

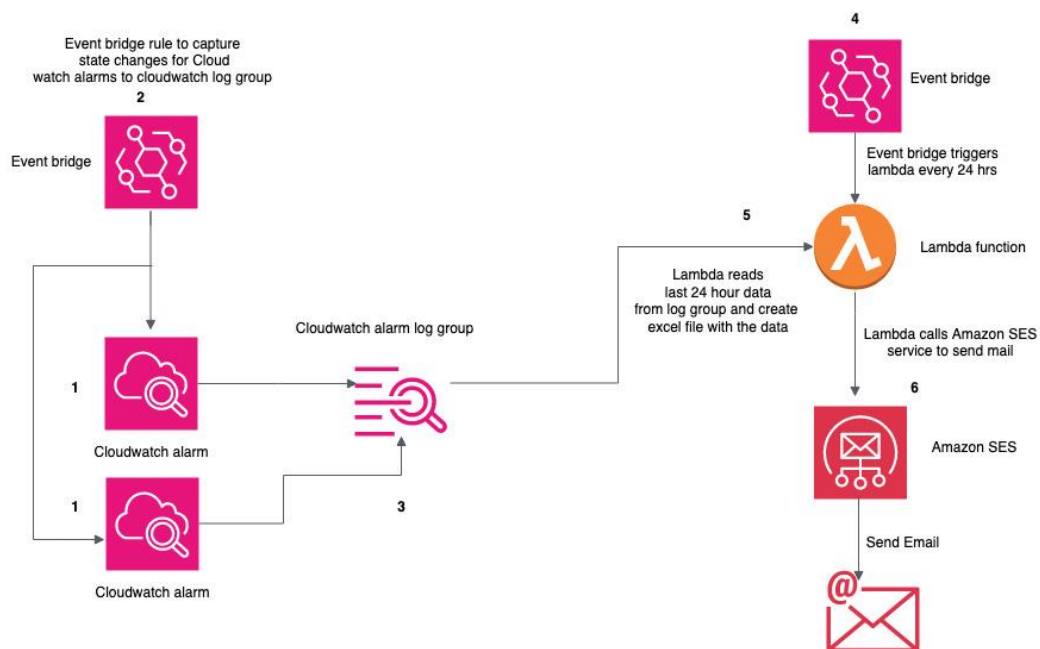
## Cloudwatch Ec2 instances, Cloudwatch Alarm Monitoring requirements: Usecase 01:

Create a csv file which includes the following details of all the cloudwatch alarm in a region. This csv file will be created by AWS lambda with Amazon Event Bridge trigger, which triggers AWS lambda function for every 24 hours and store the created csv file in S3 and also sends email attachment with the help of Amazon SES service. The data of the files are for the last 24 hours.

### CSV file columns :

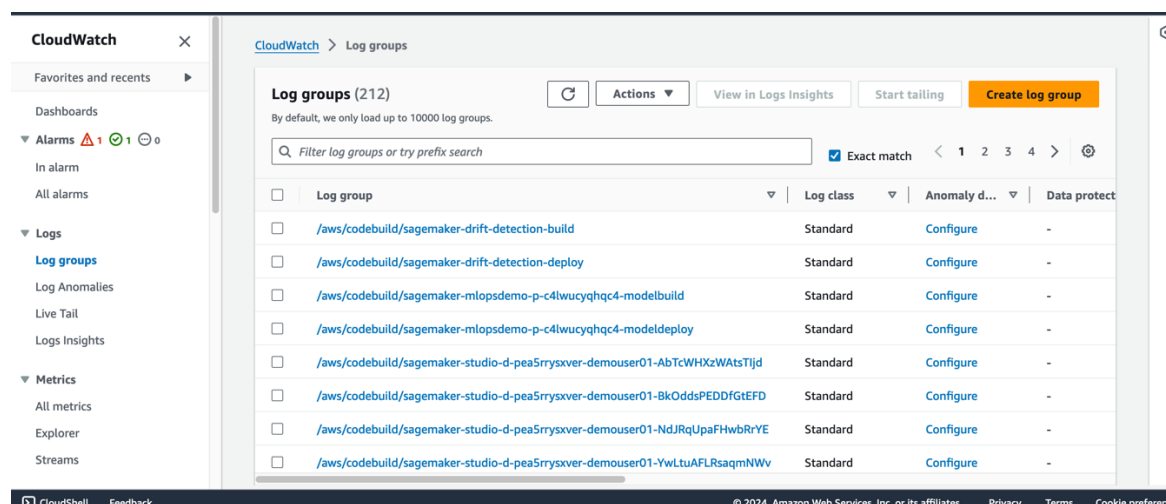
instance name (take from alarm description), instance id, Metric name, Alarm Name ,Alarm Time, Threshold value, Threshold breach value.

### Architecture Diagram:



### Step 1: Create a Log group in Amazon cloudwatch.

- Navigate to cloudwatch console and click 'Log groups' from the left panel and then Click "Create log group" button in right top corner.



- Enter any name in the “Log group name” field and click create button at the right bottom corner of the page.

CloudWatch > Log groups > Create log group

### Create log group

Log group details [info](#)

CloudWatch Logs offers two log classes: Standard and Infrequent Access. [Learn more about the features offered by each log class.](#)

Log group name:

Retention setting:

Log class:

KMS key ARN - optional:

Tags

A tag is a label that you assign to an Amazon Web Services resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your Amazon Web Services costs.

No tags are associated with this log group.

[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Create](#)

## Step 2: Create an Amazon EventBridge rule to capture the cloudwatch alarm state changes into the previously created Log group in step 1.

- Navigate to Amazon Event bridge console and click ‘Create rule’ button in the Amazon Event bridge home page or from the left panel click “Rules” and then Click “Create rule” button in right top corner.

Amazon EventBridge

Application integration

## Amazon EventBridge

A serverless service for building event-driven applications

Amazon EventBridge is a serverless service that uses events to connect application components together, making it easier for developers to build scalable event-driven applications.

How it works

Serverless 101: Amazon EventBridge

Watch on [YouTube](#)

Benefits and features

Build event-driven architectures Write less custom code

Get started

- ☒ **EventBridge Rule**  
A rule matches incoming events and sends them to targets for processing.
- ☐ **EventBridge Pipes**  
A pipe connects an event source to a target with optional filtering and enrichment.
- ☐ **EventBridge Schedule**  
A schedule involves a target one-time or at regular intervals defined by a cron or rate expression.
- ☐ **EventBridge Schema registry**  
Schema registries collect and organize schemas.

[Create rule](#)

Pricing

There is no up-front commitment or minimum fee. You simply pay for what you use and will be charged at the end of the month for your usage. Pricing varies by region.

[Cost calculator](#)

Learn

[Introduction to Amazon EventBridge](#)

[Amazon EventBridge Blog](#)

- Enter any name for this rule in the field “Name” and give the description of this rule in “Description” field and then click “Next” button.

Amazon EventBridge > Rules > Create rule

### Define rule detail [info](#)

Step 1: Define rule detail

Step 2: Build event pattern

Step 3: Select target(s)

Step 4 - optional: Configure tags

Step 5: Review and create

Rule detail

Name:

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

Description - optional:

Event bus:

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: ☒ **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.

[Cancel](#) [Next](#)

- Choose **“AWS Events or event bridge partner events”** for event source radio button. Scroll down and choose **“Use pattern form”** for **“creation method”** section. In the **“Event pattern”** section choose AWS service as **“CloudWatch”** from dropdown and choose **“CloudWatch Alarm State Change”** from the drop down for **“Event type”**. Click **Next**.

**Build event pattern** [Info](#)

**Event source**

Event source  
Select the event source from which events are sent.

☒ **AWS events or EventBridge partner events**  
Events sent from AWS services or EventBridge partners.

☐ **Other**  
Custom events or events sent from more than one source, e.g. events from AWS services and partners.

☐ **All events**  
All events sent to your account.

**Sample event - optional**

You don't have to select or enter a sample event, but it's recommended so you can reference it when writing and testing the event pattern, or filter criteria.

You can reference the sample event when you write the event pattern, or use the sample event to test if it matches the event pattern. Find a sample event, enter your own, or edit a sample event below. [Learn more about the required fields in a sample event.](#)

Sample event type

☒ **AWS events** ☐ EventBridge partner events ☐ Enter my own

Sample events  
Filter by event source and type or by keyword.

Select

1.

Enter the event JSON

[Copy](#)

**Creation method**

Method

☐ **Use schema**  
Use an Amazon EventBridge schema to generate the event pattern.

☒ **Use pattern form**  
Use a template provided by EventBridge to create an event pattern.

☐ **Custom pattern (JSON editor)**  
Write an event pattern in JSON.

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

Event source  
AWS service or EventBridge partner as source

**AWS services**

**AWS service**  
The name of the AWS service as the event source

**CloudWatch**

Event type  
The type of events as the source of the matching pattern

**CloudWatch Alarm State Change**

Event pattern  
Event pattern, or filter to match the events

```
1 {
2   "source": ["aws.cloudwatch"],
3   "detail-type": ["CloudWatch Alarm State Change"]
4 }
```

[Copy](#) [Test pattern](#) [Edit pattern](#)

[Cancel](#) [Previous](#) [Next](#)

- Choose **“AWS service”** radio button from the Target 1 section and choose **“CloudWatch log group”** from the **“select a target”** drop down. In the **“Log group”** section drop down, choose the log group name created in the step 1. Click **Next** and then next and then **“create rule”** button in the review and create section.

**Select target(s)**

**Permissions**

Note: When using the EventBridge console, EventBridge will automatically configure the proper permissions for the selected targets. If you're using the AWS CLI, SDK, or CloudFormation, you'll need to configure the proper permissions.

**Target 1**

Target types  
Select an EventBridge event bus, EventBridge API destination (SaaS partner), or another AWS service as a target.

☐ EventBridge event bus ☐ EventBridge API destination ☒ **AWS service**

Select a target [Info](#)  
Select target(s) to invoke when an event matches your event pattern or when schedule is triggered (limit of 5 targets per rule)

**CloudWatch log group**

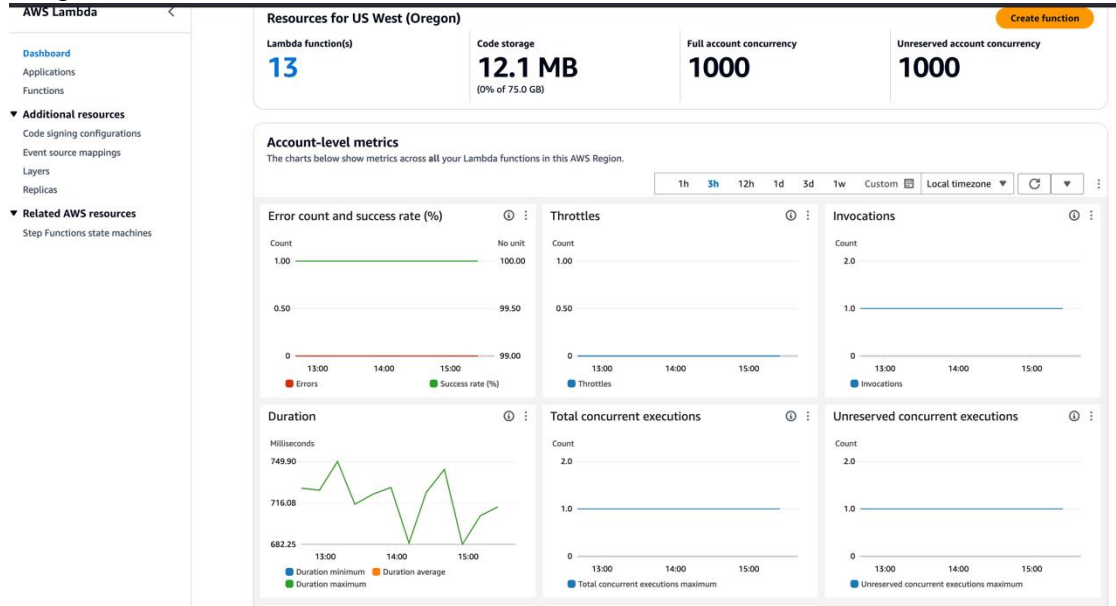
Log Group:  
☐ /aws/events/ ☒ /aws/events/cloudwatchalarms [Copy](#)

**Additional settings**

[Add another target](#) [Cancel](#) [Skip to Review and create](#) [Previous](#) [Next](#)

### Step 3: Create a AWS lambda function with the attached sample code Cloudwatch\_alarm\_dataCSV\_to\_mail\_s3.py

- Navigate to AWS Lambda dashboard and click create function button.



- Enter the function name and choose python as runtime and click create function button.

[Lambda](#) > [Functions](#) > [Create function](#)

**Create function** [Info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
Select a container image to deploy for your function.

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

[Change default execution role](#)

[Advanced settings](#)

[Cancel](#) [Create function](#)

- Click "Layers" from the function overview section and then click "Add a layer" button.

[Lambda](#) > [Functions](#) > [cloudwatch\\_alarm\\_monitor\\_aws\\_lambda](#)

[Throttle](#) [Copy ARN](#) [Actions](#)

**Function overview** [Info](#)

[Diagram](#) [Template](#)

[+ Add trigger](#) [+ Add destination](#)

**cloudwatch\_alarm\_monitor\_aws\_lambda**

[Layers](#) (0)

[Export to Application Composer](#) [Download](#)

**Description**

Last modified 45 seconds ago

**Function ARN**  
arn:aws:lambda:us-west-2:154985105880:function:cloudwatch\_alarm\_monitor\_aws\_lambda

**Function URL** [Info](#)

**Code source** [Info](#)

[Upload from](#)

**Code source**

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO: implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')}
8
9
```

1:1 Python Spaces: 4

Code properties [Info](#)

Package size

299.0 byte

SHA256 hash

HAPq9EReJVEC5gLavtc/gyd5vZtd9eiUGF932t0BkY=

Last modified

April 7, 2024 at 03:46 PM GMT+5:30

Runtime settings [Info](#)

Runtime

Python 3.12

Handler [Info](#)

lambda\_function.lambda\_handler

Architecture [Info](#)

x86\_64

▶ Runtime management configuration

Edit

Edit runtime management configuration

Layers [Info](#)

Edit

Add a layer

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
There is no data to display.					

- 
- Choose “AWSSDKPandas-Python312” as AWS layers and choose latest version from version drop down and then click “Add” button.

Lambda > Layers > Add layer

Add layer

Function runtime settings

Runtime

Python 3.12

Architecture

x86\_64

Choose a layer

Layer source [Info](#)

Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also [create a new layer](#).

☒ AWS layers  
Choose a layer from a list of layers provided by AWS.

☐ Custom layers  
Choose a layer from a list of layers created by your AWS account or organization.

☐ Specify an ARN  
Specify a layer by providing the ARN.

AWS layers

Layers provided by AWS that are compatible with your function's runtime.

AWSSDKPandas-Python312

▼

Version

6

▼

Cancel

Add

- 
- Click “configuration” tab and then click “permissions” tab. Click on the Role name and it will take us to IAM and then attach policies for this Lambda function to have access to Amazon Cloudwatch, S3 bucket and Amazon SES service.

cloudwatch\_alarm\_monitor\_aws\_lambda

Throttle

Copy ARN

Actions

Function overview [Info](#)

Diagram

Template

cloudwatch\_alarm\_monitor\_aws\_lambda

Layers (1)

+ Add trigger

+ Add destination

Description

-

Last modified

12 minutes ago

Function ARN

arn:aws:lambda:us-west-2:154985105880:function:cloudwatch\_alarm\_monitor\_aws\_lambda

Function URL [Info](#)

-

Code

Test

Monitor

Configuration

Aliases

Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

Monitoring and operations tools

Execution role

Role name

cloudwatch\_alarm\_monitor\_aws\_lambda-role-jun0714

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs

3 actions, 2 resources

By action

By resource

Resource

am:aws:logs:us-west-2:154985105880:\*

Actions

Allow: logs:CreateLogGroup

View role document

- Click “configuration” tab and then click “general configuration” tab. Click Edit button to update the Timeout and then click save button.

The screenshot shows the AWS Lambda console for the function `cloudwatch_alarm_monitor_aws_lambda`. The **Configuration** tab is selected, and the **General configuration** sub-tab is active. The **Timeout** is set to 0 min 3 sec. The **Memory** is set to 128 MB. The **Ephemeral storage** is set to 512 MB. The **Execution role** is set to `service-role/cloudwatch_alarm_monitor_aws_lambda-role-jux0714`. The **Function overview** tab shows the function name, layers, and a description.

•

The screenshot shows the **Edit basic settings** page for the function `cloudwatch_alarm_monitor_aws_lambda`. The **Basic settings** tab is selected. The **Memory** is set to 128 MB. The **Ephemeral storage** is set to 512 MB. The **SnapStart** is set to `None`. The **Timeout** is set to 5 min 0 sec. The **Execution role** is set to `service-role/cloudwatch_alarm_monitor_aws_lambda-role-jux0714`. The **Function overview** tab shows the function name, layers, and a description.

•

- Copy the sample code from “Cloudwatch\_alarm\_dataCSV\_to\_mail\_s3.py” file and paste it in “code” section into `lambda_function.py` file and then click Deploy button.

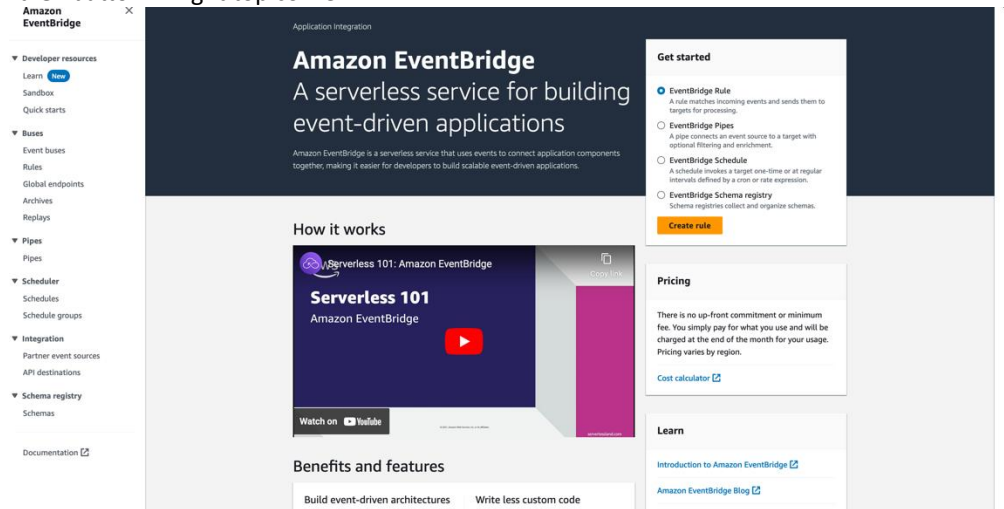
The screenshot shows the **Code source** tab for the function `cloudwatch_alarm_monitor_aws_lambda`. The **Code source** tab is selected. The **Code source** section shows the code for `lambda_function.py`. The code is as follows:

```
1 import json
2 import boto3
3 import datetime
4 from datetime import datetime
5 import time, traceback
6 import pandas as pd
7 import os.path
8 import email
9 from email.mime.multipart import MIMEMultipart
10 from email.mime.text import MIMEText
11 from email.mime.application import MIMEApplication
12
13 def lambda_handler(event, context):
14     try:
15         def convertToMili(value):
16             dt_obj = datetime.strptime(str(value), '%Y-%m-%d %H:%M:%S')
17             result = int(dt_obj.timestamp() * 1000)
18             return result
19     except Exception as e:
20         print(e)
```

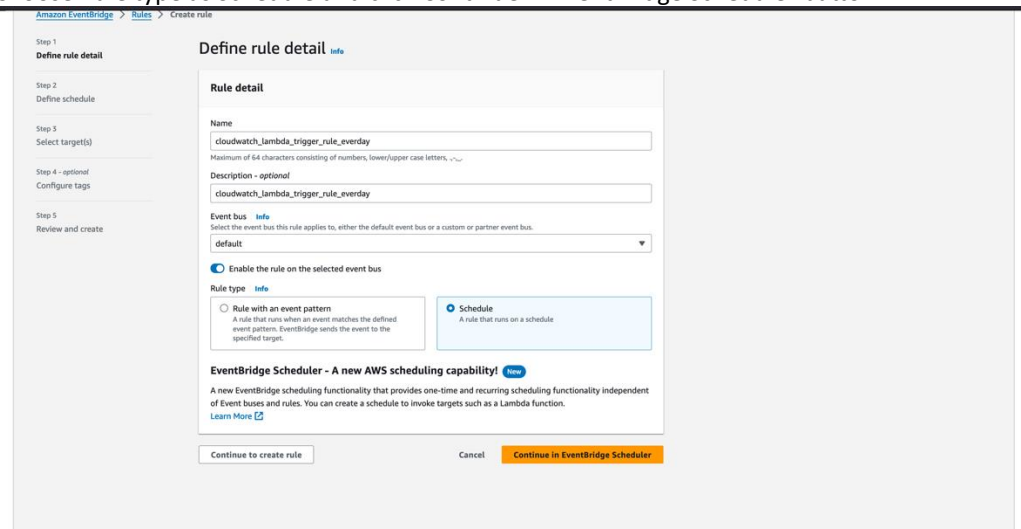
•

**Step 4: Create a Amazon Event Bridge rule to trigger the AWS Lambda function created in step:2, everyday at a specific time of a day.**

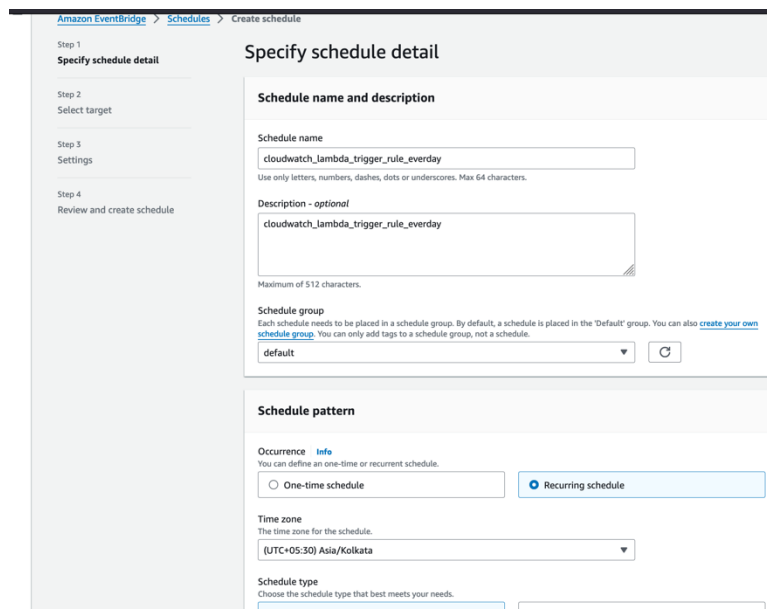
- [https://docs.aws.amazon.com/scheduler/latest/UserGuide/schedule-types.html?icmpid=docs\\_console\\_unmapped#cron-based](https://docs.aws.amazon.com/scheduler/latest/UserGuide/schedule-types.html?icmpid=docs_console_unmapped#cron-based)
- Navigate to Amazon Event bridge console and click **'Create rule'** button in the Amazon Event bridge home page or from the left panel click "Rules" and then Click **"Create rule"** button in right top corner.



- Choose Rule type as Schedule and click Continue in EventBridge Scheduler button.



- Enter the schedule name, description and choose radio button Recurring Schedule.





- Under Schedule type, choose Cron-based schedule. Enter the Cron expression, The below screenshot shows cron expression to run this schedule at everyday 2.30 a.m. Choose Flexible time window as 5 minutes and click Next.

**Cron-based schedule**  
A schedule set using a cron expression that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.

**Rate-based schedule**  
A schedule that runs at a regular rate, such as every 10 minutes.

**Cron expression** Info  
Define the cron expression for the schedule

**cron** ( 30 2 \* \* ? \* )  
Minutes Hours Day of month Month Day of the week Year

**Next 10 trigger dates**  
Date and time are displayed in your current time zone in UTC format, e.g. "Wed, Nov 9, 2022 09:00 [UTC -08:00] for Pacific time"

Mon, 08 Apr 2024 02:30:00 (UTC+05:30)  
Tue, 09 Apr 2024 02:30:00 (UTC+05:30)  
Wed, 10 Apr 2024 02:30:00 (UTC+05:30)  
Thu, 11 Apr 2024 02:30:00 (UTC+05:30)  
Fri, 12 Apr 2024 02:30:00 (UTC+05:30)  
Sat, 13 Apr 2024 02:30:00 (UTC+05:30)  
Sun, 14 Apr 2024 02:30:00 (UTC+05:30)  
Mon, 15 Apr 2024 02:30:00 (UTC+05:30)  
Tue, 16 Apr 2024 02:30:00 (UTC+05:30)  
Wed, 17 Apr 2024 02:30:00 (UTC+05:30)

**Flexible time window**  
If you choose a flexible time window, Scheduler invokes your schedule within the time window you specify. For example, if you choose 15 minutes, your schedule runs within 15 minutes after the schedule start time.

5 minutes

**Timeframe**

**Daylight saving time**  
Amazon EventBridge Scheduler automatically adjusts your schedule for daylight saving time. When time shifts forward in the Spring, if a cron expression falls on a non-existent date, your schedule invocation is skipped. When time shifts backwards in the Fall, your schedule runs only once and does not repeat its invocation. The following invocations occur normally at the specified date and time.

- Choose AWS Lambda radio button and select the lambda function which we created in step 3 and click next.

**Select target**

Step 3 - optional  
[Settings](#)

Step 4  
[Review and create schedule](#)

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

**Templated 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

**Invoke** ☒ Universal target definition  
AWS Lambda

**Lambda function**  
cloudwatch\_alarm\_monitor\_aws\_lambda

**Configure version/alias**

**Payload**

- Choose NONE from Action after schedule completion dropdown and click Next and in Review and create schedule page click the review and create.

**Select target**

Step 3 - optional  
[Settings](#)

Step 4  
[Review and create schedule](#)

**Enable schedule**  
You can choose not to enable the schedule now. You will be able to enable the schedule after it has been created.  
☒ Enable

**Action after schedule completion** Info  
If you choose DELETE, EventBridge Scheduler will automatically delete the schedule after it has completed its last invocation and has no future target invocations planned.

NONE

**Retry policy and dead-letter queue (DLQ)**

**Retry policy** Info  
By default, EventBridge Scheduler attempts to retry failed invocations for up to 24 hours. You can specify the maximum age of the event and the maximum number of times to retry.

☒ Retry

**Maximum age of event - optional**  
The maximum amount of time to keep unprocessed events. The maximum and default value is 24 hours.

24 hour(s) 0 minute(s)

**Retry attempts - optional**  
The maximum number of times to retry when a target returns an error. The maximum value is 185 times.

185 times

**Dead-letter queue (DLQ)**  
Standard Amazon SQS queues that EventBridge Scheduler uses to store events that couldn't be delivered successfully to a target.

☒ None  
☐ Select an Amazon SQS queue in my AWS account as a DLQ  
☐ Specify an Amazon SQS queue in other AWS accounts as a DLQ



Step 2 - optional

[Select target](#)

Step 3 - optional

[Settings](#)

Step 4

**Review and create schedule**

Step 1: Schedule detail

Edit

Schedule detail

Schedule name cloudwatch_lambda_trigger_rule_e verday	Description cloudwatch_lambda_trigger_rule_e verday	Schedule group default
Time zone (UTC+05:30) Asia/Kolkata	Occurrence Recurring	Start date and time -
End date and time -	Flexible time window 5 minutes	

Cron expression

30	2	*	*	?	*
Minutes	Hours	Day of month	Month	Day of week	Year

Next 10 trigger dates

Date and time are displayed in the selected time zone for which this schedule is set in UTC format, e.g. "Wed, Nov 9, 2022 09:00 (UTC - 08:00)"

Mon, 08 Apr 2024 02:30:00 (UTC+05:30)  
Tue, 09 Apr 2024 02:30:00 (UTC+05:30)  
Wed, 10 Apr 2024 02:30:00 (UTC+05:30)  
Thu, 11 Apr 2024 02:30:00 (UTC+05:30)  
Fri, 12 Apr 2024 02:30:00 (UTC+05:30)  
Sat, 13 Apr 2024 02:30:00 (UTC+05:30)  
Sun, 14 Apr 2024 02:30:00 (UTC+05:30)  
Mon, 15 Apr 2024 02:30:00 (UTC+05:30)  
Tue, 16 Apr 2024 02:30:00 (UTC+05:30)  
Wed, 17 Apr 2024 02:30:00 (UTC+05:30)

Step 2: Target

Edit

Target detail