Amazon EventBridge Onboarding Guide

Amazon EventBridge

Amazon EventBridge: Amazon EventBridge Onboarding Guide

Table of Contents

Amazon EventBridge Partner Onboarding Guide

Amazon EventBridge enables customers to build event-driven applications that subscribe to events from any participating software-as-a-service (SaaS) product without needing to write undifferentiated connection code. SaaS providers can enhance and extend their platform by publishing their events to EventBridge. They can rely on EventBridge to manage the routing of those events to customers in a reliable and secure manner. This capability enables customers to react to events emitted by SaaS applications as easily and reliably as they react to events originating from other AWS services. The events can then be sent to AWS services such as AWS Lambda, Amazon Simple Queue Service (Amazon SQS), Amazon Simple Notification Service (Amazon SNS), or Amazon Kinesis Data Firehose.

EventBridge provides a push API, PutPartnerEvents, for SaaS applications to use to push events when a change happens in the SaaS system. After EventBridge receives an event, it routes it to the event buses of the appropriate customer. The event is then matched against rules configured by the customers and delivered to the appropriate targets in those customers' accounts.

Amazon EventBridge is a set of APIs that make it easy for SaaS providers to inject events for their customers' applications to process inside AWS. This document describes how AWS Partner Network (APN) partners can set up integration with EventBridge.

Terminology

Partner event source

A resource created by a partner that can be "accepted" by customers as a source of events. To accept a partner event source, the customer creates a custom event bus tied to a partner event source with the same name. Each partner event source is customer-specific. For a partner to relay a type of event to multiple customers, the partner must create an event source for each customer.

Default event bus

An existing EventBridge resource that receives events from a variety of sources and delivers events. Currently, default event buses cannot be created or deleted, and each AWS customer has one default event bus.

Custom event bus for partner event source

A new EventBridge resource that can be created by customers and matched to event sources created by SaaS partners. A custom event bus receives events pushed by the partner and delivers them.

Resources

Amazon EventBridge Documentation

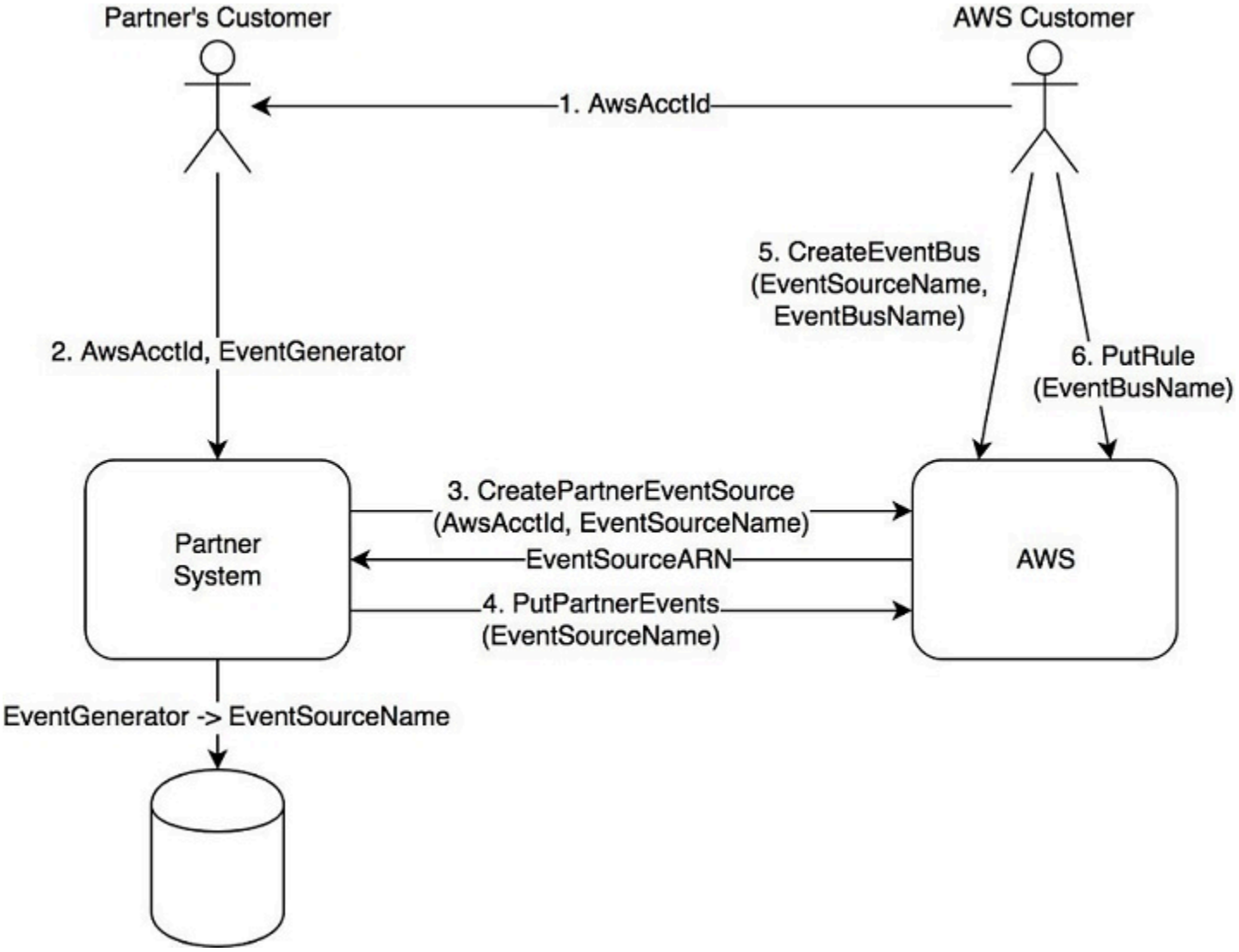
User Guide

API Reference

CLI Reference

Sending events

In this process there are two roles, the AWS customer and the partner's customer. In many cases these roles are be played by the same person, but in some cases there may be separate administrators responsible for managing a company's AWS account and their SaaS application. The following diagram shows how events are sent.



The flow for this diagram is as follows.

1. The AWS customer looks up the AWS account ID and provides it to the partner's customer (for example, 000000000101).
2. The partner's customer navigates to the partner system, selects a partner resource that generates events (for example, acct1/repo1), and enters the AWS customer account ID along with the AWS Region where they want to receive events.
3. The partner system creates an event source by calling the `CreatePartnerEventSource` in the Region specified by the customer and passing in the customer's AWS account ID and a unique name for the event source.
4. The partner sends events to the event source.

5. The AWS customer creates a new custom event bus associated with the partner event source using the console or the `CreateEventBus` API.
6. The AWS customer adds rules and targets to the custom event bus to deliver events to other AWS services.

> **Note**
>
> The partner can send events to an event source as soon as the event source is created. There is no need to wait until a customer associates an event bus before sending events.

Partner delete

The partner can delete an event source at any time by calling the `DeletePartnerEventSource` API. When this happens, the AWS customer sees that the event source is now DELETED. If the customer had an event bus associated with that event source, the event bus continues to exist but cannot receive events.

Event source names

Event sources and partner event buses share the same name, which must follow a specific convention. The name consists of two pieces: the partner name and a name that reflects the resource that will generate events. Both of these pieces must be ASCII strings so that we can use an Amazon Resource Name (ARN) to identify a partner event bus.

Partner event source name

<partner_name>/<event_generator_name>

Partner event bus name

<partner_name>/<event_generator_name>

Partner event source ARN

arn:aws:events:<region>::event-source/aws.partner/<partner_name>/<event_generator_name>

Partner event bus ARN

arn:aws:events:<region>:<aws_customer_account_id>:event-bus/aws.partner/<partner_name>/<event_generator_name>

The **partner name** is determined as part of partner registration and is a unique identifier that provides the AWS customer with confidence that the owner of the event source is a trusted APN partner and is who they claim to be. Each partner name begins with aws.partner/ followed by a fully qualified domain name owned by the partner. The fully qualified domain name (FQDN) used should be one that customers readily associate with the partner or SaaS product.

The **event generator name** is determined by the partner when creating the event source, and it should uniquely identify an event-generating resource within the partner system. It should ideally provide the AWS customer with enough information to decide whether to create a partner event bus. For example, multiple users of a SaaS system can own different resources with the same name (for example, “repo1”). To disambiguate between them, the event generator name should contain the account name as well as the repository name (for example, “acct1/repo1”). The resource names and ARNs for this example are as follows:

Partner event source name

aws.partner/partner_x/acct1/repo1

Partner event bus name

aws.partner/partner_x/acct1/repo1

Partner event source ARN

arn:aws:events:us-east-2::event-source/aws.partner/partner_x/acct1/repo1

Partner event bus ARN

arn:aws:events:us-east-2:000000000101:event-bus/aws.partner/partner_x/acct1/repo1

Event source states

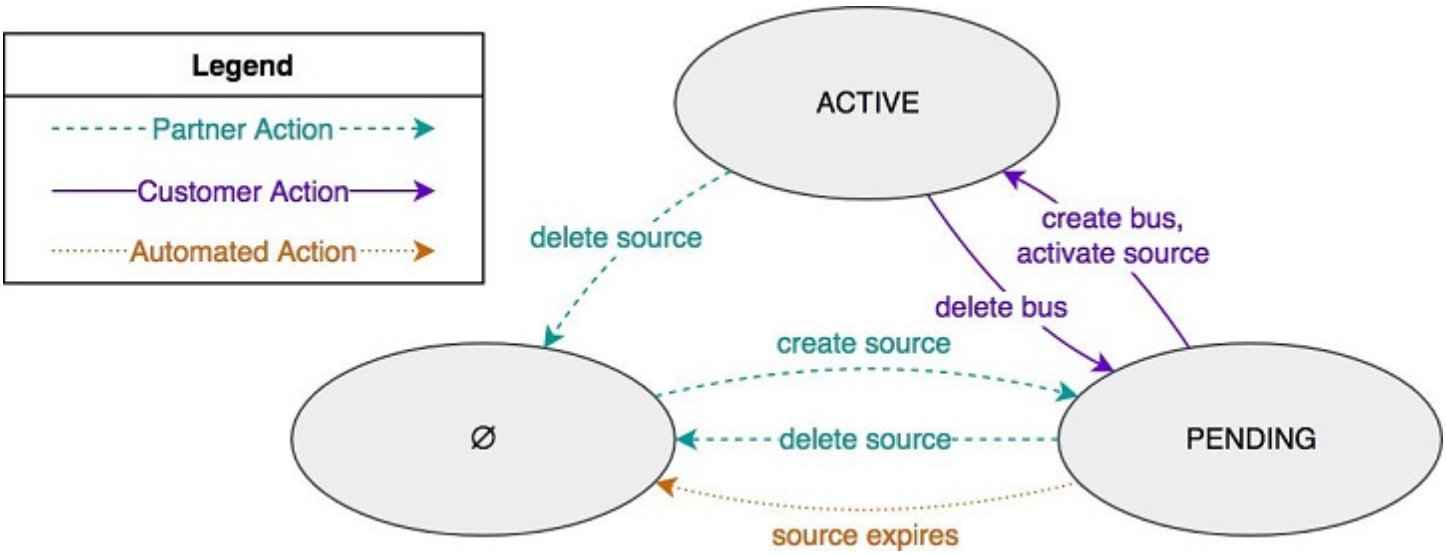
Event sources can be in one of the following states. From the partner side, this state can be found on the event source account rather than the event source itself.

- Nonexistent (∅)
  - The event source either has never been created, has been deleted, or has expired.
  - Rejects events.
- PENDING
  - The event source was recently created.

- Accepts events.

- ACTIVE
  - The AWS customer has created the corresponding partner event bus.
  - Accepts events.

The following diagram shows the relationship between these states.



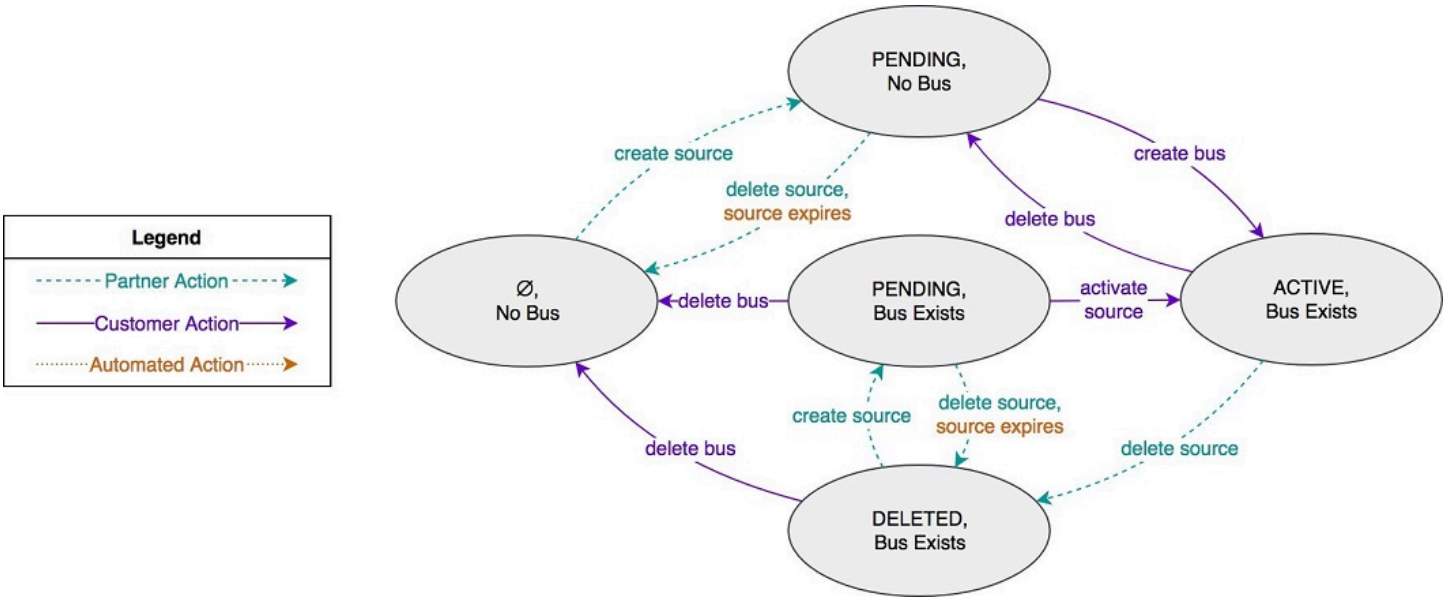
From the AWS customer side, event sources can be in one of the following states.

- Non-existent (∅), event bus does not exist
  - The event source either has never been created, has been deleted, or has expired
  - Rejects events sent by partner
- PENDING, event bus does not exist
  - The event source was recently created
  - Visible to both the partner and the AWS customer
  - Accepts events sent by partner, but does not deliver them to customer
- ACTIVE, event bus exists
  - The AWS customer created the corresponding partner event bus
  - Accepts events sent by the partner and delivers them to customer-defined rules
- DELETED, event bus exists
  - After the event source became ACTIVE, the partner deleted it
  - Informs the AWS customer that the partner is no longer set up to send events

- Allows the partner to recreate the corresponding event source without unexpectedly resurrecting a partner event bus along with any rules attached to it
- Rejects events sent by partner

- PENDING, event bus exists
  - After the event source became DELETED, the partner re-created it
  - Allows the partner to recreate the corresponding event source without unexpectedly resurrecting a partner event bus along with any rules attached to it
  - Allows the AWS customer to reattach an existing partner event bus to an event source without deleting the partner event bus
  - Accepts events sent by the partner, but does not deliver them to customer

The following diagram shows the relationship between these states.



Event source expiration

Periodically, AWS checks all PENDING event sources. Any event sources that are in a PENDING state for over 7 days are automatically deleted. This feature addresses two issues:

- A partner customer can make a mistake, by sharing events with a variety of AWS accounts that they do not own. This could confuse the AWS customer by presenting them with a large list of event sources, most of which are irrelevant or malicious. Automatic expiration helps drastically reduce the noise.

- A partner does not want to send events that no one is listening to. Automatic expiration allows AWS to give partners a definitive response that no one is listening.

Onboarding tasks

1. Partner registration

As a one-time registration step, the partner must provide AWS with the following information:

- DNS fully qualified domain that the partner owns. AWS will append .test at the end of the domain for testing purposes.
- Development/testing:
  - A list of Regions to support
  - A list of accounts to publish events
  - A list of accounts to consume events
- Production:
  - A list of Regions to support
  - A list of accounts to publish events

Additionally, AWS requires the information listed in the topic Partner Terms.

2. Creating partner event sources

The partner updates their user experience to enable the SaaS administrator to create a partner event source.

- User specifies an event generator, an AWS account ID, and a Region in which they want the event source to be created.
- Perform any required authentication checks to ensure that the SaaS administrator has appropriate permissions on the event generator, as determined by the partner.
- Call the `CreatePartnerEventSource` API using the Region supplied by the customer. Specify the event generator and the AWS account ID, using the partner's AWS credentials.
  - To avoid incorrect delivery of events, the event generator name must be unique within the SaaS system. For example, if two SaaS customers set up an event bus for an event generator named "my-channel", then the event generator name should include additional disambiguating information, such as an account ID: "1234.my-channel".

Best practices

Immediate creation: You should make the call to `CreatePartnerEventSource` immediately after the customer submits their account ID and Region selection. You should not wait for events to occur in your system before creating the event source.

Regional selection: You must allow customers to select the AWS Region where they want to receive events and create the event source in that Region. Customers can only associate event buses with event sources in the same Region.

3. Sending events

The partner updates their backend system to push events to AWS.

- When an event is generated, check the persistent data store to determine whether a partner event source exists for the resource that generated this event. If not, do nothing.
- Call the `PutPartnerEvents` API, specifying the event source name, using the partner's AWS credentials.
- Inspect the results to see if there are any failures.
  - For retry-able failures, call the API again (for example, "Internal Service Error").
  - For non-retry-able failures, delete the record from the data store (for example, "Event Bus Not Found").

Best practices

Event source status: As soon as the partner event source is successfully created you should begin sending events to it. You do not need to wait for the event source to become active (i.e. the customer associated an event bus) before sending events. The customer is not charged for events until they associate an event bus.

Batching events: When calling the `PutPartnerEvents` API call you can batch up to 20 events in a single API call as long as the aggregate size of all events is less than 256 kb.

Event detail types: Each event you publish will include a detail type field. This field should be used to identify different types of events produced by your service. All events for a given detail type should share a common schema for the primary event payload (the event detail field). Most SaaS providers will have between 1-15 different detail types. If you have significantly more types

of events, consider consolidating some of your types that share the same payload schema into a single detail type and provide a subtype identifier within the event detail itself. Customers will be able to write rules that match against both the detail type in the envelope as well as the information included in event detail. Do not include personally identifiable information (PII) in the detail type field.

Event resources: The resources field on events is optional. If you choose to use it you should provide a list of unique identifiers for the resources the event is about. For instance, when an Amazon EC2 instance is terminated, the instance's Amazon Resource Name (ARN) is listed in the resources field.

4. Managing partner event buses

The partner updates their user experience to enable the AWS customer to list and delete partner event buses associated with a given event generator.

- Perform any required authentication checks to ensure that the SaaS administrator has appropriate permissions on the event generator, as determined by the partner.
- Call the `ListPartnerEventSources` and `DeletePartnerEventSource` APIs, as appropriate, using the partner's AWS credentials.

FAQ

1. **What delivery guarantees will EventBridge make?**

   EventBridge tries to deliver an event to a target for up to 24 hours, except in scenarios where your target resource is constrained. The first attempt is made as soon as the event arrives in the event stream. However, if the target service is having problems, EventBridge automatically reschedules another delivery in the future. If 24 hours has passed since the arrival of event, no more attempts are scheduled and the `FailedInvocations` metric is published in Amazon CloudWatch. We recommend that you create a CloudWatch alarm on the `FailedInvocations` metric.

   For more information, see My event's delivery to the target experienced a delay in the *Amazon EventBridge User Guide*.

2. **What happens when the APN partner deletes an event source?**

   If the APN partner deletes an event source, any rules attached to the corresponding event bus remain in a Deleted state in the customer's AWS account. If the event source is later recreated and reassociated, these rules become active again.

3. **What happens when the AWS customer deletes an event bus?**

   If the AWS customer deletes an event bus associated with a partner event source, the event source will return to a Pending state. For more information, see Event source states. The event source will accept events sent by a SaaS partner system when calling PutPartnerEvents, but does not deliver them to customer. The event source will expire after being in a Pending state for an extended period of time and will automatically be deleted. For more infprmation, see: Event source expiration.

4. **How can customers create event buses in different Regions?**

   EventBridge is a regional service and event buses must exist in the same Region as the partner event sources they are attached to. You should allow your customers to select which Region they want to create the event source in when they register their AWS account with you. You can call `CreatePartnerEventSource` in any Region regardless of where the call originates from. You can use SDK to specify the Region when you create a client.

5. **What happens to events sent to a partner event source before the customer attaches an event bus?**

Events sent to an event source in the PENDING state will be dropped.

6. **What is the retry policy if messages cannot be delivered?**

   The separation between the partner event source and customer event bus is a logical one. Messages successfully sent to an event source in the ACTIVE state with a customer event bus attached will always propagate to the event bus.

   Message delivery to downstream customer-owned targets follows the same retry policies as the policies documented in Troubleshooting Amazon EventBridge.

Partner APIs

The following APIs can only be called by AWS accounts associated with registered partners. The sections that follow provide sample requests and sample responses for each of the APIs.

- CreatePartnerEventSource
- DeletePartnerEventSource
- DescribePartnerEventSource
- ListPartnerEventSources
- ListPartnerEventSourceAccounts
- PutPartnerEvents

CreatePartnerEventSource

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.CreatePartnerEventSource

{
  "Account": "000000000101",
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{

  "EventSourceArn": "arn:aws:events:us-east-2::event-source/aws.partner/partner_x/
acct1/channel1"
}
```

DeletePartnerEventSource

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.DeletePartnerEventSource

{
  "Account": "000000000101",
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

DescribePartnerEventSource

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.DescribePartnerEventSource

{
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{

  "Arn": "arn:aws:events:us-east-2::event-source/aws.partner/partner_x/acct1/channel1",
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

ListPartnerEventSources

Sample request

```
POST / HTTP/1.1
```

```
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.ListPartnerEventSources

{
  "NamePrefix": "aws.partner/partner_x/acct1/"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
  "PartnerEventSources": [
    {

      "Arn": "arn:aws:events:us-east-2::event-source/aws.partner/partner_x/acct1/
channel1",
      "Name": "aws.partner/partner_x/acct1/channel1"
    },
    {
      "Arn": "arn:aws:events:us-east-2::event-source/aws.partner/partner_x/acct1/
channel2",
      "Name": "aws.partner/partner_x/acct1/channel2"
    },
    {
      "Arn": "arn:aws:events:us-east-2::event-source/aws.partner/partner_x/acct1/
channel3",
      "Name": "aws.partner/partner_x/acct1/channel3"
    }
  ]
```

```
}
```

ListPartnerEventSourceAccounts

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.ListPartnerEventSourceAccounts

{
  "EventSourceName": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
  "PartnerEventSourceAccounts": [
    {
      "Account": "000000000101",
      "CreationTime": "2018-11-20T22:03:15",
      "State": "PENDING"
    }
  ]
}
```

PutPartnerEvents

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.PutPartnerEvents

{
"Entries": [
  {
      "Source": "aws.partner/partner_x/acct1/channel1",
      "Detail": "{ \"key1\": \"value1\", \"key2\": \"value2\" }",
      "Resources": [
        "resource1",
        "resource2"
      ],
      "DetailType": "myDetailType",
    },
    {
      "Source": "aws.partner/partner_x/acct2/channel1",
      "Detail": "{ \"key1\": \"value3\", \"key2\": \"value4\" }",
      "Resources": [
        "resource1",
        "resource2"
      ],
      "DetailType": "myDetailType"
    }
  ]
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
 "FailedEntryCount": 0,
 "Entries": [
   {
     "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
   },
   {
     "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
   }
 ]
}
```

Customer APIs

The APIs in this topic are for use by customers. The sections that follow provide sample requests and sample responses for each of the APIs.

ActivateEventSource

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.ActivateEventSource

{
 "Name": "partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

CreateEventBus

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.CreateEventBus

{
  "EventSourceName": "aws.partner/partner_x/acct1/channel1",
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{

  "EventBusArn": "arn:aws:events:us-east-2:000000000101:event-bus/aws.partner/
partner_x/acct1/channel1"
}
```

DeleteEventBus

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.DeleteEventBus

{
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

DescribeEventSource

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
```

```
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.DescribeEventSource

{
  "Name": "aws.partner/partner_x/acct1/channel1"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
  "Arn": "arn:aws:events:us-east-2::event-source/partner_x/acct1/channel1",
  "CreationTime": "2018-11-20T22:03:15",
  "CreatedBy": "partner_x",
  "Name": "aws.partner/partner_x/acct1/channel1",
  "State": "ACTIVE"
}
```

ListEventBuses

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
```

```
Connection: Keep-Alive
X-Amz-Target: AWSEvents.ListEventBuses

{
  "Limit": "3"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
  "EventBuses": [
    {

    "Arn": "arn:aws:events:us-east-2:000000000101:event-bus/default",
      "Name": "default"
    },
    {
      "Arn": "arn:aws:events:us-east-2:000000000101:event-bus/my-bus",
      "Name": "my-bus"
    },
    {

    "Arn": "arn:aws:events:us-east-2:000000000101:event-bus/partner_x/acct1/channel1",
      "Name": "partner_x/acct1/channel1"
    },
    {
      "Arn": "arn:aws:events:us-east-2:000000000101:event-bus/partner_y/acct1/
trigger1",
      "Name": "partner_y/acct1/trigger1"
    },
    {

      "Arn": "arn:aws:events:us-east-2:000000000101:event-bus/partner_z/acct1/repo1",
      "Name": "partner_z/acct1/repo1"
    }
  ],
  "NextToken": "AAAAAAAAAAAAAAA"
```

```
}
```

ListEventSources

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.ListEventSources

{
  "Limit": "3"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
  "EventSources": [
    {

    "Arn": "arn:aws:events:us-east-2::event-source/partner_x/acct1/channel1",
      "CreatedBy": "partner_x",
      "CreationTime": "2018-11-20T22:03:15",
      "Name": "aws.partner/partner_x/acct1/channel1",
      "State": "ACTIVE"
    },
```

```
    {
      "Arn": "arn:aws:events:us-east-2::event-source/partner_y/acct1/trigger1",
      "CreatedBy": "partner_y",
      "CreationTime": "2018-12-12T13:52:52",
      "Name": "partner_y/acct1/trigger1",
      "State": "DELETED"
    },
    {

      "Arn": "arn:aws:events:us-east-2::event-source/partner_z/acct1/repo1",
      "CreatedBy": "partner_z",
      "CreationTime": "2018-12-20T00:09:55",
      "ExpirationTime": "2019-01-03T00:09:55",
      "Name": "partner_z/acct1/repo1",
      "State": "PENDING"
    }
  ],
  "NextToken": "AAAAAAAAAAAAAAA"
}
```

PutRule

Sample request

```
POST / HTTP/1.1
Host: events.<region>.<domain>
x-amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>,
SignedHeaders=content-type;date;host;user-agent;x-amz-date;x-amz-target;x-amzn-
requestid,
Signature=<Signature>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
X-Amz-Target: AWSEvents.PutRule

{
  "Name": "everything",
  "EventBusName": "aws.partner/partner_x/acct1/channel1",
  "EventPattern": "{ \"account\": [\"000000000101\"] }"
}
```

Sample response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>

{
 "RuleArn":
 "arn:aws:events:us-east-2:000000000101:rule/aws.partner/partner_x/acct1/channel1/
everything"
}
```

Partner terms

Amazon EventBridge unlocks new use cases for your customers by extending your application's integration with AWS services. It typically takes about 5 days development time for Partners to integrate with EventBridge, and about two weeks to be added as an EventBridge SaaS integration partner, but can take longer for complex integrations. Contact the EventBridge Partner Operations team (eventbridge-partners@amazon.com) with any questions about the status of your integration.

To start the process, complete the following steps.

1. Become an APN Partner. If you are not already an APN Partner, you can register for free here.
2. Send an email to eventbridge-partners@amazon.com to request test access to the partner API. Complete and include all fields in the table listed in the section **Request for test access**. An AWS representative will provide an Onboarding Guide. Review the guide for a technical walkthrough of the Amazon EventBridge Partner APIs and how to develop your integration.
3. Complete and test your integration. You may not use customer data for testing.
4. Email a completed version of the **Partner integration questionnaire** to eventbridge-partners@amazon.com. We will respond with a date and time on which we will validate your integration.
5. Validate your integration with an AWS representative. In your validation call, you will be expected to demonstrate a working version of your integration - following the process that a customer would go through to receive events from your service. Ensure you have downloaded Amazon Chime and tested the screensharing functionality before joining your review call.

Request for test access

Required information	Details	Provided answers
APN registered email address	The email address used to register with AWS Partner Network (APN).	

Partner test account ID(s)	AWS account ID(s) that you will use to test partner API calls.	
Customer test account ID(s)	AWS account ID(s) that you will use to test customer API calls	
Regions (at least 2)	The Regions you'd like to use for testing your integration. Provide at least 2 Regions.	

Partner integration questionnaire

General information	Details	Provided answers
Company name	This name is displayed in your partner listing in the AWS Management console.	
AWS account ID(s)	AWS account ID(s) that you will use to call PutPartnerEvents. You may provide multiple accounts.	
Contact email(s)	One or more email addresses that we can use to contact you about your integration. We will use these emails to schedule your validation call, and for any future updates or issues related to your integration.	

Contact phone number(s)	This number will be used to contact you for high severity issues.	
Console marketing material		
Descriptions		
Company description	Description of the company. Displayed in the 'About' section of the integration.	
Integration description	Description of the integrati on and what it will enable users to do.Displayed as the main body of text both under your logo on the ***partner event sources*** page, and to the right of your logo when a customer chooses your integration.	
Square logo SVG (URL)	Logo displayed when a customer chooses your inte gration.	
Square Logo PNG (URL)	Logo displayed when a user clicks into your integration.	
Logo SVG (URL)	Logo displayed when a user loads the ***partner event sou rces*** page.	
Logo PNG (URL)	Logo displayed on the ***Amazon EventBridge Integrations*** page.	
Other		

Architecture diagram (Preferred)	Architecture diagram for illustrating integration use case(s). Displayed when a customer chooses your inte gration under the heading 'How does it work?'	
Use cases	Use case(s) for your integrati on. Displayed when a customer chooses your integration.	
Link(s)	Deep links to your product where a user can set up an integration and/or link to documentation where a user can learn how to setup an integration. Displayed as a 'learn more about company' link when a customer chooses your integration.	
Event detail type catalog	A catalog of detail types that illustrate the types of events your integration publishes . Include this information as part of your customer-facing documentation.	
Sample events catalog	A catalog of complete sample events. Include as part of your customer facing documentation.	

Partner asset guidelines

EventBridge partners need to adhere to the guidelines described in this topic when creating logo assets and diagrams.

Logo guidelines

All logo assets need to be vector graphic files with transparent background and in SVG format.

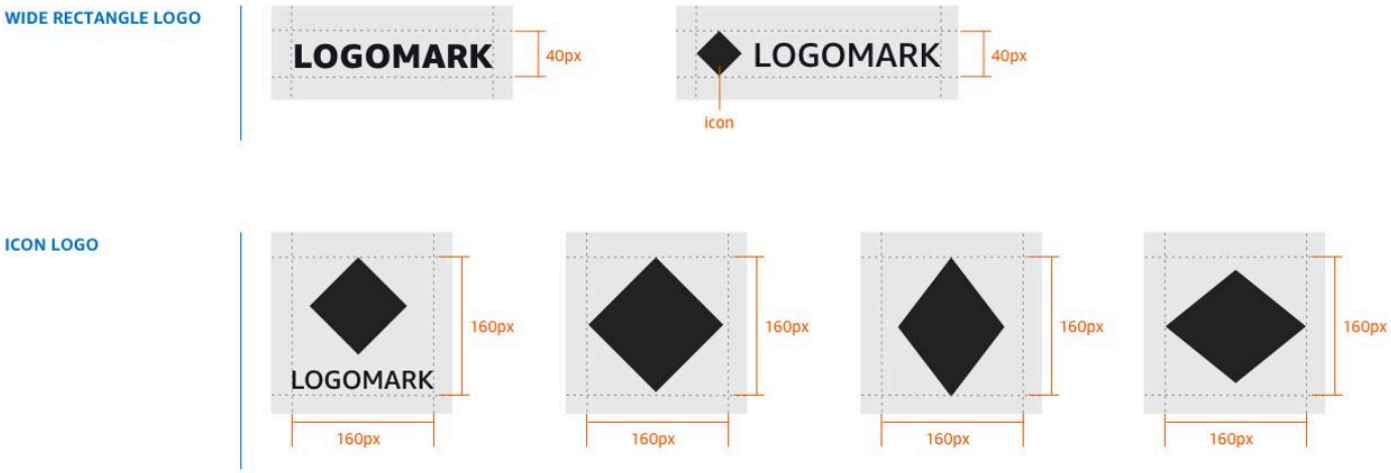
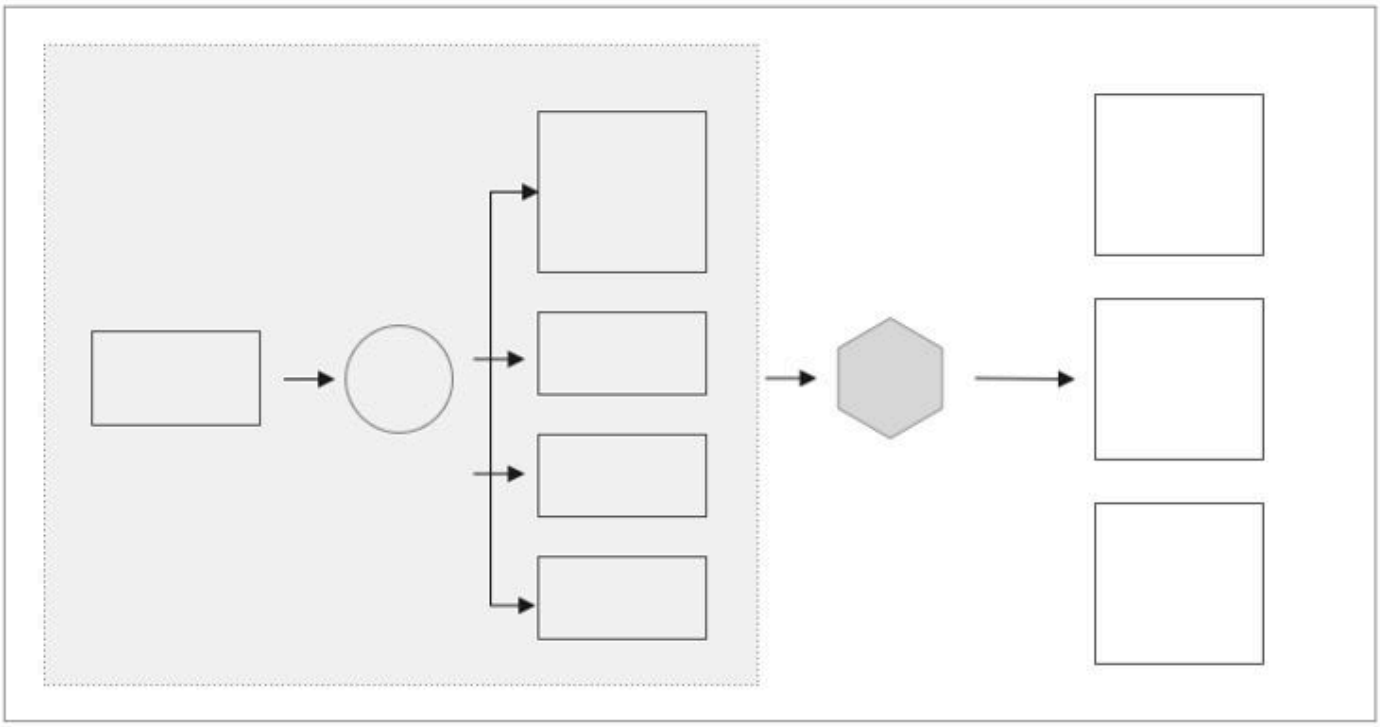


Diagram guidelines

1. If the diagrams are vector files, provide the diagrams in SVG files, otherwise, in image file format (for example, . PNG).
2. Provide the diagrams in white background.
3. Do not include the title in the diagram. You can provide the title separately.



Validation tasks

The validation tasks in this topic are to ensure that you have considered different scenarios that customer may encounter when they consume events. For example, customers may request that the event sources are created in different AWS Regions.

Create event source validation items

1. Ensure that you provide a UI for developers to register their AWS account with your service.
2. The UI should allow the selection of the destination AWS Region.
3. Direct AWS Identity and Access Management(IAM) access to the customer account is not requested nor required.
4. Ensure that the event source is created successfully.
5. The event source name scheme provides global uniqueness within each region (i.e. it includes an appropriate customer identifier or other unique ID).

Send events validation items

1. Ensure that events are received successfully.
2. Data in the event detail and other event metadata fields is formatted appropriately and semantically correct.

Delete event source validation items

- Ensure that the event source is deleted correctly.

Repeat the validation tests for other Regions

1. The event source is created successfully.
2. Events are received successfully.

Partners are required to describe how their integration handles event sources that move from a PENDING to NONEXISTENT state, as well as the mechanism used for handling errors from the `PutPartnerEvents` API call.