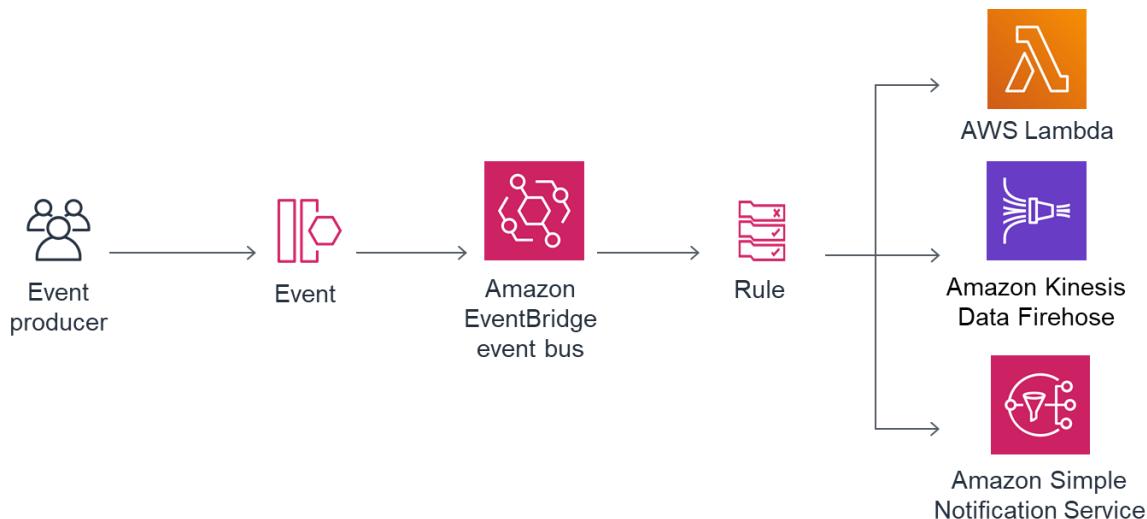


## Event Bridge

We all have been familiar with AWS services like SQS, SNS. Now, let's dive into the event bridge. Amazon event bridge was launched to make it easy for us to develop, and build powerful, event-driven architecture at any scale. Event bridge is a serverless event that lets you receive, filter, transform, route, and deliver events. It gets events from sources like AWS Services and third-party SAAS applications such as Shopify, PageDuty, and DataDog.



The event bridge consists of:

1. Event Bus: Acts as a bus that routes the event. There are 3 kinds of Event Bus.
  - a. default event bus: receives events from AWS services
  - b. custom event bus: receives events from Custom Application and Services
  - c. partner event bus: receives events from SAAS application
2. Rules: matches events from event source and sends them to the target
  - can write up to 300 rules for each event but can be raised
3. Targets: targets are where the event bus routes to

- each rule can have only one target

## Advantages of Event Bridge

- 1.Content-based filtering

- 2.DLQ can be configured per the target

- 3.Third-party service integration without writing code

Today we will learn to schedule an AWS lambda function on AWS event bridge and content-based filtering:

1. Schedule an AWS lambda function:

- a . create a scheduler lambda which will be triggered when a schedule is created

The screenshot shows the AWS Lambda console. At the top, there's a navigation bar with 'Services' and a search bar. The main area is titled 'Function overview' with a 'Info' link. It displays a Lambda function named 'schedulerLambda' with a yellow icon. Below the icon, there's a 'Layers' section with '(0)' and a 'Layers' button. To the right, there are buttons for '+ Add trigger' and '+ Add destination'. On the far right, there's a sidebar with fields for 'Description', 'Last modified' (20 seconds ago), 'Function ARN' (arn:aws:lambda:us-east-1:754763886769:function:schedulerLambda), and 'Function URL' (info). At the bottom, there are tabs for 'Code' (which is selected), 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' tab shows a 'Code source' section with an 'index.mjs' file open. The file content is as follows:

```
index.mjs
1 export const handler = async(event) => {
2   // TODO implement
3   const response = {
4     statusCode: 200,
5     body: JSON.stringify('Hello from Lambda!'),
6   };
7   return response;
8 };
9 
```

- b.Create a rule which will run the lambda function at a specific time  
scheduled rule is not supported on custom event.

## Define rule detail Info

### Rule detail

Name

Maximum of 64 characters consisting of numbers, lower/upper case letters, .,-,\_.

Description - *optional*

Event bus Info

Select the event bus this rule applies to, either the default event bus or a custom or partner event bus.

Enable the rule on the selected event bus

Rule type Info

Rule with an event pattern

A rule that runs when an event matches the defined event pattern. EventBridge sends the event to the specified target.

Schedule

A rule that runs on a schedule

### EventBridge Scheduler - A new AWS scheduling capability! New

A new EventBridge scheduling functionality that provides one-time and recurring scheduling functionality independent of Event buses and rules. You can create a schedule to invoke targets such as a Lambda function.

[Learn More](#)

[Continue to create rule](#)

Cancel

[Continue in EventBridge Scheduler](#)

## Schedule name and description

### Schedule name

Use only letters, numbers, dashes, dots or underscores. Max 64 characters.

### Description - optional

Maximum of 512 characters.

### Schedule group

Each schedule needs to be placed in a schedule group. By default, a schedule is placed in the 'Default' group. You can also [create your own schedule group](#). You can only add tags to a schedule group, not a schedule.



## Schedule pattern

### Occurrence

[Info](#)

You can define an one-time or recurrent schedule.

 One-time schedule Recurring schedule

### Date and time

The date and time to invoke the target.



## Select target

### Target detail

Target API | [Info](#)

Select an API that will be invoked as a target for your schedule.

Templatd targets

All APIs



CodeBuild

StartBuild



CodePipeline

StartPipelineExecut...



Amazon ECS

RunTask



Amazon EventBridge

PutEvents



Kinesis Data Firehose

PutRecord



Amazon Inspector V1

StartAssessmentRun



Kinesis Data Streams

PutRecord



AWS Lambda



Invoke



SageMaker

StartPipelineExecut...



AWS Step Functions

StartExecution



Amazon SNS

Publish



Amazon SQS

SendMessage

c. After the rule is set the scheduler lambda will be invoked and logs can be seen in cloud watch

CloudWatch > Log groups > /aws/lambda/schedulerLambda

## /aws/lambda/schedulerLambda

Actions ▾ View in Logs Insights Search log group

**Log group details**

ARN arn:aws:logs:us-east-1:754763886769:log-group:/aws/lambda/schedulerLambda:*	Metric filters 0	Data protection - new -
Creation time 1 minute ago	Subscription filters 0	Sensitive data found - new -
Retention Never expire	Contributor Insights rules -	KMS key ID -
Stored bytes -		

Log streams Metric filters Subscription filters Contributor Insights Tags Data protection - new

**Log streams (1)**

Filter log streams or try prefix search Exact match Show expired Info < 1 > ⌂

Log stream	Last event time
2023/05/24/[\${LATEST}]603ab3f9693449dbac483dff7dc7a428	2023-05-24 14:37:18 (UTC-05:00)

CloudWatch > Log groups > /aws/lambda/schedulerLambda > 2023/05/24/[\${LATEST}]603ab3f9693449dbac483dff7dc7a428

### Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Actions ▾ Create metric filter

Filter events Clear 1m 30m 1h 12h Custom Display ⌂

Timestamp	Message
No older events at this moment. <a href="#">Retry</a>	
2023-05-24T14:37:18.802-05:00	INIT_START Runtime Version: nodejs:18.v6 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:a19cefba18c0aa228...
2023-05-24T14:37:18.978-05:00	START RequestId: 6e646dbf-5047-41c6-bfad-1be7c1ae6c7c Version: \$LATEST
2023-05-24T14:37:18.997-05:00	END RequestId: 6e646dbf-5047-41c6-bfad-1be7c1ae6c7c
2023-05-24T14:37:18.997-05:00	REPORT RequestId: 6e646dbf-5047-41c6-bfad-1be7c1ae6c7c Duration: 18.62 ms Billed Duration: 19 ms Memory Size: 128...
No newer events at this moment. <a href="#">Auto retry paused.</a> <a href="#">Resume</a>	

## 2.Content Based Filtering

-create a publisher lambda which will trigger the the custom event bus

The screenshot shows the AWS Lambda console interface. At the top, there's a summary card for a function named 'publisherLambda'. It shows 0 triggers and 0 destinations. Below the card are buttons for '+ Add trigger' and '+ Add destination'. To the right, there are sections for 'Description' (empty), 'Last modified' (8 minutes ago), 'Function ARN' (arn:aws:lambda:us-east-1:754763886769:function:publisherLambda), and 'Function URL' (Info). Below the summary is a navigation bar with tabs: Code, Test, Monitor, Configuration, Aliases, and Versions. The 'Code' tab is selected. Under 'Code source' (Info), there's a code editor window titled 'index.mjs'. The code is as follows:

```
1 const AWS = require("aws-sdk");
2 const eventBridge = new AWS.EventBridge({apiVersion:'2015-10-07'});
3 export const handler = async(event) => {
4     var param = {
5         Entries:[
6             {
7                 Detail:JSON.stringify(event),
8                 DetailType: 'Test',
9                 EventBusName: "arn:aws:events:us-east-1:754763886769:event-bus/custom-event-bus",
10                Resources:[],
11                Source:'com.test',
12                TraceHeader:'demo'
13            }
14        ]
15    };
16    const res = await eventBridge.putEvents(param).promise();
17    console.log(res);

```

- create a consumer lambda which will be invoked by the event
- create a custom event

## Create event bus

### Event bus detail

Name

Maximum of 256 characters consisting of numbers, lower/upper case letters, .,-\_.

**Event archive** [Info](#)

Archive events published on this event buses. The default retention period is 'Indefinite'. [Event archive pricing](#)

Disabled

**Schema discovery** [Info](#)

Automatically infer schemas directly from events running on this event bus.

Disabled

The resource-based policy defines who can access your event bus. By default, only the event bus owner can send events to the event bus.

### Resource-based policy

[Load template](#)

1

No resource-based policy

Add the arn of the custom event to the publisher lambda Event Bus attribute

-create rule for the custom event bus which will filter based on the condition

## Review and create schedule

### Step 1: Schedule detail

Edit

#### Schedule detail

Schedule name custom-event	Description -	Schedule group default
Timezone (UTC -05:00) America/Chicago	Occurrence One-time	Date and time 2023-05-24 02:59 (UTC -05:00) America/Chicago
Flexible time window Off		

### Step 2: Target

Edit

#### Target detail

Target AWS Lambda <a href="#">publisherLambda</a>	Target ARN <a href="#">arn:aws:lambda:us-east-1:754763886769:function:publisherLambda</a>
Payload	

Since I am using the student account, I am not able to create the rule but using regular account we can create the rule and trigger the event of the publisher lambda and watch the logs on cloud watch.

Thank You

Sandip Mahato

