about symbols supported by the CloudWatch Logs agent, see Agent Configuration File (p. 117).

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
- 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).
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": [
        "arn:aws:logs:*:*:*"
    ]
  }
 ]
}
```

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. 117).
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. 49).

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. 117).

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. 49).

Quick Start: Install and Configure the CloudWatch Logs Agent on an EC2 Linux Instance at Launch

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).
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. 117).

The following is a sample agent configuration file for Amazon Linux

```
[general]
state_file = /var/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://myawsbucket/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. 49).

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. Youconfigure 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. In the navigation pane, choose **Systems Manager Services**, **Run Command**.
3. Choose **Run a command**.
4. For **Command document**, choose **AWS-ConfigureCloudWatch**.
5. 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*.
6. For **Status**, choose **Enabled**.
7. For **Properties**, copy and paste the JSON content you created in the previous tasks.
8. 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. 117).

```
[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. 49).

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*.

Analyze 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 quickly 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 quickly. 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.

> **Important**
> If your network security team does not allow the use of web sockets, you cannot currently access the CloudWatch Logs Insights 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 all types of logs. For every log sent to CloudWatch Logs, three system fields are automatically generated:

- `@message` contains the raw unparsed log event.
- `@timestamp` contains the time at which the log event was added to the CloudWatch Logs.
- `@logStream` contains the name of the log stream to which the log event was added.

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 does not 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. 38)

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` 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. 34).
CloudTrail logs Logs in JSON format	For more information, see Fields in JSON Logs (p. 34).
Other log types	`@timestamp`, `@logStream`, `@message`.

Fields in JSON Logs

CloudWatch Logs Insights represents nested JSON fields using the dot notation. In the following example JSON event, the field `type` in the JSON object `userIdentity` is represented as `userIdentity.type`.

JSON arrays are flattened into a list of field names and values. For example, to specify the value of `instanceId` for the first item in `requestParameters.instancesSet`, use `requestParameters.instancesSet.items.0.instanceId`.

```
{ "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"
                }
            }
        ]
    }
}
```


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 re-run 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

Start by running 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 **Insights**.

   Near the top of the screen is the query editor. When you first open CloudWatch Logs Insights, this box contains a default query that returns the 20 most recent log events.
3. Select a log group to query, above the query editor.

When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in **Discovered fields** in the right pane. It also displays a bar graph of log events in this log group over time. This bar graph shows the distribution of events in the log group that matches your query and time range, not just the events displayed in the table

4. Choose **Run query**.

   The results of the query appear. In this example, the results are the most recent 20 log events of any type.
5. To see all fields of one of the returned log events, choose the arrow to the left of that log event.

Modify the Sample Query

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

If you have not 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 you write.`

To modify the CloudWatch Logs Insights sample query

- In the query editor, change **20** to **50**. Choose **Run query**.

  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.

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 have not 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 fields contained in a particular log event, choose the arrow to the left of that row. The **Discovered fields** area shows the most common fields that CloudWatch Logs has detected in the log events received by this log group in the past 15 minutes, and the percentage of those log events in which each field appears. If you do not see **Discovered fields**, choose the left arrow near the top right of the screen to open the right-side panel.

   The **awsRegion** field may appear in your log event, depending on what events are in your logs. For the rest of this tutorial, you use **awsRegion** as the filter field, but you can use a different field if that field is not available.
2. In the query editor box, place your cursor after **50** and press Enter.
3. On the new line, first type **|** (the pipe character) and a space. Commands in a CloudWatch Logs Insights query must be separated by the pipe character.
4. Type **filter awsRegion="us-east-1"**.
5. Choose **Run query**.

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 may need to escape the field name. If the field name includes non-alphanumeric characters, you must put back-tick characters (`) before and after the field name. For example, **`error-code`="102"**.

Using the ` characters is necessary for field names containing non-alphanumeric characters, but not for values. Values are always contained in double quote 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. 38).

Tutorial: Run a Query with an Aggregation Function

In this tutorial, you run a query that returns the results of executing aggregate functions on log records.

To run an aggregation query

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**.
3. Choose a log group.
4. In the query editor, delete the query that is currently shown, then type the following and choose **Run query**. Replace *fieldname* with the name of a field that appears in the **Discovered fields** area on the right of the screen.

```
stats count(*) by fieldname
```

The results show the number of log events in the log group that were received by CloudWatch Logs that contain each different value for the field name you chose.

Tutorial: Run a Query That Produces a 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 or stacked area graph. This helps you quickly 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 **Insights**.
3. Choose a log group.
4. In the query editor, delete the current contents, then type the following and 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 stacked area chart, choose **Stacked area** at the upper right of the graph.

CloudWatch Logs Insights Query Syntax

CloudWatch Logs Insights supports a query language you can use to perform queries on your log groups. Each query can include one or more query commands separated by Unix-style pipe characters (`|`).

Six query commands are supported, along with many supporting functions and operations, including regular expressions, arithmetic operations, comparison operations, numeric functions, datetime functions, string functions, and generic functions.

CloudWatch Logs Insights Query Commands

The following table lists the six supported query commands along with basic examples. For more powerful sample queries, see Sample Queries (p. 45).

Command	Description	Examples
`fields`	Retrieves the specified fields from log events. You can use functions and operations within a fields command.	`fields \`foo-bar\`, action, abs(f3-f4)` retrieves the fields `foo-bar`, `action`, and the absolute value of the difference between `f3` and `f4` for all log events in the log group. The backtick symbol `` ` `` is necessary around `foo-bar` because that field name includes a non-alphanumeric character. Any log field named in a query that has characters other than the @ sign, the period (.), and alphanumeric characters must be surrounded by backtick characters.
`filter`	Filters the results of a query based on one or more conditions. You can use comparison operators (=, !=, <, <=, >, >=), Boolean operators (`and`, `or`, and `not`) and regular expressions.	`fields f1, f2, f3 | filter (duration>2000)` retrieves the fields `f1`, `f2`, and `f3` for all log events with a value over 2000 in the `duration` field. `filter (duration>2000)` is also a valid query, but the results do not show separate fields. Instead, the results show the @timestamp and all log data in the @message field for all log events where duration is more than 2000. `fields f1, f2 | filter (f1=10 or f3>25)` retrieves the fields `f1` and `f2` for all log events where `f1` is 10 or `f3` is greater than 25.

Command	Description	Examples
		`fields f1 | filter statusCode like /2\d\d/` returns log events where the field `statusCode` has a value between 200 and 299.
`stats`	Calculates aggregate statistics based on the values of log fields. Several statistical operators are supported, including `sum()`, `avg()`, `count()`, `,min()`, and `max()`. When you use **`stats`**, you can also use `by` to specify one or more criteria to use to group data when calculating the statistics.	`stats avg (f1) by f2` calculates the average value of `f1` for each unique value of `f2`.
`sort`	Sorts the retrieved log events. Both ascending (`asc`) and descending (`desc`) order are supported.	`fields f1, f2, f3 | sort f1 desc` retrieves the fields `f1`, `f2`, and `f3` and sorts the returned events in descending order based on the value of `f1`.
`limit`	Limits the number of log events returned by the query.	`fields f1, f2 | sort @timestamp desc | limit 25` retrieves the fields `f1` and `f2`, sorts the events in descending order based on the value of `@timestamp`, and returns 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. 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 you write.`
`parse`	Extracts data from a log field, creating one or more ephemeral fields that you can process further in the query.	`parse @message "user=*, method:*, latency := *" as @user, @method, @latency | stats avg(@latency) by @method, @user` extracts the ephemeral fields `@user`, `@method`, and `@latency` from the log field `@message` and returns the average latency for each unique combination of `@method` and `@user`

Regular Expressions in the Filter Command

You can use `like` or `=~` (equal sign followed by a tilde) in the **`filter`** query command to filter by substrings or regular expressions. Enclose your match string with double quotes or single quotes to perform substring matching. To perform regular expression matching, enclose it with forward slashes. The query returns only log events that match the criteria you set.

Examples

The following three examples return all events in which `f1` contains the word `Exception`. The first two examples use regular expressions, and the third example uses a substring match. Each of these three examples is case-sensitive.

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

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

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

The following example uses a regular expression and returns all events in which `f1` contains the word `Exception`. The query is not case-sensitive.

```
fields f1, f2, f3 | filter f1 like /(?i)Exception/
```

The following example uses a regular expression and returns all events in which `f1` is exactly the word `Exception`. The query is not case-sensitive.

```
fields f1, f2, f3 | filter f1 =~ /^(?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
```

Return 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.

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.

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

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	Description
`abs(a)`	Absolute value
`ceil(a)`	Round to ceiling (the smallest integer that is greater than the value of `a`).
`floor(a)`	Round to floor (the largest integer that is smaller than the value of `a`).
`greatest(a,b,... z)`	Returns the largest value.
`least(a, b, ... z)`	Returns the smallest value.
`log(a)`	Natural log
`sqrt(a)`	Square root

General Functions

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

Function	Arguments	Result type	Description
`ispresent(fieldname)`	Log field	Boolean	Returns `true` if the field exists.
`coalesce(fieldname1, fieldname2, ... fieldnamex)`	Log fields	Log field	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	Arguments	Result type	Description
`isempty(fieldname)`	String	Boolean	Returns `true` if the field is missing or is an empty string.
`isblank(fieldname)`	String	Boolean	Returns `true` if the field is missing, an empty string, or contains only white space.
`concat(string1, string2, ... stringz)`	Strings	String	Concatenates the strings.
`ltrim(string) or ltrim(string1, string2)`	String	String	Remove white space from the left of the string. If the function has a second string argument, it removes the characters of `string2` from the left of `string1`. For example, `ltrim("xyZfooxyZ","xyZ")` returns `"fooxyZ"`.
`rtrim(string) or rtrim(string1, string2)`	String	String	Remove white space from the right of the string. If the function has a second string argument, it removes the characters of `string2` from the right of `string1`. For example, `rtrim("xyZfooxyZ","xyZ")` returns `"xyZfoo"`.
`trim(string) or trim(string1, string2)`	String	String	Remove white space from both ends of the string. If the function has a second string argument, it removes the characters of `string2` from both sides of `string1`. For example, `trim("xyZfooxyZ","xyZ")` returns `"foo"`.
`strlen(string)`	String	Number	Returns the length of the string in Unicode code points.
`toupper(string)`	String	String	Converts the string to uppercase.
`tolower(string)`	String	String	Converts the string to lowercase.
`substr(string1, x), or substr(string1, x, y)`	String, Number or String,	String	Returns a substring from the index specified by the number argument

Function	Arguments	Result type	Description
	Number, Number		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"`.
`replace(string1, string2, string3)`	String, String, String	String	Replaces all instances of `string2` in `string1` with `string3`. For example: `replace("foo","o","0")` returns `"f00"`.
`strcontains(string1, string2)`	String	Number	Returns 1 if `string1` contains `string2` 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.

As part of datetime functions, you can use time periods that consist of a number and then either `m` for minutes or `h` for hours. For example, `10m` is 10 minutes and `1h` is one hour.

Function	Arguments	Result type	Description
`datefloor(a, period)`	Timestamp, 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(a, period)`	Timestamp, 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.
`bin(period)`	period	Timestamp	Rounds the value of `@timestamp` to the given period and then truncates.

Aggregation Functions in the Stats Command

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

Function	Arguments	Result type	Description
`avg(NumericFieldname)`	Numeric log field	Number	The average of the values in the specified field.
`count(fieldname) or count(*)`	Log field	Number	Counts the log records. count(*) counts all records in the log group, while count

Function	Arguments	Result type	Description
			(fieldname) counts all records that include the specified field name.
`count_distinct(fieldname)`	Log field	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)`	Log field	Log field value	The maximum of the values for this log field in the queried logs.
`min(fieldname)`	Log field	Log field value	The minimum of the values for this log field in the queried logs.
`pct(fieldname, value)`	Log field value, value	Log field value	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(NumericFieldname)`	Numeric log field	Number	The standard deviation of the values in the specified field.
`sum(NumericFieldname)`	Numeric log field	Number	The sum of the values in the specified field.

Visualizing Time Series Data

You can use visualizations to identify trends and patterns that occur over time within your logs. CloudWatch Logs Insights generates visualizations for all queries with the following characteristics:

- The query contains one or more aggregation functions. For more information, see Aggregation Functions in the Stats Command (p. 43).
- The query uses the `bin()` function to group the data by one field.

CloudWatch Logs Insights displays visualizations using line charts and stacked area charts.

Visualization Examples

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)
```

Common Mistakes

The following query does not generate a visualization, because it contains more than one grouping field.

```
stats avg(myfield1) by bin(5m), myfield4
```

The following query does not generate a visualization, because the `by bin()` grouping function is not used.

```
stats avg(myfield1) by myfield4
```

Sample Queries

This section includes example queries that show the power of CloudWatch Logs Insights.

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 are not exceptions:

```
fields @message | filter @message not like /Exception/
```

Filter Command Examples

To use

Queries for Lambda Logs

Determine the amount of overprovisioned memory:

```
filter @type = "REPORT"
| stats max(@memorySize / 1024 / 1024) as provisonedMemoryMB,
    min(@maxMemoryUsed / 1024 / 1024) as smallestMemoryRequestMB,
    avg(@maxMemoryUsed / 1024 / 1024) as avgMemoryUsedMB,
    max(@maxMemoryUsed / 1024 / 1024) 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 20 byte transfers for a given host:

```
filter srcAddr= "192.0.2.0/24"
| stats sum(bytes) as bytesTransferred by dstAddr
| sort bytesTransferred  desc
| limit 15
```

Find the IP addresses using 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 Region:

```
stats count(*) by eventSource, eventName, awsRegion
```

Find the Amazon EC2 hosts that were started or stopped in a given Region:

```
filter (eventName="StartInstances" or eventName="StopInstances") and region="us-east-2"
```

Find the 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
```

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 automatically re-run every time you load the dashboard and every time that the dashboard refreshes. These queries count toward your limit of four 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 **Insights**.
3. Choose a log group and run a query.
4. Choose **Add to dashboard**.
5. Select the dashboard, or choose **Create new** to create a new dashboard for the query results.
6. Choose **Add to dashboard**.

To copy query results to the clipboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**.
3. Choose a log group and run a query.
4. Choose **Actions**, **Copy query results**.

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

To view the queries that are currently running

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**.
3. Choose **Actions**, **View running queries for this account**.
4. To stop one of the running queries, choose **Cancel**.

   If this query has been added to a dashboard, choosing **Cancel** just stops the current execution. If you open the dashboard or the dashboard refreshes, the query is automatically re-run. To permanently remove it from the dashboard, open the dashboard and remove the widget.

To view your recent query history

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**.
3. Choose **Actions**, **View query history for this account**.

   A list of your recent queries appears. You can re-run any of them by choosing **Run query**.

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 into 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 **Logs**.
3. Choose **Actions**, **Create log group**.
4. Type a name for the log group, and choose **Create log group**.

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 **Logs**.
3. For **Log Groups**, choose the log group to view the streams.
4. For **Log Streams**, choose the log stream name to view the log data.
5. To change how the log data is displayed, do one of the following:
   - To expand all log events, above the list of log events, choose **Expand all**.
   - To expand all log events and view them as plain text, above the list of log events, choose **Text**.
   - To filter the log events, type the desired search filter in the search field. For more information, see Searching and Filtering Log Data (p. 55).

- To view log data for a specified date and time range, above the list of log events, choose **custom**. You can choose **Absolute** to specify a date and time range or **Relative** to choose a predefined number of minutes, hours, days, or weeks. You can also switch between **UTC** and **Local timezone**.

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 automatically deleted. 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 **Logs**.
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 the **Edit Retention** dialog box, 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.

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 and Cost Management 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 KMS

You can encrypt the log data in CloudWatch Logs using an AWS Key Management Service (AWS KMS) customer master key (CMK). Encryption is enabled at the log group level, by associating a CMK with a log group, either when you create the log group or after it exists.

After you associate a CMK with a log group, all newly ingested data for the log group is encrypted using the CMK. 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 CMK whenever encrypted data is requested.

After you disassociate a CMK from a log group, CloudWatch Logs stops encrypting newly ingested data for the log group. All previously ingested data remains encrypted.

Limits

- To associate a CMK with a log group and perform the following steps, you must have the following permissions: `kms:CreateKey`, `kms:GetKeyPolicy`, and `kms:PutKeyPolicy`.
- After you associate or disassociate a CMK 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 CMK or delete an associated CMK, your encrypted data in CloudWatch Logs can no longer be retrieved.
- You cannot associate a CMK with a log group using the CloudWatch console.

Step 1: Create an AWS KMS CMK

To create an AWS KMS CMK, use the following create-key command:

```
aws kms create-key
```

The output contains the key ID and Amazon Resource Name (ARN) of the CMK. The following is example output:

```
{
    "KeyMetadata": {
        "KeyId": "6f815f63-e628-448c-8251-e40cb0d29f59",
        "Description": "",
        "Enabled": true,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1478910250.94,
        "Arn": "arn:aws:kms:us-west-2:123456789012:key/6f815f63-e628-448c-8251-
e40cb0d29f59",
        "AWSAccountId": "123456789012"
    }
}
```

Step 2: Set Permissions on the CMK

By default, all AWS KMS CMKs are private; only the resource owner can use it to encrypt and decrypt data. However, the resource owner can grant permissions to access the CMK to other users and resources. With this step, you give the CloudWatch service principal permission to use the CMK. This service principal must be in the same region as where the CMK is stored.

First, save the default policy for your CMK 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 statement in bold, replacing *`region`* with the region to use for your log group and separating the existing statement from the new statement with a comma.

```
{
  "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": "*"
    }
```

```
  ]
}
```

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 CMK

You can associate a CMK with a log group when you create it or afterwards.

To associate the CMK 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 CMK 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 CMK associated with a log group, use the following disassociate-kms-key command:

```
aws logs disassociate-kms-key --log-group-name my-log-group
```

Searching and Filtering Log Data

After the CloudWatch Logs agent begins publishing log data to Amazon CloudWatch, you can begin searching and filtering the log data 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.

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:

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.

default value

The value reported to the metric filter during a period when no matching logs are found. By setting this to 0, you ensure that data is reported during every period, preventing "spotty" metrics with periods of no data.

Filter and Pattern Syntax

You can use metric filters to search for and match terms, phrases, or values in your log events. When a metric filter finds one of the terms, phrases, or values in your log events, you can increment the value of

a CloudWatch metric. For example, you can create a metric filter to search for and count the occurrence of the word *ERROR* in your log events.

Metric filters can also extract numerical values from space-delimited log events, such as the latency of web requests. In these examples, you can increment your metric value by the actual numerical value extracted from the log.

You can also use conditional operators and wildcards to create exact matches. Before you create a metric filter, you can test your search patterns in the CloudWatch console. The following sections explain the metric filter syntax in more detail.

Matching Terms in Log Events

To search for a term in your log events, use the term as your metric filter pattern. You can specify multiple terms in a metric filter pattern, but all terms must appear in a log event for there to be a match. Metric filters are case sensitive.

Metric filter terms that include characters other than alphanumeric or underscore must be placed inside double quotes ("").

To exclude a term, use a minus sign (-) before the term.

Example 1: Match everything

The filter pattern "" matches all log events.

Example 2: Single term

The filter pattern "ERROR" matches log event messages that contain this term, such as the following:

- [ERROR] A fatal exception has occurred
- Exiting with ERRORCODE: -1

Example 3: Include a term and exclude a term

In the previous example, if you change the filter pattern to "ERROR" - "Exiting", the log event message "Exiting with ERRORCODE: -1" would be excluded.

Example 4: Multiple terms

The filter pattern "ERROR Exception" matches log event messages that contain both terms, such as the following:

- [ERROR] Caught IllegalArgumentException
- [ERROR] Unhandled Exception

The filter pattern "Failed to process the request" matches log event messages that contain all terms, such as the following:

- [WARN] Failed to process the request
- [ERROR] Unable to continue: Failed to process the request

OR Pattern Matching

You can match terms in text-based filters using OR pattern matching. Use a question mark for OR, such as `?term`.

Look at the three log event examples below. `ERROR` matches examples 1 and 2. `?ERROR ?WARN` matches examples 1, 2, and 3, as all of them include either the word ERROR or the word WARN. `ERROR WARN` only matches example 1, as it is the only one containing both of those words. ERROR -WARN matches example 2, as it matches a string that contains ERROR but does not contain WARN.

1. `ERROR WARN message`
2. `ERROR message`
3. `WARN message`

You can match terms using OR pattern matching in space-delimited filters. With space-delimited filters, `w1` means the first word in the log event, `w2` means the second word, and so on. For the example patterns below, [w1=ERROR, w2] matches pattern 2 because ERROR is the first word, and [w1=ERROR || w1=WARN, w2] matches patterns 2 and 3. [w1!=ERROR&&w1!=WARN, w2] matches lines containing both ERROR and WARN (pattern 1).

1. ERROR WARN message
2. ERROR message
3. WARN message

You can match terms using OR pattern matching in JSON filters. For the example patterns below, {$.foo = bar} matches pattern 1, {$.foo = baz } matches pattern 2, and {$.foo = bar || $.foo = baz } matches pattern 1 and 2.

1. {"foo": "bar"}
2. {"foo": "baz"}

Matching Terms in JSON Log Events

You can extract values from JSON log events. To extract values from JSON log events, you need to create a string-based metric filter. Strings containing scientific notation are not supported. The items in the JSON log event data must exactly match the metric filter. You might want to create metric filters in JSON log events to indicate the following:

- A certain event occurs. For example eventName is "UpdateTrail".
- The IP is outside a known subnet. For example, sourceIPAddress is not in some known subnet range.
- A combination of two or more other conditions are true. For example, the eventName is "UpdateTrail" and the recipientAccountId is 123456789012.

Using Metric Filters to Extract Values from JSON Log Events

You can use metric filters to extract values from JSON log events. A metric filter checks incoming logs and modifies a numeric value when the filter finds a match in the log data. When you create a metric filter, you can simply increment a count each time the matching text is found in a log, or you can extract numerical values from the log and use those to increment the metric value.

Matching JSON Terms Using Metric Filters

The metric filter syntax for JSON log events uses the following format:

```
{ SELECTOR EQUALITY_OPERATOR STRING }
```

The metric filter must be enclosed in curly braces { }, to indicate this is a JSON expression. The metric filter contains the following parts:

SELECTOR

Specifies what JSON property to check. Property selectors always start with dollar sign ($), which signifies the root of the JSON. Property selectors are alphanumeric strings that also support '-' and '_' characters. Array elements are denoted with [NUMBER] syntax, and must follow a property. Examples are: $.eventId, $.users[0], $.users[0].id, $.requestParameters.instanceId.

EQUALITY_OPERATOR

Can be either = or !=.

STRING

A string with or without quotes. You can use the asterisk '*' wildcard character to match any text at, before, or after a search term. For example, ***Event** will match **PutEvent** and **GetEvent**. **Event*** will match **EventId** and **EventName**. **Ev*ent** will only match the actual string **Ev*ent**. Strings that consist entirely of alphanumeric characters do not need to be quoted. Strings that have unicode and other characters such as '@,' '$,' '\,' etc. must be enclosed in double quotes to be valid.

JSON Metric Filter Examples

The following is a JSON example:

```
{
  "eventType": "UpdateTrail",
  "sourceIPAddress": "111.111.111.111",
  "arrayKey": [
        "value",
        "another value"
  ],
  "objectList": [
       {
         "name": "a",
         "id": 1
       },
       {
         "name": "b",
         "id": 2
       }
  ],
  "SomeObject": null,
  "ThisFlag": true
}
```

The following filters would match:

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

Filter on the event type being UpdateTrail.

```
{ $.sourceIPAddress != 123.123.* }
```

Filter on the IP address being outside the subnet 123.123 prefix.

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

Filter on the first entry in arrayKey being "value". If arrayKey is not an array this will be false.

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

Filter on the second entry in objectList having a property called id = 2. If objectList is not an array this will be false. If the items in objectList are not objects or do not have an id property, this will be false.

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

Filter on SomeObject being set to null. This will only be true is the specified object is set to null.

```
{ $.SomeOtherObject NOT EXISTS }
```

Filter on SomeOtherObject being non-existent. This will only be true if specified object does not exist in log data.

```
{ $.ThisFlag IS TRUE }
```

Filters on ThisFlag being TRUE. This also works for boolean filters which check for FALSE value.

JSON Compound Conditions

You can combine multiple conditions into a compound expression using OR (||) and AND (&&). Parenthesis are allowed and the syntax follows standard order of operations () > && > ||.

```
{
    "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]
    ]
}
```

Examples

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

Matches the JSON above.

```
{ ($.user.id = 2 && $.users[0].email = "nonmatch") || $.actions[2] = "GET" }
```

Doesn't match the JSON above.

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

Matches the JSON above.

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

Doesn't match the JSON above.

JSON Special Considerations

The SELECTOR must point to a value node (string or number) in the JSON. If it points to an array or object, the filter will not be applied because the log format doesn't match the filter. For example, both {$.users = 1} and {$.users != 1} will fail to match a log event where users is an array:

```
{
  "users": [1, 2, 3]
}
```

Numeric Comparisons

The metric filter syntax supports precise matching on numeric comparisons. The following numeric comparisons are supported: <, >, >=, <=, =, !=

Numeric filters have a syntax of

```
{ SELECTOR NUMERIC_OPERATOR NUMBER }
```

The metric filter must be enclosed in curly braces { }, to indicate this is a JSON expression. The metric filter contains the following parts:

SELECTOR

Specifies what JSON property to check. Property selectors always start with dollar sign ($), which signifies the root of the JSON. Property selectors are alphanumeric strings that also support '-' and '_' characters. Array elements are denoted with [NUMBER] syntax, and must follow a property. Examples are: $.latency, $.numbers[0], $.errorCode, $.processes[4].averageRuntime.

NUMERIC_OPERATOR

Can be one of the following: =, !=, <, >, <=, or >=.

NUMBER

An integer with an optional + or - sign, a decimal with an optional + or - sign, or a number in scientific notation, which is an integer or a decimal with an optional + or - sign, followed by 'e', followed by an integer with an optional + or - sign.

Examples:

```
{ $.latency >= 500 }
{ $.numbers[0] < 10e3 }
{ $.numbers[0] < 10e-3 }
{ $.processes[4].averageRuntime <= 55.5 }
{ $.errorCode = 400 }
{ $.errorCode != 500 }
```

```
{ $.latency > +1000 }
```

Using Metric Filters to Extract Values from Space-Delimited Log Events

You can use metric filters to extract values from space-delimited log events. The characters between a pair of square brackets [] or two double quotes ("") are treated as a single field. For example:

```
127.0.0.1 - frank [10/Oct/2000:13:25:15 -0700] "GET /apache_pb.gif HTTP/1.0" 200 1534
127.0.0.1 - frank [10/Oct/2000:13:35:22 -0700] "GET /apache_pb.gif HTTP/1.0" 500 5324
127.0.0.1 - frank [10/Oct/2000:13:50:35 -0700] "GET /apache_pb.gif HTTP/1.0" 200 4355
```

To specify a metric filter pattern that parses space-delimited events, the metric filter pattern has to specify the fields with a name, separated by commas, with the entire pattern enclosed in square brackets. For example: [ip, user, username, timestamp, request, status_code, bytes].

In cases where you don't know the number of fields, you can use shorthand notification using an ellipsis (...). For example:

```
[..., status_code, bytes]
[ip, user, ..., status_code, bytes]
[ip, user, ...]
```

You can also add conditions to your fields so that only log events that match all conditions would match the filters. For example:

```
[ip, user, username, timestamp, request, status_code, bytes > 1000]
[ip, user, username, timestamp, request, status_code = 200, bytes]
[ip, user, username, timestamp, request, status_code = 4*, bytes]
[ip, user, username, timestamp, request = *html*, status_code = 4*, bytes]
```

You can use `&&` as an AND operator, as in the following examples:

```
[ip, user, username, timestamp, request, status_code = 4* && bytes > 1000]
[ip, user, username, timestamp, request, status_code = 4* && status_code != 403, bytes]
```

CloudWatch Logs supports both string and numeric conditional fields. For string fields, you can use = or != operators with an asterisk (*).

For numeric fields, you can use the >, <, >=, <=, =, and != operators.

If you are using a space-delimited filter, extracted fields map to the names of the space-delimited fields (as expressed in the filter) to the value of each of these fields. If you are not using a space-delimited filter, this will be empty.

Example Filter:

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

Example log event for the filter:

```
127.0.0.1 - frank [10/Oct/2000:13:25:15 -0700] \"GET /index.html HTTP/1.0\" 404 1534
```

Extracted fields for the log event and filter pattern:

```
{
   "$status_code": "404",
   "$request": "GET /products/index.html HTTP/1.0",
   "$7": "1534",
   "$4": "10/Oct/2000:13:25:15 -0700",
   "$3": "frank",
   "$2": "-",
   "$1": "127.0.0.1"
}
```

Setting How the Metric Value Changes When Matches Are Found

When a metric filter finds one of the matching terms, phrases, or values in your log events, it increments the count in the CloudWatch metric by the amount you specify for Metric Value. The metric value is aggregated and reported every minute.

If logs are ingested during a one-minute time period but no matches are found, the value specified for Default Value (if any) is reported. However, if no log events are ingested during a one-minute period, then no value is reported.

Specifying a Default Value, even if that value is 0, helps ensure that data is reported more often, helping prevent spotty metrics when matches are not found.

For example, suppose there is a log group that publishes two records every minute and the Metric Value is 1 and the Default Value is 0. If matches are found in the both log records in the first minute, the metric value for that minute is 2. If there are no matches in the log records published in the second minute, the Default Value of 0 is used for both log records and the metric value for that minute is 0.

If you don't specify a Default Value, then no data is reported for any periods where no pattern matches are found.

Publishing Numerical Values Found in Log Entries

Instead of just counting the number of matching items found in logs, you can also use the metric filter to publish values based on numerical values found in the logs. The following procedure shows how to publish a metric with the latency found in the JSON request `metricFilter: { $.latency = * } metricValue: $.latency`.

To publish a metric with the latency in a JSON request

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**.
3. In the contents pane, select a log group, and then choose **Create Metric Filter**.
4. On the **Define Logs Metric Filter** screen, for **Filter Pattern**, type **`{ $.latency = * }`**, and then choose **Assign Metric**.
5. On the **Create Metric Filter and Assign a Metric** screen, choose **Show advanced metric settings**.
6. For **Metric Name**, type **myMetric**.
7. For **Metric Value**, enter **`$.latency`**.
8. For **Default Value** type 0, and then choose **Create Filter**. 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.

The following log event would publish a value of 50 to the metric **myMetric** following filter creation.

```
{
"latency": 50,
"requestType": "GET"
}
```

Creating Metric Filters

The following examples show how to create metric filters.

Examples

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 **Logs**.
3. In the contents pane, select a log group, and then choose **Create Metric Filter**.
4. On the **Define Logs Metric Filter** screen, leave **Filter Pattern** blank.
5. Choose **Assign Metric**, and then on the **Create Metric Filter and Assign a Metric** screen, for **Filter Name**, type **`EventCount`**.
6. Under **Metric Details**, for **Metric Namespace**, type **`MyNameSpace`**.
7. For **Metric Name**, type **`MyAppEventCount`**.
8. Choose **Show advanced metric settings** and confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every log event.
9. For **Default Value** type 0, and then choose **Create Filter**. 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.

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.

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**.
3. In the contents pane, select a log group, and then choose **Create Metric Filter**.
4. On the **Define Logs Metric Filter** screen, for **Filter Pattern**, type **`Error`**.

   Note
   All entries in **Filter Pattern** are case-sensitive.
5. To test your filter pattern, for **Select Log Data to Test**, select the log group to test the metric filter against, and then choose **Test Pattern**.
6. Under **Results**, CloudWatch Logs displays a message showing how many occurrences of the filter pattern were found in the log file.

   To see detailed results, choose **Show test results**.
7. Choose **Assign Metric**, and then on the **Create Metric Filter and Assign a Metric** screen, for **Filter Name**, type **`MyAppErrorCount`**.
8. Under **Metric Details**, for **Metric Namespace**, type **MyNameSpace**.
9. For **Metric Name**, type **ErrorCount**.
10. Choose **Show advanced metric settings** and 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 **Create 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=EventCount,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 **Logs**.
3. In the contents pane, select a log group, and then choose **Create Metric Filter**.
4. On the **Define Logs Metric Filter** screen, for **Filter Pattern**, type **`[IP, UserInfo, User, Timestamp, RequestInfo, StatusCode=404, Bytes]`**.
5. To test your filter pattern, for **Select Log Data to Test**, select the log group to test the metric filter against, and then choose **Test Pattern**.

6. Under **Results**, CloudWatch Logs displays a message showing how many occurrences of the filter pattern were found in the log file.

   To see detailed results, choose **Show test results**.
7. Choose **Assign Metric**, and then on the **Create Metric Filter and Assign a Metric** screen, for **Filter Name**, type **HTTP404Errors**.
8. Under **Metric Details**, for **Metric Namespace**, type **`MyNameSpace`**.
9. For **Metric Name**, type **`ApacheNotFoundErrorCount`**.
10. Choose **Show advanced metric settings** and confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every 404 Error event.
11. For **Default Value** type 0, and then choose **Create 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. 65) 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 **Logs**.
3. In the contents pane, select a log group, and then choose **Create Metric Filter**.
4. On the **Define Logs Metric Filter** screen, for **Filter Pattern**, type **`[ip, id, user, timestamp, request, status_code=4*, size]`**.
5. To test your filter pattern, for **Select Log Data to Test**, select the log group to test the metric filter against, and then choose **Test Pattern**.
6. Under **Results**, CloudWatch Logs displays a message showing how many occurrences of the filter pattern were found in the log file.

   To see detailed results, click **Show test results**.
7. Choose **Assign Metric**, and then on the **Create Metric Filter and Assign a Metric** screen, for **Filter Name**, type **`HTTP4xxErrors`**.
8. Under **Metric Details**, for **Metric Namespace**, type **`MyNameSpace`**.
9. For **Metric Name**, type **HTTP4xxErrors**.
10. Choose **Show advanced metric settings** and confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every log containing a 4xx error.
11. For **Default Value** type 0, and then choose **Create 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

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.

To create a metric filter using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**.
3. In the contents pane, select a log group, and then choose **Create Metric Filter**.
4. On the **Define Logs Metric Filter** screen, for **Filter Pattern**, type **`[ip, id, user, timestamp, request, status_code, size]`**.
5. To test your filter pattern, for **Select Log Data to Test**, select the log group to test the metric filter against, and then choose **Test Pattern**.
6. Under **Results**, CloudWatch Logs displays a message showing how many occurrences of the filter pattern were found in the log file.

   To see detailed results, click **Show test results**.
7. Choose **Assign Metric**, and then on the **Create Metric Filter and Assign a Metric** screen, for **Filter Name**, type **size**.
8. Under **Metric Details**, for **Metric Namespace**, type **`MyNameSpace`**.
9. For **Metric Name**, type **`BytesTransferred`**
10. Choose **Show advanced metric settings** and for **Metric Value** type **`$size`**.
11. For **Default Value** type 0, and then choose **Create 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 BytesTransferred \
--filter-pattern '[ip, id, user, timestamp, request, status_code=4*, size]' \
--metric-transformations \
metricName=BytesTransferred,metricNamespace=MyNamespace,metricValue=$size,defaultValue=0
```

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 **Logs**.
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 **Logs**.
3. In the contents pane, in the **Metric Filter** column, choose the metric filter.
4. On the **Logs Metric Filters** screen, in the metric filter, choose **Delete Filter**.
5. When prompted for confirmation, choose **Yes, 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
```

Search Log Data Using Filter Patterns

You can search your log data using the Filter and Pattern Syntax (p. 55). 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 **Logs**.
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. For **Filter**, type the metric filter syntax to use and then press Enter.

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 **Logs**.
3. For **Log Groups**, choose the name of the log group containing the log stream to search.
4. Choose **Search Events**.
5. For **Filter**, type the metric filter syntax to use, select the date and time range, and then press Enter.

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 group.

```
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.

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, Amazon Kinesis Data Firehose stream, or AWS Lambda for custom processing, analysis, or loading to other systems. To begin subscribing to log events, create the receiving source, 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.

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 would be delivered to the chosen Kinesis stream if the filter pattern matches with the 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. 55).

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 Kinesis stream. 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.

Examples

Example 1: Subscription Filters with Kinesis

The following example associates a subscription filter with a log group containing AWS CloudTrail events to have every logged activity made by "Root" AWS credentials delivered to an 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.json`). Use a text editor to create this policy. Do not use the IAM console to create it.

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.region.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

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.json
```

```
{
    "Role": {
        "AssumeRolePolicyDocument": {
            "Statement": {
                "Action": "sts:AssumeRole",
                "Effect": "Allow",
                "Principal": {
                    "Service": "logs.region.amazonaws.com"
                }
            }
        },
        "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.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"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
    }
  ]
}
```

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.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" \
    --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 an 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 an 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",
    "logStream": "111111111111_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 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 = new Buffer(input.awslogs.data, 'base64');
    zlib.gunzip(payload, function(e, result) {
        if (e) {
```

```
            context.fail(e);
        } else {
            result = JSON.parse(result.toString('ascii'));
            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 file://file-path/helloWorld.zip \
    --role lambda-execution-role-arn \
    --handler helloWorld.handler \
    --runtime nodejs4.3
```

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, replacing *account-id* with your AWS account ID:

   ```
   {
     "Statement": {
       "Effect": "Allow",
       "Principal": { "Service": "firehose.amazonaws.com" },
       "Action": "sts:AssumeRole",
       "Condition": { "StringEquals": { "sts:ExternalId":"account-id" } }
     }
   }
   ```

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`:

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.region.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

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"
                }
            }
        },
        "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:*"]
      },
      {
        "Effect":"Allow",
        "Action":["iam:PassRole"],
        "Resource":["arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"]
      }
    ]
}
```

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 stream (this is known as cross-account data sharing). For example, this log event data can be read from a centralized Amazon Kinesis 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.

Kinesis streams are currently the only resource supported as a destination for cross-account subscriptions.

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 an Kinesis stream and lets CloudWatch Logs know that the recipient wants to receive log data. The recipient then shares the information about his 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.

Topics

Create a Destination

The steps in this procedure are to be done in the log data recipient account. For this example, the log data recipient account has an AWS account ID of 999999999999, while the log data sender AWS account ID is 111111111111.

This example creates a destination using an Kinesis stream called RecipientStream, and a role that enables CloudWatch Logs to write data to it.

To create a destination

1. 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 it will be passed 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": "340282366920938463463374607743176EXAMPLE",
          "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.

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.region.amazonaws.com" },
    "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 that 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",
                "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, you'll 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"
```

```
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::999999999999:role/CWLtoKinesisRole"
    }
  ]
}
```

6. Associate the permissions policy with the role 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 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 **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 enables the log data sender account (111111111111) to access the destination in the log data recipient account (999999999999). 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:region:999999999999:destination:testDestination"
    }
  ]
}
```

   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 "testDestination" \
```

```
    --access-policy file://~/AccessPolicy.json
```

This access policy allows the root user of 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*.

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 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 created above that encapsulates an Kinesis stream called "RecipientStream". 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 "CloudTrail" \
    --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.

Note
Unlike the subscriptions example Real-time Processing of Log Data with Subscriptions (p. 72), in this example you did not have to provide a role-arn. This is because role-arn is needed for impersonation while writing to an Kinesis stream, which has already been provided by the destination owner while creating destination.

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 an 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 may 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 Base64 encoded and compressed in gzip format. 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",
    "logStream": "111111111111_CloudTrail_us-east-1",
    "subscriptionFilters": [
        "RecipientStream"
    ],
    "messageType": "DATA_MESSAGE",
    "logEvents": [
        {
            "id": "31953106606966983378809025079804211143289615424 29EXAMPLE",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root\"}"
        },
        {
            "id": "31953106606966983378809025079804211143289615424 29EXAMPLE",
            "timestamp": 1432826855000,
            "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root\"}"
        },
        {
            "id": "31953106606966983378809025079804211143289615424 29EXAMPLE",
            "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 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.

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 **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 **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::123456789012:role/CWLtoKinesisRole",
     "DestinationArn": "arn:aws:logs:region:123456789012:destination:testDestination",
     "TargetArn": "arn:aws:kinesis:region:123456789012:stream/RecipientStream",
     "AccessPolicy": "{\"Version\": \"2012-10-17\", \"Statement\":
 [{\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\"AWS\":
 \"111111111111\"}, \"Action\": \"logs:PutSubscriptionFilter\", \"Resource\":
 \"arn:aws:logs:region:123456789012: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:123456789012: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 using **PutSubscriptionFilter**.

Sending Logs Directly to Amazon S3

Some AWS services can publish logs directly to Amazon S3. This way, if your main requirement for logs is storage in Amazon S3, you can easily have the service producing the logs send them directly to Amazon S3 without setting up additional infrastructure.

Logs published 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.

Even when logs are published directly to an S3 bucket, CloudWatch Logs charges apply. For more information, see *Deliver Logs to S3* on Amazon CloudWatch Pricing.

The following logs can be published directly to Amazon S3:

- VPC flow logs. For more information, see Publishing Flow Logs to Amazon S3 in the *Amazon VPC User Guide*.
- AWS Global Accelerator flow logs. For more information, see Publishing Flow Logs to Amazon S3 in the *AWS Global Accelerator Developer Guide*.

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.

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.

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.

Log data can take up to 12 hours to become available for export. For near real-time analysis of log data, see Real-time Processing of Log Data with Subscriptions (p. 72) instead.

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'll 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."

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 does not 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**, type a name for the bucket.
5. For **Region**, select the region where your CloudWatch Logs data resides.
6. Choose **Create**.

Step 2: 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 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 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** dialog box, add the following policy, changing `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**.

```
{
    "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" }
      }
    ]
}
```

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 are appropriate for the users who will access the bucket.

Step 3: 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. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**.
3. On the **Log Groups** screen, select the button next to a log group, and then choose **Actions**, **Export data to Amazon S3**.
4. On the **Export data to Amazon S3** screen, under **Define data to export**, set the time range for the data to export using **From** and **To**.
5. 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**, type the log stream prefix.
6. Under **Choose S3 bucket**, choose the account associated with the Amazon S3 bucket.
7. For **S3 bucket name**, choose an Amazon S3 bucket.
8. Choose **Advanced**, and then for **S3 Bucket prefix**, type the randomly-generated string you specified in the bucket policy.
9. Choose **Export data** to export your log data to Amazon S3.
10. To view the status of the log data that you exported to Amazon S3, choose **Actions**, **View all exports to Amazon S3**.

Export Log Data to Amazon S3 Using the AWS CLI

In the following example, you'll 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”.

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 does not 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: 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 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. Do not use the IAM console.

```
{
    "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/*",
            "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } },
            "Principal": { "Service": "logs.us-west-2.amazonaws.com" }
        }
    ]
}
```

2. Set the policy that you just added as the access policy on your bucket 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 are appropriate for the users who will access the bucket.

Step 3: 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 --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:

```
{
    "task-id": "cda45419-90ea-4db5-9833-aade86253e66"
}
```

Step 4: 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 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",
      "To": 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 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 5: 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 cancel-export-task --task-id "cda45419-90ea-4db5-9833-aade86253e66"
```

Note that you can use the describe-export-tasks command to verify that the task was canceled successfully.

Streaming CloudWatch Logs Data to Amazon Elasticsearch Service

You can configure a CloudWatch Logs log group to stream data it receives to your Amazon Elasticsearch Service (Amazon ES) cluster in near real-time through a CloudWatch Logs subscription. For more information, see Real-time Processing of Log Data with Subscriptions (p. 72).

Note
Streaming large amounts of CloudWatch Logs data to Amazon ES 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 Budgets.

Prerequisites

Before you begin, create an Amazon ES domain. The Amazon ES domain can have either public access or VPC access, but you cannot then modify the type of access after the domain is created. You might want to review your Amazon ES domain settings later, and modify your cluster configuration based on the amount of data your cluster will be processing.

For more information about Amazon ES, see the Amazon Elasticsearch Service Developer Guide.

To create an Amazon ES domain

At a command prompt, use the following create-elasticsearch-domain command:

```
aws es create-elasticsearch-domain --domain-name my-domain
```

Subscribe a Log Group to Amazon ES

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

To subscribe a log group to Amazon ES

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**.
3. Select the log group to subscribe.
4. Choose **Actions**, **Stream to Amazon Elasticsearch Service**.
5. On the **Start Streaming to Amazon Elasticsearch Service** screen, for **Amazon ES cluster**, choose the cluster you created in the previous step, and then choose **Next**.
6. Under **Lambda Function**, for **Lambda IAM Execution Role**, choose the IAM role that Lambda should use when executing calls to Amazon ES, and then choose **Next**.

   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 Amazon ES 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 Amazon ES endpoint in the VPC.

7. On the **Configure Log Format and Filters** screen, for **Log Format**, choose a log format.
8. Under **Subscription Filter Pattern**, type the terms or pattern to find in your log events. This ensures that you send only the data you are interested in to your Amazon ES cluster. For more information, see Searching and Filtering Log Data (p. 55).
9. (Optional) Under **Select Log Data to Test**, select a log stream and then click **Test Pattern** to verify that your search filter is returning the results you expect.
10. Choose **Next**, and then on the **Review & Start Streaming to Amazon Elasticsearch Service** screen, choose **Start Streaming**.

Authentication and Access Control 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

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 like 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	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) and AWS Service Namespaces 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. 105).

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. 109).

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. 83).

Topics

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",
```

```
         "Principal":{
            "AWS":"234567890123"
         },
         "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. 105). 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. 84). 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. 102) and CloudWatch Logs Permissions Reference (p. 109).

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. 102).
- **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. 109).

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 AWS Service Actions and Condition Context Keys and Global and IAM Condition Context Keys in the *IAM User Guide*.

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. 101).

This topic covers the following:

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. 109).

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
- Amazon ES
- 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. 107).

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:deleteRetentionPolicy
- logs:deleteSubscriptionFilter
- logs:describeExportTasks
- logs:describeLogGroups
- logs:describeLogStreams
- logs:describeMetricFilters
- logs:describeSubscriptionFilters
- logs:filterLogEvents
- logs:getLogEvents
- logs:putMetricFilter
- 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 in your account, are specific to CloudWatch Logs:

- **CloudWatchLogsFullAccess** – Grants full access to CloudWatch Logs.
- **CloudWatchLogsReadOnlyAccess** – Grants read-only access to CloudWatch Logs.

Note
You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also 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.

Customer Managed Policy Examples

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

The following policy allows a user read-only access to CloudWatch Logs data.

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

CloudWatch Logs Permissions Reference

When you are setting up Access Control (p. 101) 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, and you specify a wildcard character (*) as the resource value in the policy's `Resource` field.

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.
DeleteRetentionPolicy	`logs:DeleteRetentionPolicy`

CloudWatch Logs API Operations	Required Permissions (API Actions)
	Required to delete a log group's retention policy.
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.
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.
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).

CloudWatch Logs API Operations	Required Permissions (API Actions)
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.
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.
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 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)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- EU (Frankfurt)
- EU (Ireland)
- EU (London)
- EU (Paris)
- South America (São Paulo)

Create a VPC Endpoint for CloudWatch Logs

To start using CloudWatch Logs with your VPC, create an interface VPC endpoint for CloudWatch Logs. The endpoint name will be `com.amazonaws.Region.logs`. For more information, see Creating an Interface Endpoint in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch Logs. CloudWatch Logs calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch Logs, and you already have a CloudWatch Logs subscription filter for Kinesis Data Streams and an interface VPC endpoint for Kinesis Data Streams, calls between CloudWatch Logs and Kinesis Data Streams begin to flow through the interface VPC endpoint.

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.

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
- PutRetentionPolicy
- PutSubscriptionFilter
- TestMetricFilter

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

- DescribeDestinations
- DescribeExportTasks
- DescribeLogGroups
- DescribeLogStreams
- DescribeMetricFilters
- DescribeSubscriptionFilters

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

The CloudWatch Logs agent provides an automated way to send log data to CloudWatch Logs from Amazon EC2 instances. The agent is comprised of 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 FAQs

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`.

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 Usage 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
`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
`ForwardedBytes`	The volume of log events in compressed bytes forwarded to the subscription destination. Valid Dimensions: LogGroupName, DestinationType, FilterName 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
`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

Metric	Description
	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

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.
`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 Limits

CloudWatch Logs has the following limits:

Resource	Default Limit
Batch size	1 MB (maximum). This limit cannot be changed.
Data archiving	Up to 5 GB of data archiving for free. This limit cannot be changed.
DescribeLogStreams	5 transactions per second (TPS/account/Region). If you experience frequent throttling, you can request a limit increase.
Discovered log fields	CloudWatch Logs Insights can discover a maximum of 1000 log event fields in a log group. This limit cannot be changed.
Event size	256 KB (maximum). This limit cannot be changed.
Export task	One active (running or pending) export task at a time, per account. This limit cannot be changed.
FilterLogEvents	5 transactions per second (TPS)/account/Region. This limit can be changed only in special circumstances. If you experience frequent throttling, contact AWS Support. For instructions, see AWS Service Limits.
GetLogEvents	10 requests per second per account per Region. We recommend subscriptions if you are continuously processing new data. If you need historical data, we recommend exporting your data to Amazon S3. This limit can be changed only in special circumstances. If you experience frequent throttling, contact AWS Support. For instructions, see AWS Service Limits.
Incoming data	Up to 5 GB of incoming data for free. This limit cannot be changed.
Log groups	5000 log groups per account per Region. You can request a limit increase. There is no limit on the number of log streams that can belong to one log group.
Metrics filters	100 per log group. This limit cannot be changed.
PutLogEvents	5 requests per second per log stream. Additional requests are throttled. This limit cannot be changed. The maximum batch size of a PutLogEvents request is 1MB. 1500 transactions per second per account per Region, except for the following Regions where the limit is 800

Resource	Default Limit
	transactions per second per account per Region: ap-south-1, ap-northeast-1, ap-northeast-2, ap-southeast-1, ap-southeast-2, eu-central-1, eu-west-2, sa-east-1, us-east-2, and us-west-1. You can request a limit increase.
Query concurrency	A maximum of 4 concurrent CloudWatch Logs Insights queries, including queries that have been added to dashboards. You can request a limit increase.
Query results displayed in console	In CloudWatch Logs Insights query results, a maximum of 1000 log events are displayed on the console. This limit cannot be changed.
Subscription filters	1 per log group. This limit cannot be changed.

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
Support for Amazon VPC endpoints (p. 129)	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
CloudWatch Logs Insights	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 (p. 33).	27 November 2018
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. 112).	7 March 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.	7 September 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. 50).	13 December 2016
Console improvements	You can navigate from metrics graphs to the associated log groups. For more information, see Pivot from Metrics to Logs (p. 71).	7 November 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. 49).	29 August 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. 114).	10 March 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. 92).	7 December 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*.