# Table of Contents

What Is Amazon EventBridge? ................................................................. 1  
CloudWatch Events ............................................................................. 2  
Setup and prerequisites .................................................................... 4  
  Sign up for an AWS account .............................................................. 4  
  Create an administrative user ............................................................ 4  
  Sign in to the Amazon EventBridge console ........................................... 5  
Account credentials ........................................................................... 5  
Set up the AWS Command Line Interface ............................................. 6  
Regional Endpoints ........................................................................... 6  
Getting started ................................................................................... 7  
Create rule ......................................................................................... 7  
Event Bus .............................................................................................. 9  
  How event buses work ..................................................................... 9  
Event bus concepts ............................................................................ 12  
  Event buses .................................................................................... 12  
  Events ............................................................................................. 15  
  Event sources ................................................................................ 15  
  Rules .............................................................................................. 15  
  Targets ........................................................................................... 16  
  Advanced features ......................................................................... 16  
Permissions for event buses ................................................................. 18  
  Managing event bus permissions ....................................................... 18  
  Example policy: Send events to the default bus in a different account ........................................................................... 20  
  Example policy: Send events to a custom bus in a different account ...................................................................................... 20  
  Example policy: Send events to an event bus in the same account ...................................................................................... 21  
  Example policy: Send events to the same account and restrict updates ........................................................................... 21  
  Example policy: Send events only from a specific rule to the bus in a different Region ...................................................... 22  
  Example policy: Send events only from a specific Region to a different Region ................................................................. 22  
  Example policy: Deny sending events from specific Regions .............................................................................................. 23  
Creating an event bus .......................................................................... 24  
Generating a template from an event bus .............................................. 25  
  Considerations when using a generated template ........................... 25  
Events .................................................................................................. 27  
  Event structure reference ................................................................. 29  
  Minimum valid custom event ............................................................ 30  
Adding events with PutEvents ............................................................. 31  
  Handling failures with PutEvents ...................................................... 32  
  Sending events using the AWS CLI .................................................... 33  
  Calculating event entry size ............................................................. 35  
Events from AWS services ................................................................. 35  
  Service event delivery ................................................................... 36  
  Events via CloudTrail ...................................................................... 36  
  Services that generate events ........................................................... 37  
Receiving events from a SaaS partner ................................................. 42  
  Supported SaaS partner integrations .................................................. 43  
  Configuring EventBridge ................................................................. 45  
  Create a rule for SaaS partner events ................................................. 46  
  Receiving events using Lambda function URLs ............................... 47  
  Receiving events from Salesforce ...................................................... 54  
Debugging event delivery ................................................................... 57  
  Using dead-letter queues ............................................................... 58  
Event patterns .................................................................................... 62  
Creating event patterns .................................................................... 63  
Matching event values ....................................................................... 63
Considerations when creating event patterns .................................................. 64
Comparison operations for use in event patterns ........................................... 65
Example events and event patterns .................................................................. 66
   Field matching ......................................................................................... 66
   Value matching ....................................................................................... 67
Null values and empty strings .......................................................................... 68
Arrays ........................................................................................................... 69
Content-based filtering ................................................................................... 70
   Prefix matching ....................................................................................... 70
   Suffix matching ....................................................................................... 70
   Anything-but matching ........................................................................... 71
   Numeric matching ................................................................................... 72
   IP address matching ................................................................................ 72
   Exists matching ...................................................................................... 72
   Equals-ignore-case matching .................................................................. 73
   Matching using wildcards ....................................................................... 73
   Complex example with multiple matching ................................................. 74
   Complex example with $\sigma$ matching .................................................... 75
Testing an event pattern .................................................................................. 76
Best practices .................................................................................................. 78
   Avoid writing infinite loops ..................................................................... 78
   Make event patterns precise as possible ................................................ 79
   Scope your event patterns to account for event source updates ............... 80
   Validate event patterns .......................................................................... 81
Rules .............................................................................................................. 82
   Creating a rule that reacts to events .......................................................... 84
     Create a rule that reacts to events ......................................................... 84
   Using EventBridge Scheduler ................................................................... 92
     Set up the execution role ....................................................................... 92
     Create a schedule .................................................................................. 92
     Related resources .................................................................................. 95
   Creating a rule that runs on a schedule ...................................................... 95
     Create a rule that runs on a schedule ................................................... 97
     Cron expressions ................................................................................... 102
     Rate expressions ................................................................................... 104
Disabling or deleting a rule .............................................................................. 106
Best practices .................................................................................................. 106
   Set a single target for each rule ............................................................... 106
   Set rule permissions ................................................................................. 106
   Monitor rule performance ........................................................................ 106
Using AWS SAM templates ........................................................................... 108
   Combined template .................................................................................. 108
   Separated template .................................................................................. 109
Generating a template from a rule .................................................................. 110
   Considerations when using a generated template .................................... 110
Targets ............................................................................................................ 112
   Targets available in the EventBridge console .......................................... 112
   Target parameters ................................................................................... 113
     Dynamic path parameters ..................................................................... 114
Permissions .................................................................................................... 114
EventBridge target specifics .......................................................................... 114
   AWS Batch job queues ............................................................................ 114
   CloudWatch Logs group .......................................................................... 115
   CodeBuild project ................................................................................... 115
   Amazon ECS task .................................................................................... 115
   Incident ECS task .................................................................................... 115
Configure targets ............................................................................................ 116
Troubleshooting

My rule ran but my Lambda function wasn't invoked
I just created or modified a rule, but it didn't match a test event
My rule didn't run at the time I specified in the ScheduleExpression
My rule didn't run at the time that I expected
My rule matches AWS global service API calls but it didn't run
The IAM role associated with my rule is being ignored when the rule runs
My rule has an event pattern that is supposed to match a resource, but no events match
My event's delivery to the target was delayed
Some events were never delivered to my target
My rule ran more than once in response to one event
Preventing infinite loops
My events are not delivered to the target Amazon SQS queue
My rule runs, but I don't see any messages published into my Amazon SNS topic
My Amazon SNS topic still has permissions for EventBridge even after I deleted the rule associated with the Amazon SNS topic
Which IAM condition keys can I use with EventBridge?
How can I tell when EventBridge rules are broken?
What Is Amazon EventBridge?

EventBridge is a serverless service that uses events to connect application components together, making it easier for you to build scalable event-driven applications. Event-driven architecture is a style of building loosely-coupled software systems that work together by emitting and responding to events. Event-driven architecture can help you boost agility and build reliable, scalable applications.

Use EventBridge to route events from sources such as home-grown applications, AWS services, and third-party software to consumer applications across your organization. EventBridge provides simple and consistent ways to ingest, filter, transform, and deliver events so you can build applications quickly.

The following video provides a brief introduction to the features of Amazon EventBridge:

EventBridge includes two ways to process events: event buses and pipes.

- Event buses (p. 9) are routers that receive events (p. 27) and delivers them to zero or more targets. Event buses are well-suited for routing events from many sources to many targets, with optional transformation of events prior to delivery to a target.

The following video provides a high-level overview of event buses:
• **Pipes (p. 150)** EventBridge Pipes is intended for point-to-point integrations; each pipe receives events from a single source for processing and delivery to a single target. Pipes also include support for advanced transformations and enrichment of events prior to delivery to a target.

Pipes and event buses are often used together. A common use case is to create a pipe with an event bus as its target; the pipe sends events to the event bus, which then sends those events on to multiple targets. For example, you could create a pipe with a DynamoDB stream for a source, and an event bus as the target. The pipe receives events from the DynamoDB stream and sends them to the event bus, which then sends them on to multiple targets according to the rules you’ve specified on the event bus.

**EventBridge is the evolution of Amazon CloudWatch Events**

EventBridge was formerly called Amazon CloudWatch Events. The default event bus and the rules you created in CloudWatch Events also display in the EventBridge console. EventBridge uses the same CloudWatch Events API, so your code that uses the CloudWatch Events API stays the same.

EventBridge builds on the capabilities of CloudWatch Events with features such as partner events, Schema Registry, and EventBridge Pipes. New features added to EventBridge are not added to CloudWatch Events. For more information, see ??? (p. 1).

All the features you’re used to in CloudWatch Events are also present in EventBridge, including:

• ??? (p. 9)
• ??? (p. 82)
• ??? (p. 27)
EventBridge features that build on and expand the capabilities of events include:

- ?? (p. 35)
- ?? (p. 42)
- ?? (p. 150)
- ?? (p. 227)
- ?? (p. 92)
Amazon EventBridge setup and prerequisites

To use Amazon EventBridge, you need an AWS account. Your account allows you to use services such as Amazon EC2 to generate events that you can see in the EventBridge console. You can also install and configure the AWS Command Line Interface (AWS CLI) to use a command-line interface to see events.

Topics
- Sign up for an AWS account (p. 4)
- Create an administrative user (p. 4)
- Sign in to the Amazon EventBridge console (p. 5)
- Account credentials (p. 5)
- Set up the AWS Command Line Interface (p. 6)
- Regional Endpoints (p. 6)

Sign up for an AWS account

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

To sign up for an AWS account

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.

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

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

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing Root user and entering your AWS account email address. On the next page, enter your password.
For help signing in by using root user, see [Signing in as the root user](https://aws.amazon.com/privateinfo/signin/) in the AWS Sign-In User Guide.

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

For instructions, see [Enable a virtual MFA device for your AWS account root user (console)](https://docs.aws.amazon.com/identity-management/latest/userguide/iam-enable-virtual-mfa-device.html) in the IAM User Guide.

Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center.

For instructions, see [Getting started](https://docs.aws.amazon.com/identity-management/latest/userguide/iam-getting-started.html) in the AWS IAM Identity Center User Guide.

Sign in as the administrative user

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

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](https://aws.amazon.com/privateinfo/signin/) in the AWS Sign-In User Guide.

Sign in to the Amazon EventBridge console

To sign in to the Amazon EventBridge console

- Sign in to the AWS Management Console and open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).

Account credentials

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

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [ "events:*" ],
            "Effect": "Allow",
            "Resource": "arn:aws:events:*:*:*"
        },
        {
            "Action": [ "iam:PassRole" ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "events.amazonaws.com"
                }
            }
        }
    ]
}
```
Set up the AWS Command Line Interface

You can use the AWS CLI to perform EventBridge 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 the default regional endpoints to use EventBridge. For more information, see Activating and Deactivating AWS STS in an AWS Region in the IAM User Guide.
Getting started with Amazon EventBridge

The basis of EventBridge is to create rules (p. 82) that route events (p. 27) to a target (p. 112). In this section, you create a basic rule. For tutorials about specific scenarios and specific targets, see Amazon EventBridge tutorials (p. 242).

Create a rule in Amazon EventBridge

To create a rule for events, you specify an action to take when EventBridge receives an event that matches the event pattern in the rule. When an event matches, EventBridge sends the event to the specified target and triggers the action defined in the rule.

When an AWS service in your AWS account emits an event, it always goes to the default event bus (p. 9) for your account. To write a rule that matches events from AWS services in your account, you must associate it with the default event bus.

To create a rule for an AWS service

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule.
   A rule can’t have the same name as another rule in the same Region and on the same event bus.
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select AWS default event bus. When an AWS service in your account emits an event, it always goes to your account’s default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose AWS services.
9. (Optional) For Sample events, choose the type of event.
10. For Event pattern, do one of the following:
    • To use a template to create your event pattern, choose Event pattern form and choose the Event source and Event type. If you choose All Events as the event type, all events emitted by this AWS service will match the rule.
    • To customize the template, choose Custom pattern (JSON editor) and make your changes.
11. Choose Next.
12. For Target types, choose AWS service.
13. For Select a target, choose the AWS service that you want to send information to when EventBridge detects an event that matches the event pattern.
14. The fields displayed vary depending on the service you choose. Enter information specific to this target type as needed.
15. For many target types, EventBridge needs permissions to send events to the target. In these cases, EventBridge can create the IAM role needed for your rule to run. Do one of the following:

- To create an IAM role automatically, choose Create a new role for this specific resource.
- To use an IAM role that you created earlier, choose Use existing role and select the existing role from the drop-down list.

16. (Optional) For Additional settings, do the following:

   a. For Maximum age of event, enter a value between one minute (00:01) and 24 hours (24:00).
   b. For Retry attempts, enter a number between 0 and 185.
   c. For Dead-letter queue, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
      - Choose None to not use a dead-letter queue.
      - Choose Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue and then select the queue to use from the drop-down list.
      - Choose Select an Amazon SQS queue in an other AWS account as a dead-letter queue and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it. For more information, see Granting permissions to the dead-letter queue (p. 60).

17. (Optional) Choose Add another target to add another target for this rule.

18. Choose Next.

19. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 453).

20. Choose Next.

21. Review the details of the rule and choose Create rule.
Amazon EventBridge Event Bus

An event bus is a router that receives events (p. 27) and delivers them to zero or more destinations, or targets. Event buses are well-suited for routing events from many sources to many targets, with optional transformation of events prior to delivery to a target.

Rules (p. 82) associated with the event bus evaluate events as they arrive. Each rule checks whether an event matches the rule's pattern. If the event does match, EventBridge sends the event.

You associate a rule with a specific event bus, so the rule only applies to events received by that event bus.

Note
You can also process events using EventBridge Pipes. EventBridge Pipes is intended for point-to-point integrations; each pipe receives events from a single source for processing and delivery to a single target. Pipes also include support for advanced transformations and enrichment of events prior to delivery to a target. For more information, see ??? (p. 150).

Topics
- How event buses work (p. 9)
- Amazon EventBridge Event Bus concepts (p. 12)
- Permissions for Amazon EventBridge event buses (p. 18)
- Creating an Amazon EventBridge event bus (p. 24)
- Generate an AWS CloudFormation template from an Amazon EventBridge event bus (p. 25)

How event buses work

Event buses enable you to route events from multiple sources to multiple destinations, or targets.

At a high level, here's how it works:
1. An event source, which can be an AWS service, your own custom application, or a SaaS provider, sends an event to an event bus.

2. EventBridge then evaluates the event against each rule defined for that event bus.

   For each event that matches a rule, EventBridge then sends the event to the targets specified for that rule. Optionally, as part of the rule, you can also specify how EventBridge should transform the event prior to sending it to the target(s).

   An event might match multiple rules, and each rule can specify up to five targets. (An event may not match any rules, in which case EventBridge takes no action.)

Consider an example using the EventBridge default event bus, which automatically receives events from AWS services:

1. You create a rule on the default event bus for the EC2 Instance State-change Notification event:
   - You specify that the rule matches events where an Amazon EC2 instance has changed its state to running.

   You do this by specifying JSON that defines the attributes and values an event must match to trigger the rule. This is called an event pattern.
How event buses work

- You specify the target of the rule to be a given Lambda function.

2. Whenever an Amazon EC2 instance changes state, Amazon EC2 (the event source) automatically sends that event to the default event bus.

3. EventBridge evaluates all events sent to the default event bus against the rule you've created.

   If the event matches your rule (that is, if the event was an Amazon EC2 instance changing state to running), EventBridge sends the event to the specified target. In this case, that's the Lambda function.

The following video describes what event buses are and what they do: [What are event buses](#)
The following video covers the different event buses and when to use them: [The differences between event buses](#).

Amazon EventBridge Event Bus concepts

Here's a closer look at the main components of an event driven architecture built on event buses.

**Event buses**

An event bus is a router that receives [events](#) and delivers them to zero or more destinations, or targets. Use an event bus when you need to route events from many sources to many targets, with optional transformation of events prior to delivery to a target.

Your account includes a *default event bus* that automatically receives events from AWS services. You can also:

- Create additional event buses, called *custom event buses*, and specify which events they receive.
- Create [partner event buses](#), which receive events from SaaS partners.

Common use cases for event buses include:

- Using an event bus as a broker between different workloads, services, or systems.
- Using multiple event buses in your applications to divide up the event traffic. For example, creating a bus to process events containing personal identification information (PII), and another bus for events that don't.
• Aggregating events by sending events from multiple event buses to a centralized event bus. This centralized bus can be in the same account as the other buses, but can also be in a different account or Region.
Events

At its simplest, an EventBridge event is a JSON object sent to an event bus or pipe.

In the context of event-driven architecture (EDA), an event often represents an indicator of a change in a resource or environment.

For more information, see ???(p. 27).

Event sources

EventBridge can receive events from event sources including:

- AWS services
- Custom applications
- Software as a service (SaaS) partners

Rules

A rule receives incoming events and sends them as appropriate to targets for processing. You can specify how each rule invokes their target(s) based on either:

- An event pattern (p. 62), which contains one or more filters to match events. Event patterns can include filters that match on:
  - Event metadata – Data about the event, such as the event source, or the account or Region in which the event originated.
  - Event data – The properties of the event itself. These properties vary according to event.
  - Event content – The actual property values of the event data.
- A schedule to invoke the target(s) at regular intervals.

You can specify a scheduled rule within EventBridge (p. 95), or by using EventBridge Scheduler (p. 92).

Note

EventBridge offers Amazon EventBridge Scheduler, a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. EventBridge Scheduler is highly customizable, and offers improved scalability over EventBridge scheduled rules, with a wider set of target API operations and AWS services. We recommend that you use EventBridge Scheduler to invoke targets on a schedule. For more information, see ???(p. 92).

Each rule is defined for a specific event bus, and only apply to events on that event bus.

A single rule can send an event to up to five targets.

By default, you can configure up to 300 rules per event bus. This quota can be raised to thousands of rules in the Service Quotas console. Since the rule limit apply to each bus, if you require even more rules, you can create additional custom event buses in your account.

You can customize how events are received in your account by creating event buses with different permissions for different services.

To customize the structure or date of an event before EventBridge passes it to a target, use the input transformer (p. 137) to edit the information before it goes to the target.
Targets

A target is a resource or endpoint to which EventBridge sends an event when the event matches the event pattern defined for a rule.

A target can receive multiple events from multiple event buses.

For more information, see ???(p. 82).

Advanced features for event buses

EventBridge includes the following features to help you develop, manage, and use event buses.

Using API destinations to enable REST API calls between services

EventBridge API destinations (p. 117) are HTTP endpoints that you can set as the target of a rule, in the same way that you would send event data to an AWS service or resource. By using API destinations, you can use API calls to route events between AWS services, integrated SaaS applications, and your applications outside of AWS. When you create an API destination, you specify a connection to use for it. Each connection includes the details about the authorization type and parameters to use to authorize with the API destination endpoint.

Archiving and replaying events to aid development and disaster recovery

You can archive (p. 147), or save, events and then replay (p. 149) them at a later time from the archive. Archiving is useful for:

- Testing an application because you have a store of events to use rather than having to wait for new events.
- Hydrating a new service when it first comes online.
- Adding more durability to your event-driven applications.

Using the Schema Registry to jump-start event pattern creation

When you build serverless applications that use EventBridge, it can be helpful to know the structure of typical events without having to generate the event. The event structure are described in schemas (p. 227), which are available for all events generated by AWS services on EventBridge.

For events that don't come from AWS services, you can:

- Create or upload custom schemas.
- Use Schema Discovery to have EventBridge automatically create schemas for events sent to the event bus.

Once you have a schema for an event, you can download code bindings for popular programming languages.

Managing resources and access with policies

To organize AWS resources or to track costs in EventBridge, you can assign a custom label, or tag (p. 453), to AWS resources. Using tag-based policies (p. 387), you can control what resources can and can't do within EventBridge.

In addition to tag-based policies, EventBridge supports identity-based (p. 394) and resource-based (p. 405) policies to control access to EventBridge. Use identity-based policies to control the
permissions of a group, role, or user. Use resource-based policies to give specific permissions to each resource, such as a Lambda function or Amazon SNS topic.
Permissions for Amazon EventBridge event buses

The default event bus (p. 9) in your AWS account only allows events (p. 27) from one account. You can grant additional permissions to an event bus by attaching a resource-based policy (p. 405) to it. With a resource-based policy, you can allow PutEvents, PutRule, and PutTargets API calls from another account. You can also use IAM conditions (p. 416) in the policy to grant permissions to an organization, apply tags (p. 453), or filter events to only those from a specific rule or account. You can set a resource-based policy for an event bus when you create it or afterward.

EventBridge APIs that accept an event bus Name parameter such as PutRule, PutTargets, DeleteRule, RemoveTargets, DisableRule, and EnableRule also accept the event bus ARN. Use these parameters to reference cross-account or cross-Region event buses through the APIs. For example, you can call PutRule to create a rule (p. 82) on an event bus in a different account without needing to assume a role.

You can attach the example policies in this topic to an IAM role to grant permission to send events to a different account or Region. Use IAM roles to set organization control policies and boundaries on who can send events from your account to other accounts. We recommend always using IAM roles when the target of a rule is an event bus. You can attach IAM roles using PutTarget calls. For information about creating a rule to send events to a different account or Region, see Sending and receiving Amazon EventBridge events between AWS accounts (p. 131).

Topics
- Managing event bus permissions (p. 18)
- Example policy: Send events to the default bus in a different account (p. 20)
- Example policy: Send events to a custom bus in a different account (p. 20)
- Example policy: Send events to an event bus in the same account (p. 21)
- Example policy: Send events to the same account and restrict updates (p. 21)
- Example policy: Send events only from a specific rule to the bus in a different Region (p. 22)
- Example policy: Send events only from a specific Region to a different Region (p. 22)
- Example policy: Deny sending events from specific Regions (p. 23)

Managing event bus permissions

Use the following procedure to modify the permissions for an existing event bus. For information about how to use AWS CloudFormation to create an event bus policy, see AWS::Events::EventBusPolicy.

To manage permissions for an existing event bus
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the left navigation pane, choose Event buses.
3. In Name, choose the name of the event bus to manage permissions for.
   
   If a resource policy is attached to the event bus, the policy displays.
4. Choose Manage permissions, and then do one of the following:
   • Enter the policy that includes the permissions to grant for the event bus. You can paste in a policy from another source, or enter the JSON for the policy.
   • To use a template for the policy, choose Load template. Modify the policy as appropriate for your environment, and add additional actions that you authorize the principal in the policy to use.
5. Choose Update.
The template provides example policy statements that you can customize for your account and environment. The template isn't a valid policy. You can modify the template for your use case, or you can copy one of the example policies and customize it.

The template loads policies that include an example of how to grant permissions to an account to use the PutEvents action, how to grant permissions to an organization, and how to grant permissions to the account to manage rules in the account. You can customize the template for your specific account, and then delete the other sections from the template. More example policies are included later in this topic.

If you try to update the permissions for the bus but the policy contains an error, an error message indicates the specific issue in the policy.

### Choose which sections to include in the policy to match your use case. ###
### Be sure to remove all lines that start with ###, including the ### at the end of the line. ###
### The policy must include the following: ###

```json
{
    "Version": "2012-10-17",
    "Statement": [
        ### To grant permissions for an account to use the PutEvents action, include the following, otherwise delete this section: ###

        {
            "Sid": "AllowAccountToPutEvents",
            "Effect": "Allow",
            "Principal": {
                "AWS": "<ACCOUNT_ID>"
            },
            "Action": "events:PutEvents",
            "Resource": "arn:aws:events:us-east-1:123456789012:event-bus/default"
        },

        ### Include the following section to grant permissions to all members of your AWS Organizations to use the PutEvents action ###

        {
            "Sid": "AllowAllAccountsFromOrganizationToPutEvents",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "events:PutEvents",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-yourOrgID"
                }
            }
        },

        ### Include the following section to grant permissions to the account to manage the rules created in the account ###

        {
            "Sid": "AllowAccountToManageRulesTheyCreated",
            "Effect": "Allow",
            "Principal": {
                "AWS": "<ACCOUNT_ID>"
            },
            "Action": [
```
Example policy: Send events to the default bus in a different account

The following example policy grants the account 111122223333 permission to publish events to the default event bus in the account 123456789012.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "sid1",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111112222333:root"},
      "Action": ["events:PutEvents"],
      "Resource": "arn:aws:events:us-east-1:123456789012:event-bus/default"
    }
  ]
}
```

Example policy: Send events to a custom bus in a different account

The following example policy grants the account 111122223333 permission to publish events to the central-event-bus in account 123456789012, but only for events with a source value set to com.exampleCorp.webStore and a detail-type set to newOrderCreated.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {"Sid": "WebStoreCrossAccountPublish",
     "Effect": "Allow",
     "Action": ["events:PutEvents"],
     "Principal": {"AWS": "arn:aws:iam::111112222333:root"},
     "Resource": "arn:aws:events:us-east-1:123456789012:event-bus/central-event-bus"
    }
  ]
}
```
Example policy: Send events to an event bus in the same account

The following example policy attached to an event bus named CustomBus1 allows the event bus to receive events from the same account and Region.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "arn:aws:events:us-east-1:123456789012:event-bus/CustomBus1"
            ]
        }
    ]
}
```

Example policy: Send events to the same account and restrict updates

The following example policy grants account 123456789012 permission to create, delete, update, disable and enable rules, and add or remove targets. It limits these rules that match against events with a source of com.exampleCorp.webStore, and it uses the "events:creatorAccount": "$[aws:PrincipalAccount]" to ensure that only account 123456789012 can modify these rules and targets once they have been created.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "InvoiceProcessingRuleCreation",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": [
                "events:PutRule",
                "events:DeleteRule",
                "events:DescribeRule",
                "events:DisableRule",
                "events:EnableRule",
                "events:PutTargets",
                "events:RemoveTargets"
            ]
        }
    ]
}
```
Example policy: Send events only from a specific rule to the bus in a different Region

The following example policy grants the account 111122223333 permission to send events that match a rule named SendToUSE1AnotherAccount in the Middle East (Bahrain) and US West (Oregon) Regions to an event bus named CrossRegionBus in the US East (N. Virginia) in account 123456789012. The example policy is added to the event bus named CrossRegionBus in account 123456789012. The policy allows events only if they match the rules with the specified rule ARN.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSpecificRulesAsCrossRegionSource",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111112222333:root"
      },
      "Action": "events:PutEvents",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": [
            "arn:aws:events:us-west-2:111112222333:rule/CrossRegionBus/SendToUSE1AnotherAccount",
            "arn:aws:events:me-south-1:111112222333:rule/CrossRegionBus/SendToUSE1AnotherAccount"
          ]
        }
      }
    }
  ]
}
```

Example policy: Send events only from a specific Region to a different Region

The following example policy grants account 111122223333 permission to send all events that are generated in the Middle East (Bahrain) and US West (Oregon) Regions to the event bus named CrossRegionBus in account 123456789012 in the US East (N. Virginia) Region. Account 111122223333 doesn't have permission to send events that are generated in any other Region.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSpecificRulesAsCrossRegionSource",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111112222333:root"
      },
      "Action": "events:PutEvents",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": [
            "arn:aws:events:us-west-2:111112222333:rule/CrossRegionBus/SendToUSE1AnotherAccount",
            "arn:aws:events:me-south-1:111112222333:rule/CrossRegionBus/SendToUSE1AnotherAccount"
          ]
        }
      }
    }
  ]
}
```
Example policy: Deny sending events from specific Regions

The following example policy attached to an event bus named CrossRegionBus in account 123456789012 grants permission for the event bus to receive events from the account 111122223333, but not events that are generated in the US West (Oregon) Region.

```json
[  
  "Version": "2012-10-17",  
  "Statement": [  
    
    
    "Sid": "1AllowAnyEventsFromAccount111112222333",  
    "Effect": "Allow",  
    "Principal": {  
      "AWS": "arn:aws:iam::111112222333:root"  
    },  
    "Action": "events:PutEvents",  
    "Resource": "arn:aws:events:us-east-1:123456789012:event-bus/CrossRegionBus"
  ],  
  
  "Sid": "2DenyAllCrossRegionUSWest2Events",  
  "Effect": "Deny",  
  "Principal": {  
    "AWS": "*"
  },  
  "Action": "events:PutEvents",  
  "Condition": {  
    "ArnEquals": {  
      "aws:SourceArn": [  
        "arn:aws:events:us-west-2:*:*",
        "arn:aws:events:me-south-1:*:*"
      ]
    }
  }
]
```
Creating an Amazon EventBridge event bus

You can create a custom event bus (p. 9) to receive events (p. 27) from your applications. Your applications can also send events to the default event bus. When you create an event bus, you can attach a resource-based policy (p. 405) to grant permissions to other accounts. Then other accounts can send events to the event bus in the current account.

The following video goes through creating event buses: Creating an event bus

To create a custom event bus

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Event buses.
3. Choose Create event bus.
4. Enter a name for the new event bus.
5. Do one of the following:
   • Enter the policy that includes the permissions to grant for the event bus. You can paste in a policy from another source or enter the JSON for the policy. You can use one of the example policies and modify it for your environment.
   • To use a template for the policy, choose Load template. Modify the policy as appropriate for your environment, including adding additional actions that you authorize the principal in the policy to use.
6. Choose Create.
Generate an AWS CloudFormation template from an Amazon EventBridge event bus

AWS CloudFormation enables you to configure and manage your AWS resources across accounts and regions in a centralized and repeatable manner by treating infrastructure as code. AWS CloudFormation does this by letting you create templates, which define the resources you want to provision and manage.

EventBridge enables you to generate templates from the existing event buses in your account, as an aid to help you jumpstart developing AWS CloudFormation templates. In addition, EventBridge provides the option of including the rules associated with that event bus in your template. You can then use these templates as the basis for creating stacks of resources under AWS CloudFormation management.

For more information on AWS CloudFormation see The AWS CloudFormation User Guide.

Note
EventBridge does not include managed rules in the generated template.

You can also generate a template from one or more rules contained in a selected event bus.

To generate an AWS CloudFormation template from an event bus

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Event buses.
3. Choose the event bus from which you want to generate a AWS CloudFormation template.
4. From the Actions menu, choose CloudFormation Template, and then choose which format you want EventBridge to generate the template in: JSON or YAML.

   EventBridge displays the template, generated in the selected format. By default, all rules associated with the event bus are included in the template.

   • To generate the template without including rules, deselect Include rules on this EventBus.
5. EventBridge gives you the option of downloading the template file, or copying the template to the clipboard.

   • To download the template file, choose Download.
   • To copy the template to the clipboard, choose Copy.
6. To exit the template, choose Cancel.

Once you’ve customized your AWS CloudFormation template as necessary for your use case, you can use it to create stacks in AWS CloudFormation.

Considerations when using AWS CloudFormation templates generated from Amazon EventBridge

Consider the following factors when using a AWS CloudFormation template you generated from an event bus:

• EventBridge does not include any passwords in the generate template.

   You can edit the template to include template parameters that enable users to specify passwords or other sensitive information when using the template to create or update a AWS CloudFormation stack.

   In addition, users can use Secrets Manager to create a secret in the desired region and then edit the generated template to employ dynamic parameters.
• Targets in the generated template remain exactly as they were specified in the original event bus. This can lead to cross-region issues if you do not appropriately edit the template before using it to create stacks in other regions.

   Additionally, the generated template will not create the downstream targets automatically.
Amazon EventBridge events

An event indicates a change in an environment such as an AWS environment, a SaaS partner service or application, or one of your applications or services. The following are examples of events:

- Amazon EC2 generates an event when the state of an instance changes from pending to running.
- Amazon EC2 Auto Scaling generates events when it launches or terminates instances.
- AWS CloudTrail publishes events when you make API calls.

You can also set up scheduled events that are generated on a periodic basis.

For a list of services that generate events, including sample events from each service, see Events from AWS services (p. 35) and follow the links in the table.

Events are represented as JSON objects and they all have a similar structure, and the same top-level fields.

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 Events from AWS services (p. 35).

Topics
- Event structure reference (p. 29)
- Adding Amazon EventBridge events with PutEvents (p. 31)
- Events from AWS services (p. 35)
- Receiving events from a SaaS partner with Amazon EventBridge (p. 42)
- Debugging Amazon EventBridge event delivery (p. 57)
The following video explains the basics of events: What is an event

The following video covers the ways events get to EventBridge: Where do events come from
Event structure reference

The following fields appear in an event:

```
{
    "version": "0",
    "id": "UUID",
    "detail-type": "event name",
    "source": "event source",
    "account": "ARN",
    "time": "timestamp",
    "region": "region",
    "resources": [
        "ARN"
    ],
    "detail": {
        "JSON object"
    }
}
```

version

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

id

A Version 4 UUID that’s generated for every event. You can use id to trace events as they move through rules to targets.

detail-type

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

Events that are delivered by CloudTrail have AWS API Call via CloudTrail as the value for detail-type.

source

Identifies the service that generated the event. All events that come from AWS services 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 condition keys table, select a service from the list, and look for the service prefix. 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 can report the start time, so this value might be before the time the event is received.

region

Identifies the AWS Region where the event originated.

resources

A JSON array that contains ARNs that identify resources that are involved in the event. The service generating the event determines whether to include these ARNs. For example, Amazon EC2 instance
Minimum valid custom event

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 that contains information about the event. The service generating the event determines the content of this field. The detail content can be as simple as two fields. AWS API call events have detail objects with approximately 50 fields nested several levels deep.

Example Example: Amazon EC2 instance state-change notification

The following event in Amazon EventBridge indicates an Amazon EC2 instance being terminated.

```
{
  "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"
  }
}
```

Minimum information needed for a valid custom event

When you create custom events they must include the following fields:

```
{
  "detail-type": "event name",
  "source": "event source",
  "detail": {}
}
```

- detail – A JSON object that contains information about the event. It can be "{}".

  Note

  PutEvents accepts data in JSON format. For the JSON number (integer) data type, the constraints are: a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807.

- detail-type – A string that identifies the type of event.

- source – A string that identifies the source of the event. 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.
Adding Amazon EventBridge events with PutEvents

The PutEvents action sends multiple events to EventBridge in a single request. For more information, see PutEvents in the Amazon EventBridge 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 Amazon EventBridge quotas. The PutEvents operation attempts to process all entries in the natural order of the request. After you call PutEvents, EventBridge assigns each event a unique ID.

Topics
- Handling failures with PutEvents
- Sending events using the AWS CLI
- Calculating Amazon EventBridge PutEvents event entry size

The following example Java code sends two identical events to EventBridge.

AWS SDK for Java Version 2.x

```java
EventBridgeClient eventBridgeClient = EventBridgeClient.builder().build();

PutEventsRequestEntry requestEntry = PutEventsRequestEntry.builder()
    .resources("resource1", "resource2")
    .source("com.mycompany.myapp")
    .detailType("myDetailType")
    .detail("{"key1": "value1", "key2": "value2" }")
    .build();

List<
PutEventsRequestEntry> requestEntries = new ArrayList<
PutEventsRequestEntry>();
requestEntries.add(requestEntry);

PutEventsRequest eventsRequest = PutEventsRequest.builder()
    .entries(requestEntries)
    .build();

PutEventsResponse result = eventBridgeClient.putEvents(eventsRequest);

for (PutEventsResultEntry resultEntry: result.entries()) {
    if (resultEntry.eventId() != null) {
        System.out.println("Event Id: " + resultEntry.eventId());
    } else {
        System.out.println("PutEvents failed with Error Code: " + resultEntry.errorCode());
    }
}
```

AWS SDK for Java Version 1.0

```java
EventBridgeClient eventBridgeClient = EventBridgeClient.builder().build();

PutEventsRequestEntry requestEntry = new PutEventsRequestEntry()
```
Handling failures with PutEvents

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());
    }
}

After you run this code, the PutEvents result includes an array of response entries. Each entry in the response array corresponds to an entry in the request array in order from the beginning to the end of the request and response. The response Entries array always includes the same number of entries as the request array.

Handling failures with PutEvents

By default, if an individual entry within a request fails, EventBridge continues processing the rest of the entries in the request. A response Entries array can include both successful and unsuccessful entries. You must detect unsuccessful entries and include them in a subsequent call.

Successful result entries include an Id value, and unsuccessful result entries include ErrorCode and ErrorMessage values. ErrorCode describes the type of error. ErrorMessage provides more information about the error. The following example has three result entries for a PutEvents request. The second entry is unsuccessful.

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

**Note**

If you use PutEvents to publish an event to an event bus that does not exist, EventBridge event matching will not find a corresponding rule and will drop the event. Although EventBridge will send a 200 response, it will not fail the request or include the event in the FailedEntryCount value of the request response.

You can include entries that are unsuccessful in subsequent PutEvents requests. First, to find out if there are failed entries in the request, check the FailedRecordCount parameter in...
PutEventsResult. If it isn't zero, then you can add each Entry that has an ErrorCode that is not null to a subsequent request. The following example shows a failure handler.

```java
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 = PutEventsResultEntryList.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 to EventBridge so they can be processed. The following example puts one custom event into EventBridge:

```
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 JSON file that contains custom events.

```
[
    {
        "Time": "2016-01-14T01:02:03Z",
        "Source": "com.mycompany.myapp",
        "Resources": [
            "resource1",
            "resource2"
        ],
        "DetailType": "myDetailType",
        "Detail": {
            "key1": "value1", "key2": "value2"
        }
    }
]"
Then, to use the AWS CLI to read the entries from this file and send events, at a command prompt, type:

```bash
aws events put-events --entries file://entries.json
```
Calculating Amazon EventBridge PutEvents event entry size

You can send custom events to EventBridge by using the PutEvents action. You can batch multiple event entries into one request for efficiency. The total entry size must be less than 256KB. You can calculate the entry size before you send the events.

Note
The size limit is imposed on the entry. Even if the entry is less than the size limit, the event in EventBridge 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 Amazon EventBridge events.

EventBridge calculates the PutEventsRequestEntry size as follows:

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

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

```java
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;
}
```

Note
If the entry size is larger than 256KB, we recommend uploading the event to an Amazon S3 bucket and including the Object URL in the PutEvents entry.

Events from AWS services

Many AWS services generate events that EventBridge receives. When an AWS service in your account emits an event, it goes to your account’s default event bus.
Event delivery from AWS services

Each AWS service that generates events sends them to EventBridge as either best effort or durable delivery attempts.

- **Best effort delivery** means that the service attempts to send all events to EventBridge, but in some rare cases an event might not be delivered.
- **Durable delivery** means the service will successfully attempt to deliver events to EventBridge at least once.

EventBridge will accept all valid events under normal conditions. In cases where events cannot be delivered because of an EventBridge service disruption, they will be retried again later by the AWS service for up to 24 hours.

Once an event is delivered to EventBridge, EventBridge matches it against rules and then follows the retry policy and any dead-letter queue specified for the event target(s).

For a list of AWS services that generate events, see ??? (p. 37).

Accessing AWS service events via AWS CloudTrail

AWS CloudTrail is a service that automatically records events such as AWS API calls. You can create EventBridge rules that use the information from CloudTrail. For more information about CloudTrail, see What is AWS CloudTrail?.

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

**Note**
Currently, events from API actions that start with the keywords List, Get, or Describe aren't processed by EventBridge, with a few exceptions. For example, the events from the following AWS STS actions:

- GetFederationToken
- GetSessionToken

To record events with a detail-type value of AWS API Call via CloudTrail, a CloudTrail trail with logging enabled is required.

When using CloudTrail with Amazon S3, you need to configure CloudTrail to log data events. For more information, see Enabling CloudTrail event logging for S3 buckets and objects.

Some occurrences in AWS services can be reported to EventBridge both by the service itself and by CloudTrail. For example, an Amazon EC2 API call that starts or stops an instance generates EventBridge events as well as events through CloudTrail.

CloudTrail supports both API callers and resource owners to receive events in their Amazon S3 buckets by creating trails, and delivers events to API callers through EventBridge. Resource owners in addition to API callers can monitor cross-account API calls through EventBridge. CloudTrail's integration with EventBridge provides a convenient way to set automated rules-based workflows in response to events.

You can't use AWS Put*Events API call events that are larger than 256 KB in size as event patterns because the maximum size of any Put*Events requests is 256 KB. For more information about the API calls that you can use, see CloudTrail supported services and integrations.
AWS services that generate events

The following table shows AWS services that generate events. Choose the service name to see more information about how that service and EventBridge work together.

Each AWS service that generates events sends them to EventBridge as either best effort or durable delivery attempts. For more information, see ??? (p. 36).

This table includes a representation of the AWS services that send events to EventBridge, but it doesn't include every service. For services not listed that send events to EventBridge, assume a best effort delivery.

<table>
<thead>
<tr>
<th>Service</th>
<th>Attempt Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa for Business</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Account Management</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon API Gateway</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS AppConfig</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon AppFlow</td>
<td>Best effort</td>
</tr>
<tr>
<td>Application Auto Scaling</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Application Cost Profiler</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Application Migration Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Athena</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Backup</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Batch</td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon Braket</td>
<td>Durable</td>
</tr>
<tr>
<td>AWS Certificate Manager</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Chime</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Cloud Directory</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CloudFormation</td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon CloudFront</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CloudHSM</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon CloudSearch</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CloudShell</td>
<td>Best effort</td>
</tr>
<tr>
<td>Events from AWS CloudTrail</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon CloudWatch Application Insights</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon CloudWatch Internet Monitor</td>
<td>Best effort</td>
</tr>
<tr>
<td>Service</td>
<td>Attempt Type</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Amazon CloudWatch Logs</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon CloudWatch Synthetics</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CodeArtifact</td>
<td>Durable</td>
</tr>
<tr>
<td><strong>AWS CodeBuild</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS CodeCommit</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS CodeDeploy</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon CodeGuru Profiler</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS CodePipeline</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CodeStar</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CodeStar Connections</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Cognito Identity</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Cognito user pools</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Cognito Sync</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Config</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Connect</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Connect Voice ID</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Control Tower</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Database Migration Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Data Exchange</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Data Lifecycle Manager</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Data Pipeline</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS DataSync</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Device Farm</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon DevOps Guru</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Direct Connect</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Directory Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Elastic Beanstalk</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Elastic Block Store</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic Block Store volume modifications</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon ElastiCache</td>
<td>Best effort</td>
</tr>
<tr>
<td>Service</td>
<td>Attempt Type</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Amazon Elastic Compute Cloud (Amazon EC2)</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon EC2 Auto Scaling</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon EC2 Fleets</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon EC2 Spot Instance Interruption</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Elastic Container Registry</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Elastic Container Service</strong></td>
<td>Durable</td>
</tr>
<tr>
<td><strong>AWS Elastic Disaster Recovery</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Elastic File System</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Elastic Kubernetes Service</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Elastic Load Balancing</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic MapReduce</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic Transcoder</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Elemental MediaConnect</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Elemental MediaConvert</strong></td>
<td>Durable</td>
</tr>
<tr>
<td><strong>AWS Elemental MediaLive</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Elemental MediaPackage</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Elemental MediaStore</strong></td>
<td>Durable</td>
</tr>
<tr>
<td><strong>Amazon EMR</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon EMR on EKS</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon EMR Serverless</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon EventBridge scheduled rules (p. 95)</strong></td>
<td>Durable</td>
</tr>
<tr>
<td><strong>Amazon EventBridge schemas</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Fault Injection Simulator</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Forecast</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon GameLift</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Glue</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Glue DataBrew</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS IoT Greengrass V1</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS IoT Greengrass V2</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Ground Station</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon GuardDuty</td>
<td>Best effort</td>
</tr>
<tr>
<td>Service</td>
<td>Attempt Type</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>AWS Health</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Identity and Access Management (IAM)</td>
<td>Best effort</td>
</tr>
<tr>
<td>IAM Access Analyzer</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Inspector Classic</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Inspector</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS IoT</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS IoT Analytics</td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon Interactive Video Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Kinesis</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Kinesis Data Firehose</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Key Management Service CMK deletion</td>
<td>Durable</td>
</tr>
<tr>
<td>AWS Key Management Service CMK rotation</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Key Management Service imported key</td>
<td>Best effort</td>
</tr>
<tr>
<td>material expiration</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Lambda</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Location Service</td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon Machine Learning</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Macie</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Managed Blockchain</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Managed Services</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Management Console Sign-in</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Metering Marketplace</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Migration Hub</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Migration Hub Refactor Spaces</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Monitoring</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Network Manager</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon OpenSearch Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS OpsWorks</td>
<td>Durable</td>
</tr>
<tr>
<td>AWS OpsWorks CM</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Organizations</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Polly</td>
<td>Best effort</td>
</tr>
</tbody>
</table>
### Services that generate events

<table>
<thead>
<tr>
<th>Service</th>
<th>Attempt Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Private Certificate Authority</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Proton</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon QLDB</td>
<td>Durable</td>
</tr>
<tr>
<td><strong>Amazon RDS</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Recycle Bin</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Redshift</strong></td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon Redshift Data API</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Redshift Serverless</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Resource Access Manager</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Resource Groups</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Resource Groups Tagging API</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Route 53</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Route 53 Recovery Readiness</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon SageMaker</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>Savings Plans</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Secrets Manager</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Security Hub</strong></td>
<td>Durable</td>
</tr>
<tr>
<td>AWS Security Token Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Server Migration Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Service Catalog</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Signer</td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon Simple Email Service</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>Amazon Simple Storage Service (Amazon S3)</strong></td>
<td>Durable</td>
</tr>
<tr>
<td>Amazon S3 Glacier</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon S3 on Outposts</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Simple Queue Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Simple Notification Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Simple Workflow Service</td>
<td>Best effort</td>
</tr>
<tr>
<td><strong>AWS Step Functions</strong></td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Storage Gateway</td>
<td>Durable</td>
</tr>
<tr>
<td><strong>AWS Support</strong></td>
<td>Best effort</td>
</tr>
</tbody>
</table>
Receiving events from a SaaS partner with Amazon EventBridge

To receive events\(^\text{\textsuperscript{p. 27}}\) from SaaS partner applications and services, you need a partner event source from the partner. Then you can create a partner event bus\(^\text{\textsuperscript{p. 9}}\) and match it to the partner event source.
The following video covers SaaS integrations with EventBridge: [Software as a service (SaaS) partners](#)

**Topics**
- [Supported SaaS partner integrations](#)
- [Configuring Amazon EventBridge to receive events from a SaaS integration](#)
- [Creating a rule that matches SaaS partner events](#)
- [Receiving events using AWS Lambda function URLs](#)
- [Receiving events from Salesforce](#)

**Supported SaaS partner integrations**

EventBridge supports the following SaaS partner integrations:
- Auth0
- Blitline
- BUIDLHub
- Buildkite
- CleverTap
- Datadog
- Epsagon
- Freshworks
- Genesys
- GS2
- Karte
Supported SaaS partner integrations

- Kloudless
- Mackerel
- MongoDB
- New Relic
- OneLogin
- Opsgenie
- PagerDuty
- Payshield
- SaaSus Platform
- SailPoint
- Saviynt
- Segment
- Shopify
- SignalFx
- Site24x7
- Stax
- SugarCRM
- SugarCRM
- Symantec
- Thundra
- TriggerMesh
- Whispir
- Zendesk
- Amazon Seller Partner API

Partner event sources are available in the following Regions.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-1</td>
<td>US East (N. Virginia)</td>
</tr>
<tr>
<td>us-east-2</td>
<td>US East (Ohio)</td>
</tr>
<tr>
<td>us-west-1</td>
<td>US West (N. California)</td>
</tr>
<tr>
<td>us-west-2</td>
<td>US West (Oregon)</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>Canada (Central)</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>Europe (Frankfurt)</td>
</tr>
<tr>
<td>eu-central-2</td>
<td>Europe (Zurich)</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>Europe (Ireland)</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>Europe (London)</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>Europe (Paris)</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>Europe (Stockholm)</td>
</tr>
<tr>
<td>eu-south-1</td>
<td>Europe (Milan)</td>
</tr>
</tbody>
</table>
Configuring Amazon EventBridge to receive events from a SaaS integration

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Partner event sources.
3. Find the partner that you want and then choose Set up for that partner.
4. To copy your account ID to the clipboard, choose Copy.
5. In the navigation pane, choose Partner event sources.
6. Go to the partner’s website and follow the instructions to create a partner event source using your account ID. The event source that you create is available to only your account.
7. Go back to the EventBridge console and choose Partner event sources in the navigation pane.
8. Select the button next to the partner event source and then choose Associate with event bus.

The status of the event source changes from Pending to Active, and the name of the event bus updates to match the partner event source name. You can now start creating rules that match events from the partner event source. For more information, see Creating a rule that matches SaaS partner events (p. 46).
Creating a rule that matches SaaS partner events

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule.

   A rule can't have the same name as another rule in the same Region and on the same event bus.
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select AWS default event bus. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose Other.
9. (Optional) For Sample events, choose the type of event.
10. For Event pattern, enter a JSON event pattern.
11. Choose Next.
12. For Target types, choose AWS service.
13. For Select a target, choose the AWS service that you want to send information to when EventBridge detects an event that matches the event pattern.
14. The fields displayed vary depending on the service you choose. Enter information specific to this target type as needed.
15. For many target types, EventBridge needs permissions to send events to the target. In these cases, EventBridge can create the IAM role needed for your rule to run. Do one of the following:
   - To create an IAM role automatically, choose Create a new role for this specific resource.
   - To use an IAM role that you created earlier, choose Use existing role and select the existing role from the drop-down list.
16. (Optional) For Additional settings, do the following:
   a. For Maximum age of event, enter a value between one minute (00:01) and 24 hours (24:00).
   b. For Retry attempts, enter a number between 0 and 185.
   c. For Dead-letter queue, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
      - Choose None to not use a dead-letter queue.
      - Choose Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue and then select the queue to use from the drop-down list.
      - Choose Select an Amazon SQS queue in an other AWS account as a dead-letter queue and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it. For more information, see Granting permissions to the dead-letter queue (p. 60).
17. (Optional) Choose Add another target to add another target for this rule.
18. Choose Next.
19. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 453).
20. Choose Next.
21. Review the details of the rule and choose Create rule.
Receiving events using AWS Lambda function URLs

**Note**
In order for the Inbound Webhook to be accessible by our partners, we're creating an Open Lambda in your AWS account that is secured at the Lambda application level by verifying the authentication signature sent by the third-party partner. Please review this configuration with your security team. For more information, see Security and auth model for Lambda function URLs.

Your Amazon EventBridge event bus (p. 9) can use an AWS Lambda function URL created by an AWS CloudFormation template to receive events (p. 27) from supported SaaS providers. With function URLs, the event data is sent to a Lambda function. The function then converts this data into an event that can be ingested by EventBridge and sent to an event bus for processing. Once the event is on an event bus, you can use rules to filter the events, apply any configured input transformations, and then route it to the correct target.

**Note**
Creating Lambda function URLs will increase your monthly costs. For more information, see AWS Lambda pricing.

To set up a connection to EventBridge, you first select the SaaS provider that you want to set up a connection with. Then, you provide a signing secret that you’ve created with that provider, and select the EventBridge event bus to send events to. Finally, you use an AWS CloudFormation template and create the needed resources to complete the connection.

The following SaaS providers are currently available for use with EventBridge using Lambda function URLs:

- GitHub
- Stripe
- Twilio

**Topics**
- Set up a connection to GitHub (p. 47)
- Step 1: Create the AWS CloudFormation stack (p. 47)
- Step 2: Create a GitHub webhook (p. 48)
- Set up a connection to a Stripe (p. 48)
- Set up a connection to a Twilio (p. 49)
- Update webhook secret or auth token (p. 50)
- Update Lambda function (p. 51)
- Available event types (p. 51)
- Quotas, error codes, and retrying delivery (p. 52)

**Set up a connection to GitHub**

**Step 1: Create the AWS CloudFormation stack**

First, use the Amazon EventBridge console to create a CloudFormation stack:

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. From the navigation pane, choose Quick starts.
3. Under **Inbound webhooks using Lambda fURLs**, choose **Get started**.
4. Under **GitHub**, choose **Set up**.
5. Under **Step 1: Select an event bus**, select an event bus from the dropdown list. This event bus receives data from the Lambda function URL that you provide to GitHub. You can also create an event bus by selecting **New event bus**.
6. Under **Step 2: Set up using CloudFormation**, choose **New GitHub webhook**.
7. Select **I acknowledge that the Inbound Webhook I create will be publicly accessible.** and choose **Confirm**.
8. Enter a name for the stack.
9. Under parameters, verify that the correct event bus is listed, then specify a secure token for the **GitHubWebhookSecret**. For more information on creating a secure token, see **Setting your secret token** in the GitHub documentation.
10. Under **Capabilities and transforms**, select each of the following:
    - I acknowledge that AWS CloudFormation might create IAM resources.
    - I acknowledge that AWS CloudFormation might create IAM resources with custom names.
    - I acknowledge that AWS CloudFormation might require the following capability: **CAPABILITY_AUTO_EXPAND**
11. Choose **Create stack**.

**Step 2: Create a GitHub webhook**

Next, create the webhook on GitHub. You’ll need both the secure token and the Lambda function URL you created in step 2 to complete this step. For more information, see **Creating webhooks** in the GitHub documentation.

**Set up a connection to a Stripe**

**Step 1: Create a Stripe endpoint**

To set up a connection between EventBridge and Stripe, first create a Stripe endpoint and note the endpoint secret. You’ll use this endpoint secret when you set up your stack in step 2. For more information, see **Interactive webhook endpoint builder** in the Stripe documentation.

**Note**
You’ll need a dummy URL to set up the endpoint with Stripe. For example, www.example.com.

**Step 2: Create the AWS CloudFormation stack**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Quick starts**.
3. Under **Inbound webhooks using Lambda fURLs**, choose **Get started**.
4. Under **Stripe**, choose **Set up**.
5. Under **Step 1: Select an event bus**, select an event bus from the dropdown list. This event bus receives data from the Lambda function URL that you provide to Stripe. You can also create an event bus by selecting **New event bus**.
6. Under **Step 2: Set up using CloudFormation**, choose **New Stripe webhook**.
7. Select **I acknowledge that the Inbound Webhook I create will be publicly accessible.** and choose **Confirm**.
8. Enter a name for the stack.
9. Under parameters, verify that the correct event bus is listed, then enter the `StripeWebhookSecret` that you created in Step 1.

10. Under **Capabilities and transforms**, select each of the following:
    - I acknowledge that AWS CloudFormation might create IAM resources.
    - I acknowledge that AWS CloudFormation might create IAM resources with custom names.
    - I acknowledge that AWS CloudFormation might require the following capability: `CAPABILITY_AUTO_EXPAND`

11. Choose **Create stack**.

### Step 3: Update the Stripe endpoint

Now that you've created the Lambda function URL, update the Stripe endpoint to send events to the Lambda function URL.

### Set up a connection to a Twilio

#### Step 1: Find your Twilio auth token

To set up a connection between Twilio and EventBridge, first set up the connection to Twilio with the auth token, or secret, for your Twilio account. For more information, see [Auth Tokens and How To Change Them](https://www.twilio.com/docs/security/twilio-api) in the Twilio documentation.

#### Step 2: Create the AWS CloudFormation stack

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Quick starts**.
3. Under **Inbound webhooks using Lambda fURLs**, choose **Get started**.
4. Under **Twilio**, choose **Set up**.
5. Under **Step 1: Select and event bus**, select an event bus from the dropdown list. This event bus receives data from the Lambda function URL that you provide to Twilio. You can also create an event bus by selecting **New event bus**.
6. Under **Step 2: Set up using CloudFormation**, choose **New Twilio webhook**.
7. Select **I acknowledge that the Inbound Webhook I create will be publicly accessible.** and choose **Confirm**.
8. Enter a name for the stack.
9. Under parameters, verify that the correct event bus is listed, then enter the `TwilioWebhookSecret` that you created in Step 1.
10. Under **Capabilities and transforms**, select each of the following:
    - I acknowledge that AWS CloudFormation might create IAM resources.
    - I acknowledge that AWS CloudFormation might create IAM resources with custom names.
    - I acknowledge that AWS CloudFormation might require the following capability: `CAPABILITY_AUTO_EXPAND`

11. Choose **Create stack**.

### Step 3: Create a Twilio webhook

After you set up the Lambda function URL, you need to give it to Twilio so that event data can be sent. For more information, see [Configure your public URL with Twilio](https://www.twilio.com/docs/security/twilio-api) in the Twilio documentation.
Update webhook secret or auth token

Update GitHub secret

**Note**
GitHub doesn’t support having two secrets at the same time. You may experience resource downtime while the GitHub secret and the secret in the AWS CloudFormation stack are out of sync. GitHub messages sent while the secrets are out of sync will fail because of incorrect signatures. Wait until the GitHub and CloudFormation secrets are in sync, then try again.

1. Create a new GitHub secret. For more information, see [Encrypted secrets](https://github.com/docs) in the GitHub documentation.
3. From the navigation pane, choose Stacks.
4. Choose the stack for the webhook that includes the secret you want to update.
5. Choose Update.
6. Make sure Use current template is selected and choose Next.
7. Under GitHubWebhookSecret, clear Use existing value, enter the new GitHub secret you created in step 1, and choose Next.
8. Choose Next.
9. Choose Update stack.

It may take up to one hour for the secret to propagate. To reduce this downtime, you can refresh the Lambda execution context.

Update Stripe secret

1. From the Stripe dashboard, in the Webhooks section, select Roll secret and delay the expiration for at least two(2) hours. For more information, see [Roll endpoint secrets](https://stripe.com/docs) in the Stripe documentation.
3. From the navigation pane, choose Stacks.
4. Choose the stack for the webhook that includes the secret you want to update.
5. Choose Update.
6. Make sure Use current template is selected and choose Next.
7. Under StripeWebhookSecret, clear Use existing value, enter the new Stripe secret you created in step 1, and choose Next.
8. Choose Next.
9. Choose Update stack.

Stripe will send both the old signature and the new signature during the rotation period.

Update Twilio secret

**Note**
Twilio doesn’t support having two secrets at the same time. You may experience resource downtime while the Twilio secret and the secret in the AWS CloudFormation stack are out of sync. Twilio messages sent while the secrets are out of sync will fail because of incorrect signatures. Wait until the Twilio and CloudFormation secrets are in sync, then try again.

1. Create a new Twilio secret. For more information, see [Auth Tokens and How To Change Them](https://twilio.com/docs) in the Twilio documentation.
3. From the navigation pane, choose **Stacks**.
4. Choose the stack for the webhook that includes the secret you want to update.
5. Choose **Update**.
6. Make sure **Use current template** is selected and choose **Next**.
7. Under **TwilioWebhookSecret**, clear **Use existing value**, enter the new Twilio secret you created in step 1, and choose **Next**.
8. Choose **Next**.
9. Choose **Update stack**.

It may take up to one hour for the secret to propagate. To reduce this downtime, you can refresh the Lambda execution context.

### Update Lambda function

The Lambda function that's created by the CloudFormation stack creates the basic webhook. If you want to customize the Lambda function for a specific use case, such as customized logging, use the CloudFormation console to access the function and then use the Lambda console to update the Lambda function code.

#### Access the Lambda function

2. From the navigation pane, choose **Stacks**.
3. Choose the stack for the webhook that includes the Lambda function you want to update.
4. Choose **Resources** tab.
5. To open the Lambda function in the Lambda console, under **Physical ID**, choose the ID of the Lambda function.

Now that you've accessed the Lambda function, use the Lambda console to update the function code.

#### Update the Lambda function code

1. Under **Actions**, choose **Export function**.
2. Choose **Download deployment package** and save the file to your computer.
3. Unzip the deployment package .zip file, update the app .py file, and zip the updated deployment package, making sure all the files in the original .zip file are included.
4. In the Lambda console, choose the **Code** tab.
5. Under **Code source**, choose **Upload from**.
6. Choose **.zip file**, and then choose **Upload**.
   - In the file chooser, select the file you updated, choose **Open**, and then choose **Save**.
7. Under **Actions**, choose **Publish new version**.

### Available event types

The following event types are currently supported by CloudFormation event buses:

- **GitHub** – All event types are supported.
- **Stripe** – All event types are supported.
- **Twilio** – Post-event webhooks are supported.
Quotas, error codes, and retrying delivery

Quotas

The number of incoming requests to the webhook is capped by the underlying AWS services. The following table includes the relevant quotas.

<table>
<thead>
<tr>
<th>Service</th>
<th>Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>Default: 10 concurrent executions</td>
</tr>
<tr>
<td></td>
<td>For more information about quotas, including requesting quota increases, see Lambda quotas.</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>Default: 5,000 requests per second</td>
</tr>
<tr>
<td></td>
<td>For more information about quotas, including requesting quota increases, see AWS Secrets Manager quotas.</td>
</tr>
<tr>
<td>Note</td>
<td>The number of requests per second is minimized using the AWS Secrets Manager Python caching client.</td>
</tr>
<tr>
<td>Amazon EventBridge</td>
<td>256KB maximum entry size for PutEvents actions.</td>
</tr>
<tr>
<td></td>
<td>EventBridge enforces Region-based rate quotas. For more information, see ??? (p. 447).</td>
</tr>
</tbody>
</table>

Error codes

Each AWS service returns specific error codes when errors occur. The following table includes the relevant error codes.

<table>
<thead>
<tr>
<th>Service</th>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>429 “TooManyRequestsExption”</td>
<td>The concurrent execution quota is exceeded.</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>500 “Internal Server Error”</td>
<td>The requests per second quota is exceeded.</td>
</tr>
<tr>
<td>Amazon EventBridge</td>
<td>500 “Internal Server Error”</td>
<td>The rate quota is exceeded for the Region.</td>
</tr>
</tbody>
</table>

Event redelivery

When errors happen you can retry delivery of the affected events. Each SaaS provider has different retry procedures.

GitHub

Use the GitHub webhooks API to check the deliver status of any webhook call and redeliver the event, if needed. For more information, see the following GitHub documentation:

- **Organization** – Redeliver a delivery for an organization webhook
• Repository – Redeliver a delivery for a repository webhook
• App – Redeliver a delivery for an app webhook

**Stripe**

Stripe attempts to deliver your webhooks for up to three days with an exponential back off. For more information, see the following Stripe documentation:

• Delivery attempts and retries
• Handle errors

**Twilio**

Twilio users can customize event retry options using connection overrides. For more information, see Webhooks (HTTP callbacks): Connection Overrides in the Twilio documentation.
Receiving events from Salesforce

You can use Amazon EventBridge to receive events from Salesforce in following ways:

- By using Salesforce's Event Bus Relay feature to receive events directly on an EventBridge partner event bus.
- By configuring a flow in Amazon AppFlow that uses Salesforce as a data source. Amazon AppFlow then sends Salesforce events to EventBridge by using a partner event bus.

You can send event information to Salesforce using API destinations. Once the event is sent to Salesforce, it can be processed by Flows or Apex triggers. For more information about setting up a Salesforce API destination, see ??? (p. 286).

Topics
- Receiving events from Salesforce using Event Bus Relay (p. 54)
- Receiving events from Salesforce using Amazon AppFlow (p. 55)

Receiving events from Salesforce using Event Bus Relay

Step 1: Set up Salesforce Event Bus Relay and an EventBridge partner event source

When you create an event relay configuration on Salesforce, Salesforce creates a partner event source in EventBridge in the pending state.

To configure Salesforce Event Bus Relay

1. Set Up a REST API Tool
2. (Optional) Define a Platform Event
3. Create a Channel for a Custom Platform Event
4. Create a Channel Member to Associate the Custom Platform Event
5. Create a Named Credential
6. Create an Event Relay Configuration

Step 2: Activate Salesforce partner event source in the EventBridge console and start the event relay

1. Open the Partner event sources page in the EventBridge console.
2. Select the Salesforce partner event source that you created in Step 1.
3. Choose Associate with event bus.
4. Validate the name of the partner event bus.
5. Choose Associate.
6. Start the Event Relay

Now that you've set up and started the Event Bus Relay and configured the partner event source you can create an EventBridge rule that reacts to events (p. 84) to filter and send the data to a target (p. 112).
Receiving events from Salesforce using Amazon AppFlow

Amazon AppFlow encapsulates events from Salesforce in an EventBridge event envelope. The following example shows a Salesforce event received by an EventBridge partner event bus.

```
{
"version": "0",
"id": "5c42b99e-e005-43b3-c744-07990c50d2cc",
"detail-type": "AccountChangeEvent",
"source": "aws.partner/appflow.test/salesforce.com/364228160620/CustomSF-Source-Final",
"account": "000000000",
"time": "2020-08-20T18:25:51Z",
"region": "us-west-2",
"resources": [],
"detail": {
"ChangeEventHeader": {
"commitNumber": 248197218874,
"commitUser": "0056g000003XW7AAAW",
"sequenceNumber": 1,
"entityName": "Account",
"changeType": "UPDATE",
"changedFields": [
"LastModifiedDate",
"Region__c"
],
"changeOrigin": "com/salesforce/api/soap/49.0;client=SfdcInternalAPI/",
"transactionKey": "000035af-b239-0581-9f14-461e4187de11",
"commitTimestamp": 1597947935000,
"recordIds": [
"0016g00000MLhLeAAL"
],
"LastModifiedDate": "2020-08-20T18:25:35.000Z",
"Region__c": "America"
}
}
```

Step 1: Configure Amazon AppFlow to use Salesforce as a partner event source

To send events to EventBridge, you first need to configure Amazon AppFlow to use Salesforce as a partner event source.

1. In the Amazon AppFlow console, choose Create flow.
2. In the Flow details section, in Flow name enter a name for your flow.
3. (Optional) Enter a description for the flow and then choose Next.
4. Under Source details, choose Salesforce from the Source name drop-down, and then choose Connect to create a new connection.
5. In the Connect to Salesforce dialog box, choose either Production or Sandbox for the Salesforce environment.
6. In the Connection name field, enter a unique name for the connection, and then choose Continue.
7. In the Salesforce dialog box, do the following:
   a. Enter your Salesforce sign-in credentials to log in to Salesforce.
   b. Select Salesforce events for the types of data for Amazon AppFlow to process.
8. In the Choose Salesforce event drop-down, select the type of event to send to EventBridge.
9. For a destination, select Amazon EventBridge.
10. Select Create new partner event source.
11. (Optional) Specify a unique suffix for the partner event source.
12. Choose **Generate partner event source**.
13. Choose an Amazon S3 bucket to store event payload files that are larger than 256 KB.
14. In the **Flow trigger** section, ensure that **Run flow on event** is selected. This setting ensures that the flow is executed when a new Salesforce event occurs.
15. Choose **Next**.
16. For field mapping, select **Map all fields directly**. Alternatively, you can select the fields that are of interest from the **Source field name** list.
   
   For more information about field mapping, see [Map data fields](#).
17. Choose **Next**.
18. (Optional) Configure filters for data fields in Amazon AppFlow.
19. Choose **Next**.
20. Review the settings and then choose **Create flow**.

With the flow configured, Amazon AppFlow creates a new partner event source that you then need to associate with a partner event bus in your account.

**Step 2: Configure EventBridge to receive Salesforce events**

Ensure that the Amazon AppFlow flow that is triggered from Salesforce events with EventBridge as a destination is configured before following instructions in this section.

**To configure EventBridge to receive Salesforce events**

1. Open the **Partner event sources** page in the EventBridge console.
2. Select the Salesforce partner event source that you created in Step 1.
3. Choose **Associate with event bus**.
4. Validate the name of the partner event bus.
5. Choose **Associate**.
6. In the Amazon AppFlow console, open the flow you created and choose **Activate flow**.
7. Open the **Rules** page in the EventBridge console.
8. Choose **Create rule**.
9. Enter a unique name for the rule.
10. Choose **Event pattern** in the **Define pattern** section.
11. For **Event matching pattern**, select **Pre-defined pattern by service**.
12. For **Service provider** section, select **All Events**.
13. For **Select event bus**, choose **Custom or partner event bus**.
14. Select the event bus that you associated with the Amazon AppFlow partner event source.
15. For **Select targets**, choose the AWS service that is to act when the rule runs. One rule can have up to five targets.
16. Choose **Create**.

The target service receives all Salesforce events configured for your account. To filter the events or send some events to different targets, you can use [content-based filtering with event patterns](#).

**Note**

For events larger than 256KB, Amazon AppFlow doesn't send the full event to EventBridge. Instead, Amazon AppFlow puts the event into an S3 bucket in your account, and then sends an event to EventBridge with a pointer to the Amazon S3 bucket. You can use the pointer to get the full event from the bucket.
Debugging Amazon EventBridge event delivery

Event delivery issues can be hard to identify, EventBridge offers a few ways to debug and recover from event delivery failures.

Topics
• Event retry policy and using dead-letter queues (p. 58)
Event retry policy and using dead-letter queues

Sometimes an event (p. 27) isn’t successfully delivered to the target (p. 112) specified in a rule (p. 82). This can happen when, for example, the target resource is unavailable, when EventBridge lacks permission to the target resource, or due to network conditions. When an event isn’t successfully delivered to a target because of retriable errors, EventBridge retries sending the event. You set the length of time it tries, and number of retry attempts in the Retry policy settings for the target. By default, EventBridge retries sending the event for 24 hours and up to 185 times with an exponential back off and jitter, or randomized delay. If an event isn’t delivered after all retry attempts are exhausted, the event is dropped and EventBridge doesn’t continue to process it. To avoid losing events after they fail to be delivered to a target, you can configure a dead-letter queue (DLQ) and send all failed events to it for processing later.

EventBridge DLQs are standard Amazon SQS queues that EventBridge uses to store events that couldn’t successfully be delivered to a target. When you create a rule and add a target, you can choose whether or not to use a DLQ. When you configure a DLQ, you can retain any events that weren’t successfully delivered. Then you can resolve the issue that resulted in the failed event delivery and process the events at a later time.

Event errors are handled in different ways. Some events are dropped or sent to a DLQ without any retry attempts. For example, for errors that result from missing permissions to a target, or a target resource that no longer exists, all retry attempts fail until an action is taken to resolve the underlying issue. Rather than retrying, EventBridge sends these events directly to the DLQ, if you have one.

When an event delivery fails, EventBridge publishes an event to Amazon CloudWatch metrics indicating that a target invocation failed. If you use a DLQ, additional metrics are sent to CloudWatch including InvocationsSentToDLQ and InvocationsFailedToBeSentToDLQ.

Each message in your DLQ will include the following custom attributes:

- RULE_ARN
- TARGET_ARN
- ERROR_CODE

The following is a sample of the error codes a DLQ can return:

- CONNECTION_FAILURE
- CROSS_ACCOUNT_INGESTION_FAILED
- CROSS_REGION_INGESTION_FAILED
- ERROR_FROM_TARGET
- EVENTS_IN_BATCH_REQUEST_REJECTED
- EVENTS_IN_BATCH_REQUEST_REJECTED
- FAILED_TO_ASSUME_ROLE
- INTERNAL_ERROR
- INVALID_JSON
- INVALID_PARAMETER
- NO_PERMISSIONS
- NO_RESOURCE
- RESOURCE_ALREADY_EXISTS
- RESOURCE_LIMIT_EXCEEDED
- RESOURCE_MODIFICATION_COLLISION
- SDK_CLIENT_ERROR
- THIRD_ACCOUNT_HOP_DETECTED
- THIRD_REGION_HOP_DETECTED
- THROTTLING
- TIMEOUT
- TRANSIENT_ASSUME_ROLE
- UNKNOWN
- ERROR_MESSAGE
- EXHAUSTED_RETRY_CONDITION

The following conditions can be returned:
- MaximumRetryAttempts
- MaximumEventAgeInSeconds
- RETRY_ATTEMPTS

The following video goes over settings up DLQs: [Using dead-letter queues (DLQs)]

Topics
- Considerations for using a dead-letter queue (p. 59)
- Granting permissions to the dead-letter queue (p. 60)
- How to resend events from a dead-letter queue (p. 61)

**Considerations for using a dead-letter queue**

Consider the following when configuring a DLQ for EventBridge.

- Only standard queues are supported. You can’t use a FIFO queue for a DLQ in EventBridge.
EventBridge includes event metadata and message attributes in the message, including: the Error Code, Error Message, the Exhausted Retry Condition, Rule ARN, Retry Attempts, and the Target ARN. You can use these values to identify an event and the cause of the failure.

Permissions for DLQs in the same account:

- If you add a target to a rule using the console, and you choose an Amazon SQS queue in the same account, a resource-based policy (p. 405) that grants EventBridge access to the queue is attached to the queue for you.
- If you use the PutTargets operation of the EventBridge API to add or update a target for a rule, and you choose an Amazon SQS queue in the same account, you must manually grant permissions to the queue selected. To learn more, see Granting permissions to the dead-letter queue (p. 60).

Permissions for using Amazon SQS queues from a different AWS account:

- If you create a rule from the console, queues from other accounts aren’t displayed for you to select. You must provide the ARN for the queue in the other account, and then manually attach a resource-based policy to grant permission to the queue. To learn more, see Granting permissions to the dead-letter queue (p. 60).
- If you create a rule using the API, you must manually attach a resource-based policy to the SQS queues in another account that is used as the dead-letter queue. To learn more, see Granting permissions to the dead-letter queue (p. 60).
- The Amazon SQS queue you use must be in the same Region in which you create the rule.

Granting permissions to the dead-letter queue

When you configure a DLQ for a target of a rule, EventBridge sends the events with failed invocations to the Amazon SQS queue selected. To successfully deliver events to the queue, EventBridge must have permission to do so. When you configure a target for a rule and select a DLQ using the EventBridge console, the permissions are automatically added. If you create a rule using the API, or use a queue that is in a different AWS account, you must manually create a resource-based policy that grants the required permissions and then attach it to the queue.

The following resource-based policy demonstrates how to grant the required permissions for EventBridge to send event messages to an Amazon SQS queue. The policy example grants the EventBridge service permissions to use the SendMessage operation to send messages to a queue named "MyEventDLQ". The queue must be in the us-west-2 Region in AWS account 123456789012. The Condition statement allows only requests that come from a rule named "MyTestRule" that is created in the us-west-2 Region in the AWS account 123456789012.

```json
{
  "Sid": "Dead-letter queue permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Condition": {
    "ArnEquals": {
      "aws:SourceArn": "arn:aws:events:us-west-2:123456789012:rule/MyTestRule"
    }
  }
}
```

To attach the policy to the queue, use the Amazon SQS console, open the queue, then choose the Access policy and edit the policy. You can also use the AWS CLI, to learn more see Amazon SQS permissions (p. 408).
How to resend events from a dead-letter queue

You can move messages out of a DLQ in two ways:

- Avoid writing Amazon SQS consumer logic – Set your DLQ as an event source to the Lambda function to drain your DLQ.
- Write Amazon SQS consumer logic – Use the Amazon SQS API, AWS SDK, or AWS CLI to write custom consumer logic for polling, processing, and deleting the messages in the DLQ.
Amazon EventBridge event patterns

Event patterns have the same structure as the events (p. 27) they match. Rules (p. 82) use event patterns to select events and send them to targets. An event pattern either matches an event or it doesn’t.

**Important**
In EventBridge, it is possible to create rules that can lead to higher-than-expected charges and throttling. For example, you can inadvertently create a rule that leads to an infinite loop, where a rule is fired recursively without end. Suppose you created a rule to detect that ACLs have changed on an Amazon 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.
For guidance on how to write precise rules and event patterns to minimize such unexpected results, see ??? (p. 106) and ??? (p. 78).

The following video goes over the basics of event patterns: [How to filter events](#)

**Topics**
- Creating event patterns (p. 63)
- Example events and event patterns (p. 66)
- Matching null values and empty strings in Amazon EventBridge event patterns (p. 68)
- Arrays in Amazon EventBridge event patterns (p. 69)
- Content filtering in Amazon EventBridge event patterns (p. 70)
- Testing an event pattern using the EventBridge Sandbox (p. 76)
Creating event patterns

To create an event pattern, you specify the fields of an event that you want the event pattern to match. Only specify the fields that you use for matching. The previous event pattern example only provides values for three fields: the top-level fields "source" and "detail-type", and the "state" field inside the "detail" object field. EventBridge ignores all the other fields in the event when applying the rule.

For an event pattern to match an event, the event must contain all the field names listed in the event pattern. The field names must also appear in the event with the same nesting structure.

When you write event patterns to match events, you can use the TestEventPattern API or the test-event-pattern CLI command to test that your pattern matches the correct events. For more information, see TestEventPattern.

Matching event values

In an event pattern, the value to match is in a JSON array, surrounded by square brackets ("[", "]") so that you can provide multiple values. For example, to match events from Amazon EC2 or AWS Fargate, you could use the following pattern, which matches events where the value for the "source" field is either "aws.ec2" or "aws.fargate".

```json
{
    "source": ["aws.ec2", "aws.fargate"]
}
```
Considerations when creating event patterns

Below are some things to consider when constructing your event patterns:

- EventBridge ignores the fields in the event that aren't included in the event pattern. The effect is that there is a "*": "*" wildcard for fields that don't appear in the event pattern.
- The values that event patterns match follow JSON rules. You can include strings enclosed in quotation marks ("), numbers, and the keywords true, false, and null.
- For strings, EventBridge uses exact character-by-character matching without case-folding or any other string normalization.
- For numbers, EventBridge uses string representation. For example, 300, 300.0, and 3.0e2 are not considered equal.
- If multiple patterns are specified for the same JSON field, EventBridge only uses the last one.
- Be aware that when EventBridge compiles event patterns for use, it uses dot (.) as the joining character.

This means EventBridge will treat the following event patterns as identical:

```json
## has no dots in keys
{ "detail" : { "state" : { "status" : [ "running" ] } } }
## has dots in keys
{ "detail" : { "state.status" : [ "running" ] } }
```

And that both event patterns will match the following two events:

```json
## has no dots in keys
{ "detail" : { "state" : { "status" : "running" } } }
## has dots in keys
{ "detail" : { "state.status" : "running" } }
```

**Note**
This describes current EventBridge behavior, and should not be relied on to not change.

- Event patterns containing duplicate fields are invalid. If a pattern contains duplicate fields, EventBridge only considers the final field value.

For example, the following duplicate fields are invalid:

```json
## has duplicate keys
{  
  "source" : ["aws.s3"],
  "source" : ["aws.sns"],
  "detail-type" : ["AWS API Call via CloudTrail"],
  "detail" : {
    "eventSource" : ["s3.amazonaws.com"],
    "eventSource" : ["sns.amazonaws.com"
  }
}
```

```json
## has unique keys
{  
  "source" : ["aws.sns"],
  "detail-type" : ["AWS API Call via CloudTrail"],
  "detail" : { "eventSource" : ["sns.amazonaws.com"]
```
Comparison operations for use in event patterns

Below a summary of all the comparison operators available in EventBridge.

Comparison operators only work on leaf nodes, with the exception of $or and anything-but.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Example</th>
<th>Rule syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>Location is &quot;New York&quot; and Day is &quot;Monday&quot;</td>
<td>&quot;Location&quot;: [ &quot;New York&quot; ], &quot;Day&quot;: [&quot;Monday&quot;]</td>
</tr>
<tr>
<td>Begins with (p. 70)</td>
<td>Region is in the US</td>
<td>&quot;Region&quot;: [ {&quot;prefix&quot;: &quot;us-&quot; } ]</td>
</tr>
<tr>
<td>Empty (p. 68)</td>
<td>LastName is empty</td>
<td>&quot;LastName&quot;: [&quot;&quot;&quot;]</td>
</tr>
<tr>
<td>Equals</td>
<td>Name is &quot;Alice&quot;</td>
<td>&quot;Name&quot;: [ &quot;Alice&quot; ]</td>
</tr>
<tr>
<td>Equals (ignore case) (p. 73)</td>
<td>Name is &quot;Alice&quot;</td>
<td>&quot;Name&quot;: [ { &quot;equals-ignore-case&quot;: &quot;alice&quot; } ]</td>
</tr>
<tr>
<td>Ends with (p. 70)</td>
<td>FileName ends with a .png extension</td>
<td>&quot;FileName&quot;: [ { &quot;suffix&quot;: &quot;.png&quot; } ]</td>
</tr>
<tr>
<td>Exists (p. 72)</td>
<td>ProductName exists</td>
<td>&quot;ProductName&quot;: [ { &quot;exists&quot;: true } ]</td>
</tr>
<tr>
<td>Does not exist (p. 72)</td>
<td>ProductName does not exist</td>
<td>&quot;ProductName&quot;: [ { &quot;exists&quot;: false } ]</td>
</tr>
</tbody>
</table>
Example events and event patterns

You can use all of the JSON data types and values to match events. The following examples show events and the event patterns that match them.

Field matching

You can match on the value of a field. Consider the following Amazon EC2 Auto Scaling event.

```
{
  "version": "0",
  "id": "5e3c153a-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
  }
}
```

For the preceding event, you can use the "responseElements" field to match.

```
{
  "source": ["aws.autoscaling"],
  "detail-type": ["EC2 Instance Launch Successful"],
  "detail": {
    "responseElements": [null]
  }
}
```
Value matching

Consider the following Amazon Macie event, which is truncated.

```json
{
"version": "0",
"id": "0948ba87-d3b8-c6d4-f2da-732a1example",
"detail-type": "Macie Finding",
"source": "aws.macie",
"account": "123456789012",
"time": "2021-04-29T23:12:15Z",
"region": "us-east-1",
"resources": [

],
"detail": {
"schemaVersion": "1.0",
"id": "64b917aa-3843-014c-91d8-937ffexample",
"accountId": "123456789012",
"partition": "aws",
"region": "us-east-1",
"type": "Policy:IAMUser/S3BucketEncryptionDisabled",
"title": "Encryption is disabled for the S3 bucket",
"description": "Encryption is disabled for the Amazon S3 bucket. The data in the bucket isn't encrypted using server-side encryption.",
"severity": {
"score": 1,
"description": "Low"
},
"createdAt": "2021-04-29T15:46:02Z",
"updatedAt": "2021-04-29T23:12:15Z",
"count": 2,
}
```.

The following event pattern matches any event that has a severity score of 1 and a count of 2.

```json
{
"source": ["aws.macie"],
"detail-type": ["Macie Finding"],
"detail": {
"severity": {
"score": [1]
},
"count": [2]
}
}
```
Matching null values and empty strings in Amazon EventBridge event patterns

**Important**

In EventBridge, it is possible to create rules that can lead to higher-than-expected charges and throttling. For example, you can inadvertently create a rule that leads to an infinite loop, where a rule is fired recursively without end. Suppose you created a rule to detect that ACLs have changed on an Amazon 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.

For guidance on how to write precise rules and event patterns to minimize such unexpected results, see ???(p. 106) and ???(p. 78).

You can create an event pattern (p. 62) that matches a field in an event (p. 27) that has a null value or is an empty string. Consider the following example event.

See best practices to avoid higher than expected charges and throttling

```json
{
  "version": "0",
  "id": "3e3c153a-8339-4e30-8c35-687ebeff853fe",
  "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 event pattern, which matches the preceding event.

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

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

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

**Note**

Null values and empty strings are not interchangeable in pattern matching. An event pattern that matches empty strings doesn't match values of `null`. 
Arrays in Amazon EventBridge event patterns

The value of each field in an event pattern (p. 62) is an array containing one or more values. An event pattern matches the event (p. 27) 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 event pattern matches if the intersection of the event pattern array and the event array is non-empty.

**Important**
In EventBridge, it is possible to create rules that can lead to higher-than-expected charges and throttling. For example, you can inadvertently create a rule that leads to an infinite loop, where a rule is fired recursively without end. Suppose you created a rule to detect that ACLs have changed on an Amazon 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.

For guidance on how to write precise rules and event patterns to minimize such unexpected results, see ??? (p. 106) and ??? (p. 78).

For example, consider an event pattern that includes the following field.

```
"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 preceding event pattern matches an event that includes the following field because the first item in the event pattern array matches the second item in the event array.

```
"resources": [
  "arn:aws:ec2:us-east-1:123456789012:instance/i-b188560f"
]
```
Content filtering in Amazon EventBridge event patterns

Amazon EventBridge supports declarative content filtering using event patterns (p. 62). With content filtering, you can write complex event patterns that only match events under very specific conditions. For example, you can create an event pattern that matches an event when:

- A field of the event is within a specific numeric range.
- The event comes from a specific IP address.
- A specific field doesn't exist in the event JSON.

Important
In EventBridge, it is possible to create rules that can lead to higher-than-expected charges and throttling. For example, you can inadvertently create a rule that leads to an infinite loop, where a rule is fired recursively without end. Suppose you created a rule to detect that ACLs have changed on an Amazon 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.

For guidance on how to write precise rules and event patterns to minimize such unexpected results, see ???(p. 106) and ???(p. 78).

Filter types
- Prefix matching (p. 70)
- Suffix matching (p. 70)
- Anything-but matching (p. 71)
- Numeric matching (p. 72)
- IP address matching (p. 72)
- Exists matching (p. 72)
- Equals-ignore-case matching (p. 73)
- Matching using wildcards (p. 73)
- Complex example with multiple matching (p. 74)
- Complex example with $or matching (p. 75)

Prefix matching

You can match an event depending on the prefix of a value in the event source. You can use prefix matching for string values.

For example, the following event pattern would match any event where the "time" field started with "2017-10-02" such as "time": "2017-10-02T18:43:48Z".

```json
{
  "time": [ { "prefix": "2017-10-02" } ]
}
```

Suffix matching

You can match an event depending on the suffix of a value in the event source. You can use suffix matching for string values.
For example, the following event pattern would match any event where the "FileName" field ends with the .png file extension.

```json
{
   "FileName": [ { "suffix": ".png" } ]
}
```

**Anything-but matching**

*Anything-but* matching matches anything except what's provided in the rule.

You can use anything-but matching with strings and numeric values, including lists that contain only strings, or only numbers.

The following event pattern shows anything-but matching with strings and numbers.

```json
{
   "detail": {
      "state": [ { "anything-but": "initializing" } ]
   }
}
```

```json
{
   "detail": {
      "x-limit": [ { "anything-but": 123 } ]
   }
}
```

The following event pattern shows anything-but matching with a list of strings.

```json
{
   "detail": {
      "state": [ { "anything-but": [ "stopped", "overloaded" ] } ]
   }
}
```

The following event pattern shows anything-but matching with a list of numbers.

```json
{
   "detail": {
      "x-limit": [ { "anything-but": [ 100, 200, 300 ] } ]
   }
}
```

The following event pattern shows anything-but matching that matches any event that doesn't have the prefix "init" in the "state" field.

**Note**

*Anything-but* matching only works with a single prefix, not a list.

```json
{
   "detail": {
      "state": [ { "anything-but": { "prefix": "init" } } ]
   }
}
```
Numeric matching

Numeric matching works with values that are JSON numbers. It is limited to values between -5.0e9 and +5.0e9 inclusive, with 15 digits of precision, or six digits to the right of the decimal point.

The following shows numeric matching for an event pattern that only matches events that are true for all fields.

```json
{
    "detail": {
        "c-count": [ { "numeric": [ ">", 0, "\leq", 5 ] } ],
        "d-count": [ { "numeric": [ "\leq", 10 ] } ],
        "x-limit": [ { "numeric": [ "\approx", 3.018e2 ] } ]
    }
}
```

IP address matching

You can use IP address matching for IPv4 and IPv6 addresses. The following event pattern shows IP address matching to IP addresses that start with 10.0.0 and end with a number between 0 and 255.

```json
{
    "detail": {
        "sourceIPAddress": [ { "cidr": "10.0.0.0/24" } ]
    }
}
```

Exists matching

*Exists matching* works on the presence or absence of a field in the JSON of the event.

Exists matching only works on leaf nodes. It does not work on intermediate nodes.

The following event pattern matches any event that has a `detail.state` field.

```json
{
    "detail": {
        "state": [ { "exists": true } ]
    }
}
```

The preceding event pattern matches the following event.

```json
{
    "version": "0",
    "id": "7bf73129-1428-4cd3-a780-95db273d1602",
    "detail-type": "EC2 Instance State-change Notification",
    "source": "aws.ec2",
    "account": "123456789012",
    "time": "2015-11-11T21:29:54Z",
    "region": "us-east-1",
    "resources": ["arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"],
    "detail": {
        "instance-id": "i-abcd1111",
        "state": "pending"
    }
}
```
The preceding event pattern does NOT match the following event because it doesn't have a `detail.state` field.

```
{
  "detail-type": [ "EC2 Instance State-change Notification" ],
  "resources": [ "arn:aws:ec2:us-east-1:123456789012:instance/i-02ebd4584a2ebd341" ],
  "detail": {
    "c-count" : {
      "c1" : 100
    }
  }
}
```

Equals-ignore-case matching

`Equals-ignore-case` matching works on string values regardless of case.

The following event pattern matches any event that has a `detail-type` field that matches the specified string, regardless of case.

```
{
  "detail-type": [ { "equals-ignore-case": "ec2 instance state-change notification" } ]
}
```

The preceding event pattern matches the following event.

```
{
  "detail-type": [ "EC2 Instance State-change Notification" ],
  "resources": [ "arn:aws:ec2:us-east-1:123456789012:instance/i-02ebd4584a2ebd341" ],
  "detail": {
    "c-count" : {
      "c1" : 100
    }
  }
}
```

Matching using wildcards

You can use the wildcard character (*) to match string values in event patterns.

**Note**

Currently the wildcard character is supported in event bus rules only.

Considerations when using wildcards in your event patterns:

- You can specify any number of wildcard characters in a given string value; however, consecutive wildcard characters are not supported.
- `EventBridge` supports using the backslash character (\) to specify the literal * and \ characters in wildcard filters:
  - The string \* represents the literal * character
  - The string \\ represents the literal \ character

Using the backslash to escape other characters is not supported.
Wildcards and event pattern complexity

There is a limit to how complex a rule using wildcards can be. If a rule is too complex, EventBridge returns an `InvalidEventPatternException` when attempting to create the rule. If your rule generates such an error, consider using the guidance below to reduce the complexity of the event pattern:

- **Reduce the number of wildcard characters used**

  Only use wildcard characters where you truly need to match against multiple possible values. For example, consider the following event pattern, where you want to match against event buses in the same Region:

  ```json
  {  
    "EventBusArn": [ { "wildcard": "*:*:*:*:*:event-bus/*" } ]
  }
  ```

  In the above case, many of the sections of the ARN will be directly based on the Region in which your event buses reside. So if you are using the `us-east-1` Region, a less complex pattern that still matches the desired values might be the following example:

  ```json
  {  
    "EventBusArn": [ { "wildcard": "arn:aws:events:us-east-1:*:event-bus/*" } ]
  }
  ```

- **Reduce repeating character sequences that occur after a wildcard character**

  Having the same character sequence appear multiple times after the use of a wildcard increases the complexity of processing the event pattern. Recast your event pattern to minimize repeated sequences. For example, consider the following example, that matches on the file name `doc.txt` file for any user:

  ```json
  {  
    "FileName": [ { "wildcard": "/Users/*/dir/dir/dir/dir/dir/doc.txt" } ]
  }
  ```

  If you knew that the `doc.txt` file would only occur in the specified path, you could reduce the repeated character sequence in this way:

  ```json
  {  
    "FileName": [ { "wildcard": "/Users/*/doc.txt" } ]
  }
  ```

Complex example with multiple matching

You can combine multiple matching rules into a more complex event pattern. For example, the following event pattern combines anything-but and numeric.

```json
{  
  "time": [ { "prefix": "2017-10-02" } ],
  "detail": {
    "state": [ { "anything-but": "initializing" } ],
    "c-count": [ { "numeric": [ ">", 0, "<=" , 5 ] } ],
    "d-count": [ { "numeric": [ "<", 10 ] } ],
    "x-limit": [ { "anything-but": [ 100, 200, 300 ] } ]
  }
}
Complex example with $or matching

You can also create complex event patterns that check to see if any field values match, across multiple fields. Use $or to create an event pattern that matches if any of the values for multiple fields are matched.

Note that you can include other filter types, such as numeric matching (p. 72) and arrays (p. 69), in your pattern matching for individual fields in your $or construct.

The following event pattern matches if any of the following conditions are met:

- The c-count field is greater than 0 or less than or equal to 5.
- The d-count field is less than 10.
- The x-limit field equals 3.018e2.

```json
{  
  "detail": {  
    "$or": [  
      { "c-count": [ { "numeric": [ ">", 0, "\"<=\"", 5 ] } ] },  
      { "d-count": [ { "numeric": [ "\"<\"", 10 ] } ] },  
      { "x-limit": [ { "numeric": [ "\"=\"", 3.018e2 ] } ] }  
    ]  
  }  
}
```

Note

APIs that accept an event pattern (such as PutRule, CreateArchive, UpdateArchive, and TestEventPattern) will throw an InvalidEventPatternException if the use of $or results in over 1000 rule combinations.
To determine the number of rule combinations in an event pattern, multiply the total number of arguments from each $or array in the event pattern. For example, the above pattern contains a single $or array with three arguments, so the total number of rule combinations is also three. If you added another $or array with two arguments, the total rule combinations would then be six.
Testing an event pattern using the EventBridge Sandbox

Rules use event patterns to select events and send them to targets. Event patterns have the same structure as the events they match. An event pattern either matches an event or it doesn't.

Defining an event pattern is typically part of the larger process of creating a new rule (p. 84) or editing an existing one. Using the Sandbox in EventBridge, however, you can quickly define an event pattern and use a sample event to confirm the pattern matches the desired events, without having to create or edit a rule. Once you've got your event pattern tested, EventBridge give you the option of creating a new rule using that event pattern directly from the sandbox.

For more information about event patterns, see ??? (p. 62).

**Important**

In EventBridge, it is possible to create rules that can lead to higher-than-expected charges and throttling. For example, you can inadvertently create a rule that leads to an infinite loop, where a rule is fired recursively without end. Suppose you created a rule to detect that ACLs have changed on an Amazon 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.

For guidance on how to write precise rules and event patterns to minimize such unexpected results, see ??? (p. 106) and ??? (p. 78).

To test an event pattern using the EventBridge sandbox

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose Developer resources, then select Sandbox, and on the Sandbox page choose the Event pattern tab.
3. For Event source, choose AWS events or EventBridge partner events.
4. In the Sample events section, choose a Sample event type against which you want to test your event pattern.

The following sample event types are available:

- **AWS events** – Select from events emitted from supported AWS services.
- **EventBridge partner events** – Select from events emitted from third-party services that support EventBridge, such as Salesforce.
- **Enter my own** – Enter your own event in JSON text.

You can also use an AWS or partner event as the starting point for creating your own custom event.

1. Select AWS events or EventBridge partner events.
2. Use the Sample events dropdown to select the event you want to use as a starting point for your custom event.

EventBridge displays the sample event.

3. Select Copy.
4. Select Enter my own for Event type.
5. Delete the sample event structure in the JSON editing pane, and paste the AWS or partner event in its place.
6. Edit the event JSON to create your own sample event.

5. Choose a Creation method. You can create an event pattern from an EventBridge schema or template, or you can create a custom event pattern.
Existing schema

To use an existing EventBridge schema to create the event pattern, do the following:

1. In the **Creation method** section, for **Method**, select **Use schema**.
2. In the **Event pattern** section, for **Schema type**, select **Select schema from Schema registry**.
3. For **Schema registry**, choose the dropdown box and enter the name of a schema registry, such as aws.events. You can also select an option from the dropdown list that appears.
4. For **Schema**, choose the dropdown box and enter the name of the schema to use. For example, aws.s3@ObjectDeleted. You can also select an option from the dropdown list that appears.
5. In the **Models** section, choose the **Edit** button next to any attribute to open its properties. Set the **Relationship** and **Value** fields as needed, then choose **Set** to save the attribute.

   **Note**
   For information about an attribute's definition, choose the **Info** icon next to the attribute's name. For a reference on how to set attribute properties in your event, open the **Note** section of the attribute properties dialog box.
   To delete an attribute's properties, choose the **Edit** button for that attribute, then choose **Clear**.

6. Choose **Generate event pattern in JSON** to generate and validate your event pattern as JSON text.
7. To test the sample event against your test pattern, choose **Test pattern**.

   EventBridge displays a message box stating whether your sample event matches the event pattern.

   You can also choose any of the following options:
   - **Copy** – Copy the event pattern to your device's clipboard.
   - **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.

Custom schema

To write a custom schema and convert it to an event pattern, do the following:

1. In the **Creation method** section, for **Method**, choose **Use schema**.
2. In the **Event pattern** section, for **Schema type**, choose **Enter schema**.
3. Enter your schema into the text box. You must format the schema as valid JSON text.
4. In the **Models** section, choose the **Edit** button next to any attribute to open its properties. Set the **Relationship** and **Value** fields as needed, then choose **Set** to save the attribute.

   **Note**
   For information about an attribute's definition, choose the **Info** icon next to the attribute's name. For a reference on how to set attribute properties in your event, open the **Note** section of the attribute properties dialog box.
   To delete an attribute's properties, choose the **Edit** button for that attribute, then choose **Clear**.

5. Choose **Generate event pattern in JSON** to generate and validate your event pattern as JSON text.
6. To test the sample event against your test pattern, choose **Test pattern**.

   EventBridge displays a message box stating whether your sample event matches the event pattern.
You can also choose any of the following options:

- **Copy** – Copy the event pattern to your device's clipboard.
- **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.

### Event pattern

To write a custom event pattern in JSON format, do the following:

1. In the **Creation method** section, for **Method**, choose **Custom pattern (JSON editor)**.
2. For **Event pattern**, enter your custom event pattern in JSON-formatted text.
3. To test the sample event against your test pattern, choose **Test pattern**.

EventBridge displays a message box stating whether your sample event matches the event pattern.

You can also choose any of the following options:

- **Copy** – Copy the event pattern to your device's clipboard.
- **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.
- **Event pattern form** – Opens the event pattern in Pattern Builder. If the pattern can't be rendered in Pattern Builder as-is, EventBridge warns you before it opens Pattern Builder.

6. *(Optional)* To create a rule with this event pattern, and assign the rule to a specific event bus, choose **Create rule with pattern**.

EventBridge takes you to **Step 1 of Create rule**, which you can use to create a rule and assign it to the event bus of your choice.

Note that **Step 2 - Build event pattern** contains the event pattern information you've already specified, and which you can accept or update.

For more on how to create rules, see ?? (p. 84).

---

### Best practices when defining Amazon EventBridge event patterns

Below are some best practices to consider when defining event patterns in your event bus rules.

#### Avoid writing infinite loops

In EventBridge, 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 these issues, write the event patterns for your rules to be as precise as possible, so they only match the events you actually want sent to the target. In the above example, you would create an event pattern to match events so that the triggered actions do not re-fire the same rule. For example, create an event pattern in your rule that would match events only if ACLs are found to be in a bad state, instead of after any change. For more information, see ?? (p. 79) and ?? (p. 80).

An infinite loop can quickly cause higher than expected charges. It can also lead to throttling and delayed event delivery. You can monitor the upper bound of your invocation rates to be warned about unexpected spikes in volume.
Use budgeting to alert you when charges exceed your specified limit. For more information, see Managing Your Costs with Budgets.

Make event patterns precise as possible

The more precise your event pattern, the more likely it will match only the events you actually want it to, and avoid unexpected matches when new events are added to an event source, or existing events are updated to include new properties.

Event patterns can include filters that match on:

- Event metadata about the event, such as source, detail-type, account, or region.
- Event data, this is, the fields inside the detail object.
- Event content, or the actual values of the fields inside the detail object.

Most patterns are simple, such as specifying only source and detail-type filters. However, EventBridge patterns include the flexibility to filter on any key or value of the event. In addition, you can apply content filters such as prefix and suffix filters to improve the precision of your patterns. For more information, see ???(p. 70).

Specify event source and detail type as filters

You can reduce generating infinite loops and matching undesired events by making your event patterns more precise using the source and detail-type metadata fields.

When you need to match specific values within two or more fields, use the $or comparison operator, rather than listing all possible values within a single array of values.

For events that are delivered through AWS CloudTrail, we recommend you use the eventName field as a filter.

The following event pattern example matches CreateQueue or SetQueueAttributes from the Amazon Simple Queue Service service, or CreateKey or DisableKeyRotation events from the AWS Key Management Service service.

```json
{
  "detail-type": ["AWS API Call via CloudTrail"],
  "$or": [{
    "source": ["aws.sqs"],
    "detail": {
      "eventName": ["CreateQueue", "SetQueueAttributes"
    ]
  },
  {
    "source": ["aws.kms"],
    "detail": {
      "eventName": ["CreateKey", "DisableKeyRotation"
    ]
  }
}
```
Specify account and region as filters

Including account and region fields in your event pattern helps limit cross-account or cross-region event matching.

Specify content filters

Content-based filtering can help improve event pattern precision, while still keeping the length of the event pattern to a minimum. For example, matching based on a numeric range can be helpful instead of listing all possible numeric values.

For more information, see ??? (p. 70).

Scope your event patterns to account for event source updates

When creating event patterns, you should take into account that event schemas and event domains may evolve and expand over time. Here again, making your event patterns as precise as possible helps you limit unexpected matches if the event source changes or expands.

For example, suppose you are matching against events from a new micro-service that publishes payment-related events. Initially, the service uses the domain `acme.payments`, and publishes a single event, `Payment accepted`:

```json
{
  "detail-type": "Payment accepted",
  "source": "acme.payments",
  "detail": {
    "type": "credit",
    "amount": "100",
    "date": "2023-06-10",
    "currency": "USD"
  }
}
```

At this point, you could create a simple event pattern that matches Payment accepted events:

```json
{
  "source": "acme.payments"
}
```

However, suppose the service later introduces a new event for rejected payments:

```json
{
  "detail-type": "Payment rejected",
  "source": "acme.payments",
  "detail": {

  }
}
```

In this case, the simple event pattern you created will now match against both Payment accepted and Payment rejected events. EventBridge routes both types of events to the specified target for processing, possibly introducing processing failures and additional processing cost.
To scope your event pattern to only Payment accepted events, you'd want to specify both source and detail-type, at a minimum:

```json
{
    "detail-type": "Payment accepted",
    "source": "acme.payments"
}
```

You can also specify account and Region in your event pattern, to further limit when cross-account or cross-Region events match this rule.

```json
{
    "account": "012345678910",
    "source": "acme.payments",
    "region": "AWS-Region",
    "detail-type": "Payment accepted"
}
```

**Validate event patterns**

To ensure rules match the desired events, we strongly recommend you validate your event patterns. You can validate your event patterns using the EventBridge console or API:

- In the EventBridge console, you can create and test event patterns as part of creating a rule (p. 84), or separately by using the Sandbox (p. 76).
- You can test your event patterns programmatically using the TestEventPattern action.
Amazon EventBridge rules

A rule matches incoming events (p. 27) and sends them to targets (p. 112) for processing. A single rule can send an event to multiple targets, which then run in parallel. Rules are based either on an event pattern (p. 62) or a schedule. An event pattern defines the event structure and the fields that a rule matches. Rules that are based on a schedule perform an action at regular intervals.

AWS services can create and manage EventBridge 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 can also create an IAM policy (p. 388) that grants permission 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, but you should only delete them if you’re 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.

The following video goes over the basics of rules: What are rules

Topics

- Creating Amazon EventBridge rules that react to events (p. 84)
- Using Amazon EventBridge Scheduler with Amazon EventBridge (p. 92)
- Creating an Amazon EventBridge rule that runs on a schedule (p. 95)
- Disabling or deleting an Amazon EventBridge rule (p. 106)
- Best practices when defining Amazon EventBridge rules (p. 106)
• Using Amazon EventBridge and AWS Serverless Application Model templates (p. 108)
• Generate an AWS CloudFormation template from Amazon EventBridge rules (p. 110)
Creating Amazon EventBridge rules that react to events

To take action on events received by Amazon EventBridge, you can create rules. When an event matches the event pattern defined in your rule, EventBridge sends the event to the specified target and triggers the action defined in the rule.

The following video explores creating different kinds of rules and how to test them: Learning about rules.

Use the following procedure to create an Amazon EventBridge rule that responds to events.

Create a rule that reacts to events

The following steps walk you through how to create a rule that EventBridge uses to match events as they are sent to the specified event bus.

Steps

- Define the rule
- Build the event pattern
- Select targets
- Configure tags and review rule

Define the rule

First, enter a name and description for your rule to identify it. You must also define the event bus where your rule looks for events to match to an event pattern.
To define the rule detail

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a Name and, optionally, a Description for the rule.
   A rule can't have the same name as another rule in the same AWS Region and on the same event bus.
5. For Event bus, choose the event bus to associate with this rule. If you want this rule to match events that come from your account, select AWS default event bus. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.

Build the event pattern

Next, build the event pattern. To do this, specify the event source, choose the basis for the event pattern, and define the attributes and values to match on. You can also generate the event pattern in JSON and test it against a sample event.

To build the event pattern

1. For Event source, choose AWS events or EventBridge partner events.
2. (Optional) In the Sample events section, choose a Sample event type against which you want to test your event pattern.
   The following sample event types are available:
   - AWS events – Select from events emitted from supported AWS services.
   - EventBridge partner events – Select from events emitted from third-party services that support EventBridge, such as Salesforce.
   - Enter my own – Enter your own event in JSON text.
   You can also use an AWS or partner event as the starting point for creating your own custom event.
   1. Select AWS events or EventBridge partner events.
   2. Use the Sample events dropdown to select the event you want to use as a starting point for your custom event.
   EventBridge displays the sample event.
   3. Select Copy.
   4. Select Enter my own for Event type.
   5. Delete the sample event structure in the JSON editing pane, and paste the AWS or partner event in its place.
   6. Edit the event JSON to create your own sample event.
3. Choose a Creation method. You can create an event pattern from an EventBridge schema or template, or you can create a custom event pattern.

Existing schema

   To use an existing EventBridge schema to create the event pattern, do the following:
   1. In the Creation method section, for Method, select Use schema.
2. In the **Event pattern** section, for **Schema type**, select **Select schema from Schema registry**.

3. For **Schema registry**, choose the dropdown box and enter the name of a schema registry, such as `aws.events`. You can also select an option from the dropdown list that appears.

4. For **Schema**, choose the dropdown box and enter the name of the schema to use. For example, `aws.s3@ObjectDeleted`. You can also select an option from the dropdown list that appears.

5. In the **Models** section, choose the **Edit** button next to any attribute to open its properties. Set the **Relationship** and **Value** fields as needed, then choose **Set** to save the attribute.

   **Note**
   
   For information about an attribute's definition, choose the **Info** icon next to the attribute's name. For a reference on how to set attribute properties in your event, open the **Note** section of the attribute properties dialog box. To delete an attribute's properties, choose the **Edit** button for that attribute, then choose **Clear**.

6. Choose **Generate event pattern in JSON** to generate and validate your event pattern as JSON text.

7. (Optional) To test the sample event against your test pattern, choose **Test pattern**.

   EventBridge displays a message box stating whether your sample event matches the event pattern.

   You can also choose any of the following options:
   
   - **Copy** – Copy the event pattern to your device's clipboard.
   
   - **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.

Custom schema

To write a custom schema and convert it to an event pattern, do the following:

1. In the **Creation method** section, for **Method**, choose **Use schema**.

2. In the **Event pattern** section, for **Schema type**, choose **Enter schema**.

3. Enter your schema into the text box. You must format the schema as valid JSON text.

4. In the **Models** section, choose the **Edit** button next to any attribute to open its properties. Set the **Relationship** and **Value** fields as needed, then choose **Set** to save the attribute.

   **Note**
   
   For information about an attribute's definition, choose the **Info** icon next to the attribute's name. For a reference on how to set attribute properties in your event, open the **Note** section of the attribute properties dialog box. To delete an attribute's properties, choose the **Edit** button for that attribute, then choose **Clear**.

5. Choose **Generate event pattern in JSON** to generate and validate your event pattern as JSON text.

6. (Optional) To test the sample event against your test pattern, choose **Test pattern**.

   EventBridge displays a message box stating whether your sample event matches the event pattern.

   You can also choose any of the following options:
   
   - **Copy** – Copy the event pattern to your device's clipboard.
   
   - **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.
Event pattern

To write a custom event pattern in JSON format, do the following:

1. In the **Creation method** section, for **Method**, choose **Custom pattern (JSON editor)**.
2. For **Event pattern**, enter your custom event pattern in JSON-formatted text.
3. (Optional) To test the sample event against your test pattern, choose **Test pattern**.

   EventBridge displays a message box stating whether your sample event matches the event pattern.

   You can also choose any of the following options:
   - **Copy** – Copy the event pattern to your device's clipboard.
   - **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.
   - **Event pattern form** – Opens the event pattern in Pattern Builder. If the pattern can't be rendered in Pattern Builder as-is, EventBridge warns you before it opens Pattern Builder.

4. Choose **Next**.

Select targets

Choose one or more targets to receive events that match the specified pattern. Targets can include an EventBridge event bus, EventBridge API destinations, including SaaS partners such as Salesforce, or another AWS service.

**To select targets**

1. For **Target type**, choose one of the following target types:

   **Event bus**

   To select an EventBridge event bus, select **EventBridge event bus**, then do the following:

   • To use an event bus in the same AWS Region as this rule:
     1. Select **Event bus in the same account and Region**.
     2. For **Event bus for target**, choose the dropdown box and enter the name of the event bus.
        You can also select the event bus from the dropdown list.

        For more information, see ??? (p. 135).
   • To use an event bus in a different AWS Region or account as this rule:
     1. Select **Event bus in a different account or Region**.
     2. For **Event bus as target**, enter the ARN of the event bus you want to use.

        For more information, see:
        • ??? (p. 131)
        • ??? (p. 133)

   **API destination**

   To use an EventBridge API destination, select **EventBridge API destination**, then do one of the following:

   • To use an existing API destination, select **Use an existing API destination**. Then select an API destination from the dropdown list.
To create a new API destination, select **Create a new API destination**. Then, provide the following details for the destination:

- **Name** – Enter a name for the destination.
  
  Names must be unique within your AWS account. Names can have up to 64 characters. Valid characters are `A-Z`, `a-z`, `0-9`, and `_` (hyphen).

- **(Optional) Description** – Enter a description for the destination.
  
  Descriptions can have up to 512 characters.

- **API destination endpoint** – The URL endpoint for the target.
  
  The endpoint URL must start with `https`. You can include the `*` as a path parameter wildcard. You can set path parameters from the target’s `HttpParameters` attribute.

- **HTTP method** – Select the HTTP method used when you invoke the endpoint.

- **(Optional) Invocation rate limit per second** – Enter the maximum number of invocations accepted for each second for this destination.
  
  This value must be greater than zero. By default, this value is set to 300.

- **Connection** – Choose to use a new or existing connection:
  
  - To use an existing connection, select **Use an existing connection** and select the connection from the dropdown list.
  
  - To create a new connection for this destination select **Create a new connection**, then define the connection’s **Name**, **Destination type**, and **Authorization type**. You can also add an optional **Description** for this connection.

For more information, see ???.

**AWS service**

To use an AWS service, select **AWS service**, then do the following:

1. For **Select a target**, select an AWS service to use as the target. Provide the information requested for the service you select.

   **Note**
   
   The fields displayed vary depending on the service selected. For more information about available targets, see Targets available in the EventBridge console (p. 112).

2. For many target types, EventBridge needs permissions to send events to the target. In these cases, EventBridge can create the IAM role needed for your rule to run.

   For **Execution role**, do one of the following:
   
   - To create a new execution role for this rule:
     
     a. Select **Create a new role for this specific resource**.
     
     b. Either enter a name for this execution role, or use the name generated by EventBridge.

   - To use an existing execution role for this rule:
     
     a. Select **Use existing role**.
     
     b. Enter or select the name of the execution role to use from the dropdown list.

3. **(Optional) For Additional settings**, specify any of the optional settings available for your target type:
Event bus

(Optional) For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:

- Choose **None** to not use a dead-letter queue.
- Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the drop-down list.
- Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it.

For more information, see **Granting permissions to the dead-letter queue (p. 60)**.

API destination

1. (Optional) For **Configure target input**, choose how you want to customize the text sent to the target for matching events. Choose one of the following:
   - **Matched events** – EventBridge sends the entire original source event to the target. This is the default.
   - **Part of the matched events** – EventBridge only sends the specified portion of the original source event to the target.

   Under **Specify the part of the matched event**, specify a JSON path that defines the part of the event you want EventBridge to send to the target.

   - **Constant (JSON text)** – EventBridge sends only the specified JSON text to the target. No part of the original source event is sent.

   Under **Specify the constant in JSON**, specify the JSON text that you want EventBridge to send to the target instead of the event.

   - **Input transformer** – Configure an input transformer to customize the text you want EventBridge send to the target. For more information, see **??? (p. 137)**.
     a. Select **Configure input transformer**.
     b. Configure the input transformer following the steps in **??? (p. 141)**.

2. (Optional) Under **Retry policy**, specify how EventBridge should retry sending an event to a target after an error occurs.
   - **Maximum age of event** – Enter the maximum amount of time (in hours, minutes, and seconds) for EventBridge to retain unprocessed events. The default is 24 hours.
   - **Retry attempts** – Enter the maximum number of times EventBridge should retry sending an event to the target after an error occurs. The default is 185 times.

3. (Optional) For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
   - Choose **None** to not use a dead-letter queue.
   - Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the drop-down list.
   - Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it.

For more information, see **Granting permissions to the dead-letter queue (p. 60)**.
Create a rule that reacts to events

AWS service

Note that EventBridge may not display all of the following fields for a given AWS service.

1. (Optional) For Configure target input, choose how you want to customize the text sent to the target for matching events. Choose one of the following:
   - **Matched events** – EventBridge sends the entire original source event to the target. This is the default.
   - **Part of the matched events** – EventBridge only sends the specified portion of the original source event to the target.
   
   Under **Specify the part of the matched event**, specify a JSON path that defines the part of the event you want EventBridge to send to the target.
   - **Constant (JSON text)** – EventBridge sends only the specified JSON text to the target. No part of the original source event is sent.
     
     Under **Specify the constant in JSON**, specify the JSON text that you want EventBridge to send to the target instead of the event.
   - **Input transformer** – Configure an input transformer to customize the text you want EventBridge to send to the target. For more information, see ??? (p. 137).
     a. Select **Configure input transformer**.
     b. Configure the input transformer following the steps in ??? (p. 141).

2. (Optional) Under **Retry policy**, specify how EventBridge should retry sending an event to a target after an error occurs.
   - **Maximum age of event** – Enter the maximum amount of time (in hours, minutes, and seconds) for EventBridge to retain unprocessed events. The default is 24 hours.
   - **Retry attempts** – Enter the maximum number of times EventBridge should retry sending an event to the target after an error occurs. The default is 185 times.

3. (Optional) For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
   - Choose **None** to not use a dead-letter queue.
   - Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the drop-down list.
   - Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it.

   For more information, see Granting permissions to the dead-letter queue (p. 60).

4. (Optional) Choose **Add another target** to add another target for this rule.

5. Choose **Next**.

Note that EventBridge may not display all of the following fields for a given AWS service.

**Configure tags and review rule**

Finally, enter any desired tags for the rule, then review and create the rule.

**To configure tags, and review and create the rule**

1. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 453).
2. Choose Next.
3. Review the details for the new rule. To make changes to any section, choose the Edit button next to that section.

When satisfied with the rule details, choose Create rule.
Using Amazon EventBridge Scheduler with Amazon EventBridge

Amazon EventBridge Scheduler is a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. With EventBridge Scheduler, you can create schedules using cron and rate expressions for recurring patterns, or configure one-time invocations. You can set up flexible time windows for delivery, define retry limits, and set the maximum retention time for failed API invocations.

EventBridge Scheduler is highly customizable, and offers improved scalability over EventBridge scheduled rules (p. 95), with a wider set of target API operations and AWS services. We recommend that you use EventBridge Scheduler to invoke targets on a schedule.

Topics

• Set up the execution role (p. 92)
• Create a schedule (p. 92)
• Related resources (p. 95)

Set up the execution role

When you create a new schedule, EventBridge Scheduler must have permission to invoke its target API operation on your behalf. You grant these permissions to EventBridge Scheduler using an execution role. The permission policy you attach to your schedule's execution role defines the required permissions. These permissions depend on the target API you want EventBridge Scheduler to invoke.

When you use the EventBridge Scheduler console to create a schedule, as in the following procedure, EventBridge Scheduler automatically sets up an execution role based on your selected target. If you want to create a schedule using one of the EventBridge Scheduler SDKs, the AWS CLI, or AWS CloudFormation, you must have an existing execution role that grants the permissions EventBridge Scheduler requires to invoke a target. For more information about manually setting up an execution role for your schedule, see Setting up an execution role in the EventBridge Scheduler User Guide.

Create a schedule

To create a schedule by using the console

2. On the Schedules page, choose Create schedule.
3. On the Specify schedule detail page, in the Schedule name and description section, do the following:
   a. For Schedule name, enter a name for your schedule. For example, MyTestSchedule.
   b. (Optional) For Description, enter a description for your schedule. For example, My first schedule.
   c. For Schedule group, choose a schedule group from the dropdown list. If you don't have a group, choose default. To create a schedule group, choose create your own schedule.

You use schedule groups to add tags to groups of schedules.
4. Choose your schedule options.
### Create a schedule

<table>
<thead>
<tr>
<th>Occurrence</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One-time schedule</strong></td>
<td>A one-time schedule invokes a target only once at the date and time that you specify.</td>
</tr>
<tr>
<td></td>
<td>For <strong>Date and time</strong>, do the following:</td>
</tr>
<tr>
<td></td>
<td>• Enter a valid date in YYYY/MM/DD format.</td>
</tr>
<tr>
<td></td>
<td>• Enter a timestamp in 24-hour hh:mm format.</td>
</tr>
<tr>
<td></td>
<td>• For <strong>Timezone</strong>, choose the timezone.</td>
</tr>
<tr>
<td><strong>Recurring schedule</strong></td>
<td>A recurring schedule invokes a target at a rate that you specify using a cron expression or rate expression.</td>
</tr>
<tr>
<td></td>
<td>a. For <strong>Schedule type</strong>, do one of the following:</td>
</tr>
<tr>
<td></td>
<td>• To use a cron expression to define the schedule, choose <strong>Cron-based schedule</strong> and enter the cron expression.</td>
</tr>
<tr>
<td></td>
<td>• To use a rate expression to define the schedule, choose <strong>Rate-based schedule</strong> and enter the rate expression.</td>
</tr>
<tr>
<td></td>
<td>For more information about cron and rate expressions, see <strong>Schedule types on EventBridge Scheduler</strong> in the Amazon EventBridge Scheduler User Guide.</td>
</tr>
<tr>
<td></td>
<td>b. For <strong>Flexible time window</strong>, choose <strong>Off</strong> to turn off the option, or choose one of the pre-defined time windows. For example, if you choose <strong>15 minutes</strong> and you set a recurring schedule to invoke its target once every hour, the schedule runs within 15 minutes after the start of every hour.</td>
</tr>
</tbody>
</table>

5. (Optional) If you chose **Recurring schedule** in the previous step, in the **Timeframe** section, do the following:
   a. For **Timezone**, choose a timezone.
   b. For **Start date and time**, enter a valid date in YYYY/MM/DD format, and then specify a timestamp in 24-hour hh:mm format.
   c. For **End date and time**, enter a valid date in YYYY/MM/DD format, and then specify a timestamp in 24-hour hh:mm format.

6. Choose **Next**.
7. On the **Select target** page, choose the AWS API operation that EventBridge Scheduler invokes:
   
a. For **Target API**, choose **Templated targets**.
   
b. Choose **Amazon EventBridge PutEvents**.
   
c. Under **PutEvents**, specify the following:
      
      • For **EventBridge event bus**, choose the event bus from the drop-down menu. For example, *default*.
         
         You can also create a new event bus in the EventBridge console by choosing **Create new event bus**.
      
      • For **Detail-type**, enter the detail type of the events you want to match. For example, *Object Created*.
      
      • For **Source**, enter the name of the service that is the source of the events.
         
         For AWS service events, specify the service prefix as the source. Do not include the `aws.` prefix. For example, for Amazon S3 events, enter `s3`.
         
         To determine a service's prefix, see [The condition keys table](#) in the *Service Authorization Reference*. For more information about source and detail-type event values, see [p. 29](#).
         
      • (Optional): For **Detail**, enter an event pattern to further filter the events EventBridge Scheduler sends to EventBridge.
         
         For more information, see [p. 62](#).

8. Choose **Next**.

9. On the **Settings** page, do the following:
   
a. To turn on the schedule, under **Schedule state**, toggle **Enable schedule**.
   
b. To configure a retry policy for your schedule, under **Retry policy and dead-letter queue (DLQ)**, do the following:
      
      • Toggle **Retry**.
      
      • For **Maximum age of event**, enter the maximum **hour(s)** and **min(s)** that EventBridge Scheduler must keep an unprocessed event.
         
         The maximum time is 24 hours.
      
      • For **Maximum retries**, enter the maximum number of times EventBridge Scheduler retries the schedule if the target returns an error.
         
         The maximum value is 185 retries.
         
         With retry policies, if a schedule fails to invoke its target, EventBridge Scheduler re-runs the schedule. If configured, you must set the maximum retention time and retries for the schedule.
   
c. Choose where EventBridge Scheduler stores undelivered events.

<table>
<thead>
<tr>
<th>Dead-letter queue (DLQ) option</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Don't store</td>
<td>Choose <strong>None</strong>.</td>
</tr>
<tr>
<td>Store the event in the same AWS account where you’re creating the schedule</td>
<td>a. Choose <strong>Select an Amazon SQS queue in my AWS account as a DLQ</strong>.</td>
</tr>
</tbody>
</table>
### Dead-letter queue (DLQ) option

<table>
<thead>
<tr>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b. Choose the Amazon Resource Name (ARN) of the Amazon SQS queue.</td>
</tr>
<tr>
<td>Store the event in a different AWS account from where you're creating the schedule</td>
</tr>
</tbody>
</table>
| a. Choose **Specify an Amazon SQS queue in other AWS accounts as a DLQ.**  
b. Enter the Amazon Resource Name (ARN) of the Amazon SQS queue. |

d. To use a customer managed key to encrypt your target input, under **Encryption**, choose **Customize encryption settings (advanced)**.

If you choose this option, enter an existing KMS key ARN or choose **Create an AWS KMS key** to navigate to the AWS KMS console. For more information about how EventBridge Scheduler encrypts your data at rest, see **Encryption at rest** in the *Amazon EventBridge Scheduler User Guide*.

e. To have EventBridge Scheduler create a new execution role for you, choose **Create new role for this schedule**. Then, enter a name for **Role name**. If you choose this option, EventBridge Scheduler attaches the required permissions necessary for your templated target to the role.

10. Choose **Next**.

11. In the **Review and create schedule** page, review the details of your schedule. In each section, choose **Edit** to go back to that step and edit its details.

12. Choose **Create schedule**.

You can view a list of your new and existing schedules on the **Schedules** page. Under the **Status** column, verify that your new schedule is **Enabled**.

### Related resources

For more information about EventBridge Scheduler, see the following:

- [EventBridge Scheduler User Guide](#)
- [EventBridge Scheduler API Reference](#)
- [EventBridge Scheduler Pricing](#)

### Creating an Amazon EventBridge rule that runs on a schedule

A **rule** (p. 82) can run in response to an **event** (p. 27), or at certain time intervals. For example, to periodically run an AWS Lambda function, you can create a rule to run on a schedule.

**Note**

EventBridge offers Amazon EventBridge Scheduler, a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. EventBridge Scheduler is highly customizable, and offers improved scalability over EventBridge scheduled rules, with a wider set of target API operations and AWS services.
We recommend that you use EventBridge Scheduler to invoke targets on a schedule. For more information, see ??? (p. 92).

In EventBridge, you can create two types of scheduled rules:

- Rules that run at a regular rate
  
  EventBridge runs these rules at regular intervals; for example, every 20 minutes.

  To specify the rate for a scheduled rule, you define a *rate expression*.

- Rules that run at specific times
  
  EventBridge runs these rules at specific times and dates; for example, 8:00 a.m. PST on the first Monday of every month.

  To specify the time and dates a scheduled rule runs, you define a *cron expression*.

Rate expressions are simpler to define, while cron expressions offer detailed schedule control. For example, with a cron expression, you can define a rule that runs at a specified time on a certain day of each week or month. In contrast, rate expressions run a rule at a regular rate, such as once every hour or once every day.

All scheduled events use UTC+0 time zone, and the minimum precision for a schedule is one minute.

**Note**

EventBridge doesn't provide second-level precision in schedule expressions. The finest resolution using a cron expression is one minute. Due to the distributed nature of EventBridge and the target services, there can be a delay of several seconds between the time the scheduled rule is triggered and the time the target service runs the target resource.

The following video gives an overview of scheduling tasks: [Creating scheduled tasks with EventBridge](#)
Create a rule that runs on a schedule

The following steps walk you through how to create an EventBridge rule that runs on a regular schedule.

Note
You can only create scheduled rules using the default event bus.

Steps

• Define the rule (p. 97)
• Define the schedule (p. 98)
• Select targets (p. 98)
• Configure tags and review rule (p. 102)

Define the rule

First, enter a name and description for your rule to identify it.

To define the rule detail

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a Name and, optionally, a Description for the rule. A rule can't have the same name as another rule in the same AWS Region and on the same event bus.
5. For Event bus, choose the default event bus. You can only create scheduled rules using the default event bus.
6. To have the rule take effect as soon as you create it, make sure the Enable the rule on the selected event bus option is enabled.
7. For Rule type, choose Schedule.

At this point, you can choose to continue with creating a rule that runs on a schedule, or use Amazon EventBridge Scheduler.
8. Choose how you want to continue:

• Use EventBridge Scheduler to create your schedule
  
  Note
EventBridge Scheduler is a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. It provides one-time and recurring scheduling functionality independent of event buses and rules. EventBridge Scheduler is highly customizable, and offers improved scalability over EventBridge scheduled rules, with a wider set of target API operations and AWS services. We recommend that you use EventBridge Scheduler to invoke targets on a schedule. For more information, see What is Amazon EventBridge Scheduler? in the Amazon EventBridge Scheduler User Guide.

  1. Select Continue in EventBridge Scheduler
EventBridge opens the EventBridge Scheduler console to the **Create schedule** page.

2. Click **Create the schedule** in the EventBridge Scheduler console.

   - Continue using EventBridge to create a scheduled rule for the default event bus
     1. Select **Continue to create rule**.

### Define the schedule

Next, define the schedule pattern.

#### To define the schedule pattern

1. For **Schedule pattern**, choose whether you want the schedule to run at a specific time, or at a regular rate:

   - **Specific time**
     1. Choose a **fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month**.
     2. For **Cron expression**, specify fields to define the cron expression that EventBridge should use to determine when to execute this scheduled rule.

     Once you have specified all fields, EventBridge displays the next ten dates when EventBridge will execute this scheduled rule. You can choose whether to display those dates in **UTC** or **Local time zone**.

     For more information on constructing a cron expression, see ???? (p. 102).

   - **Regular rate**
     1. Choose a **schedule that runs at a regular rate, such as every 10 minutes**.
     2. For **Rate expression**, specify the **Value** and **Unit** fields to define the rate at which EventBridge should execute this scheduled rule.

     For more information on constructing a rate expression, see ???? (p. 104).

2. Choose **Next**.

### Select targets

Choose one or more targets to receive events that match the specified pattern. Targets can include an EventBridge event bus, EventBridge API destinations, including SaaS partners such as Salesforce, or another AWS service.

#### To select targets

1. For **Target type**, choose one of the following target types:

   - **Event bus**

     To select an EventBridge event bus, select **EventBridge event bus**, then do the following:

     - To use an event bus in the same AWS Region as this rule:
       1. Select **Event bus in the same account and Region**.
       2. For **Event bus for target**, choose the dropdown box and enter the name of the event bus. You can also select the event bus from the dropdown list.
Create a rule that runs on a schedule

For more information, see ??? (p. 135).

- To use an event bus in a different AWS Region or account as this rule:
  1. Select Event bus in a different account or Region.
  2. For Event bus as target, enter the ARN of the event bus you want to use.

For more information, see:
- ??? (p. 131)
- ??? (p. 133)

API destination

To use an EventBridge API destination, select EventBridge API destination, then do one of the following:

- To use an existing API destination, select Use an existing API destination. Then select an API destination from the dropdown list.
- To create a new API destination, select Create a new API destination. Then, provide the following details for the destination:
  - Name – Enter a name for the destination. Names must be unique within your AWS account. Names can have up to 64 characters. Valid characters are A-Z, a-z, 0-9, and _ - (hyphen).
  - (Optional) Description – Enter a description for the destination. Descriptions can have up to 512 characters.
  - API destination endpoint – The URL endpoint for the target. The endpoint URL must start with https. You can include the * as a path parameter wildcard. You can set path parameters from the target's HttpParameters attribute.
  - HTTP method – Select the HTTP method used when you invoke the endpoint.
  - (Optional) Invocation rate limit per second – Enter the maximum number of invocations accepted for each second for this destination. This value must be greater than zero. By default, this value is set to 300.
  - Connection – Choose to use a new or existing connection:
    - To use an existing connection, select Use an existing connection and select the connection from the dropdown list.
    - To create a new connection for this destination select Create a new connection, then define the connection's Name, Destination type, and Authorization type. You can also add an optional Description for this connection.

For more information, see ??? (p. 117).

AWS service

To use an AWS service, select AWS service, then do the following:

1. For Select a target, select an AWS service to use as the target. Provide the information requested for the service you select.

   **Note**
   The fields displayed vary depending on the service selected. For more information about available targets, see Targets available in the EventBridge console (p. 112).
2. For many target types, EventBridge needs permissions to send events to the target. In these cases, EventBridge can create the IAM role needed for your rule to run.

   For **Execution role**, do one of the following:
   
   • To create a new execution role for this rule:
     a. Select **Create a new role for this specific resource**.
     b. Either enter a name for this execution role, or use the name generated by EventBridge.
   • To use an existing execution role for this rule:
     a. Select **Use existing role**.
     b. Enter or select the name of the execution role to use from the dropdown list.

3. (Optional) For **Additional settings**, specify any of the optional settings available for your target type:

   **Event bus**

   (Optional) For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
   
   • Choose **None** to not use a dead-letter queue.
   • Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the drop-down list.
   • Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it.

   For more information, see [Granting permissions to the dead-letter queue](p. 60).

   **API destination**

   1. (Optional) For **Configure target input**, choose how you want to customize the text sent to the target for matching events. Choose one of the following:
   
   • **Matched events** – EventBridge sends the entire original source event to the target. This is the default.
   • **Part of the matched events** – EventBridge only sends the specified portion of the original source event to the target.

      Under **Specify the part of the matched event**, specify a JSON path that defines the part of the event you want EventBridge to send to the target.
   • **Constant (JSON text)** – EventBridge sends only the specified JSON text to the target. No part of the original source event is sent.

      Under **Specify the constant in JSON**, specify the JSON text that you want EventBridge to send to the target instead of the event.
   • **Input transformer** – Configure an input transformer to customize the text you want EventBridge send to the target. For more information, see [??](p. 137).
      a. Select **Configure input transformer**.
      b. Configure the input transformer following the steps in [??](p. 141).

   2. (Optional) Under **Retry policy**, specify how EventBridge should retry sending an event to a target after an error occurs.
   
   • **Maximum age of event** – Enter the maximum amount of time (in hours, minutes, and seconds) for EventBridge to retain unprocessed events. The default is 24 hours.
Create a rule that runs on a schedule

- **Retry attempts** – Enter the maximum number of times EventBridge should retry sending an event to the target after an error occurs. The default is 185 times.

3. (Optional) For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:

- Choose **None** to not use a dead-letter queue.
- Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the drop-down list.
- Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it.

For more information, see [Granting permissions to the dead-letter queue](p. 60).

AWS service

Note that EventBridge may not display all of the following fields for a given AWS service.

1. (Optional) For Configure target input, choose how you want to customize the text sent to the target for matching events. Choose one of the following:

- **Matched events** – EventBridge sends the entire original source event to the target. This is the default.
- **Part of the matched events** – EventBridge only sends the specified portion of the original source event to the target.

Under **Specify the part of the matched event**, specify a JSON path that defines the part of the event you want EventBridge to send to the target.

- **Constant (JSON text)** – EventBridge sends only the specified JSON text to the target. No part of the original source event is sent.

Under **Specify the constant in JSON**, specify the JSON text that you want EventBridge to send to the target instead of the event.

- **Input transformer** – Configure an input transformer to customize the text you want EventBridge send to the target. For more information, see [??](p. 137).
  a. Select **Configure input transformer**.
  b. Configure the input transformer following the steps in [??](p. 141).

2. (Optional) Under **Retry policy**, specify how EventBridge should retry sending an event to a target after an error occurs.

- **Maximum age of event** – Enter the maximum amount of time (in hours, minutes, and seconds) for EventBridge to retain unprocessed events. The default is 24 hours.
- **Retry attempts** – Enter the maximum number of times EventBridge should retry sending an event to the target after an error occurs. The default is 185 times.

3. (Optional) For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:

- Choose **None** to not use a dead-letter queue.
- Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the drop-down list.
- Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it.

For more information, see [Granting permissions to the dead-letter queue](p. 60).
4. (Optional) Choose **Add another target** to add another target for this rule.
5. Choose **Next**.

**Configure tags and review rule**

Finally, enter any desired tags for the rule, then review and create the rule.

**To configure tags, and review and create the rule**

1. (Optional) Enter one or more tags for the rule. For more information, see [Amazon EventBridge tags (p. 453)](#).
2. Choose **Next**.
3. Review the details for the new rule. To make changes to any section, choose the **Edit** button next to that section.

   When satisfied with the rule details, choose **Create rule**.

**Cron expressions reference**

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

**Syntax**

\[
\text{cron(fields)}
\]

<table>
<thead>
<tr>
<th>Field</th>
<th>Values</th>
<th>Wildcards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minutes</td>
<td>0-59</td>
<td>, - */</td>
</tr>
<tr>
<td>Hours</td>
<td>0-23</td>
<td>, - */</td>
</tr>
<tr>
<td>Day-of-month</td>
<td>1-31</td>
<td>, - *? / L W</td>
</tr>
<tr>
<td>Month</td>
<td>1-12 or JAN-DEC</td>
<td>- */</td>
</tr>
<tr>
<td>Day-of-week</td>
<td>1-7 or SUN-SAT</td>
<td>- *? L #</td>
</tr>
<tr>
<td>Year</td>
<td>1970-2199</td>
<td>- */</td>
</tr>
</tbody>
</table>

**Wildcards**

- The `,` (comma) wildcard includes additional values. In the Month field, JAN,FEB,MAR includes January, February, and March.
- The `-` (dash) wildcard specifies ranges. In the Day field, 1-15 includes days 1 through 15 of the specified month.
- The `*` (asterisk) wildcard includes all values in the field. In the Hours field, `*` includes every hour. You can’t 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 `/` (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 any. In the Day-of-month field you could enter 7 and if any day of the week was acceptable, 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.

**Limitations**
• 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 * (asterisk) 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.

<table>
<thead>
<tr>
<th>Minutes</th>
<th>Hours</th>
<th>Day of month</th>
<th>Month</th>
<th>Day of week</th>
<th>Year</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>*</td>
<td>*</td>
<td>?</td>
<td>*</td>
<td>Run at 10:00 am (UTC+0) every day</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>*</td>
<td>*</td>
<td>?</td>
<td>*</td>
<td>Run at 12:15 pm (UTC+0) every day</td>
</tr>
<tr>
<td>0</td>
<td>18</td>
<td>?</td>
<td>*</td>
<td>MON-FRI</td>
<td>*</td>
<td>Run at 6:00 pm (UTC +0) every Monday through Friday</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>1</td>
<td>*</td>
<td>?</td>
<td>*</td>
<td>Run at 8:00 am (UTC+0) every 1st day of the month</td>
</tr>
<tr>
<td>0/15</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>?</td>
<td>*</td>
<td>Run every 15 minutes</td>
</tr>
<tr>
<td>0/10</td>
<td>*</td>
<td>?</td>
<td>*</td>
<td>MON-FRI</td>
<td>*</td>
<td>Run every 10 minutes Monday through Friday</td>
</tr>
</tbody>
</table>
Amazon EventBridge User Guide
Rate expressions

<table>
<thead>
<tr>
<th>Minutes</th>
<th>Hours</th>
<th>Day of month</th>
<th>Month</th>
<th>Day of week</th>
<th>Year</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/5</td>
<td>8-17</td>
<td>?</td>
<td>*</td>
<td>MON-FRI</td>
<td>*</td>
<td>Run every 5 minutes Monday through Friday between 8:00 am and 5:55 pm (UTC+0)</td>
</tr>
<tr>
<td>0/30</td>
<td>20-2</td>
<td>?</td>
<td>*</td>
<td>MON-FRI</td>
<td>*</td>
<td>Run every 30 minutes Monday through Friday between 10:00 pm on the starting day to 2:00 am on the following day (UTC) Run from 12:00 am to 2:00 am on Monday morning (UTC).</td>
</tr>
</tbody>
</table>

The following example creates a rule that runs every day at 12:00pm UTC+0.

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

The following example creates a rule that runs every day, at 2:05pm and 2:35pm UTC+0.

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

The following example creates a rule that runs at 10:15am UTC+0 on the last Friday of each month during the years 2019 to 2022.

```
aws events put-rule --schedule-expression "cron(15 10 ? 6L 2019-2022)" --name MyRule3
```

Rate expressions reference

A rate expression starts when you create the scheduled event rule, and then it runs on a defined schedule. Rate expressions have two required fields 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

Limitations

If the value is equal to 1, then the unit must be singular. If the value is greater than 1, the unit must be plural. For example, rate(1 hours) and rate(5 hour) aren't 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 next triggers it every five minutes, the third example triggers it once an hour, and the final example triggers it once per day.

```bash
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
```
Disabling or deleting an Amazon EventBridge rule

To stop a rule (p. 82) from processing events (p. 27) or running on a schedule, you can delete or disable the rule. The following steps walk you through how to delete or disable an EventBridge rule.

**To delete or disable a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
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, enter 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, enter the rule name and choose **Force delete**.
   b. To temporarily disable a rule, select the button next to the rule and choose **Disable, Disable**.

You can't disable a managed rule.

**Best practices when defining Amazon EventBridge rules**

Below are some best practices to consider when you create rules for your event buses.

**Set a single target for each rule**

While you can specify up to five targets for a given rule, managing rules is easier when you specify a single target for each rule. If more than one target needs to receive the same set of events, we recommend duplicating the rule to deliver the same events to different targets. This encapsulation simplifies maintaining rules: if the needs of the event targets diverge over time, you can update each rule and its event pattern independently of the others.

**Set rule permissions**

You can enable event-consuming application components or services to be in control of managing their own rules. A common architectural approach adopted by customers is to isolate these application components or services by using separate AWS accounts. To enable the flow of events from one account to another, you must create a rule on one event bus that routes events to an event bus in another account. You can enable event-consuming teams or services to be in control of managing their own rules. You do this by specifying the appropriate permissions to their accounts through resource policies. This works across accounts and Regions.

For more information, see ??? (p. 18).

For example of resource policies, see Multi-account design patterns with Amazon EventBridge on GitHub.

**Monitor rule performance**

Monitor your rules to make sure they are performing as you expect:
Monitor rule performance

- Monitor the `TriggeredRules` metric for missing data-points or anomalies can assist you in detecting discrepancies for a publisher that made a breaking change. For more information, see ?? (p. 434).
- Alarm on anomalies or maximum expected count can also help detecting when a rule is matching against new events. This can happen when event publishers, including AWS services and SaaS partners, introduce new events when enabling new use-cases and features. When these new events are unexpected and lead to a higher volume than the processing rate of the downstream target, they can lead to an event backlog.

  Such processing of unexpected events can also lead to unwanted billing charges.

  It can also trigger throttling of rules when the account goes over its aggregate target invocations per second service quota. EventBridge will still attempt to deliver events matched by throttled rules and retry up to 24 hours or as described within the target's custom retry policy. You can detect and alarm throttled rules using the `ThrottledRules` metric.

- For low-latency use cases, you can also monitor latency using `IngestionToInvocationStartLatency`, which provides an indication of health of your event bus. Any extended periods of high latency over 30 seconds may indicate a service disruption or rule throttling.
Using Amazon EventBridge and AWS Serverless Application Model templates

You can build and test rules (p. 82) manually in the EventBridge console, which can help in the development process as you refine event patterns (p. 62). However, once you are ready to deploy your application, it’s easier to use a framework like AWS SAM to launch all your serverless resources consistently.

We’ll use this example application to look into the ways you can use AWS SAM templates to build EventBridge resources. The template.yaml file in this example is a AWS SAM template that defines four AWS Lambda functions and shows two different ways to integrate the Lambda functions with EventBridge.

For a walkthrough of this example application, see ?? (p. 247).

There are two approaches to using EventBridge and AWS SAM templates. For simple integrations where one Lambda function is invoked by one rule, the the Combined template approach is recommended. If you have complex routing logic, or you are connecting to resources outside of your AWS SAM template, the Separated template approach is the better choice.

Approaches:

- Combined template (p. 108)
- Separated template (p. 109)

Combined template

The first approach uses the Events property to configure the EventBridge rule. The following example code defines an event (p. 27) that invokes your Lambda function.

**Note**

This example automatically creates the rule on the default event bus (p. 9), which exists in every AWS account. To associate the rule with a custom event bus, you can add the EventBusName to the template.

```yaml
atmConsumerCase3Fn:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: atmConsumer/
    Handler: handler.case3Handler
    Runtime: nodejs12.x
    Events:
      Trigger:
        Type: CloudWatchEvent
        Properties:
          Pattern:
            source:
              - custom.myATMapp
            detail-type:
              - transaction
            detail:
              result:
                - "anything-but": "approved"
```

This YAML code is equivalent to an event pattern in the EventBridge console. In YAML, you only need to define the event pattern, and AWS SAM automatically creates an IAM role with the required permissions.
Separated template

In the second approach to defining an EventBridge configuration in AWS SAM, the resources are separated more clearly in the template.

1. First, you define the Lambda function:

```yaml
atmConsumerCase1Fn:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: atmConsumer/
    Handler: handler.case1Handler
    Runtime: nodejs12.x
```

2. Next, define the rule using an `AWS::Events::Rule` resource. The properties define the event pattern and can also specify targets (p. 112). You can explicitly define multiple targets.

```yaml
EventRuleCase1:
  Type: AWS::Events::Rule
  Properties:
    Description: "Approved transactions"
    EventPattern:
      source:
        - "custom.myATMapp"
      detail-type:
        - transaction
      detail:
        result:
          - "approved"
    State: "ENABLED"
  Targets:
    - Arn:
      Fn::GetAtt:
        - "atmConsumerCase1Fn"
        - "Arn"
      Id: "atmConsumerTarget1"
```

3. Finally, define an `AWS::Lambda::Permission` resource that grants permission to EventBridge to invoke the target.

```yaml
PermissionForEventsToInvokeLambda:
  Type: AWS::Lambda::Permission
  Properties:
    FunctionName:
      Ref: "atmConsumerCase1Fn"
    Action: "lambda:InvokeFunction"
    Principal: "events.amazonaws.com"
    SourceArn:
      Fn::GetAtt:
        - "EventRuleCase1"
        - "Arn"
```

109
Generate an AWS CloudFormation template from Amazon EventBridge rules

AWS CloudFormation enables you to configure and manage your AWS resources across accounts and regions in a centralized and repeatable manner by treating infrastructure as code. AWS CloudFormation does this by letting you create templates, which define the resources you want to provision and manage.

EventBridge enables you to generate templates from the existing rules in your account, as an aid to help you jumpstart developing AWS CloudFormation templates. You can select a single rule, or multiple rules to include in the template. You can then use these templates as the basis for creating stacks of resources under AWS CloudFormation management.

For more information on AWS CloudFormation see The AWS CloudFormation User Guide.

Note
EventBridge does not include managed rules in the generated template.

You can also generate a template from an existing event bus, including the rules that event bus contains.

To generate an AWS CloudFormation template from a rule

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Under Select event bus, choose the event bus that contains the rules you want to include in the template.
4. Under Rules, choose the rules you want to include in the generated AWS CloudFormation template.
5. Choose CloudFormation Template, and then choose which format you want EventBridge to generate the template in: JSON or YAML.

EventBridge displays the template, generated in the selected format.
6. EventBridge gives you the option of downloading the template file, or copying the template to the clipboard.
   • To download the template file, choose Download.
   • To copy the template to the clipboard, choose Copy.
7. To exit the template, choose Cancel.

Once you’ve customized your AWS CloudFormation template as necessary for your use case, you can use it to create stacks in AWS CloudFormation.

Considerations when using AWS CloudFormation templates generated from Amazon EventBridge

Consider the following factors when using a AWS CloudFormation template you generated from EventBridge:

• EventBridge does not include any passwords in the generate template.

You can edit the template to include template parameters that enable users to specify passwords or other sensitive information when using the template to create or update a AWS CloudFormation stack.

In addition, users can use Secrets Manager to create a secret in the desired region and then edit the generated template to employ dynamic parameters.
- Targets in the generated template remain exactly as they were specified in the original event bus. This can lead to cross-region issues if you do not appropriately edit the template before using it to create stacks in other regions.

  Additionally, the generated template will not create the downstream targets automatically.
Amazon EventBridge targets

A target is a resource or endpoint that EventBridge sends an event (p. 27) to when the event matches the event pattern defined for a rule (p. 82). The rule processes the event (p. 27) data and sends the pertinent information to the target. To deliver event data to a target, EventBridge needs permission to access the target resource. You can define up to five targets for each rule.

When you add targets to a rule and that rule runs soon after, any new or updated targets might not be immediately invoked. Allow a short period of time for changes to take effect.

The following video covers the basics of targets: What is a target

Targets available in the EventBridge console

You can configure the following targets for events in the EventBridge console:

- API destination (p. 117)
- API Gateway (p. 130)
- Batch job queue (p. 114)
- CloudWatch log group (p. 115)
- CodeBuild project (p. 115)
- CodePipeline
- Amazon EBS CreateSnapshot API call
- EC2 Image Builder
- EC2 RebootInstances API call
EC2 StopInstances API call
EC2 TerminateInstances API call
ECS task (p. 115)
Event bus in a different account or Region (p. 131)
Event bus in the same account and Region (p. 135)
Firehose delivery stream
Glue workflow
Incident Manager response plan
Inspector assessment template
Kinesis stream
Lambda function (ASYNC)
Amazon Redshift cluster data API queries
Amazon Redshift Serverless workgroup data API queries
SageMaker Pipeline
Amazon SNS topic

EventBridge does not support Amazon SNS FIFO (first in, first out) topics.

Amazon SQS queue
Step Functions state machine (ASYNC)
Systems Manager Automation
Systems Manager OpsItem
Systems Manager Run Command

Target parameters

Some targets don't send the information in the event payload to the target, instead, they treat the event as a trigger for invoking a specific API. EventBridge uses the Target parameters to determine what happens with that target. These include the following:

- API destinations (The data sent to an API destination must match the structure of the API. You must use the InputTransformer object to make sure the data is structured correctly. If you want to include the original event payload, reference it in the InputTransformer.)
- API Gateway (The data sent to API Gateway must match the structure of the API. You must use the InputTransformer object to make sure the data is structured correctly. If you want to include the original event payload, reference it in the InputTransformer.)
- Amazon EC2 Image Builder
- RedshiftDataParameters (Amazon Redshift Data API clusters)
- SageMakerPipelineParameters (Amazon SageMaker Runtime Model Building Pipelines)

Note
EventBridge does not support all JSON Path syntax and evaluate it at runtime. Supported syntax includes:

- dot notation (for example, $.detail)
- dashes
- underscores
- alphanumeric characters
- array indices
Dynamic path parameters

Some target parameters support optional dynamic JSON path syntax. This syntax allows you to specify JSON paths instead of static values (for example $.detail.state). The entire value has to be a JSON path, not just part of it. For example, RedshiftParameters.Sql can be $.detail.state but it can't be "SELECT * FROM $.detail.state". These paths are replaced dynamically at runtime with data from the event payload itself at the specified path. Dynamic path parameters can't reference new or transformed values resulting from input transformation. The supported syntax for dynamic parameter JSON paths is the same as when transforming input. For more information, see ???(p. 137)

Dynamic syntax can be used on all the string, non-enum fields of these parameters:

- EcsParameters
- HttpParameters (except HeaderParameters keys)
- RedshiftDataParameters
- SageMakerPipelineParameters

Permissions

To make API calls on the resources that you own, EventBridge needs appropriate permission. For AWS Lambda and Amazon SNS resources, EventBridge uses resource-based policies (p. 405). For EC2 instances, Kinesis data streams, and Step Functions state machines, EventBridge uses IAM roles that you specify in the RoleARN parameter in PutTargets. You can invoke an API Gateway endpoint with configured IAM authorization, but the role is optional if you haven't configured authorization. For more information, see Amazon EventBridge and AWS Identity and Access Management (p. 388).

If another account is in the same Region and has granted you permission, then you can send events to that account. For more information, see Sending and receiving Amazon EventBridge events between AWS accounts (p. 131).

If your target is encrypted, you must include the following section in your KMS key policy.

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

EventBridge target specifics

AWS Batch job queues

Certain parameters to AWS Batch submitJob can be configured via BatchParameters.
Others can be specified in the event payload. If the event payload (passed through or via `InputTransformers`) contains the following keys, they are mapped to `submitJob` request parameters:

- `ContainerOverrides`: `containerOverrides`
  
  **Note**
  This includes only command, environment, memory, and vcpus

- `DependsOn`: `dependsOn`
  
  **Note**
  This includes only jobid

- `Parameters`: `parameters`

### CloudWatch Logs group

If you don’t use an `InputTransformer` with a CloudWatch Logs target, the event payload is used as the log message, and the source of the event as the timestamp. If you do use an InputTransformer, the template must be:

```json
{"timestamp":<timestamp>, "message":<message>}
```

EventBridge batches the entries sent to a log stream; therefore, EventBridge may deliver a single or multiple events to a log stream, depending on traffic.

### CodeBuild project

If you use `InputTransformers` to shape the input event to a Target to match the CodeBuild `StartBuildRequest` structure, the parameters will be mapped 1-to-1 and passed through to `codeBuild.StartBuild`.

### Amazon ECS task

If you use `InputTransformers` to shape the input event to a Target to match the Amazon ECS RunTask `TaskOverride` structure, the parameters will be mapped 1-to-1 and passed through to `ecs.RunTask`.

### Incident Manager Response Plan

If the matched event came from CloudWatch Alarms, the alarm state change details are populated into the trigger details of the `StartIncidentRequest` call to Incident Manager.
Configure targets

Learn how to configure settings for EventBridge targets.

Targets:
- API destinations (p. 117)
- Amazon EventBridge targets for Amazon API Gateway (p. 130)
- Sending and receiving Amazon EventBridge events between AWS accounts (p. 131)
- Sending and receiving Amazon EventBridge events between AWS Regions (p. 133)
- Sending and receiving Amazon EventBridge events between event buses in the same account and Region (p. 135)
API destinations

Amazon EventBridge API destinations are HTTP endpoints that you can invoke as the target (p. 112) of a rule (p. 82), similar to how you invoke an AWS service or resource as a target. Using API destinations, you can route events (p. 27) between AWS services, integrated software as a service (SaaS) applications, and your applications outside of AWS by using API calls. When you specify an API destination as the target of a rule, EventBridge invokes the HTTP endpoint for any event that matches the event pattern (p. 62) specified in the rule and then delivers the event information with the request. With EventBridge, you can use any HTTP method except CONNECT and TRACE for the request. The most common HTTP methods to use are PUT and POST. You can also use input transformers to customize the event to the parameters of a specific HTTP endpoint parameters. For more information, see Amazon EventBridge input transformation (p. 137).

API destinations do not support private destinations, such as interface VPC endpoints. For more information, see ?? (p. 238).

Important
EventBridge requests to an API destination endpoint must have a maximum client execution timeout of 5 seconds. If the target endpoint takes longer than 5 seconds to respond, EventBridge times out the request. EventBridge retries timed out requests up to the maximums that are configured on your retry policy. By default the maximums are 24 hours and 185 times. After the maximum number of retries, events are sent to your dead-letter queue (p. 58) if you have one. Otherwise, the event is dropped.

The following video demonstrates the use of API destination: Using API destinations

In this topic:
- Connections for API destinations (p. 118)
- Create an API destination (p. 119)
- Creating rules that send events to an API destination (p. 120)
Connections for API destinations

When you create an API destination, you specify a connection to use for it. A connection specifies the authorization type and parameters to use to authorize with the API destination endpoint. You can choose an existing connection from your account or create a connection when you create an API destination. EventBridge supports Basic, OAuth, and API Key authorization.

For Basic and API Key authorization, EventBridge populates the required authorization headers for you. For OAuth authorization, EventBridge also exchanges your client ID and secret for an access token and then manages it securely. When you create a connection, you can also include the header, body, and query parameters that are required for authorization with an endpoint. You can use the same connection for more than one API destination if the authorization for the endpoint is the same.

OAuth tokens are refreshed when a 401 or 407 response is returned.

When you create a connection and add authorization parameters, EventBridge creates a secret in AWS Secrets Manager. The cost of storing the Secrets Manager secret is included with the charge for using an API destination. To learn more about best practices for using secrets with API destinations, see AWS::Events::ApiDestination in the CloudFormation User Guide.

Note
To successfully create or update a connection, you must use an account that has permission to use Secrets Manager. The required permission is included in the AmazonEventBridgeFullAccess policy. The same permission is granted to the service-linked role that's created in your account for the connection.

To create a connection

1. Log in to AWS using an account that has permissions to manage EventBridge and open the EventBridge console.
2. In the left navigation pane, choose API destinations.
3. Scroll down to the API destinations table, and then choose the Connections tab.
4. Choose Create connection.
5. On the Create connection page, enter a Connection name for the connection.
6. Enter a Description for the connection.
7. For Authorization type, select the type of authorization to use to authorize connections to the HTTP endpoint specified for the API destination that uses this connection. Do one of the following:
   • Choose Basic (Username/Password), and then enter the Username and Password to use to authorize with the HTTP endpoint.
   • Choose OAuth Client Credentials, and then enter the Authorization endpoint, HTTP method, Client ID, and Client secret to use to authorize with the endpoint.

Under OAuth Http Parameters, add any additional parameters to include for authorization with the authorization endpoint. Select a Parameter from the drop-down list, then enter a Key and Value. To include an additional parameter, choose Add parameter.

Under Invocation Http Parameters, add any additional parameters to include in the authorization request. To add a parameter, select a Parameter from the drop-down list, then enter a Key and Value. To include an additional parameter, choose Add parameter.
Choose **API key**, and then enter the **API key name** and associated **Value** to use for API Key authorization.

Under **Invocation Http Parameters**, add any additional parameters to include in the authorization request. To add a parameter, select a **Parameter** from the drop-down list, then enter a **Key** and **Value**. To include an additional parameter, choose **Add parameter**.

8. Choose **Create**.

**To edit a connection**

1. Open the **API destinations** page, and then choose **Connections**.
2. In the **Connections** table, choose the connection to edit.
3. On the **Connection details** page, choose **Edit**.
4. Update the values for the connection, and then choose **Update**.

**De-authorizing connections**

When you de-authorize a connection, it removes all authorization parameters. Removing authorization parameters removes the secret from the connection, so you can reuse it without having to create a new connection.

**Note**

You must update any API destinations that use the de-authorized connection to use a different connection to successfully send requests to the API destination endpoint.

**To de-authorize a connection**

1. In the **Connections** table, choose the connection.
2. On the **Connection details** page, choose **De-authorize**.
3. In the **Deauthorize connection?** dialog box, enter the name of the connection, and then choose **De-authorize**.

The status of the connection changes to **De-authorizing** until the process is complete. Then the status changes to **De-authorized**. Now you can edit the connection to add new authorization parameters.

**Create an API destination**

Each API destination requires a connection. A **connection** specifies the authorization type and credentials to use to authorize with the API destination endpoint. You can choose an existing connection, or create a connection at the same time that you create the API destination.

**To create an API destination**

1. Log in to AWS using an account that has permissions to manage EventBridge and open the **EventBridge console**.
2. In the left navigation pane, choose **API destinations**.
3. Scroll down to the **API destinations** table, and then choose **Create API destination**.
4. On the **Create API destination** page, enter a **Name** for the API destination. You can use up to 64 uppercase or lowercase letters, numbers, dot (.), dash (-), or underscore (_) characters.
   The name must be unique to your account in the current Region.
5. Enter a **Description** for the API destination.
6. Enter an **API destination endpoint** for the API destination. The **API destination endpoint** is an HTTP invocation endpoint target for events. The authorization information you include in the connection used for this API destination is used to authorize against this endpoint. The URL must use HTTPS.

7. Enter the **HTTP method** to use to connect to the **API destination endpoint**.

8. (Optional) For **Invocation rate limit per second** field, enter the maximum number of invocations per second to send to the API destination endpoint.

   The rate limit you set may affect how EventBridge delivers events. For more information, see [How invocation rate affects event delivery](p. 122).

9. For **Connection**, do one of the following:
   - Choose **Use an existing connection**, and then select the connection to use for this API destination.
   - Choose **Create a new connection**, and then enter the details for the connection to create. For more information, see [Connections](p. 118).

10. Choose **Create**.

### Creating rules that send events to an API destination

After you create an API destination, you can select it as the target of a **rule** (p. 82). To use an API destination as a target, you must provide an IAM role with the correct permissions. For more information, see [??](p. 398)

Selecting an API destination as a target is part of creating the rule.

#### To create a rule that sends events to an API destination using the console

1. Follow the steps in the [??](p. 84) procedure.
2. In the [??](p. 87) step, when prompted to choose a target type:
   a. Select **EventBridge API destination**.
   b. Select an existing API destination, or choose to create a new API destination.

   If you choose to create a new API destination, provide the necessary information as prompted.

3. Complete creating the rule following the procedure steps.

### Service-linked role for API destinations

When you create a connection for an API destination, a service-linked role named `AWSServiceRoleForAmazonEventBridgeApiDestinations` is added to your account. EventBridge uses the service-linked role to create and store a secret in Secrets Manager. To grant the necessary permissions to the service-linked role, EventBridge attaches the `AmazonEventBridgeApiDestinationsServiceRolePolicy` policy to the role. The policy limits the permissions granted to only those necessary for the role to interact with the secret for the connection. No other permissions are included, and the role can interact only with the connections in your account to manage the secret.

The following policy is the `AmazonEventBridgeApiDestinationsServiceRolePolicy`.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
```
For more information about service-linked roles, see Using service-linked roles in the IAM documentation.

The AmazonEventBridgeApiDestinationsServiceRolePolicy service-linked role is supported in the following AWS regions:

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

**Headers included in requests to API destinations**

In addition to the authorization headers defined for the connection used for an API destination, EventBridge includes the following headers in each request.

<table>
<thead>
<tr>
<th>Header key</th>
<th>Header value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Agent</td>
<td>Amazon/EventBridge/ApiDestinations</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json; charset=utf-8</td>
</tr>
<tr>
<td>Header key</td>
<td>Header value</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Range</td>
<td>bytes=0-1048575</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>gzip, deflate</td>
</tr>
<tr>
<td>Connection</td>
<td>close</td>
</tr>
<tr>
<td>Content-Length</td>
<td>An entity header that indicates the size of the entity-body, in bytes, sent to the recipient.</td>
</tr>
<tr>
<td>Host</td>
<td>A request header that specifies the host and port number of the server where the request is being sent.</td>
</tr>
</tbody>
</table>

**API destination error codes**

When EventBridge tries to deliver an event to an API destination and an error occurs, EventBridge does the following:

- Events associated with error codes 409, 429, and 5xx are retried.
- Events associated with error codes 1xx, 2xx, 3xx, and 4xx (excluding 429) aren't retried.

EventBridge API destinations read the standard HTTP response header `Retry-After` to find out how long to wait before making a follow-up request. EventBridge chooses the more conservative value between the defined retry policy and the `Retry-After` header. If `Retry-After` value is negative, EventBridge stops retrying delivery for that event.

**How invocation rate affects event delivery**

If you set the invocation rate per second to a value much lower than the number of invocations generated, events may not be delivered within the 24 hour retry time for events. For example, if you set the invocation rate to 10 invocations per second, but thousands of events per second are generated, you will quickly have a backlog of events to deliver that exceeds 24 hours. To ensure that no events are lost, set up a dead-letter queue to send events with failed invocations to so you can process the events at a later time. For more information, see [Event retry policy and using dead-letter queues](p. 58).

**API destination partners**

Use the information provided by the following AWS Partners to configure an API destination and connection for their service or application.

**Coralogix**

**API destination invocation endpoint URL**

For a full list of endpoints, see [Coralogix API Reference](#).

**Supported authorization types**

- API Key

**Additional authorization parameters required**

- Header "x-amz-event-bridge-access-key", the value is the Coralogix API Key

**Coralogix documentation**

[Amazon EventBridge authentication](#)
Commonly used API operations

US: https://ingress.coralogix.us/aws/event-bridge
Singapore: https://ingress.coralogixsg.com/aws/event-bridge
Ireland: https://ingress.coralogix.com/aws/event-bridge
Stockholm: https://ingress.eu2.coralogix.com/aws/event-bridge
India: https://ingress.coralogix.in/aws/event-bridge

Additional information

The events are stored as log entries with `applicationName=[AWS Account]` and `subsystemName=[event.source]`.

Datadog

API destination invocation endpoint URL

For a full list of endpoints, see Datadog API Reference.

Supported authorization types

API Key

Additional authorization parameters required

None

Datadog documentation

Authentication

Commonly used API operations

POST https://api.datadoghq.com/api/v1/events
POST https://http-intake.logs.datadoghq.com/v1/input

Additional information

Endpoint URLs differ depending on the location of your Datadog organization. For the correct URL for your organization, see documentation.

Freshworks

API destination invocation endpoint URL

For a list of endpoints, see https://developers.freshworks.com/documentation/

Supported authorization types

Basic, API Key

Additional authorization parameters required

Not applicable

Freshworks documentation

Authentication

Commonly used API operations

https://developers.freshdesk.com/api/#create_ticket
https://developers.freshdesk.com/api/#update_ticket
https://developer.freshsales.io/api/#create_lead
https://developer.freshsales.io/api/#update_lead

**Additional information**

None

**MongoDB**

**API destination invocation endpoint URL**

https://data.mongodb-api.com/app/App ID/endpoint/

**Supported authorization types**

- API Key
- Email/Password
- Custom JWT Authentication

**Additional authorization parameters required**

None

**MongoDB documentation**

- [Atlas Data API](#)
- [Endpoints](#)
- [Custom HTTPS Endpoints](#)
- [Authentication](#)

**Commonly used API operations**

None

**Additional information**

None

**New Relic**

**API destination invocation endpoint URL**

For more information, see [Our EU and US region data centers](#).

**Events**

- **US**: https://insights-collector.newrelic.com/v1/accounts/YOUR_NEW_RELIC_ACCOUNT_ID/events
- **EU**: https://insights-collector.eu01.nr-data.net/v1/accounts/YOUR_NEW_RELIC_ACCOUNT_ID/events

**Metrics**

- **US**: https://metric-api.newrelic.com/metric/v1
- **EU**: https://metric-api.eu.newrelic.com/metric/v1

**Logs**

- **US**: https://log-api.newrelic.com/log/v1
API destinations

**EU**– https://log-api.eu.newrelic.com/log/v1

**Traces**

**US**– https://trace-api.newrelic.com/trace/v1

**EU**– https://trace-api.eu.newrelic.com/trace/v1

**Supported authorization types**

API Key

**New Relic documentation**

[Metric API](#)

[Event API](#)

[Log API](#)

[Trace API](#)

**Commonly used API operations**

[Metric API](#)

[Event API](#)

[Log API](#)

[Trace API](#)

**Additional information**

[Metric API limits](#)

[Event API limits](#)

[Log API limits](#)

[Trace API limits](#)

**Operata**

**API destination invocation endpoint URL:**

https://api.operata.io/v2/aws/events/contact-record

**Supported authorization types:**

Basic

**Additional authorization parameters required:**

None

**Operata documentation:**

[How do I create, view, change and revoke API Tokens?](#)

[Operata AWS Integration using Amazon EventBridge Scheduler Pipes](#)

**Commonly used API operations:**

POST https://api.operata.io/v2/aws/events/contact-record

**Additional information:**

The *username* is the Operata Group ID and the password is your API token.
Salesforce

**API destination invocation endpoint URL**

- **Subject**– https://myDomainName.my.salesforce.com/services/data/versionNumber/subjects/SubjectEndpoint/*
- **Custom platform events**– https://myDomainName.my.salesforce.com/services/data/versionNumber/sobjects/customPlatformEndpoint/*

For a full list of endpoints, see Salesforce API Reference

**Supported authorization types**

- OAuth client credentials
  
  OAUTH tokens are refreshed when a 401 or 407 response is returned.

**Additional authorization parameters required**

- Salesforce Connected App Client Id and Client Secret.

One of the following authorization endpoints:

- **Production**– https://MyDomainName.my.salesforce.com/services/oauth2/token
- **Sandbox without enhanced domains**– https://MyDomainName--SandboxName.my.salesforce.com/services/oauth2/token
- **Sandbox with enhanced domains**– https://MyDomainName--SandboxName.sandbox.my.salesforce.com/services/oauth2/token

The following key/value pairs:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_type</td>
<td>password</td>
</tr>
<tr>
<td>username</td>
<td>Your Salesforce username</td>
</tr>
<tr>
<td>password</td>
<td>Your Salesforce password</td>
</tr>
</tbody>
</table>

**Salesforce documentation**

- REST API Developer Guide

**Commonly used API operations**

- Working with Object Metadata
- Working with Records

**Additional information**

- For a tutorial explaining how to use the EventBridge console to create a connection to Salesforce, an API Destination, and a rule to route information to Salesforce, see ??? (p. 286).

Slack

**API destination invocation endpoint URL**

- For a list of endpoints and other resources, see Using the Slack Web API

**Supported authorization types**

- OAuth 2.0
OAUTH tokens are refreshed when a 401 or 407 response is returned.

When you create a Slack application and install it to your workspace, an OAuth bearer token will be created on your behalf to be used for authenticating calls by your API destination connection.

**Additional authorization parameters required**

Not applicable

**Slack documentation**

- [Basic app setup](#)
- [Installing with OAuth](#)
- [Retrieving messages](#)
- [Sending messages](#)
- [Sending messages using Incoming Webhooks](#)

**Commonly used API operations**

https://slack.com/api/chat.postMessage

**Additional information**

When configuring your EventBridge rule there are two configurations to highlight:

- Include a header parameter that defines the content type as “application/json;charset=utf-8”.
- Use an input transformer to map the input event to the expected output for the Slack API, namely ensure that the payload sent to the Slack API has “channel” and “text” key/value pairs.

**Shopify**

**API destination invocation endpoint URL**

For a list of endpoints and other resources and methods, see [Endpoints and requests](#)

**Supported authorization types**

OAuth, API Key

**Note**

OAUTH tokens are refreshed when a 401 or 407 response is returned.

**Additional authorization parameters required**

Not applicable

**Shopify documentation**

- [Authentication and authorization overview](#)

**Commonly used API operations**

- POST - /admin/api/2022-01/products.json
- GET - admin/api/2022-01/products/{product_id}.json
- PUT - admin/api/2022-01/products/{product_id}.json
- DELETE - admin/api/2022-01/products/{product_id}.json

**Additional information**

- [Create an app](#)
- [Amazon EventBridge webhook delivery](#)
Access tokens for custom apps in the Shopify admin

Shopify Admin API

Splunk

API destination invocation endpoint URL

https://SPLUNK_HEC_ENDPOINT:optional_port/services/collector/raw

Supported authorization types

Basic, API Key

Additional authorization parameters required

None

Splunk documentation

For both authorization types, you need an HEC token ID. For more information, see Set up and use HTTP Event Collector in Splunk Web.

Commonly used API operations

POST https://SPLUNK_HEC_ENDPOINT:optional_port/services/collector/raw

Additional information

API Key – When configuring the endpoint for EventBridge, the API key name is “Authorization” and value is the Splunk HEC token ID.

Basic (Username/Password) – When configuring the endpoint for EventBridge, the username is “Splunk” and the password is the Splunk HEC token ID.

Sumo Logic

API destination invocation endpoint URL

HTTP Log and Metric Source endpoint URLs will be different for every user. For more information, see HTTP Logs and Metrics Source.

Supported authorization types

Sumo Logic doesn’t require authentication on their HTTP Sources because there's a unique key baked into the URL. For this reason, you should make sure to treat that URL as a secret.

When you configure the EventBridge API destination, an authorization type is required. To meet this requirement, select API Key and give it a key name of “dummy-key” and a key value of “dummy-value”.

Additional authorization parameters required

Not applicable

Sumo Logic documentation

Sumo Logic has already built hosted sources to collect logs and metrics from many AWS services and you can use the information on their website to work with those sources. For more information, see Amazon Web Services.

If you’re generating custom events from an application and want to send them to Sumo Logic as either logs or metrics, then use EventBridge API Destinations and Sumo Logic HTTP Log and Metric Source endpoints.
Commonly used API operations

POST https://endpoint4.collection.us2.sumologic.com/receiver/v1/
http/UNIQUE_ID_PER_COLLECTOR

Additional information

None

**TriggerMesh**

API destination invocation endpoint URL

Use the information in the Event Source for HTTP topic to formulate the endpoint URL. An endpoint URL includes the event source name and user namespace in the following format:

https://source-name.user-namespace.cloud.triggermesh.io

Include the Basic authorization parameters in the request to the endpoint.

Supported authorization types

Basic

Additional authorization parameters required

None

TriggerMesh documentation

Event Source for HTTP

Commonly used API operations

Not applicable

Additional information

None

**Zendesk**

API destination invocation endpoint URL

https://developer.zendesk.com/rest_api/docs/support/tickets

Supported authorization types

Basic, API Key

Additional authorization parameters required

None

Zendesk documentation

Security and Authentication

Commonly used API operations

POST https://your_Zendesk_subdomain/api/v2/tickets

Additional information

API requests EventBridge makes count against your Zendesk API limits. For information about Zendesk limits for your plan, see Usage limits.
To better safeguard your account and data, we recommend using an API key rather than basic sign-in credentials authentication.

Amazon EventBridge targets for Amazon API Gateway

You can use Amazon API Gateway to create, publish, maintain, and monitor APIs. Amazon EventBridge supports sending events to an API Gateway endpoint. When you specify an API Gateway endpoint as a target (p. 112), each event (p. 27) sent to the target maps to a request sent to the endpoint.

**Important**
EventBridge supports using API Gateway *Edge-optimized* and *Regional* endpoints as targets. *Private* endpoints are not currently supported. To learn more about endpoints, see [https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-endpoint-types.html](https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-endpoint-types.html).

You can use an API Gateway target for the following use cases:

- To invoke a customer-specified API hosted in API Gateway based on AWS or third-party events.
- To invoke an endpoint periodically on a schedule.

The EventBridge JSON event information is sent as the body of the HTTP request to your endpoint. You can specify the other request attributes in the target's `HttpParameters` field as follows:

- `PathParameterValues` lists the values that correspond sequentially to any path variables in your endpoint ARN, for example "arn:aws:execute-api:us-east-1:112233445566:myapi/dev/POST/pets/*/*".
- `QueryStringParameters` represents the query string parameters that EventBridge appends to the invoked endpoint.
- `HeaderParameters` defines HTTP headers to add to the request.

**Note**
For security considerations, the following HTTP header keys aren't permitted:

- Anything prefixed with `X-Amz` or `X-Amzn`
- `Authorization`
- `Connection`
- `Content-Encoding`
- `Content-Length`
- `Host`
- `Max-Forwards`
- `TE`
- `Transfer-Encoding`
- `Trailer`
- `Upgrade`
- `Via`
- `WWW-Authenticate`
- `X-Forwarded-For"
Dynamic Parameters

When invoking an API Gateway target, you can dynamically add data to events that are sent to the target. For more information, see the section called “Target parameters” (p. 113).

Invocation Retries

As with all targets, EventBridge retries some failed invocations. For API Gateway, EventBridge retries responses sent with a 5xx or 429 HTTP status code for up to 24 hours with exponential back off and jitter. After that, EventBridge publishes a FailedInvocations metric in Amazon CloudWatch. EventBridge doesn't retry other 4xx HTTP errors.

Timeout

EventBridge rule API Gateway requests must have a maximum client execution timeout of 5 seconds. If API Gateway takes longer than 5 seconds to respond, EventBridge times out the request and then retries.

EventBridge Pipes API Gateway requests have a maximum timeout of 29 seconds, the API Gateway maximum.

Sending and receiving Amazon EventBridge events between AWS accounts

You can configure EventBridge to send and receive events between event buses in AWS accounts. When you configure EventBridge to send or receive events between accounts, you can specify which AWS accounts can send events to or receive events from the event bus in your account. You can also allow or deny events from specific rules associated with the event bus, or events from specific sources. For more information, see Simplifying cross-account access with Amazon EventBridge resource policies.

Note
If you use AWS Organizations, you can specify an organization and grant access to all accounts in that organization. In addition, the sending event bus must have IAM roles attached to them when sending events to another account. For more information, see What is AWS Organizations in the AWS Organizations User Guide.

Note
If you're using an Incident Manager response plan as a target, all the response plans that are shared with your account are available by default.

You can send and receive events between event buses in AWS accounts within the same Region in all Regions and between accounts in different Regions as long as the destination Region is a supported cross-Region destination Region.

The steps to configure EventBridge to send events to or receive events from an event bus in a different account include the following:

- On the receiver account, edit the permissions on an 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 event bus as the target.

If the sender account inherits permissions to send events from an AWS Organization, 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 event bus in the receiver account, the role is created 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.
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 EventBridge 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.

The following video covers routing events between accounts: Routing events to buses in other AWS accounts

Grant permissions to allow events from other AWS accounts

To receive events from other accounts or organizations, you must first edit the permissions on the event bus where you intend to receive events. The default event bus accepts events from AWS services, other authorized AWS accounts, and PutEvents calls. The permissions for an event bus are granted or denied using a resource-based policy attached to the event bus. In the policy, you can grant permissions to other AWS accounts using the account ID, or to an AWS organization using the organization ID. To learn more about event bus permissions, including example policies, see Permissions for Amazon EventBridge event buses (p. 18).

Note
EventBridge now requires all new cross account event bus targets to add IAM roles. This only applies to event bus targets created after March 2, 2023. Applications created without an IAM role before that date are not affected. However, we recommend adding IAM roles to grant users access to resources in another account, as this ensures organization boundaries using Service Control Policies (SCPs) are applied to determine who can send and receive events from accounts in your organization.

Important
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 Amazon EventBridge events (p. 27).

Rules for events between AWS accounts

If your account is set up to receive events from event buses in other AWS accounts, you can write rules that match those events. Set the event pattern (p. 62) of the rule to match the events you are receiving from event buses in 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 event buses in other accounts trigger based on those events. If you are receiving events from event buses in 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 event buses in all AWS accounts, we strongly recommend that you add account to every EventBridge 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 event buses in 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.

Creating rules that send events between AWS accounts

Specifying an event bus in another account as a target is part of creating the rule.

To create a rule that sends events to a different AWS account using the console

1. Follow the steps in the ??? (p. 84) procedure.
2. In the ??? (p. 87) step, when prompted to choose a target type:
   a. Select EventBridge event bus.
   b. Select Event bus in a different account or Region.
   c. For Event bus as target, enter the ARN of the event bus you want to use.
3. Complete creating the rule following the procedure steps.

Sending and receiving Amazon EventBridge events between AWS Regions

You can configure EventBridge to send and receive events (p. 27) between AWS Regions. You can also allow or deny events from specific Regions, specific rules (p. 82) associated with the event bus, or events from specific sources. For more information, see Introducing cross-Region event routing with Amazon EventBridge

The following Regions are supported destination Regions:

- Africa (Cape Town) Region
- Asia Pacific (Hong Kong) Region
- Asia Pacific (Tokyo) Region
• Asia Pacific (Seoul) Region
• Asia Pacific (Osaka) Region
• Asia Pacific (Mumbai) Region
• Asia Pacific (Singapore) Region
• Asia Pacific (Sydney) Region
• Canada (Central) Region
• Europe (Frankfurt) Region
• Europe (Stockholm) Region
• Europe (Milan) Region
• Europe (Ireland) Region
• Europe (London) Region
• Europe (Paris) Region
• Middle East (UAE) Region
• Middle East (Bahrain) Region
• South America (São Paulo) Region
• US East (N. Virginia) Region
• US East (Ohio) Region
• US West (N. California) Region
• US West (Oregon) Region
• Asia Pacific (Jakarta) Region
• Asia Pacific (Melbourne) Region
• Israel (Tel Aviv) Region
The following video covers routing events between Regions using the https://console.aws.amazon.com/events/, AWS CloudFormation, and AWS Serverless Application Model: Cross-Region event routing

Creating rules that send events to a different AWS Region

Specifying an event bus in another AWS Region as a target is part of creating the rule.

To create a rule that sends events to a different AWS account using the console

1. Follow the steps in the ??? (p. 84) procedure.
2. In the ??? (p. 87) step, when prompted to choose a target type:
   a. Select EventBridge event bus.
   b. Select Event bus in a different account or Region.
   c. For Event bus as target, enter the ARN of the event bus you want to use.
3. Complete creating the rule following the procedure steps.

Sending and receiving Amazon EventBridge events between event buses in the same account and Region

You can configure EventBridge to send and receive events (p. 27) between event buses (p. 9) in the same AWS account and Region.

When you configure EventBridge to send or receive events between event buses, you use IAM roles on the sender event bus to give the sender event bus permission to send events to the receiver event bus. You use Resource-based (p. 405) policies on the receiver event bus to give the receiver event bus permission to receive events from the sender event bus. You can also allow or deny events from certain event buses, specific rules (p. 82) associated with the event bus, or events from specific sources. For more information about event bus permissions, including example policies, see Permissions for Amazon EventBridge event buses (p. 18).

The steps to configure EventBridge to send events to or receive events between event buses in your account include the following:

- To use an existing IAM role, you need to give either the sender event bus permissions to the receiver event bus or the receiver event bus permissions to the sender event bus.
- On the sender event bus, set up one or more rules that have the receiver event bus as the target and create an IAM role. For an example of the policy that should be attached to the role, see ??? (p. 21).
- On the receiver event bus, edit the permissions to allow events to be passed from the other event bus.
- On the receiver event, set up one or more rules that match events that come from the sender event bus.

Note
EventBridge can't route events received from a sender event bus to a third event bus.

Events sent from one event bus to another are charged as custom events. For more information, see Amazon EventBridge Pricing.

Creating rules that send events to a different event bus in the same AWS account and Region

To send events to another event bus, you create a rule with an event bus as a target. Specifying an event bus in the same AWS account and Region as a target is part of creating the rule.
To create a rule that sends events to a different event bus in the same AWS account and Region using the console

1. Follow the steps in the ??? (p. 84) procedure.
2. In the ??? (p. 87) step, when prompted to choose a target type:
   a. Select EventBridge event bus.
   b. Select Event bus in the same AWS account and Region.
   c. For Event bus as a target, select an event bus from the drop-down list.
3. Complete creating the rule following the procedure steps.
Amazon EventBridge input transformation

You can customize the text from an event (p. 27) before EventBridge passes the information to the target (p. 112) of a rule (p. 82). Using the input transformer in the console or the API, you define variables that use JSON path to reference values in the original event source. The transformed event is sent to a target instead of the original event. However, dynamic path parameters (p. 114) must reference the original event, not the transformed event. You can define up to 100 variables, assigning each a value from the input. Then you can use those variables in the Input Template as `<variable-name>`.

For a tutorial on using input transformer, see ??? (p. 252).

**Note**
EventBridge does not support all JSON Path syntax and evaluate it at runtime. Supported syntax includes:

- dot notation (for example, $.detail)
- dashes
- underscores
- alphanumeric characters
- array indices
- wildcards (*)

In this topic:

- Predefined variables (p. 137)
- Input transform examples (p. 137)
- Transforming input by using the EventBridge API (p. 140)
- Transforming input by using AWS CloudFormation (p. 140)
- Common Issues with transforming input (p. 140)
- Configuring an input transformer as part of creating a rule (p. 141)
- Testing a target input transformer using the EventBridge Sandbox (p. 143)

### Predefined variables

There are pre-defined variables you can use without defining a JSON path. These variables are reserved, and you can’t create variables with these names:

- `aws.events.rule-arn` — The Amazon Resource Name (ARN) of the EventBridge rule.
- `aws.events.rule-name` — The Name of the EventBridge rule.
- `aws.events.event.ingestion-time` — The time at which the event was received by EventBridge. This is an ISO 8601 timestamp. This variable is generated by EventBridge and can’t be overwritten.
- `aws.events.event` — The original event payload as JSON (without the detail field). Can only be used as a value for a JSON field, as its contents are not escaped.
- `aws.events.event.json` — The full original event payload as JSON. (with the detail field). Can only be used as a value for a JSON field, as its contents are not escaped.

### Input transform examples

The following is an example Amazon EC2 event.
When defining a rule in the console, select the **Input Transformer** option under **Configure input**. This option displays two text boxes: one for the **Input Path** and one for the **Input Template**.

**Input Path** is used to define variables. Use JSON path to reference items in your event and store those values in variables. For instance, you could create an **Input Path** to reference values in the example event by entering the following in the first text box. You can also use brackets and indices to get items from arrays.

**Note**

EventBridge replaces input transformers at runtime to ensure a valid JSON output. Because of this, put quotes around variables that refer to JSON path parameters, but do not put quotes around variables that refer to JSON objects or arrays.

```json
{
  "timestamp" : "$.time",
  "instance" : "$.detail.instance-id",
  "state" : "$.detail.state",
  "resource" : "$.resources[0]"
}
```

This defines four variables, `<timestamp>`, `<instance>`, `<state>`, and `<resource>`. You can reference these variables as you create your **Input Template**.

The **Input Template** is a template for the information you want to pass to your target. You can create a template that passes either a string or JSON to the target. Using the previous event and **Input Path**, the following **Input Template** examples will transform the event to the example output before routing it to a target.

<table>
<thead>
<tr>
<th>Description</th>
<th>Template</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple string</td>
<td>&quot;instance &lt;instance&gt; is in &lt;state&gt;&quot;</td>
<td>&quot;instance i-0123456789 is in RUNNING&quot;</td>
</tr>
<tr>
<td>String with escaped quotes</td>
<td>&quot;instance \&quot;&lt;instance&gt;\&quot; is in &lt;state&gt;&quot;</td>
<td>&quot;instance \&quot;i-0123456789\&quot; is in RUNNING&quot;</td>
</tr>
</tbody>
</table>

Note that this is the behavior in the EventBridge console. The AWS CLI escapes the slash characters and the result is
## Input transform examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Template</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple JSON</td>
<td><code>{ &quot;instance&quot; : &lt;instance&gt;, &quot;state&quot;: &lt;state&gt; }</code></td>
<td><code>{ &quot;instance&quot; : &quot;i-0123456789&quot;, &quot;state&quot;: &quot;RUNNING&quot; }</code></td>
</tr>
<tr>
<td>JSON with strings and variables</td>
<td><code>{ &quot;instance&quot; : &lt;instance&gt;, &quot;state&quot;: &quot;&lt;state&gt;&quot;, &quot;instanceStatus&quot;: &quot;instance \&quot;&lt;instance&gt;\&quot; is in \&quot;&lt;state&gt;\&quot; &quot; }</code></td>
<td><code>{ &quot;instance&quot; : &quot;i-0123456789&quot;, &quot;state&quot;: &quot;RUNNING&quot;, &quot;instanceStatus&quot;: &quot;instance \&quot;i-0123456789\&quot; is in RUNNING&quot; }</code></td>
</tr>
<tr>
<td>JSON with a mix of variables and static information</td>
<td><code>{ &quot;instance&quot; : &lt;instance&gt;, &quot;state&quot;: [ 9, &lt;state&gt;, true ], &quot;Transformed&quot;: &quot;Yes&quot; }</code></td>
<td><code>{ &quot;instance&quot; : &quot;i-0123456789&quot;, &quot;state&quot;: [ 9, &quot;RUNNING&quot;, true ], &quot;Transformed&quot;: &quot;Yes&quot; }</code></td>
</tr>
<tr>
<td>Including reserved variables in JSON</td>
<td><code>{ &quot;instance&quot; : &lt;instance&gt;, &quot;state&quot;: &lt;state&gt;, &quot;ruleArn&quot; : &lt;aws.events.rule-arn&gt;, &quot;ruleName&quot; : &lt;aws.events.rule-name&gt;, &quot;originalEvent&quot; : &lt;aws.events.event.json&gt; }</code></td>
<td><code>{ &quot;instance&quot; : &quot;i-0123456789&quot;, &quot;state&quot;: &quot;RUNNING&quot;, &quot;ruleArn&quot; : &quot;arn:aws:events:us-east-2:123456789012:rule/example&quot;, &quot;ruleName&quot; : &quot;example&quot;, &quot;originalEvent&quot; : { ... // commented for brevity } }</code></td>
</tr>
<tr>
<td>Including reserved variables in a string</td>
<td><code>&lt;aws.events.rule-name&gt;triggered&quot;</code></td>
<td>&quot;example triggered&quot;</td>
</tr>
<tr>
<td>Amazon CloudWatch log group</td>
<td><code>{ &quot;timestamp&quot; : &lt;timestamp&gt;, &quot;message&quot;: &quot;instance \&quot;&lt;instance&gt;\&quot; is in \&quot;&lt;state&gt;\&quot; &quot; }</code></td>
<td><code>{ &quot;timestamp&quot; : 2015-11-11T21:29:54Z, &quot;message&quot;: &quot;instance \&quot;i-0123456789\&quot; is in RUNNING&quot; }</code></td>
</tr>
</tbody>
</table>
Transforming input by using the EventBridge API

For information about using the EventBridge API to transform input, see [Use Input Transformer to extract data from an event and input that data to the target](#).

Transforming input by using AWS CloudFormation

For information about using AWS CloudFormation to transform input, see [AWS::Events::Rule InputTransformer](#).

Common Issues with transforming input

These are some common issues when transforming input in EventBridge:

- For Strings, quotes are required.
- There is no validation when creating JSON path for your template.
- If you specify a variable to match a JSON path that doesn't exist in the event, that variable isn't created and won't appear in the output.
- JSON properties like `aws.events.event.json` can only be used as the value of a JSON field, not inline in other strings.
- EventBridge doesn't escape values extracted by `Input Path`, when populating the `Input Template` for a target.
- If a JSON path references a JSON object or array, but the variable is referenced in a string, EventBridge removes any internal quotes to ensure a valid string. For example, for a variable `<detail>` pointed at `$.detail`, "Detail is `<detail>`" would result in EventBridge removing quotes from the object.

Therefore, if you want to output a JSON object based on a single JSON path variable, you must place it as a key. In this example, `{"detail": `<detail>`}`.

- Quotes are not required for variables that represent strings. They are permitted, but EventBridge automatically adds quotes to string variable values during transformation, to ensure the transformation output is valid JSON. EventBridge does not add quotes to variables that represent JSON objects or arrays. Do not add quotes for variables that represent JSON objects or arrays.

For example, the following input template includes variables that represent both strings and JSON objects:

```json
{
  "ruleArn" : <aws.events.rule-arn>,
  "ruleName" : <aws.events.rule-name>,
  "originalEvent" : <aws.events.event.json>
}
```

Resulting in valid JSON with proper quotation:

```json
{
  "ruleName" : "example",
  "originalEvent" : {
                ...
    // commented for brevity
  }
}
```
Configuring an input transformer as part of creating a rule

As part of creating a rule, you can specify an input transformer for EventBridge to use to process matching events prior to sending those event to the specified target. You can configure input transformers for targets that are AWS services or API destinations.

To create a target input transformer as part of a rule

1. Follow the steps for creating a rule as detailed in (p. 84).
2. In Step 3 - Select target(s), expand Additional settings.
3. For Configure target input, choose Input transformer in the dropdown.

   Click Configure input transformer.

   EventBridge displays the Configure input transformer dialog box.

4. In the Sample event section, choose a Sample event type against which you want to test your event pattern. You can choose an AWS event, a partner event, or enter your own custom event.

   AWS events

   Select from events emitted from supported AWS services.

   1. Select AWS events.
   2. Under Sample events, choose the desired AWS event. Events are organized by AWS service.

      When you select an event, EventBridge populates the sample event.

      For example, if you choose S3 Object Created, EventBridge displays a sample S3 Object Created event.

   3. (Optional) You can also select Copy to copy the sample event to your device's clipboard.

   Partner events

   Select from events emitted from third-party services that support EventBridge, such as Salesforce.

   1. Select EventBridge partner events.
   2. Under Sample events, choose the desired partner event. Events are organized by partner.

      When you select an event, EventBridge populates the sample event.

   3. (Optional) You can also select Copy to copy the sample event to your device's clipboard.

   Enter your own

   Enter your own event in JSON text.

   1. Select Enter your own.
   2. EventBridge populates the sample event with a template of required event attributes.
   3. Edit and add to the sample event as desired. The sample event must be valid JSON.
   4. (Optional) You can also choose any of the following options:

      • Copy – Copy the sample event to your device's clipboard.
      • Prettify – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.
5. (Optional) Expand the **Example input paths, Templates and Outputs** section to see examples of:

- How JSON paths are used to define variables that represent event data
- How those variables can be used in an input transformer template
- The resulting output that EventBridge sends to the target

For more detailed examples of input transformations, see ??? (p. 137).

6. In the **Target input transformer** section, define any variables you want to use in the input template.

Variables use JSON path to reference values in the original event source. You can then reference those variables in the input template in order to include data from the original source event in the transformed event that EventBridge passes to the target. You can define up to 100 variables. The input transformer must be valid JSON.

For example, suppose you had chosen AWS event **S3 Object Created** as your sample event for this input transformer. You could then define the following variables for use in your template:

```json
{
    "requester": "$.detail.requester",
    "key": "$.detail.object.key",
    "bucket": "$.detail.bucket.name"
}
```

(Optional) You can also choose **Copy** to copy the input transformer to your device's clipboard.

7. In the **Template** section, compose the template you want to use to determine what EventBridge passes to the target.

You can use JSON, strings, static information, variables you've defined as well as reserved variables. For more detailed examples of input transformations, see ??? (p. 137).

For example, suppose you had defined the variables in the previous example. You could then compose the following template, which references those variables, as well as reserved variables, and static information.

```json
{
    "<requester> has created the object "<key>" in the bucket "<bucket>",
    "RuleName": <aws.events.rule-name>,
    "ruleArn": <aws.events.rule-arn>,
    "Transformed": "Yes"
}
```

(Optional) You can also choose **Copy** to copy the template to your device's clipboard.

8. To test your template, select **Generate output**.

EventBridge processes the sample event based on the input template, and displays the transformed output generated under **Output**. This is the information that EventBridge will pass to the target in place of the original source event.

The generated output for the example input template described above would be the following:

```json
{
    "123456789012 has created the object "example-key" in the bucket "example-bucket",
    "RuleName": rule-name,
    "ruleArn": arn:aws:events:us-east-1:123456789012:rule/rule-name,
    "Transformed": "Yes"
}
```
Testing an input transformer

(Optional) You can also choose Copy to copy the generated output to your device's clipboard.

9. Select Confirm
10. Follow the rest of the steps for creating a rule as detailed in ???(p. 84).

Testing a target input transformer using the EventBridge Sandbox

You can use input transformers to customize the text from an event (p. 27) before EventBridge passes the information to the target (p. 112) of a rule (p. 82).

Configuring an input transformer is typically part of the larger process of specifying a target while creating a new rule (p. 84) or editing an existing one. Using the Sandbox in EventBridge, however, you can quickly configure an input transformer and use a sample event to confirm you are getting the desired output, without having to create or edit a rule.

For more information about input transformations, see ???(p. 137).

To test a target input transformer

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. Under Developer resources, choose Sandbox, and on the Sandbox page choose the Target input transformer tab.
3. In the Sample event section, choose a Sample event type against which you want to test your event pattern. You can choose an AWS event, a partner event, or enter your own custom event.

AWS events

Select from events emitted from supported AWS services.

1. Select AWS events.
2. Under Sample events, choose the desired AWS event. Events are organized by AWS service.

When you select an event, EventBridge populates the sample event.

For example, if you choose S3 Object Created, EventBridge displays a sample S3 Object Created event.

3. (Optional) You can also select Copy to copy the sample event to your device's clipboard.

Partner events

Select from events emitted from third-party services that support EventBridge, such as Salesforce.

1. Select EventBridge partner events.
2. Under Sample events, choose the desired partner event. Events are organized by partner.

When you select an event, EventBridge populates the sample event.

3. (Optional) You can also select Copy to copy the sample event to your device's clipboard.

Enter your own

Enter your own event in JSON text.
1. Select **Enter your own**.
2. EventBridge populates the sample event with a template of required event attributes.
3. Edit and add to the sample event as desired. The sample event must be valid JSON.
4. (Optional) You can also choose any of the following options:
   - **Copy** – Copy the sample event to your device's clipboard.
   - **Prettify** – Makes the JSON text easier to read by adding line breaks, tabs, and spaces.
5. (Optional) Expand the **Example input paths, Templates and Outputs** section to see examples of:
   - How JSON paths are used to define variables that represent event data
   - How those variables can be used in an input transformer template
   - The resulting output that EventBridge sends to the target

For more detailed examples of input transformations, see ??? (p. 137).

5. In the **Target input transformer** section, define any variables you want to use in the input template.

Variables use JSON path to reference values in the original event source. You can then reference those variables in the input template in order to include data from the original source event in the transformed event that EventBridge passes to the target. You can define up to 100 variables. The input transformer must be valid JSON.

For example, suppose you had chosen AWS event **S3 Object Created** as your sample event for this input transformer. You could then define the following variables for use in your template:

```json
{
    "requester": ".\$\{detail.requester\}",
    "key": "\$.\{detail.object.key\}",
    "bucket": "\$.\{detail.bucket.name\}"
}
```

(Optional) You can also choose **Copy** to copy the input transformer to your device's clipboard.

6. In the **Template** section, compose the template you want to use to determine what EventBridge passes to the target.

You can use JSON, strings, static information, variables you've defined as well as reserved variables. For more detailed examples of input transformations, see ??? (p. 137).

For example, suppose you had defined the variables in the previous example. You could then compose the following template, which references those variables, as well as reserved variables, and static information.

```json
{
    "<requester> has created the object \"<key>\" in the bucket \"<bucket>\"",
    "RuleName": <aws.events.rule-name>,
    "ruleArn" : <aws.events.rule-arn>,
    "Transformed": "Yes"
}
```

(Optional) You can also choose **Copy** to copy the template to your device's clipboard.

7. To test your template, select **Generate output**.

EventBridge processes the sample event based on the input template, and displays the transformed output generated under **Output**. This is the information that EventBridge will pass to the target in place of the original source event.
The generated output for the example input template described above would be the following:

```
{
    "123456789012 has created the object "example-key" in the bucket "example-bucket"",
    "RuleName": rule-name,
    "ruleArn" : arn:aws:events:us-east-1:123456789012:rule/rule-name,
    "Transformed": "Yes"
}
```

(Optional) You can also choose **Copy** to copy the generated output to your device's clipboard.
Amazon EventBridge archive and replay

In EventBridge, you can create an archive of events (p. 27) so that you can easily replay them at a later time. For example, you might want to replay events to recover from errors or to validate new functionality in your application.

Note
There may be a delay between an event being published to an event bus and the event arriving in the archive. We recommend you delay replaying archived events for 10 minutes to make sure all events are replayed.

The following video demonstrates the use of archive and replay: Creating archives and replays

Topics
- Archiving Amazon EventBridge events (p. 147)
- Replaying archived Amazon EventBridge events (p. 149)
Archiving Amazon EventBridge events

When you create an archive in EventBridge, you can determine which events (p. 27) are sent to the archive by specifying an event pattern (p. 62). EventBridge sends events that match the event pattern to the archive. You also set the retention period to store events in the archive before they are discarded.

By default, EventBridge encrypts event data in an archive using 256-bit Advanced Encryption Standard (AES-256) under an AWS owned CMK, which helps secure your data from unauthorized access.

**Note**
Expired events are usually deducted from the EventCount and SizeBytes values of the DescribeArchive operation every 24 hours. Therefore, recently expired events may not be reflected in these values.

**To create an archive for all events**
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the left navigation pane, choose Archives.
3. Choose Create archive.
4. Under Archive detail, enter a Name for the archive. The name must be unique to your account in the selected Region.
   You can’t change the name after you create the archive.
5. (Optional) Enter a Description for the archive.
6. For Source, select the event bus that emits the events to send to the archive.
7. For Retention period, do one of the following:
   • Choose Indefinite to retain the events in the archive and not ever delete them.
   • Enter the number of days to retain the events. After the number of days specified, EventBridge deletes the events from the archive.
8. Choose Next.
9. Under Event pattern, choose No event filtering.
10. Choose Create archive.

**To create an archive with an event pattern**
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the left navigation pane, choose Archives.
3. Choose Create archive.
4. Under Archive detail, enter a Name for the archive. The name must be unique to your account in the selected Region.
   You can’t change the name after you create the archive.
5. (Optional) Enter a Description for the archive.
6. For Source, select the event bus that emits the events to send to the archive.
7. For Retention period, do one of the following:
   • Choose Indefinite* to retain the events in the archive and not ever delete them.
   • Enter the number of days to retain the events. After the number of days specified, EventBridge deletes the events from the archive.
8. Choose Next.
10. Do one of the following:
Archiving events

- Select **Pattern builder**, then choose the **Service provider**. If you choose **AWS**, also select the **AWS service name** and **Event type** to use in the pattern.
- Select **JSON editor** to create a pattern manually. You can also copy the pattern from a rule and then paste it into the JSON editor.

11. Choose **Create archive**.

To confirm that events are successfully sent to the archive, you can use the **DescribeArchive** operation of the EventBridge API to see if the **EventCount** reflects the number of events in the archive. If it is 0, there are no events in the archive.
Replaying archived Amazon EventBridge events

After you create an archive, you can then replay events from the archive. For example, if you update an application with additional functionality, you can replay historical events to ensure that the events are reprocessed to keep the application consistent. You can also use an archive to replay events for new functionality. When you replay events, you can specify which archive to replay events from, the start and end time for the event to replay, the event bus, or one or more rules to replay the events to.

Events aren't necessarily replayed in the same order that they were added to the archive. A replay processes events to replay based on the time in the event, and replays them on one minute intervals. If you specify an event start time and an event end time that covers a 20 minute time range, the events are replayed from the first minute of that 20 minute range first. Then the events from the second minute are replayed. You can use the DescribeReplay operation of the EventBridge API to determine the progress of a replay. EventLastReplayedTime returns the time stamp of the last event replayed.

Events are replayed based on, but separate from, the PutEvents transactions per second limit for the AWS account. You can request an increase to the limit for PutEvents. For more information, see Amazon EventBridge Quotas.

Note
You can have a maximum of 10 active concurrent replays per account per AWS Region.

To start an event replay
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the left navigation pane, choose Replays.
3. Choose Start new replay.
4. Enter a Name for the replay and, optionally, a Description.
5. For Source, select the archive to replay events from.
6. For destination, you can replay events only to the same event bus that emitted the events.
7. For Specify rules, do one of the following:
   • Choose All rules to replay events to all rules.
   • Choose Specify rules, and then select the rule or rules to replay the events to.
8. Under Replay time frame, specify the Date, Time, and Time zone for the Start time and the End time. Only events that occurred between the Start time and End time are replayed.

When the events from the archived are replayed, the status of the replay is Completed.

If you start a replay and then want to interrupt it, you can cancel it as long as the status is Starting or Running.

To cancel a replay
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the left navigation pane, choose Replays.
3. Choose the replay to cancel.
4. Choose Cancel.
Amazon EventBridge Pipes

Amazon EventBridge Pipes connects sources to targets. Pipes are intended for point-to-point integrations between supported sources (p. 163) and targets (p. 188), with support for advanced transformations and enrichment (p. 187). It reduces the need for specialized knowledge and integration code when developing event-driven architectures, fostering consistency across your company's applications. To set up a pipe, you choose the source, add optional filtering, define optional enrichment, and choose the target for the event data.

Note
You can also route events using event buses. Event buses are well-suited for many-to-many routing of events between event-driven services. For more information, see ??? (p. 9).

How EventBridge Pipes work

At a high level, here's how EventBridge Pipes works:

1. You create a pipe in your account. This includes:
   • Specifying one of the supported event sources (p. 163) from which you want your pipe to receive events.
   • Optionally, configuring a filter so that the pipe only processes a subset of the events it receives from the source.
   • Optionally, configuring an enrichment step that enhances the event data before sending it to the target.
   • Specifying one of the supported targets (p. 188) to which you want your pipe to send events.
2. The event source begins sending events to the pipe, and the pipe processes the event before sending it to the target.
   • If you have configured a filter, the pipe evaluates the event and only sends it to the target if it matches that filter.
     You are only charged for those events that match the filter.
   • If you have configured an enrichment, the pipe performs that enrichment on the event before sending it to the target.
     If the events are batched, the enrichment maintains the ordering of the events in the batch.
For example, a pipe could be used to create an e-commerce system. Suppose you have an API that contains customer information, such as shipping addresses.

1. You then create a pipe with the following:
   - An Amazon SQS order received message queue as the event source.
   - An EventBridge API Destination as an enrichment
   - An AWS Step Functions state machine as the target
2. Then, when an Amazon SQS order received message appears in the queue, it is sent to your pipe.
3. The pipe then sends that data to the EventBridge API Destination enrichment, which returns the customer information for that order.
4. Lastly, the pipe sends the enriched data to the AWS Step Functions state machine, which processes the order.

EventBridge Pipes concepts

Here's a closer look at the basic components of EventBridge Pipes.

Pipe

A pipe routes events from a single source to a single target. The pipe also includes the ability to filter for specific events, and to perform enrichments on the event data before it is sent to the target.
Source

EventBridge Pipes receives event data from a variety of sources, applies optional filters and enrichment to that data, and sends it to a target. If a source enforces order to the events sent to pipes, that order is maintained throughout the entire process to the target.

For more information about sources, see [?? (p. 163)].
Filters

A pipe can filter a given source's events and then process only a subset of those events. To configure filtering on a pipe, you define an event pattern the pipe uses to determine which events to send to the target.

You are only charged for those events that match the filter.

For more information, see ??? (p. 183).

Enrichment

With the enrichment step of EventBridge Pipes, you can enhance the data from the source before sending it to the target. For example, you might receive Ticket created events that don't include the full ticket data. Using enrichment, you can have a Lambda function call the get-ticket API for the full ticket details. The pipe can then send that information to a target (p. 188).

For more information about enriching event data, see ??? (p. 187).

Target

After the event data has been filtered and enriched, you can specify the pipe send it to a specific target, such as an Amazon Kinesis stream or an Amazon CloudWatch log group. For a list of the available targets, see ??? (p. 188).

You can transform the data after it’s enhanced and before it’s sent by the pipe to the target. For more information, see ??? (p. 193).

Multiple pipes, each with a different source, can send events to the same target.

You can also use pipes and event buses together to send events to multiple targets. A common use case is to create a pipe with an event bus as its target; the pipe sends events to the event bus, which then sends those events on to multiple targets. For example, you could create a pipe with a DynamoDB stream for a source, and an event bus as the target. The pipe receives events from the DynamoDB stream and sends them to the event bus, which then sends them on to multiple targets according to the rules you’ve specified on the event bus.

Permissions for Amazon EventBridge Pipes

When setting up a pipe, you can use an existing execution role, or have EventBridge create one for you with the needed permissions. The permissions EventBridge Pipes requires vary based on the source type, and are listed below. If you’re setting up your own execution role, you must add these permissions yourself.

Note
If you're unsure of the exact well-scoped permissions required to access the source, use the EventBridge Pipes console to create a new role, then inspect the actions listed in the policy.

Topics
- DynamoDB execution role permissions (p. 154)
- Kinesis execution role permissions (p. 154)
- Amazon MQ execution role permissions (p. 154)
- Amazon MSK execution role permissions (p. 154)
- Self managed Apache Kafka execution role permissions (p. 155)
DynamoDB execution role permissions

For DynamoDB Streams, EventBridge Pipes requires the following permissions to manage resources that are related to your DynamoDB data stream.

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

Kinesis execution role permissions

For Kinesis, EventBridge Pipes requires the following permissions to manage resources that are related to your Kinesis data stream.

- `kinesis:DescribeStream`
- `kinesis:DescribeStreamSummary`
- `kinesis:GetRecords`
- `kinesis:GetShardIterator`
- `kinesis:ListShards`
- `kinesis:ListStreams`
- `kinesis:SubscribeToShard`

Amazon MQ execution role permissions

For Amazon MQ, EventBridge Pipes requires the following permissions to manage resources that are related to your Amazon MQ message broker.

- `mq:DescribeBroker`
- `secretsmanager:GetSecretValue`
- `ec2:CreateNetworkInterface`
- `ec2:DeleteNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DescribeSecurityGroups`
- `ec2:DescribeSubnets`
- `ec2:DescribeVpcs`
- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Amazon MSK execution role permissions

For Amazon MSK, EventBridge requires the following permissions to manage resources that are related to your Amazon MSK topic.
Note
If you’re using IAM role-based authentication, your execution role will need the permissions listed in ??? (p. 173) in addition the ones listed below.

- `kafka:DescribeClusterV2`
- `kafka:GetBootstrapBrokers`
- `ec2:CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DescribeVpcs`
- `ec2:DeleteNetworkInterface`
- `ec2:DescribeSubnets`
- `ec2:DescribeSecurityGroups`
- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Self managed Apache Kafka execution role permissions

For self managed Apache Kafka, EventBridge requires the following permissions to manage resources that are related to your self managed Apache Kafka stream.

Required permissions

To create and store logs in a log group in Amazon CloudWatch Logs, your pipe must have the following permissions in its execution role:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Optional permissions

Your pipe might also need permissions to:

- Describe your Secrets Manager secret.
- Access your AWS Key Management Service (AWS KMS) customer managed key.
- Access your Amazon VPC.

Secrets Manager and AWS KMS permissions

Depending on the type of access control that you’re configuring for your Apache Kafka brokers, your pipe might need permission to access your Secrets Manager secret or to decrypt your AWS KMS customer managed key. To access these resources, your function's execution role must have the following permissions:

- `secretsmanager:GetSecretValue`
- `kms:Decrypt`
VPC permissions

If only users within a VPC can access your self managed Apache Kafka cluster, your pipe must have permission to access your Amazon VPC resources. These resources include your VPC, subnets, security groups, and network interfaces. To access these resources, your pipe's execution role must have the following permissions:

- `ec2:CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DescribeVpcs`
- `ec2:DeleteNetworkInterface`
- `ec2:DescribeSubnets`
- `ec2:DescribeSecurityGroups`

Amazon SQS execution role permissions

For Amazon SQS, EventBridge requires the following permissions to manage resources that are related to your Amazon SQS queue.

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`

Enrichment and target permissions

To make API calls on the resources that you own, EventBridge Pipes needs appropriate permission. EventBridge Pipes uses the IAM role that you specify on the pipe for enrichment and target calls using the IAM principal `pipes.amazonaws.com`.

Creating an Amazon EventBridge pipe

EventBridge Pipes enables you to create point-to-point integrations between sources and targets, including advanced event transformations and enrichment. To create an EventBridge pipe, you perform the following steps:

1. ??? (p. 156)
2. ??? (p. 159)
3. ??? (p. 160)
4. ??? (p. 160)
5. ??? (p. 161)

For information on how to create a pipe using the AWS CLI, see `create-pipe` in the AWS CLI Command Reference.

Specifying a source

To start, specify the source from which you want the pipe to receive events.
To specify a pipe source by using the console

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. On the navigation pane, choose Pipes.
3. Choose Create pipe.
4. Enter a name for the pipe.
5. (Optional) Add a description for the pipe.
6. On the Build pipe tab, for Source, choose the type of source you want to specify for this pipe, and configure the source.

Configuration properties differ based on the type of source you choose:

DynamoDB

1. For Source, choose DynamoDB.
2. For DynamoDB stream, choose the stream you want to use as a source.
3. For Starting position, choose one of the following:
   - Latest – Start reading the stream with the most recent record in the shard.
   - Trim horizon – Start reading the stream with the last untrimmed record in the shard. This is the oldest record in the shard.
4. (Optional) For Additional setting - optional, do the following:
   a. For Batch size - optional, enter a maximum number of records for each batch. The default value is 100.
   b. For Batch window - optional, enter a maximum number of seconds to gather records before proceeding.
   c. For Concurrent batches per shard - optional, enter the number of batches from the same shard that can be read at the same time.
   d. For On partial batch item failure, choose the following:
      - AUTOMATIC_BISECT – Halve each batch and retry each half until all the records are processed or there is one failed message remaining in the batch.

Note
If you don't choose AUTOMATIC_BISECT, you can return specific failed records and only those get retried.

Kinesis

To configure a Kinesis source by using the console

1. For Source, choose Kinesis.
2. For Kinesis stream, choose the stream that you want to use as a source.
3. For Starting position, choose one of the following:
   - Latest – Start reading the stream with the most recent record in the shard.
   - Trim horizon – Start reading the stream with the last untrimmed record in the shard. This is the oldest record in the shard.
   - At timestamp – Start reading the stream from a specified time. Under Timestamp, enter a data and time using YYYY/MM/DD and hh:mm:ss format.
4. (Optional) For Additional setting - optional, do the following:
   a. For Batch size - optional, enter a maximum number of records for each batch. The default value is 100.
   b. (Optional) For Batch window - optional, enter a maximum number of seconds to gather records before proceeding.
c. For **Concurrent batches per shard - optional**, enter the number of batches from the same shard that can be read at the same time.

d. For **On partial batch item failure**, choose the following:
   - **AUTOMATIC_BISECT** – Halve each batch and retry each half until all the records are processed or there is one failed message remaining in the batch.

   **Note**
   If you don’t choose **AUTOMATIC_BISECT**, you can return specific failed records and only those get retried.

Amazon MQ

**To configure an Amazon MQ source by using the console**

1. For **Source**, choose Amazon MQ.
2. For **Amazon MQ broker**, choose the stream you want to use as a source.
3. For **Queue name**, enter the name of the queue that the pipe will read from.
4. For **Authentication Method**, choose **BASIC_AUTH**.
5. For **Secret key**, choose the secret key.
6. (Optional) For **Additional setting - optional**, do the following:
   a. For **Batch size - optional**, enter a maximum number of messages for each batch. The default value is 100.
   b. For **Batch window - optional**, enter a maximum number of seconds to gather records before proceeding.

Amazon MSK

**To configure an Amazon MSK source by using the console**

1. For **Source**, choose Amazon MSK.
2. For **Amazon MSK cluster**, choose the cluster that you want to use.
3. For **Topic name**, enter the name of topic that the pipe will read from.
4. (Optional) For **Consumer Group ID - optional**, enter the ID of the consumer group you want the pipe to join.
5. (Optional) For **Authentication - optional**, turn on **Use Authentication** and do the following:
   a. For **Authentication method**, choose the type you want.
   b. For **Secret key**, choose the secret key.
6. (Optional) For **Additional setting - optional**, do the following:
   a. For **Batch size - optional**, enter a maximum number of records for each batch. The default value is 100.
   b. For **Batch window - optional**, enter a maximum number of seconds to gather records before proceeding.
   c. For **Starting position**, choose one of the following:
      - **Latest** – Start reading the topic with the most recent record in the shard.
      - **Trim horizon** – Start reading the topic with the last untrimmed record in the shard. This is the oldest record in the shard.

   **Note**
   **Trim horizon** is the same as **Earliest** for Apache Kafka.
Self managed Apache Kafka

To configure a self managed Apache Kafka source by using the console

1. For Source, choose Self-managed Apache Kafka.
2. For Bootstrap servers, enter the host:port pair addresses of your brokers.
3. For Topic name, enter the name of topic that the pipe will read from.
4. (Optional) For VPC, choose the VPC that you want. Then, for VPC subnets, choose the desired subnets. For VPC security groups, choose the security groups.
5. (Optional) For Authentication - optional, turn on Use Authentication and do the following:
   a. For Authentication method, choose the authentication type.
   b. For Secret key, choose the secret key.
6. (Optional) For Additional setting - optional, do the following:
   a. For Starting position, choose one of the following:
      • Latest – Start reading the stream with the most recent record in the shard.
      • Trim horizon – Start reading the stream with the last untrimmed record in the shard. This is the oldest record in the shard.
   b. For Batch size - optional, enter a maximum number of records for each batch. The default value is 100.
   c. For Batch window - optional, enter a maximum number of seconds to gather records before proceeding.

Amazon SQS

To configure an Amazon SQS source by using the console

1. For Source, choose SQS.
2. For SQS queue, choose the queue you want to use.
3. (Optional) For Additional setting - optional, do the following:
   a. For Batch size - optional, enter a maximum number of records for each batch. The default value is 100.
   b. For Batch window - optional, enter a maximum number of seconds to gather records before proceeding.

Configuring event filtering (optional)

You can add filtering to your pipe so you’re sending only a subset of events from your source to the target.

To configure filtering by using the console

1. Choose Filtering.
2. Under Sample event - optional, you’ll see a sample event that you can use to build your event pattern, or you can enter your own event by choosing Enter your own.
3. Under Event pattern, enter the event pattern that you want to use to filter the events. For more information about building event patterns, see ??? (p. 62).

   The following is an example event pattern that only sends events with the value Seattle in the City field.
Now that events are being filtered, you can add optional enrichment and a target for the pipe.

### Defining event enrichment (optional)

You can send the event data for enrichment to a Lambda function, AWS Step Functions state machine, Amazon API Gateway, or API destination.

**To select enrichment**

1. Choose **Enrichment**.
2. Under **Details**, for **Service**, select the service and related settings you want to use for enrichment.

You can also transform the data before sending it to be enhanced.

**(Optional) To define the input transformer**

1. Choose **Enrichment Input Transformer - optional**.
2. For **Sample events/Event Payload**, choose the sample event type.
3. For **Transformer**, enter the transformer syntax, such as "Event happened at <$.detail.field>." where <$.detail.field> is a reference to a field from the sample event. You can also double-click a field from the sample event to add it to the transformer.
4. For **Output**, verify that the output looks like you want it to.

Now that the data has been filtered and enhanced, you must define a target to send the event data to.

### Configuring a target

**To configure a target**

1. Choose **Target**.
2. Under **Details**, for **Target service**, choose the target. The fields that display vary depending on the target that you choose. Enter information specific to this target type, as needed.

You can also transform the data before sending it to the target.

**(Optional) To define the input transformer**

1. Choose **Target Input Transformer - optional**.
2. For **Sample events/Event Payload**, choose the sample event type.
3. For **Transformer**, enter the transformer syntax, such as "Event happened at <$.detail.field>." where <$.detail.field> is a reference to a field from the sample event. You can also double-click a field from the sample event to add it to the transformer.
4. For **Output**, verify that the output looks like you want it to.

Now that the pipe is configured, make sure that its settings are configured correctly.
Configuring the pipe settings

A pipe is active by default, but you can deactivate it. You can also specify the permissions of the pipe, set up pipe logging, and add tags.

To configure the pipe settings

1. Choose the **Pipe settings** tab.
2. By default, newly created pipes are active as soon as they're created. If you want to create an inactive pipe, under **Activation**, for **Activate pipe**, turn off **Active**.
3. Under **Permissions**, for **Execution role**, do one of the following:
   a. To have EventBridge create a new execution role for this pipe, choose **Create a new role for this specific resource**. Under **Role name**, you can optionally edit the role name.
   b. To use an existing execution role, choose **Use existing role**. Under **Role name**, choose the role.
4. (Optional) If you have specified a Kinesis or DynamoDB stream as the pipe source, you can configure a retry policy and dead-letter queue (DLQ).

   For **Retry policy and Dead-letter queue - optional**, do the following:

   Under **Retry policy**, do the following:

   a. If you want to turn on retry policies, turn on **Retry**. By default, newly created pipes don't have a retry policy turned on.
   b. For **Maximum age of event**, enter a value between one minute (00:01) and 24 hours (24:00).
   c. For **Retry attempts**, enter a number between 0 and 185.
   d. If you want to use a dead-letter queue (DLQ), turn on **Dead-letter queue**, choose the method of your choice, and choose the queue or topic you'd like to use. By default, newly created pipes don't use a DLQ.

5. (Optional) Under **Logs - optional**, you can set up how EventBridge Pipes sends logging information to supported services, including how to configure those logs.

   For more information about logging pipe records, see ?? (p. 197).

   CloudWatch logs is selected as a log destination by default, as is the ERROR log level. So, by default, EventBridge Pipes creates a new CloudWatch log group to which it sends log records containing the ERROR level of detail.

   To have EventBridge Pipes send log records to any of the supported log destinations, do the following:

   a. Under **Logs - optional**, choose the destinations to which you want log records delivered.
   b. For **Log level**, choose the level of information for EventBridge to include in log records. The ERROR log level is selected by default.

      For more information, see ?? (p. 198).
   c. Select **Include execution data** if you want EventBridge to include event payload information and service request and response information in log records.

      For more information, see ?? (p. 199).
   d. Configure each log destination you selected:

      For CloudWatch Logs logs, under **CloudWatch logs** do the following:

      • For **CloudWatch log group**, choose whether to have EventBridge create a new log group, or you can select an existing log group or specifying the ARN of an existing log group.
• For new log groups, edit the log group name as desired.

CloudWatch logs is selected by default.

For Kinesis Data Firehose stream logs, under Kinesis Data Firehose stream log, select the Kinesis Data Firehose stream.

For Amazon S3 logs, under S3 logs do the following:

• Enter the name of the bucket to use as the log destination.
• Enter the AWS account ID of the bucket owner.
• Enter any prefix text you want used when EventBridge creates S3 objects.

For more information, see Organizing objects using prefixes in the Amazon Simple Storage Service User Guide.

• Choose how you want EventBridge to format S3 log records:
  • json: JSON
  • plain: Plain text
  • w3c: W3C extended logging file format

6. (Optional) Under Tags - optional, choose Add new tag and enter one or more tags for the rule. For more information, see ??? (p. 453).

7. Choose Create pipe.

Validating configuration parameters

After a pipe is created, EventBridge validates the following configuration parameters:

• **IAM role** – Because the source of a pipe can't be changed after the pipe is created, EventBridge verifies that the provided IAM role can access the source.
  
  **Note**
  
  EventBridge doesn't perform the same validation for enrichments or targets because they can be updated after the pipe is created.

• **Batching** – EventBridge validates that the batch size of the source doesn't exceed the maximum batch size of the target. If it does, EventBridge requires a lower batch size. Additionally, if a target doesn't support batching, you can't configure batching in EventBridge for the source.

• **Enrichments** – EventBridge validates that the batch size for API Gateway and API destination enrichments is 1 because only batch sizes of 1 are supported.

Starting or stopping a pipe

By default, a pipe is Running and processes events when it's created.

If you create a pipe with Amazon SQS, Kinesis, or DynamoDB sources, pipe creation can typically take a minute or two.

If you create a pipe with Amazon MSK, self managed Apache Kafka, or Amazon MQ sources, pipes creation can take up to ten minutes.

**To create a pipe without processing events using the console**

• Turn off the Activate pipe setting.
To create a pipe without processing events programmatically

- In your API call, set the `DesiredState` to `Stopped`.

To start or stop an existing pipe using the console

- On the `Pipes settings` tab, under `Activation`, for `Activate pipe`, turn `Active` on or off.

To start or stop an existing pipe programmatically

- In your API call, set the `DesiredState` parameter to either `RUNNING` or `STOPPED`.

There can be a delay between when a pipe is `STOPPED` and when it no longer processes events:

- For Amazon SQS and stream sources, this delay is typically less than two minutes.
- For Amazon MQ and Apache Kafka sources, this delay may be up to fifteen minutes.

Amazon EventBridge Pipes sources

EventBridge Pipes receives event data from a variety of sources, applies optional filters and enrichments to that data, and sends it to a destination.

If a source enforces order to the events sent to EventBridge Pipes, that order is maintained throughout the entire process to the destination.

The following sources are currently available for EventBridge Pipes:

- Amazon DynamoDB stream (p. 163)
- Amazon Kinesis stream (p. 166)
- Amazon MQ broker (p. 168)
- Amazon MSK stream (p. 171)
- Self managed Apache Kafka stream (p. 176)
- Amazon SQS queue (p. 180)

Amazon DynamoDB stream as a source

You can use EventBridge Pipes to receive records in a DynamoDB stream. You can then optionally filter or enhance these records before sending them to a target for processing. There are settings specific to Amazon DynamoDB Streams that you can choose when setting up the pipe. EventBridge Pipes maintains the order of records from the data stream when sending that data to the destination.

**Important**
Disabling a DynamoDB stream that is the source of a pipe results in that pipe becoming unusable, even if you then re-enable the stream. This happens because:

- You cannot stop, start, or update a pipe whose source is disabled.
- You cannot update a pipe with a new source after creation. When you re-enable a DynamoDB stream, that stream is assigned a new Amazon Resource Name (ARN), and is no longer associated with your pipe.

If you do re-enable the DynamoDB stream, you will then need to create a new pipe using the stream's new ARN.
Example event

The following sample event shows the information that's received by the pipe. You can use this event to create and filter your event patterns, or to define input transformation. Not all of the fields can be filtered. For more information about which fields you can filter, see [??](p. 183).

```json
[
  {
    "eventID": "1",
    "eventVersion": "1.0",
    "dynamodb": {
      "Keys": {
        "Id": {
          "N": "101"
        }
      },
      "NewImage": {
        "Message": {
          "S": "New item!"
        },
        "Id": {
          "N": "101"
        }
      },
      "StreamViewType": "NEW_AND_OLD_IMAGES",
      "SequenceNumber": "111",
      "SizeBytes": 26
    },
    "awsRegion": "us-west-2",
    "eventName": "INSERT",
    "eventSourceARN": "arn:aws:dynamodb:us-east-1:111122223333:table/EventSourceTable",
    "eventSource": "aws:dynamodb"
  },
  {
    "eventID": "2",
    "eventVersion": "1.0",
    "dynamodb": {
      "OldImage": {
        "Message": {
          "S": "New item!"
        },
        "Id": {
          "N": "101"
        }
      },
      "SequenceNumber": "222",
      "Keys": {
        "Id": {
          "N": "101"
        }
      },
      "SizeBytes": 59,
      "NewImage": {
        "Message": {
          "S": "This item has changed"
        },
        "Id": {
          "N": "101"
        }
      },
      "StreamViewType": "NEW_AND_OLD_IMAGES"
    },
    "awsRegion": "us-west-2",
    "eventName": "MODIFY",
    "eventSourceARN": "arn:aws:dynamodb:us-east-1:111122223333:table/EventSourceTable"
  }
]
```
Polling and batching streams

EventBridge polls shards in your DynamoDB stream for records at a base rate of four times per second. When records are available, EventBridge processes the event and waits for the result. If processing succeeds, EventBridge resumes polling until it receives more records.

By default, EventBridge invokes your pipe as soon as records are available. If the batch that EventBridge reads from the source has only one record in it, only one event is processed. To avoid processing a small number of records, you can tell the pipe to buffer records for up to five minutes by configuring a batching window. Before processing the events, EventBridge continues to read records from the source until it has gathered a full batch, the batching window expires, or the batch reaches the payload limit of 6 MB.

You can also increase concurrency by processing multiple batches from each shard in parallel. EventBridge can process up to 10 batches in each shard simultaneously. If you increase the number of concurrent batches per shard, EventBridge still ensures in-order processing at the partition key level.

Configure the ParallelizationFactor setting to process one shard of a Kinesis or DynamoDB data stream with more than one Pipe execution simultaneously. You can specify the number of concurrent batches that EventBridge polls from a shard via a parallelization factor from 1 (default) to 10. For example, when you set ParallelizationFactor to 2, you can have 200 concurrent EventBridge Pipe executions at maximum to process 100 Kinesis data shards. This helps scale up the processing throughput when the data volume is volatile and the IteratorAge is high. Note that parallelization factor will not work if you are using Kinesis aggregation.

Polling and stream starting position

Be aware that stream source polling during pipe creation and updates is eventually consistent.

- During pipe creation, it may take several minutes to start polling events from the stream.
- During pipe updates to the source polling configuration, it may take several minutes to stop and restart polling events from the stream.

This means that if you specify LATEST as the starting position for the stream, the pipe could miss events sent during pipe creation or updates. To ensure no events are missed, specify the stream starting position as TRIM_HORIZON.

Reporting batch item failures

When EventBridge consumes and processes streaming data from an source, by default it checkpoints to the highest sequence number of a batch, but only when the batch is a complete success. To avoid reprocessing successfully processed messages in a failed batch, you can configure your enrichment or target to return an object indicating which messages succeeded and which failed. This is called a partial batch response.

For more information, see ??? (p. 192).

Success and failure conditions

If you return any of the following, EventBridge treats a batch as a complete success:

- An empty batchItemFailure list
• A null batchItemFailure list
• An empty EventResponse
• A null EventResponse

If you return any of the following, EventBridge treats a batch as a complete failure:
• An empty string itemIdentifier
• A null itemIdentifier
• An itemIdentifier with a bad key name

EventBridge retries failures based on your retry strategy.

**Amazon Kinesis stream as a source**

You can use EventBridge Pipes to receive records in a Kinesis data stream. You can then optionally filter or enhance these records before sending them to one of the available destinations for processing. There are settings specific to Kinesis that you can choose when setting up the pipe. EventBridge Pipes maintains the order of records from the data stream when sending that data to the destination.

A Kinesis data stream is a set of **shards**. Each shard contains a sequence of data records. A **consumer** is an application that processes the data from a Kinesis data stream. You can map an EventBridge Pipe to a shared-throughput consumer (standard iterator), or to a dedicated-throughput consumer with **enhanced fan-out**.

For standard iterators, EventBridge uses the HTTP protocol to poll each shard in your Kinesis stream for records. The pipe shares the read throughput with other consumers of the shard.

To minimize latency and maximize read throughput, you can create a data stream consumer with enhanced fan-out. Stream consumers get a dedicated connection to each shard that doesn't impact other applications reading from the stream. The dedicated throughput can help if you have many applications reading the same data, or if you're reprocessing a stream with large records. Kinesis pushes records to EventBridge over HTTP/2. For information about Kinesis data streams, see [Reading Data from Amazon Kinesis Data Streams](#).

**Example event**

The following sample event shows the information that is received by the pipe. You can use this event to create and filter your event patterns, or to define input transformation. Not all of the fields can be filtered. For more information about which fields you can filter, see [??](p. 183).

```json
[
  {
    "kinesisSchemaVersion": "1.0",
    "partitionKey": "1",
    "sequenceNumber": "4959033827149025660059692538361571095921575989136588898",
    "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
    "approximateArrivalTimestamp": 1545084650.987,
    "eventSource": "aws:kinesis",
    "eventVersion": "1.0",
    "eventId": "shardId-000000000006:4959033827149025660059692538361571095921575989136588898",
    "eventName": "aws:kinesis:record",
    "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
    "awsRegion": "us-east-2",
    "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
  },
]```
Polling and batching streams

EventBridge polls shards in your Kinesis stream for records at a base rate of four times per second. When records are available, EventBridge processes the event and waits for the result. If processing succeeds, EventBridge resumes polling until it receives more records.

By default, EventBridge invokes your pipe as soon as records are available. If the batch that EventBridge reads from the source has only one record in it, only one event is processed. To avoid processing a small number of records, you can tell the pipe to buffer records for up to five minutes by configuring a batching window. Before processing the events, EventBridge continues to read records from the source until it has gathered a full batch, the batching window expires, or the batch reaches the payload limit of 6 MB.

You can also increase concurrency by processing multiple batches from each shard in parallel. EventBridge can process up to 10 batches in each shard simultaneously. If you increase the number of concurrent batches per shard, EventBridge still ensures in-order processing at the partition key level.

Configure the ParallelizationFactor setting to process one shard of a Kinesis or DynamoDB data stream with more than one Pipe execution simultaneously. You can specify the number of concurrent batches that EventBridge polls from a shard via a parallelization factor from 1 (default) to 10. For example, when you set ParallelizationFactor to 2, you can have 200 concurrent EventBridge Pipe executions at maximum to process 100 Kinesis data shards. This helps scale up the processing throughput when the data volume is volatile and the IteratorAge is high. Note that parallelization factor will not work if you are using Kinesis aggregation.

Polling and stream starting position

Be aware that stream source polling during pipe creation and updates is eventually consistent.

- During pipe creation, it may take several minutes to start polling events from the stream.
- During pipe updates to the source polling configuration, it may take several minutes to stop and restart polling events from the stream.

This means that if you specify LATEST as the starting position for the stream, the pipe could miss events sent during pipe creation or updates. To ensure no events are missed, specify the stream starting position as TRIM_HORIZON or AT_TIMESTAMP.

Reporting batch item failures

When EventBridge consumes and processes streaming data from an source, by default it checkpoints to the highest sequence number of a batch, but only when the batch is a complete success. To avoid
reprocessing successfully processed messages in a failed batch, you can configure your enrichment or target to return an object indicating which messages succeeded and which failed. This is called a partial batch response.

For more information, see ?? (p. 192).

Success and failure conditions

If you return any of the following, EventBridge treats a batch as a complete success:

- An empty batchItemFailure list
- A null batchItemFailure list
- An empty EventResponse
- A null EventResponse

If you return any of the following, EventBridge treats a batch as a complete failure:

- An empty string itemIdentifier
- A null itemIdentifier
- An itemIdentifier with a bad key name

EventBridge retries failures based on your retry strategy.

Amazon MQ message broker as a source

You can use EventBridge Pipes to receive records from an Amazon MQ message broker. You can then optionally filter or enhance these records before sending them to one of the available destinations for processing. There are settings specific to Amazon MQ that you can choose when setting up a pipe. EventBridge Pipes maintains the order of the records from the message broker when sending that data to the destination.

Amazon MQ is a managed message broker service for Apache ActiveMQ and RabbitMQ. A message broker enables software applications and components to communicate using different programming languages, operating systems, and formal messaging protocols with either topics or queues as event destinations.

Amazon MQ can also manage Amazon Elastic Compute Cloud (Amazon EC2) instances on your behalf by installing ActiveMQ or RabbitMQ brokers. After a broker is installed, it provides different network topologies and other infrastructure needs to your instances.

The Amazon MQ source has the following configuration restrictions:

- **Cross account** – EventBridge doesn’t support cross-account processing. You can’t use EventBridge to process records from an Amazon MQ message broker that is in a different AWS account.
- **Authentication** – For ActiveMQ, only the ActiveMQ SimpleAuthenticationPlugin is supported. For RabbitMQ, only the PLAIN authentication mechanism is supported. To manage credentials, use AWS Secrets Manager. For more information about ActiveMQ authentication, see Integrating ActiveMQ brokers with LDAP in the Amazon MQ Developer Guide.
- **Connection quota** – Brokers have a maximum number of allowed connections for each wire-level protocol. This quota is based on the broker instance type. For more information, see the Brokers section of *Quotas in Amazon MQ* in the Amazon MQ Developer Guide.
- **Connectivity** – You can create brokers in a public or private virtual private cloud (VPC). For private VPCs, your pipe needs access to the VPC to receive messages.
• **Event destinations** – Only queue destinations are supported. However, you can use a virtual topic, which behaves as both a topic internally and as a queue externally when it interacts with your pipes. For more information, see [Virtual Destinations](#) on the Apache ActiveMQ website, and [Virtual Hosts](#) on the RabbitMQ website.

• **Network topology** – For ActiveMQ, only one single-instance or standby broker is supported for pipe. For RabbitMQ, only one single-instance broker or cluster deployment is supported for each pipe. Single-instance brokers require a failover endpoint. For more information about these broker deployment modes, see [Active MQ Broker Architecture](#) and [Rabbit MQ Broker Architecture](#) in the Amazon MQ Developer Guide.

• **Protocols** – Supported protocols depend on the Amazon MQ integration that you use.
  - For ActiveMQ integrations, EventBridge uses the OpenWire/Java Message Service (JMS) protocol to consume messages. Message consumption isn’t supported on any other protocol. EventBridge only supports the `TextMessage` and `BytesMessage` operations within the JMS protocol. For more information about the OpenWire protocol, see [OpenWire](#) on the Apache ActiveMQ website.
  - For RabbitMQ integrations, EventBridge uses the AMQP 0-9-1 protocol to consume messages. No other protocols are supported for consuming messages. For more information about RabbitMQ’s implementation of the AMQP 0-9-1 protocol, see [AMQP 0-9-1 Complete Reference Guide](#) on the RabbitMQ website.

EventBridge automatically supports the latest versions of ActiveMQ and RabbitMQ that Amazon MQ supports. For the latest supported versions, see [Amazon MQ release notes](#) in the Amazon MQ Developer Guide.

**Note**

By default, Amazon MQ has a weekly maintenance window for brokers. During that window of time, brokers are unavailable. For brokers without standby, EventBridge won’t process messages until the window ends.

**Example events**

The following sample event shows the information that is received by the pipe. You can use this event to create and filter your event patterns, or to define input transformation. Not all of the fields can be filtered. For more information about which fields you can filter, see [???(p. 183)](#).

**ActiveMQ**

```
[
  {
    "eventSource": "aws:amq",
    "messageType": ".jms/text-message",
    "data": "QUJDOkFBQUE=",
    "connectionId": "myJMSCoID",
    "redelivered": false,
    "destination": {
      "physicalname": "testQueue"
    },
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959
  },
  {
    "eventSource": "aws:amq",
    "messageType": ".jms/text-message",
    "data": "QUJDOkFBQUE=",
    "connectionId": "myJMSCoID",
    "redelivered": false,
    "destination": {
      "physicalname": "testQueue"
    },
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959
  }
]
"messageType": "jms/bytes-message",
"data": "3D00W7cRj51prglqrVLaGQ82S48k",
"connectionId": "myJMSCoID1",
"persistent": false,
"destination": {
  "physicalname": "testQueue"
},
"timestamp": 1598827811958,
"brokerInTime": 1598827811958,
"brokerOutTime": 1598827811959
}
]

RabbitMQ

[

  {
    "eventSource": "aws:rmq",
    "eventSourceKey": "pizzaQueue::/",
    "basicProperties": {
      "contentType": "text/plain",
      "contentEncoding": null,
      "headers": {
        "header1": {
          "bytes": [118, 97, 108, 117, 101, 49]
        },
        "header2": {
          "bytes": [118, 97, 108, 117, 101, 50]
        },
        "numberInHeader": 10
      },
      "deliveryMode": 1,
      "priority": 34,
      "correlationId": null,
      "replyTo": null,
      "expiration": "60000",
      "messageId": null,
      "timestamp": "Jan 1, 1970, 12:33:41 AM",
      "type": null,
      "userId": "AIDACKCEVSQ6C2EXAMPLE",
      "appId": null,
      "clusterId": null,
      "bodySize": 80
    },
    "redelivered": false,
    "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEU1fQ=="
  }
]
Consumer group

To interact with Amazon MQ, EventBridge creates a consumer group that can read from your Amazon MQ brokers. The consumer group is created with the same ID as the pipe UUID.

For Amazon MQ sources, EventBridge batches records together and sends them to your function in a single payload. To control behavior, you can configure the batching window and batch size. EventBridge pulls messages until one of the following occurs:

- The processed records reach the payload size maximum of 6 MB.
- The batching window expires.
- The number of records reaches the full batch size.

EventBridge converts your batch into a single payload and then invokes your function. Messages aren't persisted or deserialized. Instead, the consumer group retrieves them as a BLOB of bytes. It then base64-encodes them into a JSON payload. If the pipe returns an error for any of the messages in a batch, EventBridge retries the entire batch of messages until processing succeeds or the messages expire.

Network configuration

By default, Amazon MQ brokers are created with the PubliclyAccessible flag set to false. It's only when PubliclyAccessible is set to true that the broker receives a public IP address. For full access with your pipe, your broker must either use a public endpoint or provide access to the VPC.

If your Amazon MQ broker isn't publicly accessible, EventBridge must have access to the Amazon Virtual Private Cloud (Amazon VPC) resources associated with your broker. To access the VPC of your Amazon MQ brokers, EventBridge requires outbound internet access for the subnets of your source. For public subnets this can be an internet gateway, for private subnets it needs to be a Network Address Translator (NAT), such as a NAT gateway, or your own NAT. Ensure that the NAT has a public IP address and can connect to the internet.

Configure your Amazon VPC security groups with the following rules (at minimum):

- Inbound rules – For a broker without public accessibility, allow all traffic on all ports for the security group that's specified as your source. For a broker with public accessibility, allow all traffic on all ports for all destinations.
- Outbound rules – Allow all traffic on all ports for all destinations.

Note

Your Amazon VPC configuration is discoverable through the Amazon MQ API. You don't need to configure it during setup.

Amazon Managed Streaming for Apache Kafka topic as a source

You can use EventBridge Pipes to receive records from an Amazon Managed Streaming for Apache Kafka (Amazon MSK) topic. You can optionally filter or enhance these records before sending them to one of the available destinations for processing. There are settings specific to Amazon MSK that you can choose when setting up a pipe. EventBridge Pipes maintains the order of the records from the message broker when sending that data to the destination.

Amazon MSK is a fully managed service that you can use to build and run applications that use Apache Kafka to process streaming data. Amazon MSK simplifies the setup, scaling, and management of clusters running Apache Kafka. With Amazon MSK, you can configure your application for multiple Availability
Zones and for security with AWS Identity and Access Management (IAM). Amazon MSK supports multiple open-source versions of Kafka.

Amazon MSK as an source operates similarly to using Amazon Simple Queue Service (Amazon SQS) or Amazon Kinesis. EventBridge internally polls for new messages from the source and then synchronously invokes the target. EventBridge reads the messages in batches and provides these to your function as an event payload. The maximum batch size is configurable. (The default is 100 messages.)

For Apache Kafka-based sources, EventBridge supports processing control parameters, such as batching windows and batch size.

EventBridge reads the messages sequentially for each partition. After EventBridge processes each batch, it commits the offsets of the messages in that batch. If the pipe's target returns an error for any of the messages in a batch, EventBridge retries the entire batch of messages until processing succeeds or the messages expire.

EventBridge sends the batch of messages in the event when it invokes the target. The event payload contains an array of messages. Each array item contains details of the Amazon MSK topic and partition identifier, together with a timestamp and a base64-encoded message.

**Example events**

The following sample event shows the information that is received by the pipe. You can use this event to create and filter your event patterns, or for to define input transformation. Not all of the fields can be filtered. For more information about which fields you can filter, see ??? (p. 183).

```
[
  {
    "eventSource": "aws:kafka",
    "eventSourceArn": "arn:aws:kafka:sa-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "topic": "mytopic",
    "partition": "0",
    "offset": 15,
    "timestamp": 1545084650987,
    "timestampType": "CREATE_TIME",
    "key": "abcDEFghiJKLmnoPQRstuVWXyz1234==",
    "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
    "headers": [
      {
        "headerKey": [104, 101, 97, 100, 101, 114, 86, 97, 108, 117, 101]
      }
    ]
  }
]
```

**Polling and stream starting position**

Be aware that stream source polling during pipe creation and updates is eventually consistent.
• During pipe creation, it may take several minutes to start polling events from the stream.
• During pipe updates to the source polling configuration, it may take several minutes to stop and
restart polling events from the stream.

This means that if you specify LATEST as the starting position for the stream, the pipe could miss
events sent during pipe creation or updates. To ensure no events are missed, specify the stream starting
position as TRIM_HORIZON.

MSK cluster authentication

EventBridge needs permission to access the Amazon MSK cluster, retrieve records, and perform other
tasks. Amazon MSK supports several options for controlling client access to the MSK cluster. For more
information about which authentication method is used when, see ?? (p. 175).

Cluster access options

• Unauthenticated access (p. 173)
• SASL/SCRAM authentication (p. 173)
• IAM role-based authentication (p. 173)
• Mutual TLS authentication (p. 174)
• Configuring the mTLS secret (p. 174)
• How EventBridge chooses a bootstrap broker (p. 175)

Unauthenticated access

We recommend only using unauthenticated access for development. Unauthenticated access will only
work if IAM role-based authentication is disabled for the cluster.

SASL/SCRAM authentication

Amazon MSK supports Simple Authentication and Security Layer/Salted Challenge Response
Authentication Mechanism (SASL/SCRAM) authentication with Transport Layer Security (TLS) encryption.
For EventBridge to connect to the cluster, you store the authentication credentials (sign-in credentials) in
an AWS Secrets Manager secret.

For more information about using Secrets Manager, see User name and password authentication with
AWS Secrets Manager in the Amazon Managed Streaming for Apache Kafka Developer Guide.

Amazon MSK doesn't support SASL/PLAIN authentication.

IAM role-based authentication

You can use IAM to authenticate the identity of clients that connect to the MSK cluster. If IAM
authentication is active on your MSK cluster, and you don't provide a secret for authentication,
EventBridge automatically defaults to using IAM authentication. To create and deploy IAM user or role-
based policies, use the IAM console or API. For more information, see IAM access control in the Amazon
Managed Streaming for Apache Kafka Developer Guide.

To allow EventBridge to connect to the MSK cluster, read records, and perform other required actions,
add the following permissions to your pipe's execution role.

```json
{
    "Version": "2012-10-17",
    "Statement": [

```
You can scope these permissions to a specific cluster, topic, and group. For more information, see the Amazon MSK Kafka actions in the Amazon Managed Streaming for Apache Kafka Developer Guide.

**Mutual TLS authentication**

Mutual TLS (mTLS) provides two-way authentication between the client and server. The client sends a certificate to the server for the server to verify the client, and the server sends a certificate to the client for the client to verify the server.

For Amazon MSK, EventBridge acts as the client. You configure a client certificate (as a secret in Secrets Manager) to authenticate EventBridge with the brokers in your MSK cluster. The client certificate must be signed by a certificate authority (CA) in the server's trust store. The MSK cluster sends a server certificate to EventBridge to authenticate the brokers with EventBridge. The server certificate must be signed by a CA that's in the AWS trust store.

Amazon MSK doesn't support self-signed server certificates, because all brokers in Amazon MSK use public certificates signed by Amazon Trust Services CAs, which EventBridge trusts by default.

For more information about mTLS for Amazon MSK, see Mutual TLS Authentication in the Amazon Managed Streaming for Apache Kafka Developer Guide.

**Configuring the mTLS secret**

The CLIENT_CERTIFICATE_TLS_AUTH secret requires a certificate field and a private key field. For an encrypted private key, the secret requires a private key password. Both the certificate and private key must be in PEM format.

**Note**

EventBridge supports the PBES1 (but not PBES2) private key encryption algorithms.

The certificate field must contain a list of certificates, beginning with the client certificate, followed by any intermediate certificates, and ending with the root certificate. Each certificate must start on a new line with the following structure:

```
-----BEGIN CERTIFICATE-----
<certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager supports secrets up to 65,536 bytes, which is enough space for long certificate chains.

The private key must be in PKCS #8 format, with the following structure:

```
-----BEGIN PRIVATE KEY-----
```
For an encrypted private key, use the following structure:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
<private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

The following example shows the contents of a secret for mTLS authentication using an encrypted private key. For an encrypted private key, you include the private key password in the secret.

```
{
    "privateKeyPassword": "testpassword",
    "certificate": "-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRzytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...j0Lh4/+1HfgyE2KlmII36dg4IMzNhJAFEBZiCRoP1m04/s1cRqFHXoa10QQbi1k
cmUuiAi1i9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uF9hbNC5RdfmHrzANBgqkhkiG9wBARQtsFADBb
...rOiowxbb5wXChYeYANQIfTZ6weQTgiCHCbuuMKNV6K5FkXm0vqVD/YpXk5A/no
c8PHPSOAaRWMGmASAZALvbRz8mgp==
-----END CERTIFICATE-----
"privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQQQwSDAnBgkqhkiG9w0BBQxgwGQUAFcK5HT/X7Kjmgp
...QzSekJQF+kWzmB6nAfSzg9IaoAaytVNgGTCkWeUkWm/V0Ck+LdGUxAC4RxZnoQ
zp2moJm2NvYB7AI7+1mpbazOZb+8YG2aUCiyq6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

How EventBridge chooses a bootstrap broker

EventBridge chooses a bootstrap broker based on the authentication methods available on your cluster, and whether you provide a secret for authentication. If you provide a secret for mTLS or SASL/SCRAM, EventBridge automatically chooses that authentication method. If you don't provide a secret, EventBridge chooses the strongest authentication method that's active on your cluster. The following is the order of priority in which EventBridge selects a broker, from strongest to weakest authentication:

- mTLS (secret provided for mTLS)
- SASL/SCRAM (secret provided for SASL/SCRAM)
- SASL IAM (no secret provided, and IAM authentication is active)
- Unauthenticated TLS (no secret provided, and IAM authentication is not active)
- Plaintext (no secret provided, and both IAM authentication and unauthenticated TLS are not active)

**Note**

If EventBridge can't connect to the most secure broker type, it doesn't attempt to connect to a different (weaker) broker type. If you want EventBridge to choose a weaker broker type, deactivate all stronger authentication methods on your cluster.

Network configuration

EventBridge must have access to the Amazon Virtual Private Cloud (Amazon VPC) resources associated with your Amazon MSK cluster. To access the VPC of your Amazon MSK cluster, EventBridge requires
outbound internet access for the subnets of your source. For public subnets this can be an internet gateway, for private subnets it needs to be a Network Address Translator (NAT), such as a NAT gateway, or your own NAT. Ensure that the NAT has a public IP address and can connect to the internet.

Configure your Amazon VPC security groups with the following rules (at minimum):

- **Inbound rules** – Allow all traffic on the Amazon MSK broker port (9092 for plaintext, 9094 for TLS, 9096 for SASL, 9098 for IAM) for the security groups specified for your source.
- **Outbound rules** – Allow all traffic on port 443 for all destinations. Allow all traffic on the Amazon MSK broker port (9092 for plaintext, 9094 for TLS, 9096 for SASL, 9098 for IAM) for the security groups specified for your source.

**Note**
Your Amazon VPC configuration is discoverable through the Amazon MSK API. You don't need to configure it during setup.

**Customizable consumer group ID**

When setting up Apache Kafka as an source, you can specify a consumer group ID. This consumer group ID is an existing identifier for the Apache Kafka consumer group that you want your pipe to join. You can use this feature to migrate any ongoing Apache Kafka record processing setups from other consumers to EventBridge.

If you specify a consumer group ID and there are other active pollers within that consumer group, Apache Kafka distributes messages across all consumers. In other words, EventBridge doesn't receive all messages for the Apache Kafka topic. If you want EventBridge to handle all messages in the topic, turn off any other pollers in that consumer group.

Additionally, if you specify a consumer group ID, and Apache Kafka finds a valid existing consumer group with the same ID, EventBridge ignores the StartingPosition parameter for your pipe. Instead, EventBridge begins processing records according to the committed offset of the consumer group. If you specify a consumer group ID, and Apache Kafka can't find an existing consumer group, then EventBridge configures your source with the specified StartingPosition.

The consumer group ID that you specify must be unique among all your Apache Kafka event sources. After creating a pipe with the consumer group ID specified, you can't update this value.

**Auto scaling of the Amazon MSK source**

When you initially create an Amazon MSK source, EventBridge allocates one consumer to process all partitions in the Apache Kafka topic. Each consumer has multiple processors running in parallel to handle increased workloads. Additionally, EventBridge automatically scales up or down the number of consumers, based on workload. To preserve message ordering in each partition, the maximum number of consumers is one consumer per partition in the topic.

In one-minute intervals, EventBridge evaluates the consumer offset lag of all the partitions in the topic. If the lag is too high, the partition is receiving messages faster than EventBridge can process them. If necessary, EventBridge adds or removes consumers from the topic. The scaling process of adding or removing consumers occurs within three minutes of evaluation.

If your target is overloaded, EventBridge reduces the number of consumers. This action reduces the workload on the pipe by reducing the number of messages that consumers can retrieve and send to the pipe.

**Self managed Apache Kafka stream as a source**

You can use EventBridge Pipes to receive records from self managed Apache Kafka. You can then optionally filter or enhance these records before sending them to an available destination for processing.
There are source-specific settings that you can choose when setting up the pipe. EventBridge Pipes maintains the order of the records received from the cluster when it sends that data to the destination.

Apache Kafka is an open-source event streaming platform that supports workloads such as data pipelines and streaming analytics. You can use the AWS managed Apache Kafka service Amazon Managed Streaming for Apache Kafka (Amazon MSK), or a self managed Apache Kafka cluster.

This topic describes how to use EventBridge Pipes with a self managed Apache Kafka cluster. In AWS terminology, a self-managed cluster includes non-AWS hosted Apache Kafka clusters. For example, you can host your Apache Kafka cluster with a cloud provider such as CloudKarafka. You can also use other AWS hosting options for your cluster. For more information, see Best Practices for Running Apache Kafka on AWS on the AWS Big Data Blog.

Apache Kafka as a source operates similarly to using Amazon Simple Queue Service (Amazon SQS) or Amazon Kinesis. EventBridge internally polls for new messages from the source and then synchronously invokes the target. EventBridge reads the messages in batches and provides these to your function as an event payload. The maximum batch size is configurable. (The default is 100 messages.)

For Apache Kafka-based sources, EventBridge supports processing control parameters, such as batching windows and batch size.

EventBridge sends the batch of messages in the event parameter when it invokes your pipe. The event payload contains an array of messages. Each array item contains details of the Apache Kafka topic and Apache Kafka partition identifier, together with a timestamp and a base64-encoded message.

**Example events**

The following sample event shows the information that is received by the pipe. You can use this event to create and filter your event patterns, or to define input transformation. Not all of the fields can be filtered. For more information about which fields you can filter, see ??? [p. 183].

```json
{
"eventSource": "SelfManagedKafka",
"bootstrapServers": "b-2.demo-cluster-1.albcede.cl1.kafka.us-east-1.amazonaws.com:9092,b-1.demo-cluster-1.albcede.cl1.kafka.us-east-1.amazonaws.com:9092",
"eventSourceKey": "mytopic-0",
"topic": "mytopic",
"partition": 0,
"offset": 15,
"timestamp": 1545084650987,
"timestampType": "CREATE_TIME",
"key": "abcDEFGhijklmnoPQRstuWxYz1234==",
"value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
"headers": [
  {
    "headerKey": [
      104,
      101,
      97,
      100,
      101,
      114,
      86,
      97,
      108,
      117,
      101
    ]
  }
]
}
```
Apache Kafka cluster authentication

EventBridge Pipes supports several methods to authenticate with your self managed Apache Kafka cluster. Make sure that you configure the Apache Kafka cluster to use one of these supported authentication methods. For more information about Apache Kafka security, see the Security section of the Apache Kafka documentation.

VPC access

If only Apache Kafka users within your VPC access your Apache Kafka brokers, you must configure the Apache Kafka source for Amazon Virtual Private Cloud (Amazon VPC) access.

SASL/SCRAM authentication


EventBridge Pipes supports SASL/PLAIN authentication with TLS encryption. With SASL/PLAIN authentication, EventBridge Pipes sends credentials as clear text (unencrypted) to the server.

For SASL authentication, you store the sign-in credentials as a secret in AWS Secrets Manager.

Mutual TLS authentication

Mutual TLS (mTLS) provides two-way authentication between the client and server. The client sends a certificate to the server for the server to verify the client, and the server sends a certificate to the client for the client to verify the server.

In self managed Apache Kafka, EventBridge Pipes acts as the client. You configure a client certificate (as a secret in Secrets Manager) to authenticate EventBridge Pipes with your Apache Kafka brokers. The client certificate must be signed by a certificate authority (CA) in the server's trust store.

The Apache Kafka cluster sends a server certificate to EventBridge Pipes to authenticate the Apache Kafka brokers with EventBridge Pipes. The server certificate can be a public CA certificate or a private CA/self-signed certificate. The public CA certificate must be signed by a CA that's in the EventBridge Pipes trust store. For a private CA/self-signed certificate, you configure the server root CA certificate (as a secret in Secrets Manager). EventBridge Pipes uses the root certificate to verify the Apache Kafka brokers.

For more information about mTLS, see Introducing mutual TLS authentication for Amazon MSK as an source.

Configuring the client certificate secret

The CLIENT_CERTIFICATE_TLS_AUTH secret requires a certificate field and a private key field. For an encrypted private key, the secret requires a private key password. Both the certificate and private key must be in PEM format.

Note

EventBridge Pipes supports the PBES1 (but not PBES2) private key encryption algorithms.

The certificate field must contain a list of certificates, beginning with the client certificate, followed by any intermediate certificates, and ending with the root certificate. Each certificate must start on a new line with the following structure:
Secrets Manager supports secrets up to 65,536 bytes, which is enough space for long certificate chains.

The private key must be in [PKCS #8](https://en.wikipedia.org/wiki/PKCS#8) format, with the following structure:

```
-----BEGIN PRIVATE KEY-----
<private key contents>
-----END PRIVATE KEY-----
```

For an encrypted private key, use the following structure:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
<private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

The following example shows the contents of a secret for mTLS authentication using an encrypted private key. For an encrypted private key, include the private key password in the secret.

```
{
"privateKeyPassword": "testpassword",
"certificate": "-----BEGIN CERTIFICATE-----
MIIEEQCCAgAwIBAgIRAPJdwaFaNRztytHBto0j6BAM0vDQYJKoZIhvcNAQELBQAw
...j0Lh4/+/1HFgyE2KlmT16d4IMzNhAFEBZlCRoPim048s1cRqtFHXoa10Q0bIl+xk
cmUuiAi9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCjCA2qAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...zQoiowbb5wXCheYSANQIfTZ6weQTgqICHCCbuumK6V95FkXm0vqVD/YpKwa/no
c8PH5PSoA8wMVgDDAA2ALJvbRz8mpg==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFKzBVBgkqhkiG9w0BBQowSDAnBgkqhkiG9w0BBQwwGgQUiAFcK8hT/X7Kjmgp
 ...
Q5SeqF+kwZm6B6nAfSzsg09lAoAaytYTvNgTckWeUKwNv/V0ck+kDGUZxAC4RxZnQoQ
zp2mzJn2NBY7A7z+1mp0azDz+BYG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

**Configuring the server root CA certificate secret**

You create this secret if your Apache Kafka brokers use TLS encryption with certificates signed by a private CA. You can use TLS encryption for VPC, SASL/SCRAM, SASL/PLAIN, or mTLS authentication.

The server root CA certificate secret requires a field that contains the Apache Kafka broker’s root CA certificate in PEM format. The following example shows the structure of the secret.

```
{
"certificate": "-----BEGIN CERTIFICATE-----
MIIDrzCAYgAwIBAgIBDAANBgkqhkiG9w0BAQsFADCBlMzBMDgNCgYGB0MBAAEDwQ
...Ez1BvqbwEwUv6Fgs+c+R+j8/wUIuo6Rw/m/1x3eZz0xQj8w8wGgQUiAFcK8hT/X7Kjmgp
 ...
Q5SeqF+kwZm6B6nAfSzsg09lAoAaytYTvNgTckWeUKwNv/V0ck+kDGUZxAC4RxZnQoQ
zp2mzJn2NBY7A7z+1mp0azDz+BYG2aUCiyqb6PnnA==
-----END CERTIFICATE-----"
```
Network configuration

EventBridge must have access to the Amazon Virtual Private Cloud (Amazon VPC) resources associated with your Apache Kafka brokers. To access the VPC of your Apache Kafka cluster, EventBridge requires outbound internet access for the subnets of your source. For public subnets this can be an internet gateway, for private subnets it needs to be a Network Address Translator (NAT), such as a NAT gateway, or your own NAT. Ensure that the NAT has a public IP address and can connect to the internet.

Configure your Amazon VPC security groups with the following rules (at minimum):

- Inbound rules – Allow all traffic on the Apache Kafka broker port (9092 for plaintext, 9094 for TLS, 9096 for SASL, 9098 for IAM) for the security groups specified for your source.
- Outbound rules – Allow all traffic on port 443 for all destinations. Allow all traffic on the Apache Kafka broker port (9092 for plaintext, 9094 for TLS, 9096 for SASL, 9098 for IAM) for the security groups specified for your source.

Auto scaling of the Apache Kafka source

When you initially create an Apache Kafka source, EventBridge allocates one consumer to process all partitions in the Kafka topic. Each consumer has multiple processors running in parallel to handle increased workloads. Additionally, EventBridge automatically scales up or down the number of consumers, based on workload. To preserve message ordering in each partition, the maximum number of consumers is one consumer per partition in the topic.

In one-minute intervals, EventBridge evaluates the consumer offset lag of all the partitions in the topic. If the lag is too high, the partition is receiving messages faster than EventBridge can process them. If necessary, EventBridge adds or removes consumers from the topic. The scaling process of adding or removing consumers occurs within three minutes of evaluation.

If your target is overloaded, EventBridge reduces the number of consumers. This action reduces the workload on the function by reducing the number of messages that consumers can retrieve and send to the function.

Amazon Simple Queue Service as a source

You can use EventBridge Pipes to receive records from an Amazon SQS queue. You can then optionally filter or enhance these records before sending them to an available destination for processing.

You can use a pipe to process messages in an Amazon Simple Queue Service (Amazon SQS) queue. EventBridge Pipes support standard queues and first-in, first-out (FIFO) queues. With Amazon SQS, you can offload tasks from one component of your application by sending them to a queue and processing them asynchronously.

EventBridge polls the queue and invokes your pipe synchronously with an event that contains queue messages. EventBridge reads messages in batches and invokes your pipe once for each batch. When your pipe successfully processes a batch, EventBridge deletes its messages from the queue.

By default, EventBridge polls up to 10 messages in your queue simultaneously and sends that batch to your pipe. To avoid invoking the pipe with a small number of records, you can tell the event source to buffer records for up to five minutes by configuring a batch window. Before invoking the pipe, EventBridge continues to poll messages from the Amazon SQS standard queue until one of these things occurs:

- The batch window expires.
- The invocation payload size quota is reached.
- The configured maximum batch size is reached.
Note
If you're using a batch window and your Amazon SQS queue contains low traffic, EventBridge might wait for up to 20 seconds before invoking your pipe. This is true even if you set a batch window for fewer than 20 seconds. For FIFO queues, records contain additional attributes that are related to deduplication and sequencing.

When EventBridge reads a batch, the messages stay in the queue but are hidden for the length of the queue's visibility timeout. If your pipe successfully processes the batch, EventBridge deletes the messages from the queue. By default, if your pipe encounters an error while processing a batch, all messages in that batch become visible in the queue again. For this reason, your pipe code must be able to process the same message multiple times without unintended side effects. You can modify this reprocessing behavior by including batch item failures in your pipe response. The following example shows an event for a batch of two messages.

Example events
The following sample event shows the information that is received by the pipe. You can use this event to create and filter your event patterns, or to define input transformation. Not all of the fields can be filtered. For more information about which fields you can filter, see ???(p. 183).

Standard queue

```json
[
  {
    "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
    "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgx1aS3SLy0a...",
    "body": "Test message.",
    "attributes": {
      "ApproximateReceiveCount": "1",
      "SentTimestamp": "1545082649183",
      "SenderId": "AIDAIENQZJ0L23YV34VO",
      "ApproximateFirstReceiveTimestamp": "1545082649185"
    },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "awsRegion": "us-east-2"
  },
  {
    "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
    "receiptHandle": "AQEBzWwaftRI0KuVm4tP+7q1rGnQ1cHq...",
    "body": "Test message.",
    "attributes": {
      "ApproximateReceiveCount": "1",
      "SentTimestamp": "1545082650636",
      "SenderId": "AIDAIENQZJ0L23YV34VO",
      "ApproximateFirstReceiveTimestamp": "1545082650649"
    },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "awsRegion": "us-east-2"
  }
]
```

FIFO queue

```json
[
  {
    "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
```
Scaling and processing

For standard queues, EventBridge uses long polling to poll a queue until it becomes active. When messages are available, EventBridge reads up to five batches and sends them to your pipe. If messages are still available, EventBridge increases the number of processes that are reading batches by up to 60 more instances per minute. The maximum number of batches that a pipe can process simultaneously is 1,000.

For FIFO queues, EventBridge sends messages to your pipe in the order that it receives them. When you send a message to a FIFO queue, you specify a message group ID. Amazon SQS facilitates delivering messages in the same group to EventBridge, in order. EventBridge sorts the received messages into groups and sends only one batch at a time for a group. If your pipe returns an error, the pipe attempts all retries on the affected messages before EventBridge receives additional messages from the same group.

Configuring a queue to use with EventBridge Pipes

Create an Amazon SQS queue to serve as an source for your pipe. Then configure the queue to allow time for your pipe to process each batch of events—and for EventBridge to retry in response to throttling errors as it scales up.

To allow your pipe time to process each batch of records, set the source queue's visibility timeout to at least six times the combined runtime of the pipe enrichment and target components. The extra time allows for EventBridge to retry if your pipe is throttled while processing a previous batch.

If your pipe fails to process a message multiple times, Amazon SQS can send it to a dead-letter queue. When your pipe returns an error, EventBridge keeps it in the queue. After the visibility timeout occurs, EventBridge receives the message again. To send messages to a second queue after a number of receives, configure a dead-letter queue on your source queue.

**Note**

Make sure that you configure the dead-letter queue on the source queue, not on the pipe. The dead-letter queue that you configure on a pipe is used for the pipe's asynchronous invocation queue, not for source queues.

If your pipe returns an error, or can't be invoked because it's at maximum concurrency, processing might succeed with additional attempts. To give messages more chances to be processed before sending them to the dead-letter queue, set the `maxReceiveCount` on the source queue's re overrun policy to at least 5.

Reporting batch item failures

When EventBridge consumes and processes streaming data from an source, by default it checkpoints to the highest sequence number of a batch, but only when the batch is a complete success. To avoid
reprocessing successfully processed messages in a failed batch, you can configure your enrichment or
target to return an object indicating which messages succeeded and which failed. This is called a partial
batch response.

For more information, see ?? (p. 192).

Success and failure conditions

If you return any of the following, EventBridge treats a batch as a complete success:

• An empty batchItemFailure list
• A null batchItemFailure list
• An empty EventResponse
• A null EventResponse

If you return any of the following, EventBridge treats a batch as a complete failure:

• An empty string itemIdentifier
• A null itemIdentifier
• An itemIdentifier with a bad key name

EventBridge retries failures based on your retry strategy.

Amazon EventBridge Pipes filtering

With EventBridge Pipes, you can filter a given source's events and process only a subset of them.
This filtering works in the same way as filtering on an EventBridge event bus or Lambda event source
mapping, by using event patterns. For more information about event patterns, see ?? (p. 62).

A filter criteria FilterCriteria object is a structure that consists of a list of filters (Filters). Each
filter is a structure that defines an filtering pattern (Pattern). A Pattern is a string representation of a
JSON filter rule. A FilterCriteria object looks like the following example:

```json
{
  "Filters": [
    {
      "Pattern": "{ "Metadata1": [ rule1 ], "data": { "Data1": [ rule2 ] }}"
    }
  ]
}
```

For added clarity, here is the value of the filter's Pattern expanded in plain JSON:

```json
{
  "Metadata1": [ pattern1 ],
  "data": {"Data1": [ pattern2 ]}
}
```

The main parts to a FilterCriteria object are metadata properties and data properties.

• **Metadata properties** are the fields of the event object. In the example, FilterCriteria.Metadata1 refers to a metadata property.

• **Data properties** are the fields of the event body. In the example, FilterCriteria.Data1 refers to a data property.
Filtering

For example, suppose your Kinesis stream contains an event like this:

```json
{
    "kinesisSchemaVersion": "1.0",
    "partitionKey": "1",
    "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
    "data": {
        "City": "Seattle",
        "State": "WA",
        "Temperature": "46",
        "Month": "December"
    },
    "approximateArrivalTimestamp": 1545084650.987
}
```

When the event flows through your pipe, it'll look like the following with the `data` field base64-encoded:

```json
{
    "kinesisSchemaVersion": "1.0",
    "partitionKey": "1",
    "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
    "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
    "approximateArrivalTimestamp": 1545084650.987
    "eventSource": "aws:kinesis",
    "eventVersion": "1.0",
    "eventID": "shardId-00000000006:49590338271490256608559692538361571095921575989136588898",
    "eventName": "aws:kinesis:record",
    "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
    "awsRegion": "us-east-2",
    "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
}
```

The **metadata properties** on the Kinesis event are any field outside of the `data` object, such as `partitionKey` or `sequenceNumber`.

The **data properties** of the Kinesis event are the fields inside the `data` object, such as `City` or `Temperature`.

When you filter to match this event, you can use filters on the decoded fields. For example, to filter on `partitionKey` and `City` you'd use the following filter:

```json
{  "partitionKey": [  "1"  ],  "data": {   "City": [  "Seattle"  ]  }
}
```

When you're creating event filters, EventBridge Pipes can access event content. This content is either JSON-escaped, such as the Amazon SQS body field, or base64-encoded, such as the Kinesis `data` field. If your data is valid JSON, your input templates or JSON paths for target parameters can reference the content directly. For example, if a Kinesis event source is valid JSON, you can reference a variable using `<$.data.someKey>`.

When creating event patterns, you can filter based on the fields sent by the source API, and not on fields added by the polling operation. The following fields can't be used in event patterns:

- `awsRegion`
- `eventSource`
- `eventSourceARN`
- `eventVersion`
- `eventID`
Message and data fields

Every EventBridge Pipe source contains a field which contains the core message or data. We refer to these as **message** fields or **data** fields. These fields are special because they may be JSON-escaped or base64-encoded, but when they are valid JSON they can be filtered with JSON patterns as if the body was not escaped. The contents of these fields can also be used in input transformers (p. 193) seamlessly.

Properly filtering Amazon SQS messages

If an Amazon SQS message doesn't satisfy your filter criteria, EventBridge automatically removes the message from the queue. You don't have to delete these messages manually in Amazon SQS.

For Amazon SQS, the message body can be any string. However, this can be problematic if your FilterCriteria expects body to be in a valid JSON format. The reverse scenario is also true — if the incoming message body is in a valid JSON format, but your filter criteria expects body to be a plain string, it lead to unintended behavior.

To avoid this issue, make sure that the format of body in your FilterCriteria matches the expected format of body in messages that you receive from your queue. Before filtering your messages, EventBridge automatically evaluates the format of the incoming message body and of your filter pattern for body. If there is a mismatch, EventBridge drops the message. The following table summarizes this evaluation:

<table>
<thead>
<tr>
<th>Incoming message body format</th>
<th>Filter pattern body format</th>
<th>Resulting action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain string</td>
<td>Plain string</td>
<td>EventBridge filters based on your filter criteria.</td>
</tr>
<tr>
<td>Plain string</td>
<td>No filter pattern for data properties</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Plain string</td>
<td>Valid JSON</td>
<td>EventBridge drops the message.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>Plain string</td>
<td>EventBridge drops the message.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>No filter pattern for data properties</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>Valid JSON</td>
<td>EventBridge filters based on your filter criteria.</td>
</tr>
</tbody>
</table>

If you don't include body as part of your FilterCriteria, EventBridge skips this check.

Properly filtering Kinesis and DynamoDB messages

After your filter criteria processes a Kinesis or DynamoDB record, the streams iterator advances past this record. If the record doesn't satisfy your filter criteria, you don't have to delete the record manually from
your event source. After the retention period, Kinesis and DynamoDB automatically delete these old records. If you want records to be deleted sooner, see Changing the Data Retention Period.

To properly filter events from stream event sources, both the data field and your filter criteria for the data field must be in valid JSON format. (For Kinesis, the data field is data. For DynamoDB, the data field is dynamodb.) If either field isn't in a valid JSON format, EventBridge drops the message or throws an exception. The following table summarizes the specific behavior:

<table>
<thead>
<tr>
<th>Incoming data format (data or dynamodb)</th>
<th>Filter pattern format for data properties</th>
<th>Resulting action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid JSON</td>
<td>Valid JSON</td>
<td>EventBridge filters based on your filter criteria.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>No filter pattern for data properties</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>Non-JSON</td>
<td>EventBridge throws an exception at the time of the pipe or update. The filter pattern for data properties must be in a valid JSON format.</td>
</tr>
<tr>
<td>Non-JSON</td>
<td>Valid JSON</td>
<td>EventBridge drops the record.</td>
</tr>
<tr>
<td>Non-JSON</td>
<td>No filter pattern for data properties</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Non-JSON</td>
<td>Non-JSON</td>
<td>EventBridge throws an exception at the time of the pipe creation or update. The filter pattern for data properties must be in a valid JSON format.</td>
</tr>
</tbody>
</table>

Properly filtering Amazon Managed Streaming for Apache Kafka, self managed Apache Kafka, and Amazon MQ messages

**Note**

After you attach filter criteria to a pipe with an Apache Kafka or Amazon MQ event source, it can take up to 15 minutes to apply your filtering rules to events.

For Amazon MQ sources (p. 168), the message field is data. For Apache Kafka sources (Amazon MSK (p. 171) and self managed Apache Kafka (p. 176)), there are two message fields: key and value.

EventBridge drops messages that don't match all fields included in the filter. For Apache Kafka, EventBridge commits offsets for matched and unmatched messages after successfully invoking the function. For Amazon MQ, EventBridge acknowledges matched messages after successfully invoking the function and acknowledges unmatched messages when filtering them.

Apache Kafka and Amazon MQ messages must be UTF-8 encoded strings, either plain strings or in JSON format. That's because EventBridge decodes Apache Kafka and Amazon MQ byte arrays into UTF-8 before applying filter criteria. If your messages use another encoding, such as UTF-16 or ASCII, or if the
message format doesn't match the FilterCriteria format, EventBridge processes metadata filters only. The following table summarizes the specific behavior:

<table>
<thead>
<tr>
<th>Incoming message format (data or key and value)</th>
<th>Filter pattern format for message properties</th>
<th>Resulting action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain string</td>
<td>Plain string</td>
<td>EventBridge filters based on your filter criteria.</td>
</tr>
<tr>
<td>Plain string</td>
<td>No filter pattern for data properties</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Plain string</td>
<td>Valid JSON</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>Plain string</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>No filter pattern for data properties</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
<tr>
<td>Valid JSON</td>
<td>Valid JSON</td>
<td>EventBridge filters based on your filter criteria.</td>
</tr>
<tr>
<td>Non-UTF-8 encoded string</td>
<td>JSON, plain string, or no pattern</td>
<td>EventBridge filters (on the other metadata properties only) based on your filter criteria.</td>
</tr>
</tbody>
</table>

**Differences between Lambda ESM and EventBridge Pipes**

When filtering events, Lambda ESM and EventBridge Pipes operate generally the same way. The main difference is that eventSourceKey field isn't present in ESM payloads.

**Amazon EventBridge Pipes event enrichment**

With the enrichment step of EventBridge Pipes, you can enhance the data from the source before sending it to the target. For example, you might receive Ticket created events that don't include the full ticket data. Using enrichment, you can have a Lambda function call the get-ticket API for the full ticket details. Pipes can then send that information to a target (p. 188).

You can configure the following enrichments when setting up a pipe in EventBridge:

- API destination
- Amazon API Gateway
- Lambda function
- Step Functions state machine

**Note**

EventBridge Pipes only supports Express workflows as enrichments.
EventBridge invokes enrichments synchronously because it must wait for a response from the enrichment before invoking the target.

Enrichment responses are limited to a maximum size of 6MB.

You can also transform the data you receive from the source before sending it for enhancement. For more information, see (p. 193).

Filtering events using enrichment

EventBridge Pipes passes the enrichment responses directly to the configured target. This includes array responses for targets that support batches. For more information about batch behavior, see (p. 191). You can also use your enrichment as a filter and pass fewer events than were received from the source. If you don’t want to invoke the target, return an empty response, such as "", {}, or [].

Note
If you want to invoke the target with an empty payload, return an array with empty JSON [[]].

Invoking enrichments

EventBridge invokes enrichments synchronously (invocation type set to REQUEST_RESPONSE) because it must wait for a response from the enrichment before invoking the target.

Note
For Step Functions state machines, EventBridge only supports Express workflows as enrichments, because they can be invoked synchronously.

Amazon EventBridge Pipes targets

You can send data in your pipe to a specific target. You can configure the following targets when setting up a pipe in EventBridge:

- API destination (p. 117)
- API Gateway (p. 130)
- Batch job queue (p. 190)
- CloudWatch log group (p. 190)
- ECS task (p. 190)
- Event bus in the same account and Region
- Firehose delivery stream
- Inspector assessment template
- Kinesis stream
- Lambda function (SYNC or ASYNC) (p. 190)
- Redshift cluster data API queries
- SageMaker Pipeline
- Amazon SNS topic (SNS FIFO topics not supported)
- Amazon SQS queue
- Step Functions state machine (p. 190)
- Express workflows (SYNC or ASYNC)
- Standard workflows (ASYNC)
Target parameters

Some target services don't send the event payload to the target, instead, they treat the event as a trigger for invoking a specific API. EventBridge uses the PipeTargetParameters to specify what information gets sent to that API. These include the following:

- API destinations (The data sent to an API destination must match the structure of the API. You must use the InputTemplate object to make sure the data is structured correctly. If you want to include the original event payload, reference it in the InputTemplate.)
- API Gateway (The data sent to API Gateway must match the structure of the API. You must use the InputTemplate object to make sure the data is structured correctly. If you want to include the original event payload, reference it in the InputTemplate.)
- PipeTargetRedshiftDataParameters (Amazon Redshift Data API clusters)
- PipeTargetSageMakerPipelineParameters (Amazon SageMaker Runtime Model Building Pipelines)
- PipeTargetBatchJobParameters (AWS Batch)

Note
EventBridge does not support all JSON Path syntax and evaluate it at runtime. Supported syntax includes:

- dot notation (for example, $.detail)
- dashes
- underscores
- alphanumeric characters
- array indices
- wildcards (*)

Dynamic path parameters

EventBridge Pipes target parameters support optional dynamic JSON path syntax. You can use this syntax to specify JSON paths instead of static values (for example $.detail.state). The entire value has to be a JSON path, not only part of it. For example, RedshiftParameters.Sql can be $.detail.state but it can't be "SELECT * FROM $.detail.state". These paths are replaced dynamically at runtime with data from the event payload itself at the specified path. Dynamic path parameters can't reference new or transformed values resulting from input transformation. The supported syntax for dynamic parameter JSON paths is the same as when transforming input. For more information, see ?? (p. 193).

Dynamic syntax can be used on all string, non-enum fields of all EventBridge Pipes enrichment and target parameters except:

- PipeTargetCloudWatchLogsParameters.LogStreamName
- PipeTargetEventBridgeEventBusParameters.EndpointId
- PipeEnrichmentHttpParameters.HeaderParameters
- PipeTargetHttpParameters.HeaderParameters

For example, to set the PartitionKey of a pipe Kinesis target to a custom key from your source event, set the KinesisTargetParameter.PartitionKey to:

- "$.data.someKey" for a Kinesis source
Permissions

To make API calls on the resources that you own, EventBridge Pipes needs appropriate permission. EventBridge Pipes uses the IAM role that you specify on the pipe for enrichment and target calls using the IAM principal pipes.amazonaws.com.

Invoking targets

EventBridge has the following ways to invoke a target:

- **Synchronously** (invocation type set to REQUEST_RESPONSE) – EventBridge waits for a response from the target before proceeding.
- **Asynchronously** (invocation type set to FIRE_AND_FORGET) – EventBridge doesn't wait for a response before proceeding.

By default, for pipes with ordered sources, EventBridge invokes targets synchronously because a response from the target is needed before proceeding to the next event.

If an source doesn't enforce order, such as a standard Amazon SQS queue, EventBridge can invoke a supported target synchronously or asynchronously.

With Lambda functions and Step Functions state machines, you can configure the invocation type.

**Note**
For Step Functions state machines, **Standard workflows** must be invoked asynchronously.

EventBridge Pipes target specifics

AWS Batch job queues

All AWS Batch submitJob parameters are configured explicitly with BatchParameters, and as with all Pipe parameters, these can be dynamic using a JSON path to your incoming event payload.

CloudWatch Logs group

Whether you use an input transformer or not, the event payload is used as the log message. You can set the Timestamp (or the explicit LogStreamName of your destination) through CloudWatchLogsParameters in PipeTarget. As with all pipe parameters, these parameters can be dynamic when using a JSON path to your incoming event payload.

Amazon ECS task

All Amazon ECS runTask parameters are configured explicitly through EcsParameters. As with all pipe parameters, these parameters can be dynamic when using a JSON path to your incoming event payload.

Lambda functions and Step Functions workflows

Lambda and Step Functions do not have a batch API. To process batches of events from a pipe source, the batch is converted to a JSON array and passed to as input to the Lambda or Step Functions target. For more information, see [?? (p. 191)].
Amazon EventBridge Pipes batching and concurrency

Batching behavior

EventBridge Pipes supports batching from the source and to targets that support it. In addition, batching to enrichment is supported for AWS Lambda and AWS Step Functions. Because different services support different levels of batching, you can’t configure a pipe with a larger batch size than the target supports. For example, Amazon Kinesis stream sources support a maximum batch size of 10,000 records, but Amazon Simple Queue Service supports a maximum of 10 messages per batch as a target. Therefore, a pipe from a Kinesis stream to an Amazon SQS queue can have a maximum configured batch size on the source of 10.

If you configure a pipe with an enrichment or target that doesn't support batching, you won't be able to activate batching on the source.

When batching is activated on the source, arrays of JSON records are passed through the pipe and then mapped to the batch API of a supported enrichment or target. Input transformers (p. 193) are applied separately on each individual JSON record in the array, not the array as a whole. For examples of these arrays, see ??? (p. 163) and select a specific source. Pipes will use the batch API for the supported enrichment or target even if the batch size is 1. If the enrichment or target doesn’t have a batch API but receives full JSON payloads, such as Lambda and Step Functions, the entire JSON array is sent in one request. The request will be sent as a JSON array even if the batch size is 1.

If a pipe is configured for batching at the source, and the target supports batching, you can return an array of JSON items from your enrichment. This array can include a shorter or longer array than the original source. However, if the array is larger than the batch size supported by the target, the pipe won’t invoke the target.

Supported batchable targets

<table>
<thead>
<tr>
<th>Target</th>
<th>Maximum batch size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudWatch Logs</td>
<td>10,000</td>
</tr>
<tr>
<td>EventBridge event bus</td>
<td>10</td>
</tr>
<tr>
<td>Kinesis Data Firehose stream</td>
<td>500</td>
</tr>
<tr>
<td>Kinesis stream</td>
<td>500</td>
</tr>
<tr>
<td>Lambda function</td>
<td>customer defined</td>
</tr>
<tr>
<td>Step Functions state machine</td>
<td>customer defined</td>
</tr>
<tr>
<td>Amazon SNS topic</td>
<td>10</td>
</tr>
<tr>
<td>Amazon SQS queue</td>
<td>10</td>
</tr>
</tbody>
</table>

The following enrichments and targets receive the full batch event payload for processing and are constrained by the total payload size of the event, rather than the size of the batch:

- Step Functions state machine (262144 characters)
- Lambda function (6MB)
Partial batch failure

For Amazon SQS and stream sources, such as Kinesis and DynamoDB, EventBridge Pipes supports partial batch failure handling of target failures. If the target supports batching and only part of the batch succeeds, EventBridge automatically retries batching the remainder of the payload. For the most up-to-date enriched content, this retry occurs through the entire pipe, including re-invoking any configured enrichment.

Partial batch failure handling of the enrichment is not supported.

For Lambda and Step Functions targets, you can also specify a partial failure by returning a payload with defined structure from the target. This indicates events that need to be retried.

Example partial failure payload structure

```json
{
  "batchItemFailures": [
    {
      "itemIdentifier": "id2"
    },
    {
      "itemIdentifier": "id4"
    }
  ]
}
```

In the example, the itemIdentifier match the ID of the events handled by your target from their original source. For Amazon SQS, this is the messageId. For Kinesis and DynamoDB, this is the eventID. For EventBridge Pipes to adequately handle partial batch failures from the targets, these fields need to be included in any array payload returned by the enrichment.

Throughput and concurrency behavior

Every event or batch of events received by a pipe that travel to an enrichment or target is considered a pipe execution. A pipe in STARTED state continuously polls for events from the source, scaling up and down depending on the available backlog and configured batching settings.

For quotas on concurrent pipe executions, and number of pipes per account and Region, see ??? (p. 452).

By default, a single pipe will scale to the following maximum concurrent executions, depending on the source:

- **DynamoDB** – The concurrent executions can climb as high as the ParallelizationFactor configured on the pipe multiplied by the number of shards in the stream.
- **Apache Kafka** – The concurrent executions can climb as high the number of partitions on the topic, up to 1000.
- **Kinesis** – The concurrent executions can climb as high as the ParallelizationFactor configured on the pipe multiplied by the number of shards in the stream.
- **Amazon MQ** – 5
- **Amazon SQS** – 1250

If you have requirements for higher maximum polling throughputs or concurrency limits, contact support.

**Note**

The execution limits are considered best-effort safety limitations. Although polling isn't throttled below these values, a pipe or account might burst higher than these recommend values.
Pipe executions are limited to a maximum of 5 minutes including the enrichment and target processing. This limit currently can't be increased.

Pipes with strictly ordered sources, such as Amazon SQS FIFO queues, Kinesis and DynamoDB Streams, or Apache Kafka topics are further limited in concurrency by the configuration of the source, such as the number of message group IDs for FIFO queues or the number of shards for Kinesis queues. Because ordering is strictly guaranteed within these constraints, a pipe with an ordered source can't exceed those concurrency limits.

Amazon EventBridge Pipes input transformation

Amazon EventBridge Pipes support optional input transformers when passing data to the enrichment and the target. You can use Input transformers to reshape the JSON event input payload to serve the needs of the enrichment or target service. For Amazon API Gateway and API destinations, this is how you shape the input event to the RESTful model of your API. Input transformers are modeled as an InputTemplate parameter. They can be free text, a JSON path to the event payload, or a JSON object that includes inline JSON paths to the event payload. For enrichment, the event payload is coming from the source. For targets, the event payload is what is returned from the enrichment, if one is configured on the pipe. In addition to the service-specific data in the event payload, you can use reserved variables (p. 194) in your InputTemplate to reference data for your pipe.

To access items in an array, use square bracket notation.

**Note**
EventBridge does not support all JSON Path syntax and evaluate it at runtime. Supported syntax includes:

- dot notation (for example, $.detail)
- dashes
- underscores
- alphanumeric characters
- array indices
- wildcards (*)

The following are sample InputTemplate parameters referencing an Amazon SQS event payload:

**Static string**

<table>
<thead>
<tr>
<th>InputTemplate: &quot;Hello, sender&quot;</th>
</tr>
</thead>
</table>

**JSON Path**

<table>
<thead>
<tr>
<th>InputTemplate: &lt;$.attributes.SenderId&gt;</th>
</tr>
</thead>
</table>

**Dynamic string**

<table>
<thead>
<tr>
<th>InputTemplate: &quot;Hello, &lt;$.attributes.SenderId&gt;&quot;</th>
</tr>
</thead>
</table>

**Static JSON**

```json
InputTemplate: {
```

193
Reserved variables

Input templates can use the following reserved variables:

- `<aws.pipes.pipe-arn>` – The Amazon Resource Name (ARN) of the pipe.
- `<aws.pipes.pipe-name>` – The name of the pipe.
- `<aws.pipes.source-arn>` – The ARN of the event source of the pipe.
- `<aws.pipes.enrichment-arn>` – The ARN of the enrichment of the pipe.
- `<aws.pipes.target-arn>` – The ARN of the target of the pipe.
- `<aws.pipes.event.ingestion-time>` – The time at which the event was received by the input transformer. This is an ISO 8601 timestamp. This time is different for the enrichment input transformer and the target input transformer, depending on when the enrichment completed processing the event.
- `<aws.pipes.event>` – The event as received by the input transformer.

For an enrichment input transformer, this is the event from the source. This contains the original payload from the source, plus additional service-specific metadata. See the topics in ?? (p. 163) for service-specific examples.

For a target input transformer, this is the event returned by the enrichment, if one is configured, with no additional metadata. As such, an enrichment-returned payload may be non-JSON. If no enrichment is configured on the pipe, this is the event from the source with metadata.

- `<aws.pipes.event.json>` – The same as `aws.pipes.event`, but the variable only has a value if the original payload, either from the source or returned by the enrichment, is JSON. If the pipe has an encoded field, such as the Amazon SQS body field or the Kinesis data, those fields are decoded and turned into valid JSON. Because it isn’t escaped, the variable can only be used as a value for a JSON field. For more information, see ??? (p. 195).

Note

EventBridge replaces input transformers at runtime to ensure a valid JSON output. Because of this, put quotes around variables that refer to JSON path parameters, but do not put quotes around variables that refer to JSON objects or arrays.
Input transform example

The following is an example Amazon EC2 event that we can use as our sample event.

```json
{
    "version": "0",
    "id": "7bf73129-1428-4cd3-a780-95db273d1602",
    "detail-type": "EC2 Instance State-change Notification",
    "source": "aws.ec2",
    "account": "123456789012",
    "time": "2015-11-11T21:29:54Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"
    ],
    "detail": {
        "instance-id": "i-0123456789",
        "state": "RUNNING"
    }
}
```

Let's use the following JSON as our Transformer.

```json
{
    "instance": "$.detail.instance-id",
    "state": "$.detail.state",
    "pipeArn": "aws.pipes.pipe-arn",
    "pipeName": "aws.pipes.pipe-name",
    "originalEvent": "aws.pipes.event.json"
}
```

The following will be the resulting Output:

```json
{
    "instance": "i-0123456789",
    "state": "RUNNING",
    "pipeArn": "arn:aws:pipe:us-east-1:123456789012:pipe/example",
    "pipeName": "example",
    "originalEvent": {
        "... // commented for brevity
    }
}
```

Implicit body data parsing

The following fields in the incoming payload may be JSON-escaped, such as the Amazon SQS body object, or base64-encoded, such as the Kinesis data object. For both filtering (p. 183) and input transformation, EventBridge transforms these fields into valid JSON so sub-values can be referenced directly. For example, `<$.data.someKey>` for Kinesis.

To have the target receive the original payload without any additional metadata, use an input transformer with this body data, specific to the source. For example, `<$.body>` for Amazon SQS, or `<$.data>` for Kinesis. If the original payload is a valid JSON string (for example `{"key": "value"}`), then use of the input transformer with source specific body data will result in the quotes within the original source payload being removed. For example, `{"key": "value"}` will become `"{key: value}"` when delivered to the target. If your target requires valid JSON payloads (for example,
EventBridge Lambda or Step Functions), this will cause delivery failure. To have the target receive the original source data without generating invalid JSON, wrap the source body data input transformer in JSON. For example, {"data": <$.data>}. Implicit body parsing can also be used to dynamically populate values for most pipe target or enrichment parameters. For more information, see ??? (p. 189).

**Note**
If the original payload is valid JSON, this field will contain the unescaped, non-base64-encoded JSON. However, if the payload is not valid JSON, EventBridge base64-encodes for the fields listed below, with the exception of Amazon SQS.

- **Active MQ** – data
- **Kinesis** – data
- **Amazon MSK** – key and value
- **Rabbit MQ** – data
- **Self managed Apache Kafka**; – key and value
- **Amazon SQS** – body

## Common issues with transforming input

These are some common issues when transforming input in EventBridge pipes:

- For Strings, quotes are required.
- There is no validation when creating JSON path for your template.
- If you specify a variable to match a JSON path that doesn't exist in the event, that variable isn't created and won't appear in the output.
- JSON properties like `aws.pipes.event.json` can only be used as the value of a JSON field, not inline in other strings.
- EventBridge doesn't escape values extracted by **Input Path**, when populating the **Input Template** for a target.
- If a JSON path references a JSON object or array, but the variable is referenced in a string, EventBridge removes any internal quotes to ensure a valid string. For example, "Body is <$.body>" would result in EventBridge removing quotes from the object.

Therefore, if you want to output a JSON object based on a single JSON path variable, you must place it as a key. In this example, {"body": <$.body>}. Quotes are not required for variables that represent strings. They are permitted, but EventBridge Pipes automatically adds quotes to string variable values during transformation, to ensure the transformation output is valid JSON. EventBridge Pipes does not add quotes to variables that represent JSON objects or arrays. Do not add quotes for variables that represent JSON objects or arrays.

For example, the following input template includes variables that represent both strings and JSON objects:

```json
{
    "pipeArn" : <aws.pipes.pipe-arn>,
    "pipeName" : <aws.pipes.pipe-name>,
    "originalEvent" : <aws.pipes.event.json>
}
```

Resulting in valid JSON with proper quotation:
Log pipe performance

```json
{
  "pipeArn": "arn:aws:events:us-east-2:123456789012:pipe/example",
  "pipeName": "example",
  "originalEvent": {
    ... // commented for brevity
  }
}
```

- For Lambda or Step Functions enrichments or targets, batches are delivered to the target as JSON arrays, even if the batch size is 1. However, input transformers will still be applied to individual records in the JSON Array, not the array as a whole. For more information, see ??? (p. 191).

Log Amazon EventBridge Pipes

EventBridge Pipes logging enables you to have EventBridge Pipes send records detailing pipe performance to supported AWS services. Use logs to gain insight into your pipe's execution performance, and to help with troubleshooting and debugging.

You can select the following AWS services as log destinations to which EventBridge Pipes delivers records:

- **CloudWatch Logs**
  EventBridge delivers log records to the specified CloudWatch Logs log group.
  Use CloudWatch Logs to centralize the logs from all of your systems, applications, and AWS services that you use, in a single, highly scalable service. For more information, see Working with log groups and log streams in the Amazon CloudWatch Logs User Guide.

- **Kinesis Data Firehose stream logs**
  EventBridge delivers log records to a Kinesis Data Firehose delivery stream.
  Amazon Kinesis Data Firehose is a fully-managed service for delivering real-time streaming data to destinations such as certain AWS services, as well as any custom HTTP endpoint or HTTP endpoints owned by supported third-party service providers. For more information, see Creating an Amazon Kinesis Data Firehose delivery stream in the Amazon Kinesis Data Firehose User Guide.

- **Amazon S3 logs**
  EventBridge delivers log records as Amazon S3 objects to the specified bucket.
  Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. For more information, see Uploading, downloading, and working with objects in Amazon S3 in the Amazon Simple Storage Service User Guide.

How Amazon EventBridge Pipes logging works

A pipe execution is an event or batch of events received by a pipe that travel to an enrichment and/or target. If enabled, EventBridge generates a log record for each execution step it performs as the event batch is processed. The information contained in the record applies to the event batch, be it a single event or up to 10,000 events.

You can configure the size of the event batch on the pipe source and target. For more information, see ??? (p. 191).

The record data sent to each log destination is the same.
If a Amazon CloudWatch Logs destination is configured, the log records delivered to all destinations have a limit of 256kb. Fields will be truncated as necessary.

You can customize the records EventBridge sends to the selected log destinations in the following way:

- You can specify the log level, which determines the execution steps for which EventBridge sends records to the selected log destinations. For more information, see ???(p. 198).
- You can specify whether EventBridge Pipes includes execution data in records for execution steps where it is relevant. This data includes:
  - The payload of the event batch
  - The request sent to the AWS enrichment or target service
  - The response returned by the AWS enrichment or target service

For more information, see ???(p. 199).

**Specifying EventBridge Pipes log level**

You can specify the types of execution steps for which EventBridge sends records to the selected log destinations.

Choose from the following levels of detail to include in log records. The log level applies to all log destinations specified for the pipe. Each log level includes the execution steps of the previous log levels.

- **OFF** – EventBridge does not send any records to any specified log destinations. This is the default setting.
- **ERROR** – EventBridge sends any records related to errors generated during pipe execution to the specified log destinations.
- **INFO** – EventBridge sends any records related to errors, as well as select other steps performed during pipe execution to the specified log destinations.
- **TRACE** – EventBridge sends any records generated during any steps in the pipe execution to the specified log destinations.

In the EventBridge console, CloudWatch logs is selected as a log destination by default, as is the ERROR log level. So, by default, EventBridge Pipes creates a new CloudWatch log group to which it sends log records containing the ERROR level of detail. No default is selected when you configure logs programmatically.

The following table lists the execution steps included in each log level.

<table>
<thead>
<tr>
<th>Step</th>
<th>TRACE</th>
<th>INFO</th>
<th>ERROR</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Execution Partially Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Execution Started</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Succeeded</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Throttled</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Execution Timeout</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Enrichment Invocation Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Enrichment Invocation Skipped</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Including execution data in EventBridge Pipes logs

You can specify for EventBridge to include *execution data* in the records it generates. Execution data includes fields representing the event batch payload, as well as the request sent to and the response from the enrichment and target.

Execution data is useful for troubleshooting and debugging. The *payload* field contains the actual contents of each event included in the batch, enabling you to correlate individual events to a specific pipe execution.

If you choose to include execution data, it is included for all log destinations specified for the pipe.

<table>
<thead>
<tr>
<th>Step</th>
<th>TRACE</th>
<th>INFO</th>
<th>ERROR</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enrichment Invocation Started</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enrichment Invocation Succeeded</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enrichment Stage Entered</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enrichment Stage Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Enrichment Stage Succeeded</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enrichment Transformation Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Enrichment Transformation Started</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enrichment Transformation Succeeded</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Invocation Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Target Invocation Partially Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Target Invocation Skipped</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Invocation Started</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Invocation Succeeded</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Stage Entered</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Stage Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Target Stage Partially Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Target Stage Skipped</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Stage Succeeded</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Transformation Failed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Target Transformation Started</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Transformation Succeeded</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Important
These fields may contain sensitive information. EventBridge makes no attempt to redact the contents of these fields during logging.

When including execution data, EventBridge adds the following fields to the relevant records:

- **payload**

  Represents the contents of the event batch being processed by the pipe.

  EventBridge includes the `payload` field in records generated at steps where the event batch contents may have been updated. This includes the following steps:
  - `EXECUTION_STARTED`
  - `ENRICHMENT_TRANSFORMATION_SUCCEEDED`
  - `ENRICHMENT_STAGE_SUCCEEDED`
  - `TARGET_TRANSFORMATION_SUCCEEDED`
  - `TARGET_STAGE_SUCCEEDED`

- **awsRequest**

  Represents the request sent to the enrichment or target as a JSON string. For requests sent to an API destination, this represents the HTTP request sent to that endpoint.

  EventBridge includes the `awsRequest` field in records generated at the final steps of enrichment and targeting; that is, after EventBridge has executed or attempted to execute the request against the specified enrichment or target service. This includes the following steps:
  - `ENRICHMENT_INVOCATION_FAILED`
  - `ENRICHMENT_INVOCATION_SUCCEEDED`
  - `TARGET_INVOCATION_FAILED`
  - `TARGET_INVOCATION_PARTIALLY_FAILED`
  - `TARGET_INVOCATION_SUCCEEDED`

  For examples of requests made to specific target services, see ???.

- **awsResponse**

  Represents the response returned by the enrichment or target, in JSON format. For requests sent to an API destination, this represents the HTTP response returned from that endpoint.

  As with `awsRequest`, EventBridge includes the `awsResponse` field in records generated at the final steps of enrichment and targeting; that is, after EventBridge has executed or attempted to execute a request against the specified enrichment or target service and received a response. This includes the following steps:
  - `ENRICHMENT_INVOCATION_FAILED`
  - `ENRICHMENT_INVOCATION_SUCCEEDED`
  - `TARGET_INVOCATION_FAILED`
  - `TARGET_INVOCATION_PARTIALLY_FAILED`
  - `TARGET_INVOCATION_SUCCEEDED`

  For a discussion of pipe execution steps, see ??? (p. 201).

### Truncating execution data in EventBridge Pipes log records

If you choose to have EventBridge include execution data in a pipe's log records, there is a possibility that a record may exceed the 256 KB size limit. To prevent this, EventBridge automatically truncates the
execution data fields, in the following order. EventBridge truncates each field entirely before progressing to truncate the next field. EventBridge truncates field data simply by removing characters from the end of the data string; no attempt is made to truncate based on data importance, and truncation will invalidate JSON formatting.

- payload
- awsRequest
- awsResponse

If EventBridge does truncate fields in the event, the truncatedFields field includes a list of the truncated data fields.

**Error reporting in EventBridge Pipes log records**

EventBridge also includes error data, where available, in pipe execution steps that represent failure states. These steps include:

- ExecutionThrottled
- ExecutionTimeout
- ExecutionFailed
- ExecutionPartiallyFailed
- EnrichmentTransformationFailed
- EnrichmentInvocationFailed
- EnrichmentStageFailed
- TargetTransformationFailed
- TargetInvocationFailed
- TargetInvocationPartiallyFailed
- TargetStageFailed
- TargetStagePartiallyFailed

**EventBridge Pipes execution steps**

Understanding the flow of pipe execution steps can aid you in troubleshooting or debugging your pipe's performance using logs.

A pipe execution is an event or batch of events received by a pipe that travel to an enrichment or target. If enabled, EventBridge generates a log record for each execution step it performs as the event batch is processed.

At a high level, the execution contains two stages, or collection of steps: enrichment, and target. Each of these stages consists of transformation and invocation steps.

The main steps of a successful pipe execution follows this flow:

- The pipe execution starts.
- The execution enters the enrichment stage if you have specified an enrichment for the events. If you haven't specified an enrichment, the execution proceeds to the target stage.

  In the enrichment stage, the pipe performs any transformation you have specified, then invokes the enrichment.
• In the target stage, the pipe performs any transformation you have specified, then invokes the target.

  If you haven’t specified transformation or target, the execution skips the target stage.

• The pipe execution completes successfully.

The diagram below demonstrates this flow. Diverging paths are formatted as dotted lines.

The diagram below presents a detailed view of the pipe execution flow, with all possible execution steps represented. Again, diverging paths are formatted as dotted lines.

For a complete list of pipe execution steps, see ?? (p. 198).
Note that target invocation may result in a partial failure of the batch. For more information, see ???(p. 191).

**EventBridge Pipes log schema reference**

The following reference details the schema for EventBridge Pipes log records.

Each log record represents a pipe execution step, and may contain up to 10,000 events if the pipe source and target have been configured for batching.

For more information, see ???(p. 197).

```json
{
  "executionId (p. 204)": "guid",
  "timestamp (p. 204)": "date_time",
  "messageType (p. 204)": "execution_step",
  "resourceArn (p. 204)": "arn:aws:pipes:region:account:pipe/pipe-name",
  "logLevel (p. 204)": "TRACE | INFO | ERROR",
  "payload (p. 204)": "{}
  "awsRequest (p. 205)": "{}
  "awsResponse (p. 205)": "{}
  "truncatedFields (p. 205)": ["awsRequest","awsResponse","payload"],
  "error (p. 205)": {
    "httpStatusCode (p. 205)": "code",
    "message (p. 205)": "error_message",
    "details (p. 205)": "",
    "awsService (p. 205)": "service_name",
    "requestId (p. 206)": "service_request_id"
  }
}
```

**executionId**

The ID of the pipe execution.

A pipe execution is an event or batch of events received by a pipe that travel to an enrichment or target. For more information, see ???(p. 197).

**timestamp**

The date and time the log event was emitted.

Unit: millisecond

**messageType**

The pipe execution step for which the record was generated.

For more information on pipe execution steps, see ???(p. 201).

**resourceArn**

The Amazon Resource Name (ARN) for the pipe.

**logLevel**

The level of detail specified for the pipe log.

*Valid values:* ERROR | INFO | TRACE

For more information, see ???(p. 198).

**payload**

The contents of the event batch being processed by the pipe.
EventBridge includes this field only if you have specified to include execution data in the logs for this pipe. For more information, see ???(p. 199).

**Important**
These fields may contain sensitive information. EventBridge makes no attempt to redact the contents of these fields during logging.

For more information, see ???(p. 199).

**awsRequest**

The request sent to the enrichment or target, in JSON format. For requests sent to an API destination, this represents the HTTP request sent to that endpoint.

EventBridge includes this field only if you have specified to include execution data in the logs for this pipe. For more information, see ???(p. 199).

**Important**
These fields may contain sensitive information. EventBridge makes no attempt to redact the contents of these fields during logging.

For more information, see ???(p. 199).

**awsResponse**

The response returned by the enrichment or target, in JSON format. For requests sent to an API destination, this represents the HTTP response returned from that endpoint, and not the response returned by the API Destination service itself.

EventBridge includes this field only if you have specified to include execution data in the logs for this pipe. For more information, see ???(p. 199).

**Important**
These fields may contain sensitive information. EventBridge makes no attempt to redact the contents of these fields during logging.

For more information, see ???(p. 199).

**truncatedFields**

A list of any execution data fields EventBridge has truncated to keep the record below the 256 KB size limitation.

If EventBridge did not have to truncate any of the execution data fields, this field is present but null.

For more information, see ???(p. 200).

**error**

Contains information for any error generated during this pipe execution step.

If no error was generated during this pipe execution step, this field is present but null.

**httpStatusCode**

The HTTP status code returned by the called service.

**message**

The error message returned by the called service.

**details**

Any detailed error information returned by the called service.

**awsService**

The name of the service called.
requestId

The request ID for this request from the called service.

Logging and monitoring Amazon EventBridge Pipes using AWS CloudTrail and Amazon CloudWatch Logs

You can log EventBridge Pipes invocations and using CloudTrail and monitor the health of your pipes using CloudWatch metrics.

CloudWatch metrics

EventBridge Pipes sends metrics to Amazon CloudWatch every minute for everything from a pipe executions being throttled to a target successfully being invoked.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>The number of concurrent executions of a pipe.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: AwsAccountId</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>Duration</td>
<td>Length of time the pipe execution took.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
</tr>
<tr>
<td>EventCount</td>
<td>The number of events a pipe has processed.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>EventSize</td>
<td>The size of the payload of the event that invoked the pipe.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td>ExecutionThrottled</td>
<td>How many executions of a pipe were throttled.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This value will be 0 if no executions were throttled.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: AwsAccountId, PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>ExecutionTimeout</td>
<td>How many executions of a pipe timed out before completing execution.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This value will be 0 if no executions timed out.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ExecutionFailed</td>
<td>How many executions of a pipe failed.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This value will be 0 if no executions failed.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>ExecutionPartiallyFailed</td>
<td>How many executions of a pipe partially failed.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This value will be 0 if no executions partially failed.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>EnrichmentStageDuration</td>
<td>How long the enrichment stage took to complete.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
</tr>
<tr>
<td>EnrichmentStageFailed</td>
<td>How many executions of a pipe's enrichment stage failed.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This value will be 0 if no executions failed.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>Invocations</td>
<td>Total number of invocations.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: AwsAccountIId, PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>TargetStageDuration</td>
<td>How long the target stage took to complete.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
</tr>
<tr>
<td>TargetStageFailed</td>
<td>How many executions of a pipe's target stage failed.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This value will be 0 if no executions failed.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: PipeName</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| TargetStagePartiallyFailed | How many executions of a pipe's target stage partially failed.  
  **Note**  This value will be 0 if no target stage executions partially failed.  
  Valid Dimensions: PipeName  
  Units: None |
| TargetStageSkipped     | How many executions of a pipe's target stage were skipped (for example, due to the enrichment returning an empty payload).  
  Valid Dimensions: PipeName  
  Units: Count |

**Dimensions for CloudWatch metrics**

CloudWatch metrics have *dimensions*, or sortable attributes, which are listed below.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AwsAccountId</td>
<td>Filters the available metrics by account ID.</td>
</tr>
<tr>
<td>PipeName</td>
<td>Filters the available metrics by pipe name.</td>
</tr>
</tbody>
</table>

**CloudTrail logging**

EventBridge Pipes works with AWS CloudTrail, a service that records actions from AWS services. CloudTrail captures API calls made by or on behalf of your AWS account from the EventBridge console and to EventBridge Pipes API operations.

Using the information collected by CloudTrail, you can determine what request was made to EventBridge, the IP address from which the request was made, who made the request, when it was made, and more.

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

CloudTrail is enabled on your AWS account when you create your account. When an event occurs in EventBridge, CloudTrail records the event 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 a record of events in your AWS account, including events for EventBridge, create a trail. A *trail* is a configuration that CloudTrail uses to deliver log files to an Amazon S3 bucket. By default, the trail logs events from all Regions in the AWS partition and then delivers the log files to an S3 bucket. You can configure other AWS services to analyze and act on 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](#)
You can log the following EventBridge Pipes actions as events in CloudTrail log files:

- **CreatePipe**
- **DeletePipe**
- **DescribePipe**
- **ListPipes**
- **StartPipe**
- **StopPipe**
- **UpdatePipe**

**CloudTrail log entries for actions taken by EventBridge Pipes**

EventBridge Pipes assumes the provided IAM role when reading events from sources, invoking enrichments, or invoking targets. For CloudTrail entries related to actions taken in your account on all enrichments, targets, and Amazon SQS, Kinesis, and DynamoDB sources, the `sourceIPAddress` and `invokedBy` fields will include `pipes.amazonaws.com`.

**Sample CloudTrail log entry for all enrichments, targets, and Amazon SQS, Kinesis, and DynamoDB sources**

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "...",
    "arn": "arn:aws:sts::111222333444:assumed-role/...",
    "accountId": "111222333444",
    "accessKeyId": "...",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "...",
        "arn": "...",
        "accountId": "111222333444",
        "userName": "...",
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-09-22T21:41:15Z",
        "mfaAuthenticated": "false"
      }
    },
    "invokedBy": "pipes.amazonaws.com"
  },
  "eventTime": "...",
  "eventName": "...",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "pipes.amazonaws.com",
  "userAgent": "pipes.amazonaws.com",
  "requestParameters": {
    "...
  },
  "responseElements": null,
  "requestID": "...",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "...",
  "eventCategory": "Management"
}
```
For all other sources, the sourceIPAddress field of the CloudTrail log entries will have a dynamic IP address and shouldn't be relied upon for any integration or event categorization. In addition, these entries won't have the invokedBy field.

**Sample CloudTrail log entry for all other sources**

```json
{
  "eventVersion": "1.08",
  "userIdentity": {
    ...
  },
  "eventTime": ",,",
  "eventName": ",,",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  ".userAgent": "Python-httplib2/0.8 (gzip)",
}
```

### Amazon EventBridge Pipes error handling and troubleshooting

#### Retry behavior and error handling

EventBridge Pipes automatically retries enrichment and target invocation on any retryable AWS failures with the source service, the enrichment or target services, or EventBridge. However, if there are failures returned by enrichment or target customer implementations, the pipe polling throughput will gradually back off. For nearly continuous 4xx errors (such as authorization problems with IAM or missing resources), the pipe can be automatically disabled with an explanatory message in the StateReason.

#### Pipe invocation errors and retry behavior

When you invoke a pipe, two main types of errors can occur: pipe internal errors and customer invocation errors.

**Pipe internal errors**

Pipe internal errors are errors resulting by aspects of the invocation managed by the EventBridge Pipes service.

These types of errors can include issues such as:

- A HTTP connection failure when attempting to invoke the customer target service
- A transient drop in availability on the pipe service itself

In general, EventBridge Pipes retries internal errors an indefinite number of times, and stops only when the record expires in the source.

For pipes with a stream source, EventBridge Pipes does not count retries for internal errors against the maximum number of retries specified on the retry policy for the stream source. For pipes with an Amazon SQS source, EventBridge Pipes does not count retries for internal errors against the maximum receive count for the Amazon SQS source.
Customer invocation errors

Customer invocation errors are errors resulting from configuration or code managed by the user.

These types of errors can include issues such as:

- Insufficient permissions on the pipe to invoke the target.
- A logic error in a synchronously-invoked customer Lambda, Step Functions, API destination, or API Gateway endpoint.

For customer invocation errors, EventBridge Pipes does the following:

- For pipes with a stream source, EventBridge Pipes retries up to the maximum retry times configured on the pipe retry policy or until the maximum record age expires, whichever comes first.
- For pipes with an Amazon SQS source, EventBridge Pipes retries a customer error up to the maximum receive count on the source queue.
- For pipes with a Apache Kafka or Amazon MQ source, EventBridge retries customer errors the same as it retries internal errors.

For pipes with compute targets, you must invoke the pipe synchronously in order for EventBridge Pipes to be aware of any runtime errors that are thrown from the customer compute logic and retry on such errors. Pipes cannot retry on errors thrown from the logic of a Step Functions standard workflow, as this target must be invoked asynchronously.

For Amazon SQS and stream sources, such as Kinesis and DynamoDB, EventBridge Pipes supports partial batch failure handling of target failures. For more information, see Partial batch failure.

Pipe DLQ behavior

A pipe inherits dead-letter queue (DLQ) behavior from the source:

- If the source Amazon SQS queue has a configured DLQ, messages are automatically delivered there after the specified number of attempts.
- For stream sources, such as DynamoDB and Kinesis streams, you can configure a DLQ for the pipe and route events. DynamoDB and Kinesis stream sources support Amazon SQS queues and Amazon SNS topics as DLQ targets.

If you specify a DeadLetterConfig for a pipe with a Kinesis or DynamoDB source, make sure that the MaximumRecordAgeInSeconds property on the pipe is less than the MaximumRecordAge of the source event. MaximumRecordAgeInSeconds controls when the pipe poller will give up on the event and deliver it to the DLQ and the MaximumRecordAge controls how long the message will be visible in the source stream before it gets deleted. Therefore, set MaximumRecordAgeInSeconds to a value that is less than the source MaximumRecordAge so that there's adequate time between when the event gets sent to the DLQ, and when it gets automatically deleted by the source for you to determine why the event went to the DLQ.

For Amazon MQ sources, the DLQ can be configured directly on the message broker.

EventBridge Pipes does not support first-in first-out (FIFO) DLQs for stream sources.

EventBridge Pipes does not support DLQ for Amazon MSK stream and Self managed Apache Kafka stream sources.
Pipe failure states

Creating, deleting, and updating pipes are asynchronous operations that might result in a failure state. Likewise, a pipe might be automatically disabled due to failures. In all cases, the pipe StateReason provides information to help troubleshoot the failure.

The following is a sample of the possible StateReason values:

- Stream not found. To resume processing please delete the pipe and create a new one.
- Pipes does not have required permissions to perform Queue operations (sqs:ReceiveMessage, sqs:DeleteMessage and sqs:GetQueueAttributes)
- Connection error. Your VPC must be able to connect to pipes. You can provide access by configuring a NAT Gateway. For how to setup NAT gateway, please check AWS documentation.
- MSK cluster does not have security groups associated with it

A pipe may be automatically stopped with an updated StateReason. Possible reasons include:

- A Step Functions standard workflow configured as an enrichment.
- A Step Functions standard workflow configured as as a target to be invoked synchronously.

Custom encryption failures

If you configure a source to use an AWS KMS custom encryption key (CMK), rather than an AWS-managed AWS KMS key, you must explicitly give your pipe's Execution Role decryption permission. To do so, include the following additional permission in the custom CMK policy:

```json
{
    "Sid": "Allow Pipes access",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::01234567890:role/service-role/Amazon_EventBridge_Pipe_DDBStreamSourcePipe_12345678"
    },
    "Action": "kms:Decrypt",
    "Resource": "*"
}
```

Replace the above role with your pipe's Execution Role.

This is true for all pipe sources with AWS KMS CMK, including:

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams
- Amazon MQ
- Amazon MSK
- Amazon SQS
Tutorial: Create an EventBridge pipe that filters source events

In this tutorial, you'll create a pipe that connects a DynamoDB stream source to an Amazon SQS queue target. This includes specifying an event pattern for the pipe to use when filtering events to deliver to the queue. You'll then test the pipe to ensure that only the desired events are being delivered.

Prerequisites: Create the source and target

Before you create the pipe, you'll need to create the source and target that the pipe is to connect. In this case, an Amazon DynamoDB data stream to act as the pipe source, and an Amazon SQS queue as the pipe target.

To simplify this step, you can use AWS CloudFormation to provision the source and target resources. To do this, you'll create a CloudFormation template defining the following resources:

- The pipe source
  An Amazon DynamoDB table, named pipe-tutorial-source, with a stream enabled to provide an ordered flow of information about changes to items in the DynamoDB table.
- The pipe target
  An Amazon SQS queue, named pipe-tutorial-target, to receive the DynamoDB stream of events from your pipe.

To create the CloudFormation template for provisioning pipe resources

1. Copy the JSON template text in the ??? section, below.
2. Save the template as a JSON file (for example, ~/pipe-tutorial-resources.json).

Next, use the template file you just created to provision a CloudFormation stack.

Note
Once you create your CloudFormation stack, you will be charged for the AWS resources it provisions.

Provision the tutorial prerequisites using the AWS CLI

1. Run the following CLI command, where --template-body specifies the location of your template file:

   ```bash
   aws cloudformation create-stack --stack-name pipe-tutorial-resources --template-body file://~/pipe-tutorial-resources.json
   ```

Provision tutorial prerequisites using the CloudFormation console

2. Select Stacks, then select Create stack, and choose with new resources (standard).
   CloudFormation displays the Create stack wizard.
3. For Prerequisite - Prepare template, leave the default, Template is ready, selected.
4. Under Specify template, select Upload a template file, and then choose the file and select Next.
5. Configure the stack and the resources it will provision:
   - For **Stack name**, enter `pipe-tutorial-resources`.
   - For **Parameters**, leave the default names for the DynamoDB table and Amazon SQS queue.
   - Choose **Next**.

6. Choose **Next**, then choose **Submit**.

   CloudFormation creates the stack and provisions the resources defined in the template.


### Step 1: Create the pipe

With the pipe source and target provisioned, you can now create the pipe to connect the two services.

**Create the pipe using the EventBridge console**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. On the navigation pane, choose **Pipes**.
3. Choose **Create pipe**.
4. For **Name**, name your pipe `pipe-tutorial`.
5. Specify the DynamoDB data stream source:
   a. Under **Details**, for **Source**, select **DynamoDB data stream**.
      
      EventBridge displays DynamoDB-specific source configuration settings.
   b. For **DynamoDB stream**, select `pipe-tutorial-source`.
      
      Leave **Starting position** set to the default, **Latest**.
   c. Choose **Next**.
6. Specify and test an event pattern to filter events:

   Filtering enables you to control which events the pipes sends to enrichment or the target. The pipe only sends events that match the event pattern on to enrichment or the target.

   For more information, see [?? (p. 183)](https://docs.aws.amazon.com/AWSEvents/latest/userguide/define-event-patterns.html).

   **Note**
   
   You are only billed for those events sent to enrichment or the target.

   a. Under **Sample event - optional**, leave **AWS events** selected, and make sure that **DynamoDB Stream Sample event 1** is selected.
      
      This is the sample event which you'll use to test our event pattern.
   b. Under **Event pattern**, enter the following event pattern:

      ```json
      {
        "eventName": ["INSERT", "MODIFY"]
      }
      ```
      
      c. Choose **Test pattern**.

      EventBridge displays a message that the sample event matches the event pattern. This is because the sample event has an `eventName` value of INSERT.
d. Choose Next.

7. Choose Next to skip specifying an enrichment.

   In this example, you won’t select an enrichment. Enrichments enable you to select a service to
   enhance the data from the source before sending it to the target. For more details, see ??? (p. 187).

8. Specify your Amazon SQS queue as the pipe target:
   a. Under Details, for Target service, select Amazon SQS queue.
   b. For Queue, select pipe-tutorial-target.
   c. Leave the Target Input transformer section empty.

   For more information, see ??? (p. 193).

9. Choose Create Pipe

   EventBridge creates the pipe and displays the pipe detail page. The pipe is ready once its status
   updates to Running.

Step 2: Confirm the pipe filters events

Pipe is set up, but has yet to receive events from table.

To test the pipe, you’ll update entries in the DynamoDB table. Each update will generate events that the
DynamoDB stream sends to our pipe. Some will match the event pattern you specified, some will not.
You can then examine the Amazon SQS queue to ensure that the pipe only delivered those event that
matched our event pattern.

Update table items to generate events

2. From the left navigation, select Tables. Select the pipe-tutorial-source table.

   DynamoDB displays the table details page for pipe-tutorial-source.
3. Select Explore table items, and then choose Create item.

   DynamoDB displays the Create item page.
4. Under Attributes, create a new table item:
   a. For Album enter Album A.
   b. For Artist enter Artist A.
   c. Choose Create item.
5. Update the table item:
   a. Under Items returned, choose Album A.
   b. Select Add new attribute, then select String.
   c. Enter a new value of Song, with a value of Song A.
   d. Choose Save changes.
6. Delete the table item:
   a. Under Items returned, check Album A.
   b. From the Actions menu, select Delete items.

You have made three updates to the table item; this generates three events for the DynamoDB data
stream:
• An INSERT event when you created the item.
• A MODIFY event when you added an attribute to the item.
• A REMOVE event when you deleted the item.

However, the event pattern you specified for the pipe should filter out any events that are not INSERT or MODIFY events. Next, confirm that the pipe delivered the expected events to the queue.

**Confirm the expected events were delivered to the queue**

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Choose the pipe-tutorial-target queue.
   
   Amazon SQS displays the queue details page.
3. Select **Send and receive messages**, then under **Receive messages** choose **Poll for messages**.
   
   The queue polls the pipe and then lists the events it receives.
4. Choose the event name to see the event JSON that was delivered.

There should be two events in the queue: one with an eventName of INSERT, and one with an eventName of MODIFY. However, the pipe did not deliver the event for deleting the table item, since that event had an eventName of REMOVE, which did not match the event pattern you specified in the pipe.

**Step 3: Clean up your resources**

First, delete the pipe itself.

**Delete the pipe using the EventBridge console**

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. On the navigation pane, choose **Pipes**.
3. Select the pipe-tutorial pipe, and choose **Delete**.

Then, delete the CloudFormation stack, to prevent being billed for the continued usage of the resources provisioned within it.

**Delete the tutorial prerequisites using the AWS CLI**

- Run the following CLI command, where **--stack-name** specifies the name of your stack:

```
aws cloudformation delete-stack --stack-name pipe-tutorial-resources
```

**Delete the tutorial prerequisites using the AWS CloudFormation console**

2. On the **Stacks** page, select the stack and then select **Delete**.
3. Select **Delete** to confirm your action.
AWS CloudFormation template for generating prerequisites

Use the JSON below to create a CloudFormation template for provisioning the source and target resources necessary for this tutorial.

```json
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Provisions resources to use with the EventBridge Pipes tutorial. You will be billed for the AWS resources used if you create a stack from this template.",
  "Parameters": {
    "SourceTableName": {
      "Type": "String",
      "Default": "pipe-tutorial-source",
      "Description": "Specify the name of the table to provision as the pipe source, or accept the default."
    },
    "TargetQueueName": {
      "Type": "String",
      "Default": "pipe-tutorial-target",
      "Description": "Specify the name of the queue to provision as the pipe target, or accept the default."
    }
  },
  "Resources": {
    "PipeTutorialSourceDynamoDBTable": {
      "Type": "AWS::DynamoDB::Table",
      "Properties": {
        "AttributeDefinitions": [
          {
            "AttributeName": "Album",
            "AttributeType": "S"
          },
          {
            "AttributeName": "Artist",
            "AttributeType": "S"
          }
        ],
        "KeySchema": [
          {
            "AttributeName": "Album",
            "KeyType": "HASH"
          },
          {
            "AttributeName": "Artist",
            "KeyType": "RANGE"
          }
        ],
        "ProvisionedThroughput": {
          "ReadCapacityUnits": 10,
          "WriteCapacityUnits": 10
        },
        "StreamSpecification": {
          "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "TableName": { "Ref": "SourceTableName" }
      }
    },
    "PipeTutorialTargetQueue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
      
      }
    }
  }
}
```
"QueueName": { "Ref" : "TargetQueueName" }

}
Making applications Regional-fault tolerant with global endpoints and event replication

You can improve your application’s availability with Amazon EventBridge global endpoints. Global endpoints help make your application regional-fault tolerant at no additional cost. To start, you assign an Amazon Route 53 health check to the endpoint. When failover is initiated, the health check reports an “unhealthy” state. Within minutes of failover initiation, all custom events are routed to an event bus in the secondary Region and are processed by that event bus. Once the health check reports a “healthy” state, events are processed by the event bus in the primary Region.

When you use global endpoints, you can enable event replication. Event replication sends all custom events to the event buses in the primary and secondary Regions using managed rules.

**Note**

If you're using custom buses, you'll need a custom bus in each Region with the same name and in the same account for failover to work properly.

**Topics**

- Recovery Time & Recovery Point Objectives (p. 219)
- Event replication (p. 219)
- Create a global endpoint (p. 220)
- Working with global endpoints by using an AWS SDK (p. 221)
- Available Regions (p. 222)
- Best practices for working with Amazon EventBridge global endpoints (p. 222)
- AWS CloudFormation template for setting up the Route 53 health check (p. 223)

**Recovery Time & Recovery Point Objectives**

The Recovery Time Objective (RTO) is the time that it takes for the secondary Region to start receiving events after a failure. For RTO, the time includes time period for triggering CloudWatch alarms and updating statuses for Route 53 health checks. The Recovery Point Objective (RPO) is the measure of the data that will be left unprocessed during a failure. For RPO, the time includes events that are not replicated to the secondary Region and are stuck in the primary Region until the service or Region recovers. With global endpoints, if you follow our prescriptive guidance for alarm configuration, you can expect the RTO and RPO to be 360 seconds with a maximum of 420 seconds.

**Event replication**

Events are processed in the secondary Region asynchronously. This means that events are not guaranteed to be processed at the same time in both Regions. When failover is triggered, the events are processed by the secondary Region and will be processed by the primary Region when it's available. Enabling event replication will increase your monthly costs. For more information, see Amazon EventBridge pricing.
We recommend enabling event replication when setting up global endpoints for the following reasons:

- Event replication helps you verify that your global endpoints are configured correctly. This helps to ensure that you'll be covered in the event of failover.
- Event replication is required to automatically recover from a failover event. If you don't have event replication enabled, you'll have to manually reset the Route 53 health check to "healthy" before events will go back to the primary Region.

Replicated event payload

The following is an example of a replicated event payload:

```
Note
For region, the Region that the event was replicated from is listed.

{
  "version": "0",
  "id": "a908baa3-65e5-ab77-367e-527c0e71bbc2",
  "detail-type": "Test",
  "source": "test.service.com",
  "account": "0123456789",
  "time": "1900-01-01T00:00:00Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:events:us-east-1:0123456789:endpoint/MyEndpoint"
  ],
  "detail": {
    "a": "b"
  }
}
```

Create a global endpoint

Complete the following steps to set up a global endpoint:

1. Make sure that you have matching event buses and rules in both the primary and secondary Region.
2. Create a Route 53 health check to monitor your event buses. For assistance in creating your health check, choose New Health Check when creating your global endpoint.
3. Create your global endpoint.

Once you have set up the Route 53 health check, you can create a global endpoint.

To create a global endpoint by using the console

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Global endpoints.
3. Choose Create Endpoint.
4. Enter a name and description for the endpoint.
5. For Event bus in primary Region, choose the event bus you'd like the endpoint associated with.
6. For Secondary Region, choose the Region you'd like to direct events to in the event of a failover.

Note
The Event bus in secondary Region is auto-filled and not editable.
7. For **Route 53 health check for triggering failover and recovery**, choose the health check that the endpoint will monitor. If you don't already have a health check, choose **New Health check** to open the AWS CloudFormation console and create a health check using a CloudFormation template.

   **Note**
   Missing data will cause the health check to fail. If you only need to send events intermittently, consider using a longer **MinimumEvaluationPeriod**, or treat missing data as 'missing' instead of 'breaching'.

8. (Optional) For **Event replication** do the following:

   a. Select **Event replication enabled**.
   b. For **Execution role**, choose whether to create a new AWS Identity and Access Management role or use an existing one. Do the following:
      - Choose **Create a new role for this specific resource**. Optionally, you can update the **Role name** to create a new role.
      - Choose **Use existing role**. Then, for **Execution role**, choose the desired role to use.

9. Choose **Create**.

**To create a global endpoint by using the API**

To create a global endpoint using the EventBridge API, see **CreateEndpoint** in the Amazon EventBridge API Reference.

**To create a global endpoint by using AWS CloudFormation**

To create a global endpoint using the AWS CloudFormation API, see **AWS::Events::Endpoints** in the AWS CloudFormation User Guide.

**Working with global endpoints by using an AWS SDK**

**Note**
Support for C++ is coming soon.

When using an AWS SDK to work with global endpoints, keep the following in mind:

- You'll need to have the AWS Common Runtime (CRT) library installed for your specific SDK. If you don't have the CRT installed, you'll get an exception message indicating what needs to be installed. For more information, see the following:
  - **AWS Common Runtime (CRT) libraries**
  - awslabs/aws-crt-java
  - awslabs/aws-crt-nodejs
  - awslabs/aws-crt-python
- Once you have created a global endpoint, you'll need to add the **endpointId** and **EventBusName** to any **PutEvents** calls that you use.
- Global endpoints support Signature Version 4A. This version of SigV4 allows requests to be signed for multiple AWS Regions. This is useful in API operations that might result in data access from one of several Regions. When using the AWS SDK, you supply your credentials and the requests to global
Available Regions

The following Regions support global endpoints:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- South America (São Paulo)

Best practices for working with Amazon EventBridge global endpoints

The following best practices are recommended when you set up global endpoints.

Topics

- Enabling event replication (p. 222)
- Preventing event throttling (p. 223)
- Using subscriber metrics in Amazon Route 53 health checks (p. 223)

Enabling event replication

We strongly recommend that you turn on replication and process your events in the secondary Region that you assign to your global endpoint. This ensures that your application in the secondary Region is configured correctly. You should also turn on replication to ensure automatic recovery to the primary Region after an issue has been mitigated.

Event IDs can change across API calls so correlating events across Regions requires you to have an immutable, unique identifier. Consumers should also be designed with idempotency in mind. That way, if you’re replicating events, or replaying them from archives, there are no side effects from the events being processed in both Regions.
Preventing event throttling

To prevent events from being throttled, we recommend updating your PutEvents and targets limits so they're consistent across Regions.

Using subscriber metrics in Amazon Route 53 health checks

Avoid including subscriber metrics in your Amazon Route 53 health checks. Including these metrics may cause your publisher to failover to the secondary Regions if a subscriber encounters an issue despite all other subscribers remaining healthy in the primary Region. If one of your subscribers is failing to process events in the primary Region, you should turn on replication to ensure that your subscriber in the secondary Region can process events successfully.

AWS CloudFormation template for setting up the Route 53 health check

When using global endpoints you have to have a Route 53 health check to monitor the status of your Regions. The following template defines a Amazon CloudWatch alarm and uses it to define a Route 53 health check.

Topics

- AWS CloudFormation template for defining a Route 53 health check (p. 223)
- CloudWatch alarm template properties (p. 225)
- Route 53 health check template properties (p. 226)

AWS CloudFormation template for defining a Route 53 health check

Use the following template to define your Route 53 health check.

Description: |-
Global endpoints health check that will fail when the average Amazon EventBridge latency is above 30 seconds for a duration of 5 minutes. Note, missing data will cause the health check to fail, so if you only send events intermittently, consider changing the health check to use a longer evaluation period or instead treat missing data as 'missing' instead of 'breaching'.

Metadata:
AWS::CloudFormation::Interface:
ParameterGroups:
- Label: default: "Global endpoint health check alarm configuration"
Parameters:
- HealthCheckName
- HighLatencyAlarmPeriod
- MinimumEvaluationPeriod
- MinimumThreshold
- TreatMissingDataAs
ParameterLabels:
  HealthCheckName:
    default: Health check name
HighLatencyAlarmPeriod:
  default: High latency alarm period
MinimumEvaluationPeriod:
  default: Minimum evaluation period
MinimumThreshold:
  default: Minimum threshold
TreatMissingDataAs:
  default: Treat missing data as

Parameters:
HealthCheckName:
  Description: Name of the health check
  Type: String
  Default: LatencyFailuresHealthCheck
HighLatencyAlarmPeriod:
  Description: The period, in seconds, over which the statistic is applied. Valid values are 10, 30, 60, and any multiple of 60.
  MinValue: 10
  Type: Number
  Default: 60
MinimumEvaluationPeriod:
  Description: The number of periods over which data is compared to the specified threshold. You must have at least one evaluation period.
  MinValue: 1
  Type: Number
  Default: 5
MinimumThreshold:
  Description: The value to compare with the specified statistic.
  Type: Number
  Default: 30000
TreatMissingDataAs:
  Description: Sets how this alarm is to handle missing data points.
  Type: String
  AllowedValues:
  - breaching
  - notBreaching
  - ignore
  - missing
  Default: breaching

Mappings:
  "InsufficientDataMap":
    "missing":
      "HCConfig": "LastKnownStatus"
    "breaching":
      "HCConfig": "Unhealthy"

Resources:
  HighLatencyAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: High Latency in Amazon EventBridge
      MetricName: IngestionToInvocationStartLatency
      Namespace: AWS/Events
      Statistic: Average
      Period: !Ref HighLatencyAlarmPeriod
      EvaluationPeriods: !Ref MinimumEvaluationPeriod
      Threshold: !Ref MinimumThreshold
      ComparisonOperator: GreaterThanThreshold
      TreatMissingData: !Ref TreatMissingDataAs

  LatencyHealthCheck:
    Type: AWS::Route53::HealthCheck
    Properties:
      HealthCheckTags:
        - Key: Name
Amazon EventBridge User Guide
CloudWatch alarm template properties

```yaml
Value: !Ref HealthCheckName
HealthCheckConfig:
  Type: CLOUDWATCH_METRIC
AlarmIdentifier:
  Name:
    Ref: HighLatencyAlarm
  Region: !Ref AWS::Region
  InsufficientDataHealthStatus: !FindInMap [InsufficientDataMap, !Ref TreatMissingDataAs, HCConfig]

Outputs:
  HealthCheckId:
    Description: The identifier that Amazon Route 53 assigned to the health check when you created it.
    Value: !GetAtt LatencyHealthCheck.HealthCheckId
```

Event IDs can change across API calls so correlating events across Regions requires you to have an immutable, unique identifier. Consumers should also be designed with idempotency in mind. That way, if you’re replicating events, or replaying them from archives, there are no side effects from the events being processed in both Regions.

CloudWatch alarm template properties

**Note**

For all editable fields, consider your throughput per second. If you only send events intermittently, consider changing the health check to use a longer evaluation period or instead treat missing data as missing instead of breaching.

The following properties are used in the CloudWatch alarm section of the template:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlarmDescription</td>
<td>The description of the alarm.</td>
</tr>
<tr>
<td></td>
<td>Default: High Latency in Amazon EventBridge</td>
</tr>
<tr>
<td>MetricName</td>
<td>The name of the metric associated with the alarm. This is required for an alarm based on a metric. For an alarm based on a math expression, you use Metrics instead and you can't specify MetricName.</td>
</tr>
<tr>
<td></td>
<td>Default: IngestionToInvocationStartLatency</td>
</tr>
<tr>
<td>Namespace</td>
<td>The namespace of the metric associated with the alarm. This is required for an alarm based on a metric. For an alarm based on a math expression, you can't specify Namespace and you use Metrics instead.</td>
</tr>
<tr>
<td></td>
<td>Default: AWS/Events</td>
</tr>
<tr>
<td>Statistic</td>
<td>The statistic for the metric associated with the alarm, other than percentile.</td>
</tr>
<tr>
<td></td>
<td>Default: Average</td>
</tr>
<tr>
<td>Period</td>
<td>The period, in seconds, over which the statistic is applied. This is required for an alarm based on a metric. Valid values are 10, 30, 60, and any multiple of 60.</td>
</tr>
<tr>
<td></td>
<td>Default: 60</td>
</tr>
<tr>
<td>EvaluationPeriods</td>
<td>The number of periods over which data is compared to the specified threshold. If you are setting an alarm that requires that a number of consecutive data points be breaching to trigger the alarm, this value</td>
</tr>
</tbody>
</table>
Route 53 health check template properties

**Note**
For all editable fields, consider your throughput per second. If you only send events intermittently, consider changing the health check to use a longer evaluation period or instead treat missing data as missing instead of breaching.

The following properties are used in the Route 53 health check section of the template:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HealthCheckName</td>
<td>The name of the health check.</td>
</tr>
<tr>
<td></td>
<td>Default: LatencyFailuresHealthCheck</td>
</tr>
<tr>
<td>InsufficientDataHealthStatus</td>
<td>CloudWatch has insufficient data about the metric to determine the alarm state, the status that you want Amazon Route 53 to assign to the health check</td>
</tr>
<tr>
<td></td>
<td>Valid values: Healthy: Route 53 considers the health check to be healthy.</td>
</tr>
<tr>
<td></td>
<td>Unhealthy: Route 53 considers the health check to be unhealthy.</td>
</tr>
<tr>
<td></td>
<td>LastKnownStatus: Route 53 uses the status of the health check from the last time that CloudWatch had sufficient data to determine the alarm state. For new health checks that have no last known status, the default status for the health check is healthy.</td>
</tr>
<tr>
<td></td>
<td>Default: Unhealthy</td>
</tr>
</tbody>
</table>

**Note**
This field is updated based on the input to the TreatMissingData field. If TreatMissingData is set to Missing, it will be updated to LastKnownStatus. If TreatMissingData is set to Breaching, it will be updated to Unhealthy.
Amazon EventBridge schemas

A schema defines the structure of events (p. 27) that are sent to EventBridge. EventBridge provides schemas for all events that are generated by AWS services. You can also create or upload custom schemas (p. 231) or infer schemas (p. 235) directly from events on an event bus (p. 9). Once you have a schema for an event, you can download code bindings for popular programming languages and speed up development. You can work with code bindings for schemas and manage schemas from the EventBridge console, by using the API, or directly in your IDE by using the AWS toolkits. To build serverless apps that use events, use AWS Serverless Application Model.

Note
When using the input transformer (p. 137) feature, the original event is inferred by schema discovery, not the transformed event that’s sent to the target.

EventBridge supports both OpenAPI 3 and JSONSchema Draft4 formats.

For AWS Toolkit for JetBrains and AWS Toolkit for VS Code, you can browse or search for schemas and download code bindings for schemas directly in your IDE.

The following video gives an overview of schemas and schema registries: Using the Schema Registry

Topics
- Schema registry API property value masking (p. 228)
- Finding an Amazon EventBridge schema (p. 229)
- Amazon EventBridge schema registries (p. 230)
- Creating an Amazon EventBridge schema (p. 231)
- Amazon EventBridge code bindings (p. 236)
Schema registry API property value masking

Some property values of events that are used to create a schema registry may contain sensitive customer information. To protect the customer's information, the values will be masked with asterisks (*). Because we're masking these values, EventBridge recommends not building applications that explicitly depend on the following properties or their values:

- **CreateSchema** – The Content property of the requestParameters body
- **GetDiscoveredSchema** – The Events property of the requestParameters body and the Content property of the responseElements body
- **SearchSchemas** – The keywords property of the requestParameters
- **UpdateSchema** – The Content property of the requestParameters
Finding an Amazon EventBridge schema

EventBridge includes schemas (p. 227) for all AWS services that generate events. You can find these schemas in the EventBridge console, or you can find them by using the API action SearchSchemas.

To find schemas for AWS services in the EventBridge console

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schemas.
3. On the Schemas page, select AWS event schema registry.

The first page of available schemas is displayed.

4. To find a schema, in Search AWS event schemas, enter a search term.

A search returns matches for both the name and contents of the available schemas, and then displays which versions of the schema contain matches.
5. Open an event schema by selecting the name of the schema.
Amazon EventBridge schema registries

Schema registries are containers for schemas. Schema registries collect and organize schemas so that your schemas are in logical groups. The default schema registries are:

- **All schemas** – All the schemas from the AWS event, discovered, and custom schema registries.
- **AWS event schema registry** – The built-in schemas.
- **Discovered schema registry** – The schemas discovered by Schema discovery.

You can create custom registries to organize the schemas you create or upload.

**To create a custom registry**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Schemas** and then choose **Create registry**.
3. On the **Registry details** page, enter a **Name**.
4. (Optional) Enter a description for your new registry.
5. Choose **Create**.

To create a custom schema (p. 231) in your new registry, select **Create custom schema**. To add a schema to your registry, select that registry when you’re creating a new schema.

To create a registry by using the API, use **CreateRegistry**. For more information, see [Amazon EventBridge Schema Registry API Reference](https://docs.aws.amazon.com/eventbridge/latest/APIReference/API_CreateRegistry.html).

Creating an Amazon EventBridge schema

You create schemas by using JSON files with either the OpenAPI Specification or the JSONSchema Draft4 specification. You can create or upload your own schemas in EventBridge by using a template or generating a schema based on the JSON of an event (p. 27). You can also infer the schema from events on an event bus (p. 9). To create a schema by using the EventBridge Schema Registry API, use the CreateSchema API action.

When you choose between OpenAPI 3 and JSONSchema Draft4 formats, consider the following differences:

- JSONSchema format supports additional keywords that aren't supported in OpenAPI, such as $schema, additionalItems.
- There are minor differences in how keywords are handled, such as type and format.
- OpenAPI doesn't support JSONSchema Hyper-Schema hyperlinks in JSON documents.
- Tools for OpenAPI tend to focus on build-time, whereas tools for JSONSchema tend to focus on run-time operations, such as client tools for schema validation.

We recommend using JSONSchema format to implement client-side validation so that events sent to EventBridge conform to the schema. You can use JSONSchema to define a contract for valid JSON documents, and then use a JSON schema validator before sending the associated events.

After you have a new schema, you can download code bindings (p. 236) to help create applications for events with that schema.

Topics
- Create a schema by using a template (p. 231)
- Edit a schema template directly in the console (p. 232)
- Create a schema from the JSON of an event (p. 233)
- Create a schema from events on an event bus (p. 235)

Create a schema by using a template

You can create a schema from a template or by editing a template directly in the EventBridge console. To get the template, you download it from the console. You can edit the template so that the schema matches your events. Then upload your new template through the console.

To download the schema template

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schema registry.
3. In the Getting started section under Schema template, choose Download.

Alternatively, you can copy the JSON template from the following code example.

```json
{
    "openapi": "3.0.0",
    "info": {
        "version": "1.0.0",
        "title": "Event"
    },
    "paths": {},
    "components": {
```
"schemas": {  
  "Event": {  
    "type": "object",  
    "properties": {  
      "ordinal": {  
        "type": "number",  
        "format": "int64"  
      },  
      "name": {  
        "type": "string"  
      },  
      "price": {  
        "type": "number",  
        "format": "double"  
      },  
      "address": {  
        "type": "string"  
      },  
      "comments": {  
        "type": "array",  
        "items": {  
          "type": "string"  
        }  
      },  
      "created_at": {  
        "type": "string",  
        "format": "date-time"  
      }  
    }  
  }  
}

To upload a schema template
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schemas and then choose Create schema.
3. (Optional) Select or create a schema registry.
4. Under Schema details, enter a name for your schema.
5. (Optional) Enter a description for your schema.
6. For Schema type, choose either OpenAPI 3.0 or JSON Schema Draft 4.
7. On the Create tab, in the text box, either drag your schema file to the text box, or paste the schema source.
8. Select Create.

Edit a schema template directly in the console

To edit a schema in the console
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schemas and then choose Create schema.
3. (Optional) Select or create a schema registry.
4. Under Schema details, enter a name for your schema.
5. For Schema type, choose either OpenAPI 3.0 or JSON Schema Draft 4.
6. (Optional) Enter a description for the schema to create.
Create a schema from the JSON of an event

If you have the JSON of an event, you can automatically create a schema for that type of event.

To create a schema based on the JSON of an event

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schemas and then choose Create schema.
3. (Optional) Select or create a schema registry.
4. Under Schema details enter a name for your schema.
5. (Optional) Enter a description for the schema you created.
6. For Schema type, choose OpenAPI 3.0.

   You can't use JSONSchema when you create a schema from the JSON of an event.
7. Select Discover from JSON
8. In the text box under JSON, paste or drag the JSON source of an event.

   For example, you could paste in the source from this AWS Step Functions event for a failed execution.

```json
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "012345678912",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-1",
    "resources": [
    ],
    "detail": {
        "name": "execution-name",
        "status": "FAILED",
        "startDate": 1551225146847,
        "stopDate": 1551225151881,
        "input": "{}",
        "output": null
    }
}
```
10. EventBridge generates an OpenAPI schema for the event. For example, the following schema is generated for the preceding Step Functions event.

```json
{
    "openapi": "3.0.0",
    "info": {
```
Create a schema from the JSON of an event

```json
"version": "1.0.0",
"title": "StepFunctionsExecutionStatusChange"
},
"paths": {},
"components": {
    "schemas": {
        "AWSEvent": {
            "type": "object",
            "required": ["detail-type", "resources", "detail", "id", "source", "time",
                          "region", "version", "account"],
            "x-amazon-events-detail-type": "Step Functions Execution Status Change",
            "x-amazon-events-source": "aws.states",
            "properties": {
                "detail": {
                    "$ref": "#/components/schemas/StepFunctionsExecutionStatusChange"
                },
                "account": {
                    "type": "string"
                },
                "detail-type": {
                    "type": "string"
                },
                "id": {
                    "type": "string"
                },
                "region": {
                    "type": "string"
                },
                "resources": {
                    "type": "array",
                    "items": {
                        "type": "string"
                    }
                },
                "source": {
                    "type": "string"
                },
                "time": {
                    "type": "string",
                    "format": "date-time"
                },
                "version": {
                    "type": "string"
                }
            }
        },
        "StepFunctionsExecutionStatusChange": {
            "type": "object",
            "required": ["output", "input", "executionArn", "name", "stateMachineArn",
                          "startDate", "stopDate", "status"],
            "properties": {
                "executionArn": {
                    "type": "string"
                },
                "input": {
                    "type": "string"
                },
                "name": {
                    "type": "string"
                },
                "output": {
                    "type": "string"
                },
                "startDate": {
                    "type": "integer",
                    "format": "int64"
                },
                "stateMachineArn": {
```

234
Create a schema from events on an event bus

EventBridge can infer schemas by discovering events. To infer schemas, you turn on event discovery on an event bus and every unique schema is added to the schema registry, including those for cross-account events. Schemas discovered by EventBridge appear in **Discovered schemas registry** on the **Schemas** page.

If the contents of events on the event bus change, EventBridge creates new versions of the related EventBridge schema.

**Note**
Enabling event discovery on an event bus can incur a cost. The first five million processed events in each month are free.

**Note**
EventBridge infers schemas from cross-account events by default but you can disable it by updating the **cross-account** property. For more information, see **Discoverers** in the EventBridge Schema Registry API Reference.

To enable schema discovery on an event bus

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose **Event buses**.
3. Do one of the following:
   - To enable discovery on the **Default event bus**, choose **Start discovery**.
   - To enable discovery on a **Custom event bus**, select the radio button for the custom event bus and then choose **Start discovery**.
Amazon EventBridge code bindings

You can generate code bindings for event schemas (p. 227) to speed up development in Golang, Java, Python, and TypeScript. Code bindings are available for AWS service events, schemas you create (p. 231), and for schemas you generate (p. 235) based on events (p. 27) on an event bus (p. 9). You can generate code bindings for a schema by using the EventBridge console, the EventBridge Schema Registry API, or in your IDE with an AWS toolkit.

To generate code bindings from an EventBridge schema

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schemas.
3. Find a schema that you want code bindings for, either by browsing the schema registries, or by searching for a schema.
4. Select the schema name.
6. On the Download code bindings page, select the language of the code bindings you want to download.
7. Select Download.

It may take a few seconds for your download to begin. The downloaded file is a zip file of code bindings for the language you selected.
Amazon EventBridge related services

Amazon EventBridge works with other AWS services to process events (p. 27) or invoke a resource as the target (p. 112) of a rule (p. 82). For more information about EventBridge integrations with other AWS services, see the following:

Topics

- Using Amazon EventBridge with Interface VPC Endpoints (p. 238)
- Amazon EventBridge integration with AWS X-Ray (p. 240)
Using Amazon EventBridge with Interface VPC Endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and EventBridge. Your resources on your VPC can use this connection to communicate with EventBridge.

With a VPC, you have control over your network settings, such as the IP address range, subnets, route tables, and network gateways. To connect your VPC to EventBridge, you define an interface VPC endpoint for EventBridge. The endpoint provides reliable, scalable connectivity to EventBridge 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, which enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see AWS PrivateLink and VPC endpoints.

When you use a private interface VPC endpoint, custom events (p. 27) your VPC sends to EventBridge use that endpoint. EventBridge then sends those events to other AWS services based on the rules (p. 82) and targets (p. 112) that you've configured. Once events are sent to another service you can receive them through either the public endpoint or a VPC endpoint for that service. For example, if you create a rule to send events to an Amazon SQS queue, you can configure an interface VPC endpoint for Amazon SQS to receive messages from that queue in your VPC without using the public endpoint.

Availability

EventBridge currently supports VPC endpoints in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Hong Kong)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Jakarta)
- Asia Pacific (Melbourne)
- Asia Pacific (Tokyo)
- Asia Pacific (Seoul)
- Asia Pacific (Osaka)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Zurich)
- Europe (Ireland)
- Europe (London)
• Europe (Milan)
• Europe (Spain)
• Europe (Paris)
• Europe (Stockholm)
• Middle East (UAE)
• Middle East (Bahrain)
• South America (São Paulo)
• Israel (Tel Aviv)

Creating a VPC Endpoint for EventBridge

To use EventBridge with your VPC, create an interface VPC endpoint for EventBridge and choose com.amazonaws.Region.events as the service name. For more information, see Creating an Interface Endpoint in the Amazon VPC User Guide.

EventBridge Pipes specifics

Full EventBridge Pipes support for Interface VPC endpoints is not available. To use the following sources within a VPC with EventBridge Pipes, see the following:

• Amazon MSK network configuration (p. 175)
• Self managed Apache Kafka network configuration (p. 180)
• Amazon MQ network configuration (p. 171)
Amazon EventBridge integration with AWS X-Ray

You can use AWS X-Ray to trace events (p. 27) that pass through EventBridge. EventBridge passes the original trace header to the target (p. 112) so that target services can track, analyze, and debug.

EventBridge can pass a trace header for an event only if the event came from a PutEvents request that passed the trace context. X-Ray doesn't trace events that originate from third-party partners, scheduled events, or AWS services (p. 35), and these event sources don't appear on your X-Ray service map.

X-Ray validates trace headers, and trace headers that aren't valid are dropped. However, the event is still processed.

**Important**
The trace header is not available on the event that's delivered to the invocation target.

- If you have an event archive (p. 147), the trace header isn't available on archived events. If you replay archived events, the trace header isn't included.
- If you have a dead-letter queue (DLQ) (p. 58), the trace header is included in the SendMessage request that sends the event to the DLQ. If you retrieve events (messages) from the DLQ by using ReceiveMessage, the trace header associated with the event is included on the Amazon SQS message attribute, but it isn't included in the event message.

For information about how an EventBridge event node connects source and target services, see Viewing source and targets in the X-Ray service map in the AWS X-Ray Developer Guide.

You can pass the following trace header information through EventBridge:

- **Default HTTP header** – The X-Ray SDK automatically populates the trace header as the X-Amzn-Trace-Id HTTP header for all invocation targets. To learn more about the default HTTP header, see Tracing header in the AWS X-Ray Developer Guide.

- **TraceHeader system attribute** – TraceHeader is a PutEventsRequestEntry attribute reserved by EventBridge to carry the X-Ray trace header to a target. If you also use PutEventsRequestEntry, PutEventsRequestEntry overrides the HTTP trace header.

**Note**
The trace header doesn't count towards the PutEventsRequestEntry event size. For more information, see Calculating Amazon EventBridge PutEvents event entry size (p. 35).
The following video demonstrates the use X-Ray and EventBridge together: [Using AWS X-Ray for tracing](#)
Amazon EventBridge tutorials

EventBridge integrates with a number of AWS services and SaaS partners. These tutorials are designed to help you get familiar with the basics of EventBridge and how it can be part of your serverless architecture.

**Tutorials:**
- [Amazon EventBridge get started tutorials](#) (p. 243)
- [Amazon EventBridge tutorials for integrating with other AWS services](#) (p. 256)
- [Amazon EventBridge tutorials for integrating with SaaS providers](#) (p. 282)
Amazon EventBridge get started tutorials

The following tutorials help you explore the features of EventBridge and how to use them.

Tutorials:
- Archive and replay Amazon EventBridge events (p. 244)
- Create an Amazon EventBridge sample application (p. 247)
- Tutorial: Download code bindings for events using the EventBridge schema registry (p. 251)
- Tutorial: Use input transformer to customize what EventBridge passes to the event target (p. 252)
Archive and replay Amazon EventBridge events

You can use EventBridge to route events to specific AWS Lambda functions using rules.

In this tutorial, you'll create a function to use as the target for the EventBridge rule using the Lambda console. Then, you'll create an archive and a rule that'll archive test events using the EventBridge console. Once there are events in that archive, you'll replay them.

**Steps:**
- **Step 1: Create a Lambda function**
- **Step 2: Create archive**
- **Step 3: Create rule**
- **Step 4: Send test events**
- **Step 5: Replay events**
- **Step 6: Clean up your resources**

**Step 1: Create a Lambda function**

First, create a Lambda function to log the events.

To create a Lambda function:
1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function.
3. Choose Author from scratch.
4. Enter a name and description for the Lambda function. For example, name the function LogScheduledEvent.
5. Leave the rest of the options as the defaults and choose Create function.
6. On the Code tab of the function page, double-click index.js. Replace the existing JavaScript code with the following code:

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

**Step 2: Create archive**

Next, create the archive that will hold all the test events.

To create an archive
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Archives.
3. Choose Create archive.
4. Enter a name and description for the archive. For example, name the archive ArchiveTest.
5. Leave the rest of the options as the defaults and choose Next.
Choose Create archive.

6. **Step 3: Create rule**

Create a rule to archive events that are sent to the event bus.

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule. For example, name the rule ARTestRule.
   
   A rule can't have the same name as another rule in the same Region and on the same event bus.
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select default. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose Other.
9. For Event pattern, enter the following:
   
   ```json
   {
     "detail-type": [
       "customerCreated"
     ]
   }
   ```
10. Choose Next.
11. For Target types, choose AWS service.
12. For Select a target, choose Lambda function from the drop-down list.
13. For Function, select the Lambda function that you created in the Step 1: Create a Lambda function section. In this example, select LogScheduledEvent.
15. Choose Next.
16. Review the details of the rule and choose Create rule.

7. **Step 4: Send test events**

Now that you've set up the archive and the rule, we'll send test events to make sure the archive is working correctly.

**Note**

It can take some time for events to get to the archive.

**To send test events (console)**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose Event buses.
3. In the Default event bus tile, choose Actions, Send events.
4. Enter an event source. For example, TestEvent.
5. For Detail type, enter customerCreated.
6. For **Event detail**, enter `{}`.
7. Choose **Send**.

### Step 5: Replay events

Once the test events are in the archive you can replay them.

#### To replay archived events (console)

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose **Replays**.
3. Choose **Start new replay**.
4. Enter a name and description for the replay. For example, name the replay `ReplayTest`.
5. For **Source**, select the archive you created in the **Step 2: Create archive** section.
6. For **Replay time frame**, do the following.
   a. For **Start time**, select the date you sent test events and a time before you sent them. For example, `2021/08/11` and `08:00:00`.
   b. For **End time**, select the current date and time. For example, `2021/08/11` and `09:15:00`.
7. Choose **Start Replay**.

### Step 6: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

#### To delete the Lambda function(s)

1. Open the **Functions page** of the Lambda console.
2. Select the function(s) that you created.
3. Choose **Actions, Delete**.
4. Choose **Delete**.

#### To delete the EventBridge archives(s)

1. Open the **Archives page** of the EventBridge console.
2. Select the archive(s) you created.
3. Choose **Delete**.
4. Enter the archive name and choose **Delete**.

#### To delete the EventBridge rule(s)

1. Open the **Rules page** of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose **Delete**.
4. Choose **Delete**.
Create an Amazon EventBridge sample application

You can use EventBridge to route events to specific Lambda functions using rules. In this tutorial, you'll use the AWS CLI, Node.js, and the code in the GitHub repo to create the following:

- An AWS Lambda function that produces events for bank ATM transactions.
- Three Lambda functions to use as targets of an EventBridge rule.
- And the rule that routes the created events to the correct downstream function based on an event pattern.

This example uses AWS SAM templates to define the EventBridge rules. To learn more about using AWS SAM templates with EventBridge see GitHub repo.

In the repo, the atmProducer subdirectory contains handler.js, which represents the ATM service producing events. This code is a Lambda handler written in Node.js, and publishes events to EventBridge via the AWS SDK using this line of JavaScript code.

```javascript
const result = await eventbridge.putEvents(params).promise()
```

This directory also contains events.js, listing several test transactions in an Entries array. A single event is defined in JavaScript as follows:

```javascript
{
// Event envelope fields
Source: 'custom.myATMapp',
EventBusName: 'default',
DetailType: 'transaction',
Time: new Date(),

// Main event body
Detail: JSON.stringify({
    action: 'withdrawal',
    location: 'MA-BOS-01',
    amount: 300,
    result: 'approved',
    transactionId: '123456',
    cardPresent: true,
    partnerBank: 'Example Bank',
    remainingFunds: 722.34
}),
}
```

The Detail section of the event specifies transaction attributes. These include the location of the ATM, the amount, the partner bank, and the result of the transaction.

The handler.js file in the atmConsumer subdirectory contains three functions:

```javascript
exports.case1Handler = async (event) => {
    console.log('--- Approved transactions ---')
    console.log(JSON.stringify(event, null, 2))
}

exports.case2Handler = async (event) => {
    console.log('--- NY location transactions ---')
    console.log(JSON.stringify(event, null, 2))
}
```
exports.case3Handler = async (event) => {
  console.log('--- Unapproved transactions ---')
  console.log(JSON.stringify(event, null, 2))
}

Each function receives transaction events, which are logged via the `console.log` statements to Amazon CloudWatch Logs. The consumer functions operate independently of the producer and are unaware of the source of the events.

The routing logic is contained in the EventBridge rules that are deployed by the application's AWS SAM template. The rules evaluate the incoming stream of events, and route matching events to the target Lambda functions.

The rules use event patterns that are JSON objects with the same structure as the events they match. Here's the event pattern for one of the rules:

```json
{
  "detail-type": ["transaction"],
  "source": ["custom.myATMapp"],
  "detail": {
    "location": [{
      "prefix": "NY-
    }]
  }
}
```

Steps:
- **Prerequisites (p. 248)**
- **Step 1: Create application (p. 248)**
- **Step 2: Run application (p. 249)**
- **Step 3: Check the logs and verify the application works (p. 249)**
- **Step 4: Clean up your resources (p. 246)**

**Prerequisites**

To complete this tutorial, you'll need the following resources:

- An AWS account. [Create an AWS account](#) if you don't already have one.
- AWS CLI installed. To install the AWS CLI, see the [Installing, updating, and uninstalling the AWS CLI version 2](#).
- Node.js 12.x installed. To install Node.js, see [Downloads](#).

**Step 1: Create application**

To set up the example application, you'll use the AWS CLI and Git to create the AWS resources you'll need.

**To create the application**

1. [Sign in to AWS](#).
2. Install Git and install the AWS Serverless Application Model CLI on your local machine.
3. Create a new directory, and then navigate to that directory in a terminal.
4. At the command line, enter `git clone https://github.com/aws-samples/amazon-eventbridge-producer-consumer-example`. 
5. At the command line run the following command:

```bash
cd ./amazon-eventbridge-producer-consumer-example
sam deploy --guided
```

6. In the terminal, do the following:
   a. For **Stack Name**, enter a name for the stack. For example, name the stack Test.
   b. For **AWS Region**, enter the Region. For example, us-west-2.
   c. For **Confirm changes before deploy**, enter Y.
   d. For **Allow SAM CLI IAM role creation**, enter Y.
   e. For **Save arguments to configuration file**, enter Y.
   f. For **SAM configuration file**, enter `samconfig.toml`.
   g. For **SAM configuration environment**, enter `default`.

**Step 2: Run application**

Now that you've set up the resources, you'll use the console to test the functions.

**To run the application**

1. Open the [Lambda console](#) in the same Region where you deployed the AWS SAM application.
2. There are four Lambda functions with the prefix `atm-demo`. Select the `atmProducerFn` function, then choose **Actions, Test**.
3. Enter Test for the **Name**.
4. Choose **Test**.

**Step 3: Check the logs and verify the application works**

Now that you've run the application, you'll use the console to check the CloudWatch Logs.

**To check the logs**

1. Open the [CloudWatch console](#) in the same Region where you ran the AWS SAM application.
2. Choose **Logs**, and then choose **Log groups**.
3. Select the log group containing `atmConsumerCase1`. You see two streams representing the two transactions approved by the ATM. Choose a log stream to view the output.
4. Navigate back to the list of log groups, and then select the log group containing `atmConsumerCase2`. You'll see two streams representing the two transactions matching the New York location filter.
5. Navigate back to the list of log groups, and select the log group containing `atmConsumerCase3`. Open the stream to see the denied transactions.

**Step 4: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge rule(s)**

1. Open the [Rules page](#) of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

To delete the Lambda function(s)
1. Open the Functions page of the Lambda console.
2. Select the function(s) that you created.
3. Choose Actions, Delete.
4. Choose Delete.

To delete the CloudWatch Logs log group(s)
1. Open the Cloudwatch console.
2. Choose Logs, Log groups.
3. Select the log group(s) that were created in this tutorial.
4. Choose Actions, Delete log group(s).
5. Choose Delete.
Tutorial: Download code bindings for events using the EventBridge schema registry

You can generate code bindings (p. 236) for event schemas (p. 227) to speed development for Golang, Java, Python, and TypeScript. You can get code bindings for existing AWS services, schemas you create, and for schemas you generate based on events (p. 27) on an event bus (p. 9). You can generate code bindings for a schema using one of the following:

- EventBridge console
- EventBridge schema registry API
- Your IDE with an AWS toolkit

In this tutorial you generate and download code bindings from an EventBridge schema for the events of an AWS service.

To generate code bindings from an EventBridge schema

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Schemas.
3. Select the AWS event schema registry tab.
4. Find the schema for the AWS service that you would like code bindings for, either by browsing through the schema registry, or by searching for a schema.
5. Select the schema name.
7. On the Download code bindings page, select the language of the code bindings you want to download.
8. Select Download.

   It may take a few seconds for your download to begin. The download file will be a .zip file of code bindings for the language you selected.

9. Unzip the downloaded file and add it to your project.

   The downloaded package contains a README file that explains how to configure the package’s dependencies in various frameworks.

Use these code bindings in your own code to help quickly build applications using this EventBridge event.
Tutorial: Use input transformer to customize what EventBridge passes to the event target

You can use the Input transformer (p. 137) in EventBridge to customize text from an event (p. 27) before you send it to the target of a rule (p. 82).

To do this, you define JSON paths from the event and assign their outputs to different variables. Then you can use those variables in the input template. The characters < and > can't be escaped. For more information, see Amazon EventBridge input transformation (p. 137)

Note
If you specify a variable to match a JSON path that doesn't exist in the event, that variable isn't created and doesn't appear in the output.

In this tutorial, you create a rule that matches an event with detail-type: "customerCreated". The input transformer maps the type variable to the $.detail-type JSON path from the event. Then EventBridge puts the variable into the input template "This event was <type>." The result is the following Amazon SNS message.

"This event was of customerCreated type."

Steps:
- Step 1: Create an Amazon SNS topic (p. 252)
- Step 2: Create an Amazon SNS subscription (p. 252)
- Step 3: Create a rule (p. 253)
- Step 4: Send test events (p. 254)
- Step 5: Confirm success (p. 254)
- Step 6: Clean up your resources (p. 246)

Step 1: Create an Amazon SNS topic

Create a topic to receive the events from EventBridge.

To create a topic
2. In the navigation pane, choose Topics.
3. Choose Create topic.
4. For Type, choose Standard.
5. Enter eventbridge-IT-test as the name of the topic.
6. Choose Create topic.

Step 2: Create an Amazon SNS subscription

Create a subscription to get emails with the transformed information.

To create a subscription
2. In the navigation pane, choose Subscriptions.
3. Choose Create subscription.
4. For **Topic ARN**, choose the topic you created in step 1. For this tutorial, choose **eventbridge-IT-test**.
5. For **Protocol**, choose **Email**.
6. For **Endpoint**, enter your email address.
7. Choose **Create subscription**.
8. Confirm the subscription by choosing **Confirm subscription** in the email you receive from AWS notifications.

**Step 3: Create a rule**

Create a rule to use the input transformer to customize the instance state information that goes to a target.

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule. For example, name the rule **ARTestRule**
5. For **Event bus**, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **default**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For **Rule type**, choose **Rule with an event pattern**.
7. Choose **Next**.
8. For **Event source**, choose **Other**.
9. For **Event pattern**, enter the following:

   ```json
   {
     "detail-type": [
       "customerCreated"
     ]
   }
   ```
10. Choose **Next**.
11. For **Target types**, choose **AWS service**.
12. For **Select a target**, choose **SNS topic** from the drop-down list.
13. For **Topic**, select the Amazon SNS topic that you created in step 1. For this tutorial, choose **eventbridge-IT-test**.
14. For **Additional settings**, do the following:
   a. For **Configure target input**, choose **Input transformer** from the drop-down list.
   b. Choose **Configure input transformer**
   c. For **Sample events**, enter the following:

   ```json
   {
     "detail-type": "customerCreated"
   }
   ```
   d. For **Target input transformer** do the following:
      i. For **Input Path**, enter the following:

   ```json
   ("detail-type":"$.detail-type")
   ```
ii. For **Input Template**, enter the following:

"This event was of <detail-type> type."

e. Choose **Confirm**.

15. Choose **Next**.
16. Choose **Next**.
17. Review the details of the rule and choose **Create rule**.

### Step 4: Send test events

Now that you've set up the SNS topic and the rule, we'll send test events to make sure the rule is working correctly.

**To send test events (console)**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Event buses**.
3. In the **Default event bus** tile, choose **Actions**, **Send events**.
4. Enter an event source. For example, **TestEvent**.
5. For **Detail type**, enter **customerCreated**.
6. For **Event detail**, enter `{}`.
7. Choose **Send**.

### Step 5: Confirm success

If you get an email from AWS notifications that matches the expected output, you've successfully completed the tutorial.

### Step 6: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the SNS topic**

1. Open the [Topics page](https://console.aws.amazon.com/sns/) of the SNS console.
2. Select the topic that you created.
3. Choose **Delete**.
4. Enter **delete me**.
5. Choose **Delete**.

**To delete the SNS subscription**

1. Open the [Subscriptions page](https://console.aws.amazon.com/sns/) of the SNS console.
2. Select the subscription that you created.
3. Choose **Delete**.
4. Choose **Delete**.
To delete the EventBridge rule(s)

1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.
Amazon EventBridge tutorials for integrating with other AWS services

Amazon EventBridge works with other AWS services to process events (p. 27) or invoke an AWS resource as the target (p. 112) of a rule (p. 82). The following tutorials show you how to integrate EventBridge with other AWS services.

Tutorials:

- Tutorial: Log the state of an Auto Scaling group using EventBridge (p. 257)
- Tutorial: Log AWS API calls using EventBridge (p. 260)
- Tutorial: Log the state of an Amazon EC2 instance using EventBridge (p. 263)
- Tutorial: Log Amazon S3 object-level operations using EventBridge (p. 266)
- Tutorial: Send events to an Amazon Kinesis stream using EventBridge and the aws.events schema (p. 269)
- Tutorial: Schedule automated Amazon EBS snapshots using EventBridge (p. 273)
- Tutorial: Send a notification when an Amazon S3 object is created (p. 275)
- Tutorial: Schedule AWS Lambda functions using EventBridge (p. 278)
Tutorial: Log the state of an Auto Scaling group using EventBridge

You can run an AWS Lambda function that logs an events (p. 27) whenever an Auto Scaling group launches or terminates an Amazon EC2 instance that indicates whether an event was successful.

For information about more scenarios that use Amazon EC2 Auto Scaling events, see Use EventBridge to handle Auto Scaling events in the Amazon EC2 Auto Scaling User Guide.

In this tutorial, you create a Lambda function, and you create a rule (p. 82) in the EventBridge console that calls that function when an Amazon EC2 Auto Scaling group launches or terminates an instance.

Steps:
- Prerequisites (p. 257)
- Step 1: Create a Lambda function (p. 257)
- Step 2: Create a rule (p. 258)
- Step 3: Test the rule (p. 258)
- Step 4: Confirm success (p. 254)
- Step 5: Clean up your resources (p. 246)

Prerequisites

To complete this tutorial, you'll need the following resources:

- An Auto Scaling group. For more information about creating one, see Creating an Auto Scaling group using a launch configuration in the Amazon EC2 Auto Scaling User Guide.

Step 1: Create a Lambda function

Create a Lambda function to log the scale-out and scale-in events for your Auto Scaling group.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function.
3. Choose Author from scratch.
4. Enter a name for the Lambda function. For example, name the function LogAutoScalingEvent.
5. Leave the rest of the options as the defaults and choose Create function.
6. On the Code tab of the function page, double-click index.js.
7. Replace the existing code with the following code.

```
'use strict';

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

8. Choose Deploy.
Step 2: Create a rule

Create a rule to run the Lambda function you created in Step 1. The rule runs when your Auto Scaling group starts or stops an instance.

To create a rule

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule. For example, name the rule TestRule.
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select default. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose AWS services.
9. For Event pattern, do the following:
   a. For Event source, select Auto Scaling from the drop-down list.
   b. For Event type, select Instance Launch and Terminate from the drop-down list.
   c. Choose Any instance event and Any group name.
10. Choose Next.
11. For Target types, choose AWS service.
12. For Select a target, choose Lambda function from the drop-down list.
13. For Function, select the Lambda function that you created in the Step 1: Create a Lambda function section. In this example, select LogAutoScalingEvent.
15. Choose Next.
16. Review the details of the rule and 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. Wait a few minutes for the scale-out event to occur, and then 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. In 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, enter 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're finished, choose Save.
2. To view the output from your Lambda function, do the following:
   b. In the navigation pane, choose Logs.
c. Select the name of the log group for your Lambda function (/aws/lambda/function-name).
d. Select the name of the log stream to view the data provided by the function for the instance that you launched.

3. (Optional) When you're finished, you can decrease the desired capacity by one so that the Auto Scaling group returns to its previous size.

Step 4: Confirm success

If you see the Lambda event in the CloudWatch logs, you've successfully completed this tutorial. If the event isn't in your CloudWatch logs, start troubleshooting by verifying the rule was created successfully and, if the rule looks correct, verify the code of your Lambda function is correct.

Step 5: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

To delete the EventBridge rule(s)

1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

To delete the Lambda function(s)

1. Open the Functions page of the Lambda console.
2. Select the function(s) that you created.
3. Choose Actions, Delete.
4. Choose Delete.
Tutorial: Log AWS API calls using EventBridge

You can use Amazon EventBridge rules (p. 82) to react to API calls made by an AWS service that are recorded by AWS CloudTrail.

In this tutorial, you create an AWS CloudTrail trail, a Lambda function, and a rule in the EventBridge console. The rule invokes the Lambda function when an Amazon EC2 instance is stopped.

Steps:
- Step 1: Create an AWS CloudTrail trail (p. 260)
- Step 2: Create an AWS Lambda function (p. 260)
- Step 3: Create a rule (p. 261)
- Step 4: Test the rule (p. 261)
- Step 5: Confirm success (p. 254)
- Step 6: Clean up your resources (p. 246)

Step 1: Create an AWS CloudTrail trail

If you already have a trail set up, skip to step 2.

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.
5. For AWS KMS alias, type an alias for the KMS key.
6. Choose Next.
7. Choose Next.
8. Choose Create trail.

Step 2: Create an AWS Lambda function

Create a Lambda function to log the API call events.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function.
3. Choose Author from scratch.
4. Enter a name and description for the Lambda function. For example, name the function LogEC2StopInstance.
5. Leave the rest of the options as the defaults and choose Create function.
6. On the Code tab of the function page, double-click index.js.
7. Replace the existing code with the following code.

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

8. Choose **Deploy**.

### Step 3: Create a rule

Create a rule to run the Lambda function you created in step 2 whenever you stop an Amazon EC2 instance.

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule. For example, name the rule **TestRule**.
5. For **Event bus**, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **default**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For **Rule type**, choose **Rule with an event pattern**.
7. Choose **Next**.
8. For **Event source**, choose **AWS services**.
9. For **Event pattern**, do the following:
   a. For **Event source**, select **EC2** from the drop-down list.
   b. For **Event type**, select **AWS API Call via CloudTrail** from the drop-down list.
   c. Choose **Specific operation(s)** and enter **StopInstances**.
10. Choose **Next**.
11. For **Target types**, choose **AWS service**.
12. For **Select a target**, choose **Lambda function** from the drop-down list.
13. For **Function**, select the Lambda function that you created in the **Step 1: Create a Lambda function** section. In this example, select **LogEC2StopInstance**.
14. Choose **Next**.
15. Choose **Next**.
16. Review the details of the rule and choose **Create rule**.

### Step 4: Test the rule

You can test your rule by stopping an Amazon EC2 instance using the Amazon EC2 console. Wait a few minutes for the instance to stop, and then check your AWS Lambda metrics on the CloudWatch console to verify that your function ran.

**To test your rule by stopping an instance**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](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. To view the output from your Lambda function, do the following:
b. In the navigation pane, choose Logs.
c. Select the name of the log group for your Lambda function (/aws/lambda/function-name).
d. Select the name of the log stream to view the data provided by the function for the instance that you stopped.
5. (Optional) When you're finished, terminate the stopped instance. For more information, see Terminate Your Instance in the Amazon EC2 User Guide for Linux Instances.

Step 5: Confirm success

If you see the Lambda event in the CloudWatch logs, you've successfully completed this tutorial. If the event isn't in your CloudWatch logs, start troubleshooting by verifying the rule was created successfully and, if the rule looks correct, verify the code of your Lambda function is correct.

Step 6: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

To delete the EventBridge rule(s)

1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

To delete the Lambda function(s)

1. Open the Functions page of the Lambda console.
2. Select the function(s) that you created.
3. Choose Actions, Delete.
4. Choose Delete.

To delete the CloudTrail trail(s)

1. Open the Trails page of the CloudTrail console.
2. Select the trail(s) that you created.
3. Choose Delete.
4. Choose Delete.
Tutorial: Log the state of an Amazon EC2 instance using EventBridge

You can create an AWS Lambda function that logs a state change for an Amazon EC2 instance. Then you can create a rule (p. 82) that runs your Lambda 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.

Steps:
- Step 1: Create an AWS Lambda function (p. 263)
- Step 2: Create a rule (p. 263)
- Step 3: Test the rule (p. 261)
- Step 4: Confirm success (p. 254)
- Step 5: Clean up your resources (p. 246)

Step 1: Create an AWS Lambda function

Create a Lambda function to log the state change events (p. 27). When you create your rule in Step 2, you specify this function.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function.
3. Choose Author from scratch.
4. Enter a name and description for the Lambda function. For example, name the function LogEC2InstanceStateChange.
5. Leave the rest of the options as the defaults and choose Create function.
6. On the Code tab of the function page, double-click index.js.
7. Replace the existing code with the following code.

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

Step 2: Create a rule

Create a rule to run the Lambda function you created in Step 1. The rule runs when you launch an Amazon EC2 instance.

To create the EventBridge rule

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule. For example, name the rule TestRule
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select default. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose AWS services.
9. For Event pattern, do the following:
   a. For Event source, select EC2 from the drop-down list.
   b. For Event type, choose EC2 Instance State-change Notification from the drop-down list.
   c. Choose Specific states(s) and choose running from the drop-down list.
   d. Choose Any instance
10. Choose Next.
11. For Target types, choose AWS service.
12. For Select a target, choose Lambda function from the drop-down list.
13. For Function, select the Lambda function that you created in the Step 1: Create a Lambda function section. In this example, select LogEC2InstanceStateChange.
15. Choose Next.
16. Review the details of the rule and choose Create rule.

Step 3: Test the rule

You can test your rule by stopping an Amazon EC2 instance using the Amazon EC2 console. Wait a few minutes for the instance to stop, and then check your AWS Lambda metrics on the CloudWatch console to verify that your function ran.

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. To view the output from your Lambda function, do the following:
   b. In the navigation pane, choose Logs.
   c. Select the name of the log group for your Lambda function (/aws/lambda/function-name).
   d. Select the name of the log stream to view the data provided by the function for the instance that you stopped.
5. (Optional) When you're finished, terminate the stopped instance. For more information, see Terminate Your Instance in the Amazon EC2 User Guide for Linux Instances.

Step 4: Confirm success

If you see the Lambda event in the CloudWatch logs, you've successfully completed this tutorial. If the event isn't in your CloudWatch logs, start troubleshooting by verifying the rule was created successfully and, if the rule looks correct, verify the code of your Lambda function is correct.
Step 5: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

To delete the EventBridge rule(s)
1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

To delete the Lambda function(s)
1. Open the Functions page of the Lambda console.
2. Select the function(s) that you created.
3. Choose Actions, Delete.
4. Choose Delete.
Tutorial: Log Amazon S3 object-level operations using EventBridge

You can log the object-level API operations on your Amazon S3 buckets. Before Amazon EventBridge can match these events (p. 27), you must use AWS CloudTrail to set up and configure a trail to receive these events.

In this tutorial, you create CloudTrail trail, create a AWS Lambda function, and then create rule (p. 82) in the EventBridge console that invokes that function in response to an S3 data event.

Steps:

- **Step 1: Configure your AWS CloudTrail trail (p. 266)**
- **Step 2: Create an AWS Lambda function (p. 266)**
- **Step 3: Create a Rule (p. 267)**
- **Step 4: Test the Rule (p. 268)**
- **Step 5: Confirm success (p. 254)**
- **Step 6: Clean up your resources (p. 246)**

Step 1: Configure your AWS CloudTrail trail

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

For more information, see Data Events in the AWS CloudTrail User Guide.

To create a trail

1. Open the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. Choose Trails, Create trail.
3. For Trail name, type a name for the trail.
4. For Storage location, in Create a new S3 bucket.
5. For AWS KMS alias, type an alias for the KMS key.
6. Choose Next.
7. For Event type, choose Data events
8. For Data events, do one of the following:
   - 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 in a bucket, specify an S3 bucket and 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.
9. For each resource, choose whether to log Read events, Write events, or both.
10. Choose Next.
11. Choose Create trail.

Step 2: Create an AWS Lambda function

Create a Lambda function to log data events for your S3 buckets.
To create a Lambda function
1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function.
3. Choose Author from scratch.
4. Enter a name and description for the Lambda function. For example, name the function LogS3DataEvents.
5. Leave the rest of the options as the defaults and choose Create function.
6. On the Code tab of the function page, double-click index.js.
7. Replace the existing code with the following code.

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

Step 3: Create a Rule
Create a rule to run the Lambda function you created in Step 2. This rule runs in response to an Amazon S3 data event.

To create a rule
1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule. For example, name the rule TestRule
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select default. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose AWS services.
9. For Event pattern, do the following:
   a. For Event source, select Simple Storage Service (S3) from the drop-down list.
   b. For Event type, select Object-Level API call via CloudTrail from the drop-down list.
   c. Choose Specific operation(s), and then choose PutObject.
   d. 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 enter one or more buckets.
10. Choose Next.
11. For Target types, choose AWS service.
12. For Select a target, choose Lambda function from the drop-down list.
13. For Function, select the LogS3DataEvents Lambda function that you created in step 1.
15. Choose Next.
16. Review the details of the rule and 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**

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 the log stream to view the data provided by the function for the instance that you launched.

You can also check your CloudTrail logs in the S3 bucket that you specified for your trail. For more information, see [Getting and Viewing Your CloudTrail Log Files](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-logs-viewing.html) in the AWS CloudTrail User Guide.

**Step 5: Confirm success**

If you see the Lambda event in the CloudWatch logs, you've successfully completed this tutorial. If the event isn't in your CloudWatch logs, start troubleshooting by verifying the rule was created successfully and, if the rule looks correct, verify the code of your Lambda function is correct.

**Step 6: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge rule(s)**

1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

**To delete the Lambda function(s)**

1. Open the Functions page of the Lambda console.
2. Select the function(s) that you created.
3. Choose Actions, Delete.
4. Choose Delete.

**To delete the CloudTrail trail(s)**

1. Open the Trails page of the CloudTrail console.
2. Select the trail(s) that you created.
3. Choose Delete.
4. Choose Delete.
Tutorial: Send events to an Amazon Kinesis stream using EventBridge and the aws.events schema

You can send AWS API call [events](p. 27) in EventBridge to an Amazon Kinesis stream, create Kinesis Data Streams applications, and process large amounts of data. In this tutorial, you create a Kinesis stream, and then create a [rule](p. 82) in the EventBridge console that sends events to that stream when an Amazon EC2 instance stops.

Steps:

- **Prerequisites** (p. 269)
- **Step 1: Create an Amazon Kinesis stream** (p. 269)
- **Step 2: Create a rule** (p. 269)
- **Step 3: Test the rule** (p. 271)
- **Step 4: Verify that the event was sent** (p. 271)
- **Step 5: Clean up your resources** (p. 246)

Prerequisites

In this tutorial, you'll use the following:

- Use the AWS CLI to work with Kinesis streams.

To install the AWS CLI, see the [Installing, updating, and uninstalling the AWS CLI version 2](#).

**Note**

This tutorial uses AWS events and the built in aws.events schema registry. You can also create an EventBridge rule based on the schema of your custom events by adding them to a custom schema registry manually, or by using schema discovery. For more information on schemas, see [??](p. 227). For more information on creating a rule using other event pattern options, see [??](p. 84).

Step 1: Create an Amazon Kinesis stream

To create a stream, at a command prompt, use the `create-stream` AWS CLI command.

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

When the stream status is ACTIVE, the stream is ready. To check the stream status, use the `describe-stream` command.

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

Step 2: Create a rule

Create a rule to send events to your stream when you stop an Amazon EC2 instance.

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule. For example, name the rule **TestRule**.
5. For **Event bus**, select **default**.
6. For **Rule type**, choose **Rule with an event pattern**.
7. Choose **Next**.
8. For **Event source**, choose **AWS events** or **EventBridge partner events**.
9. For **Creation method**, choose **Use schema**.
10. For **Event pattern**, do the following:
    a. For **Schema type**, choose **Select schema from Schema registry**.
    b. For **Schema registry**, choose **aws.events** from the drop-down list.
    c. For **Schema**, choose **aws.ec2@EC2InstanceStateChangeNotification** from the drop-down list.

    EventBridge displays the event schema under **Models**.

    EventBridge displays a red asterisk next to any properties that are required for the event, not for the event pattern.
    d. In **Models**, set the following event filter properties:
        i. Select + **Edit** next to the **state** property.

            Leave **Relationship** empty. For **Value**, enter **running**. Choose **Set**.
        ii. Select + **Edit** next to the **source** property.

            Leave **Relationship** empty. For **Value**, enter **aws.ec2**. Choose **Set**.
        iii. Select + **Edit** next to the **detail-type** property.

            Leave **Relationship** empty. For **Value**, enter **EC2 Instance State-change Notification**. Choose **Set**.
    e. To view the event pattern you've constructed, choose **Generate event pattern in JSON**.

    EventBridge displays the event pattern in JSON:

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

11. Choose **Next**.
12. For **Target types**, choose **AWS service**.
13. For **Select a target**, choose **Kinesis stream** from the drop-down list.
14. For **Stream**, select the Kinesis stream that you created in the **Step 1: Create an Amazon Kinesis stream** section. In this example, select **test**.
15. For **Execution role**, choose **Create a new for role for this specific resource**.
16. Choose **Next**.
17. Choose **Next**.
18. Review the details of the rule and choose **Create rule**.
Step 3: Test the rule

To test your rule, stop an Amazon EC2 instance. Wait a few minutes for the instance to stop, and then check your CloudWatch metrics to verify that your function ran.

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 Amazon EventBridge console at https://console.aws.amazon.com/events/.
4. In the navigation pane, choose Rules.
   - Choose the name of the rule that you created and choose Metrics for the rule.
5. (Optional) When you're finished, 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 was sent

You can use the AWS CLI to get the record from the stream to verify that the event was sent.

To get the record

1. To start reading from your Kinesis stream, at a command prompt, use the get-shard-iterator command.

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

   The following is example output.

   ```json
   {
     "ShardIterator": "AAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6A3J9ZG41NR1EMi+9Md/nHvtLyxfhEzYy9kT4QD9QVz/mBYWR0607ZTkw9gd+efGN2aHFDkhH1r34BL9Wy+ghYG22D2T1da2EyNSh1+LABK33gQweTJADByMwlo5r6PqcP2dzhg=
   }```

2. To get the record, use the following get-records command. Use the shard iterator from the output in the previous step.

   ```bash
   aws kinesis get-records --shard-iterator AAAAAbAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6A3J9ZG41NR1EMi+9Md/nHvtLyxfhEzYy9kT4QD9QVz/mBYWR0607ZTkw9gd+efGN2aHFDkhH1r34BL9Wy+ghYG22D2T1da2EyNSh1+LABK33gQweTJADByMwlo5r6PqcP2dzhg=
   ```

   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 that you expect, keep calling get-records.

3. Records in Kinesis are encoded in Base64. Use a Base64 decoder to decode the data so that you can verify that it's the event that was sent to the stream in JSON form.
Step 5: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

To delete the EventBridge rule(s)
1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

To delete the Kinesis stream(s)
1. Open the Data streams page of the Kinesis console.
2. Select the stream(s) that you created.
3. Choose Actions, Delete.
4. Enter delete in the field and choose Delete.
Tutorial: Schedule automated Amazon EBS snapshots using EventBridge

You can run EventBridge rules (p. 82) on a schedule. In this tutorial, you create a snapshot of an existing Amazon Elastic Block Store (Amazon EBS) volume on a schedule. You can choose a fixed rate to create a snapshot every few minutes or use a cron expression to create the snapshot at a specific time of day.

**Important**
To create rules with built-in targets (p. 112), you must use the AWS Management Console.

**Steps:**
- Step 1: Create the rule (p. 273)
- Step 2: Test the rule (p. 273)
- Step 3: Confirm success (p. 254)
- Step 4: Clean up your resources (p. 246)

### Step 1: Create the 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 Creating an Amazon EventBridge rule that runs on a schedule (p. 95).

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule.
   A rule can't have the same name as another rule in the same Region and on the same event bus.
5. For **Event bus**, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **AWS default event bus**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For **Rule type**, choose **Schedule**.
7. Choose **Next**.
8. For **Schedule pattern**, choose **A schedule that runs at a regular rate, such as every 10 minutes.** and enter **5** and choose **Minutes** from the drop-down list.
9. Choose **Next**.
10. For **Target types**, choose **AWS service**.
11. For **Select a target**, choose **EBS Create Snapshot** from the drop-down list.
12. For **Volume ID**, enter the volume ID of the Amazon EBS volume.
13. For **Execution role**, choose **Create a new for role for this specific resource.**
14. Choose **Next**.
15. Choose **Next**.
16. Review the details of the rule and choose **Create rule**.

### Step 2: Test the rule

You can verify your rule works by viewing your first snapshot after it's taken.
To test your rule

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](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.

**Step 3: Confirm success**

If you see the a snapshot in the list, you've successfully completed this tutorial. If the snapshot isn't in the list, start troubleshooting by verifying the rule was created successfully.

**Step 4: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge rule(s)**

1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.
Tutorial: Send a notification when an Amazon S3 object is created

You can send email notifications when Amazon Simple Storage Service (Amazon S3) objects are created using Amazon EventBridge and Amazon SNS. In this tutorial, you will create an SNS topic and subscription. Then, you will create a rule in the EventBridge console that sends events to that topic when Amazon S3 Object Created events are received.

Steps:
- Prerequisites (p. 275)
- Step 1: Create an Amazon SNS topic (p. 275)
- Step 2: Create an Amazon SNS subscription (p. 275)
- Step 3: Create a rule (p. 276)
- Step 4: Test the rule (p. 276)
- Step 5: Clean up your resources (p. 246)

Prerequisites

To receive Amazon S3 events in EventBridge, you must enable EventBridge in the Amazon S3 console. This tutorial assumes EventBridge is enabled. For more information, see Enabling Amazon EventBridge in the S3 console.

Step 1: Create an Amazon SNS topic

Create a topic to receive the events from EventBridge.

To create a topic

2. In the navigation pane, choose Topics.
3. Choose Create topic.
4. For Type, choose Standard.
5. Enter eventbridge-test as the name of the topic.
6. Choose Create topic.

Step 2: Create an Amazon SNS subscription

Create a subscription to get email notifications from Amazon S3 when events are received by the topic.

To create a subscription

2. In the navigation pane, choose Subscriptions.
3. Choose Create subscription.
4. For Topic ARN, choose the topic you created in step 1. For this tutorial, choose eventbridge-test.
5. For Protocol, choose Email.
6. For Endpoint, enter your email address.
7. Choose Create subscription.
8. Confirm the subscription by choosing **Confirm subscription** in the email you receive from AWS notifications.

**Step 3: Create a rule**

Create a rule to send events to your topic when an Amazon S3 object is created.

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule. For example, name the rule `s3-test`.
5. For **Event bus**, select **default**.
6. For **Rule type**, choose **Rule with an event pattern**.
7. Choose **Next**.
8. For **Event source**, choose **AWS events or EventBridge partner events**.
9. For **Creation method**, choose **Use pattern form**.
10. For **Event pattern**, do the following:
    a. For **Event source**, select **AWS services** from the drop-down list.
    b. For **AWS service**, select **Simple Storage Service (S3)** from the drop-down list.
    c. For **Event type**, choose **Amazon S3 Event Notification** from the drop-down list.
    d. Choose **Specific events(s)** and choose **Object Created** from the drop-down list.
    e. Choose **Any bucket**
11. Choose **Next**.
12. For **Target types**, choose **AWS service**.
13. For **Select a target**, choose **SNS topic** from the drop-down list.
14. For **Topic**, select the Amazon SNS topic that you created in the **Step 1: Create an SNS topic** section. In this example, select `eventbridge-test`.
15. Choose **Next**.
16. Choose **Next**.
17. Review the details of the rule and choose **Create rule**.

**Step 4: Test the rule**

To test your rule, create an Amazon S3 object by uploading a file to an EventBridge-enabled bucket. Then, wait a few minutes and verify if you receive an email from AWS notifications.

**Step 5: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the SNS topic**

1. Open the [Topics page](https://console.aws.amazon.com/sns/) of the SNS console.
2. Select the topic that you created.
3. Choose **Delete**.
4. Enter `delete me`.
5. Choose `Delete`.

**To delete the SNS subscription**

1. Open the [Subscriptions page](#) of the SNS console.
2. Select the subscription that you created.
3. Choose `Delete`.
4. Choose `Delete`.

**To delete the EventBridge rule(s)**

1. Open the [Rules page](#) of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose `Delete`.
4. Choose `Delete`.
Tutorial: Schedule AWS Lambda functions using EventBridge

You can set up a rule (p. 82) 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 want to use the AWS CLI but haven't installed it, see the Installing, updating, and uninstalling the AWS CLI version 2.

For schedules, EventBridge doesn't provide second-level precision in schedule expressions (p. 95). The finest resolution using a cron expression is one minute. Due to the distributed nature of EventBridge and the target services, there can be a delay of several seconds between the time the scheduled rule is triggered and the time the target service runs the target resource.

Steps:
- Step 1: Create a Lambda function (p. 244)
- Step 2: Create a Rule (p. 278)
- Step 3: Verify the rule (p. 280)
- Step 4: Confirm success (p. 254)
- Step 5: Clean up your resources (p. 246)

Step 1: Create a Lambda function

Create a Lambda function to log the scheduled events.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function.
3. Choose Author from scratch.
4. Enter a name and description for the Lambda function. For example, name the function LogScheduledEvent.
5. Leave the rest of the options as the defaults and choose Create function.
6. On the Code tab of the function page, double-click index.js.
7. Replace the existing code with the following code.

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

Step 2: Create a Rule

Create a rule to run the Lambda function you created in step 1 on a schedule.

You can use either the console or the AWS CLI to create the rule. To use the AWS CLI, you first 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 (console)

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule.
   A rule can't have the same name as another rule in the same Region and on the same event bus.
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select AWS default event bus. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Schedule.
7. Choose Next.
8. For Schedule pattern, choose A schedule that runs at a regular rate, such as every 10 minutes.
   and enter 5 and choose Minutes from the drop-down list.
9. Choose Next.
10. For Target types, choose AWS service.
11. For Select a target, choose Lambda function from the drop-down list.
12. For Function, select the Lambda function that you created in the Step 1: Create a Lambda function section. In this example, select LogScheduledEvent.
13. Choose Next.
15. Review the details of the rule and choose Create rule.

To create a rule (AWS CLI)

1. To create a rule that runs on a schedule, use the put-rule command.

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

   When this rule runs, it creates an event and then sends it to the targets. 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. To grant the EventBridge service principal (events.amazonaws.com) permission to run the rule, use the add-permission command.

   ```
   aws lambda add-permission \
   --function-name LogScheduledEvent \
   ```
3. Create the file targets.json with the following contents.

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

4. To add the Lambda function that you created in step 1 to the rule, use the put-targets command.

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

**Step 3: Verify the rule**

Wait at least five minutes after completing step 2, and then you can verify that your Lambda function was invoked.

**View the output from your Lambda function**

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 the log stream to view the data provided by the function for the instance that you launched.

**Step 4: Confirm success**

If you see the Lambda event in the CloudWatch logs, you've successfully completed this tutorial. If the event isn't in your CloudWatch logs, start troubleshooting by verifying the rule was created successfully and, if the rule looks correct, verify the code of your Lambda function is correct.

**Step 5: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge rule(s)**

1. Open the Rules page of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose Delete.
4. Choose Delete.

**To delete the Lambda function(s)**

1. Open the Functions page of the Lambda console.
2. Select the function(s) that you created.
3. Choose Actions, Delete.
4. Choose Delete.
Amazon EventBridge tutorials for integrating with SaaS providers

EventBridge can work directly with SaaS partner applications and services to send and receive events (p. 27). The following tutorials show you how to integrate EventBridge with SaaS partners.

**Tutorials:**
- Tutorial: Create a connection to Datadog as an API destination (p. 283)
- Tutorial: Create a connection to Salesforce as an API destination (p. 286)
- Tutorial: Create a connection to Zendesk as an API destination (p. 290)
Tutorial: Create a connection to Datadog as an API destination

You can use EventBridge to route events to third-party services, such as Datadog. In this tutorial, you'll use the EventBridge console to create a connection to Datadog, an API destination that points to Datadog, and a rule to route events to Datadog.

Steps:

- **Prerequisites**
- **Step 1: Create connection**
- **Step 2: Create API destination**
- **Step 3: Create rule**
- **Step 4: Test the rule**
- **Step 5: Clean up your resources**

Prerequisites

To complete this tutorial, you'll need the following resources:

- A Datadog account.
- A Datadog API key.
- An EventBridge-enabled Amazon Simple Storage Service (Amazon S3) bucket.

Step 1: Create connection

To send events to Datadog, you'll first have to establish a connection to the Datadog API.

To create the connection

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose API destinations.
3. Choose the Connections tab, and then choose Create connection.
4. Enter a name and description for the connection. For example, enter Datadog as a name, and Datadog API Connection as a description.
5. For Authorization type, choose API key.
6. For API key name, enter DD-API-KEY.
7. For Value, paste your Datadog secret API key.
8. Choose Create.

Step 2: Create API destination

Now that you've created the connection, next you'll create the API destination to use as the target of the rule.

To create the API Destination

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose API destinations.
3. Choose Create API destination.
4. Enter a name and description for the API destination. For example, enter DatadogAD for the name, and Datadog API Destination for the description.
5. For API destination endpoint, enter https://http-intake.logs.datadoghq.com/api/v2/logs.
6. For HTTP method, choose POST.
7. For Invocation rate limit, enter 300.
8. For Connection, choose Use an existing connection and choose the Datadog connection you created in step 1.
9. Choose Create.

**Step 3: Create rule**

Next, you'll create a rule to send events to Datadog when an Amazon S3 object is created.

**To create a rule**

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule. For example, enter DatadogRule for the name, and Rule to send events to Datadog for S3 object creation for the description.
5. For Event bus, choose default.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose Other.
9. For Event pattern, enter the following:

   ```json
   {
     "source": ["aws.s3"]
   }
   ```
10. Choose Next.
11. For Target types, choose EventBridge API destination.
12. For API destination, choose Use an existing API destination, and then choose the DatadogAD destination you created in step 2.
13. For Execution role, choose Create a new for role for this specific resource.
14. For Additional settings, do the following:
   a. For Configure target input, choose Input transformer from the drop-down list.
   b. Choose Configure input transformer
   c. For Sample events, enter the following:

      ```json
      {
        "detail":[]
      }
      ```
   d. For Target input transformer do the following:
      i. For Input Path, enter the following:
Create a connection to Datadog

For **Input Template**, enter the following:

```
{"message": "$detail"}
```

Choose **Confirm**.

15. Choose **Next**.
16. Choose **Next**.
17. Review the details of the rule and choose **Create rule**.

**Step 4: Test the rule**

To test your rule, create an Amazon S3 object by uploading a file to an EventBridge-enabled bucket. The created object will be logged in the Datadog Logs console.

**Step 5: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge Connections(s)**

1. Open the **API destination page** of the EventBridge console.
2. Choose the **Connections** tab.
3. Select the Connection(s) you created.
4. Choose **Delete**.
5. Enter the name of the connection and choose **Delete**.

**To delete the EventBridge API destination(s)**

1. Open the **API destination page** of the EventBridge console.
2. Select the API destinations(s) you created.
3. Choose **Delete**.
4. Enter the name of the API destination and choose **Delete**.

**To delete the EventBridge rule(s)**

1. Open the **Rules page** of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose **Delete**.
4. Choose **Delete**.
Tutorial: Create a connection to Salesforce as an API destination

You can use EventBridge to route events (p. 27) to third-party services, such as Salesforce.

In this tutorial, you'll use the EventBridge console to create a connection to Salesforce, an API destination (p. 117) that points to Salesforce, and a rule (p. 82) to route events to Salesforce.

Steps:
- Prerequisites (p. 286)
- Step 1: Create connection (p. 286)
- Step 2: Create API destination (p. 283)
- Step 3: Create rule (p. 284)
- Step 4: Test the rule (p. 285)
- Step 5: Clean up your resources (p. 246)

Prerequisites

To complete this tutorial, you'll need the following resources:

- A Salesforce account.
- A Salesforce connected app.
- A Salesforce security token.
- A Salesforce custom platform event.
- An EventBridge-enabled Amazon Simple Storage Service (Amazon S3) bucket.

Step 1: Create connection

To send events to Salesforce, you'll first have to establish a connection to the Salesforce API.

To create the connection

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose API destinations.
3. Choose the Connections tab, and then choose Create connection.
4. Enter a name and description for the connection. For example, enter Salesforce as a name, and Salesforce API Connection as a description.
5. For Destination type, choose Partners and for Partner Destinations, select Salesforce from the drop-down list.
6. For Authorization endpoint, enter one of these:
   - If you're using a production org, enter https://MyDomainName.my.salesforce.com/services/oauth2/token
   - If you're using a sandbox without enhanced domains, enter https://MyDomainName--SandboxName.my.salesforce.com/services/oauth2/token
   - If you're using a sandbox with enhanced domains, enter https://MyDomainName--SandboxName.sandbox.my.salesforce.com/services/oauth2/token
7. For HTTP method, choose POST from the drop-down list.
8. For **Client ID**, enter the client ID from your Salesforce connected app.
9. For **Client secret**, enter the client secret from your Salesforce connected app.
10. For **OAuth Http Parameters**, enter the following key/value pairs:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_type</td>
<td>password</td>
</tr>
<tr>
<td>username</td>
<td>Your Salesforce username</td>
</tr>
<tr>
<td>password</td>
<td>Your Salesforce password with your security token appended to it.</td>
</tr>
</tbody>
</table>

11. Choose **Create**.

**Step 2: Create API destination**

Now that you've created the connection, next you'll create the API destination to use as the **target** (p. 112) of the rule.

**To create the API Destination**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **API destinations**.
3. Choose **Create API destination**.
4. Enter a name and description for the API destination. For example, enter **SalesforceAD** for the name, and **Salesforce API Destination** for the description.
5. For **API destination endpoint**, enter `https://MyDomainName.my.salesforce.com/services/data/v54.0/sobjects/MyEvent__e` where `Myevent__e` is the platform event where you want to send information.
6. For **HTTP method**, choose **POST** from the drop-down list.
7. For **Invocation rate limit**, enter **300**.
8. For **Connection**, choose **Use an existing connection** and choose the Salesforce connection you created in step 1.
9. Choose **Create**.

**Step 3: Create rule**

Next, you'll create a rule to send events to Salesforce when an Amazon S3 object is created.

**To create a rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule. For example, enter **SalesforceRule** for the name, and **Rule to send events to Salesforce for S3 object creation** for the description.
5. For **Event bus**, choose **default**.
6. For **Rule type**, choose **Rule with an event pattern**.
7. Choose **Next**.
8. For **Event source**, choose **Other**.
9. For **Event pattern**, enter the following:

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

10. Choose **Next**.

11. For **Target types**, choose **EventBridge API destination**.

12. For **API destination**, choose **Use an existing API destination**, and then choose the SalesforceAD destination you created in step 2.

13. For **Execution role**, choose **Create a new for role for this specific resource**.

14. For **Additional settings**, do the following:

   a. For **Configure target input**, choose **Input transformer** from the drop-down list.
   b. Choose **Configure input transformer**
   c. For **Sample events**, enter the following:

   ```json
   {
   "detail":[]
   }
   ```
   d. For **Target input transformer** do the following:

   i. For **Input Path**, enter the following:

   ```json
   {"detail":"$.detail"}
   ```
   ii. For **Input Template**, enter the following:

   ```json
   {"message": <detail>}
   ```
   e. Choose **Confirm.**

15. Choose **Next**.

16. Choose **Next**.

17. Review the details of the rule and choose **Create rule**.

---

**Step 4: Test the rule**

To test your rule, create an **Amazon S3 object** by uploading a file to an EventBridge-enabled bucket. The information about the created object will be sent to the Salesforce platform event.

**Step 5: Clean up your resources**

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge Connections(s)**

1. Open the **API destination page** of the EventBridge console.
2. Choose the **Connections** tab.
3. Select the Connection(s) you created.
4. Choose **Delete**.
5. Enter the name of the connection and choose **Delete**.

**To delete the EventBridge API destination(s)**

1. Open the [API destination page](#) of the EventBridge console.
2. Select the API destinations(s) you created.
3. Choose **Delete**.
4. Enter the name of the API destination and choose **Delete**.

**To delete the EventBridge rule(s)**

1. Open the [Rules page](#) of the EventBridge console.
2. Select the rule(s) that you created.
3. Choose **Delete**.
4. Choose **Delete**.
Tutorial: Create a connection to Zendesk as an API destination

You can use EventBridge to route events (p. 27) to third-party services like Zendesk.

In this tutorial, you'll use the EventBridge console to create a connection to Zendesk, an API destination (p. 117) that points to Zendesk, and a rule (p. 82) to route events to Zendesk.

Steps:

- Prerequisites (p. 290)
- Step 1: Create connection (p. 290)
- Step 2: Create API destination (p. 290)
- Step 3: Create rule (p. 291)
- Step 4: Test the rule (p. 292)
- Step 5: Clean up your resources (p. 246)

Prerequisites

To complete this tutorial, you'll need the following resources:

- A Zendesk account.
- An EventBridge-enabled Amazon Simple Storage Service (Amazon S3) bucket.

Step 1: Create connection

To send events to Zendesk, you'll first have to establish a connection to the Zendesk API.

To create the connection

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose API destinations.
3. Choose the Connections tab, and then choose Create connection.
4. Enter a name and description for the connection. For example, enter Zendesk for the name, and Connection to Zendesk API for the description.
5. For Authorization type, choose Basic (Username/Password).
6. For Username, enter your Zendesk username.
7. For Password, enter your Zendesk password.
8. Choose Create.

Step 2: Create API destination

Now that you've created the connection, you'll next create the API destination to use as the target (p. 112) of the rule.

To create the API Destination

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose API destinations.
3. Choose Create API destination.
4. Enter a name and description for the API destination. For example, enter ZendeskAD for the name, and Zendesk API destination for the description.

5. For API destination endpoint, enter https://your-subdomain.zendesk.com/api/v2/tickets.json, where your-subdomain is the subdomain associated with your Zendesk account.

6. For HTTP method, choose POST.

7. For Invocation rate limit, enter 10.

8. For Connection, choose Use an existing connection and choose the Zendesk connection you created in step 1.

9. Choose Create.

**Step 3: Create rule**

Next, create a rule to send events to Zendesk when an Amazon S3 object is created.

**To create a rule**

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.

2. In the navigation pane, choose Rules.

3. Choose Create rule.

4. Enter a name and description for the rule. For example, enter ZendeskRule for the name, and **Rule to send events to Zendesk when S3 objects are created** for the description.

5. For Event bus, choose default.

6. For Rule type, choose Rule with an event pattern.

7. Choose Next.

8. For Event source, choose Other.

9. For Event pattern, enter the following:

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

10. Choose Next.

11. For Target types, choose EventBridge API destination.

12. For API destination, choose Use an existing API destination, and then choose the ZendeskAD destination you created in step 2.

13. For Execution role, choose Create a new for role for this specific resource.

14. For Additional settings, do the following:

   a. For Configure target input, choose Input transformer from the drop-down list.
   b. Choose Configure input transformer
   c. For Sample events, enter the following:

   ```json
   {
    "detail":[]
   }
   ```
   d. For Target input transformer do the following:
   i. For Input Path, enter the following:

   ```json
   {"detail":"$.detail"}
   ```
ii. For **Input Template**, enter the following:

```json
"message": <detail>
```

e. Choose **Confirm**.

15. Choose **Next**.

16. Choose **Next**.

17. Review the details of the rule and choose **Create rule**.

### Step 4: Test the rule

To test your rule, create an [Amazon S3 object](https://aws.amazon.com/s3/) by uploading a file to an EventBridge-enabled bucket. When the event matches the rule, EventBridge will call the [Zendesk Create Ticket API](https://developer zendesk.com). The new ticket will appear in the Zendesk dashboard.

### Step 5: Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

**To delete the EventBridge Connections(s)**

1. Open the [API destination page](https://aws.amazon.com/eb/) of the EventBridge console.
2. Choose the **Connections** tab.
3. Select the Connection(s) you created.
4. Choose **Delete**.
5. Enter the name of the connection and choose **Delete**.

**To delete the EventBridge API destination(s)**

1. Open the [API destination page](https://aws.amazon.com/eb/) of the EventBridge console.
2. Select the API destinations(s) you created.
3. Choose **Delete**.
4. Enter the name of the API destination and choose **Delete**.

**To delete the EventBridge rule(s)**

2. Select the rule(s) that you created.
3. Choose **Delete**.
4. Choose **Delete**.
Using EventBridge 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.

<table>
<thead>
<tr>
<th>SDK documentation</th>
<th>Code examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS SDK for C++</td>
<td>AWS SDK for C++ code examples</td>
</tr>
<tr>
<td>AWS SDK for Go</td>
<td>AWS SDK for Go code examples</td>
</tr>
<tr>
<td>AWS SDK for Java</td>
<td>AWS SDK for Java code examples</td>
</tr>
<tr>
<td>AWS SDK for JavaScript</td>
<td>AWS SDK for JavaScript code examples</td>
</tr>
<tr>
<td>AWS SDK for Kotlin</td>
<td>AWS SDK for Kotlin code examples</td>
</tr>
<tr>
<td>AWS SDK for .NET</td>
<td>AWS SDK for .NET code examples</td>
</tr>
<tr>
<td>AWS SDK for PHP</td>
<td>AWS SDK for PHP code examples</td>
</tr>
<tr>
<td>AWS SDK for Python (Boto3)</td>
<td>AWS SDK for Python (Boto3) code examples</td>
</tr>
<tr>
<td>AWS SDK for Ruby</td>
<td>AWS SDK for Ruby code examples</td>
</tr>
<tr>
<td>AWS SDK for Rust</td>
<td>AWS SDK for Rust code examples</td>
</tr>
<tr>
<td>AWS SDK for SAP ABAP</td>
<td>AWS SDK for SAP ABAP code examples</td>
</tr>
<tr>
<td>AWS SDK for Swift</td>
<td>AWS SDK for Swift code examples</td>
</tr>
</tbody>
</table>

For examples specific to EventBridge, see [Code examples for EventBridge using AWS SDKs (p. 294)](#).

**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 EventBridge using AWS SDKs

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

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples are sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello EventBridge

The following code examples show how to get started using EventBridge.

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
using Amazon.EventBridge;
using Amazon.EventBridge.Model;
namespace EventBridgeActions;
public static class HelloEventBridge
{
    static async Task Main(string[] args)
    {
        var eventBridgeClient = new AmazonEventBridgeClient();

        Console.WriteLine("Hello Amazon EventBridge! Following are some of your EventBuses:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five event buses.
        var response = await eventBridgeClient.ListEventBusesAsync(
            new ListEventBusesRequest()
            {
                Limit = 5
            });
    }
}
```
foreach (var eventBus in response.EventBuses)
{
    Console.WriteLine($"EventBus: {eventBus.Name}");
    Console.WriteLine($"Arn: {eventBus.Arn}");
    Console.WriteLine($"Policy: {eventBus.Policy}");
    Console.WriteLine();
}

• For API details, see [ListEventBuses](https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/java-sdk-list-eventbuses.html) in AWS SDK for .NET API Reference.

Java

**SDK for Java 2.x**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-sdk-java-get-started).

```java
/**
 * Before running this Java V2 code example, set up your development environment, including your credentials.
 * For more information, see the following documentation topic:
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class HelloEventBridge {

    public static void main(String[] args) {
        Region region = Region.US_WEST_2;
        EventBridgeClient eventBrClient = EventBridgeClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        listBuses(eventBrClient);
        eventBrClient.close();
    }

    public static void listBuses( EventBridgeClient eventBrClient) {
        try {
            ListEventBusesRequest busesRequest = ListEventBusesRequest.builder()
                .limit(10)
                .build();

            ListEventBusesResponse response = eventBrClient.listEventBuses(busesRequest);
            List<EventBus> buses = response.eventBuses();
            for (EventBus bus : buses) {
                System.out.println("The name of the event bus is: "+bus.name());
                System.out.println("The ARN of the event bus is: "+bus.arn());
            }
        } catch (EventBridgeException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}```
import aws.sdk.kotlin.services.eventbridge.EventBridgeClient
import aws.sdk.kotlin.services.eventbridge.model.ListEventBusesRequest
import aws.sdk.kotlin.services.eventbridge.model.ListEventBusesResponse

suspend fun main() {
    listBusesHello()
}

suspend fun listBusesHello() {
    val request = ListEventBusesRequest {
        limit = 10
    }

    EventBridgeClient { region = "us-west-2" }.use { eventBrClient ->
        val response: ListEventBusesResponse =
            eventBrClient.listEventBuses(request)
        response.eventBuses?.forEach { bus ->
            println("The name of the event bus is ${bus.name}"")
            println("The ARN of the event bus is ${bus.arn}"")
        }
    }
}

• For API details, see ListEventBuses in AWS SDK for Kotlin API reference.

Code examples

• Actions for EventBridge using AWS SDKs (p. 297)
  • Add a target using an AWS SDK (p. 297)
  • Create an EventBridge rule using an AWS SDK (p. 303)
  • Delete an EventBridge rule using an AWS SDK (p. 309)
  • Describe an EventBridge rule using an AWS SDK (p. 311)
  • Disable an EventBridge rule using an AWS SDK (p. 312)
  • Enable an EventBridge rule using an AWS SDK (p. 314)
  • List EventBridge rule names for a target using an AWS SDK (p. 316)
  • List EventBridge rules using an AWS SDK (p. 318)
  • List EventBridge targets for a rule using an AWS SDK (p. 320)
  • Remove EventBridge targets from a rule using an AWS SDK (p. 322)
  • Send EventBridge events using an AWS SDK (p. 324)
Actions for EventBridge using AWS SDKs

The following code examples demonstrate how to perform individual EventBridge actions with AWS SDKs. These excerpts call the EventBridge API and are code excerpts from larger programs that must be run in context. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the Amazon EventBridge API Reference.

Examples

- Add a target using an AWS SDK (p. 297)
- Create an EventBridge rule using an AWS SDK (p. 303)
- Delete an EventBridge rule using an AWS SDK (p. 309)
- Describe an EventBridge rule using an AWS SDK (p. 311)
- Disable an EventBridge rule using an AWS SDK (p. 312)
- Enable an EventBridge rule using an AWS SDK (p. 314)
- List EventBridge rule names for a target using an AWS SDK (p. 316)
- List EventBridge rules using an AWS SDK (p. 318)
- List EventBridge targets for a rule using an AWS SDK (p. 320)
- Remove EventBridge targets from a rule using an AWS SDK (p. 322)
- Send EventBridge events using an AWS SDK (p. 324)

Add a target using an AWS SDK

The following code examples show how to add a target to an Amazon EventBridge event.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with rules and targets (p. 342)

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Add an Amazon SNS topic as a target for a rule.

```csharp
/// <summary>
/// Add an Amazon SNS target topic to a rule.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
```
/// <param name="targetArn">The ARN of the Amazon SNS target.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    // Create the list of targets and add a new target.
    var targets = new List<Target>
    {
        new Target()
        {
            Arn = targetArn,
            Id = targetID
        }
    };

    // Add the targets to the rule.
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });

    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError($"Failed to add target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}");
        });
    }

    return targetID;
}

Add an input transformer to a target for a rule.

/// <summary>
/// Update an Amazon S3 object created rule with a transform on the target.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="targetArn">The ARN of the target.</param>
/// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default event bus.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName, string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    var targets = new List<Target>
    {
        new Target()
        {
            Id = targetID,
            Arn = targetArn,
            InputTransformer = new InputTransformer()
            {
                // InputTransformer properties...
            }
        }
    };

    // Add the targets to the rule.
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });

    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError($"Failed to add target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}");
        });
    }

    return targetID;
}
Add a target

```csharp
InputPathsMap = new Dictionary<string, string>()
{
    {"bucket", "$\cdot detail\cdot bucket\cdot name"},
    {"time", "$\cdot time"}
},
InputTemplate = "\"Notification: an object was uploaded to bucket <bucket> at <time>\"\"
};

var response = await _amazonEventBridge.PutTargetsAsync(
    new PutTargetsRequest()
    {
        EventBusName = eventBusArn,
        Rule = ruleName,
        Targets = targets,
    });
if (response.FailedEntryCount > 0)
{
    response.FailedEntries.ForEach(e =>
    {
        _logger.LogError(
            "$Failed to add target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}";
    }));
}
return targetID;
```

- For API details, see [PutTargets](aws-sdk-net-api-reference#puttargets) in AWS SDK for .NET API Reference.

C++

**SDK for C++**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](http://aws.amazon.com/code-examples/).

Include the required files.

```c++
#include <aws/core/Aws.h>
#include <aws/events/EventBridgeClient.h>
#include <aws/events/model/PutTargetsRequest.h>
#include <aws/events/model/PutTargetsResult.h>
#include <aws/core/utils/Outcome.h>
#include <iostream>
```

Add the target.

```c++
Aws::CloudWatchEvents::EventBridgeClient cwe;

Aws::CloudWatchEvents::Model::Target target;
target.SetArn(lambda_arn);
target.SetId(target_id);

Aws::CloudWatchEvents::Model::PutTargetsRequest request;
request.SetRule(rule_name);
request.AddTargets(target);
```
auto putTargetsOutcome = cwe.PutTargets(request);
if (!putTargetsOutcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch events target for rule " 
    << rule_name << ": " <<  
    putTargetsOutcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully created CloudWatch events target for rule " 
    << rule_name << std::endl;
}

• For API details, see PutTargets in AWS SDK for C++ API Reference.

Java

SDK for Java 2.x

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Add an Amazon SNS topic as a target for a rule.

```java
public static void addSnsEventRule(EventBridgeClient eventBrClient, String ruleName, String topicArn, String topicName, String eventRuleName, String bucketName) {
    String targetID = java.util.UUID.randomUUID().toString();
    Target myTarget = Target.builder()
        .id(targetID)
        .arn(topicArn)
        .build();

    List<Target> targets = new ArrayList<>();
    targets.add(myTarget);
    PutTargetsRequest request = PutTargetsRequest.builder()
        .eventBusName(null)
        .targets(targets)
        .rule(ruleName)
        .build();

    eventBrClient.putTargets(request);
    System.out.println("Added event rule "+eventRuleName +" with Amazon SNS target "+topicName +" for bucket "+bucketName +");
}
```

Add an input transformer to a target for a rule.

```java
public static void updateCustomRuleTargetWithTransform(EventBridgeClient eventBrClient, String topicArn, String ruleName) {
    String targetId = java.util.UUID.randomUUID().toString();
    InputTransformer inputTransformer = InputTransformer.builder()
        .inputTemplate("\"Notification: sample event was received.\""")
        .build();

    Target target = Target.builder()
```
Add a target

```java
.id(targetId)
.arn(topicArn)
.inputTransformer(inputTransformer)
.build();

try {
    PutTargetsRequest targetsRequest = PutTargetsRequest.builder()
        .rule(ruleName)
        .targets(target)
        .eventBusName(null)
        .build();

    eventBrClient.putTargets(targetsRequest);
} catch (EventBridgeException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

- For API details, see [PutTargets](https://docs.aws.amazon.com/AWSJavaSDK/latest/api/aws/api/v2/com/aws/service/eventbridge/puttargetsRequest.html) in [AWS SDK for Java 2.x API Reference](https).

**JavaScript**

**SDK for JavaScript (v3)**

**Note**

There’s more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-sdk-js-v3).

Import the SDK and client modules and call the API.

```javascript
import {
    EventBridgeClient,
    PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
    existingRuleName = "some-rule",
    uniqueId = Date.now().toString(),
) => {
    const client = new EventBridgeClient({});
    const response = await client.send(
        new PutTargetsCommand({
            Rule: existingRuleName,
            Targets: [
                {
                    Arn: targetArn,
                    Id: uniqueId,
                },
            ],
        }
    );

    console.log("PutTargets response:");
    console.log(response);
    // PutTargets response:
    // { // $metadata: {
    // httpStatusCode: 200,
    // requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
    // extendedRequestId: undefined,
    // cfId: undefined,
```
Add a target

```javascript
//     attempts: 1,
//     totalRetryDelay: 0
// },
// FailedEntries: [],
// FailedEntryCount: 0
// }

return response;
```

- For API details, see [PutTargets](https://aws.amazon.com/sdk-for-javascript/api-reference/) in AWS SDK for JavaScript API Reference.

**SDK for JavaScript (v2)**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```javascript
// 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 ebevents = new AWS.EventBridge({apiVersion: '2015-10-07'});

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

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

- For API details, see [PutTargets](https://aws.amazon.com/sdk-for-javascript/api-reference/) in AWS SDK for JavaScript API Reference.

**Kotlin**

**SDK for Kotlin**

**Note**

This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
// Add a rule that triggers an SNS target when a file is uploaded to an S3 bucket.
```
suspend fun addSnsEventRule(ruleName: String?, topicArn: String?, topicName: String, eventRuleName: String, bucketName: String) {
    val targetID = UUID.randomUUID().toString()
    val myTarget = Target {
        id = targetID
        arn = topicArn
    }
    val targetsOb = mutableListOf<Target>()
    targetsOb.add(myTarget)
    val request = PutTargetsRequest {
        eventBusName = null
        targets = targetsOb
        rule = ruleName
    }
    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putTargets(request)
        println("Added event rule $eventRuleName with Amazon SNS target $topicName for bucket $bucketName.")
    }
}

Add an input transformer to a target for a rule.

suspend fun updateCustomRuleTargetWithTransform(topicArn: String?, ruleName: String?) {
    val targetId = UUID.randomUUID().toString()
    val inputTransformerOb = InputTransformer {
        inputTemplate = "Notification: sample event was received."
    }
    val target = Target {
        id = targetId
        arn = topicArn
        inputTransformer = inputTransformerOb
    }
    val targetsRequest = PutTargetsRequest {
        rule = ruleName
        targets = listOf(target)
        eventBusName = null
    }
    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putTargets(targetsRequest)
    }
}

• For API details, see PutTargets in AWS SDK for Kotlin API reference.

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.

Create an EventBridge rule using an AWS SDK

The following code examples show how to create an Amazon EventBridge rule.
Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Create and trigger a rule](p. 328)
- [Get started with rules and targets](p. 342)

### .NET

**AWS SDK for .NET**

**Note**

There’s more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create a rule that triggers when an object is added to an Amazon Simple Storage Service bucket.

```csharp
/// <summary>
/// Create a new event rule that triggers when an Amazon S3 object is created in a bucket.
/// </summary>
/// <param name="roleArn">The ARN of the role.</param>
/// <param name="ruleName">The name to give the rule.</param>
/// <param name="bucketName">The name of the bucket to trigger the event.</param>
/// <returns>The ARN of the new rule.</returns>
public async Task<string> PutS3UploadRule(string roleArn, string ruleName, string bucketName)
{
    string eventPattern = "{" + 
    "\"source\": ["aws.s3"]," + 
    "\"detail-type\": ["Object Created"]," + 
    "\"detail\": {" + 
    "\"bucket\": {" + 
    "\"name\": ["" + bucketName + "]" + 
    "]" + 
    "}"},

    var response = await _amazonEventBridge.PutRuleAsync( 
        new PutRuleRequest() 
        { 
            Name = ruleName, 
            Description = "Example S3 upload rule for EventBridge", 
            RoleArn = roleArn, 
            EventPattern = eventPattern 
        });

    return response.RuleArn;
}
```

Create a rule that uses a custom pattern.

```csharp
/// <summary>
/// Update a rule to use a custom defined event pattern.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <returns>The ARN of the updated rule.</returns>
public async Task<string> UpdateCustomEventPattern(string ruleName)
{
    string customEventsPattern = "{" + 
```
Create a rule

```javascript
var response = await _amazonEventBridge.PutRuleAsync(
    new PutRuleRequest()
    {
        Name = ruleName,
        Description = "Custom test rule",
        EventPattern = customEventsPattern
    });

return response.RuleArn;
```

• For API details, see [PutRule](https://docs.aws.amazon.com/sdk-for-net/v1/developerguide/services/events.html) in *AWS SDK for .NET API Reference*.

C++

**SDK for C++**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-sdk-cpp).

Include the required files.

```c++
#include <aws/core/Aws.h>
#include <aws/events/EventBridgeClient.h>
#include <aws/events/model/PutRuleRequest.h>
#include <aws/events/model/PutRuleResult.h>
#include <aws/core/utils/Outcome.h>
#include <iostream>
```

Create the rule.

```c++
Aws::CloudWatchEvents::EventBridgeClient cwe;
Aws::CloudWatchEvents::Model::PutRuleRequest request;
request.SetName(rule_name);
request.SetRoleArn(role_arn);
request.SetScheduleExpression("rate(5 minutes)");
request.SetState(Aws::CloudWatchEvents::Model::RuleState::ENABLED);

auto outcome = cwe.PutRule(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch events rule " << rule_name << ": " << outcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully created CloudWatch events rule " << rule_name << " with resulting Arn " << outcome.GetResult().GetRuleArn() << std::endl;
}
```

• For API details, see [PutRule](https://docs.aws.amazon.com/sdk-for-cpp/v1/developerguide/services/events.html) in *AWS SDK for C++ API Reference*.
Create a scheduled rule.

```java
public static void createEBRule(EventBridgeClient eventBrClient, String ruleName, String cronExpression) {
    try {
        PutRuleRequest ruleRequest = PutRuleRequest.builder()
            .name(ruleName)
            .eventBusName("default")
            .scheduleExpression(cronExpression)
            .state("ENABLED")
            .description("A test rule that runs on a schedule created by the Java API")
            .build();

        PutRuleResponse ruleResponse = eventBrClient.putRule(ruleRequest);
        System.out.println("The ARN of the new rule is "+ruleResponse.ruleArn());
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

Create a rule that triggers when an object is added to an Amazon Simple Storage Service bucket.

```java
// Create a new event rule that triggers when an Amazon S3 object is created in a bucket.
public static void addEventRule( EventBridgeClient eventBrClient, String roleArn, String bucketName, String eventRuleName) {
    String pattern = "\n" +
    " \"source\": [\"aws.s3\"],\n" +
    " \"detail-type\": [\"Object Created\"],\n" +
    " \"detail\": {\n" +
    " \"bucket\": {\n" +
    " \"name\": "[\""+bucketName+"\"]\n" +
    " }\n" +
    " }\n" +
    "} ;

    try {
        PutRuleRequest ruleRequest = PutRuleRequest.builder()
            .description("Created by using the AWS SDK for Java v2")
            .name(eventRuleName)
            .eventPattern(pattern)
            .roleArn(roleArn)
            .build();

        PutRuleResponse ruleResponse = eventBrClient.putRule(ruleRequest);
        System.out.println("The ARN of the new rule is "+ruleResponse.ruleArn());
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```
System.exit(1);

• For API details, see [PutRule](https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/eventbridge-putrule-apis.html) in *AWS SDK for Java 2.x API Reference*.  

**JavaScript**

**SDK for JavaScript (v3)**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

Import the SDK and client modules and call the API.

```javascript
import { EventBridgeClient, PutRuleCommand } from '@aws-sdk/client-eventbridge';

export const putRule = async (ruleName = 'some-rule', source = 'some-source',) => {
    const client = new EventBridgeClient({});
    const response = await client.send(new PutRuleCommand({
        Name: ruleName,
        EventPattern: JSON.stringify({ source: [source] }),
        State: "ENABLED",
        EventBusName: "default",
    }));
    console.log("PutRule response:");
    console.log(response);
    // PutRule response:
    // { "$metadata": {
    //     httpStatusCode: 200,
    //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    // },
    // };
    return response;
};
```

• For API details, see [PutRule](https://docs.aws.amazon.com/sdk-for-javascript/latest/developer-guide/eventbridge-putrule-apis.html) in *AWS SDK for JavaScript API Reference*.  

**SDK for JavaScript (v2)**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

```javascript
// 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 ebevents = new AWS.EventBridge({apiVersion: '2015-10-07'});

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

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

- For API details, see PutRule in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a scheduled rule.

suspend fun createScRule(ruleName: String?, cronExpression: String?) {
  val ruleRequest = PutRuleRequest {
    name = ruleName
    eventBusName = "default"
    scheduleExpression = cronExpression
    state = RuleState.Enabled
    description = "A test rule that runs on a schedule created by the Kotlin API"
  }

  EventBridgeClient { region = "us-west-2" }.use { eventBrClient ->
    val ruleResponse = eventBrClient.putRule(ruleRequest)
    println("The ARN of the new rule is \${ruleResponse.ruleArn}\")
  }
}

Create a rule that triggers when an object is added to an Amazon Simple Storage Service bucket.

// Create a new event rule that triggers when an Amazon S3 object is created in a bucket.
suspend fun addEventRule(roleArnVal: String?, bucketName: String, eventRuleName: String?) {
Delete an EventBridge rule using an AWS SDK

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.

Delete an EventBridge rule using an AWS SDK

The following code examples show how to delete an Amazon EventBridge rule.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

• Get started with rules and targets (p. 342)

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete a rule by its name.

```csharp
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });
```
Delete a rule

```java
public static void deleteRuleByName(EventBridgeClient eventBrClient, String ruleName) {
    DeleteRuleRequest ruleRequest = DeleteRuleRequest.builder()
            .name(ruleName)
            .build();
    eventBrClient.deleteRule(ruleRequest);
    System.out.println("Successfully deleted the rule");
}
```

For API details, see [DeleteRule](https://docs.aws.amazon.com/aws-sdk-for-java/2.13.92/APIReference/API_DeleteRule.html) in AWS SDK for Java 2.x API Reference.

**SDK for Kotlin**

```kotlin
suspend fun deleteRuleByName(ruleName: String?) {
    val ruleRequest = DeleteRuleRequest {
        name = ruleName
    }
    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.deleteRule(ruleRequest)
        println("Successfully deleted the rule")
    }
}
```

For API details, see [DeleteRule](https://docs.aws.amazon.com/aws-sdk-for-kotlin/latest/api/index.html#com.amazonaws.services.events.model.DeleteRuleRequest) in AWS SDK for Kotlin API reference.

For a complete list of AWS SDK developer guides and code examples, see [Using EventBridge with an AWS SDK](https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/java-sdk-index.html). This topic also includes information about getting started and details about previous SDK versions.
Describe an EventBridge rule using an AWS SDK

The following code examples show how to describe an Amazon EventBridge rule.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with rules and targets (p. 342)

.NET

AWS SDK for .NET

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the state of a rule using the rule description.

```csharp
/// <summary>
/// Get the state for a rule by the rule name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="eventBusName">The optional name of the event bus. If empty, uses the default event bus.</param>
/// <returns>The state of the rule.</returns>
public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string? eventBusName = null)
{
    var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(new DescribeRuleRequest()
    {
        Name = ruleName,
        EventBusName = eventBusName
    });
    return ruleResponse.State;
}
```

For API details, see DescribeRule in AWS SDK for .NET API Reference.

Java

SDK for Java 2.x

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public static void checkRule(EventBridgeClient eventBrClient, String eventRuleName) {
    try {
        DescribeRuleRequest ruleRequest = DescribeRuleRequest.builder()
            .name(eventRuleName)
            .build();
        DescribeRuleResponse response = eventBrClient.describeRule(ruleRequest);
        System.out.println("The state of the rule is " + response.stateAsString());
    }
```
disable a rule

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

- For API details, see [DescribeRule](#) in *AWS SDK for Java 2.x API Reference*.

**Kotlin**

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```kotlin
suspend fun checkRule(eventRuleName: String?) {
    val ruleRequest = DescribeRuleRequest {
        name = eventRuleName
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.describeRule(ruleRequest)
        println("The state of the rule is $response")
    }
}
```

- For API details, see [DescribeRule](#) in *AWS SDK for Kotlin API reference*.

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

**Disable an EventBridge rule using an AWS SDK**

The following code examples show how to disable an Amazon EventBridge rule.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with rules and targets](#) (p. 342)

**.NET**

**AWS SDK for .NET**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Disable a rule by its rule name.
/// <summary>
/// Disable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableRuleByName(string ruleName) {
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
        new DisableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

- For API details, see [DisableRule](https://aws-sdk.amazonaws.com/documentation/dotnetapi/#AmazonEventBridge_DisableRule) in [AWS SDK for .NET API Reference](https).

### Java

**SDK for Java 2.x**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-sdk-java).

Disable a rule by using its rule name.

```java
public static void changeRuleState(EventBridgeClient eventBrClient, String eventRuleName, Boolean isEnabled) {
    try {
        if (!isEnabled) {
            System.out.println("Disabling the rule: "+eventRuleName);
            DisableRuleRequest ruleRequest = DisableRuleRequest.builder()
                .name(eventRuleName)
                .build();
            eventBrClient.disableRule(ruleRequest);
        } else {
            System.out.println("Enabling the rule: "+eventRuleName);
            EnableRuleRequest ruleRequest = EnableRuleRequest.builder()
                .name(eventRuleName)
                .build();
            eventBrClient.enableRule(ruleRequest);
        }
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [DisableRule](https://aws-sdk.amazonaws.com/documentation/javaapi/#AmazonEventBridge_DisableRule) in [AWS SDK for Java 2.x API Reference](https).

### Kotlin

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

```kotlin
public static fun changeRuleState(eventBrClient: EventBridgeClient, eventRuleName: String, isEnabled: Boolean) {
    try {
        if (!isEnabled) {
            println("Disabling the rule: "+eventRuleName);
            val ruleRequest = DisableRuleRequest.builder().name(eventRuleName).build()
            eventBrClient.disableRule(ruleRequest)
        } else {
            println("Enabling the rule: "+eventRuleName)
            val ruleRequest = EnableRuleRequest.builder().name(eventRuleName).build()
            eventBrClient.enableRule(ruleRequest)
        }
    } catch (e: EventBridgeException) {
        println(e.awsErrorDetails().errorMessage())
        System.exit(1)
    }
}
```
Enable an EventBridge rule using an AWS SDK

The following code examples show how to enable an Amazon EventBridge rule.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with rules and targets (p. 342)

.NET

AWS SDK for .NET

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Enable a rule by its rule name.

```csharp
/// <summary>
/// Enable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableRuleByName(string ruleName) {
    var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
```
Enable a rule by using its rule name.

```java
public static void changeRuleState(EventBridgeClient eventBrClient, String eventRuleName, Boolean isEnabled) {
    try {
        if (!isEnabled) {
            System.out.println("Disabling the rule: "+eventRuleName);
            DisableRuleRequest ruleRequest = DisableRuleRequest.builder()
                .name(eventRuleName)
                .build();
            eventBrClient.disableRule(ruleRequest);
        } else {
            System.out.println("Enabling the rule: "+eventRuleName);
            EnableRuleRequest ruleRequest = EnableRuleRequest.builder()
                .name(eventRuleName)
                .build();
            eventBrClient.enableRule(ruleRequest);
        }
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

• For API details, see EnableRule in AWS SDK for .NET API Reference.

SDK for Java 2.x

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Kotlin

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun changeRuleState(eventRuleName: String, isEnabled: Boolean?) {
    // Kotlin code
}
```
List all of the rule names using the target.

```csharp
/// <summary>
/// List names of all rules matching a target.
/// </summary>
/// <param name="targetArn">The ARN of the target.</param>
/// <returns>The list of rule names.</returns>
public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
{
    var results = new List<string>();
    var request = new ListRuleNamesByTargetRequest()
    {
        TargetArn = targetArn
    };
    ListRuleNamesByTargetResponse response;
```
List rule names for a target

```java
do {
    response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
    results.AddRange(response.RuleNames);
    request.NextToken = response.NextToken;
} while (response.NextToken is not null);
return results;
```

• For API details, see [ListRuleNamesByTarget](https://docs.aws.amazon.com/sdk-for-net/vload平衡.0/api/aws.amazon.eventbridge/AmazonEventBridge.ListRuleNamesByTargetAsync) in [AWS SDK for .NET API Reference](https).

**Java**

**SDK for Java 2.x**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https).

List all of the rule names by using the target.

```java
public static void listTargetRules(EventBridgeClient eventBrClient, String topicArn) {
    ListRuleNamesByTargetRequest ruleNamesByTargetRequest =
        ListRuleNamesByTargetRequest.builder()
            .targetArn(topicArn)
            .build();

    ListRuleNamesByTargetResponse response =
        eventBrClient.listRuleNamesByTarget(ruleNamesByTargetRequest);
    List<String> rules = response.ruleNames();
    for (String rule:rules) {
        System.out.println("The rule name is "+rule);
    }
}
```

• For API details, see [ListRuleNamesByTarget](https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/aws-java-sdk-core-list-rule-names-by-target.html) in [AWS SDK for Java 2.x API Reference](https).

**Kotlin**

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https).

```kotlin
suspend fun listTargetRules(topicArnVal: String?) {
    val ruleNamesByTargetRequest = ListRuleNamesByTargetRequest {
        targetArn = topicArnVal
    }
```


For API details, see [ListRuleNamesByTarget](https://docs.aws.amazon.com/sdk-for-kotlin/api/latest/api/aws-eventbridge/awseventbridge/GetRuleNamesByTarget.html) in **AWS SDK for Kotlin API reference**.

For a complete list of AWS SDK developer guides and code examples, see [Using EventBridge with an AWS SDK](p. 293). This topic also includes information about getting started and details about previous SDK versions.

## List EventBridge rules using an AWS SDK

The following code examples show how to list Amazon EventBridge rules.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with rules and targets (p. 342)](p. 342)

### .NET

**AWS SDK for .NET**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/sdks). 

List all of the rules for an event bus.

```csharp
/// <summary>
/// List the rules on an event bus.
/// </summary>
/// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses the default event bus.</param>
/// <returns>The list of rules.</returns>
public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn = null)
{
    var results = new List<Rule>();
    var request = new ListRulesRequest()
    {
        EventName = eventBusArn
    };
    // Get all of the pages of rules.
    ListRulesResponse response;
    do
    {
        response = await _amazonEventBridge.ListRulesAsync(request);
        results.AddRange(response.Rules);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);
    return results;
```
Enable a rule by using its rule name.

```java
public static void listRules(EventBridgeClient eventBrClient) {
  try {
    ListRulesRequest rulesRequest = ListRulesRequest.builder()
      .eventBusName("default")
      .limit(10)
      .build();

    ListRulesResponse response = eventBrClient.listRules(rulesRequest);
    List<Rule> rules = response.rules();
    for (Rule rule : rules) {
      System.out.println("The rule name is : " + rule.name());
      System.out.println("The rule description is : " + rule.description());
      System.out.println("The rule state is : " + rule.stateAsString());
    }
  } catch (EventBridgeException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
  }
}
```

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

For API details, see ListRules in AWS SDK for Java 2.x API Reference.

Kotlin

```kotlin
suspend fun listRules() {
  val rulesRequest = ListRulesRequest {
    eventBusName = "default"
    limit = 10
  }

  EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
    val response = eventBrClient.listRules(rulesRequest)
  }
}
```

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
List targets for a rule

```kotlin
response.rules?.forEach { rule ->
    println("The rule name is "+rule.name)
    println("The rule ARN is "+rule.arn)
}
```

• For API details, see ListRules in AWS SDK for Kotlin API reference.

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.

## List EventBridge targets for a rule using an AWS SDK

The following code examples show how to list Amazon EventBridge targets for a rule.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

• Get started with rules and targets (p. 342)

### .NET

**AWS SDK for .NET**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List all of the targets for a rule using the rule name.

```csharp
/// <summary>
/// List all of the targets matching a rule by name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>The list of targets.</returns>
public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
{
    var results = new List<Target>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse response;
    do
    {
        response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
        results.AddRange(response.Targets);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);
    return results;
}
```

• For API details, see ListTargetsByRule in AWS SDK for .NET API Reference.
List all of the targets for a rule by using the rule name.

```java
public static void listTargets(EventBridgeClient eventBrClient, String ruleName) {
    ListTargetsByRuleRequest ruleRequest = ListTargetsByRuleRequest.builder()
        .rule(ruleName)
        .build();

    ListTargetsByRuleResponse res =
        eventBrClient.listTargetsByRule(ruleRequest);
    List<Target> targets = res.targets();
    for (Target target : targets) {
        System.out.println("Target ARN: "+target.arn());
    }
}
```

- For API details, see [ListTargetsByRule](https://docs.aws.amazon.com/java-sdk/latest/api/index.html) in AWS SDK for Java 2.x API Reference.

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.
Remove EventBridge targets from a rule using an AWS SDK

The following code examples show how to remove Amazon EventBridge targets from a rule.

**AWS SDK for .NET**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Remove all of the targets for a rule using the rule name.

```csharp
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
{
    var targetIds = new List<string>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse targetsResponse;
    do
    {
        targetsResponse = await _amazonEventBridge.ListTargetsByRuleAsync(request);
        targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
        request.NextToken = targetsResponse.NextToken;
    } while (targetsResponse.NextToken is not null);
    var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(new RemoveTargetsRequest()
    {
        Rule = ruleName,
        Ids = targetIds
    });

    if (removeResponse.FailedEntryCount > 0)
    {
        removeResponse.FailedEntries.ForEach(e =>
        {
            _logger.LogError("Failed to remove target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}");
        });
    }

    return removeResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [RemoveTargets](https://docs.aws.amazon.com/sdk-for-net/v3/api/aws.eventbridge.html) in [AWS SDK for .NET API Reference](https://docs.aws.amazon.com/sdk-for-net/v3/api/aws.eventbridge.html).
Remove targets from a rule

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Remove all of the targets for a rule by using the rule name.

```java
public static void deleteTargetsFromRule(EventBridgeClient eventBrClient,
String eventRuleName) {
    // First, get all targets that will be deleted.
    ListTargetsByRuleRequest request = ListTargetsByRuleRequest.builder()
        .rule(eventRuleName)
        .build();

    ListTargetsByRuleResponse response =
    eventBrClient.listTargetsByRule(request);
    List<Target> allTargets = response.targets();

    // Get all targets and delete them.
    for (Target myTarget: allTargets) {
        RemoveTargetsRequest removeTargetsRequest =
        RemoveTargetsRequest.builder()
            .rule(eventRuleName)
            .ids(myTarget.id())
            .build();

        eventBrClient.removeTargets(removeTargetsRequest);
        System.out.println("Successfully removed the target");
    }
}
```

- For API details, see RemoveTargets in AWS SDK for Java 2.x API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun deleteTargetsFromRule(eventRuleName: String?) {
    // First, get all targets that will be deleted.
    val request = ListTargetsByRuleRequest {
        rule = eventRuleName
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.listTargetsByRule(request)
        val allTargets = response.targets

        // Get all targets and delete them.
        if (allTargets != null) {
```
Send EventBridge events using an AWS SDK

The following code examples show how to send Amazon EventBridge events.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create and trigger a rule (p. 328)
- Get started with rules and targets (p. 342)

.NET

AWS SDK for .NET

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send an event that matches a custom pattern for a rule.

```csharp
public async Task<bool> PutCustomEmailEvent(string email)
{
    var eventDetail = new
    {
        UserEmail = email,
        Message = "This event was generated by example code.",
        UtcTime = DateTime.UtcNow.ToString("g")
    };
    var response = await _amazonEventBridge.PutEventsAsync(
        new PutEventsRequest()
        {
            Entries = new List<PutEventsRequestEntry>()
            {
                new PutEventsRequestEntry()
            }
        });
    return response;
}
```
Send events

```cpp
{
    Source = "ExampleSource",
    Detail = JsonSerializer.Serialize(eventDetail),
    DetailType = "ExampleType"
}
});

return response.FailedEntryCount == 0;
```

- For API details, see **PutEvents** in *AWS SDK for .NET API Reference*.

### C++

**SDK for C++**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://aws.amazon.com/code-examples/).

Include the required files.

```cpp
#include <aws/core/Aws.h>
#include <aws/events/EventBridgeClient.h>
#include <aws/events/model/PutEventsRequest.h>
#include <aws/events/model/PutEventsResult.h>
#include <aws/core/utils/Outcome.h>
#include <iostream>
```

Send the event.

```cpp
Aws::CloudWatchEvents::EventBridgeClient cwe;

Aws::CloudWatchEvents::Model::PutEventsRequestEntry event_entry;
event_entry.SetDetail(MakeDetails(event_key, event_value));
event_entry.SetDetailType("sampleSubmitted");
event_entry.AddResources(resource_arn);
event_entry.SetSource("aws-sdk-cpp-cloudwatch-example");

Aws::CloudWatchEvents::Model::PutEventsRequest request;
request.AddEntries(event_entry);

auto outcome = cwe.PutEvents(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to post CloudWatch event: " << outcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully posted CloudWatch event" << std::endl;
}
```

- For API details, see **PutEvents** in *AWS SDK for C++ API Reference*. 

---

325
Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public static void triggerCustomRule(EventBridgeClient eventBrClient, String email) {
    String json = "{" +
        "\"UserEmail\": "+email+"\"," +
        "\"Message\": \"This event was generated by example code.\"," +
        "\"UtcTime\": \"Now.\" +
    "}";

    PutEventsRequestEntry entry = PutEventsRequestEntry.builder()  
        .source("ExampleSource")  
        .detail(json)  
        .detailType("ExampleType")  
        .build();

    PutEventsRequest eventsRequest = PutEventsRequest.builder()  
        .entries(entry)  
        .build();

    eventBrClient.putEvents(eventsRequest);
}
```

• For API details, see PutEvents in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript (v3)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```javascript
import { 
    EventBridgeClient, 
    PutEventsCommand, 
} from "@aws-sdk/client-eventbridge";

export const putEvents = async ( 
    source = "eventbridge.integration.test", 
    detailType = "greeting", 
    resources = [], 
) => { 
    const client = new EventBridgeClient({});

    const response = await client.send( 
        new PutEventsCommand({ 
            Entries: [ 
                { 
                    Detail: JSON.stringify({ greeting: "Hello there." }), 
                    DetailType: detailType, 
                    Resources: resources, 
                    Source: source, 
```
console.log("PutEvents response:");
console.log(response);
// PutEvents response:
// { $metadata: {
//   httpStatusCode: 200,
//   requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
//   extendedRequestId: undefined,
//   cfId: undefined,
//   attempts: 1,
//   totalRetryDelay: 0
// },
//   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
//   FailedEntryCount: 0
// }

return response;

• For API details, see PutEvents in AWS SDK for JavaScript API Reference.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

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

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

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

• For API details, see PutEvents in AWS SDK for JavaScript API Reference.
Kotlin

**SDK for Kotlin**

*Note*
This is prerelease documentation for a feature in preview release. It is subject to change.

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-code-examples).

```kotlin
suspend fun triggerCustomRule(email: String) {
    val json = "{" +
        "\"UserEmail\": \"" + email + "\"," +
        "\"Message\": \"This event was generated by example code.\"" +
        "\"UtcTime\": \"Now.\"" +
    "}" +

    val entry = PutEventsRequestEntry {
        source = "ExampleSource"
        detail = json
        detailType = "ExampleType"
    }

    val eventsRequest = PutEventsRequest {
        this.entries = listOf(entry)
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putEvents(eventsRequest)
    }
}
```


For a complete list of AWS SDK developer guides and code examples, see [Using EventBridge with an AWS SDK](p. 293). This topic also includes information about getting started and details about previous SDK versions.

### Scenarios for EventBridge using AWS SDKs

The following code examples show you how to implement common scenarios in EventBridge with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within EventBridge. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

#### Examples

- Create and trigger a rule in Amazon EventBridge using an AWS SDK (p. 328)
- Get started with EventBridge rules and targets using an AWS SDK (p. 342)

#### Create and trigger a rule in Amazon EventBridge using an AWS SDK

The following code example shows how to create and trigger a rule in Amazon EventBridge.
Ruby

SDK for Ruby

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Call the functions in the correct order.

```ruby
require "aws-sdk-sns"
require "aws-sdk-iam"
require "aws-sdk-cloudwatchevents"
require "aws-sdk-ec2"
require "aws-sdk-cloudwatch"
require "aws-sdk-cloudwatchlogs"
require "securerandom"
```

Checks whether the specified Amazon Simple Notification Service (Amazon SNS) topic exists among those provided to this function.

```ruby
# Checks whether the specified Amazon SNS topic exists among those provided to this function.
# This is a helper function that is called by the topic_exists? function.
#
# @param topics [Array] An array of Aws::SNS::Types::Topic objects.
# @param topic_arn [String] The ARN of the topic to find.
# @return [Boolean] true if the topic ARN was found; otherwise, false.
# @example
#   sns_client = Aws::SNS::Client.new(region: 'us-east-1')
#   response = sns_client.list_topics
#   if topic_found?(response.topics, 'arn:aws:sns:us-east-1:111111111111:aws-doc-sdk-examples-topic')
#     puts 'Topic found.'
#   end
# def topic_found?(topics, topic_arn)
#   topics.each do |topic|
#     return true if topic.topic_arn == topic_arn
#   end
#   return false
# end
```

Checks whether the specified topic exists among those available to the caller in Amazon SNS.

```ruby
# Checks whether the specified topic exists among those available to the caller in Amazon SNS.
#
# @param sns_client [Aws::SNS::Client] An initialized Amazon SNS client.
# @param topic_arn [String] The ARN of the topic to find.
# @return [Boolean] true if the topic ARN was found; otherwise, false.
# @example
#   exit 1 unless topic_exists?(Aws::SNS::Client.new(region: 'us-east-1'),
#     'arn:aws:sns:us-east-1:111111111111:aws-doc-sdk-examples-topic')
# def topic_exists?(sns_client, topic_arn)
#   puts "Searching for topic with ARN '#{topic_arn}'..."
response = sns_client.list_topics
if response.topics.count.positive?
    if topic_found?(response.topics, topic_arn)
        puts "Topic found."
        return true
    end
    while response.next_page? do
        response = response.next_page
        if response.topics.count.positive?
            if topic_found?(response.topics, topic_arn)
                puts "Topic found."
                return true
            end
        end
    end
    puts "Topic not found."
    return false
rescue StandardError => e
    puts "Topic not found: #{e.message}"
    return false
end

Create a topic in Amazon SNS and then subscribe an email address to receive notifications to that topic.

# Creates a topic in Amazon SNS
# and then subscribes an email address to receive notifications to that topic.
#
# @param sns_client [Aws::SNS::Client] An initialized Amazon SNS client.
# @param topic_name [String] The name of the topic to create.
# @param email_address [String] The email address of the recipient to notify.
# @return [String] The ARN of the topic that was created.
# @example
#   puts create_topic(
#     Aws::SNS::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-topic',
#     'mary@example.com'
#   )
# def create_topic(sns_client, topic_name, email_address)
#    puts "Creating the topic named '#{topic_name}'..."
#    topic_response = sns_client.create_topic(name: topic_name)
#    puts "Topic created with ARN '#{topic_response.topic_arn}'."
#    subscription_response = sns_client.subscribe(
#        topic_arn: topic_response.topic_arn,
#        protocol: "email",
#        endpoint: email_address,
#        return_subscription_arn: true
#    )
#    puts "Subscription created with ARN '#{subscription_response.subscription_arn}'. Have the owner of the "
#    "email address '#{email_address}' check their inbox in a few minutes "
#    "and confirm the subscription to start receiving notification emails."
#    return topic_response.topic_arn
# end

Check whether the specified AWS Identity and Access Management (IAM) role exists among those provided to this function.
# Checks whether the specified AWS Identity and Access Management (IAM) role exists among those provided to this function. # This is a helper function that is called by the role_exists? function.
#
# @param roles [Array] An array of Aws::IAM::Role objects.
# @param role_arn [String] The ARN of the role to find.
# @return [Boolean] true if the role ARN was found; otherwise, false.
# @example
#   iam_client = Aws::IAM::Client.new(region: 'us-east-1')
#   response = iam_client.list_roles
#   if role_found?( response.roles,
#     'arn:aws:iam::111111111111:role/aws-doc-sdk-examples-ec2-state-change'
#   )
#     puts 'Role found.'
#   end
def role_found?(roles, role_arn)
  roles.each do |role|
    return true if role.arn == role_arn
  end
  return false
end

Check whether the specified role exists among those available to the caller in IAM.

# Checks whether the specified role exists among those available to the caller in AWS Identity and Access Management (IAM).
#
# @param iam_client [Aws::IAM::Client] An initialized IAM client.
# @param role_arn [String] The ARN of the role to find.
# @return [Boolean] true if the role ARN was found; otherwise, false.
# @example
#   exit 1 unless role_exists?( 
#     Aws::IAM::Client.new(region: 'us-east-1'),
#     'arn:aws:iam::111111111111:role/aws-doc-sdk-examples-ec2-state-change'
#   )
def role_exists?(iam_client, role_arn)
  puts "Searching for role with ARN '#{role_arn}'..."
  response = iam_client.list_roles
  if response.roles.count.positive?
    if role_found?(response.roles, role_arn)
      puts "Role found."
      return true
    end
  end
  while response.next_page?
    response = response.next_page
    if response.roles.count.positive?
      if role_found?(response.roles, role_arn)
        puts "Role found."
        return true
      end
    end
  end
  puts "Role not found."
  return false
rescue StandardError => e
  puts "Role not found: #{e.message}"
  return false
end
Create a role in IAM.

```ruby
# Creates a role in AWS Identity and Access Management (IAM).
# This role is used by a rule in Amazon EventBridge to allow
# that rule to operate within the caller's account.
# This role is designed to be used specifically by this code example.
#
# @param iam_client [Aws::IAM::Client] An initialized IAM client.
# @param role_name [String] The name of the role to create.
# @return [String] The ARN of the role that was created.
# @example
#   puts create_role(
#     Aws::IAM::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-ec2-state-change'
#   )
# def create_role(iam_client, role_name)
#   puts "Creating the role named '#{role_name}'..."
#   response = iam_client.create_role(
#     assume_role_policy_document: {
#       'Version': "2012-10-17",
#       'Statement': [
#         {
#           'Sid': "",
#           'Effect': "Allow",
#           'Principal': {
#             'Service': "events.amazonaws.com"
#           },
#           'Action': "sts:AssumeRole"
#         }
#       ].to_json,
#     path: "/",
#     role_name: role_name
#   )
#   puts "Role created with ARN '#{response.role.arn}'."
#   puts "Adding access policy to role..."
#   iam_client.put_role_policy(
#     policy_document: {
#       'Version': "2012-10-17",
#       'Statement': [
#         {
#           'Sid': "CloudWatchEventsFullAccess",
#           'Effect': "Allow",
#           'Resource': "*",
#           'Action': "events:*"
#         },
#         {
#           'Sid': "IAMPassRoleForCloudWatchEvents",
#           'Effect': "Allow",
#           'Resource': "arn:aws:iam::*:role/AWS_Events_Invoke_Targets",
#           'Action': "iam:PassRole"
#         }
#       ].to_json,
#     policy_name: "CloudWatchEventsPolicy",
#     role_name: role_name
#   )
#   puts "Access policy added to role."
#   return response.role.arn
# rescue StandardError => e
#   puts "Error creating role or adding policy to it: #{e.message}"
#   puts "If the role was created, you must add the access policy "
#   puts "to the role yourself, or delete the role yourself and try again."
#   return "Error"
# end
```
Checks whether the specified EventBridge rule exists among those provided to this function.

```ruby
# Checks whether the specified Amazon EventBridge rule exists among
# those provided to this function.
# This is a helper function that is called by the rule_exists? function.
#
# @param rules [Array] An array of Aws::CloudWatchEvents::Types::Rule objects.
# @param rule_arn [String] The name of the rule to find.
# @return [Boolean] true if the name of the rule was found; otherwise, false.
# @example
#   cloudwatchevents_client = Aws::CloudWatch::Client.new(region: 'us-east-1')
#   response = cloudwatchevents_client.list_rules
#   if rule_found?(response.rules, 'aws-doc-sdk-examples-ec2-state-change')
#     puts 'Rule found.'
#   end
def rule_found?(rules, rule_name)
  rules.each do |rule|
    return true if rule.name == rule_name
  end
  return false
end
```

Checks whether the specified rule exists among those available to the caller in EventBridge.

```ruby
# Checks whether the specified rule exists among those available to the
caller in Amazon EventBridge.
#
# @param cloudwatchevents_client [Aws::CloudWatchEvents::Client] An initialized Amazon EventBridge client.
# @param rule_name [String] The name of the rule to find.
# @return [Boolean] true if the rule name was found; otherwise, false.
# @example
#   exit 1 unless rule_exists?(  
#     Aws::CloudWatch::Client.new(region: 'us-east-1')  
#     'aws-doc-sdk-examples-ec2-state-change'  
#   )
def rule_exists?(cloudwatchevents_client, rule_name)
  puts "Searching for rule with name '#{rule_name}'..."
  response = cloudwatchevents_client.list_rules
  if response.rules.count.positive?
    if rule_found?(response.rules, rule_name)
      puts "Rule found."
      return true
    end
    while response.next_page? do
      response = response.next_page
      if response.rules.count.positive?
        if rule_found?(response.rules, rule_name)
          puts "Rule found."
          return true
        end
      end
    end
  end
  puts "Rule not found."
  return false
rescue StandardError => e
  puts "Rule not found: #{e.message}"
  return false
end
```
Create a rule in EventBridge.

```ruby
# Creates a rule in Amazon EventBridge.
# This rule is triggered whenever an available instance in
# Amazon EC2 changes to the specified state.
# This rule is designed to be used specifically by this code example.
#
# Prerequisites:
# - A role in AWS Identity and Access Management (IAM) that is designed
#   to be used specifically by this code example.
# - A topic in Amazon SNS.
#
# @param cloudwatchevents_client [Aws::CloudWatchEvents::Client] An initialized Amazon EventBridge client.
# @param rule_name [String] The name of the rule to create.
# @param rule_description [String] Some description for this rule.
# @param instance_state [String] The state that available instances in
#   Amazon EC2 must change to, to
#   trigger this rule.
# @param role_arn [String] The Amazon Resource Name (ARN) of the IAM role.
# @param target_id [String] Some identifying string for the rule's target.
# @param topic_arn [String] The ARN of the Amazon SNS topic.
# @return [Boolean] true if the rule was created; otherwise, false.
# @example
#   exit 1 unless rule_created?(    
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-ec2-state-change',
#     'running',
#     'arn:aws:iam::111111111111:role/aws-doc-sdk-examples-ec2-state-change',
#     'sns-topic',
#     'arn:aws:sns:us-east-1:111111111111:aws-doc-sdk-examples-topic'
#   )
# def rule_created?(    
#   cloudwatchevents_client,
#   rule_name,
#   rule_description,
#   instance_state,
#   role_arn,
#   target_id,
#   topic_arn
# )
#   puts "Creating rule with name '#{rule_name}'..."
#   put_rule_response = cloudwatchevents_client.put_rule(    
#     name: rule_name,
#     description: rule_description,
#     event_pattern: {    
#       'source': [    
#         "aws.ec2"
#     ],
#     'detail-type': [    
#       "EC2 Instance State-change Notification"
#     ],
#     'detail': {    
#       'state': [    
#         instance_state
#     ]
#   }
# ).to_json,
#   state: "ENABLED",
#   role_arn: role_arn
# )
```

334
puts "Rule created with ARN '#{put_rule_response.rule_arn}'."

put_targets_response = cloudwatchevents_client.put_targets(
  rule: rule_name,
  targets: [
    { id: target_id,
      arn: topic_arn
    }
  ]
)
if put_targets_response.key?(:failed_entry_count) &&
  put_targets_response.failed_entry_count > 0
  puts "Error(s) adding target to rule:"
  put_targets_response.failed_entries.each do |failure|
    puts failure.error_message
  end
  return false
else
  return true
end
rescue StandardError => e
  puts "Error creating rule or adding target to rule: #{e.message}"
  puts "If the rule was created, you must add the target "
  "to the rule yourself, or delete the rule yourself and try again."
  return false
end

Check to see whether the specified log group exists among those available to the caller in Amazon CloudWatch Logs.

```ruby
# Checks to see whether the specified log group exists among those available
# to the caller in Amazon CloudWatch Logs.
#
# @param cloudwatchlogs_client [Aws::CloudWatchLogs::Client] An initialized
#   Amazon CloudWatch Logs client.
# @param log_group_name [String] The name of the log group to find.
# @return [Boolean] true if the log group name was found; otherwise, false.
# @example
#   exit 1 unless log_group_exists(
#     Aws::CloudWatchLogs::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-cloudwatch-log'
#   )
# def log_group_exists?(cloudwatchlogs_client, log_group_name)
  puts "Searching for log group with name '#{log_group_name}'..."
  response = cloudwatchlogs_client.describe_log_groups(
    log_group_name_prefix: log_group_name
  )
  if response.log_groups.count.positive?
    response.log_groups.each do |log_group|
      puts "Log group found."
      return true
    end
  end
  puts "Log group not found."
  return false
rescue StandardError => e
  puts "Log group not found: #{e.message}"
  return false
end
```
Create a log group in CloudWatch Logs.

```ruby
# Creates a log group in Amazon CloudWatch Logs.
#
# @param cloudwatchlogs_client [Aws::CloudWatchLogs::Client] An initialized
#   Amazon CloudWatch Logs client.
# @param log_group_name [String] The name of the log group to create.
# @return [Boolean] true if the log group name was created; otherwise, false.
# @example
#   exit 1 unless log_group_created?(n
#     Aws::CloudWatchLogs::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-cloudwatch-log'
#   )
def log_group_created?(cloudwatchlogs_client, log_group_name)
  puts "Attempting to create log group with the name '#{log_group_name}'..."
  cloudwatchlogs_client.create_log_group(log_group_name: log_group_name)
  puts 'Log group created.'
  return true
rescue StandardError => e
  puts 'Error creating log group: #{e.message}'
  return false
end
```

Write an event to a log stream in CloudWatch Logs.

```ruby
# Writes an event to a log stream in Amazon CloudWatch Logs.
#
# Prerequisites:
# - A log group in Amazon CloudWatch Logs.
# - A log stream within the log group.
#
# @param cloudwatchlogs_client [Aws::CloudWatchLogs::Client] An initialized
#   Amazon CloudWatch Logs client.
# @param log_group_name [String] The name of the log group.
# @param log_stream_name [String] The name of the log stream within
#   the log group.
# @param message [String] The message to write to the log stream.
# @param sequence_token [String] If available, the sequence token from the
#   message that was written immediately before this message. This sequence
#   token is returned by Amazon CloudWatch Logs whenever you programmatically
#   write a message to the log stream.
# @return [String] The sequence token that is returned by
#   Amazon CloudWatch Logs after successfully writing the message to the
#   log stream.
# @example
#   puts log_event(
#     Aws::EC2::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-cloudwatch-log'
#     '2020/11/19/53f985be-199f-408e-9a45-fc242df9EX',
#     "Instance 'i-033c48ef067af3dEX' restarted.';
#     '49542672468310740095796045676567882148068632824696073EX'
#   )
def log_event(
  cloudwatchlogs_client,
  log_group_name,
  log_stream_name,
  message,
  sequence_token
)
  puts "Attempting to log '#{message}' to log stream '#{log_stream_name}'..."
  event = {
    log_group_name: log_group_name,
```
Restart an Amazon Elastic Compute Cloud (Amazon EC2) instance and adds information about the related activity to a log stream in CloudWatch Logs.

```ruby
# Restarts an Amazon EC2 instance
# and adds information about the related activity to a log stream
# in Amazon CloudWatch Logs.
#
# Prerequisites:
#
# - The Amazon EC2 instance to restart.
# - The log group in Amazon CloudWatch Logs to add related activity
#   information to.
#
# @param ec2_client [Aws::EC2::Client] An initialized Amazon EC2 client.
# @param cloudwatchlogs_client [Aws::CloudWatchLogs::Client]
#   An initialized Amazon CloudWatch Logs client.
# @param instance_id [String] The ID of the instance.
# @param log_group_name [String] The name of the log group.
# @return [Boolean] true if the instance was restarted and the information
#   was written to the log stream; otherwise, false.
#
def instance_restarted?(ec2_client, cloudwatchlogs_client, instance_id, log_group_name)
  log_stream_name = "#{Time.now.year}/#{Time.now.month}/#{Time.now.day}/" \
    "#{SecureRandom.uuid}"
  cloudwatchlogs_client.create_log_stream(
    log_group_name: log_group_name,
    log_stream_name: log_stream_name
  )
  sequence_token = ""

  puts "Attempting to stop the instance with the ID '#{instance_id}'. " \
    "This might take a few minutes..."
  if ec2_client.stop_instance(instance_id: instance_id)
    puts "Instance stopped.
    cloudwatchlogs_client.put_log_events(
      log_stream_name: log_stream_name,
      log_events: [
        { 
          timestamp: (Time.now.utc.to_f.round(3) * 1_000).to_i,
          message: message 
        } 
      ]
    ) unless sequence_token.empty?
    event[:sequence_token] = sequence_token
    response = cloudwatchlogs_client.put_log_events(event)
    puts "Message logged."
    return response.next_sequence_token
  rescue StandardError => e
    puts "Message not logged: #{e.message}"
  end
end
```

Restart an Amazon elastic compute cloud (Amazon EC2) instance and adds information about the related activity to a log stream in CloudWatch Logs.
Create and trigger a rule

```ruby
ec2_client.stop_instances(instance_ids: [instance_id])
ec2_client.wait_until(:instance_stopped, instance_ids: [instance_id])
puts "Instance stopped."
sequence_token = log_event(
  cloudwatchlogs_client,
  log_group_name,
  log_stream_name,
  "Instance '#{instance_id}' stopped."
)

puts "Attempting to restart the instance. This might take a few minutes..."
ec2_client.start_instances(instance_ids: [instance_id])
ec2_client.wait_until(:instance_running, instance_ids: [instance_id])
puts "Instance restarted."
sequence_token = log_event(
  cloudwatchlogs_client,
  log_group_name,
  log_stream_name,
  "Instance '#{instance_id}' restarted."
)
return true
rescue StandardError => e
  puts "Error creating log stream or stopping or restarting the instance: "
  "#{e.message}"
  log_event(
    cloudwatchlogs_client,
    log_group_name,
    log_stream_name,
    "Error stopping or starting instance '#{instance_id}': #{e.message}"
  )
  return false
end
```

Display information about activity for a rule in EventBridge.

```ruby
# Displays information about activity for a rule in Amazon EventBridge.
#
# Prerequisites:
# - A rule in Amazon EventBridge.
# @param cloudwatch_client [Amazon::CloudWatch::Client] An initialized
#   Amazon CloudWatch client.
# @param rule_name [String] The name of the rule.
# @param start_time [Time] The timestamp that determines the first datapoint
# to return. Can also be expressed as DateTime, Date, Integer, or String.
# @param end_time [Time] The timestamp that determines the last datapoint
# to return. Can also be expressed as DateTime, Date, Integer, or String.
# @param period [Integer] The interval, in seconds, to check for activity.
# @example
def display_rule_activity(
  cloudwatch_client,
  rule_name,
  start_time,
  end_time,
  period
)
Display log information for all of the log streams in a CloudWatch Logs log group.

```ruby
# Displays log information for all of the log streams in a log group in
# Amazon CloudWatch Logs.
#
# Prerequisites:
#
# - A log group in Amazon CloudWatch Logs.
#
# @param cloudwatchlogs_client [Amazon::CloudWatchLogs::Client] An initialized
#   Amazon CloudWatch Logs client.
# @param log_group_name [String] The name of the log group.
# @example
def display_log_data(cloudwatchlogs_client, log_group_name)
  puts "Attempting to display log stream data for the log group " \""named '#{log_group_name}'..."
  describe_log_streams_response = cloudwatchlogs_client.describe_log_streams(
    log_group_name: log_group_name,
    order_by: "LastEventTime",
    descending: true
  )
  if describe_log_streams_response.key?(:log_streams) &&
    describe_log_streams_response.log_streams.count.positive?
    describe_log_streams_response.log_streams.each do |log_stream|
      get_log_events_response = cloudwatchlogs_client.get_log_events(
        log_group_name: log_group_name,
        log_stream_name: log_stream.log_stream_name,
        start_time: Time.now - 1.day,
        end_time: Time.now,
        limit: 100
      )
      if get_log_events_response.key?(:events) &&
        get_log_events_response.events.count.positive?
        get_log_events_response.events.each do |event|
          puts "Event:
            " + event.message
        end
      else
        puts "Error getting log stream data for log group: #{log_stream.log_stream_name}"
      end
    end
  else
    puts "Error getting log stream data for log group: #{log_group_name}"
  end
end
```

```
puts "Attempting to display rule activity..."
response = cloudwatch_client.get_metric_statistics(
  namespace: "AWS/Events",
  metric_name: "Invocations",
  dimensions: [
    { name: "RuleName",
      value: rule_name
    },
  ],
  start_time: start_time,
  end_time: end_time,
  period: period,
  statistics: ["Sum"],
  unit: "Count"
)
if response.key?(:datapoints) && response.datapoints.count.positive?
  puts "The event rule '#{rule_name}' was triggered:
    response.datapoints.each do |datapoint|
      puts "  #{datapoint.sum} time(s) at #{datapoint.timestamp}"
    end
  else
    puts "The event rule '#{rule_name}' was not triggered during the " \""specified time period."
  end
rescue StandardError => e
  puts "Error getting information about event rule activity: #{e.message}"
end
```
Create and trigger a rule

```ruby
log_stream_name: log_stream.log_stream_name
) puts "Log messages for '#{log_stream.log_stream_name}':" puts "-" * (log_stream.log_stream_name.length + 20) if get_log_events_response.key?(:events) && get_log_events_response.events.count.positive? get_log_events_response.events.each do |event| puts event.message end else puts "No log messages for this log stream." end end rescue StandardError => e puts "Error getting information about the log streams or their messages: " puts "#{e.message}" end

Display a reminder to the caller to manually clean up any associated AWS resources that they no longer need.

# Displays a reminder to the caller to manually clean up any associated AWS resources that they no longer need.
# @param topic_name [String] The name of the Amazon SNS topic.
# @param role_name [String] The name of the IAM role.
# @param rule_name [String] The name of the Amazon EventBridge rule.
# @param log_group_name [String] The name of the Amazon CloudWatch Logs log group.
# @param instance_id [String] The ID of the Amazon EC2 instance.
# @example
def manual_cleanup_notice(topic_name, role_name, rule_name, log_group_name, instance_id)
  puts "-" * 10
  puts "Some of the following AWS resources might still exist in your account."
  puts "If you no longer want to use this code example, then to clean up"
  puts "your AWS account and avoid unexpected costs, you might want to"
  puts "manually delete any of the following resources if they exist:"
  puts "- The Amazon SNS topic named '#{topic_name}'."
  puts "- The IAM role named '#{role_name}'."
  puts "- The Amazon EventBridge rule named '#{rule_name}'."
  puts "- The Amazon CloudWatch Logs log group named '#{log_group_name}'."
  puts "- The Amazon EC2 instance with the ID '#{instance_id}'."
end

# Full example call:
def run_me
  # Properties for the Amazon SNS topic.
  topic_name = "aws-doc-sdk-examples-topic"
  email_address = "mary@example.com"
  # Properties for the IAM role.
  role_name = "aws-doc-sdk-examples-cloudwatch-events-rule-role"
  # Properties for the Amazon EventBridge rule.
  rule_name = "aws-doc-sdk-examples-ec2-state-change"
```

340
rule_description = "Triggers when any available EC2 instance starts."
instance_state = "running"

target_id = "sns-topic"

# Properties for the Amazon EC2 instance.
instance_id = "i-033c48ef067af3dEX"

# Properties for displaying the event rule's activity.
start_time = Time.now - 600 # Go back over the past 10 minutes
end_time = Time.now
period = 60 # Look back every 60 seconds over the past 10 minutes.

# Properties for the Amazon CloudWatch Logs log group.
log_group_name = "aws-doc-sdk-examples-cloudwatch-log"

# AWS service clients for this code example.
region = "us-east-1"
sts_client = Aws::STS::Client.new(region: region)
sns_client = Aws::SNS::Client.new(region: region)
iam_client = Aws::IAM::Client.new(region: region)
cloudwatchevents_client = Aws::CloudWatchEvents::Client.new(region: region)
ec2_client = Aws::EC2::Client.new(region: region)
cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
cloudwatchlogs_client = Aws::CloudWatchLogs::Client.new(region: region)

# Get the caller's account ID for use in forming
# Amazon Resource Names (ARNs) that this code relies on later.
account_id = sts_client.get_caller_identity.account

# If the Amazon SNS topic doesn't exist, create it.
topic_arn = "arn:aws:sns:#{region}:#{account_id}:#{topic_name}"
unless topic_exists?(sns_client, topic_arn)
topic_arn = create_topic(sns_client, topic_name, email_address)
if topic_arn == "Error"
  puts "Could not create the Amazon SNS topic correctly. Program stopped."
  manual_clean up notice(
    topic_name, role_name, rule_name, log_group_name, instance_id
  )
end
end

# If the IAM role doesn't exist, create it.
role_arn = "arn:aws:iam:#{account_id}:role/#{role_name}"
unless role_exists?(iam_client, role_arn)
role_arn = create_role(iam_client, role_name)
if role_arn == "Error"
  puts "Could not create the IAM role correctly. Program stopped."
  manual_clean up notice(
    topic_name, role_name, rule_name, log_group_name, instance_id
  )
end
end

# If the Amazon EventBridge rule doesn't exist, create it.
unless rule_exists?(cloudwatchevents_client, rule_name)
unless rule_created?(cloudwatchevents_client, rule_name, rule_description, instance_state, role_arn, target_id, topic_arn)
  puts "Could not create the Amazon EventBridge rule correctly. "
  "Program stopped."
  manual_clean up notice(
    topic_name, role_name, rule_name, log_group_name, instance_id
  )
end


```ruby

# If the Amazon CloudWatch Logs log group doesn't exist, create it.
unless log_group_exists?(cloudwatchlogs_client, log_group_name)
  unless log_group_created?(cloudwatchlogs_client, log_group_name)
    puts "Could not create the Amazon CloudWatch Logs log group " \
      "correctly. Program stopped."
    manual_cleanup_notice(
      topic_name, role_name, rule_name, log_group_name, instance_id
    )
  end
end

# Restart the Amazon EC2 instance, which triggers the rule.
unless instance_restarted?(ec2_client,
                          cloudwatchlogs_client,
                          instance_id,
                          log_group_name)
  puts "Could not restart the instance to trigger the rule. " \
    "Continuing anyway to show information about the rule and logs..."
end

# Display how many times the rule was triggered over the past 10 minutes.
display_rule_activity(
  cloudwatch_client,
  rule_name,
  start_time,
  end_time,
  period
)

# Display related log data in Amazon CloudWatch Logs.
display_log_data(cloudwatchlogs_client, log_group_name)

# Reminder the caller to clean up any AWS resources that are used
# by this code example and are no longer needed.
manual_cleanup_notice(
  topic_name, role_name, rule_name, log_group_name, instance_id
)
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.
  - `PutEvents`
  - `PutRule`

For a complete list of AWS SDK developer guides and code examples, see *Using EventBridge with an AWS SDK (p. 293)*. This topic also includes information about getting started and details about previous SDK versions.

**Get started with EventBridge rules and targets using an AWS SDK**

The following code examples show how to:
• Create a rule and add a target to it.
• Enable and disable rules.
• List and update rules and targets.
• Send events, then clean up resources.

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```csharp
public class EventBridgeScenario
{
    /*
    Before running this .NET code example, set up your development environment, including your credentials.

    This .NET example performs the following tasks with Amazon EventBridge:
    - Create a rule.
    - Add a target to a rule.
    - Enable and disable rules.
    - List rules and targets.
    - Update rules and targets.
    - Send events.
    - Delete the rule.
    */

    private static ILogger logger = null!;
    private static EventBridgeWrapper _eventBridgeWrapper = null!;
    private static IConfiguration _configuration = null!;

    private static IAmazonIdentityManagementService? _iamClient = null!;
    private static IAmazonSimpleNotificationService? _snsClient = null!;
    private static IAmazonS3 _s3Client = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft", LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace)
            )
            .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace)
            .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
```
.Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<EventBridgeScenario>();

ServicesSetup(host);

string topicArn = "";
string roleArn = "";

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the Amazon EventBridge example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    roleArn = await CreateRole();
    await CreateBucketWithEventBridgeEvents();
    await AddEventRule(roleArn);
    await ListEventRules();
    topicArn = await CreateSnsTopic();
    var email = await SubscribeToSnsTopic(topicArn);
    await AddSnsTarget(topicArn);
    await ListTargets();
    await ListRulesForTarget(topicArn);
    await UploadS3File(_s3Client);
    await ChangeRuleState(false);
    await GetRuleState();
    await UpdateSnsEventRule(topicArn);
    await ChangeRuleState(true);
    await UploadS3File(_s3Client);
    await UpdateToCustomRule(topicArn);
    await TriggerCustomRule(email);
    await CleanupResources(topicArn);
}
catch (Exception ex)
{
    logger.LogError(ex, "There was a problem executing the scenario.");
    await CleanupResources(topicArn);
}

Console.WriteLine(new string('-', 80));
Console.WriteLine("The Amazon EventBridge example scenario is complete.");
Console.WriteLine(new string('-', 80));

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _eventBridgeWrapper = host.Services.GetRequiredService<EventBridgeWrapper>();
    _snsClient = host.Services.GetRequiredService<IAmazonSimpleNotificationService>();
    _s3Client = host.Services.GetRequiredService<IAmazonS3>();
    _iamClient = host.Services.GetRequiredService<IAmazonIdentityManagementService>();
}

/// <summary>
/// Create a role to be used by EventBridge.
/// </summary>
/// <returns>The role Amazon Resource Name (ARN).</returns>
public static async Task<string> CreateRole()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating a role to use with EventBridge and attaching managed policy AmazonEventBridgeFullAccess.");
    Console.WriteLine(new string('-', 80));
    var roleName = _configuration["roleName"];
    var assumeRolePolicy = "{" +
        ""Version": "2012-10-17"," +
        ""Statement": [{" +
        ""Effect": "Allow"," +
        ""Principal": {
        "$""Service": "events.amazonaws.com" +
        "}}," +
        ""Action": "sts:AssumeRole" +
        "][" +
        "}";
    var roleResult = await _iamClient!.CreateRoleAsync( new CreateRoleRequest() {
        AssumeRolePolicyDocument = assumeRolePolicy,
        Path = "/",
        RoleName = roleName
    });

    await _iamClient.AttachRolePolicyAsync( new AttachRolePolicyRequest() {
        RoleName = roleName
    });
    // Allow time for the role to be ready.
    Thread.Sleep(10000);
    return roleResult.Role.Arn;
}

/// <summary>
/// Create an Amazon Simple Storage Service (Amazon S3) bucket with EventBridge events enabled.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CreateBucketWithEventBridgeEvents()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating an S3 bucket with EventBridge events enabled.");
    var testBucketName = _configuration["testBucketName"];
var bucketExists = await Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client, testBucketName);

if (!bucketExists)
{
    await _s3Client.PutBucketAsync(new PutBucketRequest()
    {
        BucketName = testBucketName,
        UseClientRegion = true
    });
}

await _s3Client.PutBucketNotificationAsync(new PutBucketNotificationRequest()
    {
        BucketName = testBucketName,
        EventBridgeConfiguration = new EventBridgeConfiguration()
    });

Console.WriteLine($"\tAdded bucket {testBucketName} with EventBridge events enabled.");

Console.WriteLine(new string('-', 80));

/// <summary>
/// Create and upload a file to an S3 bucket to trigger an event.
/// </summary>
/// <returns>Async task.</returns>
private static async Task UploadS3File(IAmazonS3 s3Client)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Uploading a file to the test bucket. This will trigger a subscription email.");

    var testBucketName = _configuration["testBucketName"];
    var fileName = "$\example_upload_{DateTime.UtcNow.Ticks}.txt";

    // Create the file if it does not already exist.
    if (!File.Exists(fileName))
    {
        await using StreamWriter sw = File.CreateText(fileName);
        await sw.WriteLineAsync("This is a sample file for testing uploads.");
    }

    await s3Client.PutObjectAsync(new PutObjectRequest()
    {
        FilePath = fileName,
        BucketName = testBucketName
    });

    Console.WriteLine($"\tPress Enter to continue.";
    Console.ReadLine();

    Console.WriteLine(new string('-', 80));

    /// <summary>
    /// Create an Amazon Simple Notification Service (Amazon SNS) topic to use as an EventBridge target.
    /// </summary>
    /// <returns>Async task.</returns>
private static async Task<string> CreateSnsTopic()
{
    Console.WriteLine(new string('-', 80));
    var topicName = _configuration[string]"topicName"];

    var topicAttributes = new Dictionary<string, string>()
    { "Policy", topicPolicy
    }
    var topicResponse = await _snsClient!.CreateTopicAsync(new
CreateTopicRequest()
    {
        Name = topicName,
        Attributes = topicAttributes
    });
    Console.WriteLine($"tAdded topic {topicName} for email subscriptions.");
    Console.WriteLine(new string('-', 80));
    return topicResponse.TopicArn;
}

/// <summary>
/// Subscribe a user email to an SNS topic.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>The user's email.</returns>
private static async Task<string> SubscribeToSnsTopic(string topicArn)
{
    string email = "";
    while (string.IsNullOrEmpty(email))
    {
        Console.WriteLine("Enter your email to subscribe to the Amazon SNS topic:");
        email = Console.ReadLine()!
    }
    var subscriptions = new List<string>();
    var paginatedSubscriptions = _snsClient!.Paginators.ListSubscriptionsByTopic(new ListSubscriptionsByTopicRequest()
    {TopicArn = topicArn
    },
Get started with rules and targets

// Get the entire list using the paginator.
await foreach (var subscription in paginatedSubscriptions.Subscriptions)
{
    subscriptions.Add(subscription.Endpoint);
}

if (subscriptions.Contains(email))
{
    Console.WriteLine("Your email is already subscribed.");
    Console.WriteLine(new string('-', 80));
    return email;
}

await _snsClient.SubscribeAsync(new SubscribeRequest()
{
    TopicArn = topicArn,
    Protocol = "email",
    Endpoint = email
});

Console.WriteLine("Use the link in the email you received to confirm your
subscription, then press Enter to continue.");
Console.ReadLine();

Console.WriteLine(new string('-', 80));
return email;

/// <summary>
/// Add a rule which triggers when a file is uploaded to an S3 bucket.
/// </summary>
/// <param name="roleArn">The ARN of the role used by EventBridge.</param>
/// <returns>Async task.</returns>
private static async Task AddEventRule(string roleArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating an EventBridge event that sends an email when
an Amazon S3 object is created.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];

    await _eventBridgeWrapper.PutS3UploadRule(roleArn, eventRuleName,
testBucketName);
    Console.WriteLine($"Added event rule {eventRuleName} for bucket
testBucketName.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add an SNS target to the rule.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>Async task.</returns>
private static async Task AddSnsTarget(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Adding a target to the rule that sends an email when
the rule is triggered.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];
    

var topicName = _configuration["topicName"];
await _eventBridgeWrapper.AddSnsTargetToRule(eventRuleName, topicArn);
Console.WriteLine($"\tAdded event rule {eventRuleName} with Amazon SNS
target {topicName} for bucket {testBucketName}";
)

foreach (var rule in rules)
    Console.WriteLine($"\tRule: {rule.Name} Description: {rule.Description} State: {rule.State}"
);

await _eventBridgeWrapper.UpdateS3UploadRuleTargetWithTransform(eventRuleName, topicArn);
Console.WriteLine($"\tUpdated event rule {eventRuleName} with Amazon SNS
target {topicArn} for bucket {testBucketName}";
)

await _eventBridgeWrapper.UpdateCustomEventPattern(eventRuleName);
Console.WriteLine($"\tUpdated event rule {eventRuleName} to custom
pattern.";
)

await _eventBridgeWrapper.UpdateCustomRuleTargetWithTransform(eventRuleName, topicArn);

Get started with rules and targets

```csharp
/// <summary>
/// Send rule events for a custom rule using the user's email address.
/// </summary>
/// <param name="email">The email address to include.</param>
/// <returns>Async task.</returns>
private static async Task TriggerCustomRule(string email)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Sending an event to trigger the rule. This will trigger a subscription email.");
    await _eventBridgeWrapper.PutCustomEmailEvent(email);
    Console.WriteLine("Events have been sent. Press Enter to continue.");
    Console.ReadLine();
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List all of the targets for a rule.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListTargets()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("List all of the targets for a particular rule.");
    var eventRuleName = _configuration["eventRuleName"];
    var targets = await _eventBridgeWrapper.ListAllTargetsOnRule(eventRuleName);
    targets.ForEach(t => Console.WriteLine($"Target: {t.Arn} Id: {t.Id} Input: {t.Input}"));
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List all of the rules for a particular target.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>Async task.</returns>
private static async Task ListRulesForTarget(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("List all of the rules for a particular target.");
    var rules = await _eventBridgeWrapper.ListAllRuleNamesByTarget(topicArn);
    rules.ForEach(r => Console.WriteLine($"Rule: {r}"));
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Enable or disable a particular rule.
/// </summary>
/// <param name="isEnabled">True to enable the rule, otherwise false.</param>
/// <returns>Async task.</returns>
private static async Task ChangeRuleState(bool isEnabled)
{
    Console.WriteLine(new string('-', 80));
```
var eventRuleName = _configuration["eventRuleName"];

if (!isEnabled)
{
    Console.WriteLine("Disabling the rule: {eventRuleName}");
    await _eventBridgeWrapper.DisableRuleByName(eventRuleName);
}
else
{
    Console.WriteLine("Enabling the rule: {eventRuleName}");
    await _eventBridgeWrapper.EnableRuleByName(eventRuleName);
}

Console.WriteLine(new string('-', 80));

/// <summary>
/// Get the current state of the rule.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetRuleState()
{
    var eventRuleName = _configuration["eventRuleName"];
    var state = await _eventBridgeWrapper.GetRuleStateByRuleName(eventRuleName);
    Console.WriteLine("Rule {eventRuleName} is in current state {state}");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic to clean up.</param>
/// <returns>Async task.</returns>
private static async Task CleanupResources(string topicArn)
{
    var eventRuleName = _configuration["eventRuleName"];
    if (GetYesNoResponse("\tDelete all targets and event rule {eventRuleName}? (y/n)"))
    {
        Console.WriteLine("\tRemoving all targets from the event rule.");
        await _eventBridgeWrapper.RemoveAllTargetsFromRule(eventRuleName);
        Console.WriteLine("\tDeleting event rule.");
        await _eventBridgeWrapper.DeleteRuleByName(eventRuleName);
    }
    var topicName = _configuration["topicName"];
    if (GetYesNoResponse("\tDelete Amazon SNS subscription topic {topicName}? (y/n)"))
    {
        Console.WriteLine("\tDeleting topic.");
        await _snsClient!.DeleteTopicAsync(new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    }
    var bucketName = _configuration["testBucketName"];
    if (GetYesNoResponse("\tDelete Amazon S3 bucket {bucketName}? (y/n)"))
Create a class that wraps EventBridge operations.

/// <summary>
/// Wrapper for Amazon EventBridge operations.
/// </summary>
public class EventBridgeWrapper
private readonly IAmazonEventBridge _amazonEventBridge;
private readonly ILogger<EventBridgeWrapper> _logger;

/// <summary>
/// Constructor for the EventBridge wrapper.
/// </summary>
/// <param name="amazonEventBridge">The injected EventBridge client.</param>
/// <param name="logger">The injected logger for the wrapper.</param>
public EventBridgeWrapper(IAmazonEventBridge amazonEventBridge,
ILogger<EventBridgeWrapper> logger)
{
    _amazonEventBridge = amazonEventBridge;
    _logger = logger;
}

/// <summary>
/// Get the state for a rule by the rule name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="eventBusName">The optional name of the event bus. If empty, uses the default event bus.</param>
/// <returns>The state of the rule.</returns>
public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string? eventBusName = null)
{
    var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
        new DescribeRuleRequest()
            {
                Name = ruleName,
                EventBusName = eventBusName
            });
    return ruleResponse.State;
}

/// <summary>
/// Enable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
        new EnableRuleRequest()
            {
                Name = ruleName
            });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Disable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
        new DisableRuleRequest()
            {
                Name = ruleName
            });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}
/// <summary>
/// List the rules on an event bus.
/// </summary>
/// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses the default event bus.</param>
/// <returns>The list of rules.</returns>
public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn = null) {
    var results = new List<Rule>();
    var request = new ListRulesRequest()
    {
        EventBusName = eventBusArn
    };
    // Get all of the pages of rules.
    ListRulesResponse response;
    do
    {
        response = await _amazonEventBridge.ListRulesAsync(request);
        results.AddRange(response.Rules);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);
    return results;
}

/// <summary>
/// List all of the targets matching a rule by name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>The list of targets.</returns>
public async Task<List<Target>> ListAllTargetsOnRule(string ruleName) {
    var results = new List<Target>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse response;
    do
    {
        response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
        results.AddRange(response.Targets);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);
    return results;
}

/// <summary>
/// List names of all rules matching a target.
/// </summary>
/// <param name="targetArn">The ARN of the target.</param>
/// <returns>The list of rule names.</returns>
public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn) {
    var results = new List<string>();
    var request = new ListRuleNamesByTargetRequest()
    {
        TargetArn = targetArn
    };
    ListRuleNamesByTargetResponse response;
    do
    {
        }
response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
results.AddRange(response.RuleNames);
request.NextToken = response.NextToken;
}

return results;

/// <summary>
/// Create a new event rule that triggers when an Amazon S3 object is created in a bucket.
/// </summary>
/// <param name="roleArn">The ARN of the role.</param>
/// <param name="ruleName">The name to give the rule.</param>
/// <param name="bucketName">The name of the bucket to trigger the event.</param>
/// <returns>The ARN of the new rule.</returns>
public async Task<string> PutS3UploadRule(string roleArn, string ruleName, string bucketName)
{
    string eventPattern = "{" +
        "source": ["aws.s3"]," +
        "detail-type": ["Object Created"]," +
        "detail":{" +
        "bucket":{" +
        "name": ["" + bucketName + "]" +
        "name": ["" + bucketName + "]" +
        "}" +
    "}};

    var response = await _amazonEventBridge.PutRuleAsync(
        new PutRuleRequest()
        {
            Name = ruleName,
            Description = "Example S3 upload rule for EventBridge",
            RoleArn = roleArn,
            EventPattern = eventPattern
        });

    return response.RuleArn;
}

/// <summary>
/// Update an Amazon S3 object created rule with a transform on the target.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="targetArn">The ARN of the target.</param>
/// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default event bus.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName, string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    var targets = new List<Target>
    {
        new Target()
        {
            Id = targetID,
            Arn = targetArn,
            InputTransformer = new InputTransformer()
            {
                InputPathsMap = new Dictionary<string, string>();
            }
        }
    }

    return targetID;
}
Get started with rules and targets

```csharp
{
    {"bucket", "$.detail.bucket.name"},
    {"time", "$.time"}
},
InputTemplate = "Notification: an object was uploaded to bucket <bucket> at <time>.
"
}
};
var response = await _amazonEventBridge.PutTargetsAsync(new PutTargetsRequest()
    {
    EventBusName = eventBusArn,
    Rule = ruleName,
    Targets = targets,
    });
if (response.FailedEntryCount > 0)
{
    response.FailedEntries.ForEach(e =>
    {
        _logger.LogError($"Failed to add target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}");
    });
return targetID;
}
```
$"Failed to add target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}";
}
return targetID;
}

/// <summary>
/// Add an event to the event bus that includes an email, message, and time.
/// </summary>
/// <param name="email">The email to use in the event detail of the custom event.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutCustomEmailEvent(string email)
{
    var eventDetail = new
    {
        UserEmail = email,
        Message = "This event was generated by example code."
        UtcTime = DateTime.UtcNow.ToString("g")
    };
    var response = await _amazonEventBridge.PutEventsAsync(
        new PutEventsRequest()
        {
            Entries = new List<PutEventsRequestEntry>()
            {
                new PutEventsRequestEntry()
                {
                    Source = "ExampleSource",
                    Detail = JsonSerializer.Serialize(eventDetail),
                    DetailType = "ExampleType"
                }
            }
        });
    return response.FailedEntryCount == 0;
}

/// <summary>
/// Update a rule to use a custom defined event pattern.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <returns>The ARN of the updated rule.</returns>
public async Task<string> UpdateCustomEventPattern(string ruleName)
{
    string customEventsPattern = "{" +
    "\"source\": \"ExampleSource\",\" +
    "\"detail-type\": \"ExampleType\" +
    "}";
    var response = await _amazonEventBridge.PutRuleAsync(
        new PutRuleRequest()
        {
            Name = ruleName,
            Description = "Custom test rule",
            EventPattern = customEventsPattern
        });
    return response.RuleArn;
}

/// <summary>
/// Add an Amazon SNS target topic to a rule.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <param name="targetArn">The ARN of the Amazon SNS target.</param>
/// <returns>True if successful.</returns>
public async Task<bool> AddSnsTargetToRule(string ruleName, string targetArn)
{
public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();
    // Create the list of targets and add a new target.
    var targets = new List<Target>
    {
        new Target()
        {
            Arn = targetArn,
            Id = targetID
        }
    };
    // Add the targets to the rule.
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });
    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError($"Failed to add target {e.TargetId}: {e.ErrorMessage}, code {e.ErrorCode}"");
        });
    }
    return targetID;
}

/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
{
    var targetIds = new List<string>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse targetsResponse;
    do
    {
        targetsResponse = await _amazonEventBridge.ListTargetsByRuleAsync(request);
        targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
        request.NextToken = targetsResponse.NextToken;
    } while (targetsResponse.NextToken is not null);
    var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
        new RemoveTargetsRequest()
        {
            Rule = ruleName,
        });
    return true;
}
Ids = targetIds

if (removeResponse.FailedEntryCount > 0)
{
    removeResponse.FailedEntries.ForEach(e =>
    {
        _logger.LogError(
            $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
            {e.ErrorCode}";
        });
    });

    return removeResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
This Java code example performs the following tasks:

1. Creates an AWS Identity and Access Management (IAM) role to use with Amazon EventBridge.
2. Amazon Simple Storage Service (Amazon S3) bucket with EventBridge events enabled.
3. Creates a rule that triggers when an object is uploaded to Amazon S3.
4. Lists rules on the event bus.
5. Creates a new Amazon Simple Notification Service (Amazon SNS) topic and lets the user subscribe to it.
6. Adds a target to the rule that sends an email to the specified topic.
7. Creates an EventBridge event that sends an email when an Amazon S3 object is created.
8. Lists Targets.
9. Lists the rules for the same target.
10. Triggers the rule by uploading a file to the Amazon S3 bucket.
11. Disables a specific rule.
12. Checks and print the state of the rule.
13. Adds a transform to the rule to change the text of the email.
14. Enables a specific rule.
15. Triggers the updated rule by uploading a file to the Amazon S3 bucket.
16. Updates the rule to be a custom rule pattern.
17. Sending an event to trigger the rule.
18. Cleans up resources.

```
public class EventbridgeMVP {
    public static final String DASHES = new String(new char[80]).replace("\0", "-");
    public static void main(String[] args) throws InterruptedException, IOException {
        final String usage = "\n" +
        "Usage:\n" +
        "   <roleName> <bucketName> <topicName> <eventRuleName>\n" +
        "Where:\n" +
        "   roleName - The name of the role to create.\n" +
        "   bucketName - The Amazon Simple Storage Service (Amazon S3) bucket name to create.\n" +
        "   topicName - The name of the Amazon Simple Notification Service (Amazon SNS) topic to create.\n" +
        "   eventRuleName - The Amazon EventBridge rule name to create.\n" +
        "\n";

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String polJSON = "{" +
            "  "Version": "\"2012-10-17\""," +
            "  "Statement": [{" +
            "    "Effect": "\"Allow\""," +
            "    "Principal": {" +
            "      "Service": "\"events.amazonaws.com\"" +
            "}}, +
            "    "Action": ["sts:AssumeRole" +
            "]" +
        "}";

        Scanner sc = new Scanner(System.in);
        String roleName = args[0];
        String bucketName = args[1];
    }
```
String topicName = args[2];
String eventRuleName = args[3];

Region region = Region.US_EAST_1;
EventBridgeClient eventBrClient = EventBridgeClient.builder()
    .region(region)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();

S3Client s3Client = S3Client.builder()
    .region(region)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();

Region regionGl = Region.AWS_GLOBAL;
IamClient iam = IamClient.builder()
    .region(regionGl)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();

SnsClient snsClient = SnsClient.builder()
    .region(region)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();

System.out.println(DASHES);
System.out.println("Welcome to the Amazon EventBridge example scenario.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("1. Create an AWS Identity and Access Management (IAM)
role to use with Amazon EventBridge.");
String roleArn = createIAMRole(iam, roleName, polJSON);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("2. Create an S3 bucket with EventBridge events
enabled.");
if (checkBucket(s3Client, bucketName)) {
    System.out.println("Bucket "+ bucketName +" already exists. Ending this
scenario.");
    System.exit(1);
}
createBucket(s3Client, bucketName);
Thread.sleep(3000);
setBucketNotification(s3Client, bucketName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("3. Create a rule that triggers when an object is
uploaded to Amazon S3.");
Thread.sleep(10000);
addEventRule(eventBrClient, roleArn, bucketName, eventRuleName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("4. List rules on the event bus.");
listRules(eventBrClient);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("5. Create a new SNS topic for testing and let the user
subscribe to the topic.");
String topicArn = createSnsTopic(snsClient, topicName);
System.out.println(DASHES);
System.out.println(DASHES);
System.out.println("6. Add a target to the rule that sends an email to the specified topic.");
System.out.println("Enter your email to subscribe to the Amazon SNS topic:");
String email = sc.nextLine();
subEmail(snsClient, topicArn, email);
System.out.println("Use the link in the email you received to confirm your subscription. Then, press Enter to continue.");
sc.nextLine();
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("7. Create an EventBridge event that sends an email when an Amazon S3 object is created.");
addSnsEventRule(eventBrClient, eventRuleName, topicArn, topicName, eventRuleName, bucketName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("8. List Targets.");
listTargets(eventBrClient, eventRuleName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("9. List the rules for the same target.");
listTargetRules(eventBrClient, topicArn);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("10. Trigger the rule by uploading a file to the S3 bucket.");
System.out.println("Press Enter to continue.");
sc.nextLine();
uploadTextFiletoS3(s3Client, bucketName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("11. Disable a specific rule.");
changeRuleState(eventBrClient, eventRuleName, false);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("12. Check and print the state of the rule.");
checkRule(eventBrClient, eventRuleName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("13. Add a transform to the rule to change the text of the email.");
updateSnsEventRule(eventBrClient, topicArn, eventRuleName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("14. Enable a specific rule.");
changeRuleState(eventBrClient, eventRuleName, true);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("15. Trigger the updated rule by uploading a file to the S3 bucket.");
System.out.println("Press Enter to continue.");
sc.nextLine();
uploadTextFiletoS3(s3Client, bucketName);
System.out.println(DASHES);
get started with rules and targets

16. Update the rule to be a custom rule pattern.
   updateToCustomRule(eventBrClient, eventRuleName);
   System.out.println("Updated event rule "+eventRuleName + " to use a custom
   pattern.");
   updateCustomRuleTargetWithTransform(eventBrClient, topicArn,
   eventRuleName);
   System.out.println("Updated event target "+topicArn + ".");
   System.out.println(DASHES);
   System.out.println(DASHES);
   System.out.println("17. Sending an event to trigger the rule. This will
   trigger a subscription email.");
   triggerCustomRule(eventBrClient, email);
   System.out.println("Events have been sent. Press Enter to continue.");
   sc.nextLine();
   System.out.println(DASHES);
   System.out.println(DASHES);
   System.out.println("18. Clean up resources.");
   String ans = sc.nextLine();
   if (ans.compareTo("y") == 0) {
     cleanupResources(eventBrClient, snsClient, s3Client, iam, topicArn,
     eventRuleName, bucketName, roleName );
   } else {
     System.out.println("The resources will not be cleaned up. ");
   }
   System.out.println(DASHES);
   System.out.println(DASHES);
   System.out.println("The Amazon EventBridge example scenario has
   successfully completed.");
   System.out.println(DASHES);
   }
   public static void cleanupResources(EventBridgeClient eventBrClient,
   SnsClient snsClient, S3Client s3Client, IamClient iam, String topicArn, String
   eventRuleName, String bucketName, String roleName) {
     System.out.println("Removing all targets from the event rule.");
     deleteTargetsFromRule(eventBrClient, eventRuleName);
     deleteRuleByName(eventBrClient, eventRuleName);
     deleteSNSTopic(snsClient, topicArn);
     deleteS3Bucket(s3Client, bucketName);
     deleteRole(iam, roleName);
   }
   public static void deleteRole(IamClient iam, String roleName) {
     String policyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess";
     DetachRolePolicyRequest policyRequest = DetachRolePolicyRequest.builder()
     .policyArn(policyArn)
     .roleName(roleName)
     .build();

     iam.detachRolePolicy(policyRequest);
     System.out.println("Successfully detached policy "+ policyArn + " from
     role "+ roleName);

     // Delete the role.
     DeleteRoleRequest roleRequest = DeleteRoleRequest.builder()
     .roleName(roleName)
     .build();

     iam.deleteRole(roleRequest);
     System.out.println("*** Successfully deleted "+ roleName);
   }
public static void deleteS3Bucket( S3Client s3Client, String bucketName) {
    // Remove all the objects from the S3 bucket.
    ListObjectsRequest listObjects = ListObjectsRequest.builder()
        .bucket(bucketName)
        .build();

    ListObjectsResponse res = s3Client.listObjects(listObjects);
    List<S3Object> objects = res.contents();
    ArrayList<ObjectIdentifier> toDelete = new ArrayList<>();

    for (S3Object myValue : objects) {
        toDelete.add(ObjectIdentifier.builder()
            .key(myValue.key())
            .build());
    }

    DeleteObjectsRequest dor = DeleteObjectsRequest.builder()
        .bucket(bucketName)
        .delete(Delete.builder()
            .objects(toDelete).build())
        .build();

    s3Client.deleteObjects(dor);

    // Delete the S3 bucket.
    DeleteBucketRequest deleteBucketRequest = DeleteBucketRequest.builder()
        .bucket(bucketName)
        .build();

    s3Client.deleteBucket(deleteBucketRequest);
    System.out.println("You have deleted the bucket and the objects");
}

// Delete the SNS topic.
public static void deleteSNSTopic(SnsClient snsClient, String topicArn ) {
    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
            .topicArn(topicArn)
            .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("\n\nStatus was " +
            result.sdkHttpResponse().statusCode());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteRuleByName(EventBridgeClient eventBrClient, String ruleName) {
    DeleteRuleRequest ruleRequest = DeleteRuleRequest.builder()
        .name(ruleName)
        .build();

    eventBrClient.deleteRule(ruleRequest);
    System.out.println("Successfully deleted the rule");
}

public static void deleteTargetsFromRule(EventBridgeClient eventBrClient, String eventRuleName) {
    // First, get all targets that will be deleted.
    ListTargetsByRuleRequest request = ListTargetsByRuleRequest.builder()
.rule(eventRuleName)
  .build();

ListTargetsByRuleResponse response =
eventBrClient.listTargetsByRule(request);
List<Target> allTargets = response.targets();

// Get all targets and delete them.
for (Target myTarget:allTargets) {
  RemoveTargetsRequest removeTargetsRequest =
    RemoveTargetsRequest.builder()
      .rule(eventRuleName)
      .ids(myTarget.id())
      .build();

  eventBrClient.removeTargets(removeTargetsRequest);
  System.out.println("Successfully removed the target");
}

public static void triggerCustomRule(EventBridgeClient eventBrClient, String email) {
  String json = "{" +
    "\"UserEmail\": "+email\"", " +
    "\"Message\": "This event was generated by example code.\", " +
    "\"UtcTime\": "Now.\" +
    "\"";

  PutEventsRequestEntry entry = PutEventsRequestEntry.builder()
    .source("ExampleSource")
    .detail(json)
    .detailType("ExampleType")
    .build();

  PutEventsRequest eventsRequest = PutEventsRequest.builder()
    .entries(entry)
    .build();

  eventBrClient.putEvents(eventsRequest);
}

public static void updateCustomRuleTargetWithTransform(EventBridgeClient eventBrClient, String topicArn, String ruleName){
  String targetId = java.util.UUID.randomUUID().toString();
  InputTransformer inputTransformer = InputTransformer.builder()
    .inputTemplate("\"Notification: sample event was received.\"")
    .build();

  Target target = Target.builder()
    .id(targetId)
    .arn(topicArn)
    .inputTransformer(inputTransformer)
    .build();

  try {
    PutTargetsRequest targetsRequest = PutTargetsRequest.builder()
      .rule(ruleName)
      .targets(target)
      .eventBusName(null)
      .build();

    eventBrClient.putTargets(targetsRequest);
  } catch (EventBridgeException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
  }
}
public static void updateToCustomRule(EventBridgeClient eventBrClient, String ruleName) {
    String customEventsPattern = "{" +
        ""source": ["ExampleSource"]," +
        ""detail-type": ["ExampleType"] +
    "}";

    PutRuleRequest request = PutRuleRequest.builder()
        .name(ruleName)
        .description("Custom test rule")
        .eventPattern(customEventsPattern)
        .build();

    eventBrClient.putRule(request);
}

// Update an Amazon S3 object created rule with a transform on the target.
public static void updateSnsEventRule(EventBridgeClient eventBrClient, String topicArn, String ruleName) {
    String targetId = java.util.UUID.randomUUID().toString();
    Map<String, String> myMap = new HashMap<>();
    myMap.put("bucket", "$.detail.bucket.name");
    myMap.put("time", "$.time");

    InputTransformer inputTransformer = InputTransformer.builder()
        .inputTemplate("Notification: an object was uploaded to bucket <bucket> at <time>.")
        .inputPathsMap(myMap)
        .build();

    Target target = Target.builder()
        .id(targetId)
        .arn(topicArn)
        .inputTransformer(inputTransformer)
        .build();

    try {
        PutTargetsRequest targetsRequest = PutTargetsRequest.builder()
            .rule(ruleName)
            .targets(target)
            .eventBusName(null)
            .build();

        eventBrClient.putTargets(targetsRequest);
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void checkRule(EventBridgeClient eventBrClient, String eventRuleName) {
    try {
        DescribeRuleRequest ruleRequest = DescribeRuleRequest.builder()
            .name(eventRuleName)
            .build();

        DescribeRuleResponse response = eventBrClient.describeRule(ruleRequest);
        System.out.println("The state of the rule is "+response.stateAsString());
    } catch (EventBridgeException e) {

public static void changeRuleState(EventBridgeClient eventBrClient, String eventRuleName, Boolean isEnabled) {
    try {
        if (!isEnabled) {
            System.out.println("Disabling the rule: "+eventRuleName);
            DisableRuleRequest ruleRequest = DisableRuleRequest.builder()
                        .name(eventRuleName)
                        .build();
            eventBrClient.disableRule(ruleRequest);
        } else {
            System.out.println("Enabling the rule: "+eventRuleName);
            EnableRuleRequest ruleRequest = EnableRuleRequest.builder()
                        .name(eventRuleName)
                        .build();
            eventBrClient.enableRule(ruleRequest);
        }
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

// Create and upload a file to an S3 bucket to trigger an event.
public static void uploadTextFiletoS3(S3Client s3Client, String bucketName) throws IOException {
    // Create a unique file name.
    String fileSuffix = new SimpleDateFormat("yyyyMMddHHmmss").format(new Date());
    String fileName = "TextFile"+fileSuffix+".txt";
    File myFile = new File(fileName);
    FileWriter fw = new FileWriter(myFile.getAbsoluteFile());
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("This is a sample file for testing uploads.");
    bw.close();
    try {
        PutObjectRequest putOb = PutObjectRequest.builder()
                .bucket(bucketName)
                .key(fileName)
                .build();
        s3Client.putObject(putOb, RequestBody.fromFile(myFile));
    } catch (S3Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void listTargetRules(EventBridgeClient eventBrClient, String topicArn) {
    ListRuleNamesByTargetRequest ruleNamesByTargetRequest = ListRuleNamesByTargetRequest.builder()
                .targetArn(topicArn)
                .build();
    ListRuleNamesByTargetResponse response = eventBrClient.listRuleNamesByTarget(ruleNamesByTargetRequest);
public static void listTargets(EventBridgeClient eventBrClient, String ruleName) {
    ListTargetsByRuleRequest ruleRequest = ListTargetsByRuleRequest.builder()
        .rule(ruleName)
        .build();

    ListTargetsByRuleResponse res = eventBrClient.listTargetsByRule(ruleRequest);
    List<Target> tagets = res.targets();
    for (Target target : tagets) {
        System.out.println("Target ARN: "+target.arn());
    }
}

// Add a rule which triggers an SNS target when a file is uploaded to an S3 bucket.
public static void addSnsEventRule(EventBridgeClient eventBrClient, String ruleName, String topicArn, String topicName, String eventRuleName, String bucketName) {
    String targetID = java.util.UUID.randomUUID().toString();
    Target myTarget = Target.builder()
        .id(targetID)
        .arn(topicArn)
        .build();

    List<Target> targets = new ArrayList<>();
    targets.add(myTarget);
    PutTargetsRequest request = PutTargetsRequest.builder()
        .eventBusName(null)
        .targets(targets)
        .rule(ruleName)
        .build();

    eventBrClient.putTargets(request);
    System.out.println("Added event rule "+eventRuleName + " with Amazon SNS target " +topicName + " for bucket "+bucketName + ").");
}

public static void subEmail(SnsClient snsClient, String topicArn, String email) {
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("email")
            .endpoint(email)
            .returnSubscriptionArn(true)
            .topicArn(topicArn)
            .build();

        SubscribeResponse result = snsClient.subscribe(request);
        System.out.println("Subscription ARN: "+ result.subscriptionArn() + 
        "\n\n Status is " + result.sdkHttpResponse().statusCode());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void listRules(EventBridgeClient eventBrClient) {
    try {
        List<String> rules = response.ruleNames();
        for (String rule : rules) {
            System.out.println("The rule name is " + rule);
        }
    }
ListRulesRequest rulesRequest = ListRulesRequest.builder()
    .eventBusName("default")
    .limit(10)
    .build();

ListRulesResponse response = eventBrClient.listRules(rulesRequest);
List<Rule> rules = response.rules();
for (Rule rule : rules) {
    System.out.println("The rule name is: \"+rule.name());
    System.out.println("The rule description is:");
    System.out.println("The rule state is: \"+rule.stateAsString());
}
} catch (EventBridgeException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

public static String createSnsTopic(SnsClient snsClient, String topicName) {
    String topicPolicy = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\": [{" +
        "\"Sid\": \"EventBridgePublishTopic\"," +
        "\"Effect\": \"Allow\"," +
        "\"Principal": {" +
        "\"Service": \"events.amazonaws.com\" +
        "}}," +
        "\"Resource\": "\"\"," +
        "\"Action\": "\"sns:Publish\"" +
        "]}]," +
    ");

    Map<String, String> topicAttributes = new HashMap<>();
    topicAttributes.put("Policy", topicPolicy);
    CreateTopicRequest topicRequest = CreateTopicRequest.builder()
        .name(topicName)
        .attributes(topicAttributes)
        .build();

    CreateTopicResponse response = snsClient.createTopic(topicRequest);
    System.out.println("Added topic \"+topicName \" for email subscriptions.");
    return response.topicArn();
}

// Create a new event rule that triggers when an Amazon S3 object is created in
// a bucket.
public static void addEventRule( EventBridgeClient eventBrClient, String roleArn, String bucketName, String eventRuleName) {
    String pattern = "{" +
        "\"source\": [\"aws.s3\"],\n        "\"detail-type\": [\"Object Created\"],\n        "\"detail\": {\n            \"bucket\": {' +
            "\"name\": [\"\"+bucketName\"\"]},\n            "\"name\": [\"\"+eventRuleName\"\"]} +
            "]}," +
    "\"id\": "\"\"," +
    "\"version\": 1 +
    ");

    try {
        PutRuleRequest ruleRequest = PutRuleRequest.builder()
            .description("Created by using the AWS SDK for Java v2")
            .name(eventRuleName)
            .eventPattern(pattern)
            .roleArn(roleArn)
            .build();

        PutRuleResponse response = eventBrClient.putRule(ruleRequest);
        System.out.println("Created event rule \"+eventRuleName \".
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
PutRuleResponse ruleResponse = eventBrClient.putRule(ruleRequest);
System.out.println("The ARN of the new rule is "+
ruleResponse.ruleArn());

} catch (EventBridgeException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
// Determine if the S3 bucket exists.
public static Boolean checkBucket(S3Client s3Client, String bucketName) {
    try {
        HeadBucketRequest headBucketRequest = HeadBucketRequest.builder()
            .bucket(bucketName)
            .build();
        s3Client.headBucket(headBucketRequest);
        return true;
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        return false;
    }
    // Set the S3 bucket notification configuration.
    public static void setBucketNotification(S3Client s3Client, String bucketName) {
        try {
            EventBridgeConfiguration eventBridgeConfiguration =
                EventBridgeConfiguration.builder()
                .build();
            NotificationConfiguration configuration =
                NotificationConfiguration.builder()
                .eventBridgeConfiguration(eventBridgeConfiguration)
                .build();
            PutBucketNotificationConfigurationRequest configurationRequest =
                PutBucketNotificationConfigurationRequest.builder()
                .bucket(bucketName)
                .notificationConfiguration(configuration)
                .skipDestinationValidation(true)
                .build();
            s3Client.putBucketNotificationConfiguration(configurationRequest);
            System.out.println("Added bucket " + bucketName + " with EventBridge events enabled.");
        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
    public static void createBucket(S3Client s3Client, String bucketName) {
        try {
            S3Waiter s3Waiter = s3Client.waiter();
            CreateBucketRequest bucketRequest = CreateBucketRequest.builder()
                .bucket(bucketName)
                .build();
            s3Client.createBucket(bucketRequest);
            HeadBucketRequest bucketRequestWait = HeadBucketRequest.builder()
public static String createIAMRole(IamClient iam, String rolename, String polJSON ) {
    try {
        CreateRoleRequest request = CreateRoleRequest.builder()
                .roleName(rolename)
                .assumeRolePolicyDocument(polJSON)
                .description("Created using the AWS SDK for Java")
                .build();

        CreateRoleResponse response = iam.createRole(request);
        AttachRolePolicyRequest rolePolicyRequest =
                AttachRolePolicyRequest.builder()
                .roleName(rolename)
                .policyArn("arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess")
                .build();

        iam.attachRolePolicy(rolePolicyRequest);
        return response.role().arn();
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

• For API details, see the following topics in AWS SDK for Java 2.x API Reference.
  • DeleteRule
  • DescribeRule
  • DisableRule
  • EnableRule
  • ListRuleNamesByTarget
  • ListRules
  • ListTargetsByRule
  • PutEvents
  • PutRule
  • PutTargets
Before running this Kotlin code example, set up your development environment, including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This Kotlin example performs the following tasks with Amazon EventBridge:

1. Creates an AWS Identity and Access Management (IAM) role to use with Amazon EventBridge.
2. Creates an Amazon Simple Storage Service (Amazon S3) bucket with EventBridge events enabled.
3. Creates a rule that triggers when an object is uploaded to Amazon S3.
4. Lists rules on the event bus.
5. Creates a new Amazon Simple Notification Service (Amazon SNS) topic and lets the user subscribe to it.
6. Adds a target to the rule that sends an email to the specified topic.
7. Creates an EventBridge event that sends an email when an Amazon S3 object is created.
8. Lists targets.
9. Lists the rules for the same target.
10. Triggers the rule by uploading a file to the S3 bucket.
11. Disables a specific rule.
12. Checks and prints the state of the rule.
13. Adds a transform to the rule to change the text of the email.
14. Enables a specific rule.
15. Triggers the updated rule by uploading a file to the S3 bucket.
16. Updates the rule to a custom rule pattern.
17. Sends an event to trigger the rule.
18. Cleans up resources.

```kotlin
val DASHES: String = String(CharArray(80)).replace("\u0000", "-")
suspend fun main(args: Array<String>) {
    val usage = ""
    Usage: 
    <roleName> <bucketName> <topicName> <eventRuleName>
    Where:
    roleName - The name of the role to create.
    bucketName - The Amazon Simple Storage Service (Amazon S3) bucket name to create.
    topicName - The name of the Amazon Simple Notification Service (Amazon SNS) topic to create.
    eventRuleName - The Amazon EventBridge rule name to create.
    ""
    val polJSON = "{" + 
    ""Version": "2012-10-17"," + 
    ""Statement": ["{" + 
    ""Effect": "Allow"," + 
    ""Principal": {" + 
    ""Service": "events.amazonaws.com"" + 
```
Get started with rules and targets

```scala
val roleName = args[0]
val bucketName = args[1]
val topicName = args[2]
val eventRuleName = args[3]

println(DASHES)
println("Welcome to the Amazon EventBridge example scenario.")
println(DASHES)
println(DASHES)
println("1. Create an AWS Identity and Access Management (IAM) role to use with Amazon EventBridge.")
val roleArn = createIAMRole(roleName, polJSON)
println(DASHES)
println(DASHES)
println("2. Create an S3 bucket with EventBridge events enabled.")
if (checkBucket(bucketName)) {
  println("$bucketName already exists. Ending this scenario.")
  exitProcess(1)
}
createBucket(bucketName)
delay(3000)
setBucketNotification(bucketName)
println(DASHES)
println(DASHES)
println("3. Create a rule that triggers when an object is uploaded to Amazon S3.")
delay(10000)
addEventRule(roleArn, bucketName, eventRuleName)
println(DASHES)
println(DASHES)
println("4. List rules on the event bus.")
listRules()
println(DASHES)
println(DASHES)
println("5. Create a new SNS topic for testing and let the user subscribe to the topic.")
val topicArn = createSnsTopic(topicName)
println(DASHES)
println(DASHES)
println("6. Add a target to the rule that sends an email to the specified topic.")
println("Enter your email to subscribe to the Amazon SNS topic:")
val email = sc.nextLine()
subEmail(topicArn, email)
println("Use the link in the email you received to confirm your subscription. Then press Enter to continue.")
sc.nextLine()
println(DASHES)
```
println(DASHES)
println("7. Create an EventBridge event that sends an email when an Amazon S3 object is created.")
addSnsEventRule(eventRuleName, topicArn, topicName, eventRuleName, bucketName)
println(DASHES)
println(DASHES)
println("8. List targets.")
listTargets(eventRuleName)
println(DASHES)
println(DASHES)
println("9. List the rules for the same target.")
listTargetRules(topicArn)
println(DASHES)
println(DASHES)
println("10. Trigger the rule by uploading a file to the S3 bucket.")
println("Press Enter to continue.")
sc.nextLine()
uploadTextFiletoS3(bucketName)
println(DASHES)
println(DASHES)
println("11. Disable a specific rule.")
changeRuleState(eventRuleName, false)
println(DASHES)
println(DASHES)
println("12. Check and print the state of the rule.")
checkRule(eventRuleName)
println(DASHES)
println(DASHES)
println("13. Add a transform to the rule to change the text of the email.")
updateSnsEventRule(topicArn, eventRuleName)
println(DASHES)
println(DASHES)
println("14. Enable a specific rule.")
changeRuleState(eventRuleName, true)
println(DASHES)
println(DASHES)
println("15. Trigger the updated rule by uploading a file to the S3 bucket.")
println("Press Enter to continue.")
sc.nextLine()
uploadTextFiletoS3(bucketName)
println(DASHES)
println(DASHES)
println("16. Update the rule to a custom rule pattern.")
updateToCustomRule(eventRuleName)
println("Updated event rule $eventRuleName to use a custom pattern.")
updateCustomRuleTargetWithTransform(topicArn, eventRuleName)
println("Updated event target $topicArn.")
println(DASHES)
println(DASHES)
println("17. Send an event to trigger the rule. This will trigger a subscription email.")
triggerCustomRule(email)
println("Events have been sent. Press Enter to continue.")
sc.nextLine()
println(DASHES)
println("18. Clean up resources.")
println("Do you want to clean up resources (y/n)")
val ans = sc.nextLine()
if (ans.compareTo("y") == 0) {
cleanupResources(topicArn, eventRuleName, bucketName, roleName)
} else {
    println("The resources will not be cleaned up. ")
}
println(DASHES)
println(DASHES)
println("The Amazon EventBridge example scenario has successfully completed.")
println(DASHES)

suspend fun cleanupResources(topicArn: String?, eventRuleName: String?, bucketName: String?, roleName: String?) {
    println("Removing all targets from the event rule.")
deleteTargetsFromRule(eventRuleName)
deleteRuleByRuleName(eventRuleName)
deleteSNSTopic(topicArn)
deleteS3Bucket(bucketName)
deleteRole(roleName)
}

suspend fun deleteRole(roleNameVal: String?) {
    val policyArnVal = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess"
    val policyRequest = DetachRolePolicyRequest {
        policyArn = policyArnVal
        roleName = roleNameVal
    }
    IamClient { region = "us-east-1" }.use { iam ->
        iam.detachRolePolicy(policyRequest)
        println("Successfully detached policy $policyArnVal from role $roleNameVal")
    }
    // Delete the role.
    val roleRequest = DeleteRoleRequest {
        roleName = roleNameVal
    }
    iam.deleteRole(roleRequest)
    println("*** Successfully deleted $roleNameVal")
}

suspend fun deleteS3Bucket(bucketName: String?) {
    // Remove all the objects from the S3 bucket.
    val listObjects = ListObjectsRequest {
        bucket = bucketName
    }
    S3Client { region = "us-east-1" }.use { s3Client ->
        val res = s3Client.listObjects(listObjects)
        val myObjects = res.contents
        val toDelete = mutableListOf<ObjectIdentifier>()
        if (myObjects != null) {
            for (myValue in myObjects) {
                toDelete.add(
                    ObjectIdentifier {
                        key = myValue.key
                    }
                )
            }
        }
    }
val delOb = Delete {
    objects = toDelete
}

val dor = DeleteObjectsRequest {
    bucket = bucketName
    delete = delOb
}
s3Client.deleteObjects(dor)

// Delete the S3 bucket.
val deleteBucketRequest = DeleteBucketRequest {
    bucket = bucketName
}
s3Client.deleteBucket(deleteBucketRequest)
println("You have deleted the bucket and the objects")

// Delete the SNS topic.
suspend fun deleteSNSTopic(topicArnVal: String?) {
    val request = DeleteTopicRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was deleted.")
    }
}

suspend fun deleteRuleByName(ruleName: String?) {
    val ruleRequest = DeleteRuleRequest {
        name = ruleName
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.deleteRule(ruleRequest)
        println("Successfully deleted the rule")
    }
}

suspend fun deleteTargetsFromRule(eventRuleName: String?) {
    // First, get all targets that will be deleted.
    val request = ListTargetsByRuleRequest {
        rule = eventRuleName
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.listTargetsByRule(request)
        val allTargets = response.targets

        // Get all targets and delete them.
        if (allTargets != null) {
            for (myTarget in allTargets) {
                val removeTargetsRequest = RemoveTargetsRequest {
                    rule = eventRuleName
                    ids = listOf(myTarget.id.toString())
                }

                eventBrClient.removeTargets(removeTargetsRequest)
                println("Successfully removed the target")
            }
        }
    }
}
suspend fun triggerCustomRule(email: String) {
    val json = "{" +
        "\"UserEmail\": \"" + email + \"\"," +
        "\"Message\": \"This event was generated by example code.\"" +
        "\"UtcTime\": \"Now.\"" +
    "}" +

    val entry = PutEventsRequestEntry {
        source = "ExampleSource"
        detail = json
        detailType = "ExampleType"
    }

    val eventsRequest = PutEventsRequest {
        this.entries = listOf(entry)
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putEvents(eventsRequest)
    }
}

suspend fun updateCustomRuleTargetWithTransform(topicArn: String?, ruleName: String?) {
    val targetId = UUID.randomUUID().toString()

    val inputTransformerOb = InputTransformer {
        inputTemplate = "\"Notification: sample event was received.\""
    }

    val target = Target {
        id = targetId
        arn = topicArn
        inputTransformer = inputTransformerOb
    }

    val targetsRequest = PutTargetsRequest {
        rule = ruleName
        targets = listOf(target)
        eventBusName = null
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putTargets(targetsRequest)
    }
}

suspend fun updateToCustomRule(ruleName: String?) {
    val customEventsPattern = "{" +
        "\"source\": [\"ExampleSource\"]," +
        "\"detail-type\": [\"ExampleType\"] +
    "}" +

    val request = PutRuleRequest {
        name = ruleName
        description = "Custom test rule"
        eventPattern = customEventsPattern
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putRule(request)
    }
}

// Update an Amazon S3 object created rule with a transform on the target.
suspend fun updateSnsEventRule(topicArn: String?, ruleName: String?) {

}
val targetId = UUID.randomUUID().toString()
val myMap = mutableMapOf<String, String>()
myMap["bucket"] = "$.detail.bucket.name"
myMap["time"] = "$.time"

val inputTransOb = InputTransformer {
    inputTemplate = "Notification: an object was uploaded to bucket <bucket> at <time>.
    inputPathsMap = myMap
}
val targetOb = Target {
    id = targetId
    arn = topicArn
    inputTransformer = inputTransOb
}

val targetsRequest = PutTargetsRequest {
    rule = ruleName
    targets = listOf(targetOb)
    eventBusName = null
}

EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
    eventBrClient.putTargets(targetsRequest)
}

suspend fun checkRule(eventRuleName: String?) {
    val ruleRequest = DescribeRuleRequest {
        name = eventRuleName
    }
    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.describeRule(ruleRequest)
        println("The state of the rule is $response")
    }
}

suspend fun changeRuleState(eventRuleName: String, isEnabled: Boolean?) {
    if (!isEnabled!!) {
        println("Disabling the rule: $eventRuleName")
        val ruleRequest = DisableRuleRequest {
            name = eventRuleName
        }
        EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
            eventBrClient.disableRule(ruleRequest)
        }
    } else {
        println("Enabling the rule: $eventRuleName")
        val ruleRequest = EnableRuleRequest {
            name = eventRuleName
        }
        EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
            eventBrClient.enableRule(ruleRequest)
        }
    }
}

// Create and upload a file to an S3 bucket to trigger an event.
@Throws(IOException::class)
suspend fun uploadTextFiletoS3(bucketName: String?) {
    val fileSuffix = SimpleDateFormat("yyyyMMddHHmmss").format(Date())
    val fileName = "TextFile$fileSuffix.txt"
    val myFile = File(fileName)
    val fw = FileWriter(myFile.absoluteFile)
    val bw = BufferedWriter(fw)
bw.write("This is a sample file for testing uploads.")
bw.close()

val putOb = PutObjectRequest {
    bucket = bucketName
    key = fileName
    body = myFile.asByteStream()
}

s3Client { region = "us-east-1" }.use { s3Client ->
    s3Client.putObject(putOb)
}

suspend fun listTargetRules(topicArnVal: String?) {
    val ruleNamesByTargetRequest = ListRuleNamesByTargetRequest {
        targetArn = topicArnVal
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.listRuleNamesByTarget(ruleNamesByTargetRequest)
        response.ruleNames?.forEach { rule ->
            println("The rule name is $rule")
        }
    }
}

suspend fun listTargets(ruleName: String?) {
    val ruleRequest = ListTargetsByRuleRequest {
        rule = ruleName
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.listTargetsByRule(ruleRequest)
        response.targets?.forEach { target ->
            println("Target ARN: ${target.arn}")
        }
    }
}

suspend fun addSnsEventRule(ruleName: String?, topicArn: String?, topicName: String, eventRuleName: String, bucketName: String) {
    val targetID = UUID.randomUUID().toString()
    val myTarget = Target {
        id = targetID
        arn = topicArn
    }

    val targetsOb = mutableListOf<Target>()
    targetsOb.add(myTarget)

    val request = PutTargetsRequest {
        eventBusName = null
        targets = targetsOb
        rule = ruleName
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        eventBrClient.putTargets(request)
        println("Added event rule $eventRuleName with Amazon SNS target $topicName for bucket $bucketName.")
    }
}
suspend fun subEmail(topicArnVal: String?, email: String?) {
    val request = SubscribeRequest {
        protocol = "email"
        endpoint = email
        returnSubscriptionArn = true
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        println("Subscription ARN: ${result.subscriptionArn}")
    }
}

suspend fun createSnsTopic(topicName: String): String? {
    val topicPolicy = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\": [{" +
            "\"Sid\": \"EventBridgePublishTopic\"," +
            "\"Effect\": \"Allow\"," +
            "\"Principal": {" +
            "\"Service": \"events.amazonaws.com\"," +
            "\"Resource": \"*\"," +
            "\"Action": \"sns:Publish\" +
        "]",
    "}"

    val topicAttributes = mutableMapOf<String, String>()
    topicAttributes["Policy"] = topicPolicy
    val topicRequest = CreateTopicRequest {
        name = topicName
        attributes = topicAttributes
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.createTopic(topicRequest)
        println("Added topic $topicName for email subscriptions.")
        return response.topicArn
    }
}

suspend fun listRules() {
    val rulesRequest = ListRulesRequest {
        eventBusName = "default"
        limit = 10
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.listRules(rulesRequest)
        response.rules?.forEach { rule ->
            println("The rule name is ${rule.name}")
            println("The rule ARN is ${rule.arn}")
        }
    }
}

// Create a new event rule that triggers when an Amazon S3 object is created in a
// bucket.
suspend fun addEventRule(roleArnVal: String?, bucketName: String, eventRuleName: String?) {
    val pattern = "" +
        "\"source\": ["aws.s3"],
        "detail-type": ["Object Created"],
        "detail": {" +
            "bucket": {" +
            "name": "$bucketName" +
        "}",
    "}"

    val ruleRequest = PutRuleRequest {
        name = eventRuleName +
        eventBusName = "default"
        eventPattern = pattern
        roleArn = roleArnVal
    }

    EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
        val response = eventBrClient.putRule(ruleRequest)
        println("Created event rule $eventRuleName")
    }
}
```kotlin
"bucket": {
    "name": ["$bucketName"]
}
"

val ruleRequest = PutRuleRequest {
    description = "Created by using the AWS SDK for Kotlin"
    name = eventRuleName
    eventPattern = pattern
    roleArn = roleArnVal
}

EventBridgeClient { region = "us-east-1" }.use { eventBrClient ->
    val ruleResponse = eventBrClient.putRule(ruleRequest)
    println("The ARN of the new rule is ${ruleResponse.ruleArn}"
)
}

// Set the Amazon S3 bucket notification configuration.
suspend fun setBucketNotification(bucketName: String) {
    val eventBridgeConfig = EventBridgeConfiguration {
    }
    val configuration = NotificationConfiguration {
        eventBridgeConfiguration = eventBridgeConfig
    }
    val configurationRequest = PutBucketNotificationConfigurationRequest {
        bucket = bucketName
        notificationConfiguration = configuration
        skipDestinationValidation = true
    }
    S3Client { region = "us-east-1" }.use { s3Client ->
        s3Client.putBucketNotificationConfiguration(configurationRequest)
        println("Added bucket $bucketName with EventBridge events enabled.
"
    }
}

// Create an S3 bucket using a waiter.
suspend fun createBucket(bucketName: String) {
    val request = CreateBucketRequest {
        bucket = bucketName
    }
    S3Client { region = "us-east-1" }.use { s3 ->
        s3.createBucket(request)
        s3.waitUntilBucketExists {
            bucket = bucketName
        }
        println("$bucketName is ready")
    }
}

suspend fun checkBucket(bucketName: String?): Boolean {
    try {
        // Determine if the S3 bucket exists.
        val headBucketRequest = HeadBucketRequest {
            bucket = bucketName
        }
        S3Client { region = "us-east-1" }.use { s3Client ->
            s3Client.headBucket(headBucketRequest)
            return true
        }
    } catch (e: Exception) {
        return false
    }
    return false
}
```

381
suspend fun createIAMRole(rolenameVal: String?, polJSON: String?): String? {
    val request = CreateRoleRequest {
        roleName = rolenameVal
        assumeRolePolicyDocument = polJSON
        description = "Created using the AWS SDK for Kotlin"
    }
    val rolePolicyRequest = AttachRolePolicyRequest {
        roleName = rolenameVal
        policyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess"
    }
    IamClient { region = "us-east-1" }.use { iam ->
        val response = iam.createRole(request)
        iam.attachRolePolicy(rolePolicyRequest)
        return response.role?.arn
    }
}

For API details, see the following topics in AWS SDK for Kotlin API reference.

- DeleteRule
- DescribeRule
- DisableRule
- EnableRule
- ListRuleNamesByTarget
- ListRules
- ListTargetsByRule
- PutEvents
- PutRule
- PutTargets

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.

Cross-service examples for EventBridge using AWS SDKs

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

Examples

- Use scheduled events to invoke a Lambda function (p. 383)
Use scheduled events to invoke a Lambda function

The following code examples show how to create an AWS Lambda function invoked by an Amazon EventBridge scheduled event.

Java

**SDK for Java 2.x**

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda Java runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](https://github.com).

**Services used in this example**

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

**SDK for JavaScript (v3)**

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda JavaScript runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](https://github.com).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](https://aws.amazon.com/).

**Services used in this example**

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

**SDK for Python (Boto3)**

This example shows how to register an AWS Lambda function as the target of a scheduled Amazon EventBridge event. The Lambda handler writes a friendly message and the full event data to Amazon CloudWatch Logs for later retrieval.
Use scheduled events to invoke a Lambda function

- Deploys a Lambda function.
- Creates an EventBridge scheduled event and makes the Lambda function the target.
- Grants permission to let EventBridge invoke the Lambda function.
- Prints the latest data from CloudWatch Logs to show the result of the scheduled invocations.
- Cleans up all resources created during the demo.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on GitHub.

**Services used in this example**

- CloudWatch Logs
- EventBridge
- Lambda

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 293). This topic also includes information about getting started and details about previous SDK versions.
Amazon EventBridge security

Amazon EventBridge uses AWS Identity and Access Management to control access to other AWS services and resources. For an overview of how IAM works, see Overview of Access Management in the IAM User Guide. For an overview of security credentials, see AWS Security Credentials in the Amazon Web Services General Reference.

Topics
- Data protection in Amazon EventBridge (p. 386)
- Tag-based policies (p. 387)
- Amazon EventBridge and AWS Identity and Access Management (p. 388)
- Log and monitor in Amazon EventBridge (p. 428)
- Compliance validation in Amazon EventBridge (p. 430)
- Amazon EventBridge resilience (p. 431)
- Infrastructure security in Amazon EventBridge (p. 432)
- Configuration and vulnerability analysis in Amazon EventBridge (p. 433)
Data protection in Amazon EventBridge

The AWS shared responsibility model applies to data protection in Amazon EventBridge. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers’ email addresses, into tags or free-form text fields such as a Name field. This includes when you work with EventBridge or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

EventBridge encrypts event metadata and message data that it stores. By default, EventBridge encrypts data using 256-bit Advanced Encryption Standard (AES-256) under an AWS owned key, which helps secure your data from unauthorized access. There is no additional charge for encrypting your data by using the AWS owned key.

Encryption in transit

EventBridge encrypts data that passes between EventBridge and other services by using Transport layer Security (TLS).
Tag-based policies

In Amazon EventBridge, you can use policies based on tags to control access to resources.

For example, you could restrict access to resources that include a tag with the key `environment` and the value `production`. The following example policy denies any resource with this tag the ability to create, delete, or modify tags, rules, or event buses for resources that have been tagged `environment/production`.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": [
            "events:PutRule",
            "events:DescribeRule",
            "events:DeleteRule",
            "events:Create EventBus",
            "events:DescribeEventBus",
            "events:DeleteEventBus"
         ],
         "Resource": "*",
         "Condition": {
            "StringEquals": {"aws:ResourceTag/environment": "production"}
         }
      }
   ]
}
```

For more information about tagging, see the following.

- Amazon EventBridge tags (p. 453)
- Controlling Access Using IAM Tags
Amazon EventBridge and AWS Identity and Access Management

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

Topics
- Authentication (p. 388)
- Access control (p. 389)
- Managing access permissions to your Amazon EventBridge resources (p. 390)
- Using identity-based policies (IAM policies) for Amazon EventBridge (p. 394)
- Using resource-based policies for Amazon EventBridge (p. 405)
- Cross-service confused deputy prevention (p. 409)
- Resource-based policies for Amazon EventBridge schemas (p. 411)
- Amazon EventBridge permissions reference (p. 414)
- Using IAM policy conditions for fine-grained access control (p. 416)

Authentication

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

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your account. These are your root credentials, and they provide complete access to all of your AWS resources.
  
  **Important**
  For security reasons, we recommend that you use the root credentials only to create an administrator, which is an IAM user with full permissions to your account. Then you can use this administrator to create other users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the IAM User Guide.

- **IAM user** – An IAM user is an identity within your account that has specific permissions, for example, permission to send event data to a target in EventBridge. You can use an IAM sign-in credentials to sign in to secure AWS webpages such as the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to sign-in credentials, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically to cryptographically sign your request, either through one of the SDKs or by using the AWS Command Line Interface (AWS CLI). If you don’t use AWS tools, you must sign the request yourself with Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the Amazon Web Services General Reference.

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

  - **Federated user access** – Instead of creating a user, you can use identities from AWS Directory Service, your enterprise user directory, or a web identity provider (IdP). These are known as federated users. AWS assigns a role to a federated user when the user requests access through an identity.
Access control

To create or access EventBridge resources, you need both valid credentials and permissions. For example, to invoke AWS Lambda, Amazon Simple Notification Service (Amazon SNS), and Amazon Simple Queue Service (Amazon SQS) targets, you must have permissions to those services.
Managing access permissions to your Amazon EventBridge resources

You manage access to EventBridge resources such as rules (p. 82) or events (p. 27) by using identity-based (p. 394) or resource-based (p. 405) policies.

EventBridge resources

EventBridge resources and subresources have unique Amazon Resource Names (ARNs) associated with them. You use ARNs in EventBridge to create event patterns. For more information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the Amazon Web Services General Reference.

For a list of operations EventBridge provides for working with resources, see Amazon EventBridge permissions reference (p. 414).

Note

Most services in AWS treat a colon (:) or a forward slash (/) as the same character in ARNs. However, EventBridge uses an exact match in event patterns (p. 62) and rules. Be sure to use the correct ARN characters when creating event patterns so that they match the ARN syntax in the event that you want to match.

The following table shows the resources in EventBridge.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archive</td>
<td>arn:aws:events:region:account:archive/archive-name</td>
</tr>
<tr>
<td>Replay</td>
<td>arn:aws:events:region:account:replay/replay-name</td>
</tr>
<tr>
<td>Rule</td>
<td>arn:aws:events:region:account:rule/[event-bus-name]/rule-name</td>
</tr>
<tr>
<td>Event bus</td>
<td>arn:aws:events:region:account:event-bus/event-bus-name</td>
</tr>
<tr>
<td>All EventBridge resources</td>
<td>arn:aws:events:*</td>
</tr>
<tr>
<td>All EventBridge resources owned by the specified account in the specified Region</td>
<td>arn:aws:events:region:account:*</td>
</tr>
</tbody>
</table>

The following example shows how to indicate a specific rule (myRule) in your statement using its ARN.

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

To specify all rules that belong to a specific account by using the asterisk (*) wildcard as follows.

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

To specify all resources, or if a specific API action doesn't support ARNs, use the asterisk (*) wildcard in the Resource element as follows.

"Resource": "*"
To specify multiple resources or PutTargets in a single statement, separate their ARNs with commas as follows.

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

**Resource ownership**

An account owns the resources in the account, no matter who creates the resources. The resource owner is the account of the *principal entity*, the account root user, an IAM user or role that authenticates the request to create the resource. The following examples illustrate how this works:

- If you use the root user credentials of your account to create a rule, your account is the owner of the EventBridge resource.
- If you create an user in your account and grant permissions to create EventBridge resources to that user, the user can create EventBridge resources. However, your account, which the user belongs to, owns the EventBridge resources.
- If you create an IAM role in your account with permissions to create EventBridge resources, anyone who can assume the role can create EventBridge resources. Your account, which the role belongs to, owns the EventBridge resources.

**Managing access to resources**

A permissions policy describes who has access to what. The following section explains the available options for creating permissions policies.

**Note**

This section discusses using IAM in the context of EventBridge. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What is IAM? in the IAM User Guide. For information about IAM policy syntax and descriptions, see IAM policy reference in the IAM User Guide.

Policies attached to an IAM identity are referred to as identity-based policies (IAM policies) and policies attached to a resource are referred to as resource-based policies. In EventBridge, you can use both identity-based (IAM policies) and resource-based policies.

**Topics**

- Identity-based policies (IAM policies) (p. 391)
- Resource-based policies (IAM policies) (p. 392)

**Identity-based policies (IAM policies)**

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

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

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

You can create specific IAM policies to restrict the calls and resources that users in your account have access to and then attach those policies to users. For more information about how to create IAM roles and to explore example IAM policy statements for EventBridge, see Managing access permissions to your Amazon EventBridge resources (p. 390).

Resource-based policies (IAM policies)

When a rule runs in EventBridge, all of the targets (p. 112) associated with the rule are invoked, which means invoking the AWS Lambda functions, publishing to the Amazon SNS topics, or relaying the event to the Amazon Kinesis streams. To make API calls on the resources that you own, EventBridge needs the appropriate permission. For Lambda, Amazon SNS, and Amazon SQS resources, EventBridge uses resource-based policies. For Kinesis streams, EventBridge uses IAM roles.

For more information about how to create IAM roles and to explore example resource-based policy statements for EventBridge, see Using resource-based policies for Amazon EventBridge (p. 405).

Specifying policy elements: actions, effects, and principals

For each EventBridge resource, EventBridge defines a set of API operations. To grant permissions for these API operations, EventBridge defines a set of actions that you can specify in a policy. Some API operations require permissions for more than one action to perform the API operation. For more information about resources and API operations, see EventBridge resources (p. 390) and Amazon EventBridge permissions reference (p. 414).

The following are the basic policy elements:

- **Resource** – Use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information, see EventBridge resources (p. 390).
- **Action** – Use keywords to identify resource operations that you want to allow or deny. For example, the events:Describe permission allows the user to perform the Describe operation.
- **Effect** – Specify either **allow** or **deny**. If you don’t explicitly grant access to (allow) a resource, access is denied. You can also explicitly deny access to a resource, which you do to make sure that a user can’t access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only).

For more information about IAM policy syntax and descriptions, see IAM JSON Policy Reference in the IAM User Guide.

For information about EventBridge API actions and the resources that they apply to, see Amazon EventBridge permissions reference (p. 414).

Specifying conditions in a policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date.
For more information about specifying conditions in a policy language, see Condition in the IAM User Guide.

To define conditions, you use condition keys. There are AWS condition keys and EventBridge specific keys that you can use as appropriate. For a complete list of AWS keys, see Available Keys for Conditions in the IAM User Guide. For a complete list of EventBridge specific keys, see Using IAM policy conditions for fine-grained access control (p. 416).
Using identity-based policies (IAM policies) for Amazon EventBridge

Identity-based policies are permissions policies that you attach to IAM identities.

Topics
- AWS managed policies for EventBridge (p. 394)
- Permissions required for EventBridge to access targets using IAM roles (p. 398)
- Customer-managed policy example: Using tagging to control access to rules (p. 401)
- Amazon EventBridge updates to AWS managed policies (p. 401)

AWS managed policies for EventBridge

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. Managed, or predefined, policies grant the necessary permissions for common use cases, so you don't need to investigate what permissions are needed. For more information, see AWS managed policies in the IAM User Guide.

The following AWS managed policies that you can attach to users in your account are specific to EventBridge:

- **AmazonEventBridgeFullAccess (p. 394)** – Grants full access to EventBridge, including EventBridge Pipes, EventBridge Schemas and EventBridge Scheduler.
- **AmazonEventBridgeReadOnlyAccess (p. 396)** – Grants read-only access to EventBridge, including EventBridge Pipes, EventBridge Schemas and EventBridge Scheduler.

AmazonEventBridgeFullAccess policy

The AmazonEventBridgeFullAccess policy grants permissions to use all EventBridge actions, as well as the following permissions:

- **iam:CreateServiceLinkedRole** – EventBridge requires this permission to create the service role in your account for API destinations. This permission grants only the IAM service permissions to create a role in your account specifically for API destinations.
- **iam:PassRole** – EventBridge requires this permission to pass an invocation role to EventBridge to invoke the target of a rule.
- **Secrets Manager permissions** – EventBridge requires these permissions to manage secrets in your account when you provide credentials through the connection resource to authorize API Destinations.

The following JSON shows the AmazonEventBridgeFullAccess policy.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EventBridgeActions",
            "Effect": "Allow",
            "Action": [
                "events:*",
                "schemas:*",
                "scheduler:*",
                "pipes:*"
            ],
            "Resource": "*"
        }
    ]
}
```
Using identity-based policies (IAM policies)

- **Sid**: "IAMCreateServiceLinkedRoleForApiDestinations",
  - **Effect**: "Allow",
  - **Action**: "iam:CreateServiceLinkedRole",
  - **Resource**: "arn:aws:iam::*:role/aws-service-role/AmazonEventBridgeApiDestinationsServiceRolePolicy",
  - **Condition**: {
      "StringEquals": {
        "iam:AWSServiceName": "apiregistry.destinations.amazonaws.com"
      }
    },

- **Sid**: "IAMCreateServiceLinkedRoleForAmazonEventBridgeSchemas",
  - **Effect**: "Allow",
  - **Action**: "iam:CreateServiceLinkedRole",
  - **Condition**: {
      "StringEquals": {
        "iam:AWSServiceName": "schemas.amazonaws.com"
      }
    },

- **Sid**: "SecretsManagerAccessForApiDestinations",
  - **Effect**: "Allow",
  - **Resource**: "arn:aws:secretsmanager:*::*:secret:events!*

- **Sid**: "IAMPassRoleAccessForEventBridge",
  - **Effect**: "Allow",
  - **Action**: "iam:PassRole",
  - **Resource**: "arn:aws:iam::*:role/**",
  - **Condition**: {
      "StringLike": {
        "iam:PassedToService": "events.amazonaws.com"
      }
    },

- **Sid**: "IAMPassRoleAccessForScheduler",
  - **Effect**: "Allow",
  - **Action**: "iam:PassRole",
  - **Resource**: "arn:aws:iam::*:role/**",
  - **Condition**: {
      "StringLike": {
        "iam:PassedToService": "scheduler.amazonaws.com"
      }
    },

- **Sid**: "IAMPassRoleAccessForPipes",
  - **Effect**: "Allow",
  - **Action**: "iam:PassRole",
  - **Resource**: "arn:aws:iam::*:role/**",
  - **Condition**: {
      "StringLike": {
        "iam:PassedToService": "pipes.amazonaws.com"
      }
    },

395
Using identity-based policies (IAM policies)

Note
The information in this section also applies to the CloudWatchEventsFullAccess policy. However, it is strongly recommended that you use Amazon EventBridge instead of Amazon CloudWatch Events.

AmazonEventBridgeReadOnlyAccess policy
The AmazonEventBridgeReadOnlyAccess policy grants permissions to use all read EventBridge actions. The following JSON shows the AmazonEventBridgeReadOnlyAccess policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule",
        "events:DescribeEventBus",
        "events:DescribeEventSource",
        "events:ListEventBuses",
        "events:ListEventSources",
        "events:ListRuleNamesByTarget",
        "events:ListRules",
        "events:ListTargetsByRule",
        "events:TestEventPattern",
        "events:DescribeArchive",
        "events:ListArchives",
        "events:DescribeReplay",
        "events:ListReplays",
        "events:DescribeConnection",
        "events:ListConnections",
        "events:DescribeApiDestination",
        "events:ListApiDestinations",
        "events:DescribeEndpoint",
        "events:ListEndpoints",
        "schemas:DescribeCodeBinding",
        "schemas:DescribeDiscoverer",
        "schemas:DescribeRegistry",
        "schemas:DescribeSchema",
        "schemas:ExportSchema",
        "schemas:GetCodeBindingSource",
        "schemas:GetDiscoveredSchema",
        "schemas:GetResourcePolicy",
        "schemas:ListDiscoverers",
        "schemas:ListRegistries",
        "schemas:ListSchemas",
        "schemas:ListSchemaVersions",
        "schemas:ListTagsForResource",
        "schemas:SearchSchemas",
        "scheduler:GetSchedule",
        "scheduler:GetScheduleGroup",
        "scheduler:ListSchedules",
        "scheduler:ListScheduleGroups",
        "scheduler:ListTagsForResource",
        "pipes:DescribePipe",
      ]
    }
  ]
}
```
Using identity-based policies (IAM policies)

Note
The information in this section also applies to the CloudWatchEventsReadOnlyAccess policy. However, it is strongly recommended that you use Amazon EventBridge instead of Amazon CloudWatch Events.

EventBridge Schema-specific managed policies

A schema defines the structure of events that are sent to EventBridge. EventBridge provides schemas for all events that are generated by AWS services. The following AWS managed policies specific to EventBridge Schemas are available:

- AmazonEventBridgeSchemasServiceRolePolicy
- AmazonEventBridgeSchemasFullAccess
- AmazonEventBridgeSchemasReadOnlyAccess

EventBridge Scheduler-specific managed policies

Amazon EventBridge Scheduler is a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. For AWS managed policies that are specific to EventBridge Scheduler, see AWS managed policies for EventBridge Scheduler in the EventBridge Scheduler User Guide.

EventBridge Pipes-specific managed policies

Amazon EventBridge Pipes connects event sources to targets. Pipes reduces the need for specialized knowledge and integration code when developing event driven architectures. This helps ensure consistency across your company’s applications. The following AWS managed policies specific to EventBridge Pipes are available:

- AmazonEventBridgePipesFullAccess
  Provides full access to Amazon EventBridge Pipes.
  Note
  This policy provides iam:PassRole – EventBridge Pipes requires this permission to pass an invocation role to EventBridge to create, and start pipes.

- AmazonEventBridgePipesReadOnlyAccess
  Provides read-only access to Amazon EventBridge Pipes.

- AmazonEventBridgePipesOperatorAccess
  Provides read-only and operator (that is, the ability to stop and start running Pipes) access to Amazon EventBridge Pipes.

IAM roles for sending events

To relay events to targets, EventBridge needs an IAM role.

To create an IAM role for sending events to EventBridge

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. To create an IAM role, follow the steps in Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide. As you follow the steps, do the following:

- In Role Name, use a name that is unique within your account.
- In Select Role Type, choose AWS Service Roles, and then choose Amazon EventBridge. This grants EventBridge permissions to assume the role.
- In Attach Policy, choose AmazonEventBridgeFullAccess.

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

Permissions required for EventBridge to access targets using IAM roles

EventBridge targets typically require IAM roles that grant permission to EventBridge to invoke the target. The following are some examples for various AWS services and targets. For others, use the EventBridge console to create a Rule and create a new Role which will be created with a policy with well-scoped permissions preconfigured.

Amazon SQS, Amazon SNS, Lambda, CloudWatch Logs, and EventBridge bus targets do not use roles, and permissions to EventBridge must be granted via a resource policy. API Gateway targets can use either resource policies or IAM roles.

If the target is an API destination, the role that you specify must include the following policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "events:InvokeApiDestination" ],
      "Resource": [ "arn:aws:events:::api-destination/*" ]
    }
  ]
}
```

If the target is a Kinesis stream, the role used to send event data to that target must include the following policy.

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

If the target is Systems Manager run command, and you specify one or more InstanceIds values for the command, the role that you specify must include the following policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "ssm:SendCommand" ],
      "Resource": "*"
    }
  ]
}
```
Using identity-based policies (IAM policies)

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

If the target is Systems Manager run command, and you specify one or more tags for the command, the role that you specify must include the following policy.

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

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

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

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

```json
{
  "Version": "2012-10-17",
  "Statement": [
  ]
}
```
Using identity-based policies (IAM policies)

```
"Statement": [{
    "Effect": "Allow",
    "Action": [
        "ecs:RunTask"
    ],
    "Resource": [
        "arn:aws:ecs:*:
           account-id:task-definition/task-definition-name"
    ],
    "Condition": {
        "ArnLike": {
            "ecs:cluster": "arn:aws:ecs:*:
               account-id:cluster/cluster-name"
        }
    }
},
{
    "Effect": "Allow",
    "Action":"iam:PassRole",
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringLike": {
            "iam:PassedToService": "ecs-tasks.amazonaws.com"
        }
    }
}]
```

The following policy allows built-in targets in EventBridge to perform Amazon EC2 actions on your behalf. You need to use the AWS Management Console to create rules with built-in targets.

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

The following policy allows EventBridge to relay events to the Kinesis streams in your account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord"
            ],
            "Resource": "*"
        }
    ]
}
```
Customer-managed policy example: Using tagging to control access to rules

The following example shows a user policy that grants permissions for EventBridge actions. This policy works when you use the EventBridge API, AWS SDKs, or the AWS CLI.

You can grant users access to specific EventBridge rules while preventing them from accessing other rules. To do so, you tag both sets of rules and then use IAM policies that refer to those tags. For more information about tagging EventBridge resources, see Amazon EventBridge tags (p. 453).

You can grant an IAM policy to a user to allow access to only the rules with a particular tag. You choose which rules to grant access to by tagging them with that particular tag. For example, the following policy grants a user access to rules with the value of Prod for the tag key Stack.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "events:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Stack": "Prod"
        }
      }
    }
  ]
}
```

For more information about using IAM policy statements, see Controlling Access Using Policies in the IAM User Guide.

Amazon EventBridge updates to AWS managed policies

View details about updates to AWS managed policies for EventBridge since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the EventBridge Document history page.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmazonEventBridgePipesFullAccess</td>
<td>EventBridge added managed policy for full permissions for using EventBridge Pipes.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>AmazonEventBridgePipesReadOnlyAccess</td>
<td>EventBridge added managed policy for permissions to view EventBridge Pipes information resources.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>AmazonEventBridgePipesOperatorAccess</td>
<td>EventBridge added managed policy for permissions to view EventBridge Pipes information, as well as start and stop running pipes.</td>
<td>December 1, 2022</td>
</tr>
</tbody>
</table>
The following table outlines the changes made to identity-based policies (IAM policies) for Amazon EventBridge:

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
</table>
| **AmazonEventBridgeFullAccess (p. 394)** | EventBridge updated the policy to include permissions necessary for using EventBridge Pipes features. The following permissions were added:  
  - EventBridge Schema Registry actions  
  - EventBridge Scheduler actions  
  - `iam:CreateServiceLinkedRole` permission for EventBridge Schema Registry  
  - `iam:PassRole` permission for EventBridge Scheduler | December 1, 2022 |
| **AmazonEventBridgeReadOnlyAccess (p. 396)** | EventBridge added permissions necessary for view EventBridge Pipes information resources. The following actions were added:  
  - `pipes:DescribePipe`  
  - `pipes:ListPipes`  
  - `pipes:ListTagsForResource` | December 1, 2022 |
| **CloudWatchEventsReadOnlyAccess** | Updated to match AmazonEventBridgeReadOnlyAccess. | December 1, 2022 |
| **CloudWatchEventsFullAccess (p. 394)** | Updated to match AmazonEventBridgeFullAccess. | December 1, 2022 |
| **AmazonEventBridgeFullAccess (p. 394)** | EventBridge updated the policy to include permissions necessary for using schemas and scheduler features. The following permissions were added:  
  - EventBridge Schema Registry actions  
  - EventBridge Scheduler actions  
  - `iam:CreateServiceLinkedRole` permission for EventBridge Schema Registry  
  - `iam:PassRole` permission for EventBridge Scheduler | November 10, 2022 |
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmazonEventBridgeReadOnlyAccess</td>
<td>EventBridge added permissions necessary for view endpoint information. The following actions were added: events:ListEndpoints, events:DescribeEndpoint</td>
<td>April 7, 2022</td>
</tr>
<tr>
<td>AmazonEventBridgeReadOnlyAccess</td>
<td>EventBridge added permissions necessary for view connection and API destination information. The following actions were added: events:DescribeConnection, events:ListConnections, events:DescribeApiDestination, events:ListApiDestinations</td>
<td>March 4, 2021</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| **AmazonEventBridgeFullAccess (p. 394)** | EventBridge updated the policy to include `iam:CreateServiceLinkedRole` and AWS Secrets Manager permissions necessary for using API destinations. The following actions were added:  
  - secretsmanager:CreateSecret  
  - secretsmanager:UpdateSecret  
  - secretsmanager:DeleteSecret  
  - secretsmanager:GetSecretValue  
  - secretsmanager:PutSecretValue | March 4, 2021 |
| EventBridge started tracking changes | EventBridge started tracking changes for its AWS managed policies. | March 4, 2021 |
Using resource-based policies for Amazon EventBridge

When a rule (p. 82) runs in EventBridge, all of the targets (p. 112) associated with the rule are invoked. Rules can invoke AWS Lambda functions, publish to Amazon SNS topics, or relay the event to Kinesis streams. To make API calls against the resources you own, EventBridge needs the appropriate permissions. For Lambda, Amazon SNS, Amazon SQS, and Amazon CloudWatch Logs resources, EventBridge uses resource-based policies. For Kinesis streams, EventBridge uses identity-based policies.

You use the AWS CLI to add permissions to your targets. For information about how to install and configure the AWS CLI, see Getting Set Up with the AWS Command Line Interface in the AWS Command Line Interface User Guide.

Topics

- Amazon API Gateway permissions (p. 405)
- CloudWatch Logs permissions (p. 405)
- AWS Lambda permissions (p. 406)
- Amazon SNS permissions (p. 407)
- Amazon SQS permissions (p. 408)
- EventBridge Pipes specifics (p. 409)

Amazon API Gateway permissions

To invoke your Amazon API Gateway endpoint by using a EventBridge rule, add the following permission to the policy of your API Gateway endpoint.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Action": "execute-api:Invoke",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:events:region:account-id:rule/rule-name"
        }
      },
      "Resource": [
        "execute-api:/stage/GET/api"
      ]
    }
  ]
}
```

CloudWatch Logs permissions

When CloudWatch Logs is the target of a rule, EventBridge creates log streams, and CloudWatch Logs stores the text from the events as log entries. To allow EventBridge to create the log stream and log the events, CloudWatch Logs must include a resource-based policy that enables EventBridge to write to CloudWatch Logs.
If you use the AWS Management Console to add CloudWatch Logs as the target of a rule, the resource-based policy is created automatically. If you use the AWS CLI to add the target, and the policy doesn’t already exist, you must create it.

The following example allows EventBridge to write to all log groups that have names that start with /aws/events/. If you use a different naming policy for these types of logs, adjust the example accordingly.

```
{
   "Statement": [
      {
         "Action": [
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Effect": "Allow",
         "Principal": {
            "Service": ["events.amazonaws.com", "delivery.logs.amazonaws.com"]
         },
         "Resource": "arn:aws:logs:region:account:log-group:/aws/events/*:*",
         "Sid": "TrustEventsToStoreLogEvent"
      }
   ],
   "Version": "2012-10-17"
}
```


**AWS Lambda permissions**

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

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

To add the above permissions that enable EventBridge to invoke Lambda functions using the AWS CLI:

- At a command prompt, enter the following command.

```
aws lambda add-permission --statement-id "InvokeLambdaFunction" \
   --action "lambda:InvokeFunction" \
   --principal "events.amazonaws.com" \
   --function-name "arn:aws:lambda:region:account-id:function:function-name" \
   --source-arn "arn:aws:events:region:account-id:rule/rule-name"
```
For more information about setting permissions that enable EventBridge to invoke Lambda functions, see `AddPermission` and `Using Lambda with Scheduled Events` in the AWS Lambda Developer Guide.

**Amazon SNS permissions**

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

**Note**

You can't use `Condition` blocks in Amazon SNS topic policies for EventBridge.

**To add permissions that enable EventBridge to publish SNS topics**

1. To list the attributes of an SNS topic, use the following command.

   ```bash
   ```

   The following example shows the result of a new SNS topic.

   ```json
   {
     "Attributes": {
       "SubscriptionsConfirmed": "0",
       "DisplayName": "",
       "SubscriptionsDeleted": "0",
       "EffectiveDeliveryPolicy": "{\"http\":{\"defaultHealthyRetryPolicy\":{
         \"minDelayTarget\":20,\"maxDelayTarget\":20,\"numRetries\":3,\"numMaxDelayRetries\":0,\"numNoDelayRetries\":0,\"numMinDelayRetries\":0,\"backoffFunction\":\"linear\"},
         \"disableSubscriptionOverrides\":false}}",
       "Owner": "account-id",
       "SubscriptionsPending": "0"
     }
   }
   ```

2. Use a JSON to string converter to convert the following statement to a string.

   ```json
   ```

   After you convert the statement to a string, it looks like the following example.

   ```json
   ```

3. Add the string you created in the previous step to the "Statement" collection inside the "Policy" attribute.
4. Use the `aws sns set-topic-attributes` command to set the new policy.

```bash
aws sns set-topic-attributes --topic-arn "arn:aws:sns:region:account-id:topic-name" \
--attribute-name Policy \
```

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

### Amazon SQS permissions

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

If the policy for the SQS queue is empty, you first need to create a policy and then you can add the permissions statement to it. A new SQS queue has an empty policy.

If the SQS queue already has a policy, you need to copy the original policy and combine it with a new statement to add the permissions statement to it.

**To add permissions that enable EventBridge rules to invoke an SQS queue**

1. To list SQS queue attributes. At a command prompt, enter the following command.

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

2. Add the following statement.

```json
{
    "Sid": "AWSEvents_custom-eventbus-ack-sqs-rule_dlq_sqs-rule-target",
    "Effect": "Allow",
    "Principal": { "Service": "events.amazonaws.com" },
    "Action": "sqs:SendMessage",
    "Condition": { " ArnEquals": { "aws:SourceArn": "arn:aws:events:region:account-id:rule/bus-name/rule-name" } }
}
```

3. Use a JSON to string converter to convert the preceding statement into a string. After you convert the policy to a string, it looks like the following.

```json
{"Sid": "EventsToMyQueue", "Effect": "Allow", "Principal": { "Service": "events.amazonaws.com" }, "Action": "sqs:SendMessage", "Resource":
```

---

408
Create a file called `set-queue-attributes.json` with the following content.

```json
{
    "Policy": "{
        "Version": "2012-10-17",
        "Id": "arn:aws:sqs:region:account-id:queue-name/SQSDefaultPolicy",
        "Statement": [{
            "Sid": "EventsToMyQueue",
            "Effect": "Allow",
            "Principal": {"Service": "events.amazonaws.com"},
            "Action": ["sqs:SendMessage"],
            "Condition": {"ArnEquals": {"aws:SourceArn": "arn:aws:events:region:account-id:rule/rule-name"}}
        }]
    }
}
```

5. Set the policy attribute by using the `set-queue-attributes.json` file you just created as the input, as shown in the following command.

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

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

EventBridge Pipes specifics

EventBridge Pipes does not support resource-based policies and has no APIs which support resource based policy conditions.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the calling service) calls another service (the called service). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that Amazon EventBridge gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:servicename::*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

Event buses

For EventBridge event bus rule targets, the value of `aws:SourceArn` must be the rule ARN.
The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in EventBridge to prevent the confused deputy problem. This example is for use in a role trust policy, for a role used by an EventBridge rule.

```json
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "ConfusedDeputyPreventionExamplePolicy",
        "Effect": "Allow",
        "Principal": {
            "Service": "events.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    },
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "arn:aws:events:*:123456789012:rule/myRule"
        },
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        }
    }
}
```

**EventBridge Pipes**

For EventBridge Pipes, the value of `aws:SourceArn` must be the pipe ARN.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in EventBridge to prevent the confused deputy problem. This example is for use in a role trust policy, for a role used by EventBridge Pipes.

```json
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "ConfusedDeputyPreventionExamplePolicy",
        "Effect": "Allow",
        "Principal": {
            "Service": "events.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    },
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "arn:aws:pipe:*:123456789012::pipe/example"
        },
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        }
    }
}
```
Resource-based policies for Amazon EventBridge schemas

The EventBridge schema registry (p. 230) supports resource-based policies (p. 405). A resource-based policy is a policy that is attached to a resource rather than to an IAM identity. For example, in Amazon Simple Storage Service (Amazon S3), a resource policy is attached to an Amazon S3 bucket.

For more information about EventBridge Schemas and resource-based policies, see the following.

- Amazon EventBridge Schemas REST API Reference
- Identity-Based Policies and Resource-Based Policies in the IAM User Guide

Supported APIs for resource-based policies

You can use the following APIs with resource-based policies for the EventBridge schema registry.

- DescribeRegistry
- UpdateRegistry
- DeleteRegistry
- ListSchemas
- SearchSchemas
- DescribeSchema
- CreateSchema
- DeleteSchema
- UpdateSchema
- ListSchemaVersions
- DeleteSchemaVersion
- DescribeCodeBinding
- GetCodeBindingSource
- PutCodeBinding

Example policy granting all supported actions to an AWS account

For the EventBridge schema registry, you must always attach a resource-based policy to a registry. To grant access to a schema, you specify the schema ARN and the registry ARN in the policy.

To grant a user access to all available APIs for EventBridge Schemas, use a policy similar to the following, replacing the "Principal" with the account ID of the account you want to grant access.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Test",
            "Effect": "Allow",
            "Action": [
                "schemas:*"
            ],
            "Principal": {
                "AWS": [
```
```
Resource-based policies for EventBridge schemas

Example policy granting read-only actions to an AWS account

The following example grants access to an account for only the read-only APIs for EventBridge schemas.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Test",
            "Effect": "Allow",
            "Action": [
                "schemas:DescribeRegistry",
                "schemas:ListSchemas",
                "schemas:SearchSchemas",
                "schemas:DescribeSchema",
                "schemas:ListSchemaVersions",
                "schemas:DescribeCodeBinding",
                "schemas:GetCodeBindingSource"
            ],
            "Principal": {
                "AWS": [
                    "109876543210"
                ]
            },
            "Resource": [
                "arn:aws:schemas:us-east-1:012345678901:registry/default",
                "arn:aws:schemas:us-east-1:012345678901:schema/default*"
            ]
        }
    ]
}
```

Example policy granting all actions to an organization

You can use resource-based policies with the EventBridge schema registry to grant access to an organization. For more information, see the AWS Organizations User Guide. The following example grants organization with an ID of o-a1b2c3d4e5 access to the schema registry.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Test",
            "Effect": "Allow",
            "Action": [
                "schemas:*"
            ],
            "Principal": "*",
            "Resource": [
                "arn:aws:schemas:us-east-1:012345678901:registry/default",
                "arn:aws:schemas:us-east-1:012345678901:schema/default*"
            ]
        }
    ]
}
```
"arn:aws:schemas:us-east-1:012345678901:schema/default"
],
"Condition": {
 "StringEquals": {
 "aws:PrincipalOrgID": [
 "o-a1b2c3d4e5"
 ]
 }
}
}
Amazon EventBridge permissions reference

To specify an action in an EventBridge policy, use the `events:` prefix followed by the API operation name, as shown in the following example.

```
"Action": "events:PutRule"
```

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

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

To specify multiple actions, you can also use wildcards. For example, you can specify all actions that begin with the word "Put" as follows.

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

To specify all EventBridge API actions, use the `*` wildcard as follows.

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

The following table lists the EventBridge API operations and corresponding actions that you can specify in an IAM policy.

<table>
<thead>
<tr>
<th>EventBridge API operation</th>
<th>Required permissions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteRule</td>
<td>events:DeleteRule</td>
<td>Required to delete a rule.</td>
</tr>
<tr>
<td>DescribeEventBus</td>
<td>events:DescribeEventBus</td>
<td>Required to list accounts that are allowed to write events to the current account's event bus.</td>
</tr>
<tr>
<td>DescribeRule</td>
<td>events:DescribeRule</td>
<td>Required to list the details about a rule.</td>
</tr>
<tr>
<td>DisableRule</td>
<td>events:DisableRule</td>
<td>Required to disable a rule.</td>
</tr>
<tr>
<td>EnableRule</td>
<td>events:EnableRule</td>
<td>Required to enable a rule.</td>
</tr>
<tr>
<td>ListRuleNamesByTarget</td>
<td>events:ListRuleNamesByTarget</td>
<td>Required to list rules associated with a target.</td>
</tr>
<tr>
<td>ListRules</td>
<td>events:ListRules</td>
<td>Required to list all rules in your account.</td>
</tr>
<tr>
<td>ListTagsForResource</td>
<td>events:ListTagsForResource</td>
<td>Required to list all tags associated with an EventBridge resource. Currently, only rules can be tagged.</td>
</tr>
<tr>
<td>ListTargetsByRule</td>
<td>events:ListTargetsByRule</td>
<td>Required to list all targets associated with a rule.</td>
</tr>
<tr>
<td>PutEvents</td>
<td>events:PutEvents</td>
<td>Required to add custom events that can be matched to rules.</td>
</tr>
<tr>
<td>EventBridge API operation</td>
<td>Required permissions</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PutPermission</td>
<td>events:PutPermission</td>
<td>Required to give another account permission to write events to this account's default event bus.</td>
</tr>
<tr>
<td>PutRule</td>
<td>events:PutRule</td>
<td>Required to create or update a rule.</td>
</tr>
<tr>
<td>PutTargets</td>
<td>events:PutTargets</td>
<td>Required to add targets to a rule.</td>
</tr>
<tr>
<td>RemovePermission</td>
<td>events:RemovePermission</td>
<td>Required to revoke another account's permissions for writing events to this account's default event bus.</td>
</tr>
<tr>
<td>RemoveTargets</td>
<td>events:RemoveTargets</td>
<td>Required to remove a target from a rule.</td>
</tr>
<tr>
<td>TestEventPattern</td>
<td>events:TestEventPattern</td>
<td>Required to test an event pattern against a given event.</td>
</tr>
</tbody>
</table>
Using IAM policy conditions for fine-grained access control

To grant permissions, you use the IAM policy language in a policy statement to specify the conditions when a policy should take effect. For example, you can have a policy that is applied only after a specific date.

A condition in a policy consists of key-value pairs. Condition keys aren't case sensitive.

If you specify multiple conditions or keys in a single condition, all conditions and keys must be met for EventBridge to grant permission. If you specify a single condition with multiple values for one key, EventBridge grants permission if one of the values is met.

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

By default, IAM users and roles can't access the events (p. 27) in your account. To access events, a user must be authorized for the PutRule API action. If an IAM user or role is authorized for the events:PutRule action, they can create a rule (p. 82) that matches certain events. However, for the rule to be useful, the user must also have permissions for the events:PutTargets action because, if you want the rule to do more than publish a CloudWatch metric, you must also add a target (p. 112) to a rule.

You can provide a condition in the policy statement of an IAM user or role that allows the user or role to create a rule that only matches a specific set of sources and event types. To grant access to specific sources and types of events, use the events:source and events:detail-type condition keys.

Similarly, you can provide a condition in the policy statement of an IAM user or role that allows the user or role to create a rule that only matches a specific resource in your accounts. To grant access to a specific resource, use the events:TargetArn condition key.

The following example is a policy that allows users to access all events except Amazon EC2 events in EventBridge using a deny statement on the PutRule API action.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyPutRuleForAllEC2Events",
            "Effect": "Deny",
            "Action": "events:PutRule",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "events:source": "aws.ec2"
                }
            }
        }
    ]
}
```

EventBridge condition keys

The following table shows the condition keys and key and value pairs that you can use in a policy in EventBridge.
### Condition key | Key value pair | Evaluation types
--- | --- | ---
aws:SourceAccount | The account in which the rule specified by aws:SourceArn exists. | Account Id, Null
aws:SourceArn | The ARN of the rule that is sending the event. | ARN, Null
events:creatorAccount | "events:creatorAccount": "creatorAccount" For creatorAccount, use the account ID for the account that created the rule. Use this condition to authorize API calls on rules from a specific account. | creatorAccount, Null
events:detail-type | "events:detail-type": "detail-type " Where detail-type is the literal string for the detail-type field of the event such as "AWS API Call via CloudTrail" and "EC2 Instance State-change Notification". | Detail Type, Null
events:detail.eventTypeCode | "events:detail.eventTypeCode": "eventTypeCode" For eventTypeCode, use the literal string for the detail.eventTypeCode field of the event, such as "AWS_ABUSE_DOS_REPORT". | eventTypeCode, Null
events:detail.service | "events:detail.service": "service" For service, use the literal string for the detail.service field of the event, such as "ABUSE". | service, Null
events:detail.userIdentity.principalId | "events:detail.userIdentity.principalId": "principal-id" For principal-id, use the literal string for the detail.userIdentity.principalId field of the event with detail-type "AWS API Call via CloudTrail" such as "AROAIDPPEZS35WEXAMPLE:AssumedRoleSessionName.". | principal-id, Null
events:eventBusInvocation | "events:eventBusInvocation": "boolean" For boolean, use true when a rule sends an event to a target that is an event bus in another account. Use | eventBusInvocation, Null
<table>
<thead>
<tr>
<th>Condition key</th>
<th>Key value pair</th>
<th>Evaluation types</th>
</tr>
</thead>
<tbody>
<tr>
<td>false when when a PutEvents API call is used.</td>
<td>events:ManagedBy</td>
<td>Used internally by AWS services. For a rule created by an AWS service on your behalf, the value is the principal name of the service that created the rule.</td>
</tr>
<tr>
<td>&quot;events:source&quot;: &quot;source&quot;</td>
<td>events:source</td>
<td>Use source for the literal string for the source field of the event such as &quot;aws.ec2&quot; or &quot;aws.s3&quot;. For more possible values for source, see the example events in Events from AWS services (p. 35).</td>
</tr>
<tr>
<td>&quot;events:TargetArn&quot;: &quot;target-arn&quot;</td>
<td>events:TargetArn</td>
<td>For target-arn, use the ARN of the target for the rule, for example &quot;arn:aws:lambda:<em>:</em>:function:*&quot;.</td>
</tr>
</tbody>
</table>

For example policy statements for EventBridge, see Managing access permissions to your Amazon EventBridge resources (p. 390).

Topics

- EventBridge Pipes specifics (p. 418)
- Example: Using the creatorAccount condition (p. 419)
- Example: Using the eventBusInvocation condition (p. 419)
- Example: Limiting access to a specific source (p. 419)
- Example: Defining multiple sources that can be used in an event pattern individually (p. 421)
- Example: Defining a source and a DetailType that can be used in an event pattern (p. 422)
- Example: Ensuring that the source is defined in the event pattern (p. 423)
- Example: Defining a list of allowed sources in an event pattern with multiple sources (p. 424)
- Example: Limiting PutRule access by detail.service (p. 425)
- Example: Limiting PutRule access by detail.eventTypeCode (p. 425)
- Example: Ensuring that only AWS CloudTrail events for API calls from a certain PrincipalId are allowed (p. 426)
- Example: Limiting access to targets (p. 427)

EventBridge Pipes specifics

EventBridge Pipes does not support any additional IAM policy condition keys.
Example: Using the creatorAccount condition

The following example policy statement shows how to use the creatorAccount condition in a policy to only allow rules to be created if the account specified as the creatorAccount is the account that created the rule.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowPutRuleForOwnedRules",
         "Effect": "Allow",
         "Action": "events:PutRule",
         "Resource": "*",
         "Condition": {
            "StringEqualsIfExists": {
               "events:creatorAccount": "${aws:PrincipalAccount}"  
            }
         }
      }
   ]
}
```

Example: Using the eventBusInvocation condition

The eventBusInvocation indicates whether the invocation originates from a cross-account target or a PutEvents API request. The value is **true** when the invocation results from a rule that include a cross-account target, such as when the target is an event bus in another account. The value is **false** when the invocation results from a PutEvents API request. The following example indicates an invocation from a cross-account target.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCrossAccountInvocationEventsOnly",
         "Effect": "Allow",
         "Action": "events:PutEvents",
         "Resource": "*",
         "Condition": {
            "BoolIfExists": {
               "events:eventBusInvocation": "true"
            }
         }
      }
   ]
}
```

Example: Limiting access to a specific source

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

**Policy A: allow all events**

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
      
   
```

```
```
"Sid": "AllowPutRuleForAllEvents",
"Effect": "Allow",
"Action": "events:PutRule",
"Resource": "*"
]
}

Policy B:—allow events only from Amazon EC2

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

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

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

The rule would be created because the policy allows for this specific source: that is, "aws.ec2". However, if the user with Policy B calls PutRule with an event pattern like the following, the rule creation would be denied because the policy doesn't allow for this specific source: that is, "aws.s3".

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

Essentially, the user with Policy B is only allowed to create a rule that would match the events originating from Amazon EC2; hence, they're only allowed access to the events from Amazon EC2.

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

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>Allowed by Policy A</th>
<th>Allowed by Policy B</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ] }</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ec2&quot;, &quot;aws.s3&quot; ] }</td>
<td>Yes</td>
<td>No (Source aws.s3 isn't allowed)</td>
</tr>
</tbody>
</table>
**IAM policy conditions**

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>Allowed by Policy A</th>
<th>Allowed by Policy B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ &quot;source&quot;: [ &quot;aws.ec2&quot; ], &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] ]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] ]</td>
<td>Yes</td>
<td>No (Source must be specified)</td>
</tr>
</tbody>
</table>

**Example: Defining multiple sources that can be used in an event pattern individually**

The following policy allows an IAM user or role to create a rule where the source in the EventPattern is either Amazon EC2 or Amazon ECS.

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

The following table shows some examples of event patterns that are allowed or denied by this policy.

<table>
<thead>
<tr>
<th>Event pattern</th>
<th>Allowed by the policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ] }</td>
<td>Yes</td>
</tr>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ecs&quot; ] }</td>
<td>Yes</td>
</tr>
<tr>
<td>[ ]</td>
<td>No</td>
</tr>
</tbody>
</table>
Example: Defining a source and a DetailType that can be used in an event pattern

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

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

The following table shows some examples of event patterns that are allowed or denied by this policy.

<table>
<thead>
<tr>
<th>Event pattern</th>
<th>Allowed by the policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{ &quot;source&quot;: [ &quot;aws.s3&quot; ] }</td>
<td>No</td>
</tr>
<tr>
<td>`{ &quot;source&quot;: [ &quot;aws.ec2&quot;, &quot;aws.ecs&quot; ] }</td>
<td>No</td>
</tr>
<tr>
<td>`{ &quot;detail-type&quot;: [ &quot;AWS API Call via CloudTrail&quot; ] }</td>
<td>No</td>
</tr>
<tr>
<td>{</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example: Ensuring that the source is defined in the event pattern

The following policy allows users to only create rules with EventPatterns that have the source field. With this policy, an IAM user or role can't create a rule with an EventPattern that doesn't provide a specific source.

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

The following table shows some examples of event patterns that are allowed or denied by this policy.

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>Allowed by the Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ], &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] }</code></td>
<td>Yes</td>
</tr>
<tr>
<td>`{ &quot;source&quot;: [ &quot;aws.ecs&quot;, &quot;aws.ec2&quot; ] }</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>Allowed by the Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{ &quot;source&quot;: [ &quot;aws.ec2&quot; ], &quot;detail-type&quot;: [ &quot;EC2 Instance Health Failed&quot; ] }</td>
<td>No</td>
</tr>
<tr>
<td>`{ &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] }</td>
<td>No</td>
</tr>
</tbody>
</table>
### Event Pattern

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>Allowed by the Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] }</code></td>
<td>No</td>
</tr>
</tbody>
</table>

### Example: Defining a list of allowed sources in an event pattern with multiple sources

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

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

The following table shows some examples of event patterns that are allowed or denied by this policy.

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>Allowed by the Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ] }</code></td>
<td>Yes</td>
</tr>
<tr>
<td><code>{ &quot;source&quot;: [ &quot;aws.ec2&quot;, &quot;aws.s3&quot; ] }</code></td>
<td>Yes</td>
</tr>
<tr>
<td><code>{ &quot;source&quot;: [ &quot;aws.ec2&quot;, &quot;aws.s3&quot;, &quot;aws.autoscaling&quot; ] }</code></td>
<td>No</td>
</tr>
</tbody>
</table>
Event Pattern | Allowed by the Policy
--- | ---
{   "detail-type": [ "EC2 Instance State-change Notification" ] } | No

**Example: Limiting PutRule access by detail.service**

You can restrict an IAM user or role to creating rules only for events that have a certain value in the `events:details.service` field. The value of `events:details.service` isn't necessarily the name of an AWS service.

This policy condition is helpful when you work with events from AWS Health that relate to security or abuse. By using this policy condition, you can limit access to these sensitive alerts to only those users who need to see them.

For example, the following policy allows the creation of rules only for events where the value of `events:details.service` is `ABUSE`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowPutRuleEventsWithDetailServiceEC2",
         "Effect": "Allow",
         "Action": "events:PutRule",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "events:detail.service": "ABUSE"
            }
         }
      }
   ]
}
```

**Example: Limiting PutRule access by detail.eventTypeCode**

You can restrict an IAM user or role to creating rules only for events that have a certain value in the `events:details.eventTypeCode` field. This policy condition is helpful when you work with events from AWS Health that relate to security or abuse. By using this policy condition, you can limit access to these sensitive alerts to only those users who need to see them.

For example, the following policy allows the creation of rules only for events where the value of `events:details.eventTypeCode` is `AWS_ABUSE_DOS_REPORT`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowPutRuleEventsWithDetailServiceEC2",
         "Effect": "Allow",
         "Action": "events:PutRule",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "events:detail.service": "ABUSE"
            }
         }
      }
   ]
}
```
"StringEquals": {  
  "events:detail.eventTypeCode": "AWS_ABUSE_DOS_REPORT"
}
}
]

Example: Ensuring that only AWS CloudTrail events for API calls from a certain PrincipalId are allowed

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

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

The following table shows some examples of event patterns that are allowed or denied by this policy.

<table>
<thead>
<tr>
<th>Event pattern</th>
<th>Allowed by the policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{ &quot;detail-type&quot;: [ &quot;AWS API Call via CloudTrail&quot; ] }</td>
<td>No</td>
</tr>
</tbody>
</table>
| `{ "detail-type": [ "AWS API Call via CloudTrail" ],
  "detail.userIdentity.principalId": [ "AIDAJ45Q7YFFAREXAMPLE" ] }          | Yes                   |
| `{ "detail-type": [ "AWS API Call via CloudTrail" ],
  "detail.userIdentity.principalId": [ "AROAIDPPEZS35WEXAMPLE:AssumedRoleSessionName" ] } | No                    |
Example: Limiting access to targets

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

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

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

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutTargetsOnASpecificRuleAndOnlyLambdaFunctions",
            "Effect": "Allow",
            "Action": "events:PutTargets",
            "Condition": {
                "ArnLike": {
                    "events:TargetArn": "arn:aws:lambda:*:*:function:*"
                }
            }
        }
    ]
}
```
Log and monitor in Amazon EventBridge

Amazon EventBridge works with AWS CloudTrail, a service that records actions from AWS services. CloudTrail captures API calls made by or on behalf of your AWS account from the EventBridge console and to EventBridge API operations.

Using the information collected by CloudTrail, you can determine what request was made to EventBridge, the IP address from which the request was made, who made the request, when it was made, and more.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

Topics
- EventBridge information in CloudTrail (p. 428)
- Example: EventBridge log file entries (p. 429)

EventBridge information in CloudTrail

CloudTrail is enabled on your AWS account when you create your account. When an event occurs in EventBridge, CloudTrail records the event 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 a record of events in your AWS account, including events for EventBridge, create a trail. A trail is a configuration that CloudTrail uses to deliver log files to an Amazon S3 bucket. By default, the trail logs events from all Regions in the AWS partition and then delivers the log files to an S3 bucket. You can configure other AWS services to analyze and act on 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

You can log the following EventBridge actions as events in CloudTrail log files:

- DeleteRule
- Describe EventBus
- Describe Rule
- Disable Rule
- Enable Rule
- List Rule Names By Target
- List Rules
- List Targets By Rule
- Put Permission
- Put Rule
- Put Targets
- Remove Targets
- Test Event Pattern
Every event and log entry contains information about who generated the entry. You can use this information to 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: EventBridge log file entries

A *trail* is a configuration that CloudTrail uses to deliver events as log files to an Amazon S3 bucket. CloudTrail log files contain log entries. An event represents a log entry, and it includes information about the requested action, the date and time of the action, and request parameters.

**Note**
CloudTrail log files don't appear in any specific order.

The following CloudTrail log file entry shows that a user called the EventBridge `PutRule` action.

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2015-11-17T23:56:15Z"
            }
        }
    },
    "eventTime": "2015-11-18T00:11:28Z",
    "eventSource": "events.amazonaws.com",
    "eventName": "PutRule",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS CloudWatch Console",
    "requestParameters": {
        "description": "",
        "name": "cttest2",
        "state": "ENABLED",
        "eventPattern": "{"source\:"["aws.ec2\"],\"detail-type\:"["EC2 Instance State-change Notification\"]\}"
    },
    "responseElements": {
        "ruleArn": "arn:aws:events:us-east-1:123456789012:rule/cttest2"
    },
    "requestID": "e9caf887-8d88-11e5-a331-3332aa445952",
    "eventID": "49d14f36-6450-44a5-a501-b0fcdcfae9b8",
    "eventType": "AwsApiCall",
    "apiVersion": "2015-10-07",
    "recipientAccountId": "123456789012"
}
```
Compliance validation in Amazon EventBridge

Third-party auditors such as SOC, PCI, FedRAMP, and HIPAA assess the security and compliance of AWS services as part of multiple AWS compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using EventBridge is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – Architectural considerations and steps for deploying security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper** – How companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – A collection of workbooks and guides.
- **Evaluating Resources with Rules** in the AWS Config Developer Guide – Information about how AWS Config assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- **AWS Security Hub** – A comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
Amazon EventBridge resilience

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.
Infrastructure security in Amazon EventBridge

As a managed service, Amazon EventBridge is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

You use AWS published API calls to access EventBridge through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

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

You can call these API operations from any network location, and you can use resource-based access policies (p. 405) in EventBridge, which can include restrictions based on the source IP address. You can also use EventBridge policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given EventBridge resource from only the specific VPC within the AWS network.
Configuration and vulnerability analysis in Amazon EventBridge

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.
Monitoring Amazon EventBridge

EventBridge sends metrics to Amazon CloudWatch every minute for everything from the number of matched events (p. 27) to the number of times a target (p. 112) is invoked by a rule (p. 82).

The following video reviews monitoring and auditing EventBridge behavior through CloudWatch: Monitoring and auditing events

Topics
- EventBridge metrics (p. 434)
- Dimensions for EventBridge metrics (p. 439)

EventBridge metrics

The AWS/Events namespace includes the following metrics.

For the metrics that use Count as a unit, Sum and SampleCount tend to be the most useful statistics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeadLetterInvocation</td>
<td>The number of times a rule's target isn't invoked in response to an event. This includes invocations that would result in running the same rule again, causing an infinite loop.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>The number of partner events ingested by EventBridge.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: EventSourceName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>FailedInvocations</td>
<td>The number of invocations that failed permanently. This doesn't include invocations that are retried or invocations that succeeded after a retry attempt. It also doesn't count failed invocations that are counted in DeadLetterInvocations.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> EventBridge only sends this metric to CloudWatch if it isn't zero.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>Invocations</td>
<td>The number of times a target is invoked by a rule in response to an event. This includes successful and failed invocations, but doesn't include throttled or retried attempts until they fail permanently. It doesn't include DeadLetterInvocations.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> EventBridge only sends this metric to CloudWatch if it isn't zero.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None, RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>InvocationAttempt</td>
<td>Number of times EventBridge attempted invoking a target.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>InvocationsCreated</td>
<td>The total number of invocations created in response to each event. This metric is often used to monitor utilization of the Invocations throttle limit in transactions per second EventBridge service quota (p. 447).</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>InvocationsFailedToBeSentToDlq</td>
<td>Number of invocations that couldn't be moved to a dead-letter queue. Dead-letter queue errors occur due to permissions errors, unavailable resources, or size limits.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> EventBridge only sends this metric to CloudWatch if it isn't zero.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InvocationIngestiontoInvocationCompleteLatency</td>
<td>The time taken from event ingestion to completion of the first successful invocation attempt.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: EventBusName, None, RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
</tr>
<tr>
<td>InvocationIngestiontoInvocationStartLatency</td>
<td>The time to process events, measured from when an event is ingested by EventBridge to the first invocation of a target.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None, RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
</tr>
<tr>
<td>InvocationsSentToDlq</td>
<td>The number of invocations that are moved to a dead-letter queue.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> EventBridge only sends this metric to CloudWatch if it isn't zero.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>MatchedEvents</td>
<td>If EventBusName or EventSourceName is specified, the number of events that matched with any rule. If RuleName is specified, the number of events that matched with a specific rule.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: EventBusName, RuleName, EventSourceName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>RetryInvocationAttempts</td>
<td>The number of times target invocation has been retried.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> EventBridge only sends this metric to CloudWatch if it isn't zero.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>SuccessfulInvocationAttempts</td>
<td>The number of times target was successfully invoked.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>ThrottledRules</td>
<td>The number of times rule execution was throttled. Invocations for those rules may be delayed.</td>
</tr>
<tr>
<td></td>
<td>For more information, see <strong>Invocations throttle limit in transactions per second</strong> in ??? (p. 447).</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TriggeredRules</td>
<td>The number of rules that have run and matched with any event. You won't see this metric in CloudWatch until a rule is triggered.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
</tbody>
</table>

**EventBridge PutEvents metrics**

The AWS/Events namespace includes the following metrics pertaining to the [PutEvents](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html) API requests.

For the metrics that use Count as a unit, Sum and SampleCount tend to be the most useful statistics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PutEventsApproximateCallCount</td>
<td>Approximate number of received <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> requests.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>PutEventsApproximateFailedCount</td>
<td>Approximate number of failed <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> requests.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>PutEventsApproximateSuccessCount</td>
<td>Approximate number of successful <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> requests.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>PutEventsApproximateThrottledCount</td>
<td>Number of <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> requests rejected due to throttling.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>PutEventsEntriesCount</td>
<td>The number of event entries contained in a <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> request.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>PutEventsFailedEntriesCount</td>
<td>The number of event entries contained in a <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> request that failed to be ingested.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>PutEventsLatency</td>
<td>The time taken per <a href="https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/events.html">PutEvents</a> request.</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: None</td>
</tr>
</tbody>
</table>
### EventBridge PutPartnerEvents metrics

The AWS/Events namespace includes the following metrics pertaining to the `PutPartnerEvents` API requests.

**Note**
EventBridge only includes metrics pertaining to `PutPartnerEvents` requests in SaaS partner accounts that send events. For more information, see [???(p. 42)](#).

For the metrics that use Count as a unit, Sum and SampleCount tend to be the most useful statistics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PutPartnerEventsApproximateCallCount</code></td>
<td>Approximate number of received <code>PutPartnerEvents</code> requests.</td>
</tr>
<tr>
<td><code>PutPartnerEventsApproximateFailedCount</code></td>
<td>Approximate number of failed <code>PutPartnerEvents</code> requests.</td>
</tr>
<tr>
<td><code>PutPartnerEventsApproximateThrottledCount</code></td>
<td>Number of <code>PutPartnerEvents</code> requests rejected due to throttling.</td>
</tr>
<tr>
<td><code>PutPartnerEventsApproximateSuccessCount</code></td>
<td>Approximate number of successful <code>PutPartnerEvents</code> requests.</td>
</tr>
<tr>
<td><code>PutPartnerEventsEntriesCount</code></td>
<td>The number of event entries contained in a <code>PutPartnerEvents</code> request.</td>
</tr>
<tr>
<td><code>PutPartnerEventsFailedEntriesCount</code></td>
<td>The number of event entries contained in a <code>PutPartnerEvents</code> request that failed to be ingested.</td>
</tr>
</tbody>
</table>

---

Units: Milliseconds

- `PutEventsRequestSize`:
  - The size of the `PutEvents` request.
  - Valid Dimensions: None
  - Units: Bytes
### Dimensions for EventBridge metrics

EventBridge metrics have *dimensions*, or sortable attributes, which are listed below.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventBusName</td>
<td>Filters the available metrics by event bus name.</td>
</tr>
<tr>
<td>EventSourceName</td>
<td>Filters the available metrics by partner event source name.</td>
</tr>
<tr>
<td>RuleName</td>
<td>Filters the available metrics by rule name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PutPartnerEventsLatency</td>
<td>The time taken per <em>PutPartnerEvents</em> request. Valid Dimensions: None. Units: Milliseconds</td>
</tr>
</tbody>
</table>
Troubleshooting Amazon EventBridge

You can use the steps in this section to troubleshoot Amazon EventBridge.

Topics
- **My rule ran but my Lambda function wasn't invoked** (p. 440)
- **I just created or modified a rule, but it didn't match a test event** (p. 441)
- **My rule didn't run at the time I specified in the ScheduleExpression** (p. 442)
- **My rule didn't run at the time that I expected** (p. 442)
- **My rule matches AWS global service API calls but it didn't run** (p. 442)
- **The IAM role associated with my rule is being ignored when the rule runs** (p. 443)
- **My rule has an event pattern that is supposed to match a resource, but no events match** (p. 443)
- **My event's delivery to the target was delayed** (p. 443)
- **Some events were never delivered to my target** (p. 443)
- **My rule ran more than once in response to one event** (p. 443)
- **Preventing infinite loops** (p. 444)
- **My events are not delivered to the target Amazon SQS queue** (p. 444)
- **My rule runs, but I don't see any messages published into my Amazon SNS topic** (p. 444)
- **My Amazon SNS topic still has permissions for EventBridge even after I deleted the rule associated with the Amazon SNS topic** (p. 445)
- **Which IAM condition keys can I use with EventBridge?** (p. 446)
- **How can I tell when EventBridge rules are broken?** (p. 446)

---

My rule ran but my Lambda function wasn't invoked

One reason your Lambda function might not run is if you don't have the right permissions.

**To check your permissions for your Lambda function**

1. Using the AWS CLI, run the following command with your function and your AWS Region:

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

   You should see the following output.

   ```json
   {
     "Policy": "\"Version\": \"2012-10-17\",
     "Statement": [
       {"Condition": {"ArnLike": {"AWS:SourceArn": \"arn:aws:events:us-east-1:123456789012:rule/MyRule\"}}},
   ```

---

440
I just created or modified a rule, but it didn't match a test event

When you make a change to a rule (p. 82) or to its targets (p. 112), incoming events (p. 27) might not immediately start or stop matching to new or updated rules. Allow a short period of time for changes to take effect.

If events still don't match after a short period of time, check the CloudWatch metrics TriggeredRules, Invocations, and FailedInvocations for your rule. For more information about these metrics, see Monitoring Amazon EventBridge (p. 434).

If the rule is intended to match an event from an AWS service, do one of these things:
My rule didn't run at the time I specified in the ScheduleExpression

Make sure you have set the schedule for the rule (p. 82) in the UTC+0 time zone. If the ScheduleExpression is correct, then follow the steps under I just created or modified a rule, but it didn't match a test event (p. 441).

My rule didn't run at the time that I expected

EventBridge runs rules (p. 82) within one minute of the start time you set. The count down to run time begins as soon as you create the rule.

Note
Scheduled rules have delivery type of guaranteed meaning events will be triggered for each expected time at least once.

You can use a cron expression to invoke targets (p. 112) at a specified time. To create a rule that runs every four hours on the 0th minute, you do one of the following:

- In the EventBridge console, you use the cron expression `0 0/4 * * ? *`.
- Using the AWS CLI, you use the expression `cron(0 0/4 * * ? *)`.

For example, to create a rule named TestRule that runs every 4 hours by using the AWS CLI, you use the following command.

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

To run a rule every five minutes, you use the following cron expressio.

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

The finest resolution for an EventBridge rule that uses a cron expression is one minute. Your scheduled rule runs within that minute but not on the precise 0th second.

Because EventBridge and target services are distributed, there can be a delay of several seconds between the time the scheduled rule runs and the time the target service performs the action on the target resource.

My rule matches AWS global service API calls but it didn't run

AWS global services; such as, IAM and Amazon Route 53 are only available in the US East (N. Virginia) Region, so events from AWS API calls from global services are only available in that region. For more information, see Events from AWS services (p. 35).
The IAM role associated with my rule is being ignored when the rule runs

EventBridge only uses IAM roles for rules (p. 82) that send events (p. 27) to Kinesis streams. For rules that invoke Lambda functions or Amazon SNS topics, you need to provide resource-based permissions (p. 405). Make sure your regional AWS STS endpoints are enabled, so that EventBridge can use them 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.

My rule has an event pattern that is supposed to match a resource, but no events match

Most services in AWS treat a colon (:) or slash (/) as the same character in Amazon Resource Names (ARNs), but EventBridge uses an exact match in event patterns (p. 62) and rules (p. 82). Be sure to use the correct ARN characters when creating event patterns so that they match the ARN syntax in the event (p. 27) to match.

Some events, such as AWS API call events from CloudTrail, don't have anything in the resources field.

My event's delivery to the target was delayed

EventBridge tries to deliver an event (p. 27) to a target (p. 112) 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. If the target service is having problems, EventBridge automatically reschedules another delivery. If 24 hours has passed since the arrival of event, EventBridge stops trying to deliver the event and publishes the FailedInvocations metric in CloudWatch. We recommend that you set up a DLQ to store events that couldn't successfully be delivered to a target. For more information, see Event retry policy and using dead-letter queues (p. 58)

Some events were never delivered to my target

If the target (p. 112) of an EventBridge rule (p. 82) is constrained for a prolonged time, EventBridge might not retry delivery. For example, if the target is not provisioned to handle the incoming event (p. 27) traffic and the target service is throttling requests that EventBridge makes on your behalf, then EventBridge might not retry delivery.

My rule ran more than once in response to one event

In rare cases, the same rule (p. 82) can run more than once for a single event (p. 27) or scheduled time, or the same target (p. 112) can be invoked more than once for a given triggered rule.
Preventing infinite loops

In EventBridge, it is possible to create a rule (p. 82) that leads to infinite loops, where the rule runs repeatedly. If you have a rule that causes an infinite loop, rewrite it so that the actions that the rule takes don't match the same rule.

For example, a rule that detects that ACLs have changed on an Amazon S3 bucket and then runs software to change them to a new state causes an infinite loop. One way to resolve it is to rewrite the rule so that it only matches ACLs that are in a bad state.

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.

My events are not delivered to the target Amazon SQS queue

If your Amazon SQS queue is encrypted, you must create a customer-managed KMS key and include the following permission section in your KMS key policy. For more information, see Configuring AWS KMS permissions.

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

My rule runs, but I don't see any messages published into my Amazon SNS topic

Scenario 1

You need permission for messages to be published into your Amazon SNS topic. Use the following command using the AWS CLI, replacing us-east-1 with your Region and using your topic ARN.

```
```

To have the correct permission, your policy attributes similar to the following.

```json
"\"Version\":\"2012-10-17\",
\"Id\":\"__default_policy_ID\",
\"Statement\":[(\"Sid\":\"__default_statement_ID\",
\"Effect\":\"Allow\",
\"Principal\":\"AWS\":\"*\"),
\"Action\":\"SNS:Subscribe",
```
Amazon EventBridge User Guide
My Amazon SNS topic still has permissions for EventBridge even after I deleted the rule associated with the Amazon SNS topic

```
"SNS:ListSubscriptionsByTopic",
"SNS:DeleteTopic",
"SNS:GetTopicAttributes",
"SNS:Publish",
"SNS:RemovePermission",
"SNS:AddPermission",
"SNS:SetTopicAttributes"],
  "Condition": {"StringEquals": {"AWS:SourceOwner": "123456789012"}}},
{"Sid": "Allow_Publish_Events",
  "Effect": "Allow",
  "Principal": {"Service": "events.amazonaws.com"},
  "Action": "sns:Publish",
```

If you don't see `events.amazonaws.com` with `Publish` permission in your policy, first 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",
```

Then set the topic attributes by using the AWS CLI, use the following command.

```
```

**Note**
If the policy is incorrect, you can also edit the rule (p. 82) in the EventBridge console by removing and then adding it back to the rule. EventBridge sets the correct permissions on the target (p. 112).

**Scenario 2**

If your SNS topic is encrypted, you must include the following section in your KMS key policy.

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

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

When you create a rule (p. 82) with Amazon SNS as the target (p. 112), EventBridge adds permission to your Amazon SNS topic on your behalf. If you delete the rule shortly after you create it, EventBridge
might not remove the permission from your Amazon SNS topic. If this happens, you can remove the permission from the topic by using the `aws sns set-topic-attributes` command. For information about resource-based permissions for sending events, see Using resource-based policies for Amazon EventBridge (p. 405).

### Which IAM condition keys can I use with EventBridge?

EventBridge supports the AWS-wide condition keys (see IAM and AWS STS condition context keys in the IAM User Guide), plus the keys listed at Using IAM policy conditions for fine-grained access control (p. 416).

### How can I tell when EventBridge rules are broken?

You can use the following alarm to notify you when your EventBridge rules (p. 82) are broken.

**To create an alarm to alert when rules are broken**

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 aren't 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.
Amazon EventBridge quotas

There are quotas for most aspects of EventBridge.

Topics
- EventBridge quotas (p. 447)
- PutPartnerEvents quotas by Region (p. 450)
- EventBridge Schema Registry quotas (p. 451)
- EventBridge Pipes quotas (p. 452)

Note
For a list of the quotas for EventBridge Scheduler, see Quotas for EventBridge Scheduler in the EventBridge Scheduler User Guide.

EventBridge quotas

EventBridge has the following quotas.

The Service Quotas console provides information about EventBridge quotas. Along with viewing the default quotas, you can use the Service Quotas console to request quota increases for adjustable quotas.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Adjust</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Api destinations</td>
<td>Each supported Region: 3,000</td>
<td>Yes</td>
<td>The maximum number of API destinations per account per Region.</td>
</tr>
<tr>
<td>Connections</td>
<td>Each supported Region: 3,000</td>
<td>Yes</td>
<td>The maximum number of connections per account per Region.</td>
</tr>
<tr>
<td>CreateEndpoint throttle limit in transactions per second</td>
<td>Each supported Region: 5 per second</td>
<td>No</td>
<td>The maximum number of requests per second for CreateEndpoint API. Additional requests are throttled.</td>
</tr>
<tr>
<td>DeleteEndpoint throttle limit in transactions per second</td>
<td>Each supported Region: 5 per second</td>
<td>No</td>
<td>The maximum number of requests per second for DeleteEndpoint API. Additional requests are throttled.</td>
</tr>
<tr>
<td>Endpoints</td>
<td>Each supported Region: 100</td>
<td>Yes</td>
<td>The maximum number of endpoints per account per Region.</td>
</tr>
<tr>
<td>Event bus policy size</td>
<td>Each supported Region: 10,240</td>
<td>Yes</td>
<td>Maximum policy size, in characters. This policy size increases each time you grant access to another account. You can</td>
</tr>
<tr>
<td>Name</td>
<td>Default</td>
<td>Adjust</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------------------------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Event buses</td>
<td>Each supported Region: 100</td>
<td>Yes</td>
<td>Maximum event buses per account.</td>
</tr>
<tr>
<td>Event pattern size</td>
<td>Each supported Region: 2,048</td>
<td>Yes</td>
<td>Maximum size of an event pattern, in characters.</td>
</tr>
</tbody>
</table>
| Invocations throttle limit in transactions per second | us-east-1: 18,750  
us-east-2: 4,500  
us-west-1: 2,250  
us-west-2: 18,750  
af-south-1: 750  
ap-northeast-1: 2,250  
ap-northeast-3: 750  
ap-southeast-1: 2,250  
ap-southeast-2: 2,250  
ap-southeast-3: 750  
eu-central-1: 4,500  
eu-south-1: 750  
eu-west-1: 18,750  
eu-west-2: 2,250  
Each of the other supported Regions: 1,100 | Yes    | An invocation is an event matching a rule and being sent on to the rules targets. After the limit is reached, the invocations are throttled; that is, they still happen but they are delayed. |
| Number of rules                         | af-south-1: 100  
eu-south-1: 100  
Each of the other supported Regions: 300 | Yes    | Maximum number of rules an account can have per event bus.                  |
<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Adjust</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PutEvents throttle limit in transactions per second</td>
<td>us-east-1: 10,000 us-east-2: 2,400 us-west-1: 1,200 us-west-2: 10,000 af-south-1: 400 ap-northeast-1: 1,200 ap-northeast-3: 400 ap-southeast-1: 1,200 ap-southeast-2: 1,200 ap-southeast-3: 400 eu-central-1: 2,400 eu-south-1: 400 eu-west-1: 10,000 eu-west-2: 1,200 Each of the other supported Regions: 600</td>
<td>Yes</td>
<td>Maximum number of requests per second for PutEvents API. Additional requests are throttled.</td>
</tr>
<tr>
<td>Rate of invocations per API destination</td>
<td>Each supported Region: 300</td>
<td>Yes</td>
<td>The maximum number of invocations per second to send to each API destination endpoint per account per Region. Once the quota is met, future invocations to that API endpoint are throttled. The invocations will still occur, but are delayed.</td>
</tr>
<tr>
<td>Targets per rule</td>
<td>Each supported Region: 5</td>
<td>No</td>
<td>Maximum number of targets that can be associated with a rule</td>
</tr>
<tr>
<td>Throttle limit in transactions per second</td>
<td>Each supported Region: 50</td>
<td>Yes</td>
<td>Maximum number of requests per second for all EventBridge API operations except PutEvents. Additional requests are throttled</td>
</tr>
</tbody>
</table>
In addition, EventBridge has the following quotas that are not managed through the Service Quotas console.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Adjust</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateEndpoint throttle limit in transactions per second</td>
<td>Each supported Region: 5 per second</td>
<td>No</td>
<td>The maximum number of requests per second for UpdateEndpoint API. Additional requests are throttled.</td>
</tr>
</tbody>
</table>

### Event buses

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event buses</td>
<td>Each supported Region: 100</td>
<td>Maximum event buses per account.</td>
</tr>
</tbody>
</table>

### Event bus policy size

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event bus policy size</td>
<td>Each supported Region: 10240</td>
<td>Maximum policy size, in characters. This policy size increases each time you grant access to another account. You can see your current policy and its size by using the DescribeEventBus API.</td>
</tr>
</tbody>
</table>

### Event pattern size

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event pattern size</td>
<td>Each supported Region: 2048</td>
<td>Maximum size of an event pattern, in characters. This is adjustable up to 4096 characters. If you have requirements for the higher maximum limit, contact support.</td>
</tr>
</tbody>
</table>

### Rules containing wildcards

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules containing wildcards</td>
<td>Each supported Region: 30 rules per event bus</td>
<td>Maximum number of rules, per event bus per account, that can contain event filters that include wildcards. This quota cannot be adjusted. For more information on using wildcards in event patterns, see ?? (p. 73).</td>
</tr>
</tbody>
</table>

### Schema discovery levels

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema discovery levels</td>
<td>Each supported Region: 255 levels</td>
<td>Maximum number of levels schema discovery will infer events that are nested. Any events past 255 levels are ignored.</td>
</tr>
</tbody>
</table>

## PutPartnerEvents quotas by Region

If you have requirements for higher maximum limits, contact support.

<table>
<thead>
<tr>
<th>Regions</th>
<th>Transactions per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>• AWS GovCloud (US-West)</td>
<td>PutPartnerEvents has a soft limit of 1,400 throughput requests per second and 3,600 burst requests per second by default in all Regions.</td>
</tr>
<tr>
<td>• AWS GovCloud (US-East)</td>
<td></td>
</tr>
<tr>
<td>• US East (N. Virginia)</td>
<td></td>
</tr>
<tr>
<td>• US East (Ohio)</td>
<td></td>
</tr>
<tr>
<td>• US West (N. California)</td>
<td></td>
</tr>
<tr>
<td>• US West (Oregon)</td>
<td></td>
</tr>
<tr>
<td>• Africa (Cape Town)</td>
<td></td>
</tr>
</tbody>
</table>
EventBridge Schema Registry quotas

EventBridge Schema Registry has the following quotas.

The Service Quotas console provides information about EventBridge quotas. Along with viewing the default quotas, you can use the Service Quotas console to request quota increases for adjustable quotas.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Adjust</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiscoveredSchemas</td>
<td>Each supported Region: 200</td>
<td><strong>Yes</strong></td>
<td>The maximum number of schemas for a discovered schema registry that you can create in the current region.</td>
</tr>
<tr>
<td>Discoverers</td>
<td>Each supported Region: 10</td>
<td><strong>Yes</strong></td>
<td>The maximum number of discoverers that you can create in the current region.</td>
</tr>
<tr>
<td>Registries</td>
<td>Each supported Region: 10</td>
<td><strong>Yes</strong></td>
<td>The maximum number of registries that you can create in the current region.</td>
</tr>
<tr>
<td>SchemaVersions</td>
<td>Each supported Region: 100</td>
<td><strong>Yes</strong></td>
<td>The maximum number of versions per schema that you can create in the current region.</td>
</tr>
<tr>
<td>Schemas</td>
<td>Each supported Region: 100</td>
<td><strong>Yes</strong></td>
<td>The maximum number of schemas per registry.</td>
</tr>
</tbody>
</table>
EventBridge Pipes quotas

EventBridge Pipes has the following quotas. If you have requirements for higher maximum limits, contact support.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Regions</th>
<th>Default limit</th>
</tr>
</thead>
</table>
| Concurrent pipe executions per account| • AWS GovCloud (US-West)  
• AWS GovCloud (US-East)  
• China (Ningxia)  
• China (Beijing)  
• Asia Pacific (Osaka)  
• Africa (Cape Town)  
• Europe (Milan)  
• US East (Ohio)  
• Europe (Frankfurt)  
• US West (N. California)  
• Europe (London)  
• Asia Pacific (Sydney)  
• Asia Pacific (Tokyo)  
• Asia Pacific (Singapore)  
• Canada (Central)  
• Europe (Paris)  
• Europe (Stockholm)  
• South America (São Paulo)  
• Asia Pacific (Seoul)  
• Asia Pacific (Mumbai)  
• Asia Pacific (Hong Kong)  
• Middle East (Bahrain)  
• China (Ningxia)  
• China (Beijing)  
• Asia Pacific (Osaka)  
• Africa (Cape Town)  
• Europe (Milan)  | 1000 |
Amazon EventBridge tags

A tag is a custom attribute label that you or AWS assigns to an AWS resource. In EventBridge, you can assign tags to rule (p. 82) and event buses (p. 9). Each resource can have a maximum of 50 tags.

You use tags to 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 an EventBridge rule that you assign to an EC2 instance.

A tag has two parts:

- A tag key, for example, CostCenter, Environment, or Project.  
  - Tag keys are case sensitive.  
  - The maximum tag key length is 128 Unicode characters in UTF-8.  
  - For each resource, each tag key must be unique.  
  - Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: `+ @ / -` (hyphen).  
  - 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 limit.
- An optional tag value field, for example, `111122223333` or `Production`.  
  - Each tag key can have only one value.  
  - Tag values are case sensitive.  
  - Omitting the tag value is the same as using an empty string.  
  - 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).

Tip
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 then use the same convention for all tags.

You can use the EventBridge console, the EventBridge API, or the AWS CLI to add, edit, or delete tags. For more information, 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
# Document History

The following table describes important changes in each release of the *Amazon EventBridge User Guide*, beginning in July 2019. For notification about updates to this documentation, you can subscribe to an RSS feed.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate AWS CloudFormation templates from event buses and rules.</td>
<td>You can now generate AWS CloudFormation templates from your existing Amazon EventBridge event buses and rules.</td>
<td>November 18, 2022</td>
</tr>
<tr>
<td>Launched EventBridge Pipes documentation.</td>
<td>You can now create pipes to connect sources to targets, with optional filtering and enrichment.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>Generate AWS CloudFormation templates from event buses and rules.</td>
<td>You can now generate AWS CloudFormation templates from your existing Amazon EventBridge event buses and rules.</td>
<td>November 18, 2022</td>
</tr>
<tr>
<td>Added the AmazonEventBridgePipesFullAccess policy.</td>
<td>Provides full access to Amazon EventBridge Pipes.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>Added the AmazonEventBridgePipesReadOnlyAccess policy.</td>
<td>Provides read-only access to Amazon EventBridge Pipes.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>Added the AmazonEventBridgePipesOperatorAccess policy.</td>
<td>Provides read-only and operator (that is, the ability to stop and start running Pipes) access to Amazon EventBridge Pipes.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>Updated the CloudWatchEventsFullAccess policy.</td>
<td>Updated to match AmazonEventBridgeFullAccess policy.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>Updated the CloudWatchEventsReadOnlyAccess policy.</td>
<td>Updated to match AmazonEventBridgeReadOnlyAccess policy.</td>
<td>December 1, 2022</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Release Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Updated content filtering in event patterns.</td>
<td>You can now use suffix, equals-ignore-case, and $or filtering options to create event patterns.</td>
<td>November 14, 2022</td>
</tr>
<tr>
<td></td>
<td>• Content filtering in Amazon EventBridge event patterns (p. 70)</td>
<td></td>
</tr>
<tr>
<td>Updated the AmazonEventBridgeFullAccess policy.</td>
<td>Added permissions necessary for using EventBridge, Schema Registry and EventBridge Scheduler.</td>
<td>November 10, 2022</td>
</tr>
<tr>
<td></td>
<td>• AmazonEventBridgeFullAccess policy (p. 394)</td>
<td></td>
</tr>
<tr>
<td>Updated the AmazonEventBridgeReadOnlyAccess policy.</td>
<td>You can now view EventBridge Schema Registry and EventBridge Scheduler information.</td>
<td>November 10, 2022</td>
</tr>
<tr>
<td></td>
<td>• AmazonEventBridgeReadOnlyAccess policy (p. 396)</td>
<td></td>
</tr>
<tr>
<td>Updated content filtering in event patterns.</td>
<td>You can now use suffix, equals-ignore-case, and $or filtering options to create event patterns.</td>
<td>November 14, 2022</td>
</tr>
<tr>
<td></td>
<td>• Content filtering in Amazon EventBridge event patterns (p. 70)</td>
<td></td>
</tr>
<tr>
<td>Updated the AmazonEventBridgeFullAccess policy.</td>
<td>Added permissions necessary for using EventBridge, Schema Registry and EventBridge Scheduler.</td>
<td>November 10, 2022</td>
</tr>
<tr>
<td></td>
<td>• AmazonEventBridgeFullAccess policy (p. 394)</td>
<td></td>
</tr>
<tr>
<td>Updated the AmazonEventBridgeReadOnlyAccess policy.</td>
<td>You can now view EventBridge Schema Registry and EventBridge Scheduler information.</td>
<td>November 10, 2022</td>
</tr>
<tr>
<td></td>
<td>• AmazonEventBridgeReadOnlyAccess policy (p. 396)</td>
<td></td>
</tr>
<tr>
<td>Updated the AmazonEventBridgeReadOnlyAccess policy.</td>
<td>You can now view endpoint information.</td>
<td>April 07, 2022</td>
</tr>
<tr>
<td></td>
<td>• AmazonEventBridgeReadOnlyAccess</td>
<td></td>
</tr>
<tr>
<td>Added support for global endpoints.</td>
<td>Amazon EventBridge now supports using global endpoints to help make your application regional-fault tolerant at no additional cost. To learn more, see the following:</td>
<td>April 07, 2022</td>
</tr>
<tr>
<td></td>
<td>• Making applications Regional-fault tolerant with global endpoints and event replication (p. 219)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CreateEndpoint</td>
<td></td>
</tr>
<tr>
<td>Added support for archives and event replays.</td>
<td>Amazon EventBridge now supports using archives to store events, and event replays to replay the events from an archive. To learn more, see the following:</td>
<td>November 05, 2020</td>
</tr>
<tr>
<td></td>
<td>• Archiving Amazon EventBridge events (p. 147).</td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Release Date</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| Added support for dead-letter queues and retry policy for targets. | Amazon EventBridge now supports using dead-letter queues and defining a retry policy for targets. To learn more, see the following:  
  • [Event retry policy and using dead-letter queues](p. 58).  
  • [PutTargets](PutTargets)                                                                 | October 12, 2020   |
| Added support for JSONSchema Draft4 format schemas. | Amazon EventBridge now supports schemas in JSONSchema Draft 4 format. You can also now export schemas using the EventBridge API. To learn more, see the following.  
  • [Amazon EventBridge schemas](p. 227)  
  • [Export](Export) in the EventBridge Schema Registry API Reference.                                                                 | September 28, 2020 |
| Resource-based policies for the EventBridge Schema Registry | The Amazon EventBridge Schema Registry now supports resource-based policies. For more information, see the following.  
  • [Resource-based policies for Amazon EventBridge schemas](p. 411)  
  • [Policy](Policy) in the EventBridge Schema Registry API Reference  
  • [RegistryPolicy Resource Type](RegistryPolicy Resource Type) in the AWS CloudFormation User Guide                                                                 | April 30, 2020     |
| Tags for Event Buses | This release allows you to create and manage tags for event buses. You can add tags when creating an event bus, and add or manage existing tags by calling the related API. For more information, see the following.  
  • [Amazon EventBridge tags](p. 453)  
  • [Tag-based policies](Tag-based policies)  
  • [TagResource](TagResource)  
  • [UntagResource](UntagResource)  
  • [ListTagsForResource](ListTagsForResource)                                                                 | February 24, 2020  |
<p>| Increased service quotas | Amazon EventBridge has increased quotas for invocations and for PutEvents. Quotas vary by region, and can be increased if necessary.                                                                                                                                   | February 11, 2020  |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
</table>
| Added a new topic on transforming target input, and added a link to Application Auto Scaling Events. | Improved documentation on the input transformer.  
- *Amazon EventBridge input transformation (p. 137)*  
- *Use Input Transformer to extract data from an event and input that data to the target*  
- *Tutorial: Use input transformer to customize what EventBridge passes to the event target (p. 252)*  
Added a link to Application Auto Scaling Events.  
- *Application Auto Scaling Events and EventBridge*  
- *Events from AWS services (p. 35)* | December 20, 2019 |
| Content-based filtering | | December 19, 2019 |
| Added links to Amazon Augmented AI event examples. | Added a link to the Amazon Augmented AI topic in the Amazon SageMaker Developer Guide that provides example events for Amazon Augmented AI. For more information, see the following.  
- *Use Events in Amazon Augmented AI*  
- *Events from AWS services (p. 35)* | December 13, 2019 |
| Added links to Amazon Chime event examples. | Added a link to the Amazon Chime topic that provides example events for that service. For more information, see the following.  
- *Automating Amazon Chime with EventBridge*  
- *Events from AWS services (p. 35)* | December 12, 2019 |
| Amazon EventBridge Schemas | You can now manage schemas and generate code bindings for events in Amazon EventBridge. For more information, see the following.  
- *Amazon EventBridge schemas (p. 227)*  
- *EventBridge Schemas API Reference*  
- *EventSchemas Resource Type Reference* in AWS CloudFormation | December 1, 2019 |
| AWS CloudFormation support for Event Buses | AWS CloudFormation now supports the EventBus resource. It also supports the EventBusName parameter in both the EventBusPolicy and Rule resources. For more information, see *Amazon EventBridge Resource Type Reference*. | October 7, 2019 |
| New service | Initial release of Amazon EventBridge. | July 11, 2019 |