# Table of Contents

What Is Amazon EventBridge? .......................................................................................................................... 1
   How it works .................................................................................................................................................. 1

Setup and prerequisites ..................................................................................................................................... 3
   Sign up for Amazon Web Services (AWS) .................................................................................................. 3
   Sign in to the Amazon EventBridge console ............................................................................................ 3
   Account credentials ................................................................................................................................... 3
   Set up the AWS Command Line Interface .................................................................................................. 4
   Regional Endpoints .................................................................................................................................. 4

Getting started ................................................................................................................................................ 5
   Create rule ................................................................................................................................................ 5

Event buses .................................................................................................................................................... 7
   Permissions for event buses ...................................................................................................................... 9
   Managing event bus permissions .............................................................................................................. 9
   Example policy: Send events to the default bus in a different account .................................................. 11
   Example policy: Send events to a custom bus in a different account .................................................... 11
   Example policy: Send events to the same account and restrict updates ................................................. 12
   Example policy: Send events only from a specific rule to the bus in a different Region ....................... 12
   Example policy: Send events only from a specific Region to a different Region ................................... 13
   Example policy: Deny sending events from specific Regions ................................................................. 14

Events ............................................................................................................................................................ 16
   Minimum information needed for a valid custom event ........................................................................... 18
   Event patterns ............................................................................................................................................ 19
   Create event patterns ............................................................................................................................... 20
   Example events and event patterns ......................................................................................................... 21
   Null values and empty strings .................................................................................................................. 24
   Arrays ....................................................................................................................................................... 25
   Content-based filtering ............................................................................................................................. 26

Adding events with PutEvents ..................................................................................................................... 30
   Handling failures with PutEvents ............................................................................................................. 31
   Sending events using the AWS CLI ............................................................................................................ 32
   Calculating PutEvent event entry sizes .................................................................................................... 33

Rules ............................................................................................................................................................... 34
   Creating a rule that runs when events are received .................................................................................. 35
   Creating a rule that runs on a schedule ..................................................................................................... 37
   Cron Expressions ....................................................................................................................................... 38
   Rate Expressions ....................................................................................................................................... 40
   Create rule ................................................................................................................................................ 41

Disabling or deleting a rule ............................................................................................................................ 43

Using AWS SAM templates .......................................................................................................................... 44
   Combined template .................................................................................................................................. 44
   Separated template .................................................................................................................................... 45

Targets ............................................................................................................................................................. 46
   Targets available in the EventBridge console ............................................................................................ 46
   Target parameters ...................................................................................................................................... 47
   Permissions ................................................................................................................................................. 48

Configure targets ........................................................................................................................................... 49
   API destinations ....................................................................................................................................... 50
   API Gateway ............................................................................................................................................. 61
   Cross-account events .............................................................................................................................. 62
   Cross-Region events ............................................................................................................................... 65
   Same account events .............................................................................................................................. 67

Transforming target input .............................................................................................................................. 70
   Predefined variables ............................................................................................................................... 70
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving events from a SaaS partner</td>
<td>107</td>
</tr>
<tr>
<td>Transforming input by using the EventBridge API</td>
<td>70</td>
</tr>
<tr>
<td>Transforming input by using AWS CloudFormation</td>
<td>72</td>
</tr>
<tr>
<td>Common Issues with Transforming Input</td>
<td>72</td>
</tr>
<tr>
<td>Archive and replay</td>
<td>73</td>
</tr>
<tr>
<td>Archiving events</td>
<td>74</td>
</tr>
<tr>
<td>Replaying archived events</td>
<td>76</td>
</tr>
<tr>
<td>Global endpoints</td>
<td>77</td>
</tr>
<tr>
<td>Recovery Time &amp; Recovery Point Objectives</td>
<td>77</td>
</tr>
<tr>
<td>Event replication</td>
<td>77</td>
</tr>
<tr>
<td>Replicated event payload</td>
<td>78</td>
</tr>
<tr>
<td>Create a global endpoint</td>
<td>78</td>
</tr>
<tr>
<td>To create a global endpoint by using the console</td>
<td>78</td>
</tr>
<tr>
<td>To create a global endpoint by using the API</td>
<td>79</td>
</tr>
<tr>
<td>To create a global endpoint by using AWS CloudFormation</td>
<td>79</td>
</tr>
<tr>
<td>Working with global endpoints by using an AWS SDK</td>
<td>79</td>
</tr>
<tr>
<td>Available Regions</td>
<td>80</td>
</tr>
<tr>
<td>Best practices</td>
<td>80</td>
</tr>
<tr>
<td>Enabling event replication</td>
<td>80</td>
</tr>
<tr>
<td>Preventing event throttling</td>
<td>81</td>
</tr>
<tr>
<td>Using subscriber metrics in Amazon Route 53 health checks</td>
<td>81</td>
</tr>
<tr>
<td>AWS CloudFormation template</td>
<td>81</td>
</tr>
<tr>
<td>AWS CloudFormation template for defining a Route 53 health check</td>
<td>81</td>
</tr>
<tr>
<td>CloudWatch alarm template properties</td>
<td>83</td>
</tr>
<tr>
<td>Route 53 health check template properties</td>
<td>84</td>
</tr>
<tr>
<td>Schemas</td>
<td>85</td>
</tr>
<tr>
<td>Finding a schema</td>
<td>86</td>
</tr>
<tr>
<td>Schema registries</td>
<td>87</td>
</tr>
<tr>
<td>Creating a schema</td>
<td>88</td>
</tr>
<tr>
<td>Create a schema by using a template</td>
<td>88</td>
</tr>
<tr>
<td>Edit a schema template directly in the console</td>
<td>89</td>
</tr>
<tr>
<td>Create a schema from the JSON of an event</td>
<td>90</td>
</tr>
<tr>
<td>Create a schema from events on an event bus</td>
<td>92</td>
</tr>
<tr>
<td>Code bindings</td>
<td>93</td>
</tr>
<tr>
<td>Debugging event delivery</td>
<td>94</td>
</tr>
<tr>
<td>Using dead-letter queues</td>
<td>95</td>
</tr>
<tr>
<td>Considerations for using a dead-letter queue</td>
<td>96</td>
</tr>
<tr>
<td>Granting permissions to the dead-letter queue</td>
<td>97</td>
</tr>
<tr>
<td>How to resend events from a dead-letter queue</td>
<td>97</td>
</tr>
<tr>
<td>Events from AWS services</td>
<td>98</td>
</tr>
<tr>
<td>Related AWS services</td>
<td>102</td>
</tr>
<tr>
<td>Interface VPC Endpoints</td>
<td>103</td>
</tr>
<tr>
<td>Availability</td>
<td>103</td>
</tr>
<tr>
<td>Creating a VPC Endpoint for EventBridge</td>
<td>104</td>
</tr>
<tr>
<td>AWS X-Ray</td>
<td>105</td>
</tr>
<tr>
<td>Receiving events from a SaaS partner</td>
<td>107</td>
</tr>
<tr>
<td>Supported SaaS partner integrations</td>
<td>107</td>
</tr>
<tr>
<td>Configuring EventBridge</td>
<td>109</td>
</tr>
<tr>
<td>Create a rule for SaaS partner events</td>
<td>110</td>
</tr>
<tr>
<td>Receiving events from Salesforce</td>
<td>112</td>
</tr>
<tr>
<td>Step 1: Configure Amazon AppFlow to use Salesforce as a partner event source</td>
<td>112</td>
</tr>
<tr>
<td>Step 2: Configure EventBridge to receive Salesforce events</td>
<td>113</td>
</tr>
<tr>
<td>Tutorials</td>
<td>115</td>
</tr>
<tr>
<td>Get started tutorials</td>
<td>116</td>
</tr>
<tr>
<td>Archive and replay events</td>
<td>117</td>
</tr>
<tr>
<td>Create a sample application</td>
<td>120</td>
</tr>
<tr>
<td>Download code bindings</td>
<td>124</td>
</tr>
</tbody>
</table>
Use input transformer .......................................................... 125
AWS tutorials ............................................................................ 129
Log Auto Scaling group states ............................................. 130
Log AWS API calls ................................................................. 133
Log Amazon EC2 instance states ........................................... 136
Log Amazon S3 object level operations ............................... 139
Send events to a Kinesis stream ............................................... 142
Schedule Automated Amazon EBS Snapshots ..................... 145
Send a notification when an S3 object is created .................... 147
Schedule AWS Lambda functions .......................................... 150
SaaS tutorials ........................................................................... 154
Create a connection to Datadog .............................................. 155
Create a connection to Salesforce .......................................... 158
Create a connection to Zendesk ............................................. 162
Working with AWS SDKs .......................................................... 165
Code examples .......................................................................... 166
Actions ..................................................................................... 166
Add a Lambda function target ................................................ 167
Create a scheduled rule ............................................................. 169
Delete a scheduled rule ............................................................. 172
Send events .................................................................................. 173
Scenarios .................................................................................. 177
Create and trigger a rule ............................................................ 177
Cross-service examples .............................................................. 191
Use scheduled events to invoke a Lambda function ............. 191
Security ...................................................................................... 193
Data protection ........................................................................ 194
Encryption at rest ................................................................. 194
Encryption in transit ............................................................... 194
Tag-based policies ................................................................. 195
IAM .......................................................................................... 196
Authentication ........................................................................ 196
Access control ........................................................................ 197
Managing access ........................................................................ 198
Using identity-based policies (IAM policies) .................. 202
Using resource-based policies ........................................... 212
Resource-based policies for EventBridge schemas .... 217
Permissions reference ................................................................. 220
IAM policy conditions ............................................................... 222
Log and monitor ....................................................................... 234
EventBridge information in CloudTrail ............................. 234
Example: EventBridge log file entries ........................................ 235
Compliance validation ................................................................. 236
Resilience .................................................................................. 237
Infrastructure security ................................................................. 238
Security and vulnerability analysis ........................................ 239
Monitoring ................................................................................ 240
EventBridge metrics ................................................................. 240
Dimensions for EventBridge metrics ......................................... 242
Troubleshooting ........................................................................ 243
My rule ran but my Lambda function wasn't invoked ........ 243
I just created or modified a rule, but it didn't match a test event ......................................................................................... 244
My rule didn't run at the time I specified in the ScheduleExpression ................................................................. 245
My rule didn't run at the time I expected .................................. 245
My rule matches AWS global service API calls but it didn't run ......................................................................................... 245
The IAM role associated with my rule is being ignored when the rule runs ................................................................. 246
My rule has an event pattern that is supposed to match a resource, but no events match ................................................................. 246
What Is Amazon EventBridge?

Amazon EventBridge is a serverless event bus service that you can use to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your applications, software as a service (SaaS) applications, and AWS services to targets such as AWS Lambda functions, HTTP invocation endpoints using API destinations, or event buses in other AWS accounts.

Note
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. New features added to EventBridge are not added to CloudWatch Events.

The following video explains more: What's the difference between CloudWatch Events and EventBridge?

How it works

EventBridge receives an event (p. 16), an indicator of a change in environment, and applies a rule (p. 34) to route the event to a target (p. 46). Rules match events to targets based on either the structure of the event, called an event pattern (p. 19), or on a schedule. For example, when an Amazon EC2 instance changes from pending to running, you can have a rule that sends the event to a Lambda function.

All events that come to EventBridge are associated with an event bus (p. 7). Rules are tied to a single event bus, so they can only be applied to events on that event bus. Your account has a default event bus.
bus which receives events from AWS services, and you can create custom event buses to send or receive events from a different account or Region.

When an AWS Partner wants to send events to an AWS customer account, they set up a partner event source (p. 107). Then the customer must associate an event bus with the partner event source.

EventBridge API destinations (p. 50) 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 REST 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.

To customize the text from an event before EventBridge passes it to a target, use the input transformer (p. 70) to edit the information before it goes to the target.

You can archive (p. 74), or save, events and then replay (p. 76) 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.

When you build serverless applications that use EventBridge, it can be helpful to know the event pattern of typical events without having to generate the event. The event patterns are described in schemas (p. 85), which are available for all events generated by AWS services on EventBridge. You can also create or upload custom schemas for events that don't come from AWS services. Once you have a schema for an event, you can download code bindings for popular programming languages.

To organize AWS resources or to track costs in EventBridge, you can assign a custom label, or tag (p. 256), to AWS resources. Using tag-based policies (p. 195), you can control what resources can and can't do within EventBridge.

In addition to tag-based policies, EventBridge supports identity-based (p. 202) and resource-based (p. 212) 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.
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 Amazon Web Services (AWS) (p. 3)
• Sign in to the Amazon EventBridge console (p. 3)
• Account credentials (p. 3)
• Set up the AWS Command Line Interface (AWS CLI) (p. 4)
• Regional Endpoints (p. 4)

Sign up for Amazon Web Services (AWS)

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

If you already have an account, you can skip this step. If you don't have an account, use the following procedure to create one.

To sign up for an 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.

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/.

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.
"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"
      }
    }
  }
]

For more information, see Authentication (p. 196).

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. 34) that route events (p. 16) to a target (p. 46). In this section, you create a basic rule. For tutorials about specific scenarios and specific targets, see Amazon EventBridge tutorials (p. 115).

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. 7) 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.
    - To use a custom event pattern, choose Custom pattern (JSON editor) and create your 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. 97).

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. 256).
20. Choose **Next**.
21. Review the details of the rule and choose **Create rule**.
Amazon EventBridge event buses

An event bus is a pipeline that receives events (p. 16). Rules (p. 34) associated with the event bus evaluate events as they arrive. Each rule checks whether an event matches the rule's criteria. You associate a rule with a specific event bus, so the rule only applies to events received by that event bus.

To manage permissions for an event bus, you can configure a resource-based policy (p. 212) for it. A resource-based policy specifies which events to allow, and which entities have permission to create or modify rules or targets for an event. For example, you can use a policy on an event bus to allow or deny events from sources such as a rule or an event bus in a different AWS account or AWS Region. By using policies, you can aggregate all events from your application or organization in a single account and Region.

You can configure up to 300 rules for each event bus. If you have more than 300 rules in your environment, you can create custom event buses in your account and then associate an additional 300 rules with each event bus. You can customize how events are received in your account by creating event buses with different permissions for different services.

The most common event buses are:

- The default event bus in each account receives events from AWS services.
- A custom event bus sends events to or receives events from a different account.
- A custom event bus sends events to or receives events from a different Region to aggregate events in a single location.
- A partner event bus receives events from a SaaS partner. For more information, see Receiving events from a SaaS partner with Amazon EventBridge (p. 107).
The following video describes what event buses are and explains some of the basics of them: What are event buses

The following video covers the different event buses and when to use them: The differences between event buses

**Topics**
- Permissions for Amazon EventBridge event buses (p. 9)
- Creating an Amazon EventBridge event bus (p. 15)
Permissions for Amazon EventBridge event buses

The default event bus (p. 7) in your AWS account only allows events (p. 16) from one account. You can grant additional permissions to an event bus by attaching a resource-based policy (p. 212) 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. 222) in the policy to grant permissions to an organization, apply tags (p. 256), 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. 34) 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 in 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. 62).

Topics
- Managing event bus permissions (p. 9)
- Example policy: Send events to the default bus in a different account (p. 11)
- Example policy: Send events to a custom bus in a different account (p. 11)
- Example policy: Send events to the same account and restrict updates (p. 12)
- Example policy: Send events only from a specific rule to the bus in a different Region (p. 12)
- Example policy: Send events only from a specific Region to a different Region (p. 13)
- Example policy: Deny sending events from specific Regions (p. 14)

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.

```json
### 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: ###
{
  "Version": "2012-10-17",
  "Statement": [
    ### To grant permissions for an account to use the PutEvents action, include the following, otherwise delete this section: ###
    {
      "Sid": "allow_account_to_put_events",
      "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": "allow_all_accounts_from_organization_to_put_events",
      "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": "allow_account_to_manage_rules_they_created",
      "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 use all API operations on the default event bus in the account 123456789012.

```json
{
  "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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WebStoreCrossAccountPublish",
      "Effect": "Allow",
      "Action": [ "events:PutEvents" ],
      "Principal": { "AWS": "111112222333" }
    }
  ]
}
```
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"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "events:creatorAccount": "${aws:PrincipalAccount}",
                    "events:source": "com.exampleCorp.webStore"
                }
            }
        }
    ]
}
```

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 a rule specified for the event bus in account 111122223333. The Condition statement restricts events to only events that match the rules with the specified rule ARN.
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.
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 11112223333, but not events that are generated in the US West (Oregon) Region.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1_allow_any_events_from_account_11112223333",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111112222333:root"
      },
      "Action": "events:PutEvents",
      "Resource": "arn:aws:events:us-east-1:123456789012:event-bus/CrossRegionBus"
    },
    {
      "Sid": "2_deny-all-cross-region-us-west-2-events",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": "events:PutEvents",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": [
            "arn:aws:events:us-west-2:*:*"
          ]
        }
      }
    }
  ]
}
```
Creating an Amazon EventBridge event bus

You can create a custom event bus (p. 7) to receive events (p. 16) 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. 212) 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.
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. 98) 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. 98).

**Topics**

- Minimum information needed for a valid custom event (p. 18)
- Amazon EventBridge event patterns (p. 19)
- Adding Amazon EventBridge events with PutEvents (p. 30)
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

The following fields appear in an event:

**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.
Minimum information needed for a valid custom event

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

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

Event patterns have the same structure as the events (p. 16) they match. Rules (p. 34) 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 lead to infinite loops, where a rule is fired repeatedly. For example, a rule might detect that ACLs have changed on an S3 bucket, and trigger software to change them to the desired state. If the rule is not written carefully, the subsequent change to the ACLs fires the rule again, creating an infinite loop.

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

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

The following video goes over the basics of event patterns: How to filter events

Topics
- Create event patterns (p. 20)
- Example events and event patterns (p. 21)
- Matching null values and empty strings in Amazon EventBridge event patterns (p. 24)
- Arrays in Amazon EventBridge event patterns (p. 25)
- Content filtering in Amazon EventBridge event patterns (p. 26)

The following event shows a simple AWS event from Amazon EC2.
Create 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.

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.

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.

Here's a summary of all the comparison operators available in EventBridge:

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Example</th>
<th>Rule syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>UserID is null</td>
<td>&quot;UserID&quot;: [ null ]</td>
</tr>
</tbody>
</table>
### Comparison

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Example</th>
<th>Rule syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</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>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>Or</td>
<td>PaymentType is &quot;Credit&quot; or &quot;Debit&quot;</td>
<td>&quot;PaymentType&quot;: [&quot;Credit&quot;, &quot;Debit&quot;]</td>
</tr>
<tr>
<td>Not</td>
<td>Weather is anything but &quot;Raining&quot;</td>
<td>&quot;Weather&quot;: [{ &quot;anything-but&quot;: [&quot;Raining&quot;] }]</td>
</tr>
<tr>
<td>Numeric (equals)</td>
<td>Price is 100</td>
<td>&quot;Price&quot;: [ { &quot;numeric&quot;: [&quot;=&quot;, 100 ] } ]</td>
</tr>
<tr>
<td>Numeric (range)</td>
<td>Price is more than 10, and less than or equal to 20</td>
<td>&quot;Price&quot;: [ { &quot;numeric&quot;: [&quot;&gt;&quot;, 10, &quot;&lt;=&quot; , 20 ] } ]</td>
</tr>
<tr>
<td>Exists</td>
<td>ProductName exists</td>
<td>&quot;ProductName&quot;: [{ &quot;exists&quot;: true } ]</td>
</tr>
<tr>
<td>Does not exist</td>
<td>ProductName does not exist</td>
<td>&quot;ProductName&quot;: [{ &quot;exists&quot;: false } ]</td>
</tr>
<tr>
<td>Begins with</td>
<td>Region is in the US</td>
<td>&quot;Region&quot;: [{&quot;prefix&quot;: &quot;us-&quot;}]</td>
</tr>
</tbody>
</table>

### Match 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"]
}
```

### 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.

```json
{
  "version": "0",
  "id": "3e3c153a-8339-4e30-8c35-687ebe53fe",
  "detail-type": "EC2 Instance Launch Successful",
  "source": "awsautoscaling",
  "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.

{
    "version": "0",
    "id": "0948ba87-d3b8-c6d4-f2da-732aexample",
    "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.

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

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

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

You can create an event pattern (p. 19) that matches a field in an event (p. 16) that has a null value or is an empty string. Consider the following example event.

```json
{
    "version": "0",
    "id": "3e3c153a-8339-4e30-8c35-687ebeff53fe",
    "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": [""],
        "responseElements": null
    }
}
```

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

```json
{
    "detail": {
        "eventVersion": [""],
        "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. 19) is an array containing one or more values. An event pattern matches the event (p. 16) 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 lead to infinite loops, where a rule is fired repeatedly. For example, a rule might detect that ACLs have changed on an S3 bucket, and trigger software to change them to the desired state. If the rule is not written carefully, the subsequent change to the ACLs fires the rule again, creating an infinite loop.

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

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

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. 19). 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 (p. 16) is within a specific numeric range, if the event comes from a specific IP address, or only if a specific field doesn't exist in the event JSON.

**Important**

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

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

**Filter types**

- Prefix matching (p. 26)
- Anything-but matching (p. 26)
- Numeric matching (p. 27)
- IP address matching (p. 27)
- Exists matching (p. 28)
- Complex example with multiple matching (p. 28)

**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".

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

**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.

```
{
  "detail": {
    "state": [ { "anything-but": "initializing" } ]
  }
}
```
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 -1.0e9 and +1.0e9 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, "<=" , 5 ] } ],
    "d-count": [ { "numeric": [ "<" , 10 ] } ],
    "x-limit": [ { "numeric": [ "=" , 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 10.0.0.0 and end with a number between 0 and 255.
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.

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

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": { 
    "state": [ "initializing", "running" ] 
  } 
}
```

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 
    } 
  } 
}
```

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.

```
{  
  "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 ] } ] 
  } 
}
```
Note
When building event patterns, if you include a key more than once the last reference will be the one used to evaluate events. For example, for the following pattern:

```json
{
    "detail": {
        "location": [ { "prefix": "us-" } ],
        "location": [ { "anything-but": "us-east" } ]
    }
}
```

only { "anything-but": "us-east" } will be taken into account when evaluating the location.
Adding Amazon EventBridge events with PutEvents

The PutEvents action sends multiple events (p. 16) 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 (p. 250). 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 (p. 31)
- Sending events using the AWS CLI (p. 32)
- Calculating Amazon EventBridge PutEvents event entry size (p. 33)

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()
    .withTime(new Date());
```
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"
    }
  ]
}
```

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")
    .withDetail("{"key1": "value1", "key2": "value2" }");
```

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.
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:

```bash
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.

```json
[
  {
    "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 (p. 16) 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 (p. 16).

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 putting the event in an Amazon S3 object and including a link to that object in the PutEvents entry.
Amazon EventBridge rules

A rule matches incoming events (p. 16) and sends them to targets (p. 46) 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. 19) 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. 196) 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. 35)
- Creating an Amazon EventBridge rule that runs on a schedule (p. 37)
- Disabling or deleting an Amazon EventBridge rule (p. 43)
- Using Amazon EventBridge and AWS Serverless Application Model templates (p. 44)
Creating Amazon EventBridge rules that react to events

To create a rule (p. 34) for events (p. 16), you specify an action to take when EventBridge receives an event that matches the event pattern (p. 19) in the rule. When an event matches, EventBridge sends the event to the specified target (p. 46) 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

To create a rule that reacts to 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 AWS services.
9. (Optional) For Sample events, choose the type of event.
10. For Event pattern, do one 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.
• To use a custom event pattern, choose Custom pattern (JSON editor) and create your 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. 97).

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. 256).
20. Choose Next.
21. Review the details of the rule and choose Create rule.
Creating an Amazon EventBridge rule that runs on a schedule

A rule (p. 34) can run in response to an event (p. 16) or at certain time intervals. For example, to periodically run an AWS Lambda function, you can create a rule to run on a schedule. You can create rules that run on a schedule by using cron or rate expressions. All scheduled events use UTC+0 time zone, and the minimum precision for a schedule is one minute. Your scheduled rule runs within that minute, but not on the precise 0th second.

EventBridge supports cron expressions and rate expressions. Rate expressions are simpler to define and cron expressions offer the 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.

**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

**Formats**
- Cron Expressions (p. 38)
- Rate Expressions (p. 40)
- Create rule (p. 41)
Cron Expressions

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

Syntax

```
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>
<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</td>
</tr>
</tbody>
</table>
Rate Expressions

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 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
```

Create rule

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

**Note**

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

**To create a rule that runs on a regular schedule**

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**, do one of the following:
   - To use a cron expression to define the schedule, 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.** and enter the cron expression.
   - To use a rate expression to define the schedule, choose **A schedule that runs at a regular rate, such as every 10 minutes.** and enter the rate expression.
9. Choose **Next**.
10. For **Target types**, choose **AWS service**.
11. 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.
12. The fields displayed vary depending on the service you choose. Enter information specific to this target type as needed.
13. 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.

14. (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. 97).

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

16. Choose Next.

17. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 256).

18. Choose Next.

19. Review the details of the rule and choose Create rule.
Disabling or deleting an Amazon EventBridge rule

To stop a rule (p. 34) from processing events (p. 16) 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/.
2. In the navigation pane, choose Rules.
   Under Event bus, select the event bus that is associated with the rule.
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.
Using Amazon EventBridge and AWS Serverless Application Model templates

You can build and test rules (p. 34) manually in the EventBridge console, which can help in the development process as you refine event patterns (p. 19). 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. 120).

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. 44)
- Separated template (p. 45)

Combined template

The first approach uses the Events property to configure the EventBridge rule. The following example code defines an event (p. 16) that invokes your Lambda function.

**Note**
This example automatically creates the rule on the default event bus (p. 7), 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:

   ```json
   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. 46). You can explicitly define multiple targets.

   ```json
   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.

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

A target is a resource or endpoint that EventBridge sends an event (p. 16) to when the event matches the event pattern defined for a rule (p. 34). The rule processes the event (p. 16) 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. 50)
- API Gateway (p. 61)
- Batch job queue
- CloudWatch log group
- CodeBuild project
- CodePipeline
- EC2 CreateSnapshot API call
- EC2 Image Builder
• EC2 RebootInstances API call
• EC2 StopInstances API call
• EC2 TerminateInstances API call
• ECS task
• Event bus in a different account or Region (p. 62)
• Event bus in the same account and Region (p. 67)
• Firehose delivery stream
• Glue workflow
• Incident Manager response plan
• Inspector assessment template
• Kinesis stream
• Lambda function
• Redshift cluster
• SageMaker Pipeline
• SNS topic
• SQS queue
• Step Functions state machine
• Systems Manager Automation
• Systems Manager OpsItem
• Systems Manager Run Command

Target parameters

When you configure targets, there are additional parameters you can specify for certain AWS services. These include the following:

• BatchParameters (AWS Batch jobs)
• EcsParameters (Amazon ECS tasks)
• HttpParameters (Amazon API Gateway and 3rd party ApiDestination endpoints)
• KinesisParameters (Amazon Kinesis streams)
• RedshiftDataParameters (Amazon Redshift Data API clusters)
• RunCommandParameters (Amazon EC2 Instance commands)
• SageMakerPipelineParameters (Amazon SageMaker Model Building Pipelines)
• SqsParameters (Amazon SQS queues)

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). These paths are replaced dynamically at runtime with data from the event payload itself at the specified path. The supported syntax for dynamic parameter JSON paths is the same as when transformoning input. For more information, see ??? (p. 70)

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

• EcsParameters
• HttpParameters (except HeaderParameters)
• 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. 212). 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 REST 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. 196).

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. 62).

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": "*"
}
```
Configure targets

Learn how to configure settings for EventBridge targets.

Targets:
- API destinations (p. 50)
- Amazon EventBridge targets for Amazon API Gateway (p. 61)
- Sending and receiving Amazon EventBridge events between AWS accounts (p. 62)
- Sending and receiving Amazon EventBridge events between AWS Regions (p. 65)
- Sending and receiving Amazon EventBridge events between event buses in the same account and Region (p. 67)
API destinations

Amazon EventBridge API destinations are HTTP endpoints that you can invoke as the target (p. 46) of a rule (p. 34), similar to how you invoke an AWS service or resource as a target. Using API destinations, you can route events (p. 16) between AWS services, integrated software as a service (SaaS) applications, and your applications outside of AWS by using REST 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. 19) 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 Transforming Amazon EventBridge target input (p. 70).

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. 95) 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. 51)
- Create an API destination (p. 52)
- Service-linked role for API destinations (p. 53)
- Headers included in requests to API destinations (p. 53)
- API destination error codes (p. 54)
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 (p. 203). The same permission is granted to the service-linked role (p. 53) 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. 54).

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. 51).

10. Choose **Create**.

After you create an API destination, you can select it as the target of a rule (p. 34). To use an API destination as a target, you must provide an IAM role with the correct permissions. For more information, see ?? (p. 205)

### 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",
            "secretsmanager:UpdateSecret",
            "secretsmanager:DescribeSecret",
            "secretsmanager:DeleteSecret",
            "secretsmanager:GetSecretValue",
            "secretsmanager:PutSecretValue"
         ],
         "Resource": "arn:aws:secretsmanager:*::*:secret:events!connection/*"
      }
   ]
}
```

For more information about service-linked roles, see Using service-linked roles in the IAM documentation.

### 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>Header key</td>
<td>Header value</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json; charset=utf-8</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 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. 95).

**API destination partners**

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

**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://webhooks.mongodb-realm.com/api/client/v2.0/app/Realm App ID/service/HTTP Service Name/incoming_webhook/Webhook Name

Supported authorization types

API Key

Additional authorization parameters required

None

MongoDB documentation

API Key
Authentication Providers

Commonly used API operations

POST https://webhooks.mongodb-realm.com/api/client/v2.0/app/\textit{Realm App ID}/service/\textit{HTTP Service Name}/incoming_webhook/\textit{Webhook Name}

Additional information

\textit{EventbridgeAtlas} demonstrates how you can use Realm Webhooks to perform CRUD operations on collections in MongoDB Atlas from EventBridge API endpoints.

New Relic

API destination invocation endpoint URL

For more information, see Our EU and US region data centers.

Events


EU– https://insights-collector.eu01.nr-data.net/v1/accounts/\textit{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

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
API destinations

Salesforce

**API destination invocation endpoint URL**

- **Subject**– https://myDomainName.my.salesforce.com/services/data/versionNumber/sobjects/ObjectEndpoint/*
- **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. 158).

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
- Product
- 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.

- To sign up and create a free Sumo Logic instance, see Start your free trial today.
- For more information about using Sumo Logic, see HTTP Logs and Metrics Source.

Commonly used API operations

POST https://endpoint4.collection.us2.sumologic.com/receiver/v1/http/
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 username and password authentication.

Amazon EventBridge targets for Amazon API Gateway

You can use Amazon API Gateway to create, publish, maintain, and monitor REST APIs. Amazon EventBridge supports sending events to an API Gateway REST endpoint. When you specify an API Gateway endpoint as a target (p. 46), each event (p. 16) 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.

You can use an API Gateway target for the following use cases:

- To invoke a customer-specified REST 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/*/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. 47).

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 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

Sending and receiving Amazon EventBridge events between AWS accounts

You can configure EventBridge to send and receive events (p. 16) between event buses (p. 7) 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 (p. 34) 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 (p. 65) 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 Organizations, 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](https://aws.amazon.com/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 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. 9).

**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. 16).

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. 19) 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.

To create a rule that sends events to a different AWS account using the 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 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 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.
   • To use a custom event pattern, choose Custom pattern (JSON editor) and create your event pattern.

11. Choose Next.

12. For Target types, choose EventBridge event bus and choose Event bus in another AWS account or Region.

13. Enter the ARN of the event bus to use as the target.

14. 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.

15. (Optional) For Additional settings, do the following:
   • 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. 97).

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

17. Choose Next.

18. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 256).

19. Choose Next.

20. Review the details of the rule and choose Create rule.

Sending and receiving Amazon EventBridge events between AWS Regions

You can configure EventBridge to send and receive events (p. 16) between AWS Regions. You can also allow or deny events from specific Regions, specific rules (p. 34) 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 NOT supported destination Regions:
   • AWS GovCloud (US-West)
   • AWS GovCloud (US-East)
   • China (Ningxia)
   • China (Beijing)
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

To create a rule that sends events to a different AWS Region using the 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 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.

    • To use a custom event pattern, choose Custom pattern (JSON editor) and create your event pattern.
11. Choose Next.
12. For Target types, choose EventBridge event bus and choose Event bus in another AWS account or Region.

13. Enter the ARN of the event bus to use as the target.

14. For Execution role, to have EventBridge create a new IAM role that has permissions to send events to the specified event bus, choose Create a new role for this specific resource.

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

   • 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. 97).

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

17. Choose Next.

18. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 256).

19. Choose Next.

20. Review the details of the rule and choose Create rule.

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. 16) between event buses (p. 7) 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. 212) 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. 34) 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. 9).

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.

• On the receiver event bus, edit the permissions to allow events to be passed from the other event bus.

• On the receiver event bus, 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.

To send events to another event bus, you create a rule with an event bus as a target.

To create a rule that sends events to a different event bus using the 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 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 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.
    • To use a custom event pattern, choose Custom pattern (JSON editor) and create your event pattern.
11. Choose Next.
12. For Target types, choose EventBridge event bus and choose Event bus in the same AWS account and Region.
13. For Event bus as a target, select an event bus from the drop-down list.
14. For Execution role, to have EventBridge create a new IAM role that has permissions to send events to the specified event bus, choose Create a new role for this specific resource.
15. (Optional) For Additional settings, do the following:
    • 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. 97).
16. (Optional) Choose Add another target to add another target for this rule.
17. Choose Next.
18. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags (p. 256).
19. Choose Next.
20. Review the details of the rule and choose Create rule.
Transforming Amazon EventBridge target input

You can customize the text from an event (p. 16) before EventBridge passes the event to the target (p. 46) of a rule (p. 34). 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. 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. 125).

In this topic:
- Predefined variables (p. 70)
- Input Transform Examples (p. 70)
- Transforming input by using the EventBridge API (p. 72)
- Transforming input by using AWS CloudFormation (p. 72)
- Common Issues with Transforming Input (p. 72)

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 — A copy of the original event.
- aws.events.event.ingestion-time — The time at which the event was received by EventBridge. This variable is generated by EventBridge and can't be overwritten.
- aws.events.event.json — The exact payload of an event as a string.

Input Transform Examples

The following is an example Amazon EC2 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"
    }
}
```

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:

```json
{
    "timestamp" : "$.time",
    "instance" : "$.detail.instance-id",
    "state" : "$.detail.state"
}
```

This defines two variables, `<instance>` and `<state>`. 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>
<tr>
<td></td>
<td></td>
<td>Note that this is the behavior in the EventBridge console. The AWS CLI escapes the slash characters and the result is &quot;instance &quot;i-0123456789&quot; is in RUNNING&quot;.</td>
</tr>
<tr>
<td>Simple JSON</td>
<td>{ &quot;instance&quot; : &lt;instance&gt;, &quot;state&quot;: &lt;state&gt; }</td>
<td>{ &quot;instance&quot; : &quot;i-0123456789&quot;, &quot;state&quot;: &quot;RUNNING&quot; }</td>
</tr>
<tr>
<td>JSON with strings and variables</td>
<td>{ &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 &lt;state&gt;&quot; }</td>
<td>{ &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; }</td>
</tr>
<tr>
<td>JSON with a mix of variables and static information</td>
<td>{ &quot;instance&quot; : &lt;instance&gt;, &quot;state&quot;: [ 9, &lt;state&gt;, true ], &quot;Transformed&quot; : &quot;Yes&quot; }</td>
<td>{ &quot;instance&quot; : &quot;i-0123456789&quot;, &quot;state&quot;: [ 9, &quot;RUNNING&quot;, true ], }</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.
- The JSON that is passed to the target is minified and escaped.
- EventBridge doesn't escape values extracted by Input Path, when populating the Input Template for a target.
Amazon EventBridge archive and replay

In EventBridge, you can create an archive of events (p. 16) 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. 74)
- Replaying archived Amazon EventBridge events (p. 76)
Archiving Amazon EventBridge events

When you create an archive in EventBridge, you can determine which events (p. 16) are sent to the archive by specifying an event pattern (p. 19). 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.

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:
    • 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 retrieve details for the archive. The value returned for `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 (p. 16) 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 (p. 7), or one or more rules (p. 34) 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.
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 (p. 16) are routed to an event bus (p. 7) 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 (p. 77). 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. 77)
- Event replication (p. 77)
- Create a global endpoint (p. 78)
- Working with global endpoints by using an AWS SDK (p. 79)
- Available Regions (p. 80)
- Best practices for working with Amazon EventBridge global endpoints (p. 80)
- AWS CloudFormation template for setting up the Route 53 health check (p. 81)

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:

```
{  
    "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/](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. 80)
- Preventing event throttling (p. 81)
- Using subscriber metrics in Amazon Route 53 health checks (p. 81)

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. 81)
- CloudWatch alarm template properties (p. 83)
- Route 53 health check template properties (p. 84)

**AWS CloudFormation template for defining a Route 53 health check**

Use the following template to define your Route 53 health check.

```yaml
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
```

81
Amazon EventBridge User Guide
AWS CloudFormation template for defining a Route 53 health check

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
CloudWatch alarm template properties

**Value:** `!Ref HealthCheckName`

**HealthCheckConfig:**
- **Type:** CLOUDWATCH_METRIC
- **AlarmIdentifier:**
  - **Name:** `!Ref HighLatencyAlarm`
  - **Region:** `!Ref AWS::Region`
- **InsufficientDataHealthStatus:** `!FindInMap [InsufficientDataMap, !Ref TreatMissingDataAs, HConfig]`

**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>
</table>
| AlarmDescription | The description of the alarm.  
Default: High Latency in Amazon EventBridge |
| MetricName   | 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.  
Default: IngestionToInvocationStartLatency |
| Namespace    | 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.  
Default: AWS/Events |
| Statistic    | The statistic for the metric associated with the alarm, other than percentile.  
Default: Average |
| Period       | 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.  
Default: 60 |
| EvaluationPeriods | 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 |
Metric | Description
--- | ---
specifies that number. If you are setting an "M out of N" alarm, this value is the N, and DatapointsToAlarm is the M. | Default: 5

Threshold | The value to compare with the specified statistic. | Default: 30,000

ComparisonOperator | The arithmetic operation to use when comparing the specified statistic and threshold. The specified statistic value is used as the first operand. | Default: GreaterThanThreshold

TreatMissingData | Sets how this alarm is to handle missing data points. | Valid values: breaching, notBreaching, ignore, and missing
| Default: breaching

### 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>
</tbody>
</table>

InsufficientDataHealthStatus | When 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. | Valid values:
- Healthy: Route 53 considers the health check to be healthy.
- Unhealthy: Route 53 considers the health check to be unhealthy.
- 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.
| Default: Unhealthy

**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. 16) 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. 88) or infer schemas (p. 92) directly from events on an event bus (p. 7). 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. 70) 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
• Finding an Amazon EventBridge schema (p. 86)
• Amazon EventBridge schema registries (p. 87)
• Creating an Amazon EventBridge schema (p. 88)
• Amazon EventBridge code bindings (p. 93)
Finding an Amazon EventBridge schema

EventBridge includes schemas (p. 85) 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.
   <result>
   The first page of available schemas is displayed.
   </result>
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.
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/.
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. 88) 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.

For information about using the EventBridge schema registry through AWS CloudFormation, see EventSchemas Resource Type Reference in AWS CloudFormation.
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. 16). You can also infer the schema from events on an event bus (p. 7). 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. 93) to help create applications for events with that schema.

Topics
- Create a schema by using a template (p. 88)
- Edit a schema template directly in the console (p. 89)
- Create a schema from the JSON of an event (p. 90)
- Create a schema from events on an event bus (p. 92)

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": {
```
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": {
```
"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": {},
        "startDate": {
          "type": "integer",
          "format": "int64"
        },
        "stateMachineArn": {
          "type": "string"


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. 85) to speed up development in Golang, Java, Python, and TypeScript. Code bindings are available for AWS service events, schemas you create (p. 88), and for schemas you generate (p. 92) based on events (p. 16) on an event bus (p. 7). 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.
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. 95)
Event retry policy and using dead-letter queues

Sometimes an event (p. 16) isn't successfully delivered to the target (p. 46) specified in a rule (p. 34). 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. For more information about EventBridge metrics, see Monitoring Amazon EventBridge (p. 240).

The following information is sent as part of the DLQ message metadata:

- The rule ARN
- The target ARN
- The error code
- The error message
- The retry condition that was exhausted
- The number of attempts
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. 212) 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. 97).
- 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. 97).

- 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. 97).
- 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. 215).

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.
Events from AWS services

Many AWS services generate events (p. 16) that EventBridge receives. When an AWS service in your account emits an event, it goes to your account’s default event bus.

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. Events from API actions that start with the keywords List, Get, or Describe aren’t processed by EventBridge, with the exception of 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.

**Important**

CloudTrail is extending event delivery through EventBridge to resource owners. Currently, CloudTrail supports both API callers and resource owners to receive events in their S3 buckets by creating trails, and delivers events to API callers through EventBridge. With this change, resource owners in addition to API callers will be able to 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.

The CloudTrail team began deploying this update on April 12, 2021 and expect to complete this update in all commercial regions early this month. As a result, some customers might experience an increase in the EventBridge events being delivered through EventBridge coinciding with this update. No customer action is required, but should you have any questions, please reach out to AWS Support.

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.

**Note**

Each AWS service that generates events sends them to EventBridge as either best effort or guaranteed delivery. 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. Guaranteed delivery means that all events from the service are successfully delivered to EventBridge.

This table includes a representation of the AWS services that send events to EventBridge, but it doesn't include every service.

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.
<table>
<thead>
<tr>
<th>Service</th>
<th>Delivery type</th>
</tr>
</thead>
<tbody>
<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>Amazon Athena</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Backup</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Batch</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Amazon Braket</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Certificate Manager Private Certificate Authority</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Chime</td>
<td>Best effort</td>
</tr>
<tr>
<td>Events from AWS CloudTrail</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS CodeArtifact</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS CodeBuild</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CodeCommit</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS CodeDeploy</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Config</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Connect</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Control Tower</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 DataSync</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon DevOps Guru</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Elastic Beanstalk</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic Block Store</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic Block Store volume modifications</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic Compute Cloud (Amazon EC2)</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon EC2 Auto Scaling</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon EC2 Fleets</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon EC2 Spot Instance Interruption</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Elastic Container Registry</td>
<td>Best effort</td>
</tr>
<tr>
<td>Service</td>
<td>Delivery type</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Amazon Elastic Container Service</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Elemental MediaConvert</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Elemental MediaLive</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Elemental MediaPackage</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Elemental MediaStore</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Amazon EMR</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon EventBridge scheduled rules (p. 37)</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Amazon GameLift</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Glue</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Glue DataBrew</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Ground Station</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Health</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Inspector</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Interactive Video Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS IoT Analytics</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS IoT Greengrass V1</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS IoT Greengrass V2</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Key Management Service CMK deletion</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Key Management Service CMK rotation</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Key Management Service imported key material expiration</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Location Service</td>
<td>Guaranteed</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 OpsWorks</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Proton</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon QLDB</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Amazon RDS</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Redshift</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Resource Access Manager</td>
<td>Best effort</td>
</tr>
<tr>
<td>Service</td>
<td>Delivery type</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Savings Plans</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Security Hub</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Signer</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Amazon Simple Storage Service (Amazon S3)</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Amazon Simple Workflow Service</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Step Functions</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Storage Gateway</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Support</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Systems Manager</td>
<td>Best effort</td>
</tr>
<tr>
<td>Tag changes on resources</td>
<td>Best effort</td>
</tr>
<tr>
<td>AWS Transit Gateway</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon Translate</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>AWS Trusted Advisor</td>
<td>Best effort</td>
</tr>
<tr>
<td>Amazon WorkSpaces</td>
<td>Best effort</td>
</tr>
</tbody>
</table>
Amazon EventBridge related services

Amazon EventBridge works with other AWS services to process events (p. 16) or invoke a resource as the target (p. 46) of a rule (p. 34). For more information about EventBridge integrations with other AWS services, see the following:

**Topics**
- Using Amazon EventBridge with Interface VPC Endpoints (p. 103)
- Amazon EventBridge integration with AWS X-Ray (p. 105)
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. 16) your VPC sends to EventBridge use that endpoint. EventBridge then sends those events to other AWS services based on the rules (p. 34) and targets (p. 46) 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 (Hong Kong)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Jakarta)
- Asia Pacific (Tokyo)
- Asia Pacific (Osaka)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- South America (São Paulo)
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.
Amazon EventBridge integration with AWS X-Ray

You can use AWS X-Ray to trace events (p. 16) that pass through EventBridge. EventBridge passes the original trace header to the target (p. 46) 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. 98), 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. 74), 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. 95), 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. 33).
The following video demonstrates the use X-Ray and EventBridge together: Using AWS X-Ray for tracing
Receiving events from a SaaS partner with Amazon EventBridge

To receive events (p. 16) from SaaS partner applications and services, you need a partner event source from the partner. Then you can create a partner event bus (p. 7) and match it to the partner event source.

Supported SaaS partner integrations

EventBridge supports the following SaaS partner integrations:

- Auth0
- Blitline
• BUIDLHub
• Buildkite
• Camunda
• CleverTap
• Datadog
• Epsagon
• Freshworks
• Genesys
• GS2
• Karte
• Kloudless
• Mackerel
• MongoDB
• New Relic
• OneLogin
• Opsgenie
• PagerDuty
• Payshield
• SailPoint
• Saviynt
• Segment
• Shopify
• SignalFx
• Site24x7
• Stax
• 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>
</tbody>
</table>
Configuring Amazon EventBridge to receive events from a SaaS integration

The following procedure describes how to configure EventBridge to receive events (p. 16) from a SaaS integration partner.

To receive events from an SaaS partner

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
Creating a rule that matches SaaS partner events

Before you can create rules (p. 34) for events (p. 16) from SaaS partner applications and services, you need a partner event bus (p. 7). Then you can match the partner event bus to the partner event source. For more information, see Receiving events from a SaaS partner with Amazon EventBridge (p. 107).

To create a rule that matches an event from a SaaS partner

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.
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:
   a. To create an IAM role automatically, choose Create a new role for this specific resource.
   b. 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:
      i. Choose None to not use a dead-letter queue.
      ii. 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.
      iii. 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. 97).
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. 256).
20. Choose **Next**.
21. Review the details of the rule and choose **Create rule**.
Receiving events from Salesforce

You can use Amazon EventBridge to receive events (p. 16) from Salesforce 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 (p. 107).

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:
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.
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 (p. 26).

**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.
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. 116)
- Amazon EventBridge tutorials for integrating with other AWS services (p. 129)
- Amazon EventBridge tutorials for integrating with SaaS providers (p. 154)
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. 117)
- Create an Amazon EventBridge sample application (p. 120)
- Tutorial: Download code bindings for events using the EventBridge schema registry (p. 124)
- Tutorial: Use input transformer to customize what EventBridge passes to the event target (p. 125)
Archive and replay Amazon EventBridge events

You can use EventBridge to route events (p. 16) to specific AWS Lambda functions using rules (p. 34).

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 (p. 74) and a rule that'll archive test events using the EventBridge console. Once there are events in that archive, you'll replay (p. 76) them.

Steps:
- Step 1: Create a Lambda function (p. 117)
- Step 2: Create archive (p. 117)
- Step 3: Create rule (p. 118)
- Step 4: Send test events (p. 118)
- Step 5: Replay events (p. 119)
- Step 6: Clean up your resources (p. 119)

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.
7. 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');
};
```

8. 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.
6. Choose Create archive.

**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/.
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.

**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/.
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/](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 08:00:00**.
   b. For **End time**, select the current date and time. For example, **2021/08/11 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 (p. 16) to specific Lambda functions using rules (p. 34).

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 (p. 46) of an EventBridge rule.
- And the rule that routes the created events to the correct downstream function based on an event pattern (p. 19).

This example uses AWS SAM templates to define the EventBridge rules. To learn more about using AWS SAM templates with EventBridge see ??? (p. 44).

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. 121)**
- **Step 1: Create application (p. 121)**
- **Step 2: Run application (p. 122)**
- **Step 3: Check the logs and verify the application works (p. 122)**
- **Step 4: Clean up your resources (p. 119)**

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`.

121
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. 93) for event schemas (p. 85) 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. 16) on an event bus (p. 7). 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. 70) in EventBridge to customize text from an event (p. 16) before you send it to the target of a rule (p. 34).

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 Transforming Amazon EventBridge target input (p. 70)

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. 125)
- Step 2: Create an Amazon SNS subscription (p. 125)
- Step 3: Create a rule (p. 126)
- Step 4: Send test events (p. 127)
- Step 5: Confirm success (p. 127)
- Step 6: Clean up your resources (p. 119)

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** 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.
Amazon EventBridge tutorials for integrating with other AWS services

Amazon EventBridge works with other AWS services to process events (p. 16) or invoke an AWS resource as the target (p. 46) of a rule (p. 34). 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. 130)
- Tutorial: Log AWS API calls using EventBridge (p. 133)
- Tutorial: Log the state of an Amazon EC2 instance using EventBridge (p. 136)
- Tutorial: Log Amazon S3 object-level operations using EventBridge (p. 139)
- Tutorial: Send events to an Amazon Kinesis stream using EventBridge (p. 142)
- Tutorial: Schedule automated Amazon EBS snapshots using EventBridge (p. 145)
- Tutorial: Send a notification when an Amazon S3 object is created (p. 147)
- Tutorial: Schedule AWS Lambda functions using EventBridge (p. 150)
Tutorial: Log the state of an Auto Scaling group using EventBridge

You can run an AWS Lambda function that logs an events 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 Getting CloudWatch Events When Your Auto Scaling Group Scales in the Amazon EC2 Auto Scaling User Guide.

In this tutorial, you create a Lambda function, and you create a rule in the EventBridge console that calls that function when an Amazon EC2 Auto Scaling group launches or terminates an instance.

Steps:
- Prerequisites (p. 130)
- Step 1: Create a Lambda function (p. 130)
- Step 2: Create a rule (p. 131)
- Step 3: Test the rule (p. 131)
- Step 4: Confirm success (p. 127)
- Step 5: Clean up your resources (p. 119)

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 an AWS Lambda function to log AWS API calls. For example, you can create a rule (p. 34) to log any operation in Amazon EC2, or you can limit this rule to log only a specific API call.

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. 133)
- Step 2: Create an AWS Lambda function (p. 133)
- Step 3: Create a rule (p. 134)
- Step 4: Test the rule (p. 134)
- Step 5: Confirm success (p. 127)
- Step 6: Clean up your resources (p. 119)

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.

```javascript
'use strict';
exports.handler = (event, context, callback) => {
    console.log('LogEC2StopInstance');
};
```
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. 34) 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. 136)
- Step 2: Create a rule (p. 136)
- Step 4: Test the rule (p. 134)
- Step 4: Confirm success (p. 127)
- Step 5: Clean up your resources (p. 119)

Step 1: Create an AWS Lambda function

Create a Lambda function to log the state change events (p. 16). 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`.
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 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. 16), 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. 34) in the EventBridge console that invokes that function in response to an S3 data event.

Steps:
• Step 1: Configure your AWS CloudTrail trail (p. 139)
• Step 2: Create an AWS Lambda function (p. 139)
• Step 3: Create a Rule (p. 140)
• Step 4: Test the Rule (p. 141)
• Step 5: Confirm success (p. 127)
• Step 6: Clean up your resources (p. 119)

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** 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

You can send AWS API call events (p. 16) 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. 34) in the EventBridge console that sends events to that stream when an Amazon EC2 instance stops.

Steps:
- Prerequisites (p. 142)
- Step 1: Create an Amazon Kinesis stream (p. 142)
- Step 2: Create a rule (p. 142)
- Step 3: Test the rule (p. 143)
- Step 4: Verify that the event was sent (p. 143)
- Step 5: Clean up your resources (p. 119)

Prerequisites

In this tutorial you 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.

Step 1: Create an Amazon Kinesis stream

To create a stream, at a command prompt, use the create-stream AWS CLI command.

```bash
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.

```bash
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/.
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 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 Kinesis stream from the drop-down list.
13. For Stream, select the Kinesis stream that you created in the Step 1: Create an Amazon Kinesis stream section. In this example, select test.
14. For Execution role, choose Create a new for role for this specific resource.
15. Choose Next.
16. Choose Next.
17. 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.

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

The following is example output.

```
{
  "ShardIterator": "AAAAAAAASywWjv9zEgPX4NyKdZ5wryMz9FA9e8NeKbUjp11xtZs1SpKEd916A9EYl0lR1EM1+9Mk/nHvtLxxyphEzYvKZ24D9dQV2/mBYWRO6OTZKrKw9gd+efGNoaHfedH1rJ14BL9Wyryk+ghYG22Z2T1daEBYNSH1+LAbM13gQwefJADBdyMwlo5r6PqcP2dzhg=
}
```

2. To get the record, use the following get-records command. Use the shard iterator from the output in the previous step.
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. 34) 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. 46), you must use the AWS Management Console.

Steps:
- Step 1: Create the rule (p. 145)
- Step 2: Test the rule (p. 145)
- Step 3: Confirm success (p. 127)
- Step 4: Clean up your resources (p. 119)

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. 37).

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.
   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.
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/.
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 (p. 34) in the EventBridge console that sends events (p. 16) to that topic when Amazon S3 Object Created events are received.

Steps:
- Prerequisites (p. 147)
- Step 1: Create an Amazon SNS topic (p. 147)
- Step 2: Create an Amazon SNS subscription (p. 147)
- Step 3: Create a rule (p. 148)
- Step 4: Test the rule (p. 148)
- Step 5: Clean up your resources (p. 119)

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/.
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 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, choose Amazon S3 Event Notification from the drop-down list.
   c. Choose Specific events(s) and choose Object Created from the drop-down list.
   d. Choose Any bucket
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 Kinesis stream that you created in the Step 1: Create an SNS topic section. In this example, select eventbridge-test.
15. Choose Next.
16. 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 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. 34) 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. 37). 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. 117)
- Step 2: Create a Rule (p. 150)
- Step 3: Verify the rule (p. 152)
- Step 4: Confirm success (p. 127)
- Step 5: Clean up your resources (p. 119)

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.

   ```bash
   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.

   ```json
   {
   "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.

   ```bash
   aws lambda add-permission \
   --function-name LogScheduledEvent \
   ```
Amazon EventBridge User Guide
Schedule AWS Lambda functions

3. 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
   ```

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

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

**Step 3: Verify the rule**

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. 16). The following tutorials show you how to integrate EventBridge with SaaS partners.

**Tutorials:**
- Tutorial: Create a connection to Datadog as an API destination (p. 155)
- Tutorial: Create a connection to Salesforce as an API destination (p. 158)
- Tutorial: Create a connection to Zendesk as an API destination (p. 162)
Tutorial: Create a connection to Datadog as an API destination

You can use EventBridge to route events (p. 16) 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 (p. 50) that points to Datadog, and a rule (p. 34) to route events to Datadog.

Steps:
- Prerequisites (p. 155)
- Step 1: Create connection (p. 155)
- Step 2: Create API destination (p. 155)
- Step 3: Create rule (p. 156)
- Step 4: Test the rule (p. 157)
- Step 5: Clean up your resources (p. 119)

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 (p. 46) 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:
  
  ```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 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. 16) 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. 50) that points to Salesforce, and a rule (p. 34) to route events to Salesforce.

Steps:
- Prerequisites (p. 158)
- Step 1: Create connection (p. 158)
- Step 2: Create API destination (p. 155)
- Step 3: Create rule (p. 156)
- Step 4: Test the rule (p. 157)
- Step 5: Clean up your resources (p. 119)

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.
Create a connection to Salesforce

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. 46) 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 **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/.
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:

```
{
  "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:

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

   d. For **Target input transformer** do the following:
      i. For **Input Path**, enter the following:

```
{"detail":"$.detail"}
```

      ii. For **Input Template**, enter the following:

```
{"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 to third-party services like Zendesk.

In this tutorial, you'll use the EventBridge console to create a connection to Zendesk, an API destination that points to Zendesk, and a rule to route events to Zendesk.

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 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 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/](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:

   ```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 **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": "$.*"}
      ```
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. When the event matches the rule, EventBridge will call the Zendesk Create Ticket API. 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://example.com) 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://example.com) 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](https://example.com) of the EventBridge console.
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 .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>
</tbody>
</table>

For examples specific to EventBridge, see Code examples for EventBridge using AWS SDKs (p. 166).

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).

The examples are divided into the following categories:

**Actions**

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

**Scenarios**

Code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

**Cross-service examples**

Sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using EventBridge with an AWS SDK (p. 165). This topic also includes information about getting started and details about previous SDK versions.

**Code examples**

- Actions for EventBridge using AWS SDKs (p. 166)
  - Add a Lambda function target using an AWS SDK (p. 167)
  - Create an EventBridge scheduled rule using an AWS SDK (p. 169)
  - Delete an EventBridge scheduled rule using an AWS SDK (p. 172)
  - Send EventBridge events using an AWS SDK (p. 173)
- Scenarios for EventBridge using AWS SDKs (p. 177)
  - Create and trigger a rule in Amazon EventBridge using an AWS SDK (p. 177)
- Cross-service examples for EventBridge using AWS SDKs (p. 191)
  - Use scheduled events to invoke a Lambda function (p. 191)

**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 not intended to be run in isolation. Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

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

**Examples**

- Add a Lambda function target using an AWS SDK (p. 167)
Add a Lambda function target using an AWS SDK

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

C++

SDK for C++

Include the required files.

```cpp
#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.

```cpp
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;
}
```

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

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.
import { EventBridgeClient } from "@aws-sdk/client-eventbridge";
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon EventBridge service client object.
export const ebClient = new EventBridgeClient({ region: REGION });

import the SDK and client modules and call the API.

// Import required AWS SDK clients and commands for Node.js.
import { PutTargetsCommand } from "@aws-sdk/client-eventbridge";
import { ebClient } from "./libs/eventBridgeClient.js";

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

export const run = async () => {
  try {
    const data = await ebClient.send(new PutTargetsCommand(params));
    console.log("Success, target added; requestID: ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
}

// Uncomment this line to run execution within this file.
// run();

• Find instructions and more code on GitHub.
• For API details, see PutTargets in AWS SDK for JavaScript API Reference.

SDK for JavaScript V2

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

// Create CloudWatchEvents service object
var 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);
    }
});

- Find instructions and more code on [GitHub](https://github.com).
- For API details, see [PutTargets](https://aws.amazon.com/documents) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using EventBridge with an AWS SDK](https://aws.amazon.com/documentation/). This topic also includes information about getting started and details about previous SDK versions.

### Create an EventBridge scheduled rule using an AWS SDK

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

#### C++

**SDK for C++**

Include the required files.

```cpp
#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.

```cpp
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;
}
```
Create a scheduled rule

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

Java

SDK for Java 2.x

```java
public static void createEBRule(EventBridgeClient eventBrClient, String ruleName) {
    try {
        PutRuleRequest ruleRequest = PutRuleRequest.builder()
            .name(ruleName)
            .eventBusName("default")
            .eventPattern("{"source": ["aws.s3"], "detail-type": ["AWS API Call via CloudTrail"],
            "detail": {"eventSource": ["s3.amazonaws.com"],
            "eventName": ["DeleteBucket"]})")
            .description("A test rule 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);
    }
}
```

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

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```javascript
import { EventBridgeClient } from "@aws-sdk/client-eventbridge";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon EventBridge service client object.
export const ebClient = new EventBridgeClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```javascript
// Import required AWS SDK clients and commands for Node.js.
import { PutRuleCommand } from "@aws-sdk/client-eventbridge";
import { ebClient } from "./libs/eventBridgeClient.js";

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

export const run = async () => {
  try {
    const data = await ebClient.send(new PutRuleCommand(params));
    console.log("Success, scheduled rule created; Rule ARN:", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();

• Find instructions and more code on GitHub.
• For API details, see PutRule in AWS SDK for JavaScript API Reference.

SDK for JavaScript V2

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

// Create CloudWatchEvents service object
var 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);
  }
});

• Find instructions and more code on GitHub.
• 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.

suspend fun createEBRule(ruleNameVal: String) {
Delete a scheduled rule

```kotlin
val request = PutRuleRequest {
    name = ruleNameVal
    eventBusName = "default"
    eventPattern = "{"source":["aws.s3"],"detail-type":["AWS API Call via CloudTrail"],"detail":{"eventSource":["s3.amazonaws.com"],"eventName":"["DeleteBucket"]"},
        description = "A test rule created by the AWS SDK for Kotlin"
}

EventBridgeClient { region = "us-west-2" }.use { eventBrClient ->
    val ruleResponse = eventBrClient.putRule(request)
    println("The ARN of the new rule is ${ruleResponse.ruleArn}"
}
}
```

- Find instructions and more code on GitHub.
- For API details, see PutRule 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. 165). This topic also includes information about getting started and details about previous SDK versions.

Delete an EventBridge scheduled rule using an AWS SDK

The following code examples show how to delete an Amazon EventBridge scheduled rule.

Java

```java
public static void deleteEBRule(EventBridgeClient eventBrClient, String ruleName) {
    try {
        DisableRuleRequest disableRuleRequest = DisableRuleRequest.builder()
            .name(ruleName)
            .eventBusName("default")
            .build();

        eventBrClient.disableRule(disableRuleRequest);
        DeleteRuleRequest ruleRequest = DeleteRuleRequest.builder()
            .name(ruleName)
            .eventBusName("default")
            .build();

        eventBrClient.deleteRule(ruleRequest);
        System.out.println("Rule " + ruleName + " was successfully deleted!");
    } catch (EventBridgeException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

SDK for Java 2.x
Send events

- Find instructions and more code on GitHub.
- For API details, see DeleteRule 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.

```kotlin
suspend fun deleteEBRule(ruleName: String) {
    val request = DisableRuleRequest {
        name = ruleName
        eventBusName = "default"
    }
    EventBridgeClient { region = "us-west-2" }.use { eventBrClient ->
        eventBrClient.disableRule(request)
        val ruleRequest = DeleteRuleRequest {
            name = ruleName
            eventBusName = "default"
        }
        eventBrClient.deleteRule(ruleRequest)
        println("Rule $ruleName was successfully deleted!")
    }
}
```

- Find instructions and more code on GitHub.
- For API details, see DeleteRule 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. 165). This topic also includes information about getting started and details about previous SDK versions.

Send EventBridge events using an AWS SDK

The following code examples show how to send Amazon EventBridge events.

C++

SDK for C++

Include the required files.

```c++
#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.
### Send events

```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;
}
```

- Find instructions and more code on [GitHub](https://github.com).

### Java

**SDK for Java 2.x**

```java
public static void putEBEvents(EventBridgeClient eventBrClient, String resourceArn, String resourceArn2) {
    try {
        // Populate a List with the resource ARN values.
        List<String> resources = new ArrayList<>();
        resources.add(resourceArn);
        resources.add(resourceArn2);

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

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

        PutEventsResponse result = eventBrClient.putEvents(eventsRequest);
        for (PutEventsResultEntry resultEntry : result.entries()) {
            if (resultEntry.eventId() != null) {
                System.out.println("Event Id: " + resultEntry.eventId());
            } else {
                System.out.println("Injection failed with Error Code: " + resultEntry.errorCode());
            }
        }
    } catch (EventBridgeException e) {
```
System.err.println(e.awsErrorDetails().errorMessage());  
System.exit(1);  
}
}

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

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```javascript
import { EventBridgeClient } from '@aws-sdk/client-eventbridge';
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon EventBridge service client object.
export const ebClient = new EventBridgeClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```javascript
// Import required AWS SDK clients and commands for Node.js.
import { PutEventsCommand } from '@aws-sdk/client-eventbridge';
import { ebClient } from './libs/eventBridgeClient.js';

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

export const run = async () => {
    try {
        const data = await ebClient.send(new PutEventsCommand(params));
        console.log("Success, event sent; requestID:", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
    // Uncomment this line to run execution within this file.
    // run();
```
SDK for JavaScript V2

```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 = {
    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);
    }
});
```

- Find instructions and more code on GitHub.
- For API details, see PutEvents in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

```kotlin
suspend fun putEBEvents(resourceArn: String, resourceArn2: String) {

    // Populate a List with the resource ARN values.
    val resourcesOb = mutableListOf<String>()
    resourcesOb.add(resourceArn)
    resourcesOb.add(resourceArn2)

    val reqEntry = PutEventsRequestEntry {
        resources = resourcesOb
        source = "com.mycompany.myapp"
        detailType = "myDetailType"
        detail = "\"key1\": \"value1\", \"key2\": \"value2\""
    }

    val request = PutEventsRequest {
        entries = listOf(reqEntry)
    }
}
```

Note
This is prerelease documentation for a feature in preview release. It is subject to change.
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. 177)**

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**

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.
# 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-1:11111111111: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.

# 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?(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
#   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.

```ruby
# 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.

```ruby
# Checks whether the specified AWS Identity and Access Management (IAM) role exists among those provided to this 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::11111111111: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.

```ruby
# 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
    while response.next_page? do
      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
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': {
```

180
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_arn)
  rules.each do |rule|
    return true if rule.name == rule_arn
  end
  false
end
```

Checks whether the specified EventBridge rule exists among those provided to this function.
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?(cloudwatchevents_client, '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
  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
  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',
#   'Triggers when any available EC2 instance starts.',
#   '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
)
  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    
  end
  return true
else    
  return false
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|
      if log_group.log_group_name == log_group_name
        puts 'Log group found.'
        return true
      end
    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?( # 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.
# 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.

---

```ruby
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,
    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
  end
  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
```

---

Restart an Amazon Elastic Compute Cloud (Amazon EC2) instance and adds information about the related activity to a log stream in CloudWatch Logs.

---

# Restarts an Amazon EC2 instance
# and adds information about the related activity to a log stream
Create and trigger a rule

# 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.

@example
exit 1 unless instance_restarted?(  
  Aws::EC2::Client.new(region: 'us-east-1'),  
  Aws::CloudWatchLogs::Client.new(region: 'us-east-1'),  
  'i-033c48ef067af3dEX',  
  'aws-doc-sdk-examples-cloudwatch-log'  
)
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}'. "  
  puts 'This might take a few minutes...'
  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.",  
    sequence_token  
  )  
  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.",  
    sequence_token  
  )  
  return true
rescue StandardError => e  
  puts 'Error creating log stream or stopping or restarting the instance: '  
  puts "#{e.message}"  
  log_event(  
    ec2_client,  
    cloudwatchlogs_client,  
    instance_id,  
    log_group_name  
)
```ruby
cloudwatchlogs_client,
log_group_name,
log_stream_name,
"Error stopping or starting instance '#{instance_id}': #{e.message}",
sequence_token
)
return false
end

Display information about activity for a rule in EventBridge.

# 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
#   display_rule_activity(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'aws-doc-sdk-examples-ec2-state-change',
#     Time.now - 600, # Start checking from 10 minutes ago.
#     Time.now, # Check up until now.
#     60 # Check every minute during those 10 minutes.
#   )
#
def display_rule_activity(
  cloudwatch_client,
  rule_name,
  start_time,
  end_time,
  period
)
  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 "
```
Create and trigger a rule

'specified time period.'

rescue StandardError => e
  puts "Error getting information about event rule activity: #{e.message}"
end

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
      )
      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
  else
    puts 'No log messages for this log stream.'
  end
rescue StandardError => e
  puts 'Error getting information about the log streams or their messages: ' \
    '#{e.message}"
end
```

Display a reminder to the caller to manually clean up any associated AWS resources that they no longer need.

```ruby
# Displays a reminder to the caller to manually clean up any associated
# AWS resources that they no longer need.
```
# Create and trigger a rule

```ruby
# @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
    manual_cleanup_notice(
        'aws-doc-sdk-examples-topic',
        'aws-doc-sdk-examples-cloudwatch-events-rule-role',
        'aws-doc-sdk-examples-ec2-state-change',
        'aws-doc-sdk-examples-cloudwatch-log',
        'i-033c48ef067af3dEX'
    )
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 '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:
```ruby
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'
    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
    period = 60 # Look back every 60 seconds over the past 10 minutes.
    end_time = Time.now
    # 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)
    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_cleanup_notice( 
      topic_name, role_name, rule_name, log_group_name, instance_id
    )
    exit 1
  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_cleanup_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_cleanup_notice( 
      topic_name, role_name, rule_name, log_group_name, instance_id
    )
  end
end

# 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( 
  cloudwatchlogs_client, 
  instance_id, 
  log_group_name
)
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. 191)

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.
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.

This example is also available in the AWS SDK for JavaScript v3 developer guide.

**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.

- 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. 165). 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. 194)
- Tag-based policies (p. 195)
- Amazon EventBridge and AWS Identity and Access Management (p. 196)
- Log and monitor in Amazon EventBridge (p. 234)
- Compliance validation in Amazon EventBridge (p. 236)
- Amazon EventBridge resilience (p. 237)
- Infrastructure security in Amazon EventBridge (p. 238)
- Configuration and vulnerability analysis in Amazon EventBridge (p. 239)
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 user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers’ email addresses, into tags or free-form fields such as a Name field. This includes when you work with EventBridge or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

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:CreateEventBus",
                "events:DescribeEventBus",
                "events:DeleteEventBus"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:TagKey/environment": "production"}
            }
        }
    ]
}
```

For more information about tagging, see the following.

- Amazon EventBridge tags (p. 256)
- 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. 196)
- Access control (p. 197)
- Managing access permissions to your Amazon EventBridge resources (p. 198)
- Using identity-based policies (IAM policies) for Amazon EventBridge (p. 202)
- Using resource-based policies for Amazon EventBridge (p. 212)
- Resource-based policies for Amazon EventBridge schemas (p. 217)
- Amazon EventBridge permissions reference (p. 220)
- Using IAM policy conditions for fine-grained access control (p. 222)

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 IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the IAM User Guide.

- **IAM user** – An IAM user is 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 user name and password to sign in to secure AWS webpages such as the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically 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 an IAM 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 provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.
• **Cross-account access** – You can use an IAM role in your account to grant another account permission to access your account’s resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the IAM User Guide.

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

• **Applications running on Amazon EC2** – For Amazon EC2 applications that need access to EventBridge, you can either store access keys in the EC2 instance or you can use an IAM role to manage temporary credentials. To assign an AWS role to an EC2 instance, you create an instance profile that is attached to the instance. An instance profile contains the role, and it provides temporary credentials to applications running on the EC2 instance. For more information, see Using Roles for Applications on Amazon EC2 in the IAM User Guide.

**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. 34) or events (p. 16) by using identity-based (p. 202) or resource-based (p. 212) 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. 220).

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. 19) 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/rewrite-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.

```json
"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 an IAM 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 IAM 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. 199)
- Resource-based policies (IAM policies) (p. 200)

## 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 IAM 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. 198).

Resource-based policies (IAM policies)

When a rule runs in EventBridge, all of the targets (p. 46) 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. 212).

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. 198) and Amazon EventBridge permissions reference (p. 220).

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. 198).
- **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. 220).

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. 222).
Using identity-based policies (IAM policies) for Amazon EventBridge

Identity-based policies are permissions policies that you attach to IAM identities.

Topics
- Permissions required to use EventBridge (p. 202)
- AWS managed policies for EventBridge (p. 203)
- Permissions required for EventBridge to access targets using IAM roles (p. 205)
- Customer-managed policy examples (p. 207)
- Amazon EventBridge updates to AWS managed policies (p. 210)

Permissions required to use EventBridge

For a user to work with the EventBridge console or API, that user must have a minimum set of permissions to access other AWS resources. Other AWS services can send events (p. 16) to EventBridge or be a target (p. 46) of an EventBridge rule (p. 34). The following list shows examples of AWS services and their corresponding minimum permissions:

- Automation
  - automation:CreateAction
  - automation:DescribeAction
  - automation:UpdateAction
- Amazon EC2 Auto Scaling
  - autoscaling:DescribeAutoScalingGroups
- AWS CloudTrail
  - cloudtrail:DescribeTrails
- Amazon EC2
  - ec2:DescribeInstances
  - ec2:DescribeVolumes
- EventBridge
  - events:DeleteRule
  - events:DescribeRule
  - events:DisableRule
  - events:EnableRule
  - events:ListRuleNamesByTarget
  - events:ListRules
  - events:ListTargetsByRule
  - events:PutEvents
  - events:PutRule
  - events:PutTargets
  - events:RemoveTargets
  - events:TestEventPattern
- IAM
  - iam:ListRoles
- Kinesis
• kinesis:ListStreams
• Lambda
  • lambda:AddPermission
  • lambda:ListFunctions
  • lambda:RemovePermission
• Amazon SNS
  • sns:GetTopicAttributes
  • sns:ListTopics
  • sns:SetTopicAttributes
• Amazon SWF
  • swf:DescribeAction
  • swf:ReferenceAction
  • swf:RegisterAction
  • swf:RegisterDomain
  • swf:UpdateAction

If you create an IAM policy that is more restrictive than the minimum required permissions, the EventBridge console won't function as intended for users with that IAM policy. To ensure that those users can still use the EventBridge console, also attach the AmazonEventBridgeReadOnlyAccess managed policy to the user, as described in AWS managed policies for EventBridge (p. 203).

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

**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. 203)** – Grants full access to EventBridge.
• **AmazonEventBridgeReadOnlyAccess (p. 204)** – Grants read-only access to EventBridge.

**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.
Using identity-based policies (IAM policies)

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "events:*",
    "Resource": "*"
  }, {
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/AmazonEventBridgeApiDestinationsServiceRolePolicy",
    "Condition": {
      "StringEquals": {
        "iam:AWSServiceName": "apidestinations.events.amazonaws.com"
      }
    }
  }, {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:CreateSecret",
      "secretsmanager:UpdateSecret",
      "secretsmanager:DeleteSecret",
      "secretsmanager:GetSecretValue",
      "secretsmanager:PutSecretValue"
    ],
    "Resource": "arn:aws:secretsmanager::*:*:secret:events:*"
  }, {
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::*:role/**",
    "Condition": {
      "StringLike": {
        "iam:PassedToService": "events.amazonaws.com"
      }
    }
  }]
}
```

**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:DescribeEventBus",
        "events:DescribeEventSource",
        "events:ListEventBuses",
        "events:ListEventSources",
        "events:ListRuleNamesByTarget",
        "events:ListRules",
        "events:ListTargetsByRule",
        "events:TestEventPattern",
        "events:DescribeArchive"
      ],
      "Resource": "events.amazonaws.com"
    }
  ]
}
```
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

For EventBridge to access targets that are an API destination, a Kinesis stream, a Systems Manager Run Command, an AWS Step Functions state machine, or an Amazon Elastic Container Service task, you must specify an IAM role for accessing that target, and the role must have a certain policy attached.

If the target is an API destination, the role that you specify must include the following policy.

```
{
    "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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [ "events:PutRecord" ],
        }
    ]
}
```
Using identity-based policies (IAM policies)

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": [
    {
      "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": [
    {
      "Action": "states:SendTaskInput",
      "Effect": "Allow",
    }
  ]
}
```
If the target is an Amazon ECS task, the role that you specify must include the following policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "ecs:RunTask" ],
      "Resource": [ "arn:aws:ecs:*:account-id:task-definition/task-definition-name" ],
      "Condition": {
        "ArnLike": {
          "ecs:cluster": "arn:aws:ecs:*:account-id:cluster/cluster-name"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [ "iam:PassRole" ],
      "Resource": [ "*" ],
      "Condition": {
        "StringLike": {
          "iam:PassedToService": "ecs-tasks.amazonaws.com"
        }
      }
    }
  ]
}
```

**Customer-managed policy examples**

The following examples show user policies that grant permissions for EventBridge actions. These policies work when you use the EventBridge API, AWS SDKs, or the AWS CLI.

**Note**

All examples use the US West (Oregon) Region (us-west-2) and contain fictitious account IDs that you need to replace to use these policies.

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

**Example 1: Access to Amazon EC2 targets**

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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "states:StartExecution" ],
      "Resource": [ "arn:aws:states:*:*:stateMachine:*" ]
    }
  ]
}
```
Using identity-based policies (IAM policies)

Example 2: Kinesis

The following policy allows EventBridge to relay events to the Kinesis streams in your account.

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

Example 3: Console access

The following policy allows IAM users to use the EventBridge console.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ConsoleAccess",
      "Effect": "Allow",
      "Action": [
        "automation:CreateAction",
        "automation:DescribeAction",
        "automation:UpdateAction",
        "autoscaling:DescribeAutoScalingGroups",
        "cloudtrail:DescribeTrails",
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "events:*",
        "iam:ListRoles",
        "kinesis:ListStreams",
        "lambda:AddPermission",
        "lambda:ListFunctions",
        "lambda:RemovePermission",
        "sns:GetTopicAttributes",
        "sns:ListTopics",
        "sns:SetTopicAttributes",
      ],
      "Resource": "*"
    }
  ]
}
```
Using identity-based policies (IAM policies)

Example 4: EventBridgeFullAccess

The following policy allows all AWS resources to perform actions against EventBridge through the AWS CLI and the SDK.

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

Example 5: ReadOnlyAccess

The following policy allows all AWS resources to have read-only access to EventBridge.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnlyAccess",
      "Effect": "Allow",
      "Action": ["events:Describe*", "events:List*", "events:TestEventPattern"],
      "Resource": "*"
    }
  ]
}
```
Example 6: Using tagging to control access to rules

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. 256).

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`.

```json
{
    "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>AmazonEventBridgeReadOnlyAccess</td>
<td>EventBridge added permissions necessary for view endpoint information.</td>
<td>April 7, 2022</td>
</tr>
<tr>
<td>– Update to an existing policy</td>
<td>The following actions were added:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• events:ListEndpoints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• events:DescribeEndpoint</td>
<td></td>
</tr>
<tr>
<td>AmazonEventBridgeReadOnlyAccess</td>
<td>EventBridge added permissions necessary for view connection and API destination information.</td>
<td>March 4, 2021</td>
</tr>
<tr>
<td>– Update to an existing policy</td>
<td>The following actions were added:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• events:DescribeConnection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• events:ListConnections</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• events:DescribeApiDestination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• events:ListApiDestinations</td>
<td></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. 203) – Update to an existing policy | 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. 34) runs in EventBridge, all of the targets (p. 46) 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 (p. 202) 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. 212)
- CloudWatch Logs permissions (p. 212)
- AWS Lambda permissions (p. 213)
- Amazon SNS permissions (p. 214)
- Amazon SQS permissions (p. 215)

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.

```json
{
  "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.

212
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"
}
```

For more information, see PutResourcePolicy in the CloudWatch Logs API Reference guide.

**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 an Amazon SNS topic, use the aws sns get-topic-attributes and the aws sns set-topic-attributes commands.

**Note**

You can't use of 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",
       "Policy": "{"Version":"2012-10-17","Id":"__default_policy_ID",
       "Statement": [{"Sid":"__default_statement_ID","Effect":"Allow","Principal":
       {"AWS":"*"},"Action": ["SNS:GetTopicAttributes","SNS:SetTopicAttributes
       ","SNS:AddPermission","SNS:RemovePermission","SNS:DeleteTopic"],
       "SNS:Subscribe","SNS:ListSubscriptionsByTopic","SNS:Publish"],"Resource
       ":"arn:aws:sns:region:account-id:topic-name","Condition": {"StringEquals":
       {"AWS:SourceOwner": "account-id"}}}],
       "SubscriptionsPending": "0"
     }
   }
   ```

2. Use a JSON to string converter to convert the following statement to a string.

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

   After you convert the statement to a string, it looks like the following example.

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

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
```

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": "EventsToMyQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Condition": {
    "ArnEquals": {
      "aws:SourceArn": "arn:aws:events:region:account-id:rule/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
  "ArnEquals": {
    "aws:SourceArn": "arn:aws:events:region:account-id:rule/rule-name"
  }
}
```
4. Create a file called `set-queue-attributes.json` with the following content.

```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.
Resource-based policies for Amazon EventBridge schemas

The EventBridge schema registry (p. 87) supports resource-based policies (p. 212). 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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Test",
      "Effect": "Allow",
      "Action": [
        "schemas:*"
      ],
      "Principal": {
        "AWS": [217
```
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*"
      ]
    }
  ]
}
```
Resource-based policies for EventBridge schemas

```json
"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. 16) 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. 34) 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. 46) 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"
            }
         }
      }
   ]
}
```

The following table shows the condition keys and key and value pairs that you can use in a policy in EventBridge.
<table>
<thead>
<tr>
<th>Condition key</th>
<th>Key value pair</th>
<th>Evaluation types</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws:SourceAccount</td>
<td>The account in which the rule specified by aws:SourceArn exists.</td>
<td>Account Id, Null</td>
</tr>
<tr>
<td>aws:SourceArn</td>
<td>The ARN of the rule that is sending the event.</td>
<td>ARN, Null</td>
</tr>
<tr>
<td>events:creatorAccount</td>
<td>&quot;events:creatorAccount&quot;: &quot;creatorAccount&quot;</td>
<td>creatorAccount, Null</td>
</tr>
<tr>
<td></td>
<td>For creatorAccount, use the account ID for the account that the rule was created in. Use this condition to authorize API calls on rules from a specific account.</td>
<td></td>
</tr>
<tr>
<td>events:detail-type</td>
<td>&quot;events:detail-type&quot;: &quot;detail-type&quot;</td>
<td>Detail Type, Null</td>
</tr>
<tr>
<td></td>
<td>Where detail-type is the literal string for the detail-type field of the event such as &quot;AWS API Call via CloudTrail&quot; and &quot;EC2 Instance State-change Notification&quot;.</td>
<td></td>
</tr>
<tr>
<td>events:detail.eventTypeCode</td>
<td>&quot;events:detail.eventTypeCode&quot;: &quot;eventTypeCode&quot;</td>
<td>eventTypeCode, Null</td>
</tr>
<tr>
<td></td>
<td>For eventTypeCode, use the literal string for the detail.eventTypeCode field of the event, such as &quot;AWS_ABUSE_DOS_REPORT&quot;.</td>
<td></td>
</tr>
<tr>
<td>events:detail.service</td>
<td>&quot;events:detail.service&quot;: &quot;service&quot;</td>
<td>service, Null</td>
</tr>
<tr>
<td></td>
<td>For service, use the literal string for the detail.service field of the event, such as &quot;ABUSE&quot;.</td>
<td></td>
</tr>
<tr>
<td>events:detail.userIdentity.principalId</td>
<td>&quot;events:detail.userIdentity.principalId&quot;</td>
<td>principalId, Null</td>
</tr>
<tr>
<td></td>
<td>For principalId, use the literal string for the detail.userIdentity.principalId field of the event with detail-type &quot;AWS API Call via CloudTrail&quot; such as &quot;AROAIDPPEZS35WEXAMPLE:AssumedRoleSessionName.&quot;.</td>
<td></td>
</tr>
<tr>
<td>events:eventBusInvocation</td>
<td>&quot;events:eventBusInvocation&quot;: &quot;eventBusInvocation&quot;</td>
<td>eventBusInvocation, Null</td>
</tr>
<tr>
<td></td>
<td>For boolean, use true when a rule sends an event to a target that is an event bus in another account. Use false when when a PutEvents API call is used.</td>
<td></td>
</tr>
</tbody>
</table>
### Condition key

<table>
<thead>
<tr>
<th>Condition key</th>
<th>Key value pair</th>
<th>Evaluation types</th>
</tr>
</thead>
<tbody>
<tr>
<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>
<td>Not intended for use in customer policies.</td>
</tr>
<tr>
<td>events:source</td>
<td>&quot;events:source&quot;: &quot;source &quot; Use <code>source</code> 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 <code>source</code>, see the example events in Events from AWS services (p. 98).</td>
<td>Source, Null</td>
</tr>
<tr>
<td>events:TargetArn</td>
<td>&quot;events:TargetArn&quot;: &quot;target-arn &quot; For <code>target-arn</code>, use the ARN of the target for the rule, for example &quot;arn:aws:lambda:<em>:</em>:function:*&quot;.</td>
<td>ArrayOfARN, Null</td>
</tr>
</tbody>
</table>

For example policy statements for EventBridge, see Managing access permissions to your Amazon EventBridge resources (p. 198).

### Topics

- Example: Using the creatorAccount condition (p. 224)
- Example: Using the eventBusInvocation condition (p. 225)
- Example: Limiting access to a specific source (p. 225)
- Example: Defining multiple sources that can be used in an event pattern individually (p. 227)
- Example: Defining a source and a DetailType that can be used in an event pattern (p. 228)
- Example: Ensuring that the source is defined in the event pattern (p. 229)
- Example: Defining a list of allowed sources in an event pattern with multiple sources (p. 230)
- Example: Limiting PutRule access by detail.service (p. 231)
- Example: Limiting PutRule access by detail.eventTypeCode (p. 231)
- Example: Ensuring that only AWS CloudTrail events for API calls from a certain PrincipalId are allowed (p. 232)
- Example: Limiting access to targets (p. 233)

### 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutRuleForOwnedRules",
            "Effect": "Allow",
```
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.

```json
{
    "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**

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

**Policy B: allow events only from Amazon EC2**

```json
{}
"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><code>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ] }</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>{ &quot;source&quot;: [ &quot;aws.ec2&quot;, &quot;aws.s3&quot; ] }</code></td>
<td>Yes</td>
<td>No (Source aws.s3 isn't allowed)</td>
</tr>
<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>
<td>Yes</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>{ &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>{</td>
<td>No</td>
</tr>
</tbody>
</table>
Example: Defining a source and a $\texttt{DetailType}$ that can be used in an event pattern

The following policy allows events only from the $\texttt{aws.ec2}$ source with $\texttt{DetailType}$ equal to $\texttt{EC2 instance state change notification}$.

```json
{
   "Version": "2012-10-17",
   "Statement": [
   ]
}
```

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>No</td>
</tr>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ecs&quot; ]</td>
<td>No</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>
</tr>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ], &quot;detail-type&quot;: [ &quot;EC2 Instance Health Failed&quot; ]</td>
<td>No</td>
</tr>
</tbody>
</table>
IAM policy conditions

Event pattern | Allowed by the policy
---|---
{ "detail-type": [ "EC2 Instance State-change Notification" ] } | No

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>{ &quot;source&quot;: [ &quot;aws.ec2&quot; ], &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] }</td>
<td>Yes</td>
</tr>
<tr>
<td>{ &quot;source&quot;: [ &quot;aws.ecs&quot;, &quot;aws.ec2&quot; ] }</td>
<td>Yes</td>
</tr>
<tr>
<td>{ &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] }</td>
<td>No</td>
</tr>
</tbody>
</table>

229
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.

```json
{
    "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.autoscaling&quot; ] }</code></td>
<td>No</td>
</tr>
<tr>
<td><code>{ &quot;detail-type&quot;: [ &quot;EC2 Instance State-change Notification&quot; ] }</code></td>
<td>No</td>
</tr>
</tbody>
</table>
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.

```json
{
  "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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPutRuleEventsWithDetailServiceEC2",
      "Effect": "Allow",
      "Action": "events:PutRule",
      "Resource": "*",
      "Condition": {
        "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": [ "AIDAJ45QYFFAREXAMPLE" ]
      }
    }
  }
]
```

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>
</table>
| {  
  "detail-type": [ "AWS API Call via CloudTrail" ]
} | No |
| {  
  "detail-type": [ "AWS API Call via CloudTrail" ],
  "detail.userIdentity.principalId": [ "AIDAJ45QYFFAREXAMPLE" ]
} | Yes |
| {  
  "detail-type": [ "AWS API Call via CloudTrail" ],
  "detail.userIdentity.principalId": [ "AROAIDPPEZ535WEXAMPLE: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`.

```
{
  "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.

```
{
  "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. 234)
- Example: EventBridge log file entries (p. 235)

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
- DescribeEventBus
- DescribeRule
- DisableRule
- EnableRule
- ListRuleNamesByTarget
- ListRules
- ListTargetsByRule
- PutPermission
- PutRule
- PutTargets
- RemoveTargets
- TestEventPattern
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"]}",
        "scheduleExpression": ""
    },
    "responseElements": {
        "ruleArn": "arn:aws:events:us-east-1:123456789012:rule/cttest2"
    },
    "requestID": "e9caf887-8d88-11e5-a331-3332aa445952",
    "eventID": "49d14f36-6450-44a5-a501-b0fcd1faeb98",
    "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 the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

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

Additionally, requests must be signed 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. 212) 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. 16) to the number of times a target (p. 46) is invoked by a rule (p. 34).

The following video reviews monitoring and auditing EventBridge behavior through CloudWatch:
Monitoring and auditing events

Topics
- EventBridge metrics (p. 240)
- Dimensions for EventBridge metrics (p. 242)

EventBridge metrics

The AWS/Events namespace includes the following metrics.
All of these metrics use Count as the unit, so Sum and SampleCount are the most useful statistics.

<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>Valid Dimensions: RuleName</td>
<td></td>
</tr>
<tr>
<td>Units: Count</td>
<td></td>
</tr>
</tbody>
</table>
### EventBridge metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Valid Dimensions</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>The number of partner events ingested by EventBridge.</td>
<td>EventSourceName</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: EventSourceName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
<td></td>
<td></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>
<td>RuleName</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IngestionToInvocationTime</td>
<td>The time to process events measured from when they're ingested by EventBridge to the first invocation of a target in your rules. This is a service-level metric measured across all of your rules and buses and provides an indication of the health of the EventBridge service. Any extended periods of high latency over 30 seconds may indicate a service disruption.</td>
<td>EventBusName, RuleName</td>
<td>Milliseconds</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: EventBusName, RuleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
<td></td>
<td></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>
<td>RuleName</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InvocationsFailedToBeSentToDlq</td>
<td>The 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>
<td>RuleName</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InvocationsSentToDlq</td>
<td>The number of invocations that are moved to a dead-letter queue.</td>
<td>RuleName</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>Valid Dimensions: RuleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Dimensions for EventBridge metrics

EventBridge metrics have one *dimension*, or sortable attribute, which is 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>

---

**Metric** | **Description**
---|---
MatchedEvents | The number of events that matched with any rule.  
| | Valid Dimensions: None  
| | Units: Count  
ThrottledRules | The number of rules that have tried to run but are being throttled.  
| | Valid Dimensions: RuleName  
| | Units: Count  
TriggeredRules | 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.  
| | Valid Dimensions: RuleName  
| | Units: Count  

---


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\"}}},
   ```
I just created or modified a rule, but it didn't match a test event

When you make a change to a rule (p. 34) or to its targets (p. 46), incoming events (p. 16) 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. 240).

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 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.

My rule didn't run at the time that I expected

EventBridge runs rules 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 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.

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

To run a rule every five minutes, you use the following cron expression.

```
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.
The IAM role associated with my rule is being ignored when the rule runs

EventBridge only uses IAM roles for rules (p. 34) that send events (p. 16) to Kinesis streams. For rules that invoke Lambda functions or Amazon SNS topics, you need to provide resource-based permissions (p. 212).

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. 19) and rules (p. 34). Be sure to use the correct ARN characters when creating event patterns so that they match the ARN syntax in the event (p. 16) 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. 16) to a target (p. 46) 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. 95)

Some events were never delivered to my target

If the target (p. 46) of an EventBridge rule (p. 34) 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. 16) 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. 34) can run more than once for a single event (p. 16) or scheduled time, or the same target (p. 46) can be invoked more than once for a given triggered rule.
Preventing infinite loops

In EventBridge, it is possible to create a rule (p. 34) 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.

```
{
"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.

```
{"Version":"2012-10-17","Id":"__default_policy_ID","Statement": [{"Sid":"__default_statement_ID","Effect":"Allow","Principal":{"AWS":"*"},"Action": ["SNS:Subscribe"]
```


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

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:

```json
```

Then set the topic attributes by using the AWS CLI, use the following command.

```bash
```

**Note**

If the policy is incorrect, you can also edit the rule (p. 34) in the EventBridge console by removing and then adding it back to the rule. EventBridge sets the correct permissions on the target (p. 46).

**Scenario 2**

If your SNS topic 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": "*"
}
```

*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. 34) with Amazon SNS as the target (p. 46), EventBridge adds permission to your Amazon SNS topic on your behalf. If you delete the rule shortly after you create it, EventBridge
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. 222).

How can I tell when EventBridge rules are broken?

You can use the following alarm to notify you when your EventBridge rules (p. 34) 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. 250)
- PutEvents quotas by Region (p. 252)
- PutPartnerEvents quotas by Region (p. 253)
- Invocation quotas by Region (p. 254)

### EventBridge quotas

EventBridge has the following quotas.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event publishing API requests</td>
<td>PutEvents operations are limited based on AWS Region. See PutEvents quotas by Region (p. 252). 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.</td>
</tr>
<tr>
<td>All other API requests</td>
<td>All EventBridge APIs other than PutEvents are limited to 50 requests per second by default. 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.</td>
</tr>
<tr>
<td>API destinations</td>
<td>Default: 3,000</td>
</tr>
<tr>
<td></td>
<td>The maximum number of API destinations (p. 50) per account per Region. API destinations are HTTP invocation endpoints that you can use as the target for a rule. 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.</td>
</tr>
<tr>
<td>Rate of invocations per API destination</td>
<td>Default: 300 TPS.</td>
</tr>
<tr>
<td></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. <strong>Note</strong> This soft limit prevents your endpoint from being overloaded with event traffic. If your endpoint can...</td>
</tr>
</tbody>
</table>
### EventBridge quotas

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>handle more event traffic, use the Service Quota console to raise this limit.</td>
</tr>
<tr>
<td>Connections</td>
<td>Default: 3,000.</td>
</tr>
<tr>
<td></td>
<td>The maximum number of connections per account per Region. Connections are used with API destination to define the authorization parameters to use to connect to the destination HTTP endpoint.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Event buses</td>
<td>Maximum 100 event buses per account.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Event buses - other quotas</td>
<td>There's no restriction on the rate of events (p. 16) that can be received from AWS services or other AWS accounts. If you send custom events to your event bus (p. 7) using the PutEvents API, the PutEvents API quotas (p. 252) apply. Any events that are sent to the targets of the rules in your account count against your invocations quota. This includes cross-account and cross-Region event delivery.</td>
</tr>
<tr>
<td></td>
<td>The policy size of an event bus is limited to 10240 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>
<tr>
<td></td>
<td>An event bus can be associated with up to a 100 global endpoints (p. 77).</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Event pattern</td>
<td>2048 characters maximum.</td>
</tr>
<tr>
<td>Global endpoints</td>
<td>Maximum 100 endpoints (p. 77) per account per Region.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Global endpoint API requests</td>
<td>CreateEndpoint, DeleteEndpoint, and UpdateEndpoint: 5 TPS.</td>
</tr>
<tr>
<td>Resource</td>
<td>Default limit</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Invocations</td>
<td>An invocation is an event matching a rule and being sent to the rule’s targets. Quotas vary by region. See [Invocation quotas by Region](p. 254). 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.</td>
</tr>
<tr>
<td>ListRuleNamesByTarget</td>
<td>Maximum 100 results per page for requests.</td>
</tr>
<tr>
<td>ListRules</td>
<td>Maximum 100 results per page for requests.</td>
</tr>
<tr>
<td>ListTargetsByRule</td>
<td>Maximum 100 results per page for requests.</td>
</tr>
<tr>
<td>PutEvents entry size</td>
<td>Maximum 256KB</td>
</tr>
<tr>
<td>PutTargets</td>
<td>10 entries per request. Up to 5 targets per rule.</td>
</tr>
<tr>
<td>RemoveTargets</td>
<td>10 entries per request.</td>
</tr>
<tr>
<td>Rules</td>
<td>300 per event bus.</td>
</tr>
<tr>
<td>Schema discovery</td>
<td>Schema discovery will only infer events that are nested up to 255 levels. Any events past 255 levels are ignored.</td>
</tr>
<tr>
<td>Systems Manager Run Command target</td>
<td>1 target key and 1 target value</td>
</tr>
<tr>
<td></td>
<td>Systems Manager Run Command doesn't currently support multiple target values.</td>
</tr>
<tr>
<td>Targets</td>
<td>Maximum 5 targets per rule.</td>
</tr>
</tbody>
</table>

### PutEvents quotas by Region

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. For quotas above 100,000 TPS, our service team will host a call to best support you.

**Note**

Events sent to a different Region using PutEvents count towards the PutEvents quota of the destination Region for the account that owns the role used to send the events.
PutPartnerEvents quotas by Region

To request quota increases, please contact support.

<table>
<thead>
<tr>
<th>Regions</th>
<th>Transactions per second</th>
</tr>
</thead>
</table>
| • AWS GovCloud (US-West)  
  • AWS GovCloud (US-East)  
  • US East (N. Virginia)  
  • US East (Ohio)  
  • US West (N. California)  
  • US West (Oregon)  
  • Africa (Cape Town)  
  • Asia Pacific (Hong Kong)  
  • Asia Pacific (Mumbai) | PutPartnerEvents has a soft limit of 1,400 throughput requests per second and 3,600 burst requests per second by default in all Regions. |
## Invocation quotas by Region

An invocation is an event matching a rule and then being sent to the rule’s targets. If the invocation of a target fails due to a problem with the target service or account throttling, EventBridge reattempts for up to 24 hours.

If you receive events from another account, each event that matches a rule in your account and is sent to the rule’s targets counts against your account’s quota of invocations per second.

After your reach the invocation quota in your region, EventBridge throttles invocations. They still happen but are delayed.

The Service Quotas console provides information about EventBridge quotas. You can also use the Service Quotas console to request quota increases for adjustable quotas. For quotas above 100,000 TPS, our service team will set up a call to best support you.

<table>
<thead>
<tr>
<th>Regions</th>
<th>Transactions per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia Pacific (Osaka)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td></td>
</tr>
<tr>
<td>Canada (Central)</td>
<td></td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
<td></td>
</tr>
<tr>
<td>Europe (Ireland)</td>
<td></td>
</tr>
<tr>
<td>Europe (London)</td>
<td></td>
</tr>
<tr>
<td>Europe (Milan)</td>
<td></td>
</tr>
<tr>
<td>Europe (Paris)</td>
<td></td>
</tr>
<tr>
<td>Europe (Stockholm)</td>
<td></td>
</tr>
<tr>
<td>China (Ningxia)</td>
<td></td>
</tr>
<tr>
<td>China (Beijing)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regions</th>
<th>Invocations per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td></td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td></td>
</tr>
<tr>
<td>Europe (Ireland)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The invocations quota has a soft limit of 18,750 requests per second by default in these Region.</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td></td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The invocations quota has a soft limit of 4,500 requests per second by default in these Regions.</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td></td>
</tr>
<tr>
<td>Europe (London)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td></td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The invocations quota has a soft limit of 2,250 requests per second by default in these Regions.</td>
</tr>
</tbody>
</table>
## Invocation quotas by Region

<table>
<thead>
<tr>
<th>Regions</th>
<th>Invocations per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Canada (Central)</td>
<td>The invocations quota has a soft limit of 1,100 requests per second by default in these Regions.</td>
</tr>
<tr>
<td>• South America (São Paulo)</td>
<td></td>
</tr>
<tr>
<td>• Europe (Paris)</td>
<td></td>
</tr>
<tr>
<td>• Europe (Stockholm)</td>
<td></td>
</tr>
<tr>
<td>• Asia Pacific (Seoul)</td>
<td></td>
</tr>
<tr>
<td>• Asia Pacific (Mumbai)</td>
<td></td>
</tr>
<tr>
<td>• Asia Pacific (Hong Kong)</td>
<td></td>
</tr>
<tr>
<td>• Middle East (Bahrain)</td>
<td></td>
</tr>
<tr>
<td>• AWS GovCloud (US-West)</td>
<td></td>
</tr>
<tr>
<td>• AWS GovCloud (US-East)</td>
<td></td>
</tr>
<tr>
<td>• China (Ningxia)</td>
<td></td>
</tr>
<tr>
<td>• China (Beijing)</td>
<td></td>
</tr>
<tr>
<td>• Asia Pacific (Osaka)</td>
<td></td>
</tr>
<tr>
<td>• Africa (Cape Town)</td>
<td></td>
</tr>
<tr>
<td>• Europe (Milan)</td>
<td></td>
</tr>
<tr>
<td>• China (Beijing)</td>
<td></td>
</tr>
<tr>
<td>• Africa (Cape Town)</td>
<td></td>
</tr>
<tr>
<td>• Europe (Milan)</td>
<td></td>
</tr>
</tbody>
</table>

The invocations quota has a soft limit of 750 requests per second by default in these Regions.
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. 34) and event buses (p. 7). 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>
</table>
| Updated the AmazonEventBridgeReadOnlyAccess policy. | You can now view endpoint information.  
- AmazonEventBridgeReadOnlyAccess policy (p. 204) | April 07, 2022 |
| Added support for global endpoints. | 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:  
- Making applications Regional-fault tolerant with global endpoints and event replication (p. 77)  
- CreateEndpoint | April 07, 2022 |
| Added support for archives and event replays. | 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:  
- Archiving Amazon EventBridge events (p. 74).  
- CreateArchive  
- StartReplay | November 05, 2020 |
| 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. 95).  
- 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. 85)  
- 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. 217)  
- Policy in the EventBridge Schema Registry API Reference | April 30, 2020 |
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tags for Event Buses</td>
<td>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.</td>
<td>February 24, 2020</td>
</tr>
<tr>
<td>• Amazon EventBridge tags (p. 256)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Tag-based policies (p. 195)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TagResource</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• UntagResource</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• ListTagsForResource</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increased service quotas</td>
<td>Amazon EventBridge has increased quotas for invocations and for <code>PutEvents</code>. Quotas vary by region, and can be increased if necessary.</td>
<td>February 11, 2020</td>
</tr>
<tr>
<td>• Amazon EventBridge quotas (p. 250)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• PutEvents quotas by Region (p. 252)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Invocation quotas by Region (p. 254)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Request a Quota Increase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Added a new topic on transforming target input, and added a link to Application Auto Scaling Events.</td>
<td>Improved documentation on the input transformer.</td>
<td>December 20, 2019</td>
</tr>
<tr>
<td>• Transforming Amazon EventBridge target input (p. 70)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Use Input Transformer to extract data from an event and input that data to the target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Tutorial: Use input transformer to customize what EventBridge passes to the event target (p. 125)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Added a link to Application Auto Scaling Events.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Application Auto Scaling Events and EventBridge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Events from AWS services (p. 98)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-based filtering</td>
<td>December 19, 2019</td>
<td></td>
</tr>
<tr>
<td>Added links to Amazon Augmented AI event examples.</td>
<td>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.</td>
<td>December 13, 2019</td>
</tr>
<tr>
<td>• Use Events in Amazon Augmented AI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Events from AWS services (p. 98)</td>
<td></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 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. 98) | 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. 85)  
- 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 |