Amazon CloudWatch Events

User Guide

Amazon CloudWatch Events: User Guide

Table of Contents

What is Amazon CloudWatch Events? ... 1
- Concepts ... 1
- Related AWS Services ... 1
- Limits ... 2

Setting Up ... 4
- Sign Up for Amazon Web Services (AWS) ... 4
- Sign in to the Amazon CloudWatch Console ... 4
- Account Credentials ... 5
- Set Up the Command Line Interface ... 6
- Regional Endpoints ... 6

Getting Started ... 7
- Tutorial: Relay Events to Amazon EC2 Run Command ... 8
- Tutorial: Log EC2 Instance States ... 9
  - Step 1: Create a Lambda Function ... 9
  - Step 2: Create a Rule ... 10
  - Step 3: Test the Rule ... 10
- Tutorial: Log Auto Scaling Group States ... 11
  - Step 1: Create a Lambda Function ... 11
  - Step 2: Create a Rule ... 12
  - Step 3: Test the Rule ... 12
- Tutorial: Log S3 Object Level Operations ... 13
  - Step 1: Create an Event Selector ... 13
  - Step 2: Create a Lambda Function ... 14
  - Step 3: Create a Rule ... 14
  - Step 4: Test the Rule ... 15
- Tutorial: Log AWS API Calls ... 16
  - Prerequisite ... 16
  - Step 1: Create a Lambda Function ... 16
  - Step 2: Create a Rule ... 17
  - Step 3: Test the Rule ... 17
- Tutorial: Schedule EBS Snapshots ... 18
  - Step 1: Create a Rule ... 18
  - Step 2: Test the Rule ... 19
- Tutorial: Schedule Lambda Functions ... 19
  - Step 1: Create a Lambda Function ... 19
  - Step 2: Create a Rule ... 20
  - Step 3: Test the Rule ... 21
- Tutorial: Relay Events to a Stream ... 22
  - Prerequisite ... 22
  - Step 1: Create an Amazon Kinesis Stream ... 22
  - Step 2: Create a Rule ... 22
  - Step 3: Test the Rule ... 23
  - Step 4: Verify that the Event is Relayed ... 23

Schedule Expressions for Rules ... 24
- Cron Expressions ... 24
- Rate Expressions ... 26

Events and Event Patterns ... 28
- Event Patterns ... 29

Adding Events with PutEvents ... 31
- Handling Failures When Using PutEvents ... 31
- Sending Events Using the AWS CLI ... 33
- Calculating PutEvents Event Entry Sizes ... 33

Event Types ... 35
- Amazon EBS Events ... 35

Amazon EC2 Events ... 36
Amazon EC2 System Manager Events ... 37
Amazon EC2 Maintenance Window Events ... 38
Amazon ECS Events ... 40
Amazon EMR Events ... 41
Auto Scaling Events ... 43
AWS API Call Events ... 46
AWS CodeDeploy Events ... 49
AWS Console Sign-in Events ... 50
AWS Health Events ... 50
AWS KMS Events ... 52
Scheduled Events ... 53
Trusted Advisor Events ... 53
Authentication and Access Control ... 56
Authentication ... 56
Access Control ... 57
Overview of Managing Access ... 58
Resources and Operations ... 58
Understanding Resource Ownership ... 59
Managing Access to Resources ... 59
Specifying Policy Elements: Actions, Effects, and Principals ... 60
Specifying Conditions in a Policy ... 61
Using Identity-Based Policies (IAM Policies) ... 61
Permissions Required to Use the CloudWatch Console ... 62
AWS Managed (Predefined) Policies for CloudWatch Events ... 63
Permissions Required for CloudWatch Events to Access Certain Targets ... 64
Customer Managed Policy Examples ... 65
Using Resource-Based Policies ... 68
AWS Lambda Permissions ... 68
Amazon SNS Permissions ... 69
Amazon SQS Permissions ... 70
CloudWatch Events Permissions Reference ... 71
Using Conditions ... 73
Example 1: Limit Access to a Specific Source ... 74
Example 2: Define Multiple Sources That Can Be Used in an Event Pattern Individually ... 76
Example 3: Define a Source and a DetailType That Can Be Used in an Event Pattern ... 77
Example 4: Ensure That the Source Is Defined in the Event Pattern ... 78
Example 5: Define a List of Allowed Sources in an Event Pattern with Multiple Sources ... 79
Example 6: Ensure That AWS CloudTrail Events for API Calls from a Certain PrincipalId Are Consumed ... 80
Example 7: Limiting Access to Targets ... 81
Logging API Calls ... 82
CloudWatch Events Information in CloudTrail ... 82
Understanding Log File Entries ... 83
Troubleshooting ... 85
My rule was triggered but my Lambda function was not invoked ... 85
I have just created/modified a rule but it did not match a test event ... 86
My rule did not self-trigger at the time specified in the ScheduleExpression ... 87
My rule did not trigger at the time that I expected ... 87
My rule matches IAM API calls but my rule was not triggered ... 87
My rule is not working because the IAM role associated with the rule is ignored when the rule is triggered ... 87
I created a rule with an EventPattern that is supposed to match a resource, but I don't see any events that match the rule ... 88
My event's delivery to the target experienced a delay ... 88
My rule was triggered more than once in response two identical events. What guarantee does CloudWatch Events offer for triggering rules or delivering events to the targets? ... 88

My rule is being triggered but I don't see any messages published into my Amazon SNS topic 88
My Amazon SNS topic still has permissions for CloudWatch Events even after I deleted the rule associated with the Amazon SNS topic 90
Which IAM condition keys can I use with CloudWatch Events 90
How can I tell when CloudWatch Events rules are broken 90
Document History 91
AWS Glossary 93

What is Amazon CloudWatch Events?

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources to Amazon EC2 instances, AWS Lambda functions, Amazon Kinesis streams, Amazon ECS tasks, Step Functions state machines, Amazon SNS topics, Amazon SQS queues, or built-in targets. Using simple rules that you can quickly set up, you can match events and route them to one or more target functions or streams. CloudWatch Events becomes aware of operational changes as they occur. CloudWatch Events responds to these operational changes and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information.

Concepts

Before you begin using CloudWatch Events, you should understand the following concepts:

- **Events**—An event indicates a change in your AWS environment. AWS resources can generate events when their state changes. For example, Amazon EC2 generates an event when the state of an EC2 instance changes from pending to running, and Auto Scaling generates events when it launches or terminates instances. AWS CloudTrail publishes events when you make API calls. You can generate custom application-level events and publish them to CloudWatch Events. You can also set up scheduled events that are generated on a periodic basis. For a list of services that generate events, and sample events from each service, see Event Types for CloudWatch Events (p. 35).
- **Targets**—A target processes events. Targets can include Amazon EC2 instances, AWS Lambda functions, Amazon Kinesis streams, Amazon ECS tasks, Step Functions state machines, Amazon SNS topics, Amazon SQS queues, and built-in targets. A target receives events in JSON format.
- **Rules**—A rule matches incoming events and routes them to targets for processing. A single rule can route to multiple targets, all of which are processed in parallel. Rules are not processed in a particular order. This enables different parts of an organization to look for and process the events that are of interest to them. A rule can customize the JSON sent to the target, by passing only certain parts or by overwriting it with a constant.

Related AWS Services

The following services are used in conjunction with CloudWatch Events:

- **AWS CloudTrail** enables you to monitor the calls made to the CloudWatch Events API for your account, including calls made by the AWS Management Console, the AWS CLI and other services. When CloudTrail logging is turned on, CloudWatch Events writes log files to an S3 bucket. Each log file contains one or more records, depending on how many actions are performed to satisfy a request. For more information, see Logging Amazon CloudWatch Events API Calls in AWS CloudTrail (p. 82).
- **AWS CloudFormation** enables you to model and set up your AWS resources. You create a template that describes the AWS resources you want, and AWS CloudFormation takes care of provisioning and configuring those resources for you. You can use CloudWatch Events rules in your AWS CloudFormation templates. For more information, see AWS::Events::Rule in the AWS CloudFormation User Guide.
- **AWS Config** enables you to create rules that check the configuration of your AWS resources. These rules are primarily used to check for compliance with your organization's policies. When an AWS Config rule is triggered, it generates an event which can be captured by CloudWatch Events.
- **AWS Identity and Access Management (IAM)** helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication), what resources they can use, and how they can use them (authorization). For more information, see Authentication and Access Control for Amazon CloudWatch Events (p. 56).
- **Amazon Kinesis Streams** enables 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 the Amazon Kinesis Streams Developer Guide.
- **AWS Lambda** enables you 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. Lambda performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning, automatic scaling, code and security patch deployment, and code monitoring and logging. For more information, see the AWS Lambda Developer Guide.

CloudWatch Events Limits

CloudWatch Events has the following limits:

Resource	**Default Limit**
API requests	Up to 5 requests per second for all CloudWatch Events API operations except PutEvents. PutEvents is limited to 10 requests per second.
Event pattern	2048 characters maximum.
Invocations	20/second (after 20 invocations, the invocations are throttled; that is, they still happen but they are delayed). If the invocation of a target fails due to a problem with the target service, account throttling, etc., new attempts are made for up to 24 hours for a specific invocation.
ListRuleNamesByTarget	Up to 100 results per page for requests.
ListRules	Up to 100 results per page for requests.
ListTargetsByRule	Up to 100 results/page for requests.
PutEvents	10 entries/request and 10 requests/second. Each request can be up to 256 KB in size.
PutTargets	10 entries/request.

Resource	Default Limit
RemoveTargets	10 entries/request.
Rules	50 per region per account. You can request a limit increase. For instructions, see AWS Service Limits. Before requesting a limit increase, examine your rules. You may have multiple rules each matching to very specific events. Consider broadening their scope by using fewer identifiers in your Events and Event Patterns (p. 28). In addition, a rule can invoke several targets each time it matches an event. Consider adding more targets to your rules.
Targets	5/rule.

Setting Up Amazon CloudWatch Events

To use Amazon CloudWatch Events you need an AWS account. Your AWS account allows you to use services (for example, Amazon EC2) to generate events 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) to use a command-line interface.

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

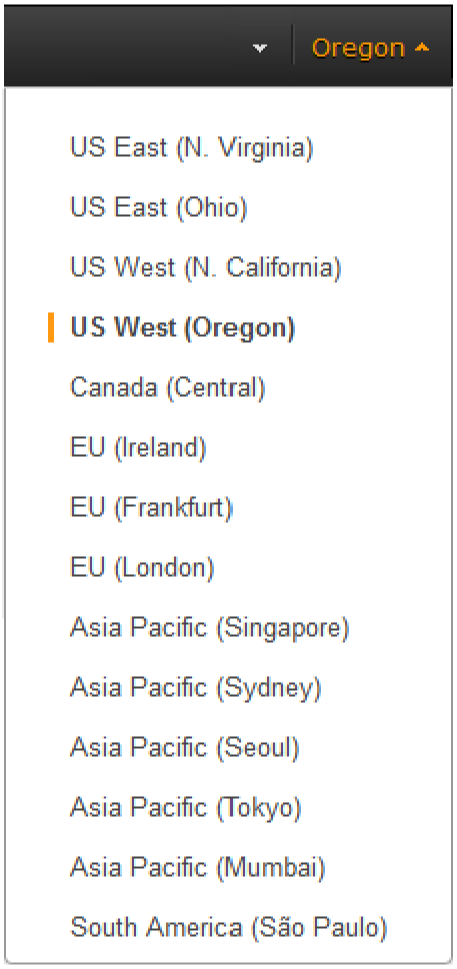
1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN 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 **Events**.

Account Credentials

Although you can use your root account credentials to access CloudWatch Events, we recommend that you use an AWS Identity and Access Management (IAM) account. If you're using an IAM account to access CloudWatch, you must have the following permissions:

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

For more information, see Authentication (p. 56).

Set Up the Command Line Interface

You can use the AWS CLI to perform CloudWatch Events 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*.

Regional Endpoints

You must enable regional endpoints (the default) in order to use CloudWatch Events. For more information, see Activating and Deactivating AWS STS in an AWS Region in the *IAM User Guide*.

Getting Started with Amazon CloudWatch Events

You can set up simple rules that match events and route them to one or more of the following:

- Amazon EC2 instances
- AWS Lambda functions
- Streams in Amazon Kinesis Streams
- Amazon ECS tasks
- AWS Step Functions state machines
- Amazon SNS topics or Amazon SQS queues
- Built-in targets

These tutorials walk you sample usage scenarios for CloudWatch Events.

Limits

- Some target types might not be available in every region. For more information, see Regions and Endpoints in the *Amazon Web Services General Reference*.
- Amazon SQS FIFO (first-in-first-out) queues are not supported.
- Creating rules with built-in targets is supported only in the AWS Management Console.

Tutorials

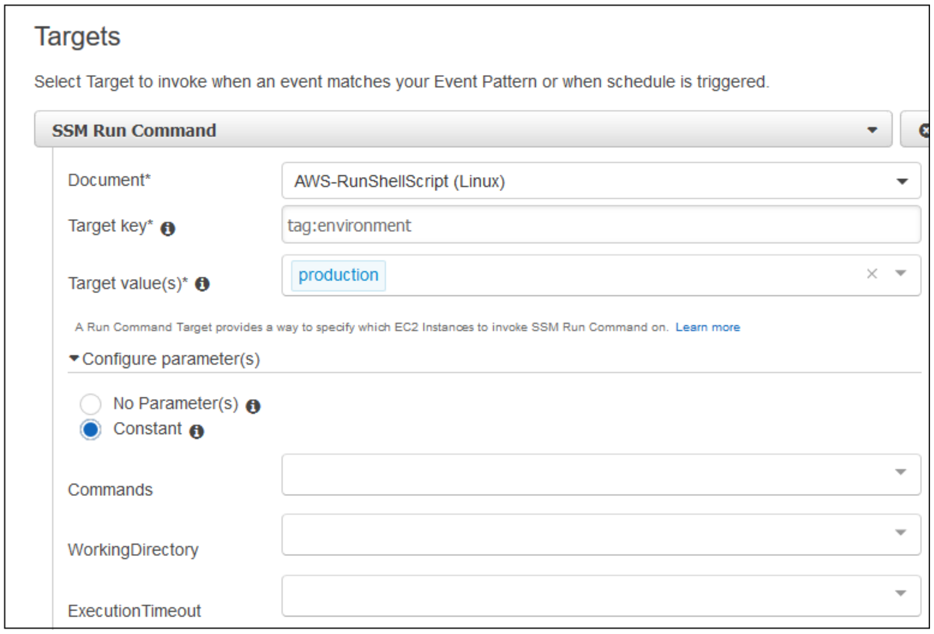
- Tutorial: Use Amazon CloudWatch Events and Amazon EC2 Run Command to Configure Instances Launched in an Auto Scaling Group (p. 8)
- Tutorial: Log the State of an EC2 Instance Using CloudWatch Events (p. 9)
- Tutorial: Log the State of an Auto Scaling Group Using CloudWatch Events (p. 11)
- Tutorial: Log S3 Object Level Operations Using CloudWatch Events (p. 13)
- Tutorial: Log AWS API Calls Using CloudWatch Events (p. 16)
- Tutorial: Schedule EBS Snapshots Using CloudWatch Events (p. 18)
- Tutorial: Schedule Lambda Functions Using CloudWatch Events (p. 19)
- Tutorial: Relay Events to a Stream Using CloudWatch Events (p. 22)

Tutorial: Use Amazon CloudWatch Events and Amazon EC2 Run Command to Configure Instances Launched in an Auto Scaling Group

You can use Amazon CloudWatch Events to invoke Amazon EC2 Systems Manager Run Command and perform actions on Amazon EC2 instances when certain events happen. In this tutorial, set up Run Command to run shell commands and configure each new instance that is launched in an Auto Scaling group. This tutorial assumes that you have already assigned a tag to the Auto Scaling group, with `environment` as the key and `production` as the value.

To create the CloudWatch Events rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**, **Create rule**.
3. For **Event source**, do the following:
   a. Choose **Event Pattern**, **Build event pattern to match events by service**.
   b. For **Service Name**, choose **Auto Scaling**. For **Event Type**, choose **Instance Launch and Terminate**.
   c. Choose **Specific instance event(s)**, **EC2 Instance-launch Lifecycle Action**.
   d. By default, the rule matches any Auto Scaling group in the region. To make the rule match a specific group, choose **Specific group name(s)** and then select one or more groups.
4. For **Targets**, choose **Add Target**, **SSM Run Command.**
5. For **Document**, choose **AWS-RunShellScript (Linux)**. (Note that there are many other **Document** options which cover both Linux and Windows instances.) For **Target key**, type **`tag:environment`**. For **Target value(s)**, type **`production`** and choose **Add**.
6. Under **Configure parameter(s)**, choose **Constant**.
7. For **Commands**, type a shell command and choose **Add**. Repeat this step for all commands to run when an instance launches.
8. If necessary, type the appropriate information in **WorkingDirectory** and **ExecutionTimeout**.
9. CloudWatch Events can create the IAM role needed for your event to run:
   - To create an IAM role automatically, choose **Create a new role for this specific resource.**
   - To use an IAM role that you created before, choose **Use existing role**.



Targets

Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

SSM Run Command

Document*: AWS-RunShellScript (Linux)

Target key*: tag:environment

Target value(s)*: production

A Run Command Target provides a way to specify which EC2 Instances to invoke SSM Run Command on. Learn more

Configure parameter(s)

No Parameter(s) [✓ Constant]

Commands

WorkingDirectory

ExecutionTimeout

10. Choose **Configure details**. For **Rule definition**, type a name and description for the rule.
11. Choose **Create rule**.

Tutorial: Log the State of an EC2 Instance Using CloudWatch Events

You can create a simple AWS Lambda function that logs the changes in state for an Amazon EC2 instance. You can choose to create a rule that runs the function whenever there is a state transition or a transition to one or more states that are of interest. In this tutorial, you log the launch of any new instance.

Step 1: Create a Lambda Function

Create a Lambda function to log the state change events. You'll specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. If you are new to Lambda, you see a welcome page; choose **Get Started Now**; otherwise, choose **Create a Lambda function**.
3. On the **Select blueprint** page, type `hello` for the filter, and then choose the **hello-world** blueprint.
4. On the **Configure triggers** page, choose **Next**.
5. On the **Configure function** page, do the following:

    a. Type a name and description for the Lambda function. (For example, name the function "LogEC2InstanceStateChange").
    b. Edit the sample code for the Lambda function. For example:

```
'use strict';
```

```
exports.handler = (event, context, callback) => {
    console.log('LogEC2InstanceStateChange');
    console.log('Received event:', JSON.stringify(event, null, 2));
    callback(null, 'Finished');
};
```

   c. For **Role**, choose **Choose an existing role** and then choose your basic execution role from **Existing role**. Otherwise, create a new basic execution role.
   d. Choose **Next**.

6. On the **Review** page, choose **Create function**.

Step 2: Create a Rule

Create a rule to run your Lambda function whenever you launch an Amazon EC2 instance.

To create a CloudWatch Events rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event source**, do the following:
   a. Choose **Event Pattern**.
   b. Choose **Build event pattern to match events by service**.
   c. Choose **EC2** and then choose **EC2 Instance State-change Notification**.
   d. Choose **Specific state(s)** and then choose **Running**.
   e. By default, the rule matches any instance in the region. To make the rule match a specific instance, choose **Specific instance(s)** and then choose one or more instances.
5. For **Targets**, choose **Add target** and then choose **Lambda function**.
6. For **Function**, select the Lambda function that you created.
7. Choose **Configure details**.
8. For **Rule definition**, type a name and description for the rule.
9. Choose **Create rule**.

Step 3: Test the Rule

To test your rule, launch an Amazon EC2 instance. After waiting a few minutes for the instance to launch and initialize, you can verify that your Lambda function was invoked.

To test your rule by launching an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Launch an instance. For more information, see Launch Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
4. To view metrics for the event, do the following:
   a. In the navigation pane, choose **Events**, **Rules**.

   b. Choose the name of the rule you created.
   c. Choose **Show metrics for the rule**.
5. To view the output from your Lambda function, do the following:
   a. In the navigation pane, choose **Logs**.
   b. Choose the name of the log group for your Lambda function (/aws/lambda/*function-name*).
   c. Choose the name of log stream to view the data provided by the function for the instance you launched.
6. (Optional) When you are finished, you can open the Amazon EC2 console and stop or terminate the instance you launched. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Tutorial: Log the State of an Auto Scaling Group Using CloudWatch Events

You can run an AWS Lambda function that logs an event whenever an Auto Scaling group launches or terminates an Amazon EC2 instance and whether the launch or terminate event was successful.

For additional CloudWatch Events scenarios using Auto Scaling events, see Getting CloudWatch Events When Your Auto Scaling Group Scales in the *Auto Scaling User Guide*.

Step 1: Create a Lambda Function

Create a Lambda function to log the scale out and scale in events for your Auto Scaling group. You'll specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. If you are new to Lambda, you see a welcome page; choose **Get Started Now**; otherwise, choose **Create a Lambda function**.
3. On the **Select blueprint** page, type `hello` for the filter, and then choose the **hello-world** blueprint.
4. On the **Configure triggers** page, choose **Next**.
5. On the **Configure function** page, do the following:
   a. Type a name and description for the Lambda function. (For example, name the function "LogAutoScalingEvent").
   b. Edit the sample code for the Lambda function. For example:

```
'use strict';

exports.handler = (event, context, callback) => {
    console.log('LogAutoScalingEvent');
    console.log('Received event:', JSON.stringify(event, null, 2));
    callback(null, 'Finished');
};
```

   c. For **Role**, choose **Choose an existing role** and then choose your basic execution role from **Existing role**. Otherwise, create a new basic execution role.
   d. Choose **Next**.
6. Choose **Create function**.

Step 2: Create a Rule

Create a rule to run your Lambda function whenever your Auto Scaling group launches or terminates an instance.

To create a rule

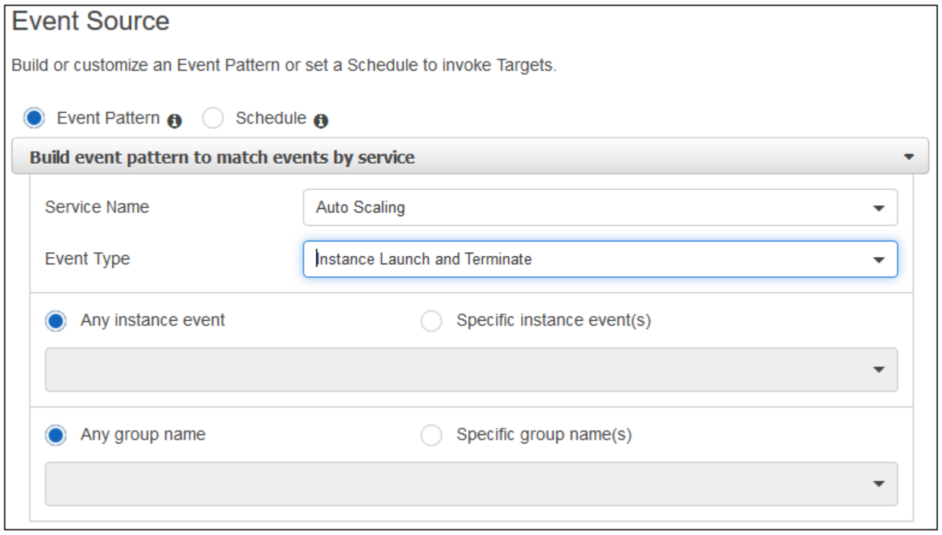
1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event source**, do the following:

   a. Choose **Event Pattern**.

   b. Choose **Build event pattern to match events by service**.

   c. Choose **Auto Scaling** and then choose **Instance Launch and Terminate**.

   d. Choose **Any instance event** to capture all successful and unsuccessful instance launch and terminate events.



Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

[✓ Event Pattern] Schedule

Build event pattern to match events by service

Service Name: Auto Scaling

Event Type: Instance Launch and Terminate

[✓ Any instance event] Specific instance event(s)

[✓ Any group name] Specific group name(s)

5. By default, the rule matches any Auto Scaling group in the region. To make the rule match a specific Auto Scaling group, choose **Specific group name(s)** and then choose one or more Auto Scaling groups.
6. For **Targets**, choose **Add target** and then choose **Lambda function**.
7. For **Function**, select the Lambda function that you created.
8. Choose **Configure details**.
9. For **Rule definition**, type a name and description for the rule. (For example, describe the rule as "Log whenever an Auto Scaling group scales out or in".)
10. Choose **Create rule**.

Step 3: Test the Rule

You can test your rule by manually scaling an Auto Scaling group so that it launches an instance. After waiting a few minutes for the scale out event to occur, you can verify that your Lambda function was invoked.

To test your rule using an Auto Scaling group

1. To increase the size of your Auto Scaling group, do the following:
   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
   b. On the navigation pane, choose **Auto Scaling**, **Auto Scaling Groups**.
   c. Select the checkbox for your Auto Scaling group.
   d. On the **Details** tab, choose **Edit**. For **Desired**, increase the desired capacity by one. For example, if the current value is 2, type 3. The desired capacity must be less than or equal to the maximum size of the group. Therefore, you must update **Max** if your new value for **Desired** is greater than **Max**. When you are finished, choose **Save**.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. To view metrics for the event, do the following:
   a. In the navigation pane, choose **Events**, **Rules**.
   b. Choose the name of the rule you created.
   c. Choose **Show metrics for the rule**.
4. To view the output from your Lambda function, do the following:
   a. In the navigation pane, choose **Logs**.
   b. Choose the name of the log group for your Lambda function (/aws/lambda/*function-name*).
   c. Choose the name of log stream to view the data provided by the function for the instance you launched.
5. (Optional) When you are finished, you can decrease the desired capacity by one so that the Auto Scaling group returns to its previous size.

Tutorial: Log S3 Object Level Operations Using CloudWatch Events

You can log the object level API operations on your Amazon S3 buckets. Before Amazon CloudWatch Events can match these events, you must use AWS CloudTrail to set up a trail configured to receive these events.

Step 1: Create an Event Selector

To log data events for an S3 bucket to CloudTrail and CloudWatch Events, configure an event selector. You can add an event selector to an existing trail, or you can create a trail and then add a selector. For more information, see Data Events in the *AWS CloudTrail User Guide*.

To create a trail

1. Open the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. In the navigation pane, choose **Trails**.
3. (Optional) If you do not have a trail, you can create one.
   a. Choose **Add new trail**.
   b. For **Trail name**, type a name for the trail.
   c. For **S3 bucket**, type the name for the new bucket where CloudTrail will deliver logs.
   d. Choose **Create**.
4. Choose the name of the trail.

5. Choose the pencil icon next to **Event selectors (Optional)**.
6. For **Data events**, select one or more S3 buckets to monitor. To log only the data events for the buckets, choose **No** for **Management events**.
7. Choose **Save**.

Step 2: Create a Lambda Function

Create a Lambda function to log data events for your S3 buckets. You'll specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. If you are new to Lambda, you see a welcome page; choose **Get Started Now**; otherwise, choose **Create a Lambda function**.
3. On the **Select blueprint** page, type `hello` for the filter, and then choose the **hello-world** blueprint.
4. On the **Configure triggers** page, choose **Next**.
5. On the **Configure function** page, do the following:

    a. Type a name and description for the Lambda function. (For example, name the function "LogS3DataEvents".)

    b. Edit the code for the Lambda function. For example:

    ```
    'use strict';

    exports.handler = (event, context, callback) => {
        console.log('LogS3DataEvents');
        console.log('Received event:', JSON.stringify(event, null, 2));
        callback(null, 'Finished');
    };
    ```

    c. For **Role**, choose **Choose an existing role** and then choose your basic execution role from **Existing role**. Otherwise, create a new basic execution role.

    d. Choose **Next**.
6. On the **Review** page, choose **Create function**.

Step 3: Create a Rule

Create a rule to run your Lambda function in response to an S3 data event.

To create a rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event source**, do the following:

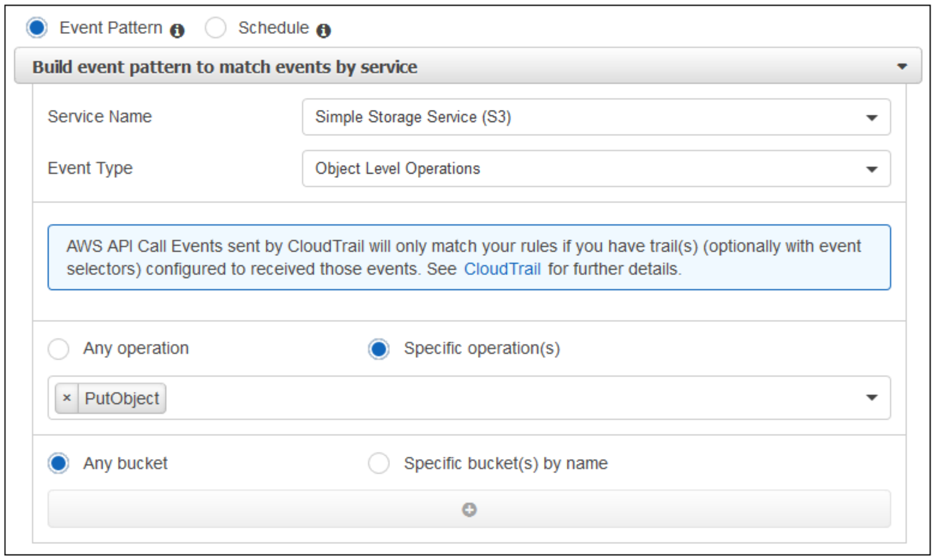
    a. Choose **Event Pattern**.

    b. Choose **Build event pattern to match events by service**.

    c. Choose **Simple Storage Service (S3)** and then choose **Object Level Operations**.

    d. Choose **Specific operation(s)** and then choose **PutObject**.

e. By default, the rule matches data events for all buckets in the region. To match data events for specific buckets, choose **Specify bucket(s) by name** and then specify one or more buckets.



Service Name: Simple Storage Service (S3)
Event Type: Object Level Operations

AWS API Call Events sent by CloudTrail will only match your rules if you have trail(s) (optionally with event selectors) configured to received those events. See CloudTrail for further details.

[ ] Any operation [✓ Specific operation(s)]
PutObject

[✓ Any bucket] [ ] Specific bucket(s) by name

5. For **Targets**, choose **Add target**, and then choose **Lambda function**.
6. For **Function**, select the Lambda function that you created.
7. Choose **Configure details**.
8. For **Rule definition**, type a name and description for the rule.
9. Choose **Create rule**.

Step 4: Test the Rule

To test the rule, put an object in your S3 bucket. You can verify that your Lambda function was invoked.

To view the logs for your Lambda function

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**.
3. Choose the name of the log group for your Lambda function (/aws/lambda/*function-name*).
4. Choose the name of log stream to view the data provided by the function for the instance you launched.

You can also check the contents of your CloudTrail logs in the S3 bucket that you specified for your trail. For more information, see Getting and Viewing Your CloudTrail Log Files in the *AWS CloudTrail User Guide*.

Tutorial: Log AWS API Calls Using CloudWatch Events

You can use a simple AWS Lambda function that logs each AWS API call. For example, you can create a rule to log any operation within Amazon EC2, or you can limit this rule to log only a specific API call. In this tutorial, you log every time an Amazon EC2 instance is stopped.

Prerequisite

Before you can match these events, you must use AWS CloudTrail to set up a trail. If you do not have a trail, complete the following procedure.

To create a trail

1. Open the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. Choose **Add new trail**.
3. For **Trail name**, type a name for the trail.
4. For **S3 bucket**, type the name for the new bucket where CloudTrail will deliver logs.
5. Choose **Create**.

Step 1: Create a Lambda Function

Create a Lambda function to log the API call events. You'll specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. If you are new to Lambda, you see a welcome page; choose **Get Started Now**; otherwise, choose **Create a Lambda function**.
3. On the **Select blueprint** page, type `hello` for the filter, and then choose the **hello-world** blueprint.
4. On the **Configure triggers** page, choose **Next**.
5. On the **Configure function** page, do the following:

    a. Type a name and description for the Lambda function. (For example, name the function "LogEC2StopInstance".)

    b. Edit the sample code for the Lambda function. For example:

    ```
    'use strict';

    exports.handler = (event, context, callback) => {
        console.log('LogEC2StopInstance');
        console.log('Received event:', JSON.stringify(event, null, 2));
        callback(null, 'Finished');
    };
    ```

    c. For **Role**, choose **Choose an existing role** and then choose your basic execution role from **Existing role**. Otherwise, create a new basic execution role.

    d. Choose **Next**.
6. On the **Review** page, choose **Create function**.

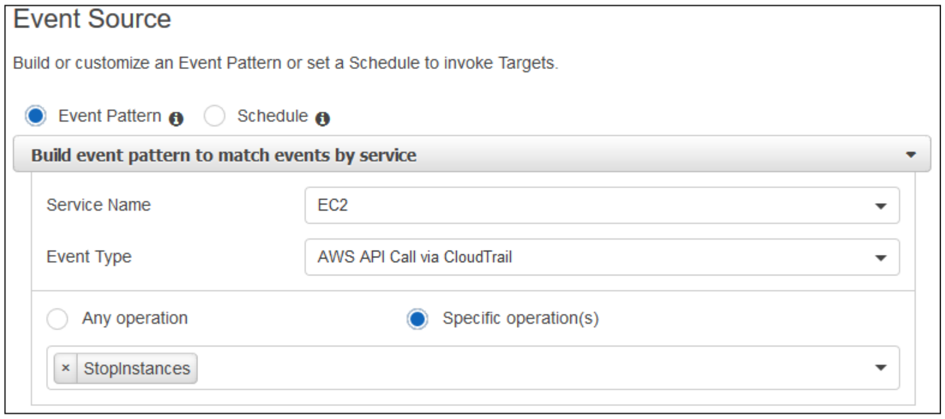
Step 2: Create a Rule

Create a rule to run your Lambda function whenever you stop an Amazon EC2 instance.

To create a rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event source**, do the following:

   a. Choose **Event Pattern**.
   b. Choose **Build event pattern to match events by service**.
   c. Choose **EC2** and then choose **AWS API Call via CloudTrail**.
   d. Choose **Specific operation(s)** and then choose **StopInstances**.


Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

[✓ Event Pattern] Schedule

Build event pattern to match events by service

Service Name: EC2

Event Type: AWS API Call via CloudTrail

Any operation [✓ Specific operation(s)]

StopInstances

5. For **Targets**, choose **Add target** and then choose **Lambda function**.
6. For **Function**, select the Lambda function that you created.
7. Choose **Configure details**.
8. For **Rule definition**, type a name and description for the rule.
9. Choose **Create rule**.

Step 3: Test the Rule

You can test your rule by stopping an Amazon EC2 instance using the Amazon EC2 console. After waiting a few minutes for the instance to stop, check your AWS Lambda metrics in the CloudWatch console to verify that your function was invoked.

To test your rule by stopping an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Launch an instance. For more information, see Launch Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
3. Stop the instance. For more information, see Stop and Start Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
4. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

5. To view metrics for the event, do the following:
   a. In the navigation pane, choose **Events**.
   b. Choose the name of the rule you created.
   c. Choose **Show metrics for the rule**.
6. To view the output from your Lambda function, do the following:
   a. In the navigation pane, choose **Logs**.
   b. Choose the name of the log group for your Lambda function (/aws/lambda/*function-name*).
   c. Choose the name of log stream to view the data provided by the function for the instance you stopped.
7. (Optional) When you are finished, you can terminate the stopped instance. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Tutorial: Schedule EBS Snapshots Using CloudWatch Events

You can run CloudWatch Events rules according to a schedule. In this tutorial, you create a snapshot of an existing Amazon Elastic Block Store (Amazon EBS) volume on a schedule. You can choose a fixed rate to create a snapshot every few minutes or use a Cron expression to specify that the snapshot is made at a specific time of day.

Important
Creating rules with built-in targets is supported only in the AWS Management Console.

Step 1: Create a Rule

Create a rule that takes snapshots on a schedule. You can use a rate expression or a Cron expression to specify the schedule. For more information, see Schedule Expressions for Rules (p. 24).

To create a rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event Source**, do the following:
   a. Choose **Schedule**.
   b. Choose **Fixed rate of** and specify the schedule interval (for example, 5 minutes). Alternatively, choose **Cron expression** and specify a Cron expression (for example, every 15 minutes Monday through Friday, starting at the current time).
5. For **Targets**, choose **Add target** and then choose **Built-in target**.
6. For **Action**, choose **Create a snapshot of an EBS volume**.
7. For **Volume ID**, choose an EBS volume.
8. Choose **Configure details**.
9. For **Rule definition**, type a name and description for the rule.
10. For **AWS permissions**, choose the option to create a new role. This opens the IAM console in a new tab. The new role grants the built-in target permission to access resources on your behalf. Choose **Allow**. The tab with the IAM window closes.

11. Choose **Create rule**.

Step 2: Test the Rule

You can verify your rule by viewing your first snapshot after it is taken.

To test your rule

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Elastic Block Store**, **Snapshots**.
3. Verify that the first snapshot appears in the list.
4. (Optional) When you are finished, you can disable the rule to prevent additional snapshots from being taken.

   a. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
   b. In the navigation pane, choose **Events**, **Rules**.
   c. Select the rule and then choose **Actions**, **Disable**.
   d. When prompted for confirmation, choose **Disable**.

Tutorial: Schedule Lambda Functions Using CloudWatch Events

You can set up a rule to run an AWS Lambda function on a schedule. This tutorial shows how to use the AWS Management Console or the AWS CLI to create the rule. If you would like to use the AWS CLI but have not installed it, see the AWS Command Line Interface User Guide.

CloudWatch Events does not provide second-level precision in schedule expressions. The finest resolution using a Cron expression is a minute. Due to the distributed nature of the CloudWatch Events and the target services, the delay between the time the scheduled rule is triggered and the time the target service honors the execution of the target resource might be several seconds. Your scheduled rule will be triggered within that minute but not on the precise 0th second.

Step 1: Create a Lambda Function

Create a Lambda function to log the scheduled events. You'll specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. If you are new to Lambda, you see a welcome page; choose **Get Started Now**; otherwise, choose **Create a Lambda function**.
3. On the **Select blueprint** page, type `hello` for the filter, and then choose the **hello-world** blueprint.
4. On the **Configure triggers** page, choose **Next**.
5. On the **Configure function** page, do the following:

   a. Type a name and description for the Lambda function. (For example, name the function "LogScheduledEvent".)
   b. Edit the sample code for the Lambda function. For example:

```
'use strict';

exports.handler = (event, context, callback) => {
    console.log('LogScheduledEvent');
    console.log('Received event:', JSON.stringify(event, null, 2));
    callback(null, 'Finished');
};
```

   c. For **Role**, choose **Choose an existing role** and then choose your basic execution role from **Existing role**. Otherwise, create a new basic execution role.
   d. Choose **Next**.
6. On the **Review** page, choose **Create function**.

Step 2: Create a Rule

Create a rule to run your Lambda function on a schedule.

To create a rule using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event Source**, do the following:
   a. Choose **Schedule**.
   b. Choose **Fixed rate of** and specify the schedule interval (for example, 5 minutes).
5. For **Targets**, choose **Add target** and then choose **Lambda function**.
6. For **Function**, select the Lambda function that you created.
7. Choose **Configure details**.
8. For **Rule definition**, type a name and description for the rule.
9. Choose **Create rule**.

If you prefer, you can create the rule using the AWS CLI. First, you must grant the rule permission to invoke your Lambda function. Then you can create the rule and add the Lambda function as a target.

To create a rule using the AWS CLI

1. Use the following put-rule command to create a rule that triggers itself on a schedule:

```
aws events put-rule \
--name my-scheduled-rule \
--schedule-expression 'rate(5 minutes)'
```

   When this rule triggers, it generates an event that serves as input to the targets of this rule. The following is an example event:

```
{
    "version": "0",
    "id": "53dc4d37-cffa-4f76-80c9-8b7d4a4d2eaa",
    "detail-type": "Scheduled Event",
    "source": "aws.events",
    "account": "123456789012",
    "time": "2015-10-08T16:53:06Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:events:us-east-1:123456789012:rule/my-scheduled-rule"
    ],
    "detail": {}
}
```

2. Use the following add-permission command to trust the CloudWatch Events service principal (events.amazonaws.com) and scope permissions to the rule with the specified Amazon Resource Name (ARN):

```
aws lambda add-permission \
--function-name LogScheduledEvent \
--statement-id my-scheduled-event \
--action 'lambda:InvokeFunction' \
--principal events.amazonaws.com \
--source-arn arn:aws:events:us-east-1:123456789012:rule/my-scheduled-rule
```

3. Use the following put-targets command to add the Lambda function you created to this rule so that it runs every 5 minutes:

```
aws events put-targets --rule my-scheduled-rule --targets file://targets.json
```

   Create the file `targets.json` with the following contents:

```
[
  {
    "Id": "1",
    "Arn": "arn:aws:lambda:us-east-1:123456789012:function:LogScheduledEvent"
  }
]
```

Step 3: Test the Rule

You can verify that your Lambda function was invoked.

To test your rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. To view metrics for the event, do the following:
   a. In the navigation pane, choose **Events**, **Rules**.
   b. Choose the name of the rule you created.
   c. Choose **Show metrics for the rule**.
3. To view the output from your Lambda function, do the following:
   a. In the navigation pane, choose **Logs**.
   b. Choose the name of the log group for your Lambda function (/aws/lambda/*function-name*).
   c. Choose the name of log stream to view the data provided by the function for the instance you launched.
4. (Optional) When you are finished, you can disable the rule.
   a. In the navigation pane, choose **Events**, **Rules**.
   b. Select the rule and then choose **Actions**, **Disable**.

c. When prompted for confirmation, choose **Disable**.

Tutorial: Relay Events to a Stream Using CloudWatch Events

You can relay AWS API call events in CloudWatch Events to a stream in Amazon Kinesis.

Prerequisite

Install the AWS CLI. For more information, see the AWS Command Line Interface User Guide.

Step 1: Create an Amazon Kinesis Stream

Use the following create-stream command to create a stream.

```
aws kinesis create-stream --stream-name test --shard-count 1
```

When the stream status is `ACTIVE`, the stream is ready. Use the following describe-stream command to check the stream status:

```
aws kinesis describe-stream --stream-name test
```

Step 2: Create a Rule

As an example, create a rule to send events to your stream when you stop an Amazon EC2 instance.

To create a rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**.
4. For **Event source**, do the following:

   a. Choose **Event Pattern**.
   b. Choose **Build event pattern to match events by service**.
   c. Choose **EC2** and then choose **Instance State-change Notification**.
   d. Choose **Specific state(s)** and then choose **Running**.

5. For **Targets**, choose **Add target**, and then choose **Kinesis stream**.
6. For **Stream**, select the stream that you created.
7. Choose **Configure details**.
8. For **Rule definition**, type a name and description for the rule.
9. For **AWS permissions**, choose the option to create a new role. This opens the IAM console in a new tab. The new role grants CloudWatch Events permission to write records to your streams. Choose **Allow**. The tab with the IAM window closes.
10. Choose **Create rule**.

Step 3: Test the Rule

To test your rule, stop an Amazon EC2 instance. After waiting a few minutes for the instance to stop, check your CloudWatch metrics to verify that your function was invoked.

To test your rule by stopping an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Launch an instance. For more information, see Launch Your Instance in the *Amazon EC2 User Guide for Linux Instances*.
3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
4. To view metrics for the event, do the following:
   a. In the navigation pane, choose **Events**, **Rules**.
   b. Choose the name of the rule you created.
   c. Choose **Show metrics for the rule**.
5. (Optional) When you are finished, you can terminate the instance. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Step 4: Verify that the Event is Relayed

You can get the record from the stream to verify that the event was relayed.

To get the record

1. Use the following get-shard-iterator command to start reading from your Amazon Kinesis stream:

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

   The following is example output:

```
{
    "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd
+efGN2aHFdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

2. Use the following get-records command to get the record. The shard iterator is the one you got in the previous step:

```
aws kinesis get-records --shard-
iterator AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG4lNR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

   If the command is successful, it requests records from your stream for the specified shard. You can receive zero or more records. Any records returned might not represents all records in your stream. If you don't receive the data you expect, keep calling `get-records`.

   Records in Amazon Kinesis are Base64 encoded. However, the streams support in the AWS CLI does not provide Base64 decoding. If you use a Base64 decoder to manually decode the data, you will see that it is the event relayed to the stream in JSON form.

Schedule Expressions for Rules

You can create rules that self-trigger on schedule in CloudWatch Events using Cron or rate expressions. All scheduled events use UTC time zone and the minimum precision for schedules is 1 minute.

CloudWatch Events does not provide second-level precision in schedule expressions. The finest resolution using a Cron expression is a minute. Due to the distributed nature of the CloudWatch Events and the target services, the delay between the time the scheduled rule is triggered and the time the target service honors the execution of the target resource might be several seconds. Your scheduled rule is triggered within that minute, but not on the precise 0th second.

CloudWatch Events supports the following formats for schedule expressions.

Formats

- Cron Expressions (p. 24)
- Rate Expressions (p. 26)

Cron Expressions

Cron expressions have six required fields. Fields are separated by white space.

Syntax

```
cron(fields)
```

Field	Values	Wildcards
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-week	1-7 or SUN-SAT	, - * ? / L
Year	1970-2199	, - * /

Wildcards

- The **,** (comma) wildcard includes additional values. In the Month field, JAN,FEB,MAR would include January, February, and March.
- The **-** (dash) wildcard specifies ranges. In the Day field, 1-15 would include days 1 through 15 of the specified month.
- The ***** (asterisk) wildcard includes all values in the field. In the Hours field, ***** would include every hour.
- The **/** (forward slash) wildcard specifies increments. In the Minutes field, you could enter 1/10 to specify every tenth minute, starting from the first minute of the hour (for example, the 11th, 21st, and 31st minute, and so on).
- The **?** (question mark) wildcard specifies one or another. In the Day-of-month field you could enter **7** and if you didn't care what day of the week the 7th was, you could enter **?** in the Day-of-week field.
- The **L** wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week.
- The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, 3W specifies the day closest to the third weekday of the month.

Limits

- You can't specify the Day-of-month and Day-of-week fields in the same Cron expression. If you specify a value in one of the fields, you must use a **?** (question mark) in the other.
- Cron expressions that lead to rates faster than 1 minute are not supported. Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).

Examples

You can use the following sample cron strings when creating a rule with schedule.

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0	10	*	*	?	*	Run at 10:00 am (UTC) every day
15	12	*	*	?	*	Run at 12:15 pm (UTC) every day
0	18	?	*	MON-FRI	*	Run at 6:00 pm (UTC) every Monday through Friday
0	8	1	*	?	*	Run at 8:00 am (UTC) every 1st day of the month
0/15	*	*	*	?	*	Run every 15 minutes

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

The following examples show how to use Cron expressions with the AWS CLI put-rule command.

```
aws events put-rule --schedule-expression 'cron(0 12 * * ? *)' --name MyRule1
```

```
aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name MyRule2
```

Rate Expressions

A rate expression starts when you create the scheduled event rule, and then runs on its defined schedule.

Rate expressions have two required fields. Fields are separated by white space.

Syntax

```
rate(value unit)
```

value

A positive number.

unit

The unit of time.

Valid values: minute | minutes | hour | hours | day | days

Limits

If the value is equal to 1, then the unit must be singular. Similarly, for values greater than 1, the unit must be plural. For example, rate(1 hours) and rate(5 hour) are not valid, but rate(1 hour) and rate(5 hours) are valid.

Examples

The following examples show how to use rate expressions with the AWS CLI put-rule command.

```
aws events put-rule --schedule-expression 'rate(5 minutes)' --name MyRule3
```

```
aws events put-rule --schedule-expression 'rate(1 hour)' --name MyRule4
```

```
aws events put-rule --schedule-expression 'rate(1 day)' --name MyRule5
```

Events and Event Patterns

Events in Amazon CloudWatch Events are represented as JSON objects. For more information about JSON objects, see RFC 7159. The following is an example event:

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "EC2 Instance State-change Notification",
  "source": "aws.ec2",
  "account": "111122223333",
  "time": "2015-12-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ec2:us-east-1:123456789012:instance/i-12345678"
  ],
  "detail": {
    "instance-id": "i-12345678",
    "state": "terminated"
  }
}
```

It is important to remember the following details about an event:

- They all have the same top-level fields – the ones appearing in the example above – which are never absent.
- The contents of the **detail** top-level field will be different depending on which service generated the event and what the event is.
- The combination of the top-level **source** field and **detail-type** fields serve to identify the fields and values found in the **detail** field.

Each event field is described below.

version

By default, this is set to 0 (zero) in all events.

id

A unique value is generated for every event. This can be helpful in tracing events as they move through rules to targets, and are processed.

detail-type

Identifies, in combination with the **source** field, the fields and values that will appear in the **detail** field.

source

Identifies the service that sourced the event. All events sourced from within AWS will begin with "aws." Customer-generated events can have any value here as long as it doesn't begin with "aws." We recommend the use of java package-name style reverse domain-name strings.

account

The 12-digit number identifying an AWS account.

time

The event timestamp, which can be specified by the service originating the event. If the event spans a time interval, the service might choose to report the start time, so this value can be noticeably before the time the event is actually received.

region

Identifies the AWS region where the event originated.

resources

This JSON array contains ARNs that identify resources that are involved in the event. Inclusion of these ARNs is at the discretion of the service. For example, Amazon EC2 instance state-changes include Amazon EC2 instance ARNs, Auto Scaling events include ARNs for both instances and Auto Scaling groups, but API calls with AWS CloudTrail do not include resource ARNs.

detail

A JSON object, whose content is at the discretion of the service originating the event. The detail content in the example above is very simple, just two fields. AWS API call events have detail objects with around 50 fields nested several levels deep.

Event Patterns

Rules use event patterns to select events and route them to targets. A pattern either matches an event or it doesn't. Event patterns are represented as JSON objects with a structure that is similar to that of events, for example:

```
{
  "source": [ "aws.ec2" ],
  "detail-type": [ "EC2 Instance State-change Notification" ],
  "detail": {
    "state": [ "running" ]
  }
}
```

It is important to remember the following things about event pattern matching:

- For a pattern to match an event, the event must contain all the field names listed in the pattern. The field names must appear in the event with the same nesting structure.
- Other fields of the event not mentioned in the pattern are ignored; effectively, there is a "*": "*" wildcard for fields not mentioned.
- The value of each field in the pattern is an array containing one or more values, and the pattern matches if any of the values in the array match the value in the event.
- If the value in the event is an array, then the pattern matches if the intersection of the pattern array and the event array is non-empty.

- The matching is exact (character-by-character), without case-folding or any other string normalization.
- The values being matched follow JSON rules: Strings enclosed in quotes, numbers, and the unquoted keywords `true`, `false`, and `null`.
- Number matching is at the string representation level. For example, 300, 300.0, and 3.0e2 are not considered equal.

The following event patterns would match the event at the top of this page. The first pattern matches because one of the instance values specified in the pattern matches the event (and the pattern does not specify any additional fields not contained in the event). The second one matches because the "terminated" state is contained in the event.

```
{
  "resources": [
    "arn:aws:ec2:us-east-1:123456789012:instance/i-12345678",
    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcdefgh"
  ]
}
```

```
{
  "detail": {
    "state": [ "terminated" ]
  }
}
```

These event patterns do not match the event at the top of this page. The first pattern does not match because the pattern specifies a "pending" value for state, and this value does not appear in the event. The second pattern does not match because the resource value specified in the pattern does not appear in the event.

```
{
  "source": [ "aws.ec2" ],
  "detail-type": [ "EC2 Instance State-change Notification" ],
  "detail": {
    "state": [ "pending" ]
  }
}
```

```
{
  "source": [ "aws.ec2" ],
  "detail-type": [ "EC2 Instance State-change Notification" ],
  "resources": [ "arn:aws:ec2:us-east-1::image/ami-12345678" ]
}
```

Adding Events with PutEvents

The `PutEvents` action sends multiple events to CloudWatch Events in a single request. Each `PutEvents` request can support a limited number of entries. For more information, see CloudWatch Events Limits (p. 2). The `PutEvents` operation attempts to process all entries in the natural order of the request. Each event has a unique id that is assigned by CloudWatch Events after you call `PutEvents`.

The following example Java code sends two identical events to CloudWatch Events:

```
PutEventsRequestEntry requestEntry = new PutEventsRequestEntry()
        .withTime(new Date())
        .withSource("com.mycompany.myapp")
        .withDetailType("myDetailType")
        .withResources("resource1", "resource2")
        .withDetail("{ \"key1\": \"value1\", \"key2\": \"value2\" }");

PutEventsRequest request = new PutEventsRequest()
        .withEntries(requestEntry, requestEntry);

PutEventsResult result = awsEventsClient.putEvents(request);

for (PutEventsResultEntry resultEntry : result.getEntries()) {
    if (resultEntry.getEventId() != null) {
        System.out.println("Event Id: " + resultEntry.getEventId());
    } else {
        System.out.println("Injection failed with Error Code: " +
 resultEntry.getErrorCode());
    }
}
```

The `PutEvents` result includes an array of response entries. Each entry in the response array directly correlates with an entry in the request array using natural ordering, from the top to the bottom of the request and response. The response `Entries` array always includes the same number of entries as the request array.

Handling Failures When Using PutEvents

By default, failure of individual entries within a request does not stop the processing of subsequent entries in the request. This means that a response Entries array includes both successfully and unsuccessfully processed entries. You must detect unsuccessfully processed entries and include them in a subsequent call.

Successful result entries include Id value, and unsuccessful result entries include `ErrorCode` and `ErrorMessage` values. The `ErrorCode` parameter reflects the type of error. `ErrorMessage` provides more detailed information about the error. The example below has three result entries for a `PutEvents` request. The second entry has failed and is reflected in the response.

Example: PutEvents Response Syntax

```
{
    "FailedEntryCount": 1,
    "Entries": [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {   "ErrorCode": "InternalFailure",
            "ErrorMessage": "Internal Service Failure"
        },
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        }
    ]
}
```

Entries that were unsuccessfully processed can be included in subsequent `PutEvents` requests. First, check the `FailedRecordCount` parameter in the `PutEventsResult` to confirm if there are failed records in the request. If so, each `Entry` that has an `ErrorCode` that is not null should be added to a subsequent request. For an example of this type of handler, refer to the following code.

Example: PutEvents failure handler

```
PutEventsRequestEntry requestEntry = new PutEventsRequestEntry()
        .withTime(new Date())
        .withSource("com.mycompany.myapp")
        .withDetailType("myDetailType")
        .withResources("resource1", "resource2")
        .withDetail("{ \"key1\": \"value1\", \"key2\": \"value2\" }");

List<PutEventsRequestEntry> putEventsRequestEntryList = new ArrayList<>();
for (int i = 0; i < 3; i++) {
    putEventsRequestEntryList.add(requestEntry);
}

PutEventsRequest putEventsRequest = new PutEventsRequest();
putEventsRequest.withEntries(putEventsRequestEntryList);
PutEventsResult putEventsResult = awsEventsClient.putEvents(putEventsRequest);

while (putEventsResult.getFailedEntryCount() > 0) {
    final List<PutEventsRequestEntry> failedEntriesList = new ArrayList<>();
    final List<PutEventsResultEntry> PutEventsResultEntryList =
 putEventsResult.getEntries();
    for (int i = 0; i < PutEventsResultEntryList.size(); i++) {
        final PutEventsRequestEntry putEventsRequestEntry =
 putEventsRequestEntryList.get(i);
        final PutEventsResultEntry putEventsResultEntry = PutEventsResultEntryList.get(i);
        if (putEventsResultEntry.getErrorCode() != null) {
            failedEntriesList.add(putEventsRequestEntry);
        }
    }
    putEventsRequestEntryList = failedEntriesList;
    putEventsRequest.setEntries(putEventsRequestEntryList);
    putEventsResult = awsEventsClient.putEvents(putEventsRequest);
    }
```

Sending Events Using the AWS CLI

You can use the AWS CLI to send custom events. The following example puts one custom event into CloudWatch Events:

```
aws events put-events \
--entries '[{"Time": "2016-01-14T01:02:03Z", "Source": "com.mycompany.myapp", "Resources":
 ["resource1", "resource2"], "DetailType": "myDetailType", "Detail": "{ \"key1\":
 \"value1\", \"key2\": \"value2\" }"}]'
```

You can also create a file for example, **entries.json** like the following:

```
[
  {
    "Time": "2016-01-14T01:02:03Z",
    "Source": "com.mycompany.myapp",
    "Resources": [
      "resource1",
      "resource2"
    ],
    "DetailType": "myDetailType",
    "Detail": "{ \"key1\": \"value1\", \"key2\": \"value2\" }"
  }
]
```

You can use the AWS CLI to read the entries from this file and send events. At a command prompt, type:

```
aws events put-events --entries file://entries.json
```

Calculating PutEvents Event Entry Sizes

You can inject custom events into CloudWatch Events using the `PutEvents` action. You can inject multiple events using the `PutEvents` action as long as the total entry size is less than 256KB. You can calculate the event entry size beforehand by following the steps below. You can then batch multiple even entries into one request for efficiency.

Note
The size limit is imposed on the entry. Even if the entry is less than the size limit, it does not mean that the event in CloudWatch Events will also be less than this size. On the contrary, the event size will always be larger than the entry size due to the necessary characters and keys of the JSON representation of the event. For more information, see Events and Event Patterns (p. 28).

The `PutEventsRequestEntry` size is calculated as follows:

- If the `Time` parameter is specified, it is measured as 14 bytes.
- The `Source` and `DetailType` parameters are measured as the number of bytes for their UTF-8 encoded forms.
- If the `Detail` parameter is specified, it is measured as the number of bytes for its UTF-8 encoded form.
- If the `Resources` parameter is specified, each entry is measured as the number of bytes for their UTF-8 encoded forms.

The following example Java code calculates the size of a given `PutEventsRequestEntry` object:

```
int getSize(PutEventsRequestEntry entry) {
    int size = 0;
    if (entry.getTime() != null) {
        size += 14;
    }
    size += entry.getSource().getBytes(StandardCharsets.UTF_8).length;
    size += entry.getDetailType().getBytes(StandardCharsets.UTF_8).length;
    if (entry.getDetail() != null) {
        size += entry.getDetail().getBytes(StandardCharsets.UTF_8).length;
    }
    if (entry.getResources() != null) {
        for (String resource : entry.getResources()) {
            if (resource != null) {
                size += resource.getBytes(StandardCharsets.UTF_8).length;
            }
        }
    }
    return size;
}
```

Event Types for CloudWatch Events

Amazon CloudWatch Events supports the following events:

Event Types

- Amazon EBS Events (p. 35)
- Amazon EC2 Events (p. 36)
- Amazon EC2 System Manager Events (p. 37)
- Amazon EC2 Maintenance Window Events (p. 38)
- Amazon ECS Events (p. 40)
- Amazon EMR Events (p. 41)
- Auto Scaling Events (p. 43)
- AWS API Call Events (p. 46)
- AWS CodeDeploy Events (p. 49)
- AWS Console Sign-in Events (p. 50)
- AWS Health Events (p. 50)
- AWS KMS Events (p. 52)
- Scheduled Events (p. 53)
- Trusted Advisor Events (p. 53)

Amazon EBS Events

The following are examples of the events for Amazon Elastic Block Store (Amazon EBS). For more information, see Amazon CloudWatch Events for Amazon EBS in the *Amazon EC2 User Guide for Linux Instances*.

EBS Snapshot Notification

Amazon EBS created a snapshot (`createSnapshot`), copied a snapshot (`copySnapshot`), or shared a snapshot (`shareSnapshot`). Note that the `source` field within the `detail` field does not include the account-id as part of the Volume Arn.

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "EBS Snapshot Notification",
  "source": "aws.ec2",
  "account": "123456789012",
  "time": "2016-11-14T01:30:00Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ec2::us-west-2:snapshot/snap-01234567"
  ],
  "detail": {
    "event": "createSnapshot",
    "result": "succeeded",
    "cause": "",
    "request-id": "",
    "snapshot_id": "arn:aws:ec2::us-west-2:snapshot/snap-01234567",
    "source": "arn:aws:ec2::us-west-2:volume/vol-01234567",
    "StartTime": "2016-11-14T00:00:00Z",
    "EndTime": "2016-11-ddT01:30:00Z"
  }
}
```

EBS Volume Notification

When a volume is successfully created or deleted, no events are generated. Events are generated when Amazon EBS fails to create a volume (`createVolume`), fails to attach a volume (`attachVolume`), or fails to reattach a volume (`reattachVolume`). The following example shows a failed attempt to create a volume.

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-0123456789ab",
  "detail-type": "EBS Volume Notification",
  "source": "aws.ec2",
  "account": "012345678901",
  "time": "2016-11-14T00:30:07Z",
  "region": "sa-east-1",
  "resources": [
    "arn:aws:ec2:sa-east-1:0123456789ab:volume/vol-01234567",
  ],
  "detail": {
    "event": "createVolume",
    "result": "failed",
    "cause": "arn:aws:kms:sa-east-1:0123456789ab:key/01234567-0123-0123-0123-0123456789ab
 is disabled.",
    "request-id": "01234567-0123-0123-0123-0123456789ab",

}
```

Amazon EC2 Events

The following is an example of an **EC2 Instance State-change Notification** event, with an instance in the `pending` state:

```
{
   "id":"7bf73129-1428-4cd3-a780-95db273d1602",
   "detail-type":"EC2 Instance State-change Notification",
   "source":"aws.ec2",
   "account":"123456789012",
   "time":"2015-11-11T21:29:54Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"
   ],
   "detail":{
      "instance-id":"i-abcd1111",
      "state":"pending"
   }
}
```

Amazon EC2 System Manager Events

The following are examples of the events for Amazon EC2 Systems Manager (Systems Manager). For more information, see Log Command Execution Status Changes for Run Command in the *Amazon EC2 User Guide for Linux Instances*.

EC2 Command Status-change Notification

```
{
    "version": "0",
    "id": "51c0891d-0e34-45b1-83d6-95db273d1602",
    "detail-type": "EC2 Command Status-change Notification",
    "source": "aws.ssm",
    "account": "123456789012",
    "time": "2016-07-10T21:51:32Z",
    "region": "us-east-1",
    "resources": ["arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"],
    "detail": {
        "command-id": "e8d3c0e4-71f7-4491-898f-c9b35bee5f3b",
        "document-name": "AWS-RunPowerShellScript",
        "expire-after": "2016-07-14T22:01:30.049Z",
        "parameters": {
            "executionTimeout": ["3600"],
            "commands": ["date"]
        },
        "requested-date-time": "2016-07-10T21:51:30.049Z",
        "status": "Success"
    }
}
```

EC2 Command Invocation Status-change Notification

```
{
    "version": "0",
    "id": "4780e1b8-f56b-4de5-95f2-95db273d1602",
    "detail-type": "EC2 Command Invocation Status-change Notification",
    "source": "aws.ssm",
    "account": "123456789012",
    "time": "2016-07-10T21:51:32Z",
    "region": "us-east-1",
    "resources": ["arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"],
    "detail": {
        "command-id": "e8d3c0e4-71f7-4491-898f-c9b35bee5f3b",
        "document-name": "AWS-RunPowerShellScript",
        "instance-id": "i-9bb89e2b",
        "requested-date-time": "2016-07-10T21:51:30.049Z",
        "status": "Success"
    }
}
```

EC2 Automation Step Status-change Notification

```
{
  "version": "0",
  "id": "eeca120b-a321-433e-9635-dab369006a6b",
  "detail-type": "EC2 Automation Step Status-change Notification",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2016-11-29T19:43:35Z",
  "region": "us-east-1",
  "resources": ["arn:aws:ssm:us-east-1:123456789012:automation-
execution/333ba70b-2333-48db-b17e-a5e69c6f4d1c",
    "arn:aws:ssm:us-east-1:123456789012:automation-definition/runcommand1:1"],
  "detail": {
    "ExecutionId": "333ba70b-2333-48db-b17e-a5e69c6f4d1c",
    "Definition": "runcommand1",
    "DefinitionVersion": 1.0,
    "Status": "Success",
    "EndTime": "Nov 29, 2016 7:43:25 PM",
    "StartTime": "Nov 29, 2016 7:43:23 PM",
    "Time": 2630.0,
    "StepName": "runFixedCmds",
    "Action": "aws:runCommand"
  }
}
```

EC2 Automation Execution Status-change Notification

```
{
  "version": "0",
  "id": "d290ece9-1088-4383-9df6-cd5b4ac42b99",
  "detail-type": "EC2 Automation Execution Status-change Notification",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2016-11-29T19:43:35Z",
  "region": "us-east-1",
  "resources": ["arn:aws:ssm:us-east-1:123456789012:automation-
execution/333ba70b-2333-48db-b17e-a5e69c6f4d1c",
    "arn:aws:ssm:us-east-1:123456789012:automation-definition/runcommand1:1"],
  "detail": {
    "ExecutionId": "333ba70b-2333-48db-b17e-a5e69c6f4d1c",
    "Definition": "runcommand1",
    "DefinitionVersion": 1.0,
    "Status": "Success",
    "StartTime": "Nov 29, 2016 7:43:20 PM",
    "EndTime": "Nov 29, 2016 7:43:26 PM",
    "Time": 5753.0,
    "ExecutedBy": "arn:aws:iam::123456789012:user/userName"
  }
}
```

Amazon EC2 Maintenance Window Events

The following are examples of the events for Amazon EC2 Maintenance Window.

Register a Target

The status could also be `DEREGISTERED`.

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-0123456789ab",
   "detail-type":"Maintenance Window Target Registration Notification",
   "source":"aws.ssm",
   "account":"012345678901",
   "time":"2016-11-16T00:58:37Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ssm:us-west-2:001312665065:maintenancewindow/mw-0ed7251d3fcf6e0c2",
      "arn:aws:ssm:us-west-2:001312665065:windowtarget/
e7265f13-3cc5-4f2f-97a9-7d3ca86c32a6"
   ],
   "detail":{
      "window-target-id":"e7265f13-3cc5-4f2f-97a9-7d3ca86c32a6",
      "window-id":"mw-0ed7251d3fcf6e0c2",
      "status":"REGISTERED"
   }
}
```

Window Execution Type

The other possibilities for status are `PENDING`, `IN_PROGRESS`, `SUCCESS`, `FAILED`, `TIMED_OUT`, and `SKIPPED_OVERLAPPING`.

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-0123456789ab",
   "detail-type":"Maintenance Window Execution State-change Notification",
   "source":"aws.ssm",
   "account":"012345678901",
   "time":"2016-11-16T01:00:57Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ssm:us-west-2:0123456789ab:maintenancewindow/mw-123456789012345678"
   ],
   "detail":{
      "start-time":"2016-11-16T01:00:56.427Z",
      "end-time":"2016-11-16T01:00:57.070Z",
      "window-id":"mw-0ed7251d3fcf6e0c2",
      "window-execution-id":"b60fb56e-776c-4e5c-84ee-123456789012",
      "status":"TIMED_OUT"
   }
}
```

Task Execution Type

The other possibilities for status are `IN_PROGRESS`, `SUCCESS`, `FAILED`, and `TIMED_OUT`.

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-0123456789ab",
   "detail-type":"Maintenance Window Task Execution State-change Notification",
   "source":"aws.ssm",
   "account":"012345678901",
   "time":"2016-11-16T01:00:56Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ssm:us-west-2:0123456789ab:maintenancewindow/mw-123456789012345678"
   ],
   "detail":{
      "start-time":"2016-11-16T01:00:56.759Z",
      "task-execution-id":"6417e808-7f35-4d1a-843f-123456789012",
      "end-time":"2016-11-16T01:00:56.847Z",
      "window-id":"mw-0ed7251d3fcf6e0c2",
      "window-execution-id":"b60fb56e-776c-4e5c-84ee-123456789012",
      "status":"TIMED_OUT"
   }
}
```

Task Target Processed

The other possibilities for status are `IN_PROGRESS`, `SUCCESS`, `FAILED`, and `TIMED_OUT`.

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-0123456789ab",
   "detail-type":"Maintenance Window Task Target Invocation State-change Notification",
   "source":"aws.ssm",
   "account":"012345678901",
   "time":"2016-11-16T01:00:57Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ssm:us-west-2:0123456789ab:maintenancewindow/mw-123456789012345678"
   ],
   "detail":{
      "start-time":"2016-11-16T01:00:56.427Z",
      "end-time":"2016-11-16T01:00:57.070Z",
      "window-id":"mw-0ed7251d3fcf6e0c2",
      "window-execution-id":"b60fb56e-776c-4e5c-84ee-123456789012",
      "task-execution-id":"6417e808-7f35-4d1a-843f-123456789012",
      "window-target-id":"e7265f13-3cc5-4f2f-97a9-123456789012",
      "status":"TIMED_OUT",
      "owner-information":"Owner"
   }
}
```

Window State Change

The possibilities for status are `ENABLED` and `DISABLED`.

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-0123456789ab",
   "detail-type":"Maintenance Window State-change Notification",
   "source":"aws.ssm",
   "account":"012345678901",
   "time":"2016-11-16T00:58:37Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ssm:us-west-2:0123456789ab:maintenancewindow/mw-123456789012345678"
   ],
   "detail":{
      "window-id":"mw-123456789012",
      "status":"DISABLED"
   }
}
```

Amazon ECS Events

The following are examples of the events for Amazon EC2 Container Service (Amazon ECS). For more information, see Amazon ECS Event Stream for CloudWatch Events in the *Amazon EC2 Container Service Developer Guide*.

ECS Container Instance State Change

Note that the contents of the detail section are not shown here. They resemble the contents of the ContainerInstance object.

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-0123456789ab",
  "detail-type": "ECS Container Instance State Change",
  "source": "aws.ecs",
  "account": "123456789012",
  "time": "2016-11-18T22:15:15Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:123456789012:container-instance/a48e7e5a-4709-47b3-
a698-819dab95c16f"
  ],
  "detail": {
    ...
  }
}
```

ECS Task State Change

Note that the contents of the detail section are not shown here. They resemble the contents of the Task object.

```
{
  "version": "0",
  "id": "2d882db5-3d34-4d75-b299-5a6af9d2a59c",
  "detail-type": "ECS Task State Change",
  "source": "aws.ecs",
  "account": "123456789012",
  "time": "2016-11-18T22:48:57Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:123456789012:task/9dd06983-dfd7-437c-8bb1-0dc78d90da91"
  ],
  "detail": {
    ...
  }
}
```

Amazon EMR Events

The following are examples of events for Amazon EMR.

Amazon EMR AutoScaling Policy State Change

```

{
   "version":"0",
   "id":"2f8147ab-8c48-47c6-b0b6-3ee23ec8d300",
   "detail-type":"EMR Auto Scaling Policy State Change",
   "source":"aws.emr",
   "account":"123456789012",
   "time":"2016-12-16T20:42:44Z",
   "region":"us-east-1",
   "resources":[],
   "detail":{
      "resourceId":"ig-X2LBMHTGPCBU",
      "clusterId":"j-1YONHTCP3YZKC",
      "state":"PENDING",
      "message":"AutoScaling policy modified by user request",
      "scalingResourceType":"INSTANCE_GROUP"
   }
}
```

Amazon EMR Cluster State Change - Starting

```
{
  "version": "0",
  "id": "999cccaa-eaaa-0000-1111-123456789012",
  "detail-type": "EMR Cluster State Change",
  "source": "aws.emr",
  "account": "123456789012",
  "time": "2016-12-16T20:43:05Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "severity": "INFO",
    "stateChangeReason": "{\"code\":\"\"}",
    "name": "Development Cluster",
    "clusterId": "j-123456789ABCD",
    "state": "STARTING",
    "message": "Amazon EMR cluster j-123456789ABCD (Development Cluster) was requested at
 2016-12-16 20:42 UTC and  is being created."
  }
}
```

Amazon EMR Cluster State Change - Terminated

```
{
  "version": "0",
  "id": "1234abb0-f87e-1234-b7b6-000000123456",
  "detail-type": "EMR Cluster State Change",
  "source": "aws.emr",
  "account": "123456789012",
  "time": "2016-12-16T21:00:23Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "severity": "INFO",
    "stateChangeReason": "{\"code\":\"USER_REQUEST\",\"message\":\"Terminated by user
 request\"}",
    "name": "Development Cluster",
    "clusterId": "j-123456789ABCD",
    "state": "TERMINATED",
    "message": "Amazon EMR Cluster jj-123456789ABCD (Development Cluster) has terminated at
 2016-12-16 21:00 UTC with a reason of USER_REQUEST."
  }
}
```

Amazon EMR Instance Group State Change

```
{
  "version": "0",
  "id": "999cccaa-eaaa-0000-1111-123456789012",
  "detail-type": "EMR Instance Group State Change",
  "source": "aws.emr",
  "account": "123456789012",
  "time": "2016-12-16T20:57:47Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "market": "ON_DEMAND",
    "severity": "INFO",
    "requestedInstanceCount": "2",
    "instanceType": "m3.xlarge",
    "instanceGroupType": "CORE",
    "instanceGroupId": "ig-ABCDEFGHIJKL",
    "clusterId": "j-123456789ABCD",
    "runningInstanceCount": "2",
    "state": "RUNNING",
    "message": "The resizing operation for instance group ig-ABCDEFGHIJKL in Amazon EMR
 cluster j-123456789ABCD (Development Cluster) is complete. It now has an instance count of
 2. The resize started at 2016-12-16 20:57 UTC and took 0 minutes to complete."
  }
}
```

Amazon EMR Step Status Change

```
{
  "version": "0",
  "id": "999cccaa-eaaa-0000-1111-123456789012",
  "detail-type": "EMR Step Status Change",
  "source": "aws.emr",
  "account": "123456789012",
  "time": "2016-12-16T20:53:09Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "severity": "ERROR",
    "actionOnFailure": "CONTINUE",
    "stepId": "s-ZYXWVUTSRQPON",
    "name": "CustomJAR",
    "clusterId": "j-123456789ABCD",
    "state": "FAILED",
    "message": "Step s-ZYXWVUTSRQPON (CustomJAR) in Amazon EMR cluster j-123456789ABCD
 (Development Cluster) failed at 2016-12-16 20:53 UTC."
  }
}
```

Auto Scaling Events

The following are examples of the events for Auto Scaling. For more information, see Getting CloudWatch Events When Your Auto Scaling Group Scales in the *Auto Scaling User Guide*.

EC2 Instance-launch Lifecycle Action

Auto Scaling moved an instance to a `Pending:Wait` state due to a lifecycle hook.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "EC2 Instance-launch Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2015-12-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:autoscaling:us-east-1:123456789012:autoScalingGroup:59fcbb81-
bd02-485d-80ce-563ef5b237bf:autoScalingGroupName/sampleASG"
  ],
  "detail": {
    "LifecycleActionToken": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "sampleASG",
    "LifecycleHookName": "SampleLifecycleHook-12345",
    "EC2InstanceId": "i-12345678",
    "LifecycleTransition": "autoscaling:EC2_INSTANCE_LAUNCHING"
  }
}
```

EC2 Instance Launch Successful

Auto Scaling successfully launched an instance.

```
{
  "id": "3e3c153a-8339-4e30-8c35-687ebef853fe",
  "detail-type": "EC2 Instance Launch Successful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2015-11-11T21:31:47Z",
  "region": "us-east-1",
  "resources": [
      "arn:aws:autoscaling:us-east-1:123456789012:autoScalingGroup:eb56d16b-bbf0-401d-b893-
d5978ed4a025:autoScalingGroupName/ASGLaunchSuccess",
      "arn:aws:ec2:us-east-1:123456789012:instance/i-b188560f"
      ],
  "detail": {
      "StatusCode": "InProgress",
      "AutoScalingGroupName": "ASGLaunchSuccess",
      "ActivityId": "9cabb81f-42de-417d-8aa7-ce16bf026590",
      "Details": {
            "Availability Zone": "us-east-1b",
            "Subnet ID": "subnet-95bfcebe"
      },
      "RequestId": "9cabb81f-42de-417d-8aa7-ce16bf026590",
      "EndTime": "2015-11-11T21:31:47.208Z",
      "EC2InstanceId": "i-b188560f",
      "StartTime": "2015-11-11T21:31:13.671Z",
      "Cause": "At 2015-11-11T21:31:10Z a user request created an Auto Scaling group
 changing the desired capacity from 0 to 1. At 2015-11-11T21:31:11Z an instance was started
 in response to a difference between desired and actual capacity, increasing the capacity
 from 0 to 1."
      }
}
```

EC2 Instance Launch Unsuccessful

Auto Scaling failed to launch an instance.

```
{
  "id": "1681ab87-4a09-459f-95a2-7fa09403c4b7",
  "detail-type": "EC2 Instance Launch Unsuccessful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2015-11-11T21:42:36Z",
  "region": "us-east-1",
  "resources": [
      "arn:aws:autoscaling:us-east-1:123456789012:autoScalingGroup:528ffce5-
ef9f-4c1d-8d18-5d005b4a438c:autoScalingGroupName/brokenASG",
      "arn:aws:ec2:us-east-1:123456789012:instance/"
      ],
  "detail": {
      "StatusCode": "Failed",
      "AutoScalingGroupName": "brokenASG",
      "ActivityId": "06076c51-4874-487d-b15b-7895a713ab55",
      "Details": {
            "Availability Zone": "us-east-1e",
            "Subnet ID": "subnet-16c5df2c"
       },
      "RequestId": "06076c51-4874-487d-b15b-7895a713ab55",
      "EndTime": "2015-11-11T21:42:36.000Z",
      "EC2InstanceId": "",
      "StartTime": "2015-11-11T21:42:36.698Z",
      "Cause": "At 2015-11-11T21:42:09Z a user request update of Auto Scaling group
 constraints to min: 0, max: 10, desired: 2 changing the desired capacity from 0 to 2. At
 2015-11-11T21:42:35Z an instance was started in response to a difference between desired
 and actual capacity, increasing the capacity from 0 to 2."
      }
      }
```

EC2 Instance-terminate Lifecycle Action

Auto Scaling moved an instance to a `Terminating:Wait` state due to a lifecycle hook.

```
{
  "version": "0",
  "id": "468fe059-f4b7-445f-bb22-2a271b94974d",
  "detail-type": "EC2 Instance-terminate Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2015-12-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:autoscaling:us-east-1:123456789012:autoScalingGroup:59fcbb81-
bd02-485d-80ce-563ef5b237bf:autoScalingGroupName/sampleASG"
  ],
  "detail": {
    "LifecycleActionToken": "630aa23f-48eb-45e7-aba6-799ea6093a0f",
    "AutoScalingGroupName": "sampleASG",
    "LifecycleHookName": "SampleLifecycleHook-6789",
    "EC2InstanceId": "i-12345678",
    "LifecycleTransition": "autoscaling:EC2_INSTANCE_TERMINATING"
  }
}
```

EC2 Instance Terminate Successful

Auto Scaling successfully terminated an instance.

```
{
  "id": "156d01c9-a6c3-4d7e-b883-5758266b95af",
  "detail-type": "EC2 Instance Terminate Successful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2015-11-11T21:36:57Z",
  "region": "us-east-1",
  "resources": [
      "arn:aws:autoscaling:us-east-1:123456789012:autoScalingGroup:eb56d16b-bbf0-401d-b893-
d5978ed4a025:autoScalingGroupName/ASGTerminate",
      "arn:aws:ec2:us-east-1:123456789012:instance/i-b188560f"
      ],
  "detail": {
      "StatusCode": "InProgress",
      "AutoScalingGroupName": "ASGTerminate",
      "ActivityId": "56472e79-538a-4ba7-b3cc-768d889194b0",
      "Details": {
            "Availability Zone": "us-east-1b",
            "Subnet ID": "subnet-95bfcebe"
            },
      "RequestId": "56472e79-538a-4ba7-b3cc-768d889194b0",
      "EndTime": "2015-11-11T21:36:57.498Z",
      "EC2InstanceId": "i-b188560f",
      "StartTime": "2015-11-11T21:36:12.649Z",
      "Cause": "At 2015-11-11T21:36:03Z a user request update of Auto Scaling group
 constraints to min: 0, max: 1, desired: 0 changing the desired capacity from 1 to
 0. At 2015-11-11T21:36:12Z an instance was taken out of service in response to a
 difference between desired and actual capacity, shrinking the capacity from 1 to 0. At
 2015-11-11T21:36:12Z instance i-b188560f was selected for termination."
      }
}
```

EC2 Instance Terminate Unsuccessful

Auto Scaling failed to terminate an instance.

```
{
  "id": "5e3df53a-0239-4e31-7d15-087ebef903ce",
  "detail-type": "EC2 Instance Terminate Unsuccessful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2015-12-01T23:34:57Z",
  "region": "us-east-1",
  "resources": [
      "arn:aws:autoscaling:us-east-1:123456789012:autoScalingGroup:cf5ebd9c-8e2a-4197-
abe2-2fb94e8d1f87:autoScalingGroupName/ASGTermFail",
      "arn:aws:ec2:us-east-1:123456789012:instance/i-b188560f"
      ],
  "detail": {
      "StatusCode": "InProgress",
      "Description": "Terminating EC2 instance: i-b188560f",
      "AutoScalingGroupName": "ASGTermFail",
      "ActivityId": "c1a8f6ce-82e8-4517-96ba-67d1999ceee4",
      "Details": {
            "Availability Zone": "us-east-1e",
            "Subnet ID": "subnet-915643ba"
            },
      "RequestId": "c1a8f6ce-82e8-4517-96ba-67d1999ceee4",
      "StatusMessage": "",
      "EndTime": "2015-12-01T23:34:57.721Z",
      "EC2InstanceId": "i-b188560f",
      "StartTime": "2015-12-01T23:33:48.489Z",
      "Cause": "At 2015-12-01T23:33:41Z a user request explicitly set group desired
 capacity changing the desired capacity from 2 to 0. At 2015-12-01T23:33:47Z an instance
 was taken out of service in response to a difference between desired and actual capacity,
 shrinking the capacity from 2 to 0. At 2015-12-01T23:33:47Z instance i-0867b4292c0cff474
 was selected for termination. At 2015-12-01T23:33:48Z instance i-b188560f was selected for
 termination."
      }
}
```

AWS API Call Events

The following is an example of an AWS API call event to Amazon S3 to create a bucket:

```
{
    "version": "0",
    "id": "36eb8523-97d0-4518-b33d-ee3579ff19f0",
    "detail-type": "AWS API Call via CloudTrail",
    "source": "aws.s3",
    "account": "123456789012",
    "time": "2016-02-20T01:09:13Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "eventVersion": "1.03",
        "userIdentity": {
            "type": "Root",
            "principalId": "123456789012",
            "arn": "arn:aws:iam::123456789012:root",
            "accountId": "123456789012",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2016-02-20T01:05:59Z"
                }
            }
        },
        "eventTime": "2016-02-20T01:09:13Z",
        "eventSource": "s3.amazonaws.com",
        "eventName": "CreateBucket",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "100.100.100.100",
        "userAgent": "[S3Console/0.4]",
        "requestParameters": {
            "bucketName": "bucket-test-iad"
        },
        "responseElements": null,
        "requestID": "9D767BCC3B4E7487",
        "eventID": "24ba271e-d595-4e66-a7fd-9c16cbf8abae",
        "eventType": "AwsApiCall"
    }
}
```

Only the read/write events from the following services are supported. Read-only APIs, such as those that begin with **List**, **Get**, or **Describe** aren't supported. In addition, AWS API call events that are larger than 256KB in size are not supported.

- Auto Scaling
- AWS Certificate Manager
- AWS CloudFormation
- Amazon CloudFront
- AWS CloudHSM
- Amazon CloudSearch
- AWS CloudTrail
- Amazon CloudWatch
- Amazon CloudWatch Events
- Amazon CloudWatch Logs
- AWS CodeDeploy
- AWS CodePipeline
- Amazon Cognito Identity
- Amazon Cognito Sync

- AWS Config
- AWS Data Pipeline
- AWS Device Farm
- AWS Direct Connect
- AWS Directory Service
- AWS Database Migration Service
- Amazon DynamoDB
- Amazon EC2 Container Registry
- Amazon EC2 Container Service
- Amazon EC2 Systems Manager
- Amazon ElastiCache
- AWS Elastic Beanstalk
- Amazon Elastic Compute Cloud
- Amazon Elastic File System
- Elastic Load Balancing
- Amazon EMR
- Amazon Elastic Transcoder
- Amazon Elasticsearch Service
- Amazon GameLift
- Amazon Glacier
- AWS Identity and Access Management [US East (N. Virginia) only]
- Amazon Inspector
- AWS IoT
- AWS Key Management Service
- Amazon Kinesis
- Amazon Kinesis Firehose
- AWS Lambda
- Amazon Machine Learning
- AWS OpsWorks
- Amazon Polly
- Amazon Redshift
- Amazon Relational Database Service
- Amazon Route 53
- AWS Security Token Service
- Amazon Simple Email Service
- Amazon Simple Notification Service
- Amazon Simple Queue Service
- Amazon Simple Storage Service
- Amazon Simple Workflow Service
- AWS Step Functions
- AWS Storage Gateway
- AWS Support
- AWS WAF

- Amazon WorkDocs
- Amazon WorkSpaces

AWS CodeDeploy Events

The following are examples of the events for AWS CodeDeploy. For more information, see Monitoring Deployments with CloudWatch Events in the *AWS CodeDeploy User Guide*.

CodeDeploy Deployment State-change Notification

There was a change in the state of a deployment.

```
{
  "account": "123456789012",
  "region": "us-east-1",
  "detail-type": "CodeDeploy Deployment State-change Notification",
  "source": "aws.codedeploy",
  "version": "0",
  "time": "2016-06-30T22:06:31Z",
  "id": "c071bfbf-83c4-49ca-a6ff-3df053957145",
  "resources": [
    "arn:aws:codedeploy:us-east-1:123456789012:application:myApplication",
    "arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:myApplication/
myDeploymentGroup"
  ],
  "detail": {
    "instanceGroupId": "9fd2fbef-2157-40d8-91e7-6845af69e2d2",
    "region": "us-east-1",
    "application": "myApplication",
    "deploymentId": "d-123456789",
    "state": "SUCCESS",
    "deploymentGroup": "myDeploymentGroup"
  }
}
```

CodeDeploy Instance State-change Notification

There was a change in the state of an instance that belongs to a deployment group.

```
{
  "account": "123456789012",
  "region": "us-east-1",
  "detail-type": "CodeDeploy Instance State-change Notification",
  "source": "aws.codedeploy",
  "version": "0",
  "time": "2016-06-30T23:18:50Z",
  "id": "fb1d3015-c091-4bf9-95e2-d98521ab2ecb",
  "resources": [
    "arn:aws:ec2:us-east-1:123456789012:instance/i-0000000aaaaaaaaaa",
    "arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:myApplication/
myDeploymentGroup",
    "arn:aws:codedeploy:us-east-1:123456789012:application:myApplication"
  ],
  "detail": {
    "instanceId": "i-0000000aaaaaaaaaa",
    "region": "us-east-1",
    "state": "SUCCESS",
    "application": "myApplication",
    "deploymentId": "d-123456789",
    "instanceGroupId": "8cd3bfa8-9e72-4cbe-a1e5-da4efc7efd49",
    "deploymentGroup": "myDeploymentGroup"
  }
}
```

AWS Console Sign-in Events

AWS console sign-in events are supported only in the US East (N. Virginia) region. The following is an example of an AWS console sign-in event:

```
{
  "id": "6f87d04b-9f74-4f04-a780-7acf4b0a9b38",
  "detail-type": "AWS Console Sign In via CloudTrail",
  "source": "aws.signin",
  "account": "123456789012",
  "time": "2016-01-05T18:21:27Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
      "eventVersion": "1.02",
      "userIdentity": {
            "type": "Root",
            "principalId": "123456789012",
            "arn": "arn:aws:iam::123456789012:root",
            "accountId": "123456789012"
            },
      "eventTime": "2016-01-05T18:21:27Z",
      "eventSource": "signin.amazonaws.com",
      "eventName": "ConsoleLogin",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "0.0.0.0",
      "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36",
      "requestParameters": null,
      "responseElements": {
            "ConsoleLogin": "Success"
            },
      "additionalEventData": {
            "LoginTo": "https://console.aws.amazon.com/console/home?state=hashArgs
%23&isauthcode=true",
            "MobileVersion": "No",
            "MFAUsed": "No" },
      "eventID": "324731c0-64b3-4421-b552-dfc3c27df4f6",
      "eventType": "AwsConsoleSignIn"
      }
}
```

AWS Health Events

The following is the format for the AWS Personal Health Dashboard (AWS Health) events. For more information, see Managing AWS Health Events with Amazon CloudWatch Events in the *AWS Health User Guide*.

AWS Health Event Format

```
{
  "version": "0",
  "id": "7bf73129-1428-4cd3-a780-95db273d1602",
  "detail-type": "AWS Health Event",
  "source": "aws.health",
  "account": "123456789012",
  "time": "2016-06-05T06:27:57Z",
  "region": "region",
  "resources": [],
  "detail": {
    "eventArn": "arn:aws:health:region::event/id",
    "service": "service",
    "eventTypeCode": "AWS_service_code",
    "eventTypeCategory": "category",
    "startTime": "Sun, 05 Jun 2016 05:01:10 GMT",
    "endTime": "Sun, 05 Jun 2016 05:30:57 GMT",
    "eventDescription": [{
      "language": "lang-code",
      "latestDescription": "description"
    }]
    ...
  }
}
```

category

The category code of the event. The possible values are `issue`, `accountNotification`, and `scheduledChange`.

code

The unique identifier for the event type.

id

The unique identifier for the event.

service

The AWS service affected by the event. For example, `EC2`, `S3`, `REDSHIFT`, and `RDS`.

Elastic Load Balancing API Issue

```
{
  "version": "0",
  "id": "121345678-1234-1234-1234-123456789012",
  "detail-type": "AWS Health Event",
  "source": "aws.health",
  "account": "123456789012",
  "time": "2016-06-05T06:27:57Z",
  "region": "ap-southeast-2",
  "resources": [],
  "detail": {
    "eventArn": "arn:aws:health:ap-southeast-2::event/
AWS_ELASTICLOADBALANCING_API_ISSUE_90353408594353980",
    "service": "ELASTICLOADBALANCING",
    "eventTypeCode": "AWS_ELASTICLOADBALANCING_API_ISSUE",
    "eventTypeCategory": "issue",
    "startTime": "Sat, 11 Jun 2016 05:01:10 GMT",
    "endTime": "Sat, 11 Jun 2016 05:30:57 GMT",
    "eventDescription": [{
      "language": "en_US",
      "latestDescription": "A description of the event will be provided here"
  }
}
```

Amazon EC2 Instance Store Drive Performance Degraded

```
{
  "version": "0",
  "id": "121345678-1234-1234-1234-123456789012",
  "detail-type": "AWS Health Event",
  "source": "aws.health",
  "account": "123456789012",
  "time": "2016-06-05T06:27:57Z",
  "region": "us-west-2",
  "resources": [
    "i-abcd1111"
  ],
  "detail": {
    "eventArn": "arn:aws:health:us-west-2::event/
AWS_EC2_INSTANCE_STORE_DRIVE_PERFORMANCE_DEGRADED_90353408594353980",
    "service": "EC2",
    "eventTypeCode": "AWS_EC2_INSTANCE_STORE_DRIVE_PERFORMANCE_DEGRADED",
    "eventTypeCategory": "issue",
    "startTime": "Sat, 05 Jun 2016 15:10:09 GMT",
    "eventDescription": [{
      "language": "en_US",
      "latestDescription": "A description of the event will be provided here"
    }],
    "affectedEntities": [{
      "entityValue": "i-abcd1111",
      "tags": {
        "stage": "prod",
        "app": "my-app"
  }
}
```

AWS KMS Events

The following are examples of the AWS Key Management Service (AWS KMS) events. For more information, see AWS KMS Events in the *AWS Key Management Service Developer Guide*.

KMS CMK Rotation

AWS KMS automatically rotated a CMK's key material.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "KMS CMK Rotation",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2016-08-25T21:05:33Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

KMS Imported Key Material Expiration

AWS KMS deleted a CMK's expired key material.

```
{
  "version": "0",
  "id": "9da9af57-9253-4406-87cb-7cc400e43465",
  "detail-type": "KMS Imported Key Material Expiration",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2016-08-22T20:12:19Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

KMS CMK Deletion

AWS KMS completed a scheduled CMK deletion.

```
{
  "version": "0",
  "id": "e9ce3425-7d22-412a-a699-e7a5fc3fbc9a",
  "detail-type": "KMS CMK Deletion",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2016-08-19T03:23:45Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

Scheduled Events

The following is an example of a scheduled event:

```
{
  "id": "53dc4d37-cffa-4f76-80c9-8b7d4a4d2eaa",
  "detail-type": "Scheduled Event",
  "source": "aws.events",
  "account": "123456789012",
  "time": "2015-10-08T16:53:06Z",
  "region": "us-east-1",
  "resources": [ "arn:aws:events:us-east-1:123456789012:rule/MyScheduledRule" ],
  "detail": {}
}
```

Trusted Advisor Events

The following are examples of the events for AWS Trusted Advisor. For more information, see Monitoring Trusted Advisor Check Results with Amazon CloudWatch Events in the *AWS Support User Guide*.

Low Utilization Amazon EC2 Instances

```
{
  "check-name": "Low Utilization Amazon EC2 Instances",
  "check-item-detail": {
    "Day 1": "0.0%  0.00MB",
    "Day 2": "0.0%  0.00MB",
    "Day 3": "0.0%  0.00MB",
    "Region/AZ": "eu-central-1a",
    "Estimated Monthly Savings": "$10.80",
    "14-Day Average CPU Utilization": "0.0%",
    "Day 14": "0.0%  0.00MB",
    "Day 13": "0.0%  0.00MB",
    "Day 12": "0.0%  0.00MB",
    "Day 11": "0.0%  0.00MB",
    "Day 10": "0.0%  0.00MB",
    "14-Day Average Network I/O": "0.00MB",
    "Number of Days Low Utilization": "14 days",
    "Instance Type": "t2.micro",
    "Instance ID": "i-1a2b3e4f",
    "Day 8": "0.0%  0.00MB",
    "Instance Name": null,
    "Day 9": "0.0%  0.00MB",
    "Day 4": "0.0%  0.00MB",
    "Day 5": "0.0%  0.00MB",
    "Day 6": "0.0%  0.00MB",
    "Day 7": "0.0%  0.00MB"
  },
  "status": "WARN",
  "resource_id": "arn:aws:ec2:eu-central-1:111122223333:instance/i-1a2b3e4f",
  "uuid": "6ba6d96a-d3dd-4fca-8020-350b2e54719c"
}
```

Load Balancer Optimization

```
{
  "check-name": "Load Balancer Optimization",
  "check-item-detail": {
    "Instances in Zone a": "0",
    "Status": "Yellow",
    "Instances in Zone b": null,
    "# of Zones": "1",
    "Region": "ap-northeast-2",
    "Load Balancer Name": "xyz-elb-test",
    "Instances in Zone e": null,
    "Instances in Zone c": null,
    "Reason": "No active instances",
    "Instances in Zone d": null
  },
  "status": "WARN",
  "resource_id": "arn:aws:elasticloadbalancing:ap-northeast-2:444455556666:loadbalancer/
xyz-elb-test",
  "uuid": "a1bc339a-59c8-4b5f-b248-44c437b68b83"
}
```

Exposed Access Keys

```
{
  "check-name": "Exposed Access Keys",
  "check-item-detail": {
    "Case ID": "02648f3b-e18f-4019-8d68-ce25efe080ff",
    "Usage (USD per Day)": "0",
    "User Name (IAM or Root)": "jane-roe-test",
    "Deadline": "1440453299248",
    "Access Key ID": "AKIAIOSFODNN7EXAMPLE",
    "Time Updated": "1440021299248",
    "Fraud Type": "Exposed",
    "Location": "www.github.com"
  },
  "status": "ERROR",
  "resource_id": "",
  "uuid": "cce6d28f-e44b-4e61-aba1-5b4af96a0f59"
}
```

Authentication and Access Control for Amazon CloudWatch Events

Access to Amazon CloudWatch Events requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources, such as retrieving event data from other AWS resources. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and CloudWatch Events to help secure your resources by controlling who can access them:

- Authentication (p. 56)
- Access Control (p. 57)

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 send event data to a target in CloudWatch Events). 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 Events 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 Events resources. For example, you must have permissions to invoke AWS Lambda, Amazon Simple Notification Service (Amazon SNS), and Amazon Simple Queue Service (Amazon SQS) targets associated with your CloudWatch Events rules.

The following sections describe how to manage permissions for CloudWatch Events. We recommend that you read the overview first.

- Overview of Managing Access Permissions to Your CloudWatch Events Resources (p. 58)
- Using Identity-Based Policies (IAM Policies) for CloudWatch Events (p. 61)
- Using Resource-Based Policies for CloudWatch Events (p. 68)
- CloudWatch Events Permissions Reference (p. 71)

Overview of Managing Access Permissions to Your CloudWatch Events 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 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 Events Resources and Operations (p. 58)
- Understanding Resource Ownership (p. 59)
- Managing Access to Resources (p. 59)
- Specifying Policy Elements: Actions, Effects, and Principals (p. 60)
- Specifying Conditions in a Policy (p. 61)

CloudWatch Events Resources and Operations

In CloudWatch Events, the primary resource is a rule. CloudWatch Events supports other resources that can be used with the primary resource, such as events. These are referred to as subresources. These resources and subresources have unique Amazon Resource Names (ARNs) associated with them. For more information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the *Amazon Web Services General Reference*.

Resource Type	ARN Format
Rule	`arn:aws:events:region:account:rule/rule-name`
All CloudWatch Events resources	`arn:aws:events:*`
All CloudWatch Events resources owned by the specified account in the specified region	`arn:aws:events:region:account:*`

Note
Most services in AWS treat a colon (:) or a forward slash (/) as the same character in ARNs. However, CloudWatch Events uses an exact match in event patterns and rules. Be sure to use the correct ARN characters when creating event patterns so that they match the ARN syntax in the event you want to match.

For example, you can indicate a specific rule (*myRule*) in your statement using its ARN as follows:

```
"Resource": "arn:aws:events:us-east-1:123456789012:rule/myRule"
```

You can also specify all rules that belong to a specific account by using the asterisk (*) wildcard as follows:

```
"Resource": "arn:aws:events:us-east-1:123456789012:rule/*"
```

To specify all resources, or if a specific API action does not support ARNs, use the asterisk (*) wildcard in the Resource element as follows:

```
"Resource": "*"
```

Some CloudWatch Events API actions accept multiple resources (that is, `PutTargets`). To specify multiple resources in a single statement, separate their ARNs with commas, as follows:

```
"Resource": ["arn1", "arn2"]
```

CloudWatch Events provides a set of operations to work with the CloudWatch Events resources. For a list of available operations, see CloudWatch Events Permissions Reference (p. 71).

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 rule, your AWS account is the owner of the CloudWatch Events resource.
- If you create an IAM user in your AWS account and grant permissions to create CloudWatch Events resources to that user, the user can create CloudWatch Events resources. However, your AWS account, to which the user belongs, owns the CloudWatch Events resources.
- If you create an IAM role in your AWS account with permissions to create CloudWatch Events resources, anyone who can assume the role can create CloudWatch Events resources. Your AWS account, to which the role belongs, owns the CloudWatch Events 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 Events. 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 AWS 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 Events supports both identity-based (IAM policies) and resource-based policies.

Topics

- Identity-Based Policies (IAM Policies) (p. 59)
- Resource-Based Policies (p. 60)

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 rules in the CloudWatch 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 can also be an AWS service principal 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*.

You can create specific IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users. For more information about how to create IAM roles and to explore example IAM policy statements for CloudWatch Events, see Overview of Managing Access Permissions to Your CloudWatch Events Resources (p. 58).

Resource-Based Policies

When a rule is triggered in CloudWatch Events, all the targets associated with the rule are invoked. *Invocation* means invoking the AWS Lambda functions, publishing to the Amazon SNS topics, and relaying the event to the Amazon Kinesis streams. In order to be able to make API calls against the resources you own, CloudWatch Events needs the appropriate permissions. For Lambda, Amazon SNS, and Amazon SQS resources, CloudWatch Events relies on resource-based policies. For Amazon Kinesis streams, CloudWatch Events relies on IAM roles.

For more information about how to create IAM roles and to explore example resource-based policy statements for CloudWatch Events, see Using Resource-Based Policies for CloudWatch Events (p. 68).

Specifying Policy Elements: Actions, Effects, and Principals

For each CloudWatch Events resource, the service defines a set of API operations. To grant permissions for these API operations, CloudWatch Events 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 Events Resources and Operations (p. 58) and CloudWatch Events Permissions Reference (p. 71).

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 Events Resources and Operations (p. 58).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, the `events:Describe` permission allows the user permissions to perform the `Describe` 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).

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 Events API actions and the resources that they apply to, see CloudWatch Events Permissions Reference (p. 71).

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. There are AWS-wide condition keys and CloudWatch Events–specific keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*. For a complete list of CloudWatch Events–specific keys, see Using IAM Policy Conditions for Fine-Grained Access Control (p. 73).

Using Identity-Based Policies (IAM Policies) for CloudWatch Events

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

The following shows an example of a permissions policy that allows a user to put event data into Amazon Kinesis.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchEventsInvocationAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord"
            ],
            "Resource": "*"
        }
    ]
}
```

The sections in this topic cover the following:

Topics

- Permissions Required to Use the CloudWatch Console (p. 62)
- AWS Managed (Predefined) Policies for CloudWatch Events (p. 63)
- Permissions Required for CloudWatch Events to Access Certain Targets (p. 64)

- Customer Managed Policy Examples (p. 65)

Permissions Required to Use the CloudWatch Console

For a user to work with CloudWatch Events in the CloudWatch console, that user must have a minimum set of permissions that allows the user to describe other AWS resources for their AWS account. In order to use CloudWatch Events in the CloudWatch console, you must have permissions from the following services:

- Automation
- Auto Scaling
- CloudTrail
- CloudWatch
- CloudWatch Events
- IAM
- Amazon Kinesis
- Lambda
- Amazon SNS
- Amazon SWF

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 `CloudWatchEventsReadOnlyAccess` managed policy to the user, as described in AWS Managed (Predefined) Policies for CloudWatch Events (p. 63).

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the CloudWatch API.

The full set of permissions required to work with the CloudWatch console are listed below:

- `automation:CreateAction`
- `automation:DescribeAction`
- `automation:UpdateAction`
- `autoscaling:DescribeAutoScalingGroups`
- `cloudtrail:DescribeTrails`
- `ec2:DescribeInstances`
- `ec2:DescribeVolumes`
- `events:DeleteRule`
- `events:DescribeRule`
- `events:DisableRule`
- `events:EnableRule`
- `events:ListRuleNamesByTarget`
- `events:ListRules`
- `events:ListTargetsByRule`
- `events:PutEvents`
- `events:PutRule`
- `events:PutTargets`
- `events:RemoveTargets`

- `events:TestEventPattern`
- `iam:ListRoles`
- `kinesis:ListStreams`
- `lambda:AddPermission`
- `lambda:ListFunctions`
- `lambda:RemovePermission`
- `sns:GetTopicAttributes`
- `sns:ListTopics`
- `sns:SetTopicAttributes`
- `swf:DescribeAction`
- `swf:ReferenceAction`
- `swf:RegisterAction`
- `swf:RegisterDomain`
- `swf:UpdateAction`

AWS Managed (Predefined) Policies for CloudWatch Events

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 Events:

- **CloudWatchEventsFullAccess** – Grants full access to CloudWatch Events.
- **CloudWatchEventsInvocationAccess** – Allows CloudWatch Events to relay events to the streams in Amazon Kinesis Streams in your account.
- **CloudWatchEventsReadOnlyAccess** – Grants read-only access to CloudWatch Events.
- **CloudWatchEventsBuiltInTargetExecutionAccess** – Allows built-in targets in CloudWatch Events to perform Amazon EC2 actions on your behalf.

IAM Roles for Sending Events

In order for CloudWatch Events to relay events to your Amazon Kinesis stream targets, you must create an IAM role.

To create an IAM role for sending CloudWatch Events

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Follow the steps in Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide* to create an IAM role. As you follow the steps to create a role, do the following:
   - In **Role Name**, use a name that is unique within your AWS account (for example, **CloudWatchEventsSending**).
   - In **Select Role Type**, choose **AWS Service Roles**, and then choose **Amazon CloudWatch Events**. This grants CloudWatch Events permissions to assume the role.

- In **Attach Policy**, choose **CloudWatchEventsInvocationAccess**.

You can also create your own custom IAM policies to allow permissions for CloudWatch Events actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions. For more information about IAM policies, see Overview of IAM Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies in the *IAM User Guide*.

Permissions Required for CloudWatch Events to Access Certain Targets

For CloudWatch Events to access certain targets, you must specify an IAM role for accessing that target, and that role must have a certain policy attached.

If the target is an Amazon Kinesis stream, the role used to send event data to that target must include the following policy:

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

If the target is Amazon EC2 Run Command and you are specifying one or more InstanceIds values for the command, the role that you specify must include the following policy:

```
{
     "Version": "2012-10-17",
     "Statement": [
           {
                "Action": "ssm:SendCommand",
                "Effect": "Allow",
                "Resource": [
                     "arn:aws:ec2:{{region}}:{{accountId}}:instance/[[instanceIds]]",
                     "arn:aws:ssm:{{region}}:*:document/{{documentName}}"
                ]
           }
     ]
}
```

If the target is Amazon EC2 Run Command and you are specifying one or more tags for the command, the role that you specify must include the following policy:

```
{
     "Version": "2012-10-17",
     "Statement": [
           {
                "Action": "ssm:SendCommand",
                "Effect": "Allow",
                "Resource": [
                    "arn:aws:ec2:{{region}}:{{accountId}}:instance/*"
                ],
                "Condition": {
                    "StringEquals": {
                        "ec2:ResourceTag/*": [
                            "[[tagValues]]"
                        ]
                    }
                }
            },
            {
                "Action": "ssm:SendCommand",
                "Effect": "Allow",
                "Resource": [
                     "arn:aws:ssm:{{region}}:*:document/{{documentName}}"
                ]
            }
     ]
}
```

If the target is an Step Functions state machine, the role that you specify must include the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Action": [ "states:StartExecution" ],
            "Resource": [ "arn:aws:states:*:*:stateMachine:*" ]
        }
     ]
}
```

If the target is an ECS task, the role that you specify must include the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "ecs:RunTask"
        ],
        "Resource": [
            "arn:aws:ecs:*:{{account-id}}:task-definition/{{task-definition-name}}"
        ],
        "Condition": {
            "ArnLike": {
                "ecs:cluster": "arn:aws:ecs:*:{{account-id}}:cluster/{{cluster-name}}"
            }
        }
    }]
}
```

Customer Managed Policy Examples

In this section, you can find example user policies that grant permissions for various CloudWatch Events actions. These policies work when you are using the CloudWatch Events API, AWS SDKs, or the AWS CLI.

Note
All examples use the US West (Oregon) Region (us-west-2) and contain fictitious account IDs.

You can use the following sample IAM policies listed to limit the CloudWatch Events access for your IAM users and roles.

Examples

- Example 1: CloudWatchEventsBuiltInTargetExecutionAccess (p. 66)
- Example 2: CloudWatchEventsInvocationAccess (p. 66)
- Example 3: CloudWatchEventsConsoleAccess (p. 66)
- Example 4: CloudWatchEventsFullAccess (p. 67)
- Example 5: CloudWatchEventsReadOnlyAccess (p. 68)

Example 1: CloudWatchEventsBuiltInTargetExecutionAccess

The following policy allows built-in targets in CloudWatch Events to perform Amazon EC2 actions on your behalf.

Important
Creating rules with built-in targets is supported only in the AWS Management Console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchEventsBuiltInTargetExecutionAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:RebootInstances",
                "ec2:StopInstances",
                "ec2:TerminateInstances",
                "ec2:CreateSnapshot"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 2: CloudWatchEventsInvocationAccess

The following policy allows CloudWatch Events to relay events to the streams in Amazon Kinesis streams in your account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchEventsInvocationAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 3: CloudWatchEventsConsoleAccess

The following policy ensures that IAM users can use the CloudWatch Events console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchEventsConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "automation:CreateAction",
                "automation:DescribeAction",
                "automation:UpdateAction",
                "autoscaling:DescribeAutoScalingGroups",
                "cloudtrail:DescribeTrails",
                "ec2:DescribeInstances",
                "ec2:DescribeVolumes",
                "events:*",
                "iam:ListRoles",
                "kinesis:ListStreams",
                "lambda:AddPermission",
                "lambda:ListFunctions",
                "lambda:RemovePermission",
                "sns:GetTopicAttributes",
                "sns:ListTopics",
                "sns:SetTopicAttributes",
                "swf:DescribeAction",
                "swf:ReferenceAction",
                "swf:RegisterAction",
                "swf:RegisterDomain",
                "swf:UpdateAction"
            ],
            "Resource": "*"
        },
        {
            "Sid": "IAMPassRoleForCloudWatchEvents",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::*:role/AWS_Events_Invoke_Targets",
                "arn:aws:iam::*:role/AWS_Events_Actions_Execution"
            ]
        }
    ]
}
```

Example 4: CloudWatchEventsFullAccess

The following policy allows performing actions against CloudWatch Events through the AWS CLI and SDK.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchEventsFullAccess",
            "Effect": "Allow",
            "Action": "events:*",
            "Resource": "*"
        },
        {
            "Sid": "IAMPassRoleForCloudWatchEvents",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
        }
    ]
}
```

Example 5: CloudWatchEventsReadOnlyAccess

The following policy provides read-only access to CloudWatch Events.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchEventsReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "events:Describe*",
                "events:List*",
                "events:TestEventPattern"
            ],
            "Resource": "*"
        }
    ]
}
```

Using Resource-Based Policies for CloudWatch Events

When a rule is triggered in CloudWatch Events, all the targets associated with the rule are invoked. *Invocation* means invoking the AWS Lambda functions, publishing to the Amazon SNS topics, and relaying the event to the Amazon Kinesis streams. In order to be able to make API calls against the resources you own, CloudWatch Events needs the appropriate permissions. For Lambda, Amazon SNS, and Amazon SQS resources, CloudWatch Events relies on resource-based policies. For Amazon Kinesis streams, CloudWatch Events relies on IAM roles.

You can use the following permissions to invoke the targets associated with your CloudWatch Events rules. The procedures below use the AWS CLI to add permissions to your targets. 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*.

Topics

- AWS Lambda Permissions (p. 68)
- Amazon SNS Permissions (p. 69)
- Amazon SQS Permissions (p. 70)

AWS Lambda Permissions

To invoke your AWS Lambda function using a CloudWatch Events rule, add the following permission to the policy of your Lambda function.

```
{
  "Effect": "Allow",
  "Action": "lambda:InvokeFunction",
  "Resource": "arn:aws:lambda:region:account-id:function:function-name",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:events:region:account-id:rule/rule-name"
    }
  },
  "Sid": "TrustCWEToInvokeMyLambdaFunction"
}
```

To add permissions that enable CloudWatch Events to invoke Lambda functions

- At a command prompt, enter the following command:

```
aws lambda add-permission --statement-id "TrustCWEToInvokeMyLambdaFunction" \
--action "lambda:InvokeFunction" \
--principal "events.amazonaws.com" \
--function-name "arn:aws:lambda:region:account-id:function:function-name" \
--source-arn "arn:aws:events:region:account-id:rule/rule-name"
```

For more information about setting permissions that enable CloudWatch Events to invoke Lambda functions, see AddPermission and Using Lambda with Scheduled Events in the *AWS Lambda Developer Guide*.

Amazon SNS Permissions

To allow CloudWatch Events to publish an Amazon SNS topic, use the `aws sns get-topic-attributes` and the `aws sns set-topic-attributes` commands.

To add permissions that enable CloudWatch Events to publish SNS topics

1. First, list SNS topic attributes. At a command prompt, type the following:

```
aws sns get-topic-attributes --topic-arn "arn:aws:sns:region:account-id:topic-name"
```

The command returns all attributes of the SNS topic. The following example shows the result of a newly created SNS topic.

```
{
    "Attributes": {
        "SubscriptionsConfirmed": "0",
        "DisplayName": "",
        "SubscriptionsDeleted": "0",
        "EffectiveDeliveryPolicy": "{\"http\":{\"defaultHealthyRetryPolicy\":
{\"minDelayTarget\":20,\"maxDelayTarget\":20,\"numRetries\":3,\"numMaxDelayRetries
\":0,\"numNoDelayRetries\":0,\"numMinDelayRetries\":0,\"backoffFunction\":\"linear\"},
\"disableSubscriptionOverrides\":false}}",
        "Owner": "account-id",
        "Policy": "{\"Version\":\"2012-10-17\",\"Id\":\"__default_policy_ID\",
\"Statement\":[{\"Sid\":\"__default_statement_ID\",\"Effect\":\"Allow\",\"Principal
\":{\"AWS\":\"*\"},\"Action\":[\"SNS:GetTopicAttributes\",\"SNS:SetTopicAttributes
\",\"SNS:AddPermission\",\"SNS:RemovePermission\",\"SNS:DeleteTopic\",\"SNS:Subscribe
\",\"SNS:ListSubscriptionsByTopic\",\"SNS:Publish\",\"SNS:Receive\"],\"Resource
\":\"arn:aws:sns:region:account-id:topic-name\",\"Condition\":{\"StringEquals\":
{\"AWS:SourceOwner\":\"account-id\"}}}]}",
        "TopicArn": "arn:aws:sns:region:account-id:topic-name",
        "SubscriptionsPending": "0"
    }
}
```

2. Next, convert the following statement to a string and add it to the "Statement" collection inside the "Policy" attribute.

```
{
  "Sid": "TrustCWEToPublishEventsToMyTopic",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "sns:Publish",
  "Resource": "arn:aws:sns:region:account-id:topic-name"
}
```

   After you convert the statement to a string, it should look like the following:

```
{\"Sid\":\"TrustCWEToPublishEventsToMyTopic\",\"Effect\":\"Allow\",\"Principal\":
{\"Service\":\"events.amazonaws.com\"},\"Action\":\"sns:Publish\",\"Resource\":
\"arn:aws:sns:region:account-id:topic-name\"}
```

3. After you've added the statement string to the statement collection, use the `aws sns set-topic-attributes` command to set the new policy.

```
aws sns set-topic-attributes --topic-arn "arn:aws:sns:region:account-id:topic-name" \
--attribute-name Policy \
--attribute-value "{\"Version\":\"2012-10-17\",\"Id\":\"__default_policy_ID\",
\"Statement\":[{\"Sid\":\"__default_statement_ID\",\"Effect\":\"Allow\",\"Principal
\":{\"AWS\":\"*\"},\"Action\":[\"SNS:GetTopicAttributes\",\"SNS:SetTopicAttributes
\",\"SNS:AddPermission\",\"SNS:RemovePermission\",\"SNS:DeleteTopic\",\"SNS:Subscribe
\",\"SNS:ListSubscriptionsByTopic\",\"SNS:Publish\",\"SNS:Receive\"],\"Resource
\":\"arn:aws:sns:region:account-id:topic-name\",\"Condition\":{\"StringEquals\":
{\"AWS:SourceOwner\":\"account-id\"}}}, {\"Sid\":\"TrustCWEToPublishEventsToMyTopic\",
\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"events.amazonaws.com\"},\"Action\":
\"sns:Publish\",\"Resource\":\"arn:aws:sns:region:account-id:topic-name\"}]}"
```

For more information, see the SetTopicAttributes action in the *Amazon Simple Notification Service API Reference*.

Amazon SQS Permissions

To allow a CloudWatch Events rule to invoke an Amazon SQS queue, use the `aws sqs get-queue-attributes` and the `aws sqs set-queue-attributes` commands.

To add permissions that enable CloudWatch Events rules to invoke an SQS queue

1. First, list SQS queue attributes. At a command prompt, type the following:

```
aws sqs get-queue-attributes \
--queue-url https://sqs.region.amazonaws.com/account-id/queue-name \
--attribute-names Policy
```

   For a newly created SQS queue, its policy is empty by default. In addition to adding a statement, you also need to create a policy that contains this statement.

2. The following statement enables CloudWatch Events to send messages to an SQS queue:

```
{
  "Sid": "TrustCWEToSendEventsToMyQueue",
  "Effect": "Allow",
  "Principal": {
     "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "arn:aws:sqs:region:account-id:queue-name",
  "Condition": {
    "ArnEquals": {
      "aws:SourceArn": "arn:aws:events:region:account-id:rule/rule-name"
    }
  }
}
```

3. Next, convert the statement above into a string. After you convert the policy to a string, it should look like the following:

```
{\"Sid\": \"TrustCWEToSendEventsToMyQueue\", \"Effect\": \"Allow\", \"Principal
\": {\"AWS\": \"*\"}, \"Action\": \"sqs:SendMessage\", \"Resource\":
 \"arn:aws:sqs:region:account-id:queue-name\", \"Condition\": {\"ArnEquals\":
 {\"aws:SourceArn\": \"arn:aws:events:region:account-id:rule/rule-name\"}}
```

4. Create a file called **set-queue-attributes.json** with the following content:

```
{
    "Policy": "{\"Version\":\"2012-10-17\",\"Id\":\"arn:aws:sqs:region:account-
id:queue-name/SQSDefaultPolicy\",\"Statement\":[{\"Sid\":
 \"TrustCWEToSendEventsToMyQueue\", \"Effect\": \"Allow\", \"Principal\": {\"AWS\":
 \"*\"}, \"Action\": \"sqs:SendMessage\", \"Resource\": \"arn:aws:sqs:region:account-
id:queue-name\", \"Condition\": {\"ArnEquals\": {\"aws:SourceArn\":
 \"arn:aws:events:region:account-id:rule/rule-name\"}}}]}"
}
```

5. Set the policy attribute using the set-queue-attributes.json file as the input. At a command prompt, type:

```
aws sqs set-queue-attributes \
--queue-url https://sqs.region.amazonaws.com/account-id/queue-name \
--attributes file://set-queue-attributes.json
```

   If the SQS queue already has a policy, you need to copy the original policy and combine it with a new statement in the set-queue-attributes.json file and run the above command to update the policy.

For more information, see Amazon SQS Policy Examples in the *Amazon Simple Queue Service Developer Guide*.

CloudWatch Events Permissions Reference

When you are setting up Access Control (p. 57) 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 Events 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 Events policies to express conditions. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

Note
To specify an action, use the `events:` prefix followed by the API operation name. For example: `events:PutRule`, `events:EnableRule`, or `events:*` (for all CloudWatch Events actions).

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["events:action1", "events:action2"]
```

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Put" as follows:

```
"Action": "events:Put*"
```

To specify all CloudWatch Events API actions, use the * wildcard as follows:

```
"Action": "events:*"
```

The actions you can specify in an IAM policy for use with CloudWatch Events are listed below.

CloudWatch Events API Operations and Required Permissions for Actions

CloudWatch Events API Operations	Required Permissions (API Actions)
DeleteRule	`events:DeleteRule` Required to delete a rule.
DescribeRule	`events:DescribeRule` Required to list the details about a rule.
DisableRule	`events:DisableRule` Required to disable a rule.
EnableRule	`events:EnableRule` Required to enable a rule.
ListRuleNamesByTarget	`events:ListRuleNamesByTarget` Required to list rules associated with a target.
ListRules	`events:ListRules` Required to list all rules in your account.
ListTargetsByRule	`events:ListTargetsByRule` Required to list all targets associated with a rule.
PutEvents	`events:PutEvents` Required to add custom events that can be matched to rules.
PutRule	`events:PutRule` Required to create or update a rule.
PutTargets	`events:PutTargets` Required to add targets to a rule.

CloudWatch Events API Operations	Required Permissions (API Actions)
RemoveTargets	`events:RemoveTargets` Required to remove a target from a rule.
TestEventPattern	`events:TestEventPattern` Required to test an event pattern against a given event.

Using IAM Policy Conditions for Fine-Grained Access Control

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. In a policy statement, you can optionally specify conditions that control when it is in effect. Each condition contains one or more key-value pairs. Condition keys are not case-sensitive. For example, you might want a policy to be applied only after a specific date.

If you specify multiple conditions, or multiple keys in a single condition, they are evaluated using a logical AND operation. If you specify a single condition with multiple values for one key, they are evaluated using a logical OR operation. For permission to be granted, all conditions must be met.

You can also use placeholders when you specify conditions. For more information, see Policy Variables in the *IAM User Guide*. For more information about specifying conditions in an access policy language, see Condition in the *IAM User Guide*.

By default, IAM users and roles can't access the events in your account. To consume events, a user must be authorized for the `PutRule` API action. If you allow an IAM user or role for the `events:PutRule` action in their policy, then they will be able to create a rule that matches certain events. You must add a target to a rule, otherwise, a rule without a target does nothing except publish a CloudWatch metric when it matches an incoming event. Your IAM user or role must have permissions for the `events:PutTargets` action.

It is possible to limit access to the events by scoping the authorization to specific sources and types of events (using the `events:source` and `events:detail-type` condition keys). You can provide a condition in the policy statement of the IAM user or role that allows them to create a rule that only matches a specific set of sources and detail types. For a list showing all of condition key values and the CloudWatch Events actions and resources that they apply to, see Using IAM Policy Conditions for Fine-Grained Access Control (p. 73).

Similarly, through setting conditions in your policy statements, you can decide which specific resources in your accounts can be added to a rule by an IAM user or role (using the `events:TargetArn` condition key). For example, if you turn on CloudTrail in your account and you have a CloudTrail stream, CloudTrail events are also available to the users in your account through CloudWatch Events. If you want your users to use CloudWatch Events and access all the events but the CloudTrail events, you can add a deny statement on the `PutRule` API action with a condition that any rule created by that user or role cannot match the CloudTrail event type.

For CloudTrail events, you can limit the access to a specific principal that the original API call was originated from (using the `events:detail.userIdentity.principalId` condition key). For example, you can allow a user to see all the CloudTrail events, except the ones that are made by a specific IAM role in your account that you use for auditing or forensics.

Condition Key	Key/Value Pair	Evaluation Types
`events:source`	`"events:source":"`*`source`*` "`	Source, Null

Condition Key	Key/Value Pair	Evaluation Types
	Where *source* is the literal string for the source field of the event such as "aws.ec2" and "aws.s3".	
`events:detail-type`	`"events:detail-type":"detail-type "` Where *detail-type* is the literal string for the **detail-type** field of the event such as "AWS API Call via CloudTrail" and "EC2 Instance State-change Notification".	Detail Type, Null
`events:detail.userIdentity.principalId`	`"events:detail.userIdentity.principalId":"principal-id"` Where *principal-id* is the literal string for the **detail.userIdentity.principalId** field of the event with detail-type "AWS API Call via CloudTrail" such as `"AROAIDPPEZS35WEXAMPLE:AssumedRoleSessionName."`.	Principal Id, Null
`events:TargetArn`	`"events:TargetArn":"target-arn "` Where *target-arn* is the ARN of the target that can be put to a rule such as `"arn:aws:lambda:*:*:function:*"`.	ARN, Null

For example policy statements for CloudWatch Events, see Overview of Managing Access Permissions to Your CloudWatch Events Resources (p. 58).

Topics

- Example 1: Limit Access to a Specific Source (p. 74)
- Example 2: Define Multiple Sources That Can Be Used in an Event Pattern Individually (p. 76)
- Example 3: Define a Source and a DetailType That Can Be Used in an Event Pattern (p. 77)
- Example 4: Ensure That the Source Is Defined in the Event Pattern (p. 78)
- Example 5: Define a List of Allowed Sources in an Event Pattern with Multiple Sources (p. 79)
- Example 6: Ensure That AWS CloudTrail Events for API Calls from a Certain PrincipalId Are Consumed (p. 80)
- Example 7: Limiting Access to Targets (p. 81)

Example 1: Limit Access to a Specific Source

The following example policies can be attached to an IAM user. Policy A allows the `PutRule` API action for all events, whereas Policy B allows `PutRule` only if the event pattern of the rule being created matches Amazon EC2 events.

Policy A:—allow any events

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleForAllEvents",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*"
        }
    ]
    }
```

Policy B:—allow events only from Amazon EC2

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleForAllEC2Events",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "events:source": "aws.ec2"
                }
            }
        }
    ]
}
```

EventPattern is a mandatory argument to `PutRule`. Hence, if the user with Policy B calls `PutRule` with an event pattern like the following:

```
{
    "source": [ "aws.ec2" ]
}
```

The rule would be created because the policy allows for this specific source, that is, "aws.ec2". However, if the user with Policy B calls `PutRule`with an event pattern like the following:

```
{
    "source": [ "aws.s3" ]
}
```

The rule creation would be denied because the policy does not allow for this specific source, that is, "aws.s3". Essentially, the user with Policy B is only allowed to create a rule that would match the events originating from Amazon EC2; hence, they are only allowed access to the events from Amazon EC2.

See the following table for a comparison of Policy A and Policy B:

Event Pattern	Allowed by Policy A	Allowed by Policy B
`{ "source": [ "aws.ec2" ] }`	Yes	Yes
`{`	Yes	No (Source aws.s3 is not allowed)

Event Pattern	Allowed by Policy A	Allowed by Policy B
`"source": [ "aws.ec2", "aws.s3" ] }`		
`{ "source": [ "aws.ec2" ], "detail-type": [ "EC2 Instance State-change Notification" ] }`	Yes	Yes
`{ "detail-type": [ "EC2 Instance State-change Notification" ] }`	Yes	No (Source must be specified)

Example 2: Define Multiple Sources That Can Be Used in an Event Pattern Individually

The following policy allows events from Amazon EC2 or CloudWatch Events. In other words, it allows an IAM user or role to create a rule where the source in the EventPattern is specified as either "aws.ec2" or "aws.ecs". Not defining the source results in a "deny".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleIfSourceIsEC2OrECS",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "events:source": [ "aws.ec2", "aws.ecs" ]
                }
            }
        }
    ]
}
```

See the following table for examples of event patterns that would be allowed or denied by this policy:

Event Pattern	Allowed by the Policy
`{ "source": [ "aws.ec2" ] }`	Yes
`{ "source": [ "aws.ecs" ]`	Yes

Event Pattern	Allowed by the Policy
`}`	
`{ "source": [ "aws.s3" ] }`	No
`{ "source": [ "aws.ec2", "aws.ecs" ] }`	No
`{ "detail-type": [ "AWS API Call via CloudTrail" ] }`	No

Example 3: Define a Source and a DetailType That Can Be Used in an Event Pattern

The following policy allows events only from the `aws.ec2` source with DetailType equal to `EC2 instance state change notification`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid":
 "AllowPutRuleIfSourceIsEC2AndDetailTypeIsInstanceStateChangeNotification",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "events:source": "aws.ec2",
                    "events:detail-type": "EC2 Instance State-change Notification"
                }
            }
        }
    ]
}
```

See the following table for examples of event patterns that would be allowed or denied by this policy:

Event Pattern	Allowed by the Policy
`{ "source": [ "aws.ec2" ] }`	No
`{ "source": [ "aws.ecs" ]`	No

Event Pattern	Allowed by the Policy
`}`	
`{ "source": [ "aws.ec2" ], "detail-type": [ "EC2 Instance State-change Notification" ] }`	Yes
`{ "source": [ "aws.ec2" ], "detail-type": [ "EC2 Instance Health Failed" ] }`	No
`{ "detail-type": [ "EC2 Instance State-change Notification" ] }`	No

Example 4: Ensure That the Source Is Defined in the Event Pattern

The following policy allows creating rules with `EventPatterns` that must have the source field. In other words, an IAM user or role can't create a rule with an `EventPattern` that does not provide a specific source.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleIfSourceIsSpecified",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "Null": {
                    "events:source": "false"
                }
            }
        }
    ]
}
```

See the following table for examples of event patterns that would be allowed or denied by this policy:

Event Pattern	Allowed by the Policy
`{ "source": [ "aws.ec2" ], "detail-type": [ "EC2 Instance State-change Notification" ] }`	Yes

Event Pattern	Allowed by the Policy
`{ "source": [ "aws.ecs", "aws.ec2" ] }`	Yes
`{ "detail-type": [ "EC2 Instance State-change Notification" ] }`	No

Example 5: Define a List of Allowed Sources in an Event Pattern with Multiple Sources

The following policy allows creating rules with `EventPatterns` that can have multiple sources in them. Each source listed in the event pattern must be a member of the list provided in the condition. When using the ForAllValues condition, make sure that at least one of the items in the condition list is defined.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleIfSourceIsSpecifiedAndIsEitherS3OrEC2OrBoth",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "events:source": [ "aws.ec2", "aws.s3" ]
                },
                "Null": {
                    "events:source": "false"
                }
            }
        }
    ]
}
```

See the following table for examples of event patterns that would be allowed or denied by this policy:

Event Pattern	Allowed by the Policy
`{ "source": [ "aws.ec2" ] }`	Yes
`{ "source": [ "aws.ec2", "aws.s3" ] }`	Yes
`{ "source": [ "aws.ec2", "aws.autoscaling" ]`	No

Event Pattern	Allowed by the Policy
`}`	
`{ "detail-type": [ "EC2 Instance State-change Notification" ] }`	No

Example 6: Ensure That AWS CloudTrail Events for API Calls from a Certain PrincipalId Are Consumed

All AWS CloudTrail events have the ID of the user who made the API call (PrincipalId) in the `detail.userIdentity.principalId` path of an event. With the help of the `events:detail.userIdentity.principalId` condition key, you can limit the access of IAM users or roles to the CloudTrail events for only those coming from a specific account.

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleOnlyForCloudTrailEventsWhereUserIsASpecificIAMUser",
            "Effect": "Allow",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "events:detail-type": [ "AWS API Call via CloudTrail" ],
                    "events:detail.userIdentity.principalId": [ "AIDAJ45Q7YFFAREXAMPLE" ]
                }
            }
        }
    ]
}
```

See the following table for examples of event patterns that would be allowed or denied by this policy:

Event Pattern	Allowed by the Policy
`{ "detail-type": [ "AWS API Call via CloudTrail" ] }`	No
`{ "detail-type": [ "AWS API Call via CloudTrail" ], "detail.userIdentity.principalId": [ "AIDAJ45Q7YFFAREXAMPLE" ] }`	Yes
`{ "detail-type": [ "AWS API Call via CloudTrail" ],`	No

Event Pattern	Allowed by the Policy

`"detail.userIdentity.principalId":`
`[ "AROAIDPPEZS35WEXAMPLE:AssumedRoleSessionName" ]`
`}`	

Example 7: Limiting Access to Targets

If an IAM user or role has `events:PutTargets` permission, they can add any target under the same account to the rules that they are allowed to access. For example, the following policy limits adding targets to only a specific rule (MyRule under account 123456789012).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutTargetsOnASpecificRule",
            "Effect": "Allow",
            "Action": "events:PutTargets",
            "Resource": "arn:aws:events:us-east-1:123456789012:rule/MyRule"
        }
    ]
}
```

To limit what target can be added to the rule, use the `events:TargetArn` condition key. For example, you can limit targets to only Lambda functions, as in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutTargetsOnASpecificRuleAndOnlyLambdaFunctions",
            "Effect": "Allow",
            "Action": "events:PutTargets",
            "Resource": "arn:aws:events:us-east-1:123456789012:rule/MyRule",
            "Condition": {
                "ArnLike": {
                    "events:TargetArn": "arn:aws:lambda:*:*:function:*"
                }
            }
        }
    ]
}
```

Logging Amazon CloudWatch Events API Calls in AWS CloudTrail

AWS CloudTrail is a service that captures API calls made by or on behalf of your AWS account. This information is collected and written to log files that are stored in an Amazon S3 bucket that you specify. API calls are logged whenever you use the API, the console, or the AWS CLI. Using the information collected by CloudTrail, you can determine what request was made, the source IP address the request was made from, who made the request, when it was made, and so on.

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

Topics

- CloudWatch Events Information in CloudTrail (p. 82)
- Understanding Log File Entries (p. 83)

CloudWatch Events Information in CloudTrail

If CloudTrail logging is turned on, calls made to API actions are captured in log files. Every log file entry contains information about who generated the request. For example, if a request is made to create a CloudWatch Events rule (`PutRule`), CloudTrail logs the user identity of the person or service that made the request.

The user identity information in the log entry helps you determine the following:

- Whether the request was made with root or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity Element in the *AWS CloudTrail User Guide*.

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

If you want to be notified upon log file delivery, you can configure CloudTrail to publish Amazon SNS notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications for CloudTrail in the *AWS CloudTrail User Guide*.

You can also aggregate Amazon CloudWatch Logs log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts in the *AWS CloudTrail User Guide*.

When logging is turned on, the following API actions are written to CloudTrail:

- DeleteRule
- DescribeRule
- DisableRule
- EnableRule
- ListRuleNamesByTarget
- ListRules
- ListTargetsByRule
- PutRule
- PutTargets
- RemoveTargets
- TestEventPattern

For more information about these actions, see the Amazon CloudWatch Events API Reference.

Understanding Log File Entries

CloudTrail log files contain one or more log entries. Each entry lists multiple JSON-formatted events. A log entry 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. The log entries are not an ordered stack trace of the public API calls, so they do not appear in any specific order. Log file entries for all API actions are similar to the examples below.

The following log file entry shows that a user called the CloudWatch Events **PutRule** action.

```
{
         "eventVersion":"1.03",
         "userIdentity":{
            "type":"Root",
            "principalId":"123456789012",
            "arn":"arn:aws:iam::123456789012:root",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext":{
               "attributes":{
                  "mfaAuthenticated":"false",
                  "creationDate":"2015-11-17T23:56:15Z"
               }
            }
         },
         "eventTime":"2015-11-18T00:11:28Z",
         "eventSource":"events.amazonaws.com",
         "eventName":"PutRule",
         "awsRegion":"us-east-1",
         "sourceIPAddress":"AWS Internal",
         "userAgent":"AWS CloudWatch Console",
         "requestParameters":{
            "description":"",
            "name":"cttest2",
            "state":"ENABLED",
            "eventPattern":"{\"source\":[\"aws.ec2\"],\"detail-type\":[\"EC2 Instance
 State-change Notification\"]}",
            "scheduleExpression":""
         },
         "responseElements":{
            "ruleArn":"arn:aws:events:us-east-1:123456789012:rule/cttest2"
         },
         "requestID":"e9caf887-8d88-11e5-a331-3332aa445952",
         "eventID":"49d14f36-6450-44a5-a501-b0fdcdfaeb98",
         "eventType":"AwsApiCall",
         "apiVersion":"2015-10-07",
         "recipientAccountId":"123456789012"
}
```

Troubleshooting CloudWatch Events

You can use the steps in this section to troubleshoot CloudWatch Events.

Topics

- My rule was triggered but my Lambda function was not invoked (p. 85)
- I have just created/modified a rule but it did not match a test event (p. 86)
- My rule did not self-trigger at the time specified in the ScheduleExpression (p. 87)
- My rule did not trigger at the time that I expected (p. 87)
- My rule matches IAM API calls but my rule was not triggered (p. 87)
- My rule is not working because the IAM role associated with the rule is ignored when the rule is triggered (p. 87)
- I created a rule with an EventPattern that is supposed to match a resource, but I don't see any events that match the rule (p. 88)
- My event's delivery to the target experienced a delay (p. 88)
- My rule was triggered more than once in response two identical events. What guarantee does CloudWatch Events offer for triggering rules or delivering events to the targets? (p. 88)
- My rule is being triggered but I don't see any messages published into my Amazon SNS topic (p. 88)
- My Amazon SNS topic still has permissions for CloudWatch Events even after I deleted the rule associated with the Amazon SNS topic (p. 90)
- Which IAM condition keys can I use with CloudWatch Events (p. 90)
- How can I tell when CloudWatch Events rules are broken (p. 90)

My rule was triggered but my Lambda function was not invoked

Make sure you have the right permissions set for your Lambda function. Run the following command using AWS CLI (replace the function name with your function and use the AWS region your function is in):

```
aws lambda get-policy --function-name MyFunction --region us-east-1
```

You should see an output similar to the following:

```
{
    "Policy": "{\"Version\":\"2012-10-17\",
    \"Statement\":[
        {\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:events:us-
east-1:123456789012:rule/MyRule\"}},
        \"Action\":\"lambda:InvokeFunction\",
        \"Resource\":\"arn:aws:lambda:us-east-1:123456789012:function:MyFunction\",
        \"Effect\":\"Allow\",
        \"Principal\":{\"Service\":\"events.amazonaws.com\"},
        \"Sid\":\"MyId\"}
    ],
    \"Id\":\"default\"}"
}
```

If you see the following:

```
A client error (ResourceNotFoundException) occurred when calling the GetPolicy operation:
 The resource you requested does not exist.
```

Or, you see the output but you can't locate events.amazonaws.com as a trusted entity in the policy, run the following command:

```
aws lambda add-permission \
--function-name MyFunction \
--statement-id MyId \
--action 'lambda:InvokeFunction' \
--principal events.amazonaws.com \
--source-arn arn:aws:events:us-east-1:123456789012:rule/MyRule
```

Note

If the policy is incorrect, you can also edit the rule in the CloudWatch Events console by removing and then adding it back to the rule. The CloudWatch Events console will set the correct permissions on the target.

If you're using a specific Lambda alias or version, you must add the `--qualifier` parameter in the `aws lambda get-policy` and `aws lambda add-permission` commands.

```
aws lambda add-permission \
--function-name MyFunction \
--statement-id MyId \
--action 'lambda:InvokeFunction' \
--principal events.amazonaws.com \
--source-arn arn:aws:events:us-east-1:123456789012:rule/MyRule
--qualifier alias or version
```

I have just created/modified a rule but it did not match a test event

When you make a change to a rule or to its targets, incoming events might not immediately start or stop matching to new or updated rules. Please allow a short period of time for changes to take effect. If, after this short period, events still do not match, you can also check several Events metrics for your rule in CloudWatch such as `TriggeredRules`, `Invocations`, and `FailedInvocations` for further debugging.

You can also use the `TestEventPattern` action to test the event pattern of your rule with a test event to make sure the event pattern of your rule is correctly set. For more information, see TestEventPattern in the *Amazon CloudWatch Events API Reference*.

My rule did not self-trigger at the time specified in the ScheduleExpression

ScheduleExpressions are in UTC. Make sure you have set the schedule for rule to self-trigger in the UTC timezone. If the ScheduleExpression is correct, then follow the steps under I have just created/modified a rule but it did not match a test event (p. 86).

My rule did not trigger at the time that I expected

CloudWatch Events doesn't support setting an exact start time when you create a rule to run every time period. The count down to run time begins as soon as you create the rule.

You can use a cron expression to invoke targets at a specified time. For example, you can use a cron expression to create a rule that is triggered every 4 hours exactly on 0 minute. In the CloudWatch console, you'd use the cron expression `0 0/4 * * ? *`, and with the AWS CLI you'd use the cron expression `cron(0 0/4 * * ? *)`. For example, to create a rule named TestRule that is triggered every 4 hours using the AWS CLI, you would type the following at a command prompt:

```
aws events put-rule --name TestRule --schedule-expression 'cron(0 0/4 * * ? *)'
```

You can use the `0/5 * * * ? *` cron expression to trigger a rule every 5 minutes. For example:

```
aws events put-rule --name TestRule --schedule-expression 'cron(0/5 * * * ? *)'
```

CloudWatch Events does not provide second-level precision in schedule expressions. The finest resolution using a cron expression is a minute. Due to the distributed nature of the CloudWatch Events and the target services, the delay between the time the scheduled rule is triggered and the time the target service honors the execution of the target resource might be several seconds. Your scheduled rule will be triggered within that minute but not on the precise 0th second.

My rule matches IAM API calls but my rule was not triggered

The IAM service is only available in the US East (N. Virginia) Region, so any AWS API call events from IAM are only available in that region. For more information, see Event Types for CloudWatch Events (p. 35).

My rule is not working because the IAM role associated with the rule is ignored when the rule is triggered

IAM roles for rules are only used for relating events to Amazon Kinesis streams. For Lambda functions and Amazon SNS topics, you need to provide resource-based permissions.

Make sure your regional AWS STS endpoints are enabled. CloudWatch Events talks to the regional AWS STS endpoints when assuming the IAM role you provided. For more information, see Activating and Deactivating AWS STS in an AWS Region in the *IAM User Guide*.

I created a rule with an EventPattern that is supposed to match a resource, but I don't see any events that match the rule

Most services in AWS treat : or / as the same character in Amazon Resource Names (ARNs). However, CloudWatch Events uses an exact match in event patterns and rules. Be sure to use the correct ARN characters when creating event patterns so that they match the ARN syntax in the event you want to match.

Moreover, not every event has the resources field populated (e.g. AWS API Call events from CloudTrail).

My event's delivery to the target experienced a delay

Amazon CloudWatch Events tries to deliver an event to a target for up to 24 hours. The first attempt is made as soon as the event arrives in the event stream. However, if the target service is having problems or your account is being throttled, CloudWatch Events automatically reschedules another delivery in the future. If 24 hours has passed since the arrival of event, no more attempts are scheduled and the `FailedInvocations` metric is published in Amazon CloudWatch.

My rule was triggered more than once in response two identical events. What guarantee does CloudWatch Events offer for triggering rules or delivering events to the targets?

Amazon CloudWatch Events guarantees triggering a rule at least once in response to an event. In rare cases, the same rule can be triggered more than once for a given event, or the same target can be invoked more than once for a given triggered rule.

My rule is being triggered but I don't see any messages published into my Amazon SNS topic

Make sure you have the right permission set for your Amazon SNS topic. Run the following command using AWS CLI (replace the topic ARN with your topic and use the AWS region your topic is in):

```
aws sns get-topic-attributes --region us-east-1 --topic-arn "arn:aws:sns:us-
east-1:123456789012:MyTopic"
```

You should see policy attribute similar to the following:

```
"{\"Version\":\"2012-10-17\",
\"Id\":\"__default_policy_ID\",
\"Statement\":[{\"Sid\":\"__default_statement_ID\",
\"Effect\":\"Allow\",
\"Principal\":{\"AWS\":\"*\"},
\"Action\":[\"SNS:Subscribe\",
\"SNS:ListSubscriptionsByTopic\",
\"SNS:DeleteTopic\",
\"SNS:GetTopicAttributes\",
\"SNS:Publish\",
\"SNS:RemovePermission\",
\"SNS:AddPermission\",
\"SNS:Receive\",
\"SNS:SetTopicAttributes\"],
\"Resource\":\"arn:aws:sns:us-east-1:123456789012:MyTopic\",
\"Condition\":{\"StringEquals\":{\"AWS:SourceOwner\":\"123456789012\"}}},{\"Sid\":
\"Allow_Publish_Events\",
\"Effect\":\"Allow\",
\"Principal\":{\"Service\":\"events.amazonaws.com\"},
\"Action\":\"sns:Publish\",
\"Resource\":\"arn:aws:sns:us-east-1:123456789012:MyTopic\"}]}"
```

If you see a policy similar to the following, you have only the default policy set:

```
"{\"Version\":\"2008-10-17\",
\"Id\":\"__default_policy_ID\",
\"Statement\":[{\"Sid\":\"__default_statement_ID\",
\"Effect\":\"Allow\",
\"Principal\":{\"AWS\":\"*\"},
\"Action\":[\"SNS:Subscribe\",
\"SNS:ListSubscriptionsByTopic\",
\"SNS:DeleteTopic\",
\"SNS:GetTopicAttributes\",
\"SNS:Publish\",
\"SNS:RemovePermission\",
\"SNS:AddPermission\",
\"SNS:Receive\",
\"SNS:SetTopicAttributes\"],
\"Resource\":\"arn:aws:sns:us-east-1:123456789012:MyTopic\",
\"Condition\":{\"StringEquals\":{\"AWS:SourceOwner\":\"123456789012\"}}}]}"
```

If you don't see `events.amazonaws.com` with Publish permission in your policy, use AWS CLI to set topic policy attribute.

Copy current policy and add statement below to list of statements:

```
{\"Sid\":\"Allow_Publish_Events\",
\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"events.amazonaws.com\"},
\"Action\":\"sns:Publish\",
\"Resource\":\"arn:aws:sns:us-east-1:123456789012:MyTopic\"}
```

The new policy should look like the one described above.

Set topic attributes with the AWS CLI:

```
aws sns set-topic-attributes --region us-east-1 --topic-arn "arn:aws:sns:us-
east-1:123456789012:MyTopic" --attribute-name Policy --attribute-value NEW_POLICY_STRING
```

Note
If the policy is incorrect, you can also edit the rule in the CloudWatch Events console by removing and then adding it back to the rule. The CloudWatch Events console will set the correct permissions on the target.

My Amazon SNS topic still has permissions for CloudWatch Events even after I deleted the rule associated with the Amazon SNS topic

When you create a rule with Amazon SNS as the target, CloudWatch Events adds the permission to your Amazon SNS topic on your behalf. If you delete the rule shortly after you create it, CloudWatch Events might be unable to remove the permission from your Amazon SNS topic. If this happens, you can remove the permission from the topic using the aws sns set-topic-attributes command. For more information about resource-based permissions for sending events, see Using Resource-Based Policies for CloudWatch Events (p. 68).

Which IAM condition keys can I use with CloudWatch Events

Amazon CloudWatch Events supports the AWS-wide condition keys (see Available Keys in the *IAM User Guide*), plus the following service-specific condition keys. For more information, see Using IAM Policy Conditions for Fine-Grained Access Control (p. 73).

How can I tell when CloudWatch Events rules are broken

You can use the following alarm to notify you when your CloudWatch Events rules are broken.

To create an alarm to alert when rules are broken

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Click **Create Alarm**, and then in the **CloudWatch Metrics by Category** pane, select **Events Metrics**.
3. On the **Create Alarm** dialog box, in the list of metrics, select the **FailedInvocations** check box.
4. Above the graph, select **Sum** from the **Statistic** drop-down list.
5. Select a period from the **Period** drop-down list, for example: **5 minutes**.
6. Click **Next**, and then under **Alarm Threshold**, in the **Name** field, enter a unique name for the alarm, for example: **myFailedRules**.
7. In the **Description** field, enter a description of the alarm, for example: **Rules are not delivering events to targets**.
8. In the **is** drop-down list, select **>=**.
9. In the field next to the **is** drop-down list, enter **1** and in the **for** field, enter **10**.
10. Under **Actions**, in the **Whenever this alarm** drop-down list, select **State is ALARM**.
11. In the **Send notification to** drop-down list, select an existing Amazon SNS topic or create a new one.
12. To create a new Amazon SNS topic, select **New list**.
13. In the **Send notification to** field, enter a name for the new Amazon SNS topic for example: **myFailedRules**, and in the **Email list** field, enter a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state.
14. In the navigation pane, choose **Create Alarm** to complete the alarm creation process.

Document History

The following table describes the important changes to the *Amazon CloudWatch Events User Guide*.

Change	Description	Release Date
Additional targets supported	You can now set two additional AWS services as targets for event actions: Amazon EC2 instances (via Run Command), and Step Functions state machines. For more information, see Getting Started with Amazon CloudWatch Events (p. 7).	7 March 2017
Amazon EMR events	Added support for events for Amazon EMR. For more information, see Amazon EMR Events (p. 41).	7 March 2017
AWS Health events	Added support for events for AWS Health. For more information, see AWS Health Events (p. 50).	1 December 2016
Amazon EC2 Container Service events	Added support for events for Amazon ECS. For more information, see Amazon ECS Events (p. 40).	21 November 2016
AWS Trusted Advisor events	Added support for events for Trusted Advisor. For more information, see Trusted Advisor Events (p. 53).	18 November 2016
Amazon Elastic Block Store events	Added support for events for Amazon EBS. For more information, see Amazon EBS Events (p. 35).	14 November 2016
AWS CodeDeploy events	Added support for events for AWS CodeDeploy. For more information, see AWS CodeDeploy Events (p. 49).	9 September 2016
Scheduled events with 1 minute granularity	Added support for scheduled events with 1 minute granularity. For more information, see Cron Expressions (p. 24) and Rate Expressions (p. 26).	19 April 2016
Amazon Simple Queue Service queues as targets	Added support for Amazon SQS queues as targets. For more information, see What is Amazon CloudWatch Events? (p. 1).	30 March 2016
Auto Scaling events	Added support for events for Auto Scaling lifecycle hooks. For more information, see Auto Scaling Events (p. 43).	24 February 2016

Change	Description	Release Date
New service	Initial release of CloudWatch Events.	14 January 2016

AWS Glossary

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