
Amazon CloudWatch Events

User Guide



Amazon CloudWatch Events: User Guide

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

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

Table of Contents

| | |
|---|----|
| What Is Amazon CloudWatch Events? | 1 |
| Concepts | 1 |
| Related AWS Services | 2 |
| Setting Up | 3 |
| Sign Up for Amazon Web Services (AWS) | 3 |
| Sign in to the Amazon CloudWatch Console | 3 |
| Account Credentials | 3 |
| Set Up the Command Line Interface | 4 |
| Regional Endpoints | 4 |
| Getting Started | 5 |
| Creating a Rule That Triggers on an Event | 6 |
| Creating a Rule That Triggers on an AWS API Call via CloudTrail | 7 |
| Creating a Rule That Triggers on a Schedule | 8 |
| Deleting or Disabling a Rule | 8 |
| Tutorials | 10 |
| Tutorial: Relay Events to Systems Manager Run Command | 10 |
| Tutorial: Log EC2 Instance States | 11 |
| Step 1: Create an AWS Lambda Function | 11 |
| Step 2: Create a Rule | 12 |
| Step 3: Test the Rule | 12 |
| Tutorial: Log Auto Scaling Group States | 13 |
| Step 1: Create an AWS Lambda Function | 13 |
| Step 2: Create a Rule | 14 |
| Step 3: Test the Rule | 14 |
| Tutorial: Log S3 Object Level Operations | 15 |
| Step 1: Configure Your AWS CloudTrail Trail | 15 |
| Step 2: Create an AWS Lambda Function | 16 |
| Step 3: Create a Rule | 16 |
| Step 4: Test the Rule | 17 |
| Tutorial: Use Input Transformer to Customize What is Passed to the Event Target | 17 |
| Create a Rule | 18 |
| Tutorial: Log AWS API Calls | 18 |
| Prerequisite | 18 |
| Step 1: Create an AWS Lambda Function | 19 |
| Step 2: Create a Rule | 19 |
| Step 3: Test the Rule | 20 |
| Tutorial: Schedule Automated EBS Snapshots | 20 |
| Step 1: Create a Rule | 21 |
| Step 2: Test the Rule | 21 |
| Tutorial: Schedule Lambda Functions | 21 |
| Step 1: Create an AWS Lambda Function | 22 |
| Step 2: Create a Rule | 22 |
| Step 3: Verify the Rule | 24 |
| Tutorial: Set Systems Manager Automation as a Target | 24 |
| Tutorial: Relay Events to a Kinesis Stream | 25 |
| Prerequisite | 25 |
| Step 1: Create an Amazon Kinesis Stream | 25 |
| Step 2: Create a Rule | 25 |
| Step 3: Test the Rule | 26 |
| Step 4: Verify That the Event is Relayed | 26 |
| Tutorial: Run an Amazon ECS Task When a File is Uploaded to an Amazon S3 Bucket | 27 |
| Tutorial: Schedule Automated Builds Using CodeBuild | 28 |
| Tutorial: Log State Changes of Amazon EC2 Instances | 29 |
| Schedule Expressions for Rules | 30 |

| | |
|---|----|
| Cron Expressions | 30 |
| Rate Expressions | 32 |
| Event Patterns | 34 |
| Event Patterns | 35 |
| Matching Null Values and Empty Strings In Event Patterns | 36 |
| Arrays In Event Patterns | 37 |
| Events From Supported Services | 39 |
| Amazon Augmented AI Events | 40 |
| Application Auto Scaling Events | 40 |
| AWS Batch Events | 40 |
| Amazon CloudWatch Events Scheduled Events | 40 |
| Amazon Chime Events | 41 |
| Events from CloudWatch | 41 |
| CodeBuild Events | 41 |
| CodeCommit Events | 41 |
| AWS CodeDeploy Events | 41 |
| CodePipeline Events | 42 |
| AWS Config Events | 43 |
| Amazon EBS Events | 43 |
| Amazon EC2 Auto Scaling Events | 44 |
| Amazon EC2 Instance Rebalance Recommendation Events | 44 |
| Amazon EC2 Spot Instance Interruption Events | 44 |
| Amazon EC2 State Change Events | 44 |
| Amazon ECR Events | 44 |
| Amazon ECS Events | 45 |
| AWS Elemental MediaConvert Events | 45 |
| AWS Elemental MediaPackage Events | 45 |
| AWS Elemental MediaStore Events | 45 |
| Amazon EMR Events | 45 |
| Amazon GameLift Event | 47 |
| AWS Glue Events | 54 |
| AWS Ground Station Events | 59 |
| Amazon GuardDuty Events | 59 |
| AWS Health Events | 59 |
| AWS KMS Events | 61 |
| Amazon Macie Events | 62 |
| AWS Management Console Sign-in Events | 62 |
| AWS OpsWorks Stacks Events | 63 |
| SageMaker Events | 65 |
| AWS Security Hub Events | 65 |
| AWS Server Migration Service Events | 65 |
| AWS Systems Manager Events | 66 |
| AWS Systems Manager Automation Events | 67 |
| AWS Systems Manager Change Calendar Events | 67 |
| AWS Systems Manager Compliance Events | 68 |
| AWS Systems Manager Maintenance Windows Events | 70 |
| AWS Systems Manager Parameter Store Events | 72 |
| AWS Systems Manager Run Command Events | 73 |
| AWS Systems Manager State Manager Events | 74 |
| AWS Step Functions Events | 75 |
| Tag Change Events on AWS Resources | 75 |
| AWS Trusted Advisor Events | 75 |
| WorkSpaces Events | 77 |
| Events Delivered Via CloudTrail | 77 |
| Sending and Receiving Events Between AWS Accounts | 79 |
| Enabling Your AWS Account to Receive Events from Other AWS Accounts | 79 |
| Sending Events to Another AWS Account | 81 |

| | |
|--|-----|
| Writing Rules that Match Events from Another AWS Account | 82 |
| Migrate a Sender-Receiver Relationship to Use AWS Organizations | 84 |
| Adding Events with PutEvents | 86 |
| Handling Failures When Using PutEvents | 86 |
| Sending Events Using the AWS CLI | 88 |
| Calculating PutEvents Event Entry Sizes | 88 |
| Using CloudWatch Events with Interface VPC Endpoints | 90 |
| Availability | 90 |
| Creating a VPC Endpoint for CloudWatch Events | 91 |
| Controlling Access to Your CloudWatch Events VPC Endpoint | 91 |
| Monitoring Usage with CloudWatch Metrics | 93 |
| CloudWatch Events Metrics | 93 |
| Dimensions for CloudWatch Events Metrics | 93 |
| Managed Rules | 95 |
| Working with AWS SDKs | 96 |
| Code examples | 97 |
| Actions | 97 |
| Adding a Lambda function target | 97 |
| Create a scheduled rule | 99 |
| Send events | 101 |
| Security | 104 |
| Tagging Your CloudWatch Events Resources | 105 |
| Supported Resources in CloudWatch Events | 105 |
| Managing Tags | 105 |
| Tag Naming and Usage Conventions | 106 |
| Logging API Calls | 107 |
| CloudWatch Events Information in CloudTrail | 107 |
| Example: CloudWatch Events Log File Entries | 108 |
| Service Quotas | 110 |
| Troubleshooting | 111 |
| My rule was triggered but my Lambda function was not invoked | 111 |
| I have just created/modified a rule but it did not match a test event | 112 |
| My rule did not self-trigger at the time specified in the ScheduleExpression | 113 |
| My rule did not trigger at the time that I expected | 113 |
| My rule matches IAM API calls but my rule was not triggered | 113 |
| My rule is not working because the IAM role associated with the rule is ignored when the rule is triggered | 114 |
| 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 | 114 |
| My event's delivery to the target experienced a delay | 114 |
| Some events were never delivered to my target | 114 |
| My rule was triggered more than once in response to one event. What guarantee does CloudWatch Events offer for triggering rules or delivering events to the targets? | 115 |
| Preventing Infinite Loops | 115 |
| My events are not delivered to the target Amazon SQS queue | 115 |
| My rule is being triggered but I don't see any messages published into my Amazon SNS topic | 115 |
| My Amazon SNS topic still has permissions for CloudWatch Events even after I deleted the rule associated with the Amazon SNS topic | 117 |
| Which IAM condition keys can I use with CloudWatch Events | 117 |
| How can I tell when CloudWatch Events rules are broken | 117 |
| Document History | 119 |
| AWS glossary | 121 |

What Is Amazon CloudWatch Events?

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. 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.

You can also use CloudWatch Events to schedule automated actions that self-trigger at certain times using cron or rate expressions. For more information, see [Schedule Expressions for Rules \(p. 30\)](#).

You can configure the following AWS services as targets for CloudWatch Events:

- Amazon EC2 instances
- AWS Lambda functions
- Streams in Amazon Kinesis Data Streams
- Delivery streams in Amazon Kinesis Data Firehose
- Log groups in Amazon CloudWatch Logs
- Amazon ECS tasks
- Systems Manager Run Command
- Systems Manager Automation
- AWS Batch jobs
- Step Functions state machines
- Pipelines in CodePipeline
- CodeBuild projects
- Amazon Inspector assessment templates
- Amazon SNS topics
- Amazon SQS queues
- Built-in targets: EC2 `CreateSnapshot` API call, EC2 `RebootInstances` API call, EC2 `StopInstances` API call, and EC2 `TerminateInstances` API call.
- The default event bus of another AWS account

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 Amazon EC2 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 [CloudWatch Events Event Examples From Supported Services](#) (p. 39).

- **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.
- **Targets** – A target processes events. Targets can include Amazon EC2 instances, AWS Lambda functions, 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.

A rule's targets must be in the same Region as the rule.

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 with AWS CloudTrail](#) (p. 107).
- **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 record configuration changes to your AWS resources. This includes how resources relate to one another and how they were configured in the past, so that you can see how the configurations and relationships change over time. You can also create AWS Config rules to check whether your resources are compliant or noncompliant with your organization's policies. For more information, see the [AWS Config Developer Guide](#).
- **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 [IAM User Guide](#).
- **Amazon Kinesis Data Streams** enables rapid and nearly 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 Data 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](#).

Setting Up Amazon CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

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

Sign in to the Amazon CloudWatch Console

To sign in to the Amazon CloudWatch console

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

Account Credentials

Although you can use your root user 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": "*"
    }
  ]
}
```

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

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Use the procedures in this section to create and delete CloudWatch Events rules. These are general procedures usable for any event source or target. For tutorials written for specific scenarios and specific targets, see [Tutorials](#).

Each rule

Contents

- [Creating a CloudWatch Events Rule That Triggers on an Event \(p. 6\)](#)
- [Creating a CloudWatch Events Rule That Triggers on an AWS API Call Using AWS CloudTrail \(p. 7\)](#)
- [Creating a CloudWatch Events Rule That Triggers on a Schedule \(p. 8\)](#)
- [Deleting or Disabling a CloudWatch Events Rule \(p. 8\)](#)

Restrictions

- The targets you associate with a rule must be in the same Region as the rule.
- Some target types might not be available in every region. For more information, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.
- Creating rules with built-in targets is supported only in the AWS Management Console.
- If you create a rule with an encrypted Amazon SQS queue as a target, you must have the following section included in your KMS key policy. It allows the event to be successfully delivered to the encrypted queue.

```
{
    "Sid": "Allow CWE to use the key",
    "Effect": "Allow",
    "Principal": {
        "Service": "events.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

Creating a CloudWatch Events Rule That Triggers on an Event

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Use the following steps to create a CloudWatch Events rule that triggers on an event emitted by an AWS service.

To create a rule that triggers on an event:

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 the service that emits the event to trigger the rule.
 - c. For **Event Type**, choose the specific event that is to trigger the rule. If the only option is **AWS API Call via CloudTrail**, the selected service does not emit events and you can only base rules on API calls made to this service. For more information about creating this type of rule, see [Creating a CloudWatch Events Rule That Triggers on an AWS API Call Using AWS CloudTrail](#) (p. 7).
 - d. Depending on the service emitting the event, you may see options for **Any...** and **Specific....** Choose **Any...** to have the event trigger on any type of the selected event, or choose **Specific...** to choose one or more specific event types.
4. For **Targets**, choose **Add Target** and choose the AWS service that is to act when an event of the selected type is detected.
5. In the other fields in this section, enter information specific to this target type, if any is needed.
6. For many target types, CloudWatch Events needs permissions to send events to the target. In these cases, 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**.
7. Optionally, repeat steps 4-6 to add another target for this rule.
8. Choose **Configure details**. For **Rule definition**, type a name and description for the rule.

The rule name must be unique within this Region.
9. Choose **Create rule**.

Creating a CloudWatch Events Rule That Triggers on an AWS API Call Using AWS CloudTrail

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

To create a rule that triggers on an action by an AWS service that does not emit events, you can base the rule on API calls made by that service. The API calls are recorded by AWS CloudTrail. For more information about the API calls that you can use as triggers for rules, see [Services Supported by CloudTrail Event History](#).

Rules in CloudWatch Events work only in the Region in which they are created. If you configure CloudTrail to track API calls in multiple Regions, and you want a rule based on CloudTrail to trigger in each of those Regions, you must create a separate rule in each Region that you want to track.

All events that are delivered via CloudTrail have `AWS API Call via CloudTrail` as the value for `detail-type`.

Note

In CloudWatch Events, it is possible to create rules that lead to infinite loops, where a rule is fired repeatedly. For example, a rule might detect that ACLs have changed on an S3 bucket, and trigger software to change them to the desired state. If the rule is not written carefully, the subsequent change to the ACLs fires the rule again, creating an infinite loop.

To prevent this, write the rules so that the triggered actions do not re-fire the same rule. For example, your rule could fire only if ACLs are found to be in a bad state, instead of after any change.

An infinite loop can quickly cause higher than expected charges. We recommend that you use budgeting, which alerts you when charges exceed your specified limit. For more information, see [Managing Your Costs with Budgets](#).

To create a rule that triggers on an API call via CloudTrail:

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 the service that uses the API operations to use as the trigger.
 - c. For **Event Type**, choose **AWS API Call via CloudTrail**.
 - d. To trigger your rule when any API operation for this service is called, choose **Any operation**. To trigger your rule only when certain API operations are called, choose **Specific operation(s)**, type the name of an operation in the next box, and then press ENTER. To add more operations, choose **+**.
4. For **Targets**, choose **Add Target** and choose the AWS service that is to act when an event of the selected type is detected.
5. In the other fields in this section, enter information specific to this target type, if any is needed.
6. For many target types, CloudWatch Events needs permissions to send events to the target. In these cases, 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**.
7. Optionally, repeat steps 4-6 to add another target for this rule.
 8. Choose **Configure details**. For **Rule definition**, type a name and description for the rule.

The rule name must be unique within this Region.
 9. Choose **Create rule**.

Creating a CloudWatch Events Rule That Triggers on a Schedule

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Use the following steps to create a CloudWatch Events rule that triggers on a regular schedule.

To create a rule that triggers on a regular schedule

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**, choose **Schedule**.
4. Choose **Fixed rate of** and specify how often the task is to run, or choose **Cron expression** and specify a cron expression that defines when the task is to be triggered. For more information about cron expression syntax, see [Schedule Expressions for Rules \(p. 30\)](#).
5. For **Targets**, choose **Add Target** and choose the AWS service that is to act when an event of the selected type is detected.
6. In the other fields in this section, enter information specific to this target type, if any is needed.
7. For many target types, CloudWatch Events needs permissions to send events to the target. In these cases, 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**.
8. Optionally, repeat steps 5-7 to add another target for this rule.
9. Choose **Configure details**. For **Rule definition**, type a name and description for the rule.

The rule name must be unique within this Region.
10. Choose **Create rule**.

Deleting or Disabling a CloudWatch Events Rule

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Use the following steps to delete or disable a CloudWatch Events rule.

To delete or disable a rule

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

Managed rules have a box icon next to their names. For more information, see [Amazon CloudWatch Events Managed Rules \(p. 95\)](#).

3. Do one of the following:
 - a. To delete a rule, select the button next to the rule and choose **Actions, Delete, Delete**.

If the rule is a managed rule, you must type the name of the rule to acknowledge that it is a managed rule, and that deleting it may stop functionality in the service that created the rule. To continue, type the rule name and choose **Force delete**.
 - b. To temporarily disable a rule, select the button next to the rule and choose **Actions, Disable, Disable**.

You cannot disable a managed rule.

CloudWatch Events Tutorials

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

The following tutorials show you how to create CloudWatch Events rules for certain tasks and targets.

Tutorials:

- [Tutorial: Use CloudWatch Events to Relay Events to AWS Systems Manager Run Command \(p. 10\)](#)
- [Tutorial: Log the State of an Amazon EC2 Instance Using CloudWatch Events \(p. 11\)](#)
- [Tutorial: Log the State of an Auto Scaling Group Using CloudWatch Events \(p. 13\)](#)
- [Tutorial: Log Amazon S3 Object-Level Operations Using CloudWatch Events \(p. 15\)](#)
- [Tutorial: Use Input Transformer to Customize What is Passed to the Event Target \(p. 17\)](#)
- [Tutorial: Log AWS API Calls Using CloudWatch Events \(p. 18\)](#)
- [Tutorial: Schedule Automated Amazon EBS Snapshots Using CloudWatch Events \(p. 20\)](#)
- [Tutorial: Schedule AWS Lambda Functions Using CloudWatch Events \(p. 21\)](#)
- [Tutorial: Set AWS Systems Manager Automation as a CloudWatch Events Target \(p. 24\)](#)
- [Tutorial: Relay Events to an Amazon Kinesis Stream Using CloudWatch Events \(p. 25\)](#)
- [Tutorial: Run an Amazon ECS Task When a File is Uploaded to an Amazon S3 Bucket \(p. 27\)](#)
- [Tutorial: Schedule Automated Builds Using CodeBuild \(p. 28\)](#)
- [Tutorial: Log State Changes of Amazon EC2 Instances \(p. 29\)](#)

Tutorial: Use CloudWatch Events to Relay Events to AWS Systems Manager Run Command

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can use Amazon CloudWatch Events to invoke AWS 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 Amazon EC2 Auto Scaling group. This tutorial assumes that you have already assigned a tag to the Amazon EC2 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 Amazon EC2 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)**. There are many other **Document** options that 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**.
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 Amazon EC2 Instance Using CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can create an 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 an AWS Lambda Function

Create a Lambda function to log the state change events. You 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 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**. For **Existing role**, select your basic execution 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**, **Create rule**.
3. For **Event source**, do the following:
 - a. Choose **Event Pattern**.
 - b. Choose **Build event pattern to match events by service**.
 - c. Choose **EC2, EC2 Instance State-change Notification**.
 - d. Choose **Specific state(s), 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 select one or more instances.
4. For **Targets**, choose **Add target, Lambda function**.
5. For **Function**, select the Lambda function that you created.
6. Choose **Configure details**.
7. For **Rule definition**, type a name and description for the rule.
8. 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. In the navigation pane, choose **Events, Rules**, select the name of the rule that you created, and 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 that you launched.
6. (Optional) When you are finished, you can open the Amazon EC2 console and stop or terminate the instance that 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

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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 information about additional CloudWatch Events scenarios using Amazon EC2 Auto Scaling events, see [Getting CloudWatch Events When Your Auto Scaling Group Scales](#) in the *Amazon EC2 Auto Scaling User Guide*.

Step 1: Create an AWS Lambda Function

Create a Lambda function to log the scale-out and scale-in events for your Auto Scaling group. 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 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**. For **Existing role**, select your basic execution 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**, **Create rule**.
3. For **Event source**, do the following:
 - a. Choose **Event Pattern**.
 - b. Choose **Build event pattern to match events by service**.
 - c. Choose **Auto Scaling**, **Instance Launch and Terminate**.
 - d. To capture all successful and unsuccessful instance launch and terminate events, choose **Any instance event**.
4. 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 select one or more Auto Scaling groups.
5. For **Targets**, choose **Add target**, **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. For example, describe the rule as "Log whenever an Auto Scaling group scales out or in".
9. 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, 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 check box 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. If your new value for **Desired** is greater than **Max**, you must update **Max**. When you are finished, choose **Save**.

2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Events, Rules**, select the name of the rule that you created, and then 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. Select the name of the log group for your Lambda function (`/aws/lambda/function-name`).
 - c. Select the name of log stream to view the data provided by the function for the instance that 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 Amazon S3 Object-Level Operations Using CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can log the object-level API operations on your 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: Configure Your AWS CloudTrail Trail

To log data events for an S3 bucket to AWS CloudTrail and CloudWatch Events, create a trail. A trail captures API calls and related events in your account and delivers the log files to an S3 bucket that you specify. You can update an existing trail or create a new one.

To create a trail

1. Open the CloudTrail console at <https://console.aws.amazon.com/cloudtrail/>.
2. In the navigation pane, choose **Trails, Create trail**.
3. For **Trail name**, type a name for the trail.
4. For **Data events**, type the bucket name and prefix (optional). For each trail, you can add up to 250 Amazon S3 objects.
 - To log data events for all Amazon S3 objects in a bucket, specify an S3 bucket and an empty prefix. When an event occurs on an object in that bucket, the trail processes and logs the event.
 - To log data events for specific Amazon S3 objects, choose **Add S3 bucket**, then specify an S3 bucket and optionally the object prefix. When an event occurs on an object in that bucket and the object starts with the specified prefix, the trail processes and logs the event.
5. For each resource, specify whether to log **Read** events, **Write** events, or both.
6. For **Storage location**, create or choose an existing S3 bucket to designate for log file storage.
7. Choose **Create**.

For more information, see [Data Events](#) in the AWS CloudTrail User Guide.

Step 2: Create an AWS Lambda Function

Create a Lambda function to log data events for your S3 buckets. You 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 **Create a function**. Otherwise, choose **Create function**.
3. Choose **Author from scratch**.
4. Under **Author from scratch**, do the following:
 - a. Type a name for the Lambda function. For example, name the function "LogS3DataEvents".
 - b. For **Role**, choose **Create a custom role**.

A new window opens. Change the **Role name** if necessary, and choose **Allow**.
 - c. Back in the Lambda console, choose **Create function**.
5. Edit the code for the Lambda function to the following, and choose **Save**.

```
'use strict';

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

Step 3: Create a Rule

Create a rule to run your Lambda function in response to an Amazon 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 **Rules**, **Create rule**.
3. 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), Object Level Operations**.
 - d. Choose **Specific operation(s), 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.
4. For **Targets**, choose **Add target, Lambda function**.
5. For **Function**, select the Lambda function that you created.
6. Choose **Configure details**.
7. For **Rule definition**, type a name and description for the rule.
8. 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. Select the name of the log group for your Lambda function (`/aws/lambda/function-name`).
4. Select the name of log stream to view the data provided by the function for the instance that 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: Use Input Transformer to Customize What is Passed to the Event Target

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can use the input transformer feature of CloudWatch Events to customize the text that is taken from an event before it is input to the target of a rule.

You can define multiple JSON paths from the event and assign their outputs to different variables. Then you can use those variables in the input template as `<variable-name>`. The characters `<` and `>` cannot be escaped.

If you specify a variable to match a JSON path that does not exist in the event, then that variable is not created and does not appear in the output.

In this tutorial, we extract the instance-id and state of an Amazon EC2 instance from the instance state change event. We use the input transformer to put that data into an easy-to-read message that is sent to an Amazon SNS topic. The rule is triggered when any instance changes to any state. For example, with this rule, the following Amazon EC2 instance state-change notification event produces the Amazon SNS message **The EC2 instance i-1234567890abcdef0 has changed state to stopped.**

```
{
  "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-1234567890abcdef0"
  ],
}
```

```
"detail":{  
  "instance-id":" i-1234567890abcdef0",  
  "state":"stopped"  
}  
}
```

We achieve this by mapping the *instance* variable to the `$.detail.instance-id` JSON path from the event, and the *state* variable to the `$.detail.state` JSON path. We then set the input template as "The EC2 instance <instance> has changed state to <state>."

Create a Rule

To customize instance state change information sent to a target using the input transformer

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**.
 - b. Choose **Build event pattern to match events by service**.
 - c. Choose **EC2, EC2 Instance State-change Notification**.
 - d. Choose **Any state, Any instance**.
4. For **Targets**, choose **Add target, SNS topic**.
5. For **Topic**, select the Amazon SNS topic for which to be notified when Amazon EC2 instances change state.
6. Choose **Configure input, Input Transformer**.
7. In the next box, type `{"state" : "$.detail.state", "instance" : "$.detail.instance-id"}`
8. In the following box, type "The EC2 instance <instance> has changed state to <state>."
9. Choose **Configure details**.
10. Type a name and description for the rule, and choose **Create rule**.

Tutorial: Log AWS API Calls Using CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can use an 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 **Trails, Create trail**.
3. For **Trail name**, type a name for the trail.
4. For **Storage location**, in **Create a new S3 bucket** type the name for the new bucket that CloudTrail should deliver logs to.
5. Choose **Create**.

Step 1: Create an AWS Lambda Function

Create a Lambda function to log the API call events. 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 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**. For **Existing role**, select your basic execution 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, Create rule**.
3. For **Event source**, do the following:
 - a. Choose **Event Pattern**.
 - b. Choose **Build event pattern to match events by service**.
 - c. Choose **EC2, AWS API Call via CloudTrail**.

- d. Choose **Specific operation(s)** and then type `StopInstances` in the box below.
4. For **Targets**, choose **Add target, Lambda function**.
5. For **Function**, select the Lambda function that you created.
6. Choose **Configure details**.
7. For **Rule definition**, type a name and description for the rule.
8. 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. In the navigation pane, choose **Events**, select the name of the rule that you created, and 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. Select the name of the log group for your Lambda function (`/aws/lambda/function-name`).
 - c. Select the name of log stream to view the data provided by the function for the instance that 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 Automated Amazon EBS Snapshots Using CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can run CloudWatch Events rules according to a schedule. In this tutorial, you create an automated 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 \(p. 30\)](#).

To create a 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 **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).
4. For **Targets**, choose **Add target** and then select **EC2 CreateSnapshot API call**. You may have to scroll up in the list of possible targets to find **EC2 CreateSnapshot API call**.
5. For **Volume ID**, type the volume ID of the targeted Amazon EBS volume.
6. Choose **Create a new role for this specific resource**. The new role grants the target permissions to access resources on your behalf.
7. Choose **Configure details**.
8. For **Rule definition**, type a name and description for the rule.
9. 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 choose **Actions**, **Disable**.
 - d. When prompted for confirmation, choose **Disable**.

Tutorial: Schedule AWS Lambda Functions Using CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features.

Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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 is triggered within that minute but not on the precise 0th second.

Step 1: Create an AWS Lambda Function

Create a Lambda function to log the scheduled events. 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 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**. For **Existing role**, select your basic execution 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**, **Create rule**.
3. For **Event Source**, do the following:

- a. Choose **Schedule**.
 - b. Choose **Fixed rate of** and specify the schedule interval (for example, 5 minutes).
4. For **Targets**, choose **Add target, Lambda function**.
5. For **Function**, select the Lambda function that you created.
6. Choose **Configure details**.
7. For **Rule definition**, type a name and description for the rule.
8. 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 that you created to this rule so that it runs every five 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: Verify the Rule

At least five minutes after completing Step 2, 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. In the navigation pane, choose **Events, Rules**, select the name of the rule that you created, and 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. Select the name of the log group for your Lambda function (`/aws/lambda/function-name`).
 - c. Select the name of log stream to view the data provided by the function for the instance that you launched.
4. (Optional) When you are finished, you can disable the rule.
 - 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 choose **Actions, Disable**.
 - d. When prompted for confirmation, choose **Disable**.

Tutorial: Set AWS Systems Manager Automation as a CloudWatch Events Target

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can use CloudWatch Events to invoke AWS Systems Manager Automation on a regular timed schedule, or when specified events are detected. This tutorial assumes that you are invoking Systems Manager Automation based on certain events.

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** and choose **Build event pattern to match events by service**.

- b. For **Service Name** and **Event Type**, choose the service and event type to use as the trigger.

Depending on the service and event type you choose, you may need to specify additional options under **Event Source**.

4. For **Targets**, choose **Add Target**, **SSM Automation**.
5. For **Document**, choose the Systems Manager document to run when the target is triggered.
6. (Optional), To specify a certain version of the document, choose **Configure document version**.
7. Under **Configure parameter(s)**, choose **No Parameter(s)** or **Constant**.

If you choose **Constant**, specify the constants to pass to the document execution.

8. 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**.
9. Choose **Configure details**. For **Rule definition**, type a name and description for the rule.
10. Choose **Create rule**.

Tutorial: Relay Events to an Amazon Kinesis Stream Using CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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**, **Create rule**.
3. For **Event source**, do the following:
 - a. Choose **Event Pattern**.
 - b. Choose **Build event pattern to match events by service**.
 - c. Choose **EC2, Instance State-change Notification**.
 - d. Choose **Specific state(s), Running**.
4. For **Targets**, choose **Add target, Kinesis stream**.
5. For **Stream**, select the stream that you created.
6. Choose **Create a new role for this specific resource**.
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, 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. In the navigation pane, choose **Events**, **Rules**, select the name of the rule that you created, and 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 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": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjplIxtZs1Sp
+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 represent all records in your stream. If you don't receive the data you expect, keep calling `get-records`.

Records in 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 see that it is the event relayed to the stream in JSON form.

Tutorial: Run an Amazon ECS Task When a File is Uploaded to an Amazon S3 Bucket

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can use CloudWatch Events to run Amazon ECS tasks when certain AWS events occur. In this tutorial, you set up a CloudWatch Events rule that runs an Amazon ECS task whenever a file is uploaded to a certain Amazon S3 bucket using the Amazon S3 PUT operation.

This tutorial assumes that you have already created the task definition in Amazon ECS.

To run an Amazon ECS task whenever a file is uploaded to an S3 bucket using the PUT operation

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**.
 - b. Choose **Build event pattern to match events by service**.
 - c. For **Service Name**, choose **Simple Storage Service (S3)**.
 - d. For **Event Type**, choose **Object Level Operations**.
 - e. Choose **Specific operation(s)**, **Put Object**.
 - f. Choose **Specific bucket(s) by name**, and enter type the name of the bucket.
4. For **Targets**, do the following:
 - a. Choose **Add target**, **ECS task**.
 - b. For **Cluster** and **Task Definition**, select the resources that you created.

- c. For **Launch Type**, choose `FARGATE` or `EC2`. `FARGATE` is shown only in regions where AWS Fargate is supported.
- d. (Optional) Specify a value for **Task Group**. If the **Launch Type** is `FARGATE`, optionally specify a **Platform Version**. Specify only the numeric portion of the platform version, such as 1.1.0.
- e. (Optional) specify a task definition revision and task count. If you do not specify a task definition revision, the latest is used.
- f. If your task definition uses the `awsipc` network mode, you must specify subnets and security groups. All subnets and security groups must be in the same VPC.

If you specify more than one security group or subnet, separate them with commas but not spaces.

For **Subnets**, specify the entire `subnet-id` value for each subnet, as in the following example:

```
subnet-123abcd, subnet-789abcd
```

- g. Choose whether to allow the public IP address to be auto-assigned.
- h. CloudWatch Events can create the IAM role needed for your task 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**. This must be a role that already has sufficient permissions to invoke the build. CloudWatch Events does not grant additional permissions to the role that you select.
5. Choose **Configure details**.
6. For **Rule definition**, type a name and description for the rule.
7. Choose **Create rule**.

Tutorial: Schedule Automated Builds Using CodeBuild

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

In the example in this tutorial, you schedule CodeBuild to run a build every week night at 8PM GMT. You also pass a constant to CodeBuild to be used for this scheduled build.

To create a rule scheduling an CodeBuild project build nightly at 8PM

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 **Schedule**.
 - b. Choose **Cron expression** and specify the following as the expression: `0 20 ? * MON-FRI *`. For more information about cron expressions, see [Schedule Expressions for Rules \(p. 30\)](#).
4. For **Targets**, choose **Add target**, **CodeBuild project**.
5. For **Project ARN**, type the ARN of the build project.

6. In this tutorial, we add the optional step of passing a parameter to CodeBuild, to override the default. This is not required when you set CodeBuild as the target. To pass the parameter, choose **Configure input, Constant (JSON text)**.

In the box under **Constant (JSON text)**, type the following to set the timeout override to 30 minutes for these scheduled builds: `{ "timeoutInMinutesOverride": 30 }`

For more information about the parameters you can pass, see [StartBuild](#). You cannot pass the `projectName` parameter in this field. Instead, you specify the project using the ARN in **Project ARN**.

7. CloudWatch Events can create the IAM role needed for your build project 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**. This must be a role that already has sufficient permissions to invoke the build. CloudWatch Events does not grant additional permissions to the role that you select.
8. Choose **Configure details**.
9. For **Rule definition**, type a name and description for the rule.
10. Choose **Create rule**.

Tutorial: Log State Changes of Amazon EC2 Instances

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

In the example in this tutorial, you create a rule causing state-change notifications in Amazon EC2 to be logged in CloudWatch Logs.

To create a rule to log Amazon EC2 state-change notifications in CloudWatch Logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Events** and then **Create rule**.
3. For **Event Source**, do the following:
 - a. Choose **Event Pattern**.
 - b. For **Service Name**, choose **EC2**.
 - c. For **Event Type**, choose **EC2 Instance State-change Notification**.
4. For **Targets**, choose **Add target**. In the list of services, choose **CloudWatch log group**.
5. For **Log Group**, enter a name for the log group to receive the state-change notifications.
6. Choose **Configure details**.
7. For **Rule definition**, enter a name and description for the rule.
8. Choose **Create rule**.

Schedule Expressions for Rules

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can create rules that self-trigger on an automated 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 supports cron expressions and rate expressions. Rate expressions are simpler to define but don't offer the fine-grained schedule control that cron expressions support. For example, with a cron expression, you can define a rule that triggers at a specified time on a certain day of each week or month. In contrast, rate expressions trigger a rule at a regular rate, such as once every hour or once every day.

Note

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.

Formats

- [Cron Expressions \(p. 30\)](#)
- [Rate Expressions \(p. 32\)](#)

Cron Expressions

Cron expressions have six required fields, which 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 | , - * / |

| Field | Values | Wildcards |
|-------------|----------------|-------------|
| 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. You cannot use * in both the Day-of-month and Day-of-week fields. If you use it in one, you must use ? in the other.
- 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 weekday closest to the third day of the month.
- The # wildcard in the Day-of-week field specifies a certain instance of the specified day of the week within a month. For example, 3#2 would be the second Tuesday of the month: the 3 refers to Tuesday because it is the third day of each week, and the 2 refers to the second day of that type within the month.

Note

If you use a '#' character, you can define only one expression in the day-of-week field. For example, "3#1, 6#3" is not valid because it is interpreted as two expressions.

Restrictions

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value (or a *) 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.

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 |

| Minutes | Hours | Day of month | Month | Day of week | Year | Meaning |
|---------|-------|--------------|-------|-------------|------|---|
| 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 |
| 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. The first example creates a rule that is triggered every day at 12:00pm UTC.

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

The next example creates a rule that is triggered every day, at 5 and 35 minutes past 2:00pm UTC.

```
aws events put-rule --schedule-expression "cron(5,35 14 * * ? *)" --name MyRule2
```

The next example creates a rule that is triggered at 10:15am UTC on the last Friday of each month during the years 2002 to 2005.

```
aws events put-rule --schedule-expression "cron(15 10 ? * 6L 2002-2005)" --name MyRule3
```

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. Different units are required for values of 1, such as `minute`, and values over 1, such as `minutes`.

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

Restrictions

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. The first example triggers the rule every minute, the second example triggers it every 5 minutes, the third triggers it once an hour, and the final example triggers it once a day.

```
aws events put-rule --schedule-expression "rate(1 minute)" --name MyRule2
```

```
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
```

Event Patterns in CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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": "2017-12-22T18:43:48Z",
  "region": "us-west-1",
  "resources": [
    "arn:aws:ec2:us-west-1:123456789012:instance/ i-1234567890abcdef0"
  ],
  "detail": {
    "instance-id": " i-1234567890abcdef0",
    "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 are different depending on which service generated the event and what the event is. The combination of the **source** and **detail-type** fields serves to identify the fields and values found in the **detail** field. For examples of events generated by AWS services, see [Event Types for CloudWatch Events](#).

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 appear in the **detail** field.

All events that are delivered via CloudTrail have `AWS API Call via CloudTrail` as the value for **detail-type**. For more information, see [Events Delivered Via CloudTrail \(p. 77\)](#).

source

Identifies the service that sourced the event. All events sourced from within AWS 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.

To find the correct value for `source` for an AWS service, see the table in [AWS Service Namespaces](#). For example, the `source` value for Amazon CloudFront is `aws.cloudfront`.

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 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 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.

When you write patterns to match events, you can use the `TestEventPattern` API or the `test-event-pattern` CLI command to make sure that your pattern will match the desired events. For more information, see [TestEventPattern](#) or [test-event-pattern](#).

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" ]
}
```

Matching Null Values and Empty Strings In Event Patterns

You can create a pattern that matches an event field that has a null value or an empty string. To see how this works, consider the following example event:

```
{
  "version": "0",
  "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": [
],
"detail": {
  "eventVersion": "",
  "responseElements": null
}
}
```

To match events where the value of `eventVersion` is an empty string, use the following pattern, which would match the event example.

```
{
  "detail": {
    "eventVersion": [""]
  }
}
```

To match events where the value of `responseElements` is null, use the following pattern, which would match the event example.

```
{
  "detail": {
    "responseElements": [null]
  }
}
```

Null values and empty strings are not interchangeable in pattern matching. A pattern that is written to detect empty strings will not catch values of `null`.

Arrays In CloudWatch Events Patterns

The value of each field in a 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.

For example, an example event pattern includes the following text:

```
"resources": [
  "arn:aws:ec2:us-east-1:123456789012:instance/i-b188560f",
  "arn:aws:ec2:us-east-1:111122223333:instance/i-b188560f",
  "arn:aws:ec2:us-east-1:444455556666:instance/i-b188560f",
]
```

The example pattern would match an event that includes the following text, because the first item in the pattern array matches the second item in the event array.

```
"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"
]
```

```
]
```

CloudWatch Events Event Examples From Supported Services

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

The AWS services in the following list emit events that can be detected by CloudWatch Events.

Additionally, you can also use CloudWatch Events with services that do not emit events and are not listed on this page, by watching for events delivered via CloudTrail. For more information, see [Events Delivered Via CloudTrail \(p. 77\)](#).

Event Types

- [Amazon Augmented AI Events \(p. 40\)](#)
- [Application Auto Scaling Events \(p. 40\)](#)
- [AWS Batch Events \(p. 40\)](#)
- [Amazon CloudWatch Events Scheduled Events \(p. 40\)](#)
- [Amazon Chime Events \(p. 41\)](#)
- [Events from CloudWatch \(p. 41\)](#)
- [CodeBuild Events \(p. 41\)](#)
- [CodeCommit Events \(p. 41\)](#)
- [AWS CodeDeploy Events \(p. 41\)](#)
- [CodePipeline Events \(p. 42\)](#)
- [AWS Config Events \(p. 43\)](#)
- [Amazon EBS Events \(p. 43\)](#)
- [Amazon EC2 Auto Scaling Events \(p. 44\)](#)
- [Amazon EC2 Instance Rebalance Recommendation Events \(p. 44\)](#)
- [Amazon EC2 Spot Instance Interruption Events \(p. 44\)](#)
- [Amazon EC2 State Change Events \(p. 44\)](#)
- [Amazon Elastic Container Registry Events \(p. 44\)](#)
- [Amazon Elastic Container Service Events \(p. 45\)](#)
- [AWS Elemental MediaConvert Events \(p. 45\)](#)
- [AWS Elemental MediaPackage Events \(p. 45\)](#)
- [AWS Elemental MediaStore Events \(p. 45\)](#)
- [Amazon EMR Events \(p. 45\)](#)
- [Amazon GameLift Event \(p. 47\)](#)
- [AWS Glue Events \(p. 54\)](#)

- [AWS Ground Station Events \(p. 59\)](#)
- [Amazon GuardDuty Events \(p. 59\)](#)
- [AWS Health Events \(p. 59\)](#)
- [AWS KMS Events \(p. 61\)](#)
- [Amazon Macie Events \(p. 62\)](#)
- [AWS Management Console Sign-in Events \(p. 62\)](#)
- [AWS OpsWorks Stacks Events \(p. 63\)](#)
- [SageMaker Events \(p. 65\)](#)
- [AWS Security Hub Events \(p. 65\)](#)
- [AWS Server Migration Service Events \(p. 65\)](#)
- [AWS Systems Manager Events \(p. 66\)](#)
- [AWS Step Functions Events \(p. 75\)](#)
- [Tag Change Events on AWS Resources \(p. 75\)](#)
- [AWS Trusted Advisor Events \(p. 75\)](#)
- [WorkSpaces Events \(p. 77\)](#)
- [Events Delivered Via CloudTrail \(p. 77\)](#)

Amazon Augmented AI Events

For examples of events generated by Amazon Augmented AI, see [Use Events in Amazon Augmented AI](#).

Application Auto Scaling Events

For examples of events generated by Application Auto Scaling, see [Application Auto Scaling Events and EventBridge](#).

AWS Batch Events

For examples of events generated by AWS Batch, see [AWS Batch Events](#).

Amazon CloudWatch Events 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": "2019-10-08T16:53:06Z",
  "region": "us-east-1",
  "resources": [ "arn:aws:events:us-east-1:123456789012:rule/MyScheduledRule" ],
  "detail": {}
}
```

Amazon Chime Events

For examples of events generated by Amazon Chime, see [Automating Amazon Chime with EventBridge](#).

Events from CloudWatch

For sample events from CloudWatch, see [Alarm Events and EventBridge](#) in the *AWS CodeBuild User Guide*.

CodeBuild Events

For CodeBuild sample events, see [Build Notifications Input Format Reference](#) in the *AWS CodeBuild User Guide*.

CodeCommit Events

For CodeCommit sample events, see [Monitoring CodeCommit Events in EventBridge and CloudWatch Events](#) in the *AWS CodeCommit User Guide*.

AWS CodeDeploy Events

The following are examples of the events for 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-0000000aaaaaaaa",
    "arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:myApplication/myDeploymentGroup",
    "arn:aws:codedeploy:us-east-1:123456789012:application:myApplication"
  ],
  "detail": {
    "instanceId": "i-0000000aaaaaaaa",
    "region": "us-east-1",
    "state": "SUCCESS",
    "application": "myApplication",
    "deploymentId": "d-123456789",
    "instanceGroupId": "8cd3bfa8-9e72-4cbe-a1e5-da4efc7efd49",
    "deploymentGroup": "myDeploymentGroup"
  }
}
```

CodePipeline Events

The following are examples of events for CodePipeline.

Pipeline Execution State Change

```
{
  "version": "0",
  "id": "CWE-event-id",
  "detail-type": "CodePipeline Pipeline Execution State Change",
  "source": "aws.codepipeline",
  "account": "123456789012",
  "time": "2017-04-22T03:31:47Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:codepipeline:us-east-1:123456789012:pipeline:myPipeline"
  ],
  "detail": {
    "pipeline": "myPipeline",
    "version": "1",
    "state": "STARTED",
    "execution-id": "01234567-0123-0123-0123-012345678901"
  }
}
```

Stage Execution State Change

```
{
  "version": "0",
  "id": "CWE-event-id",
  "detail-type": "CodePipeline Stage Execution State Change",
  "source": "aws.codepipeline",
  "account": "123456789012",
```

```
"time": "2017-04-22T03:31:47Z",
"region": "us-east-1",
"resources": [
  "arn:aws:codepipeline:us-east-1:123456789012:pipeline:myPipeline"
],
"detail": {
  "pipeline": "myPipeline",
  "version": "1",
  "execution-id": "01234567-0123-0123-0123-012345678901",
  "stage": "Prod",
  "state": "STARTED"
}
}
```

Action Execution State Change

In this sample, there are two `region` fields. The one at the top is the name of the AWS Region where the action in the target pipeline is executed. In this example, this is `us-east-1`. The `region` in the `detail` section is the AWS Region where the event was created. This is the same as the Region where the pipeline was created. In this example, this is `us-west-2`.

```
{
  "version": "0",
  "id": "CWE-event-id",
  "detail-type": "CodePipeline Action Execution State Change",
  "source": "aws.codepipeline",
  "account": "123456789012",
  "time": "2017-04-22T03:31:47Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:codepipeline:us-east-1:123456789012:pipeline:myPipeline"
  ],
  "detail": {
    "pipeline": "myPipeline",
    "version": 1,
    "execution-id": "01234567-0123-0123-0123-012345678901",
    "stage": "Prod",
    "action": "myAction",
    "state": "STARTED",
    "region": "us-west-2",
    "type": {
      "owner": "AWS",
      "category": "Deploy",
      "provider": "CodeDeploy",
      "version": 1
    }
  }
}
```

AWS Config Events

For information about the AWS Config events, see [Monitoring AWS Config with Amazon CloudWatch Events](#) in the *AWS Config Developer Guide*.

Amazon EBS Events

For information about the Amazon EBS events, see [Amazon CloudWatch Events for Amazon EBS](#) in the *Amazon EC2 User Guide for Linux Instances*.

Amazon EC2 Auto Scaling Events

For information about the Auto Scaling events, see [Getting CloudWatch Events When Your Auto Scaling Group Scales](#) in the *Amazon EC2 Auto Scaling User Guide*.

Amazon EC2 Instance Rebalance Recommendation Events

For information about the events for EC2 instance rebalance recommendations, see [Monitoring rebalance recommendation signals](#) in the *Amazon EC2 User Guide for Linux Instances*.

Amazon EC2 Spot Instance Interruption Events

For information about the events for Spot Instance interruptions, see [Spot Instance interruption notices](#) in the *Amazon EC2 User Guide for Linux Instances*.

Amazon EC2 State Change Events

The following is an example of the events for Amazon EC2 instances when the instance state changes.

EC2 Instance State-change Notification

This example is for an instance in the pending state. The other possible values for `state` include `running`, `shutting-down`, `stopped`, `stopping`, and `terminated`.

```
{
  "id": "7bf73129-1428-4cd3-a780-95db273d1602",
  "detail-type": "EC2 Instance State-change Notification",
  "source": "aws.ec2",
  "account": "123456789012",
  "time": "2019-11-11T21:29:54Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"
  ],
  "detail": {
    "instance-id": "i-abcd1111",
    "state": "pending"
  }
}
```

Amazon Elastic Container Registry Events

Amazon ECR sends image actions events to EventBridge. Events are sent when images are pushed, scanned, or deleted.

For Amazon ECS sample events, see [Amazon ECR Events](#) in the *Amazon Elastic Container Registry User Guide*.

Amazon Elastic Container Service Events

Amazon ECS sends two types of events to EventBridge: container instance events and task events. Container instance events are only sent if you are using the EC2 launch type for your tasks. For tasks using the Fargate launch type, you only receive task state events. Amazon ECS tracks the state of container instances and tasks. If either resources changes, an event is triggered. These events are classified as either container instance state change events or task state change events.

For Amazon ECS sample events, see [Amazon ECS Events](#) in the *Amazon Elastic Container Service Developer Guide*.

AWS Elemental MediaConvert Events

For MediaConvert sample events, see [Using CloudWatch Events to Monitor AWS Elemental MediaConvert Jobs](#) in the *AWS Elemental MediaConvert User Guide*.

AWS Elemental MediaPackage Events

For MediaPackage sample events, see [Monitoring AWS Elemental MediaPackage with Amazon CloudWatch Events](#) in the *AWS Elemental MediaPackage User Guide*.

AWS Elemental MediaStore Events

For MediaStore sample events, see [Automating AWS Elemental MediaStore with CloudWatch Events](#) in the *AWS Elemental MediaStore User Guide*.

Amazon EMR Events

Events reported by Amazon EMR have `aws.emr` as the value for `Source`, while Amazon EMR API events reported via CloudTrail have `aws.elasticmapreduce` as the value for `Source`.

The following are examples of events reported by Amazon EMR.

Amazon EMR Auto Scaling 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\", \"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-aaaa-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."
  }
}
```

Amazon GameLift Event

The following are examples of Amazon GameLift events. For more information, see [FlexMatch Events Reference](#) in the *Amazon GameLift Developer Guide*.

Matchmaking Searching

```
{
  "version": "0",
  "id": "cc3d3ebe-1d90-48f8-b268-c96655b8f013",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-08T21:15:36.421Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration:SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-08T21:15:35.676Z",
        "players": [
          {
            "playerId": "player-1"
          }
        ]
      }
    ]
  }
}
```

```
    }
  ]
}
],
"estimatedWaitMillis": "NOT_AVAILABLE",
"type": "MatchmakingSearching",
"gameSessionInfo": {
  "players": [
    {
      "playerId": "player-1"
    }
  ]
}
}
}
```

Potential Match Created

```
{
  "version": "0",
  "id": "fce8633f-aea3-45bc-aebe-99d639cad2d4",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-08T21:17:41.178Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-08T21:15:35.676Z",
        "players": [
          {
            "playerId": "player-1",
            "team": "red"
          }
        ]
      },
      {
        "ticketId": "ticket-2",
        "startTime": "2017-08-08T21:17:40.657Z",
        "players": [
          {
            "playerId": "player-2",
            "team": "blue"
          }
        ]
      }
    ]
  },
  "acceptanceTimeout": 600,
  "ruleEvaluationMetrics": [
    {
      "ruleName": "EvenSkill",
      "passedCount": 3,
      "failedCount": 0
    },
    {
      "ruleName": "EvenTeams",
      "passedCount": 3,
      "failedCount": 0
    }
  ],
  {
```

```
        "ruleName": "FastConnection",
        "passedCount": 3,
        "failedCount": 0
    },
    {
        "ruleName": "NoobSegregation",
        "passedCount": 3,
        "failedCount": 0
    }
],
"acceptanceRequired": true,
"type": "PotentialMatchCreated",
"gameSessionInfo": {
    "players": [
        {
            "playerId": "player-1",
            "team": "red"
        },
        {
            "playerId": "player-2",
            "team": "blue"
        }
    ]
},
"matchId": "3faf26ac-f06e-43e5-8d86-08feff26f692"
}
}
```

Accept Match

```
{
    "version": "0",
    "id": "b3f76d66-c8e5-416a-aa4c-aa1278153edc",
    "detail-type": "GameLift Matchmaking Event",
    "source": "aws.gamelift",
    "account": "123456789012",
    "time": "2017-08-09T20:04:42.660Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
    ],
    "detail": {
        "tickets": [
            {
                "ticketId": "ticket-1",
                "startTime": "2017-08-09T20:01:35.305Z",
                "players": [
                    {
                        "playerId": "player-1",
                        "team": "red"
                    }
                ]
            },
            {
                "ticketId": "ticket-2",
                "startTime": "2017-08-09T20:04:16.637Z",
                "players": [
                    {
                        "playerId": "player-2",
                        "team": "blue",
                        "accepted": false
                    }
                ]
            }
        ]
    }
},
]
```

```
"type": "AcceptMatch",
"gameSessionInfo": {
  "players": [
    {
      "playerId": "player-1",
      "team": "red"
    },
    {
      "playerId": "player-2",
      "team": "blue",
      "accepted": false
    }
  ]
},
"matchId": "848b5f1f-0460-488e-8631-2960934d13e5"
}
```

Accept Match Completed

```
{
  "version": "0",
  "id": "b1990d3d-f737-4d6c-b150-af5ace8c35d3",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-08T20:43:14.621Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-08T20:30:40.972Z",
        "players": [
          {
            "playerId": "player-1",
            "team": "red"
          }
        ]
      },
      {
        "ticketId": "ticket-2",
        "startTime": "2017-08-08T20:33:14.111Z",
        "players": [
          {
            "playerId": "player-2",
            "team": "blue"
          }
        ]
      }
    ]
  },
  "acceptance": "TimedOut",
  "type": "AcceptMatchCompleted",
  "gameSessionInfo": {
    "players": [
      {
        "playerId": "player-1",
        "team": "red"
      },
      {
        "playerId": "player-2",
        "team": "blue"
      }
    ]
  }
}
```

```
    }  
  ],  
  },  
  "matchId": "a0d9bd24-4695-4f12-876f-ea6386dd6dce"  
}  
}
```

Matchmaking Succeeded

```
{  
  "version": "0",  
  "id": "5ccb6523-0566-412d-b63c-1569e00d023d",  
  "detail-type": "GameLift Matchmaking Event",  
  "source": "aws.gamelift",  
  "account": "123456789012",  
  "time": "2017-08-09T19:59:09.159Z",  
  "region": "us-west-2",  
  "resources": [  
    "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"  
  ],  
  "detail": {  
    "tickets": [  
      {  
        "ticketId": "ticket-1",  
        "startTime": "2017-08-09T19:58:59.277Z",  
        "players": [  
          {  
            "playerId": "player-1",  
            "playerSessionId": "psess-6e7c13cf-10d6-4756-a53f-db7de782ed67",  
            "team": "red"  
          }  
        ]  
      },  
      {  
        "ticketId": "ticket-2",  
        "startTime": "2017-08-09T19:59:08.663Z",  
        "players": [  
          {  
            "playerId": "player-2",  
            "playerSessionId": "psess-786b342f-9c94-44eb-bb9e-c1de46c472ce",  
            "team": "blue"  
          }  
        ]  
      }  
    ]  
  },  
  "type": "MatchmakingSucceeded",  
  "gameSessionInfo": {  
    "gameSessionArn": "arn:aws:gamelift:us-west-2:123456789012:gamesession/836cf48d-bcb0-4a2c-bec1-9c456541352a",  
    "ipAddress": "192.168.1.1",  
    "port": 10777,  
    "players": [  
      {  
        "playerId": "player-1",  
        "playerSessionId": "psess-6e7c13cf-10d6-4756-a53f-db7de782ed67",  
        "team": "red"  
      },  
      {  
        "playerId": "player-2",  
        "playerSessionId": "psess-786b342f-9c94-44eb-bb9e-c1de46c472ce",  
        "team": "blue"  
      }  
    ]  
  },  
  "matchId": "c0ec1a54-7fec-4b55-8583-76d67adb7754"  
}
```



```
}  
}
```

Matchmaking Timed Out

```
{  
  "version": "0",  
  "id": "fe528a7d-46ad-4bdc-96cb-b094b5f6bf56",  
  "detail-type": "GameLift Matchmaking Event",  
  "source": "aws.gamelift",  
  "account": "123456789012",  
  "time": "2017-08-09T20:11:35.598Z",  
  "region": "us-west-2",  
  "resources": [  
    "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"  
  ],  
  "detail": {  
    "reason": "TimedOut",  
    "tickets": [  
      {  
        "ticketId": "ticket-1",  
        "startTime": "2017-08-09T20:01:35.305Z",  
        "players": [  
          {  
            "playerId": "player-1",  
            "team": "red"  
          }  
        ]  
      }  
    ]  
  },  
  "ruleEvaluationMetrics": [  
    {  
      "ruleName": "EvenSkill",  
      "passedCount": 3,  
      "failedCount": 0  
    },  
    {  
      "ruleName": "EvenTeams",  
      "passedCount": 3,  
      "failedCount": 0  
    },  
    {  
      "ruleName": "FastConnection",  
      "passedCount": 3,  
      "failedCount": 0  
    },  
    {  
      "ruleName": "NoobSegregation",  
      "passedCount": 3,  
      "failedCount": 0  
    }  
  ],  
  "type": "MatchmakingTimedOut",  
  "message": "Removed from matchmaking due to timing out.",  
  "gameSessionInfo": {  
    "players": [  
      {  
        "playerId": "player-1",  
        "team": "red"  
      }  
    ]  
  }  
}
```

Matchmaking Cancelled

```
{
  "version": "0",
  "id": "8d6f84da-5e15-4741-8d5c-5ac99091c27f",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-09T20:00:07.843Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
  ],
  "detail": {
    "reason": "Cancelled",
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-09T19:59:26.118Z",
        "players": [
          {
            "playerId": "player-1"
          }
        ]
      }
    ]
  },
  "ruleEvaluationMetrics": [
    {
      "ruleName": "EvenSkill",
      "passedCount": 0,
      "failedCount": 0
    },
    {
      "ruleName": "EvenTeams",
      "passedCount": 0,
      "failedCount": 0
    },
    {
      "ruleName": "FastConnection",
      "passedCount": 0,
      "failedCount": 0
    },
    {
      "ruleName": "NoobSegregation",
      "passedCount": 0,
      "failedCount": 0
    }
  ],
  "type": "MatchmakingCancelled",
  "message": "Cancelled by request.",
  "gameSessionInfo": {
    "players": [
      {
        "playerId": "player-1"
      }
    ]
  }
}
```

Matchmaking Failed

```
{
  "version": "0",
```

```
"id": "025b55a4-41ac-4cf4-89d1-f2b3c6fd8f9d",
"detail-type": "GameLift Matchmaking Event",
"source": "aws.gamelift",
"account": "123456789012",
"time": "2017-08-16T18:41:09.970Z",
"region": "us-west-2",
"resources": [
  "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
],
"detail": {
  "tickets": [
    {
      "ticketId": "ticket-1",
      "startTime": "2017-08-16T18:41:02.631Z",
      "players": [
        {
          "playerId": "player-1",
          "team": "red"
        }
      ]
    }
  ],
  "customEventData": "foo",
  "type": "MatchmakingFailed",
  "reason": "UNEXPECTED_ERROR",
  "message": "An unexpected error was encountered during match placing.",
  "gameSessionInfo": {
    "players": [
      {
        "playerId": "player-1",
        "team": "red"
      }
    ]
  },
  "matchId": "3ea83c13-218b-43a3-936e-135cc570cba7"
}
}
```

AWS Glue Events

The following is the format for AWS Glue events.

Successful Job Run

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Glue Job State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2017-09-07T18:57:21Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "INFO",
    "state": "SUCCEEDED",
    "jobRunId": "jr_abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789",
    "message": "Job run succeeded"
  }
}
```

Failed Job Run

```
{
  "version": "0",
  "id": "abcdef01-1234-5678-9abc-def012345678",
  "detail-type": "Glue Job State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2017-09-07T06:02:03Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "ERROR",
    "state": "FAILED",
    "jobRunId": "jr_0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef",
    "message": "JobName:MyJob and
JobRunId:jr_0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef failed to
execute with exception Role arn:aws:iam::123456789012:role/Glue_Role should be given
assume role permissions for Glue Service."
  }
}
```

Timeout

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Glue Job State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2017-11-20T20:22:06Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "WARN",
    "state": "TIMEOUT",
    "jobRunId": "jr_abc0123456789abcdef0123456789abcdef0123456789abcdef0123456789def",
    "message": "Job run timed out"
  }
}
```

Stopped Job Run

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Glue Job State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2017-11-20T20:22:06Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "INFO",
    "state": "STOPPED",
    "jobRunId": "jr_abc0123456789abcdef0123456789abcdef0123456789abcdef0123456789def",
    "message": "Job run stopped"
  }
}
```

Crawler Started

```
{
  "version": "0",
  "id": "05efe8a2-c309-6884-a41b-3508bcd9695",
  "detail-type": "Glue Crawler State Change",
  "source": "aws.glue",
  "account": "561226563745",
  "time": "2017-11-11T01:09:46Z",
  "region": "us-east-1",
  "resources": [
  ],
  "detail": {
    "accountId": "561226563745",
    "crawlerName": "S3toS3AcceptanceTestCrawlera470bd94-9e00-4518-8942-e80c8431c322",
    "startTime": "2017-11-11T01:09:46Z",
    "state": "Started",
    "message": "Crawler Started"
  }
}
```

Crawler Succeeded

```
{
  "version": "0",
  "id": "3d675db5-59b9-6388-b8e8-e0a9b6d567a9",
  "detail-type": "Glue Crawler State Change",
  "source": "aws.glue",
  "account": "561226563745",
  "time": "2017-11-11T01:25:00Z",
  "region": "us-east-1",
  "resources": [
  ],
  "detail": {
    "tablesCreated": "0",
    "warningMessage": "N/A",
    "partitionsUpdated": "0",
    "tablesUpdated": "0",
    "message": "Crawler Succeeded",
    "partitionsDeleted": "0",
    "accountId": "561226563745",
    "runningTime (sec)": "7",
    "tablesDeleted": "0",
    "crawlerName": "SchedulerTestCrawler51fb3a8b-1015-49f0-a969-ca126680b94b",
    "completionDate": "2017-11-11T01:25:00Z",
    "state": "Succeeded",
    "partitionsCreated": "0",
    "cloudWatchLogLink": "https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logEventViewer:group=/aws-glue/crawlers;stream=SchedulerTestCrawler51fb3a8b-1015-49f0-a969-ca126680b94b"
  }
}
```

Crawler Failed

```
{
  "version": "0",
  "id": "f7965b59-470f-2e06-bb89-a8cebaabefac",
  "detail-type": "Glue Crawler State Change",
  "source": "aws.glue",
  "account": "782104008917",

```

```
"time": "2017-10-20T05:10:08Z",
"region": "us-east-1",
"resources": [
],
"detail": {
  "crawlerName": "test-crawler-notification",
  "errorMessage": "Internal Service Exception",
  "accountId": "1234",
  "cloudWatchLogLink": "https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logEventViewer:group=/aws-glue/crawlers;stream=test-crawler-notification",
  "state": "Failed",
  "message": "Crawler Failed"
}
}
```

Job Run is in Starting State

```
{
  "version": "0",
  "id": "66fbc5e1-aac3-5e85-63d0-856ec669a050",
  "detail-type": "Glue Job Run Status",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2018-04-24T20:57:34Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "INFO",
    "notificationCondition": {
      "NotifyDelayAfter": 1.0
    },
    "state": "STARTING",
    "jobRunId": "jxr_6aa58e7a3aa44e2e4c7db2c50e2f7396cb57901729e4b702dcb2cfbb3f7a86",
    "message": "Job is in STARTING state",
    "startedOn": "2018-04-24T20:55:47.941Z"
  }
}
```

Job Run is in Running State

```
{
  "version": "0",
  "id": "66fbc5e1-aac3-5e85-63d0-856ec669a050",
  "detail-type": "Glue Job Run Status",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2018-04-24T20:57:34Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "INFO",
    "notificationCondition": {
      "NotifyDelayAfter": 1.0
    },
    "state": "RUNNING",
    "jobRunId": "jxr_6aa58e7a3aa44e2e4c7db2c50e2f7396cb57901729e4b702dcb2cfbb3f7a86",
    "message": "Job is in RUNNING state",
    "startedOn": "2018-04-24T20:55:47.941Z"
  }
}
```

Job Run is in Stopping State

```
{
  "version": "0",
  "id": "66fbc5e1-aac3-5e85-63d0-856ec669a050",
  "detail-type": "Glue Job Run Status",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2018-04-24T20:57:34Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "INFO",
    "notificationCondition": {
      "NotifyDelayAfter": 1.0
    },
    "state": "STOPPING",
    "jobRunId": "jr_6aa58e7a3aa44e2e4c7db2c50e2f7396cb57901729e4b702dcb2cfbb3f7a86",
    "message": "Job is in STOPPING state",
    "startedOn": "2018-04-24T20:55:47.941Z"
  }
}
```

AWS Glue Data Catalog Table State Change

```
{
  "version": "0",
  "id": "2617428d-715f-edef-70b8-d210da0317a0",
  "detail-type": "Glue Data Catalog Table State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2019-01-16T18:16:01Z",
  "region": "eu-west-1",
  "resources": [
    "arn:aws:glue:eu-west-1:123456789012:table/d1/t1"
  ],
  "detail": {
    "databaseName": "d1",
    "changedPartitions": [
      "[C.pdf, dir3]",
      "[D.doc, dir4]"
    ],
    "typeOfChange": "BatchCreatePartition",
    "tableName": "t1"
  }
}
```

AWS Glue Data Catalog Database State Change

In the following example, the `typeOfChange` is `CreateTable`. Other possible values for this field are `CreateDatabase` and `UpdateTable`.

```
{
  "version": "0",
  "id": "60e7ddc2-a588-5328-220a-21c060f6c3f4",
  "detail-type": "Glue Data Catalog Database State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2019-01-16T18:08:48Z",
  "region": "eu-west-1",
  "resources": [
    "arn:aws:glue:eu-west-1:123456789012:table/d1/t1"
  ]
}
```

```
    ],
    "detail": {
      "databaseName": "d1",
      "typeOfChange": "CreateTable",
      "changedTables": [
        "t1"
      ]
    }
  }
}
```

AWS Ground Station Events

For information about example AWS Ground Station events, see [Automating AWS Ground Station with CloudWatch Events](#) in the *AWS Ground Station User Guide*.

Amazon GuardDuty Events

For information about example Amazon GuardDuty events, see [Monitoring Amazon GuardDuty with Amazon CloudWatch Events](#) in the *Amazon GuardDuty User Guide*.

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"
    }]
    ...
  }
}
```

eventTypeCategory

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

eventTypeCode

The unique identifier for the event type. Examples include `AWS_EC2_INSTANCE_NETWORK_MAINTENANCE_SCHEDULED` and `AWS_EC2_INSTANCE_REBOOT_MAINTENANCE_SCHEDULED`. Events that include `MAINTENANCE_SCHEDULED` are usually pushed out about two weeks before the `startTime`.

id

The unique identifier for the event.

service

The AWS service affected by the event. For example, EC2, S3, REDSHIFT, or 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"
}
}
```

Amazon Macie Events

For examples of events generated by Amazon Macie, see [Event schema for Amazon Macie findings](#).

AWS Management Console Sign-in Events

AWS Management Console sign-in events can be detected by CloudWatch Events only in the US East (N. Virginia) Region.

The following is an example of a 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 OpsWorks Stacks Events

The following are examples of AWS OpsWorks Stacks events.

AWS OpsWorks Stacks instance state change

Indicates a change in the state of an AWS OpsWorks Stacks instance. The following are instance states.

- booting
- connection_lost
- online
- pending
- rebooting
- requested
- running_setup
- setup_failed
- shutting_down
- start_failed
- stopping
- stop_failed
- stopped
- terminating
- terminated

```
{  
  "version": "0",  
  "id": "dc5fa8df-48f1-2108-b1b9-1fe5ebcf2296",  
  "detail-type": "OpsWorks Instance State Change",  
  "source": "aws.opsworks",  
  "account": "123456789012",  
  "time": "2018-01-25T11:12:23Z",  
  "region": "us-east-1",  
  "resources": [  
    "arn:aws:opsworks:us-east-1:123456789012:instance/a648d98f-fdd8-4323-952a-a50z3e4z500z"  
  ],  
  "detail": {  
    "initiated_by": "user",  
    "hostname": "testing1",  
    "stack-id": "acd3df16-e859-4598-8414-377b12a902da",  
    "layer-ids": [  
      "d1a0cb7f-c7e9-4a63-811c-976f0267b2c8"  
    ],  
    "instance-id": "a648d98f-fdd8-4323-952a-a50z3e4z500z",  
    "ec2-instance-id": "i-08b1c2b67aa292276",  
    "status": "requested"  
  }  
}
```

The `initiated_by` field is only populated when the instance is in the requested, terminating, or stopping states. The `initiated_by` field can contain one of the following values.

- `user` - A user requested the instance state change by using either the API or AWS Management Console.
- `auto-scaling` - The AWS OpsWorks Stacks automatic scaling feature initiated the instance state change.
- `auto-healing` - The AWS OpsWorks Stacks automatic healing feature initiated the instance state change.

AWS OpsWorks Stacks command state change

A change occurred in the state of an AWS OpsWorks Stacks command. The following are command states.

- `expired` - A command timed out.
- `failed` - A general command failure occurred.
- `skipped` - A command was skipped because the instance has a different state in AWS OpsWorks Stacks than in Amazon EC2.
- `successful` - A command succeeded.
- `superseded` - A command was skipped because it would have applied configuration changes that have already been applied.

```
{
  "version": "0",
  "id": "96c778b6-a40e-c8c1-aa5c-c9852a3a7b52",
  "detail-type": "OpsWorks Command State Change",
  "source": "aws.opsworks",
  "account": "123456789012",
  "time": "2018-01-26T08:54:40Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:opsworks:us-east-1:123456789012:instance/a648d98f-fdd8-4323-952a-a50a3e4e500f"
  ],
  "detail": {
    "command-id": "acc9f4f3-a3ec-4fab-b70f-c7d04e71e3ec",
    "instance-id": "a648d98f-fdd8-4323-952a-a50a3e4e500f",
    "type": "setup",
    "status": "successful"
  }
}
```

AWS OpsWorks Stacks deployment state change

A change occurred in the state of an AWS OpsWorks Stacks deployment. The following are deployment states.

- `running`
- `successful`
- `failed`

```
{
  "version": "0",
  "id": "b8230afa-60c7-f43f-b632-841c1cfb22ff",
  "detail-type": "OpsWorks Deployment State Change",
}
```

```
{
  "source": "aws.opsworks",
  "account": "123456789012",
  "time": "2018-01-25T11:15:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:opsworks:us-east-1:123456789012:instance/a648d98f-fdd8-4323-952a-a50a3e4e500f"
  ],
  "detail": {
    "duration": 16,
    "stack-id": "acd3df16-e859-4598-8414-377b12a902da",
    "instance-ids": [
      "a648d98f-fdd8-4323-952a-a50a3e4e500f"
    ],
    "deployment-id": "606419dc-418e-489c-8531-bff9770fc346",
    "command": "configure",
    "status": "successful"
  }
}
```

The `duration` field is only populated when a deployment is finished, and shows time in seconds.

AWS OpsWorks Stacks alert

An AWS OpsWorks Stacks service error was raised.

```
{
  "version": "0",
  "id": "f99faa6f-0e27-e398-95bb-8f190806d275",
  "detail-type": "OpsWorks Alert",
  "source": "aws.opsworks",
  "account": "123456789012",
  "time": "2018-01-20T16:51:29Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "stack-id": "2f48f2be-ac7d-4dd5-80bb-88375f94db7b",
    "instance-id": "986efb74-69e8-4c6d-878e-5b77c054cbb0",
    "type": "InstanceStop",
    "message": "The shutdown of the instance timed out. Please try stopping it again."
  }
}
```

SageMaker Events

For information about example SageMaker events, see [Automating SageMaker with Amazon EventBridge](#) in the *SageMaker Developer Guide*.

AWS Security Hub Events

For information about example Security Hub events, see [Monitoring AWS Security Hub with Amazon CloudWatch Events](#) in the *AWS Security Hub User Guide*.

AWS Server Migration Service Events

The following are examples of the events for AWS Server Migration Service.

Deleted replication job notification

```
{
  "version": "0",
  "id": "5630992d-92cd-439f-f2a8-92c8212aee24",
  "detail-type": "Server Migration Job State Change",
  "source": "aws.sms",
  "account": "123456789012",
  "time": "2018-02-07T22:30:11Z",
  "region": "us-west-1",
  "resources": [
    "arn:aws:sms:us-west-1:123456789012:sms-job-21a64348"
  ],
  "detail": {
    "state": "Deleted",
    "replication-run-id": "N/A",
    "replication-job-id": "sms-job-21a64348",
    "version": "1.0"
  }
}
```

Completed replication job notification

```
{
  "version": "0",
  "id": "3f9c59cc-f941-522a-be6d-f08e44ff1715",
  "detail-type": "Server Migration Job State Change",
  "source": "aws.sms",
  "account": "123456789012",
  "time": "2018-02-07T22:54:00Z",
  "region": "us-west-1",
  "resources": [
    "arn:aws:sms:us-west-1:123456789012:sms-job-2ea64347",
    "arn:aws:sms:us-west-1:123456789012:sms-job-2ea64347/sms-run-e1a64388"
  ],
  "detail": {
    "state": "Completed",
    "replication-run-id": "sms-run-e1a64388",
    "replication-job-id": "sms-job-2ea64347",
    "ami-id": "ami-746d6314",
    "version": "1.0"
  }
}
```

AWS Systems Manager Events

The following are examples of the events for AWS Systems Manager. For more information, see [Monitoring Systems Manager events with Amazon EventBridge](#) in the *AWS Systems Manager User Guide*.

Systems Manager event types

- [AWS Systems Manager Automation Events](#) (p. 67)
- [AWS Systems Manager Change Calendar Events](#) (p. 67)
- [AWS Systems Manager Compliance Events](#) (p. 68)
- [AWS Systems Manager Maintenance Windows Events](#) (p. 70)
- [AWS Systems Manager Parameter Store Events](#) (p. 72)
- [AWS Systems Manager Run Command Events](#) (p. 73)
- [AWS Systems Manager State Manager Events](#) (p. 74)

AWS Systems Manager Automation Events

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"
  }
}
```

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"
  }
}
```

AWS Systems Manager Change Calendar Events

The following are examples of the events for AWS Systems Manager Change Calendar.

Note

State changes for calendars shared from other AWS accounts are not currently supported.

Calendar OPEN


```
{
  "version": "0",
  "id": "47a3f03a-f30d-1011-ac9a-du3bdEXAMPLE",
  "detail-type": "Calendar State Change",
  "source": "aws.ssm",
  "account": "111222333444",
  "time": "2020-09-19T18:00:07Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:ssm:us-east-2:111222333444:document/MyCalendar"
  ],
  "detail": {
    "state": "OPEN",
    "atTime": "2020-09-19T18:00:07Z",
    "nextTransitionTime": "2020-10-11T18:00:07Z"
  }
}
```

Calendar CLOSED

```
{
  "version": "0",
  "id": "f30df03a-1011-ac9a-47a3-f761eEXAMPLE",
  "detail-type": "Calendar State Change",
  "source": "aws.ssm",
  "account": "111222333444",
  "time": "2020-09-17T21:40:02Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:ssm:us-east-2:111222333444:document/MyCalendar"
  ],
  "detail": {
    "state": "CLOSED",
    "atTime": "2020-08-17T21:40:00Z",
    "nextTransitionTime": "2020-09-19T18:00:07Z"
  }
}
```

AWS Systems Manager Compliance Events

The following are examples of the events for AWS Systems Manager Compliance.

Association Compliant

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Configuration Compliance State Change",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2017-07-17T19:03:26Z",
  "region": "us-west-1",
  "resources": [
    "arn:aws:ssm:us-west-1:461348341421:managed-instance/i-01234567890abcdef"
  ],
  "detail": {
    "last-runtime": "2017-01-01T10:10:10Z",
    "compliance-status": "compliant",
    "resource-type": "managed-instance",
    "resource-id": "i-01234567890abcdef",
    "compliance-type": "Association"
  }
}
```

```
}  
}
```

Association Non-Compliant

```
{  
  "version": "0",  
  "id": "01234567-0123-0123-0123-012345678901",  
  "detail-type": "Configuration Compliance State Change",  
  "source": "aws.ssm",  
  "account": "123456789012",  
  "time": "2017-07-17T19:02:31Z",  
  "region": "us-west-1",  
  "resources": [  
    "arn:aws:ssm:us-west-1:461348341421:managed-instance/i-01234567890abcdef"  
  ],  
  "detail": {  
    "last-runtime": "2017-01-01T10:10:10Z",  
    "compliance-status": "non_compliant",  
    "resource-type": "managed-instance",  
    "resource-id": "i-01234567890abcdef",  
    "compliance-type": "Association"  
  }  
}
```

Patch Compliant

```
{  
  "version": "0",  
  "id": "01234567-0123-0123-0123-012345678901",  
  "detail-type": "Configuration Compliance State Change",  
  "source": "aws.ssm",  
  "account": "123456789012",  
  "time": "2017-07-17T19:03:26Z",  
  "region": "us-west-1",  
  "resources": [  
    "arn:aws:ssm:us-west-1:461348341421:managed-instance/i-01234567890abcdef"  
  ],  
  "detail": {  
    "resource-type": "managed-instance",  
    "resource-id": "i-01234567890abcdef",  
    "compliance-status": "compliant",  
    "compliance-type": "Patch",  
    "patch-baseline-id": "PB789",  
    "severity": "critical"  
  }  
}
```

Patch Non-Compliant

```
{  
  "version": "0",  
  "id": "01234567-0123-0123-0123-012345678901",  
  "detail-type": "Configuration Compliance State Change",  
  "source": "aws.ssm",  
  "account": "123456789012",  
  "time": "2017-07-17T19:02:31Z",  
  "region": "us-west-1",  
  "resources": [  
    "arn:aws:ssm:us-west-1:461348341421:managed-instance/i-01234567890abcdef"  
  ],  
  "detail": {
```

```
"resource-type": "managed-instance",
"resource-id": "i-01234567890abcdef",
"compliance-status": "non_compliant",
"compliance-type": "Patch",
"patch-baseline-id": "PB789",
"severity": "critical"
}
}
```

AWS Systems Manager Maintenance Windows Events

The following are examples of the events for Systems Manager Maintenance Windows.

Register a Target

The other valid status value is 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 valid status values 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 valid status values 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 valid status values 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 valid status values 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"
  }
}
```

AWS Systems Manager Parameter Store Events

The following are examples of the events for Systems Manager Parameter Store.

Create Parameter

```
{
  "version": "0",
  "id": "6a7e4feb-b491-4cf7-a9f1-bf3703497718",
  "detail-type": "Parameter Store Change",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2017-05-22T16:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ssm:us-east-1:123456789012:parameter/foo"
  ],
  "detail": {
    "operation": "Create",
    "name": "foo",
    "type": "String",
    "description": "Sample Parameter"
  }
}
```

Update Parameter

```
{
  "version": "0",
  "id": "9547ef2d-3b7e-4057-b6cb-5fdf09ee7c8f",
  "detail-type": "Parameter Store Change",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2017-05-22T16:44:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ssm:us-east-1:123456789012:parameter/foo"
  ],
  "detail": {
    "operation": "Update",
    "name": "foo",
    "type": "String",
    "description": "Sample Parameter"
  }
}
```

Delete Parameter

```
{
```

```
"version": "0",
"id": "80e9b391-6a9b-413c-839a-453b528053af",
"detail-type": "Parameter Store Change",
"source": "aws.ssm",
"account": "123456789012",
"time": "2017-05-22T16:45:48Z",
"region": "us-east-1",
"resources": [
  "arn:aws:ssm:us-east-1:123456789012:parameter/foo"
],
"detail": {
  "operation": "Delete",
  "name": "foo",
  "type": "String",
  "description": "Sample Parameter"
}
}
```

AWS Systems Manager Run Command Events

Run 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"
  }
}
```

Run 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"
  }
}
```

```
}
```

AWS Systems Manager State Manager Events

State Manager Association State Change

```
{
  "version": "0",
  "id": "db839caf-6f6c-40af-9a48-25b2ae2b7774",
  "detail-type": "EC2 State Manager Association State Change",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2017-05-16T23:01:10Z",
  "region": "us-west-1",
  "resources": [
    "arn:aws:ssm:us-west-1::document/AWS-RunPowerShellScript"
  ],
  "detail": {
    "association-id": "6e37940a-23ba-4ab0-9b96-5d0a1a05464f",
    "document-name": "AWS-RunPowerShellScript",
    "association-version": "1",
    "document-version": "Optional.empty",
    "targets": "[{\\"key\\":\\"InstanceIds\\",\\"values\\":[\\\"i-12345678\\\"]}]",
    "creation-date": "2017-02-13T17:22:54.458Z",
    "last-successful-execution-date": "2017-05-16T23:00:01Z",
    "last-execution-date": "2017-05-16T23:00:01Z",
    "last-updated-date": "2017-02-13T17:22:54.458Z",
    "status": "Success",
    "association-status-aggregated-count": "{\\"Success\\":1}",
    "schedule-expression": "cron(0 */30 * * * ? *)",
    "association-cwe-version": "1.0"
  }
}
```

State Manager Instance Association State Change

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "EC2 State Manager Instance Association State Change",
  "source": "aws.ssm",
  "account": "123456789012",
  "time": "2017-02-23T15:23:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ec2:us-east-1:123456789012:instance/i-12345678",
    "arn:aws:ssm:us-east-1:123456789012:document/my-custom-document"
  ],
  "detail": {
    "association-id": "34fcb7e0-9a14-4984-9989-0e04e3f60bd8",
    "instance-id": "i-12345678",
    "document-name": "my-custom-document",
    "document-version": "1",
    "targets": "[{\\"key\\":\\"instanceids\\",\\"values\\":[\\\"i-12345678\\\"]}]",
    "creation-date": "2017-02-23T15:23:48Z",
    "last-successful-execution-date": "2017-02-23T16:23:48Z",
    "last-execution-date": "2017-02-23T16:23:48Z",
    "status": "Success",
    "detailed-status": "",
    "error-code": "testErrorCode",
    "execution-summary": "testExecutionSummary",
    "output-url": "sampleurl",
  }
}
```

```
    "instance-association-cwe-version": "1"  
  }  
}
```

AWS Step Functions Events

For Step Functions sample events, see [Step Functions Event Examples](#) in the *AWS Step Functions Developer Guide*.

Tag Change Events on AWS Resources

The following is an example of a tag event.

```
{  
  "version": "0",  
  "id": "ffd8a6fe-32f8-ef66-c85c-111111111111",  
  "detail-type": "Tag Change on Resource",  
  "source": "aws.tag",  
  "account": "123456789012",  
  "time": "2018-09-18T20:41:06Z",  
  "region": "us-east-1",  
  "resources": [  
    "arn:aws:ec2:us-east-1:123456789012:instance/i-00000000aaaaaaaaa"  
  ],  
  "detail": {  
    "changed-tag-keys": [  
      "key2",  
      "key3"  
    ],  
    "service": "ec2",  
    "resource-type": "instance",  
    "version": 5,  
    "tags": {  
      "key4": "value4",  
      "key1": "value1",  
      "key2": "value2"  
    }  
  }  
}
```

AWS 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

```
{  
  "version": "0",  
  "id": "1234abcd-ab12-123a-123a-1234567890ab",  
  "detail-type": "Trusted Advisor Check Item Refresh Notification",  
  "source": "aws.trustedadvisor",  
  "account": "123456789012",  
  "time": "2018-01-12T20:07:49Z",  
  "region": "us-east-2",  
}
```



```
"resources": [],
"detail": {
  "check-name": "Low Utilization Amazon EC2 Instances",
  "check-item-detail": {
    "Day 1": "0.1% 0.00MB",
    "Day 2": "0.1% 0.00MB",
    "Day 3": "0.1% 0.00MB",
    "Region/AZ": "ca-central-1a",
    "Estimated Monthly Savings": "$9.22",
    "14-Day Average CPU Utilization": "0.1%",
    "Day 14": "0.1% 0.00MB",
    "Day 13": "0.1% 0.00MB",
    "Day 12": "0.1% 0.00MB",
    "Day 11": "0.1% 0.00MB",
    "Day 10": "0.1% 0.00MB",
    "14-Day Average Network I/O": "0.00MB",
    "Number of Days Low Utilization": "14 days",
    "Instance Type": "t2.micro",
    "Instance ID": "i-01234567890abcdef",
    "Day 8": "0.1% 0.00MB",
    "Instance Name": null,
    "Day 9": "0.1% 0.00MB",
    "Day 4": "0.1% 0.00MB",
    "Day 5": "0.1% 0.00MB",
    "Day 6": "0.1% 0.00MB",
    "Day 7": "0.1% 0.00MB"
  },
  "status": "WARN",
  "resource_id": "arn:aws:ec2:ca-central-1:123456789012:instance/i-01234567890abcdef",
  "uuid": "aa12345f-55c7-498e-b7ac-123456789012"
}
}
```

Load Balancer Optimization

```
{
  "version": "0",
  "id": "1234abcd-ab12-123a-123a-1234567890ab",
  "detail-type": "Trusted Advisor Check Item Refresh Notification",
  "source": "aws.trustedadvisor",
  "account": "123456789012",
  "time": "2018-01-12T20:07:03Z",
  "region": "us-east-2",
  "resources": [],
  "detail": {
    "check-name": "Load Balancer Optimization ",
    "check-item-detail": {
      "Instances in Zone a": "1",
      "Status": "Yellow",
      "Instances in Zone b": "0",
      "# of Zones": "2",
      "Region": "eu-central-1",
      "Load Balancer Name": "my-load-balance",
      "Instances in Zone e": null,
      "Instances in Zone c": null,
      "Reason": "Single AZ",
      "Instances in Zone d": null
    },
    "status": "WARN",
    "resource_id": "arn:aws:elasticloadbalancing:eu-central-1:123456789012:loadbalancer/my-load-balancer",
    "uuid": "aa12345f-55c7-498e-b7ac-123456789012"
  }
}
```

Exposed Access Keys

```
{
  "version": "0",
  "id": "1234abcd-ab12-123a-123a-1234567890ab",
  "detail-type": "Trusted Advisor Check Item Refresh Notification",
  "source": "aws.trustedadvisor",
  "account": "123456789012",
  "time": "2018-01-12T19:38:24Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "check-name": "Exposed Access Keys",
    "check-item-detail": {
      "Case ID": "12345678-1234-1234-abcd-1234567890ab",
      "Usage (USD per Day)": "0",
      "User Name (IAM or Root)": "my-username",
      "Deadline": "1440453299248",
      "Access Key ID": "AKIAIOSFODNN7EXAMPLE",
      "Time Updated": "1440021299248",
      "Fraud Type": "Exposed",
      "Location": "www.example.com"
    },
    "status": "ERROR",
    "resource_id": "",
    "uuid": "aa12345f-55c7-498e-b7ac-123456789012"
  }
}
```

WorkSpaces Events

For information about the WorkSpaces events, see [Monitor Your WorkSpaces Using CloudWatch Events](#) in the *Amazon WorkSpaces Administration Guide*.

Events Delivered Via CloudTrail

You can also use CloudWatch Events with services that do not emit events and are not listed on this page. AWS CloudTrail is a service that automatically records events such as AWS API calls. You can create CloudWatch Events rules that trigger on the information captured by CloudTrail. For more information about CloudTrail, see [What is AWS CloudTrail?](#) For more information about creating a CloudWatch Events rule that uses CloudTrail, see [Creating a CloudWatch Events Rule That Triggers on an AWS API Call Using AWS CloudTrail \(p. 7\)](#).

All events that are delivered via CloudTrail have `AWS API Call via CloudTrail` as the value for `detail-type`.

Some occurrences in AWS can be reported to CloudWatch Events both by the service itself and by CloudTrail, but in different ways. For example, an Amazon EC2 API call that launches or terminates an instance generates events available to CloudWatch Events through CloudTrail. However, the Amazon EC2 instance state changes, from 'running' to 'terminating' for example, are CloudWatch Events events themselves.

The following is an example of an event delivered via CloudTrail. The event was generated by an AWS API call 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"
  }
}
```

AWS API call events that are larger than 256 KB in size are not supported. For more information about the API calls that you can use as triggers for rules, see [Services Supported by CloudTrail Event History](#).

Sending and Receiving Events Between AWS Accounts

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

You can set up your AWS account to send events to other AWS accounts, or to receive events from other accounts. This can be useful if the accounts belong to the same organization, or belong to organizations that are partners or have a similar relationship.

If you set up your account to send or receive events, you specify which individual AWS accounts can send events to or receive events from yours. If you use the AWS Organizations feature, you can specify an organization and grant access to all accounts in that organization. For more information, see [What is AWS Organizations](#) in the *AWS Organizations User Guide*.

The overall process is as follows:

- On the *receiver* account, edit the permissions on the default *event bus* to allow specified AWS accounts, an organization, or all AWS accounts to send events to the receiver account.
- On the *sender* account, set up one or more rules that have the receiver account's default event bus as the target.

If the sender account has permissions to send events because it is part of an AWS organization that has permissions, the sender account also must have an IAM role with policies that enable it to send events to the receiver account. If you use the AWS Management Console to create the rule that targets the receiver account, this is done automatically. If you use the AWS CLI, you must create the role manually.

- On the *receiver* account, set up one or more rules that match events that come from the sender account.

The AWS Region where the receiver account adds permissions to the default event bus must be the same region where the sender account creates the rule to send events to the receiver account.

Events sent from one account to another are charged to the sending account as custom events. The receiving account is not charged. For more information, see [Amazon CloudWatch Pricing](#).

If a receiver account sets up a rule that sends events received from a sender account on to a third account, these events are not sent to the third account.

Enabling Your AWS Account to Receive Events from Other AWS Accounts

To receive events from other accounts or organizations, you must first edit the permissions on your account's default *event bus*. The default event bus accepts events from AWS services, other authorized AWS accounts, and `PutEvents` calls.

When you edit the permissions on your default event bus to grant permission to other AWS accounts, you can specify accounts by account ID or organization ID. Or you can choose to receive events from all AWS accounts.

Warning

If you choose to receive events from all AWS accounts, be careful to create rules that match only the events to receive from others. To create more secure rules, make sure that the event pattern for each rule contains an `Account` field with the account IDs of one or more accounts from which to receive events. Rules that have an event pattern containing an `Account` field do not match events sent from accounts that are not listed in the `Account` field. For more information, see [Event Patterns in CloudWatch Events \(p. 34\)](#).

To enable your account to receive events from other AWS accounts using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Event Buses**, **Add Permission**.
3. Choose **AWS Account** or **Organization**.

If you choose **AWS Account**, enter the 12-digit AWS account ID of the account from which to receive events. To receive events from all other AWS accounts, choose **Everybody(*)**.

If you choose **Organization**, choose **My organization** to grant permissions to all accounts in the organization of which the current account is a member. Or choose **Another organization** and enter the organization ID of that organization. You must include the `o-` prefix when you type the organization ID.

4. Choose **Add**.
5. You can repeat these steps to add other accounts or organizations.

To enable your account to receive events from other AWS accounts using the AWS CLI

1. To enable one specific AWS account to send events, run the following command:

```
aws events put-permission --action events:PutEvents --statement-id MySid --  
principal SenderAccountID
```

To enable an AWS organization to send events, run the following command:

```
aws events put-permission --action events:PutEvents --statement-id MySid  
--principal \* --condition '{"Type" : "StringEquals", "Key":  
"aws:PrincipalOrgID", "Value": "SenderOrganizationID"}'
```

To enable all other AWS accounts to send events, run the following command:

```
aws events put-permission --action events:PutEvents --statement-id MySid --principal \*
```

You can run `aws events put-permission` multiple times to grant permissions to both individual AWS accounts and organizations, but you cannot specify both an individual account and an organization in a single command.

2. After setting permissions for your default event bus, you can optionally use the `describe-event-bus` command to check the permissions:

```
aws events describe-event-bus
```

Sending Events to Another AWS Account

To send events to another account, configure a CloudWatch Events rule that has the default event bus of another AWS account as the target. The default event bus of that receiving account must also be configured to receive events from your account.

To send events from your account to another AWS account using the console

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**, choose **Event Pattern** and select the service name and event types to send to the other account.
4. Choose **Add Target**.
5. For **Target**, choose **Event bus in another AWS account**. For **Account ID**, enter the 12-digit account ID of the AWS account to which to send events.
6. An IAM role is needed when this sender account has permissions to send events because the receiver account granted permissions to an entire organization.
 - To create an IAM role automatically, choose **Create a new role for this specific resource**.
 - Otherwise, choose **Use existing role**. Choose a role that already has sufficient permissions to invoke the build. CloudWatch Events does not grant additional permissions to the role that you select.
7. At the bottom of the page, choose **Configure Details**.
8. Type a name and description for the rule, and choose **Create Rule**.

To send events to another AWS account using the AWS CLI

1. If the sender account has permissions to send events because it is part of an AWS organization to which the receiver account has granted permissions, the sender account also must have a role with policies that enable it to send events to the receiver account. This step explains how to create that role.

If the sender account was given permission to send events by way of its AWS account ID, and not through an organization, this step is optional. You can skip to step 2.

- a. If the sender account was granted permissions through an organization, create the IAM role needed. First, create a file named `assume-role-policy-document.json`, with the following content:

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

- b. To create the role, enter the following command:

```
$ aws iam create-role \
--profile sender \
```

```
--role-name event-delivery-role \  
--assume-role-policy-document file://assume-role-policy-document.json
```

- c. Create a file named `permission-policy.json` with the following content:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "events:PutEvents"  
      ],  
      "Resource": [  
        "arn:aws:events:us-east-1:${receiver_account_id}:event-bus/default"  
      ]  
    }  
  ]  
}
```

- d. Enter the following command to attach this policy to the role:

```
$ aws iam put-role-policy \  
--profile sender \  
--role-name event-delivery-role \  
--policy-name EventBusDeliveryRolePolicy  
--policy-document file://permission-policy.json
```

2. Use the `put-rule` command to create a rule that matches the event types to send to the other account.
3. Add the other account's default event bus as the target of the rule.

If the sender account was given permissions to send events by its account ID, specifying a role is not necessary. Run the following command:

```
aws events put-targets --rule NameOfRuleMatchingEventsToSend --targets  
"Id"="MyId", "Arn"="arn:aws:events:region:$ReceiverAccountID:event-bus/default"
```

If the sender account was given permissions to send events by its organization, specify a role, as in the following example:

```
aws events put-targets --rule NameOfRuleMatchingEventsToSend --targets  
"Id"="MyId", "Arn"="arn:aws:events:region:$ReceiverAccountID:event-bus/  
default", "RoleArn"="arn:aws:iam:${sender_account_id}:role/event-delivery-role"
```

Writing Rules that Match Events from Another AWS Account

If your account is set up to receive events from other AWS accounts, you can write rules that match those events. Set the event pattern of the rule to match the events you are receiving from the other account.

Unless you specify account in the event pattern of a rule, any of your account's rules, both new and existing, that match events you receive from other accounts trigger based on those events. If you are receiving events from another account, and you want a rule to trigger only on that event pattern when it is generated from your own account, you must add account and specify your own account ID to the event pattern of the rule.

If you set up your AWS account to accept events from all AWS accounts, we strongly recommend that you add `account` to every CloudWatch Events rule in your account. This prevents rules in your account from triggering on events from unknown AWS accounts. When you specify the `account` field in the rule, you can specify the account IDs of more than one AWS account in the field.

To have a rule trigger on a matching event from any AWS account that you have granted permissions to, do not specify `*` in the `account` field of the rule. Doing so would not match any events, because `*` never appears in the `account` field of an event. Instead, just omit the `account` field from the rule.

To write a rule matching events from another account using the console

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**, choose **Event Pattern**, and select the service name and event types that the rule should match.
4. Near **Event Pattern Preview**, choose **Edit**.
5. In the edit window, add an `Account` line specifying which AWS accounts sending this event should be matched by the rule. For example, the edit window originally shows the following:

```
{
  "source": [
    "aws.ec2"
  ],
  "detail-type": [
    "EBS Volume Notification"
  ]
}
```

Add the following to make the rule match EBS volume notifications that are sent by the AWS accounts 123456789012 and 111122223333:

```
{
  "account": [
    "123456789012", "111122223333"
  ],
  "source": [
    "aws.ec2"
  ],
  "detail-type": [
    "EBS Volume Notification"
  ]
}
```

6. After editing the event pattern, choose **Save**.
7. Finish creating the rule as usual, setting one or more targets in your account.

To write a rule matching events from another AWS account using the AWS CLI

- Use the `put-rule` command. In the `Account` field in the rule's event pattern, specify the other AWS accounts for the rule to match. The following example rule matches Amazon EC2 instance state changes in the AWS accounts 123456789012 and 111122223333:

```
aws events put-rule --name "EC2InstanceStateChanges" --event-pattern "{\"account\": [\"123456789012\", \"111122223333\"], \"source\": [\"aws.ec2\"], \"detail-type\": [\"EC2 Instance State-change Notification\"]}" --role-arn "arn:aws:iam::123456789012:role/MyRoleForThisRule"
```


Migrate a Sender-Receiver Relationship to Use AWS Organizations

If you have a sender account that had permissions granted directly to its account ID, and you now want to revoke those permissions and give the sending account access by granting permissions to an organization, you must take some additional steps. These steps ensure that the events from the sender account can still get to the receiver account. This is because accounts that are given permission to send events via an organization must also use an IAM role to do so.

To add the permissions necessary to migrate a sender-receiver relationship

1. In the sender account, create an IAM role with policies that enable it to send events to the receiver account.
 - a. Create a file named `assume-role-policy-document.json`, with the following content:

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

- b. To create the IAM role, enter the following command:

```
$ aws iam create-role \
--profile sender \
--role-name event-delivery-role \
--assume-role-policy-document file://assume-role-policy-document.json
```

- c. Create a file named `permission-policy.json` with the following content:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "arn:aws:events:us-east-1:${receiver_account_id}:event-bus/default"
      ]
    }
  ]
}
```

- d. Enter the following command to attach this policy to the role:

```
$ aws iam put-role-policy \
--profile sender \
--role-name event-delivery-role \
--policy-name EventBusDeliveryRolePolicy
```

```
--policy-document file://permission-policy.json
```

2. Edit each existing rule in the sender account that has the receiver account default event bus as a target. Edit the rule by adding the role that you created in step 1 to the target information. Use the following command:

```
aws events put-targets --rule Rulename --targets  
  "Id"="MyID", "Arn"="arn:aws:events:region:$ReceiverAccountID:event-bus/  
default", "RoleArn"="arn:aws:iam:{sender_account_id}:role/event-delivery-role"
```

3. In the receiver account, run the following command to grant permissions for the accounts in the organization to send events to the receiver account:

```
aws events put-permission --action events:PutEvents --statement-id Sid-For-Organization  
  --principal \* --condition '{"Type" : "StringEquals", "Key": "aws:PrincipalOrgID",  
  "Value": "SenderOrganizationID"}'
```

Optionally, you can also revoke the permissions originally granted directly to the sender account:

```
aws events remove-permission --statement-id Sid-for-SenderAccount
```

Adding Events with PutEvents

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

The `PutEvents` action sends multiple events to CloudWatch Events in a single request. For more information, see [PutEvents](#) in the *Amazon CloudWatch Events API Reference* and [put-events](#) in the *AWS CLI Command Reference*.

Each `PutEvents` request can support a limited number of entries. For more information, see [CloudWatch Events Quotas \(p. 110\)](#). 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

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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 event entries into one request for efficiency.

Note

The size restriction is imposed on the entry. Even if the entry is less than the size restriction, it does not mean that the event in CloudWatch Events is also less than this size. On the contrary, the event size is always larger than the entry size due to the necessary characters and keys of the JSON representation of the event. For more information, see [Event Patterns in CloudWatch Events \(p. 34\)](#).

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;
}
```

Using CloudWatch Events with Interface VPC Endpoints

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and CloudWatch Events. You can use this connection to enable CloudWatch Events to communicate with your resources on your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. To connect your VPC to CloudWatch Events, you define an *interface VPC endpoint* for CloudWatch Events. This type of endpoint enables you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to CloudWatch Events without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see [What is Amazon VPC](#) in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see [New – AWS PrivateLink for AWS Services](#).

The following steps are for users of Amazon VPC. For more information, see [Getting Started](#) in the *Amazon VPC User Guide*.

Availability

CloudWatch Events currently supports VPC endpoints in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)

- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)

Creating a VPC Endpoint for CloudWatch Events

To start using CloudWatch Events with your VPC, create an interface VPC endpoint for CloudWatch Events. The service name to choose is **com.amazonaws.*Region*.events**. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch Events. CloudWatch Events calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch Events, and you already have a CloudWatch Events rule that sends notifications to Amazon SNS when it is triggered, the notifications begin to flow through the interface VPC endpoint.

Controlling Access to Your CloudWatch Events VPC Endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, we attach a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch Events. This policy enables users connecting to CloudWatch Events through the VPC to send events to CloudWatch Events, and prevents them from performing other CloudWatch Events actions.

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "events:PutEvents"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To modify the VPC endpoint policy for CloudWatch Events

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.

3. If you have not already created the endpoint for CloudWatch Events, choose **Create Endpoint**. Then select **com.amazonaws.*Region*.events** and choose **Create endpoint**.
4. Select the **com.amazonaws.*Region*.events** endpoint, and choose the **Policy** tab in the lower half of the screen.
5. Choose **Edit Policy** and make the changes to the policy.

Monitoring Usage with CloudWatch Metrics

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

CloudWatch Events sends metrics to Amazon CloudWatch every minute.

CloudWatch Events Metrics

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

All of these metrics use `Count` as the unit, so `Sum` and `SampleCount` are the most useful statistics.

| Metric | Description |
|------------------------------------|--|
| <code>DeadLetterInvocations</code> | <p>Measures the number of times a rule's target is not invoked in response to an event. This includes invocations that would result in triggering the same rule again, causing an infinite loop.</p> <p>Valid Dimensions: <code>RuleName</code></p> <p>Units: <code>Count</code></p> |
| <code>Invocations</code> | <p>Measures the number of times a target is invoked for a rule in response to an event. This includes successful and failed invocations, but does not include throttled or retried attempts until they fail permanently. It does not include <code>DeadLetterInvocations</code>.</p> <p>Note CloudWatch Events only sends this metric to CloudWatch if it has a non-zero value.</p> <p>Valid Dimensions: <code>RuleName</code></p> <p>Units: <code>Count</code></p> |
| <code>FailedInvocations</code> | <p>Measures the number of invocations that failed permanently. This does not include invocations that are retried, or that succeeded after a retry attempt. It also does not count failed invocations that are counted in <code>DeadLetterInvocations</code>.</p> <p>Valid Dimensions: <code>RuleName</code></p> <p>Units: <code>Count</code></p> |

| Metric | Description |
|----------------|---|
| TriggeredRules | Measures the number of triggered rules that matched with any event. Valid Dimensions: RuleName Units: Count |
| MatchedEvents | Measures the number of events that matched with any rule. Valid Dimensions: None Units: Count |
| ThrottledRules | Measures the number of triggered rules that are being throttled. Valid Dimensions: RuleName Units: Count |

Dimensions for CloudWatch Events Metrics

CloudWatch Events metrics have one dimension, which is listed below.

| Dimension | Description |
|-----------|---|
| RuleName | Filters the available metrics by rule name. |

Amazon CloudWatch Events Managed Rules

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Other AWS services can create and manage CloudWatch Events rules in your AWS account that are needed for certain functions in those services. These are called *managed rules*.

When a service creates a managed rule, it may also create an IAM policy that grants permissions to that service to create the rule. IAM policies created this way are scoped narrowly with resource-level permissions, to allow the creation of only the necessary rules.

You can delete managed rules by using the **Force delete** option. Do so only if you are sure that the other service no longer needs the rule. Otherwise, deleting a managed rule causes the features that rely on it to stop working.

Using CloudWatch Events with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

| SDK documentation | Code examples |
|--|--|
| AWS SDK for C++ | AWS SDK for C++ code examples |
| AWS SDK for Go | AWS SDK for Go code examples |
| AWS SDK for Java | AWS SDK for Java code examples |
| AWS SDK for JavaScript | AWS SDK for JavaScript code examples |
| AWS SDK for .NET | AWS SDK for .NET code examples |
| AWS SDK for PHP | AWS SDK for PHP code examples |
| AWS SDK for Python (Boto3) | AWS SDK for Python (Boto3) code examples |
| AWS SDK for Ruby | AWS SDK for Ruby code examples |

For examples specific to CloudWatch Events, see [Code examples for CloudWatch Events using AWS SDKs](#) (p. 97).

Example availability

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

Code examples for CloudWatch Events using AWS SDKs

The following code examples show how to use CloudWatch Events with an AWS software development kit (SDK).

The examples are divided into the following categories:

Actions

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

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Events with an AWS SDK \(p. 96\)](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Actions for CloudWatch Events using AWS SDKs \(p. 97\)](#)
 - [Add a Lambda function target using an AWS SDK \(p. 97\)](#)
 - [Create a CloudWatch Events scheduled rule using an AWS SDK \(p. 99\)](#)
 - [Send CloudWatch Events events using an AWS SDK \(p. 101\)](#)

Actions for CloudWatch Events using AWS SDKs

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

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

Examples

- [Add a Lambda function target using an AWS SDK \(p. 97\)](#)
- [Create a CloudWatch Events scheduled rule using an AWS SDK \(p. 99\)](#)
- [Send CloudWatch Events events using an AWS SDK \(p. 101\)](#)

Add a Lambda function target using an AWS SDK

The following code examples show how to add an AWS Lambda function target to an Amazon CloudWatch Events event.

Java

SDK for Java 2.x

```
public static void putCWTTargets(CloudWatchEventsClient cwe, String ruleName,
String functionArn, String targetId ) {
```

```
try {
    Target target = Target.builder()
        .arn(functionArn)
        .id(targetId)
        .build();

    PutTargetsRequest request = PutTargetsRequest.builder()
        .targets(target)
        .rule(ruleName)
        .build();

    PutTargetsResponse response = cwe.putTargets(request);
    System.out.printf(
        "Successfully created CloudWatch events target for rule %s",
        ruleName);

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutTargets](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

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

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { cweClient } from "../libs/cloudWatchEventsClient.js";

// Set the parameters
export const params = {
    Rule: "DEMO_EVENT",
    Targets: [
        {
            Arn: "LAMBDA_FUNCTION_ARN", //LAMBDA_FUNCTION_ARN
            Id: "myCloudWatchEventsTarget",
        },
    ],
};

export const run = async () => {
    try {
        const data = await cweClient.send(new PutTargetsCommand(params));
        console.log("Success, target added; requestID: ", data);
    }
}
```

```
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutTargets](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({apiVersion: '2015-10-07'});

var params = {
  Rule: 'DEMO_EVENT',
  Targets: [
    {
      Arn: 'LAMBDA_FUNCTION_ARN',
      Id: 'myCloudWatchEventsTarget',
    }
  ]
};

cwevents.putTargets(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutTargets](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Events with an AWS SDK \(p. 96\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create a CloudWatch Events scheduled rule using an AWS SDK

The following code examples show how to create an Amazon CloudWatch Events scheduled rule.

Java

SDK for Java 2.x


```
public static void putCWRule(CloudWatchEventsClient cwe, String ruleName,
String roleArn) {

    try {
        PutRuleRequest request = PutRuleRequest.builder()
            .name(ruleName)
            .roleArn(roleArn)
            .scheduleExpression("rate(5 minutes)")
            .state(RuleState.ENABLED)
            .build();

        PutRuleResponse response = cwe.putRule(request);
        System.out.printf(
            "Successfully created CloudWatch events rule %s with arn %s",
            roleArn, response.ruleArn());
    } catch (
        CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutRule](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

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

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { cweClient } from "../libs/cloudWatchEventsClient.js";

// Set the parameters
export const params = {
    Name: "DEMO_EVENT",
    RoleArn: "IAM_ROLE_ARN", //IAM_ROLE_ARN
    ScheduleExpression: "rate(5 minutes)",
    State: "ENABLED",
};

export const run = async () => {
    try {
        const data = await cweClient.send(new PutRuleCommand(params));
        console.log("Success, scheduled rule created; Rule ARN:", data);
        return data; // For unit tests.
    } catch (err) {
```

```
        console.log("Error", err);
    }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutRule](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({apiVersion: '2015-10-07'});

var params = {
    Name: 'DEMO_EVENT',
    RoleArn: 'IAM_ROLE_ARN',
    ScheduleExpression: 'rate(5 minutes)',
    State: 'ENABLED'
};

cwevents.putRule(params, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data.RuleArn);
    }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutRule](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Events with an AWS SDK \(p. 96\)](#). This topic also includes information about getting started and details about previous SDK versions.

Send CloudWatch Events events using an AWS SDK

The following code examples show how to send Amazon CloudWatch Events events.

Java

SDK for Java 2.x

```
public static void putCWEvents(CloudWatchEventsClient cwe, String resourceArn )
{
    try {
```

```
final String EVENT_DETAILS =
    "{ \"key1\": \"value1\", \"key2\": \"value2\" }";

PutEventsRequestEntry requestEntry = PutEventsRequestEntry.builder()
    .detail(EVENT_DETAILS)
    .detailType("sampleSubmitted")
    .resources(resourceArn)
    .source("aws-sdk-java-cloudwatch-example")
    .build();

PutEventsRequest request = PutEventsRequest.builder()
    .entries(requestEntry)
    .build();

cwe.putEvents(request);
System.out.println("Successfully put CloudWatch event");

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutEvents](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

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

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { cweClient } from "../libs/cloudWatchEventsClient.js";

// Set the parameters
export const params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: [
        "RESOURCE_ARN", //RESOURCE_ARN
      ],
      Source: "com.company.app",
    },
  ],
};
```

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

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutEvents](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({apiVersion: '2015-10-07'});

var params = {
  Entries: [
    {
      Detail: '{ \"key1\": \"value1\", \"key2\": \"value2\" }',
      DetailType: 'appRequestSubmitted',
      Resources: [
        'RESOURCE_ARN',
      ],
      Source: 'com.company.app'
    }
  ]
};

cwevents.putEvents(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutEvents](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch Events with an AWS SDK \(p. 96\)](#). This topic also includes information about getting started and details about previous SDK versions.

Security for Amazon CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

For CloudWatch Events security information, see [Security in Amazon EventBridge](#).

Tagging Your Amazon CloudWatch Events Resources

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

A *tag* is a custom attribute label that you assign or that AWS assigns to an AWS resource. Each tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to a CloudWatch Events rule that you assign to an EC2 instance.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see [Use Cost Allocation Tags](#) in the [AWS Billing User Guide](#).

The following sections provide more information about tags for CloudWatch Events.

Supported Resources in CloudWatch Events

The following resources in CloudWatch Events support tagging:

- Rules

For information about adding and managing tags, see [Managing Tags \(p. 105\)](#).

Managing Tags

Tags consist of the `Key` and `Value` properties on a resource. You can use the CloudWatch console, the AWS CLI, or the CloudWatch Events API to add, edit, or delete the values for these properties. For information about working with tags, see the following:

- [TagResource](#), [UntagResource](#), and [ListTagsForResource](#) in the *Amazon CloudWatch Events API Reference*
- [tag-resource](#), [untag-resource](#), and [list-tags-for-resource](#) in the *Amazon CloudWatch CLI Reference*

- [Working with Tag Editor](#) in the *Resource Groups User Guide*

Tag Naming and Usage Conventions

The following basic naming and usage conventions apply to using tags with CloudWatch Events resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: `. : * = @ _ / -` (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter` and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws :` prefix is prohibited for tags because it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags per resource quota.

Logging Amazon CloudWatch Events API Calls with AWS CloudTrail

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

Amazon CloudWatch Events is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in CloudWatch Events. CloudTrail captures API calls made by or on behalf of your AWS account. The calls captured include calls from the CloudWatch console and code calls to the CloudWatch Events API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for CloudWatch Events. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to CloudWatch Events, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Topics

- [CloudWatch Events Information in CloudTrail \(p. 107\)](#)
- [Example: CloudWatch Events Log File Entries \(p. 108\)](#)

CloudWatch Events Information in CloudTrail

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

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

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

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

- [DeleteRule](#)
- [DescribeEventBus](#)
- [DescribeRule](#)
- [DisableRule](#)
- [EnableRule](#)
- [ListRuleNamesByTarget](#)
- [ListRules](#)
- [ListTargetsByRule](#)
- [PutPermission](#)
- [PutRule](#)
- [PutTargets](#)
- [RemoveTargets](#)
- [TestEventPattern](#)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Example: CloudWatch Events Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following CloudTrail 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": "{\n  \"source\": [\"aws.ec2\"],\n  \"detail-type\": [\"EC2 Instance\nState-change Notification\"]\n}",
      "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"
  }
}
```

CloudWatch Events Quotas

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

For information on CloudWatch Events and EventBridge service quotas, see [Amazon EventBridge Quotas](#).

For more information, see the following.

- [Amazon EventBridge](#)
- [EventBridge Service Quotas](#)
- [Amazon EventBridge API Reference](#)

Troubleshooting CloudWatch Events

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

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

Topics

- [My rule was triggered but my Lambda function was not invoked \(p. 111\)](#)
- [I have just created/modified a rule but it did not match a test event \(p. 112\)](#)
- [My rule did not self-trigger at the time specified in the ScheduleExpression \(p. 113\)](#)
- [My rule did not trigger at the time that I expected \(p. 113\)](#)
- [My rule matches IAM API calls but my rule was not triggered \(p. 113\)](#)
- [My rule is not working because the IAM role associated with the rule is ignored when the rule is triggered \(p. 114\)](#)
- [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 \(p. 114\)](#)
- [My event's delivery to the target experienced a delay \(p. 114\)](#)
- [Some events were never delivered to my target \(p. 114\)](#)
- [My rule was triggered more than once in response to one event. What guarantee does CloudWatch Events offer for triggering rules or delivering events to the targets? \(p. 115\)](#)
- [Preventing Infinite Loops \(p. 115\)](#)
- [My events are not delivered to the target Amazon SQS queue \(p. 115\)](#)
- [My rule is being triggered but I don't see any messages published into my Amazon SNS topic \(p. 115\)](#)
- [My Amazon SNS topic still has permissions for CloudWatch Events even after I deleted the rule associated with the Amazon SNS topic \(p. 117\)](#)
- [Which IAM condition keys can I use with CloudWatch Events \(p. 117\)](#)
- [How can I tell when CloudWatch Events rules are broken \(p. 117\)](#)

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
```

Another reason the Lambda function would fail to trigger is if the policy you see when running `get-policy` contains a `SourceAccount` field. A `SourceAccount` setting prevents CloudWatch Events from being able to invoke the function.

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. 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 CloudWatch metrics for your rule such as `TriggeredRules`, `Invocations`, and `FailedInvocations` for further debugging. For more information about these metrics, see [Amazon CloudWatch Events Metrics and Dimensions](#) in the *Amazon CloudWatch User Guide*.

If the rule is triggered by an event from an AWS service, 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 \(p. 112\)](#).

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 [CloudWatch Events Event Examples From Supported Services \(p. 39\)](#).

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 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 the colon (:) or forward slash (/) 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 to match.

Moreover, not every event has the resources field populated (such as AWS API call events from CloudTrail).

My event's delivery to the target experienced a delay

CloudWatch Events tries to deliver an event to a target for up to 24 hours, except in scenarios where your target resource is constrained. The first attempt is made as soon as the event arrives in the event stream. However, if the target service is having problems, 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 CloudWatch. We recommend that you create a CloudWatch alarm on the `FailedInvocations` metric.

Some events were never delivered to my target

If a target of a CloudWatch Events rule is constrained for a prolonged time, CloudWatch Events may not retry delivery. For example, if the target is not provisioned to handle the incoming event traffic and the target service is throttling the requests that CloudWatch Events makes on your behalf, then CloudWatch Events may not retry delivery.

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

In rare cases, the same rule can be triggered more than once for a single event or scheduled time, or the same target can be invoked more than once for a given triggered rule.

Preventing Infinite Loops

In CloudWatch Events, it is possible to create rules that lead to infinite loops, where a rule is fired repeatedly. For example, a rule might detect that ACLs have changed on an S3 bucket, and trigger software to change them to the desired state. If the rule is not written carefully, the subsequent change to the ACLs fires the rule again, creating an infinite loop.

To prevent this, write the rules so that the triggered actions do not re-fire the same rule. For example, your rule could fire only if ACLs are found to be in a bad state, instead of after any change.

An infinite loop can quickly cause higher than expected charges. We recommend that you use budgeting, which alerts you when charges exceed your specified quota. For more information, see [Managing Your Costs with Budgets](#).

My events are not delivered to the target Amazon SQS queue

The Amazon SQS queue may be encrypted. If you create a rule with an encrypted Amazon SQS queue as a target, you must have the following section included in your KMS key policy for the event to be successfully delivered to the encrypted queue.

```
{
    "Sid": "Allow CWE to use the key",
    "Effect": "Allow",
    "Principal": {
        "Service": "events.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

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 attributes 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 the AWS CLI to set topic policy attribute.

Copy the current policy and add the following statement to the 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 earlier.

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. CloudWatch Events sets 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 [Amazon SNS set-topic-attributes](#) command.

Which IAM condition keys can I use with CloudWatch Events

CloudWatch Events supports the AWS-wide condition keys (see [Available Keys](#) in the *IAM User Guide*), plus the following service-specific condition keys.

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. Choose **Create Alarm**. In the **CloudWatch Metrics by Category** pane, choose **Events Metrics**.
3. In the list of metrics, select **FailedInvocations**.
4. Above the graph, choose **Statistic, Sum**.
5. For **Period**, choose a value, for example **5 minutes**. Choose **Next**.
6. Under **Alarm Threshold**, for **Name**, type a unique name for the alarm, for example **myFailedRules**. For **Description**, type a description of the alarm, for example **Rules are not delivering events to targets**.
7. For **is**, choose **>=** and **1**. For **for**, enter **10**.
8. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**.
9. For **Send notification to**, select an existing Amazon SNS topic or create a new one. To create a new topic, choose **New list**. Type a name for the new Amazon SNS topic, for example: **myFailedRules**.
10. For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state.

11. Choose **Create Alarm**.

Document History

Note

Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console. For more information, see [Amazon EventBridge](#).

The following table describes important changes in each release of the CloudWatch Events User Guide, beginning in June 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

| update-history-change | update-history-description | update-history-date |
|---|---|---------------------|
| Support for tagging (p. 119) | You can now tag some CloudWatch Events resources. For more information, see Tagging Your Amazon CloudWatch Events Resources in the <i>Amazon CloudWatch Events User Guide</i> . | March 21, 2019 |
| Support for Amazon VPC endpoints (p. 119) | You can now establish a private connection between your VPC and CloudWatch Events. For more information, see Using CloudWatch Events with Interface VPC Endpoints in the <i>Amazon CloudWatch Events User Guide</i> . | June 28, 2018 |

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

| Change | Description | Release Date |
|----------------------------------|--|-------------------|
| CodeBuild as a target | Added CodeBuild as a target for event rules. For more information, see Tutorial: Schedule Automated Builds Using CodeBuild (p. 28) . | 13 December 2017 |
| AWS Batch as a target | Added AWS Batch as a target for Event rules. For more information, see AWS Batch Events . | September 8, 2017 |
| CodePipeline and AWS Glue events | Added support for events from CodePipeline and AWS Glue. For more information, see CodePipeline Events (p. 42) and AWS Glue Events (p. 54) . | September 8, 2017 |
| CodeBuild and CodeCommit events | Added support for events from CodeBuild and CodeCommit. For more information, see CodeBuild Events (p. 41) . | August 3, 2017 |

| Change | Description | Release Date |
|---|---|-------------------|
| Additional targets supported | CodePipeline and Amazon Inspector can be targets of events. | June 29, 2017 |
| Support for sending and receiving events between AWS accounts | An AWS account can send events to another AWS account. For more information, see Sending and Receiving Events Between AWS Accounts (p. 79) . | June 29, 2017 |
| 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. 5) . | March 7, 2017 |
| Amazon EMR events | Added support for events for Amazon EMR. For more information, see Amazon EMR Events (p. 45) . | March 7, 2017 |
| AWS Health events | Added support for events for AWS Health. For more information, see AWS Health Events (p. 59) . | December 1, 2016 |
| Amazon Elastic Container Service events | Added support for events for Amazon ECS. For more information, see Amazon Elastic Container Service Events (p. 45) . | November 21, 2016 |
| AWS Trusted Advisor events | Added support for events for Trusted Advisor. For more information, see AWS Trusted Advisor Events (p. 75) . | November 18, 2016 |
| Amazon Elastic Block Store events | Added support for events for Amazon EBS. For more information, see Amazon EBS Events (p. 43) . | November 14, 2016 |
| AWS CodeDeploy events | Added support for events for CodeDeploy. For more information, see AWS CodeDeploy Events (p. 41) . | September 9, 2016 |
| Scheduled events with 1 minute granularity | Added support for scheduled events with 1 minute granularity. For more information, see Cron Expressions (p. 30) and Rate Expressions (p. 32) . | April 19, 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) . | March 30, 2016 |
| Auto Scaling events | Added support for events for Auto Scaling lifecycle hooks. For more information, see Amazon EC2 Auto Scaling Events (p. 44) . | February 24, 2016 |
| New service | Initial release of CloudWatch Events. | January 14, 2016 |

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.