14.08.2024

Overview

This document outlines a comprehensive guide on how to leverage AWS Lambda to create a serverless solution for starting and stopping EC2 instances. It aims to provide step-by-step instructions, code examples, and best practices to efficiently automate server management tasks..

Objectives

This document will cover the following key areas:

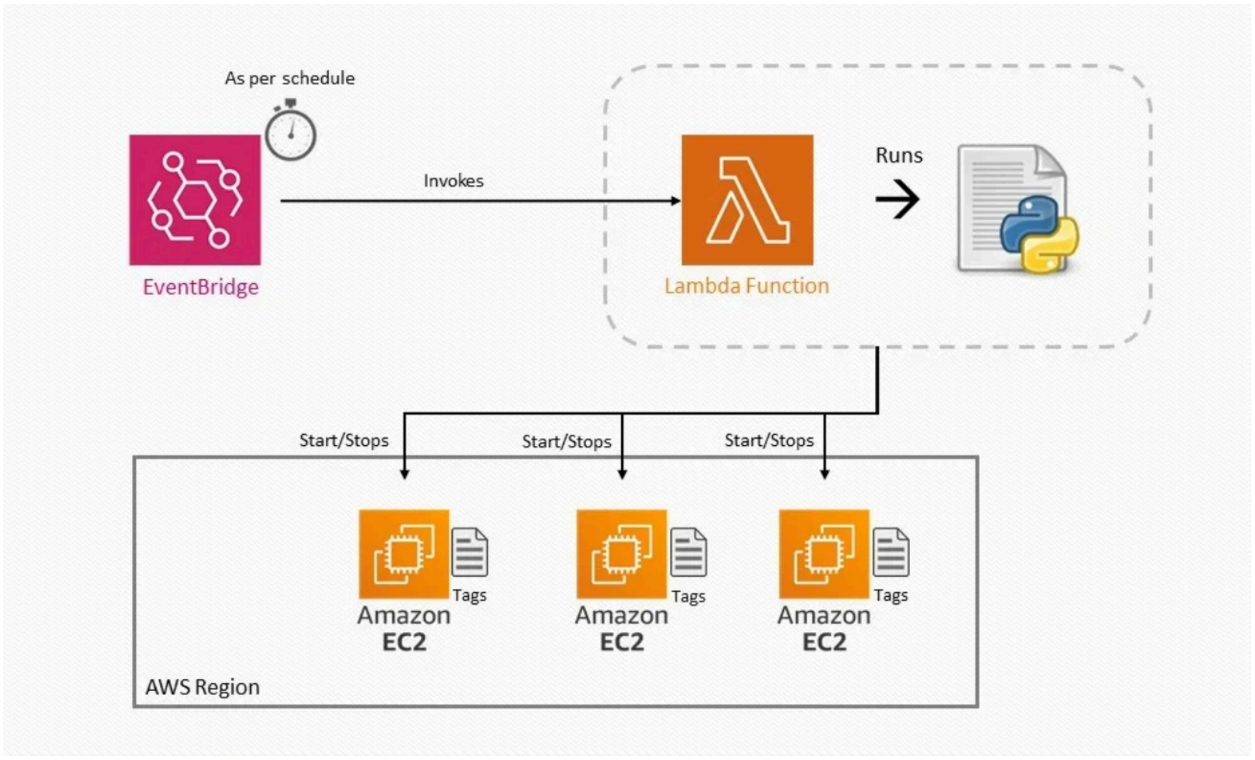
- **Understanding the Problem:** Discussing the challenges of manually starting and stopping servers and the benefits of automation.
- **Lambda Function Architecture:** Explaining the core components and design considerations for building the Lambda functions.
- **Triggering the Functions:** Exploring various methods to initiate server actions, such as API Gateway, CloudWatch Events, or custom triggers.
- **Security and Permissions:** Addressing security best practices and IAM role configuration for Lambda functions.
- **Error Handling and Logging:** Implementing robust error handling mechanisms and utilizing CloudWatch Logs for monitoring.
- **Deployment and Testing:** Providing guidance on deploying the Lambda functions and conducting thorough testing.
- **Best Practices and Optimization:** Sharing recommendations for optimizing performance, cost, and reliability.

Goal

To provide a comprehensive guide and reference for creating and implementing a serverless solution to automate EC2 instance start and stop operations using AWS Lambda.

Overview

Let’s get an overview of the whole procedure :

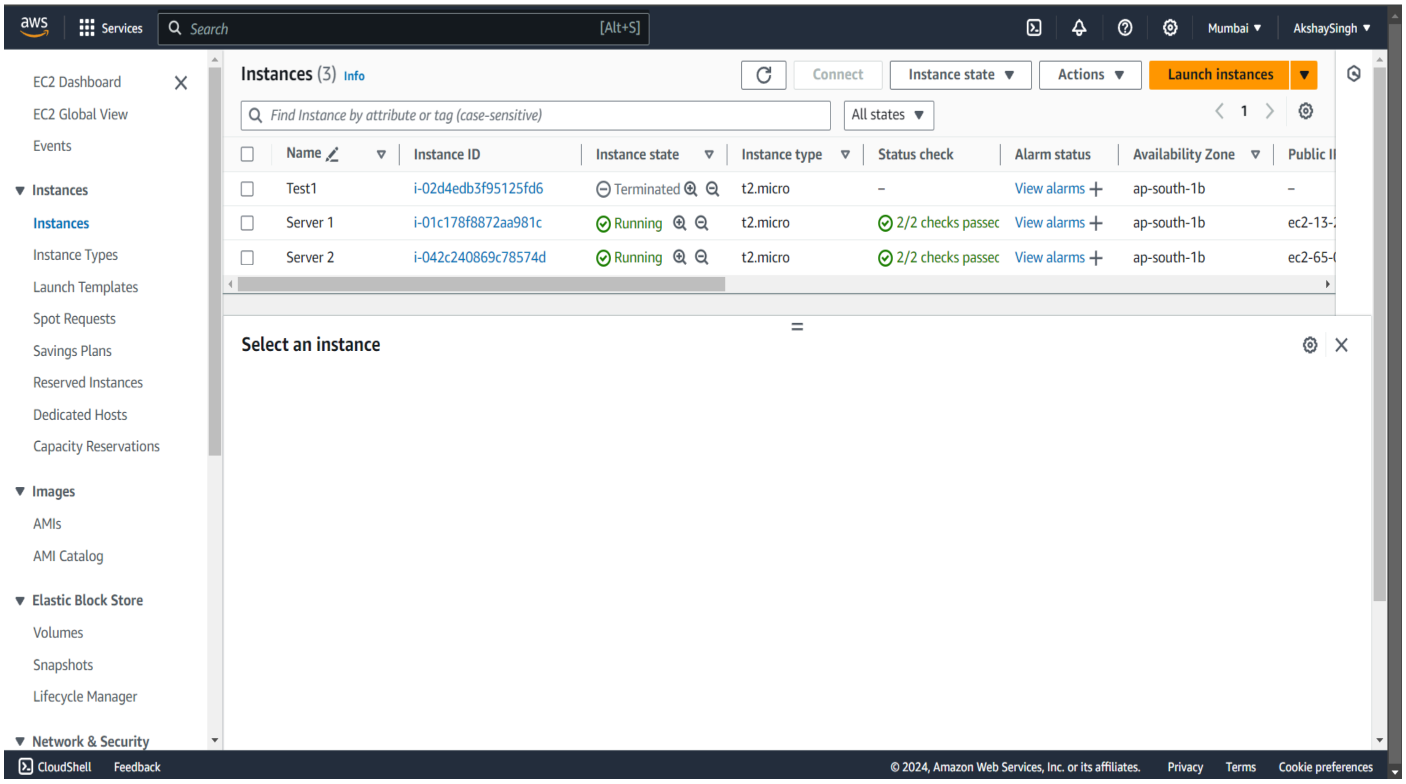


Steps involved

Step 1: Create Two EC2 Instances

1. **Log in to AWS Management Console**:
   - Navigate to AWS Management Console.
   - Log in with your credentials.
2. **Navigate to EC2 Dashboard**:
   - In the AWS Management Console, search for "EC2" in the search bar and click on "EC2" to go to the EC2 Dashboard.

3. **Launch an EC2 Instance**:
    - Click on the "Launch Instances" button.
    - **Choose an Amazon Machine Image (AMI)**: Select an appropriate AMI (e.g., Amazon Linux 2).
    - **Choose an Instance Type**: Select an instance type (e.g., `t2.micro`).
    - **Configure Instance Details**: You can leave the default settings or modify them according to your needs.
    - **Add Storage**: Leave the default settings or modify the storage as needed.
    - **Add Tags**: Add tags for easy identification (e.g., Name: `Server1`).
    - **Configure Security Group**: Create a new security group or select an existing one.
    - **Review and Launch**: Review your settings, then click "Launch".
    - **Select or Create a Key Pair**: Choose an existing key pair or create a new one, then click "Launch Instances".
4. **Repeat the Above Steps to Launch a Second Instance**:
    - Create a second EC2 instance with the same or different configuration and name it `Server2`.
5. **Note the Instance IDs**:
    - After the instances are launched, note down their Instance IDs. You will use

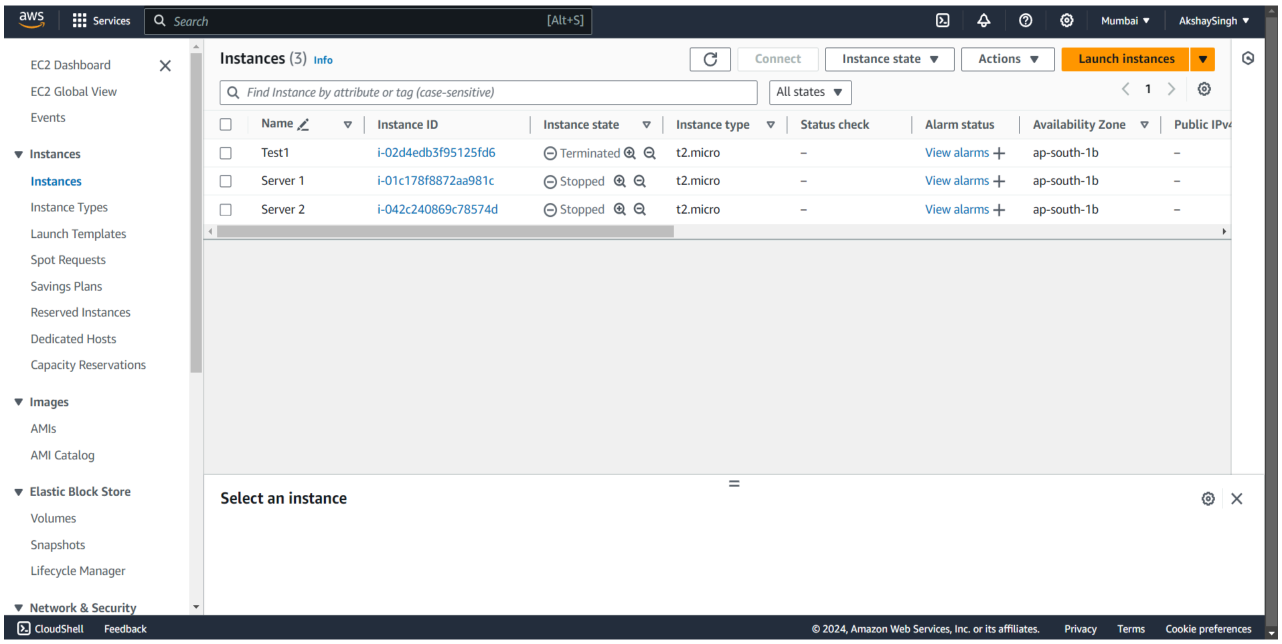


Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public I
Test1	i-02d4edb3f95125fd6	Terminated	t2.micro	–	View alarms	ap-south-1b	–
Server 1	i-01c178f8872aa981c	Running	t2.micro	2/2 checks passec	View alarms	ap-south-1b	ec2-13-
Server 2	i-042c240869c78574d	Running	t2.micro	2/2 checks passec	View alarms	ap-south-1b	ec2-65-

these IDs in the Lambda functions.

Step 2: Manually Stop the EC2 Instances

1. **Navigate to the EC2 Dashboard**:
   - Go to the "Instances" section in the EC2 Dashboard.
2. **Stop the Instances**:
   - Select both instances (`Server1` and `Server2`).
   - Click on "Instance State" and select "Stop Instance".
   - Wait for the instances to stop.

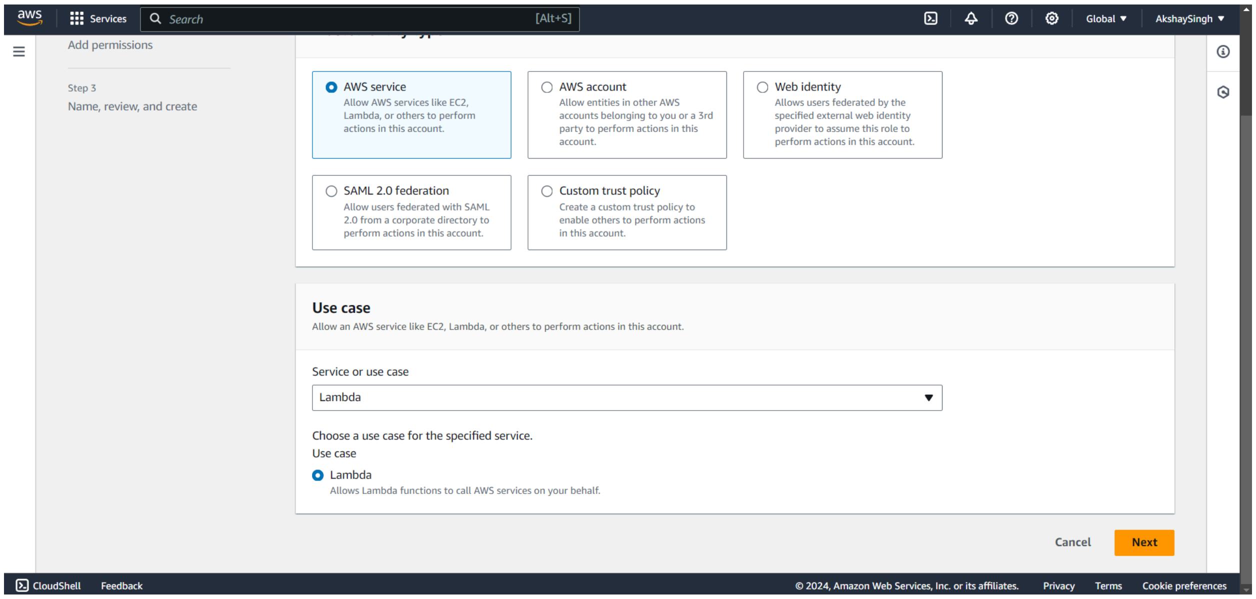


Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv
Test1	i-02d4edb3f95125fd6	Terminated	t2.micro	–	View alarms +	ap-south-1b	–
Server 1	i-01c178f8872aa981c	Stopped	t2.micro	–	View alarms +	ap-south-1b	–
Server 2	i-042c240869c78574d	Stopped	t2.micro	–	View alarms +	ap-south-1b	–

Step 3: Create a Lambda Role with Full EC2 Access

1. **Navigate to IAM Dashboard**:
   - In the AWS Management Console, search for "IAM" and click on it to open the IAM Dashboard.
2. **Create a New Role**:
   - Click on "Roles" in the left sidebar, then click "Create Role".
   - **Select Trusted Entity**: Choose "AWS Service" and select "Lambda" as the service that will use this role.
   - **Attach Policies**: Search for and select the `AmazonEC2FullAccess` policy.

- **Add Tags**: Optionally, add tags for easier management.
- **Review and Create**: Review the role details and click "Create Role".
- **Name the Role**: Give the role a name (e.g., `LambdaEC2FullAccess`).

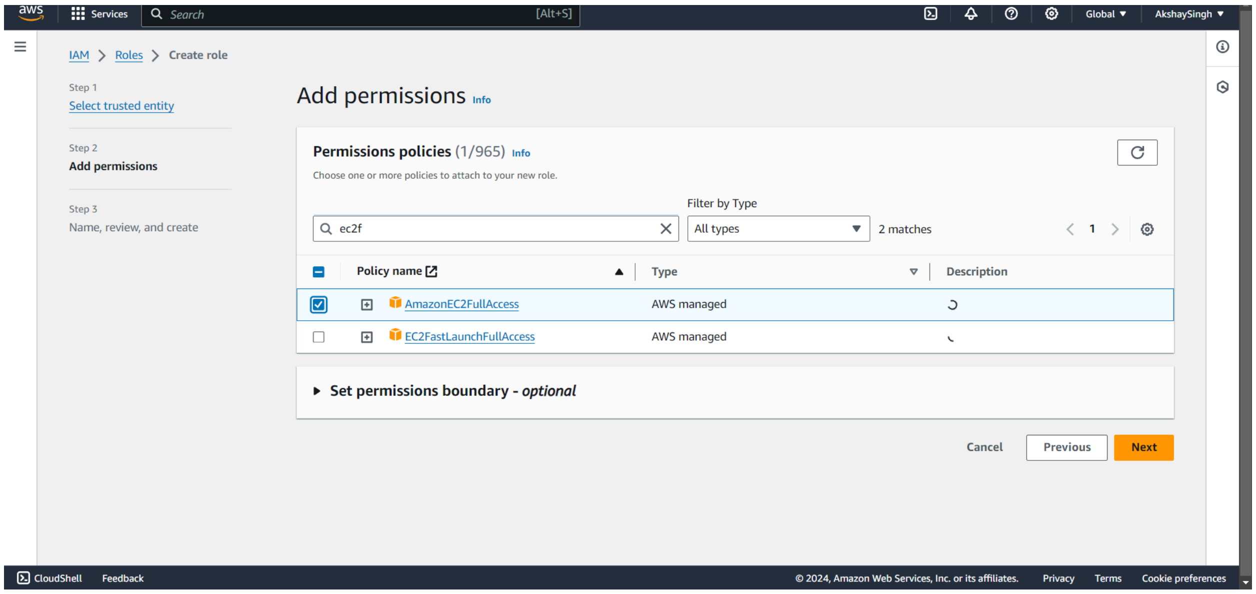


AWS service — Allow AWS services like EC2, Lambda, or others to perform actions in this account.
AWS account — Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
Web identity — Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
SAML 2.0 federation — Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
Custom trust policy — Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case: Lambda

Choose a use case for the specified service.
Use case
Lambda — Allows Lambda functions to call AWS services on your behalf.



Add permissions Info

Permissions policies (1/965) Info
Choose one or more policies to attach to your new role.

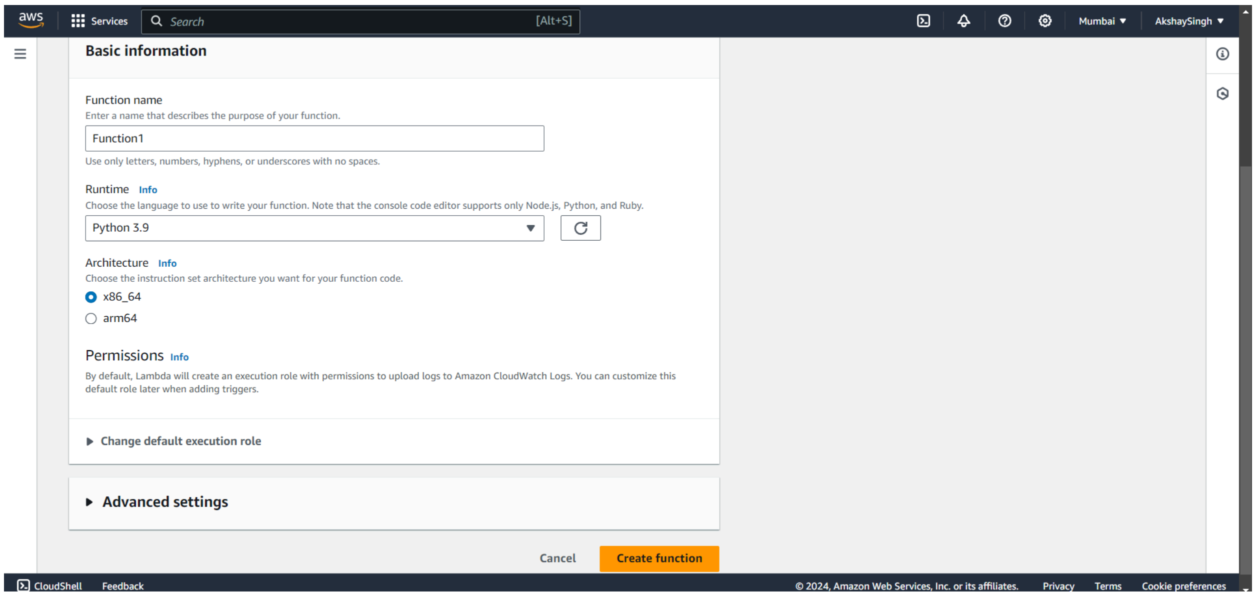
Filter by Type: All types
Search: ec2f — 2 matches

	Policy name	Type	Description
✓	AmazonEC2FullAccess	AWS managed	
	EC2FastLaunchFullAccess	AWS managed	

Set permissions boundary - *optional*

Step 4: Create Two Lambda Functions

1. **Navigate to Lambda Dashboard**:
   - In the AWS Management Console, search for "Lambda" and click on it to open the Lambda Dashboard.
2. **Create the First Lambda Function**:
   - Click "Create Function".
   - **Function Name**: Enter a name (e.g., `StartEC2Instance`).
   - **Runtime**: Select Python 3.9.
   - **Permissions**: Choose "Use an existing role" and select the role you created (`LambdaEC2FullAccess`).
   - Click "Create Function".



Basic information

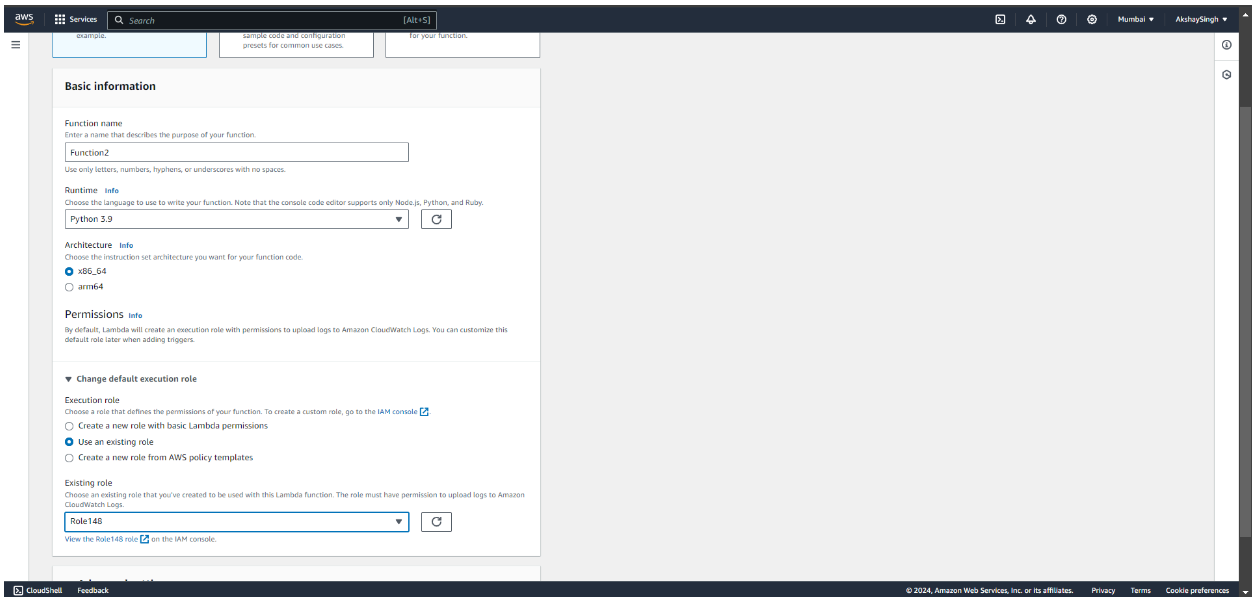
Function name
Enter a name that describes the purpose of your function.
Function1
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

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.

3. **Create the Second Lambda Function**:
   - Repeat the steps above to create a second function with the name `StopEC2Instance`.



example.

sample code and configuration presets for common use cases.

for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
Function2
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.
Python 3.9

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

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
[✓ Use an existing role]
Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
Role148
View the Role148 role on the IAM console.

Step 5: Add Start/Stop EC2 Code to Lambda Functions

1. **Search for AWS Lambda EC2 Start/Stop Code**:
   - In a new tab, search for the AWS Lambda Python code snippets to start and stop EC2 instances

```
import boto3
region = 'us-west-1'
instances = ['i-12345cb6de4f78g9h', 'i-08ce9b2d7eccf6d26']
ec2 = boto3.client('ec2', region_name=region)

def lambda_handler(event, context):
    ec2.stop_instances(InstanceIds=instances)
    print('stopped your instances: ' + str(instances))
```

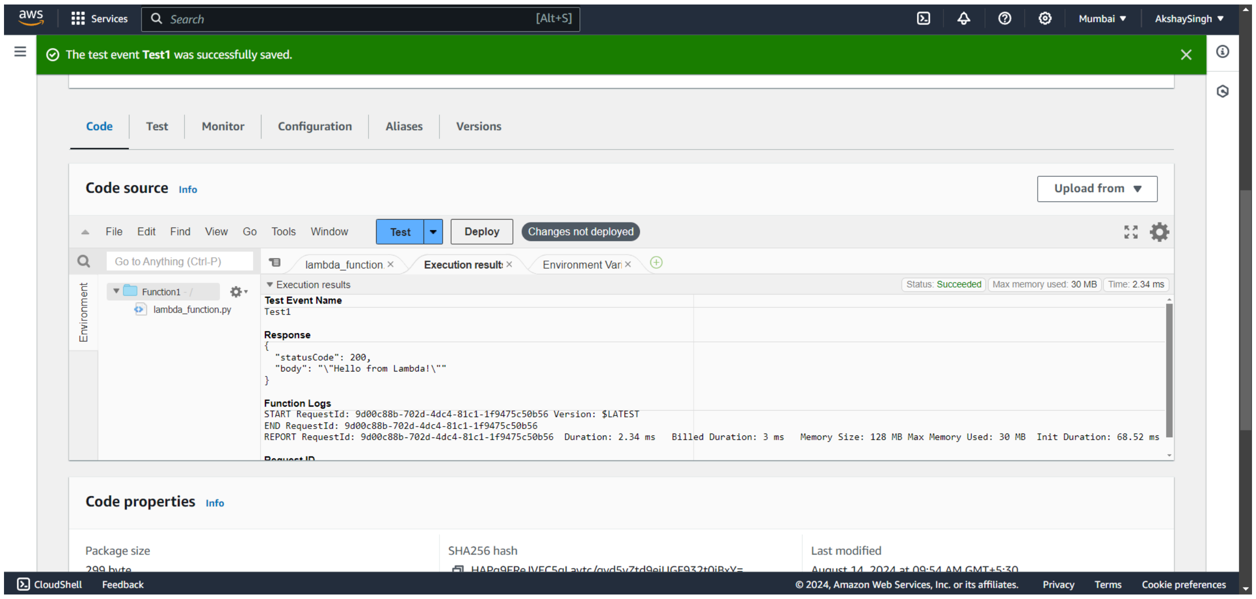
```
import boto3
region = 'us-west-1'
instances = ['i-12345cb6de4f78g9h', 'i-08ce9b2d7eccf6d26']
ec2 = boto3.client('ec2', region_name=region)

def lambda_handler(event, context):
    ec2.start_instances(InstanceIds=instances)
    print('started your instances: ' + str(instances))
```

2. **Edit the `StartEC2Instance` Function**:
    - Go to the Lambda Dashboard and select the `StartEC2Instance` function.
    - Scroll down to the "Code source" section and delete the default code.
    - Paste the start EC2 instance code snippet.
    - Replace the `InstanceId` value with the ID of `Server1` or `Server2`.
    - Click "Deploy" to save your changes.
3. **Edit the `StopEC2Instance` Function**:
    - Go to the Lambda Dashboard and select the `StopEC2Instance` function.
    - Scroll down to the "Code source" section and delete the default code.

- Paste the stop EC2 instance code snippet.
- Replace the `InstanceId` value with the ID of the other server.
- Click "Deploy" to save your changes.

Step 6: Test and Deploy the Functions

1. **Test the StartEC2Instance Function**:
    - Click "Test" on the Lambda function page.
    - Create a test event (you can use a default template).
    - Run the test to start the EC2 instance.



The test event **Test1** was successfully saved.

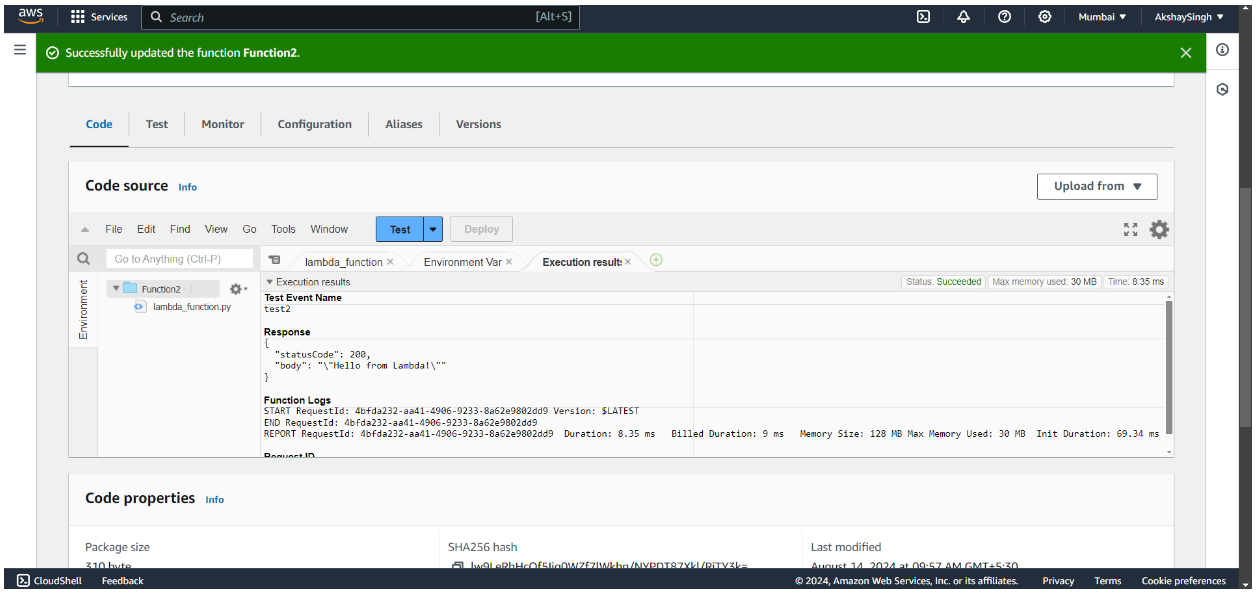
Execution results — Status: Succeeded | Max memory used: 30 MB | Time: 2.34 ms

Test Event Name
```
Test1
```

Response
```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Function Logs
```
START RequestId: 9d00c88b-702d-4dc4-81c1-1f9475c50b56 Version: $LATEST
END RequestId: 9d00c88b-702d-4dc4-81c1-1f9475c50b56
REPORT RequestId: 9d00c88b-702d-4dc4-81c1-1f9475c50b56  Duration: 2.34 ms   Billed Duration: 3 ms   Memory Size: 128 MB Max Memory Used: 30 MB  Init Duration: 68.52 ms
```

2. **Test the StopEC2Instance Function**:
   ○ Repeat the same steps as above to test the `StopEC2Instance` function.



Successfully updated the function **Function2**.

Code source Info

Execution results — Status: Succeeded | Max memory used: 30 MB | Time: 8.35 ms

Test Event Name
```
test2
```

Response
```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

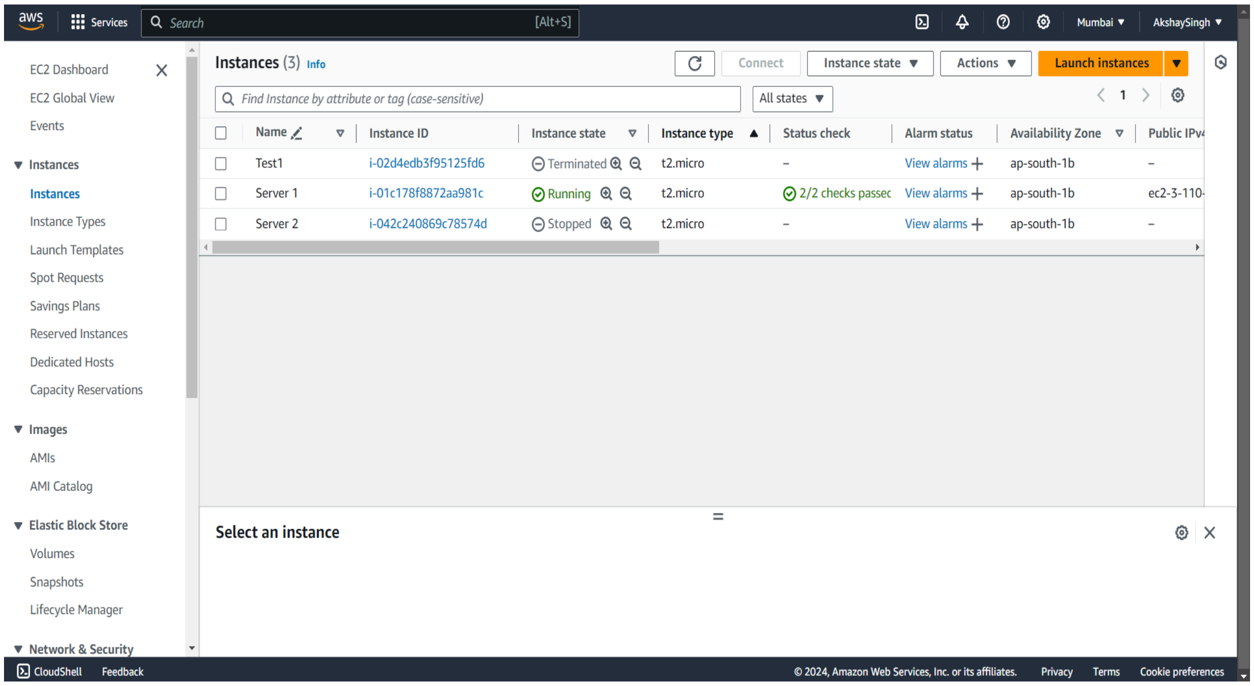
Function Logs
```
START RequestId: 4bfda232-aa41-4906-9233-8a62e9802dd9 Version: $LATEST
END RequestId: 4bfda232-aa41-4906-9233-8a62e9802dd9
REPORT RequestId: 4bfda232-aa41-4906-9233-8a62e9802dd9  Duration: 8.35 ms   Billed Duration: 9 ms   Memory Size: 128 MB Max Memory Used: 30 MB  Init Duration: 69.34 ms
```

Code properties Info

Package size: 310 byte
SHA256 hash: lw9LeRbHcOf5Iig0WZf7lWkbn/NYPDT87XkI/RiTY3k=
Last modified: August 14, 2024 at 09:57 AM GMT+5:30

© 2024, Amazon Web Services, Inc. or its affiliates.

3. **Deploy and Monitor**:
   ○ Ensure both functions are successfully deployed and are working as expected.



Instances (3) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
Test1	i-02d4edb3f95125fd6	Terminated	t2.micro	–	View alarms +	ap-south-1b	–
Server 1	i-01c178f8872aa981c	Running	t2.micro	2/2 checks passec	View alarms +	ap-south-1b	ec2-3-110-
Server 2	i-042c240869c78574d	Stopped	t2.micro	–	View alarms +	ap-south-1b	–

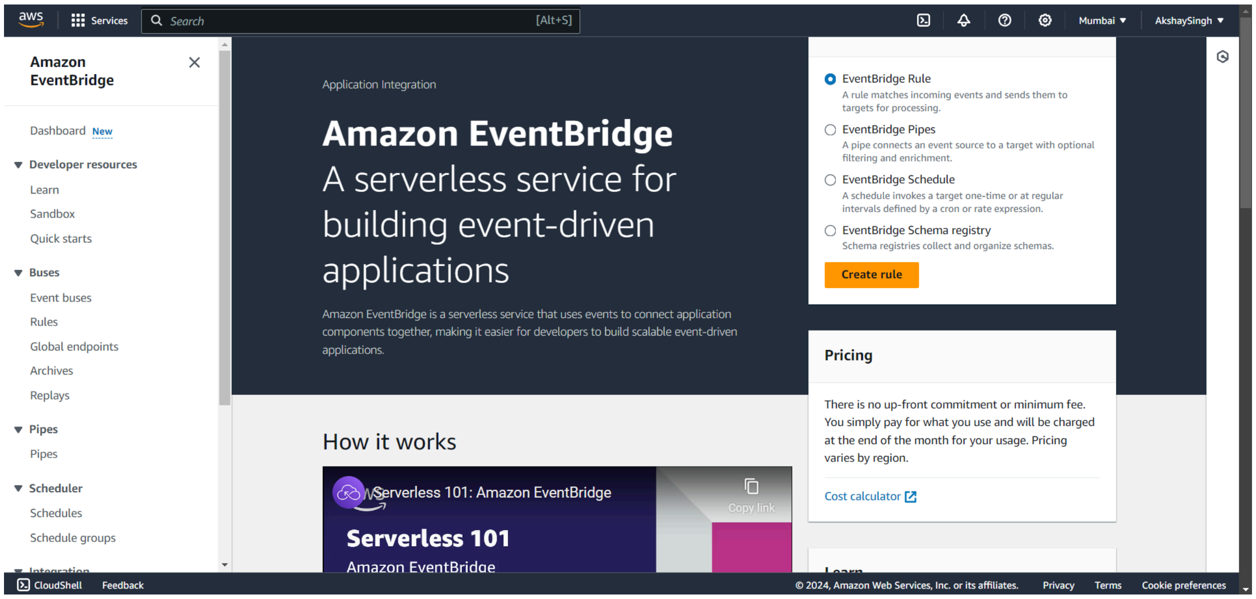
Select an instance

You now have two Lambda functions that can start and stop your EC2 instances based on the instance IDs you provided.

Now let us schedule our server using Event Bridge feature of aws

1. **Navigate to EventBridge**:
   - In the AWS Management Console, search for "EventBridge" and select it to open the EventBridge dashboard.
2. **Create a Rule**:
   - Click on "Rules" in the left sidebar.

- Click on the "Create rule" button.



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.

[✓ 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 invokes a target one-time or at regular intervals defined by a cron or rate expression.
EventBridge Schema registry: Schema registries collect and organize schemas.

How it works

Pricing

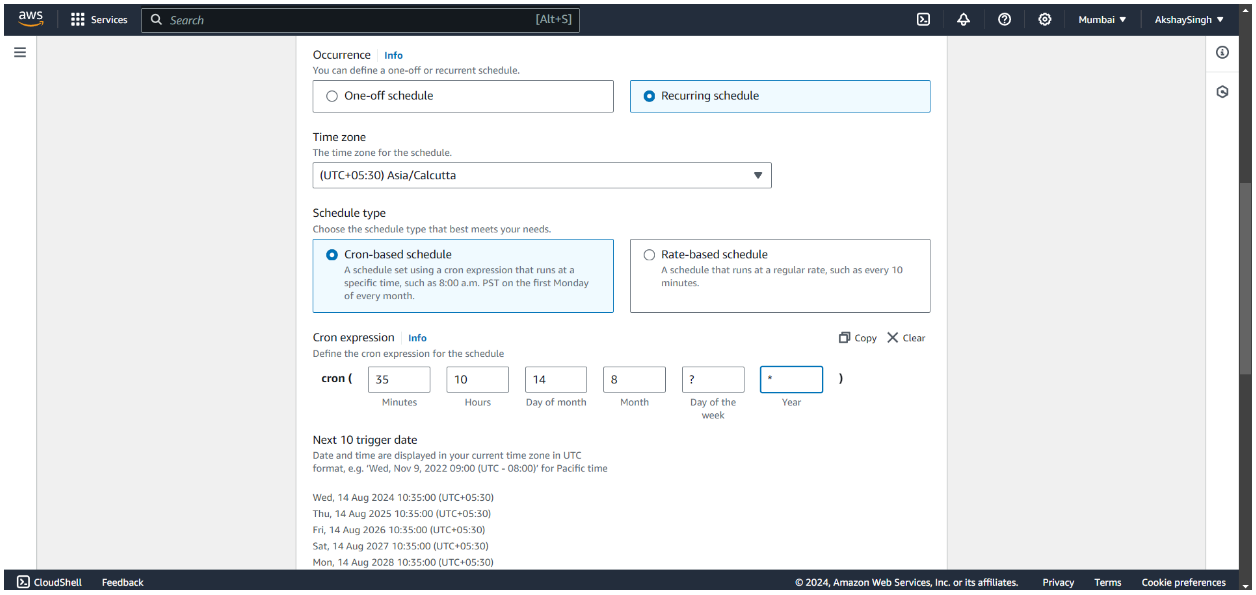
There is no up-front commitment or minimum fee. You simply pay for what you use and will be charged at the end of the month for your usage. Pricing varies by region.

© 2024, Amazon Web Services, Inc. or its affiliates.

3. **Configure Rule Details**:
   - **Name**: Enter a meaningful name for the rule (e.g., `StartEC2InstanceScheduler`).
   - **Description**: Optionally, add a description to describe the purpose of the rule.
   - **Event Bus**: Leave it as default.

4. **Set Schedule Pattern**:
   - **Rule Type**: Select "Schedule".
   - **Define the schedule**:
     - **Schedule expression**: Select "Recurring schedule".

- **Schedule details**:
  - **Minutes**: Enter the minute interval you want
  - **Hours**: Enter the hour you want the task to run
  - **Leave the rest as * to run daily.**



Occurrence | Info
You can define a one-off or recurrent schedule.
○ One-off schedule [✓ Recurring schedule]

Time zone
The time zone for the schedule.
(UTC+05:30) Asia/Calcutta

Schedule type
Choose the schedule type that best meets your needs.
[✓ 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 — Copy — Clear
Define the cron expression for the schedule

	Minutes	Hours	Day of month	Month	Day of the week	Year	
cron (	35	10	14	8	?	*	)

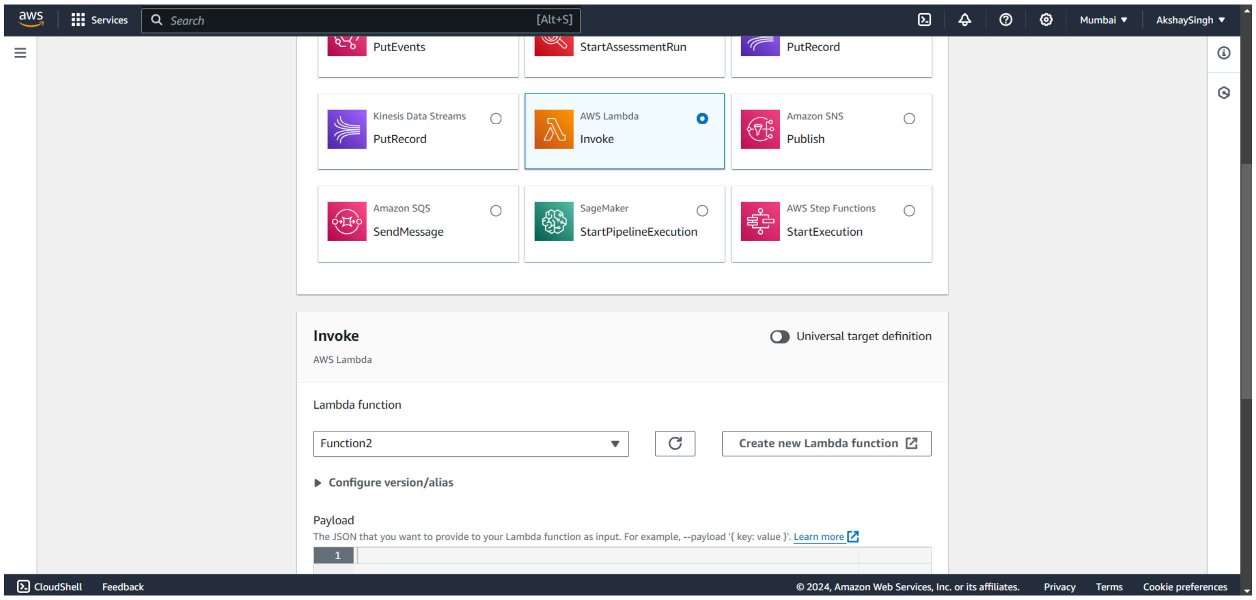
Next 10 trigger date
Date and time are displayed in your current time zone in UTC format, e.g. 'Wed, Nov 9, 2022 09:00 (UTC - 08:00)' for Pacific time

Wed, 14 Aug 2024 10:35:00 (UTC+05:30)
Thu, 14 Aug 2025 10:35:00 (UTC+05:30)
Fri, 14 Aug 2026 10:35:00 (UTC+05:30)
Sat, 14 Aug 2027 10:35:00 (UTC+05:30)
Mon, 14 Aug 2028 10:35:00 (UTC+05:30)

- **Flexible time window**: Select "5 minutes". This gives a 5-minute window for the action to occur.
- **Start date**: Optionally, you can set a start date. Leave other fields as they are if not needed.

5. **Configure Target**:
   - Click "Next" to configure the target.
   - **Target type**: Select "AWS service".
   - **Service**: Choose "Lambda function".

- **Function**: Select the Lambda function you created earlier (`StartEC2Instance`).

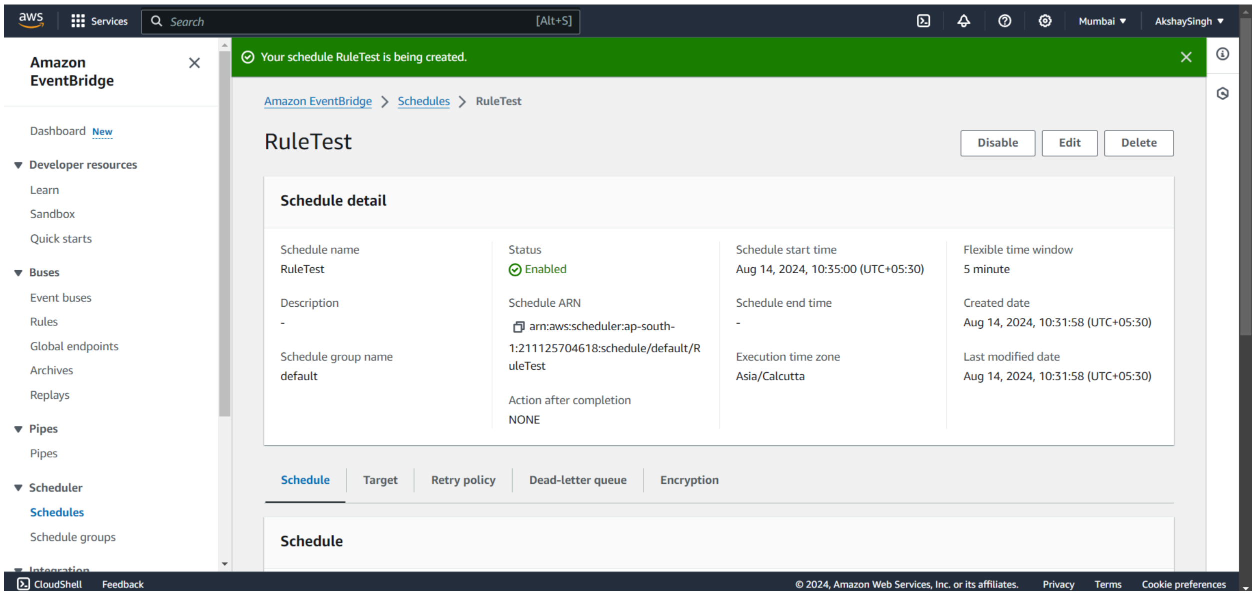


Invoke
AWS Lambda

Lambda function
Function2

Payload
The JSON that you want to provide to your Lambda function as input. For example, --payload '{ key: value }'. Learn more

6. **Review and Create**:
   - Click "Next" to review your rule settings.
   - Confirm everything is correct, then click "Create rule".



Your schedule RuleTest is being created.

RuleTest

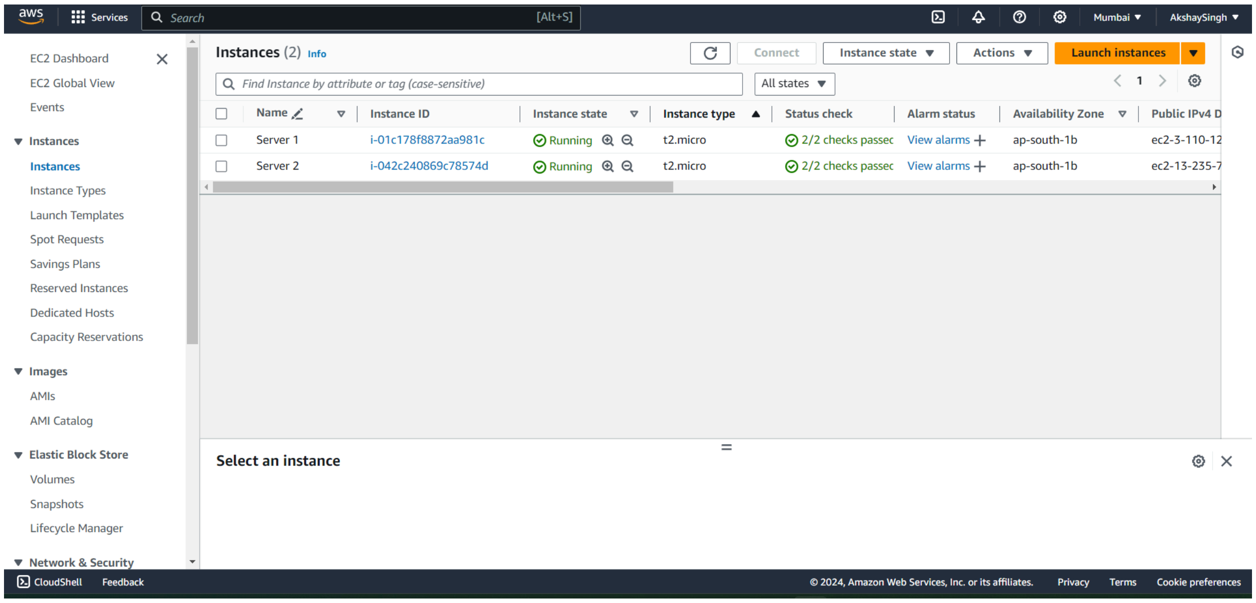
Schedule detail

Schedule name RuleTest	Status Enabled	Schedule start time Aug 14, 2024, 10:35:00 (UTC+05:30)	Flexible time window 5 minute
Description -	Schedule ARN arn:aws:scheduler:ap-south-1:211125704618:schedule/default/RuleTest	Schedule end time -	Created date Aug 14, 2024, 10:31:58 (UTC+05:30)
Schedule group name default		Execution time zone Asia/Calcutta	Last modified date Aug 14, 2024, 10:31:58 (UTC+05:30)
	Action after completion NONE		

Schedule

Step 3: Verify the Scheduled Invocation

- **Wait for the Scheduled Time**:
  - After the scheduled time has passed, check the status of the EC2 instance that was stopped (`Server2` in our case).
  - It should now be in a **running state**, triggered by the EventBridge rule you created.



Instances (2) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 D
Server 1	i-01c178f8872aa981c	Running	t2.micro	2/2 checks passec	View alarms	ap-south-1b	ec2-3-110-12
Server 2	i-042c240869c78574d	Running	t2.micro	2/2 checks passec	View alarms	ap-south-1b	ec2-13-235-7

Summary

This guide outlines the process of managing Amazon EC2 instances through AWS Lambda functions and automating their invocation using EventBridge. Initially, two EC2 instances were created and manually stopped. Subsequently, a Lambda role with full EC2 access was established, followed by the creation of two Lambda functions using Python 3.9 runtime. These functions were configured to use the existing role, with one function dedicated to starting an EC2 instance and the other to stopping it.

After sourcing the necessary AWS Lambda code for starting and stopping EC2 instances, the code was integrated into the respective Lambda functions. The specific instance IDs of the created EC2 servers were inserted into the functions, which were then saved, tested, and deployed.

To automate the invocation of the EC2 instances at regular intervals, Amazon EventBridge was utilized. A rule was created with a recurring schedule, specifying the timing parameters and selecting the Lambda function for invocation. As a result, the stopped EC2 instance was automatically started according to the defined schedule, demonstrating a successful implementation of scheduled EC2 instance management.