

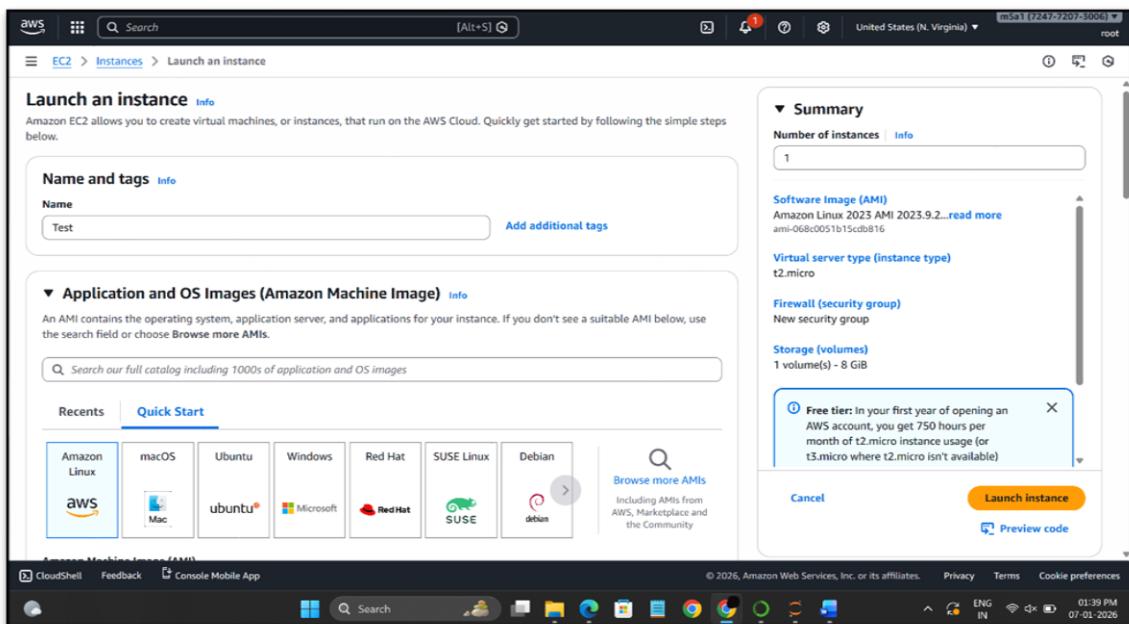
To Stop and Start Amazon EC2 Instances at regular time intervals using Lambda.

Requirements:

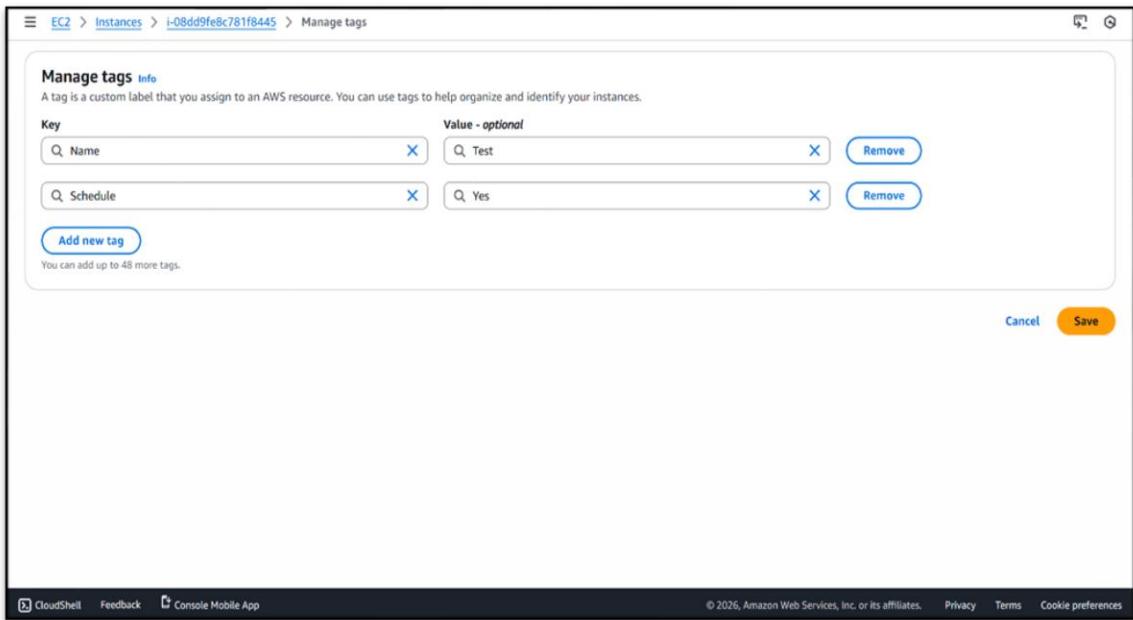
- Reduce EC2 Usage by scheduling start and stop automatically.
- Create IAM policy and execution role for Lambda function
- Create Lambda function to start and stop EC2 instances.
- Create EventBridge to rule Lambda function.

EC2:

- Open the AWS console → Navigate to EC2 → Launch instance → Name → Key pair → Allow SSH and HTTP in security group → Launch instance.



- After creation of instance select the instance → Navigate to tags → Manage tags → Add tags → Key='Schedule', Value='Yes' → Save. (This is done to help Lambda identify the instance)



IAM:

- Navigate to IAM policies → Create policy → JSON → Add the required permissions:

- EC2
- Logs

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "Statement1",
6       "Effect": "Allow",
7       "Action": [
8         "ec2:DescribeInstances",
9         "ec2:StopInstances",
10        "ec2:StartInstances",
11        "logs:CreateLogGroup",
12        "logs:CreateLogStream",
13        "logs:PutLogEvents"
14      ],
15      "Resource": "*"
16    }
17  ]
18 }

```

- Review and create: Policy name → "EC2LambdaPolicy" → Create policy.

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	All resources	None
EC2	Limited: List, Write	All resources	None

Add tags - optional Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Previous](#) [Create policy](#)

- Navigate to Roles → Create Execution role → Select Lambda as service → Attach EC2 Lambda Policy → Next → Name: "Lambda EC2Role" → Create role.

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
EC2LambdaPolicy	Customer managed	Permissions policy

Step 3: Add tags

Add tags - optional Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

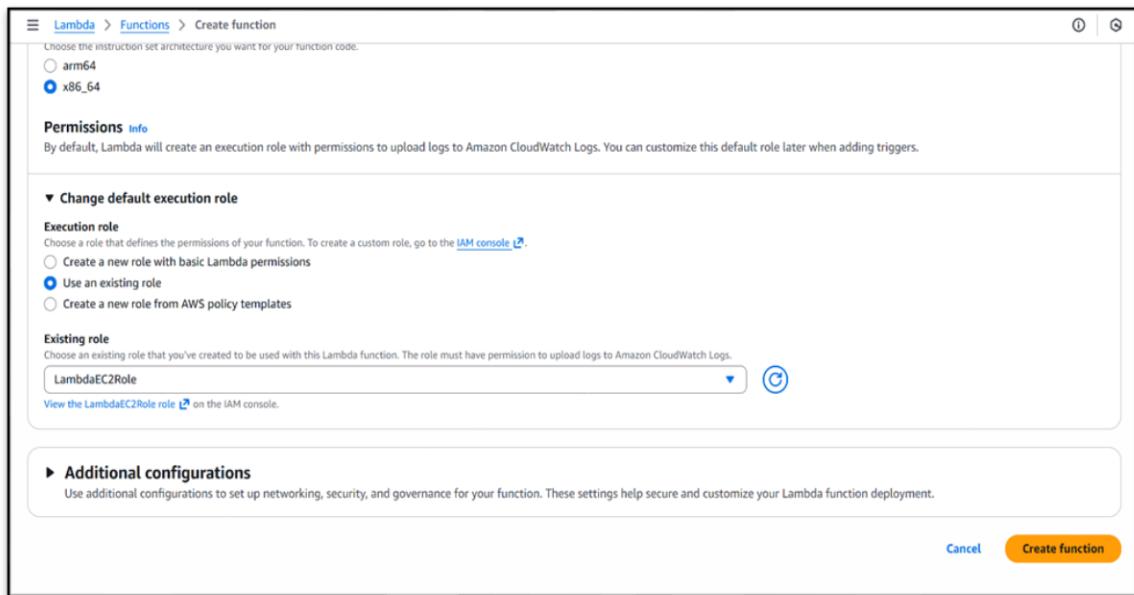
[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Previous](#) [Create role](#)

Lambda:

- Navigate to Lambda Functions → Create function → Function name → Runtime(Python 3.12) → Use the existing Lambda EC2 Role → Create function.

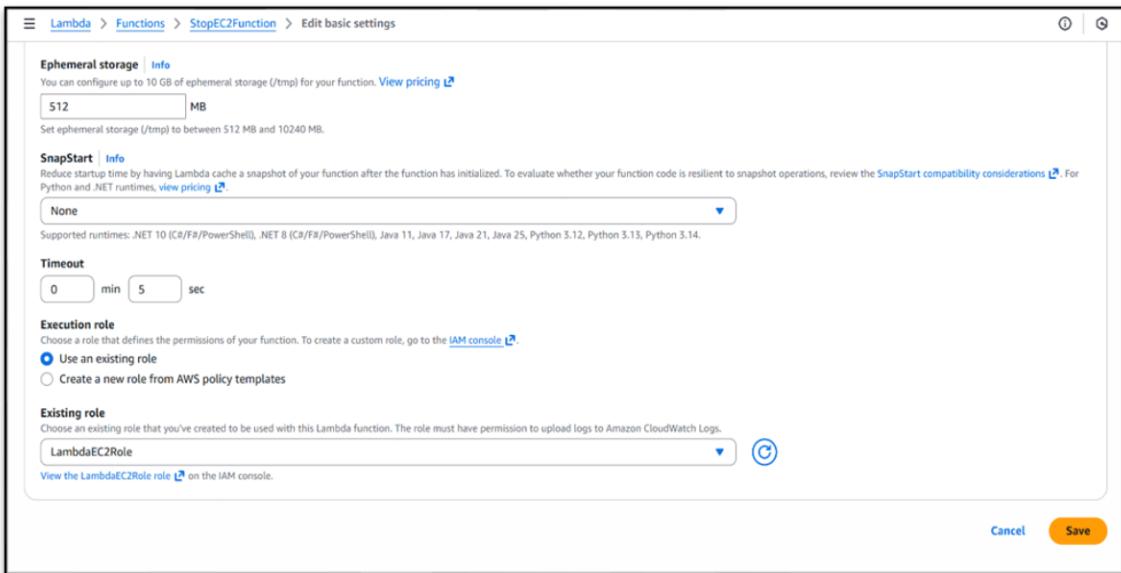


- Provide necessary code for the **Lambda Stop Function** → Deploy.

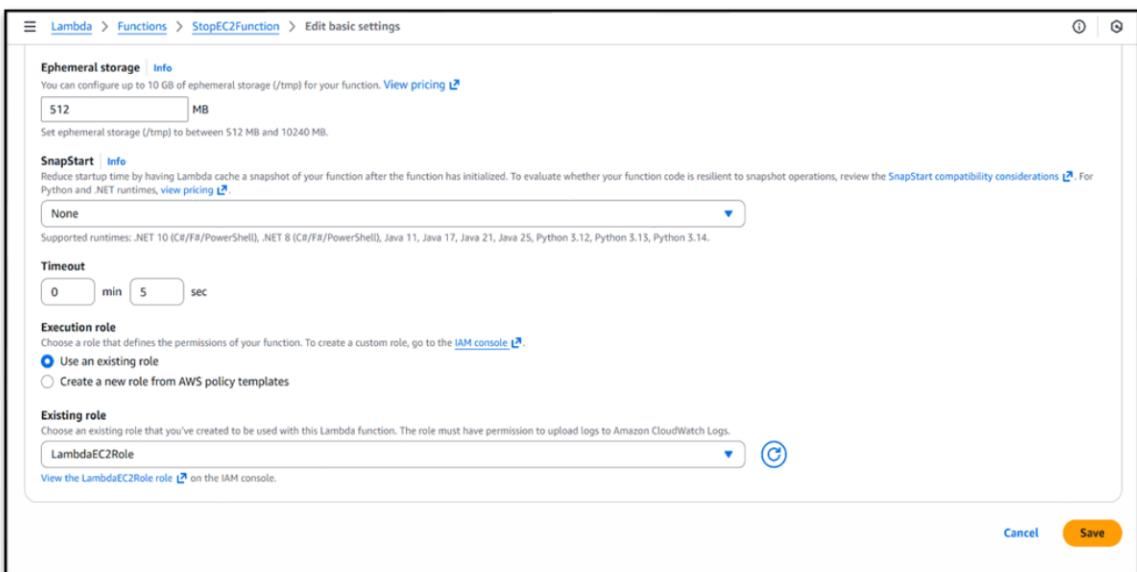
```

lambda_function.py
1 import json
2
3 import boto3
4 def lambda_handler(event, context):
5     ec2 = boto3.client('ec2', region_name='us-east-1')
6     response = ec2.describe_instances(Filters=[{'Name': 'tag:Schedule', 'Values': ['Yes']}])
7     instance_ids = []
8     for reservation in response['Reservations']:
9         for instance in reservation['Instances']:
10             if instance['State']['Name'] == 'running':
11                 instance_ids.append(instance['InstanceId'])
12     if instance_ids:
13         ec2.stop_instances(InstanceIds=instance_ids)
14     return f'Stopped: {instance_ids}'
15
16
17
    
```

- Configure the code by navigating to configuration → Enter the timeout seconds → Save.



- Test the Lambda function by navigating to Test → Proceed with empty JSON code → Save → Test.



- You can find that the instance has been stopped and the stop function has been executed successfully.

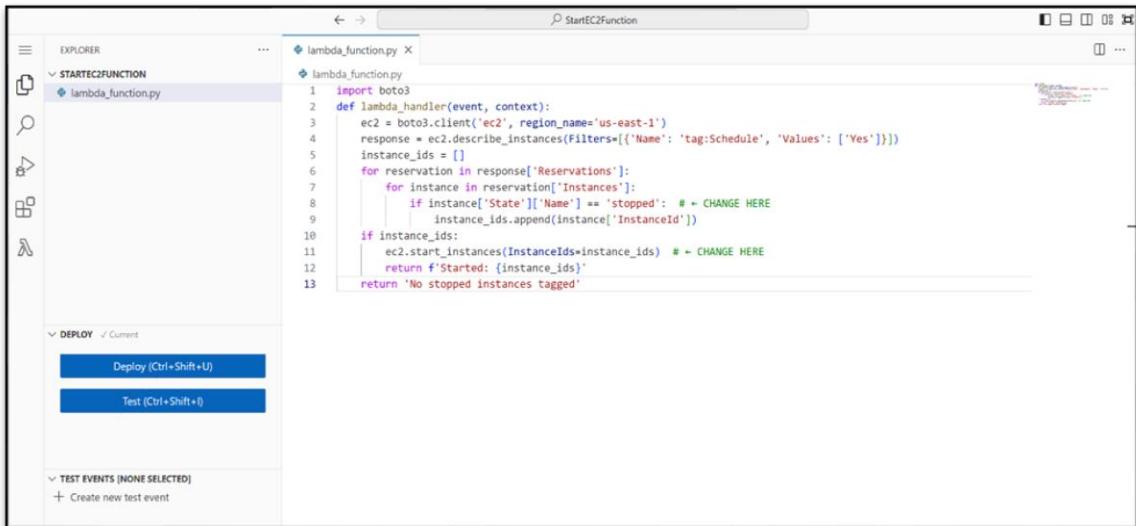
The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links like Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager (New), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes). The main area displays 'Instances (1/1) Info' with a table. The table has columns for Name (Test), Instance ID (i-08dd9fe8c781f8445), Instance state (Stopped), Instance type (t2.micro), Status check (-), Alarm status (-), and Availability Zone (us-east-1d). Below the table, a detailed view for instance i-08dd9fe8c781f8445 (Test) is shown with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under Details, there's an 'Instance summary' section with fields for Instance ID (i-08dd9fe8c781f8445), Public IPv4 address (-), Private IPv4 addresses (172.31.26.160), IPv6 address (-), Instance state (Stopped), and Public DNS (-).

The screenshot shows the AWS Lambda Functions page. The path is Lambda > Functions > StopECFunction. The main area displays the execution history for the function 'StopECFunction'. The most recent execution is listed as 'Executing function: succeeded (logs)' with a timestamp of '5 seconds ago'. The log output shows the message 'Stopped: ['i-08dd9fe8c781f8445']'. Below the log, there are sections for Summary, Log output, and a detailed view of the execution environment.

- Create Start Function by navigating to Lambda → Create function → Name: "StartECFunction" → Provide the existing Lambda EC2 Role → Create function.

The screenshot shows the AWS Lambda Create function page. The path is Lambda > Functions > Create function. The 'Create function' section has three options: 'Author from scratch' (selected), 'Use a blueprint' (Build a Lambda application from sample code and configuration presets for common use cases), and 'Container image' (Select a container image to deploy for your function). The 'Basic information' section includes fields for 'Function name' (StartECFunction), 'Runtime' (Python 3.12), 'Durable execution' (disabled), and 'Architecture' (x86_64). There are also sections for 'Environment variables', 'Code', and 'Role'.

- Provide necessary code for the **Lambda Start Function** → Deploy.



The screenshot shows the AWS Lambda function editor interface. On the left, the Explorer sidebar displays a project named 'STARTEC2FUNCTION' containing a file 'lambda_function.py'. The main area shows the Python code for the Lambda function:

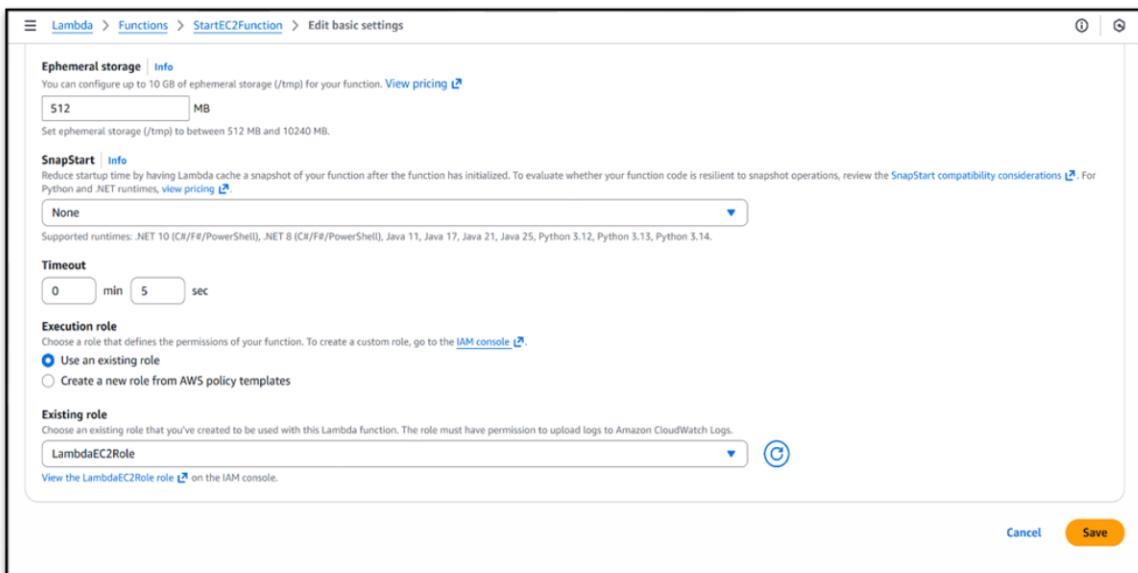
```

1 import boto3
2 def lambda_handler(event, context):
3     ec2 = boto3.client('ec2', region_name='us-east-1')
4     response = ec2.describe_instances(Filters=[{'Name': 'tag:Schedule', 'Values': ['Yes']}])
5     instance_ids = []
6     for reservation in response['Reservations']:
7         for instance in reservation['Instances']:
8             if instance['State']['Name'] == 'stopped': # ← CHANGE HERE
9                 instance_ids.append(instance['InstanceId'])
10    if instance_ids:
11        ec2.start_instances(InstanceIds=instance_ids) # ← CHANGE HERE
12    return f'Started: {instance_ids}'
13

```

Below the code, there are deployment buttons: 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)'. The 'TEST EVENTS' section is currently empty.

- Configure the code by navigating to configuration → Enter the timeout seconds → Save.

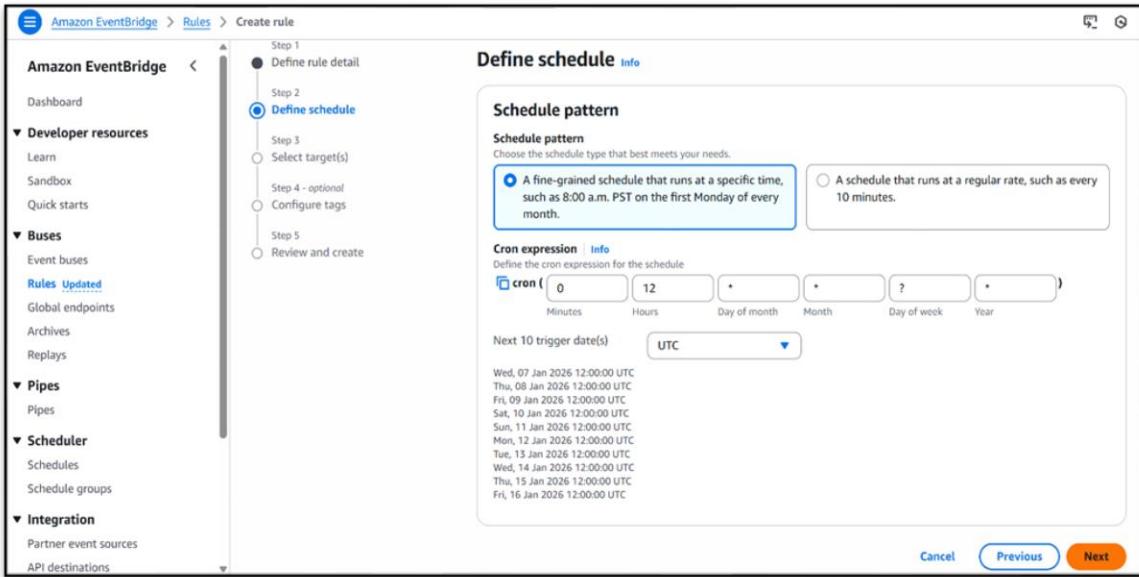


- Test the Lambda function by navigating to Test → Proceed with empty JSON code → Save → Test.
- You can find that the instance has been started and the start function has been executed successfully.

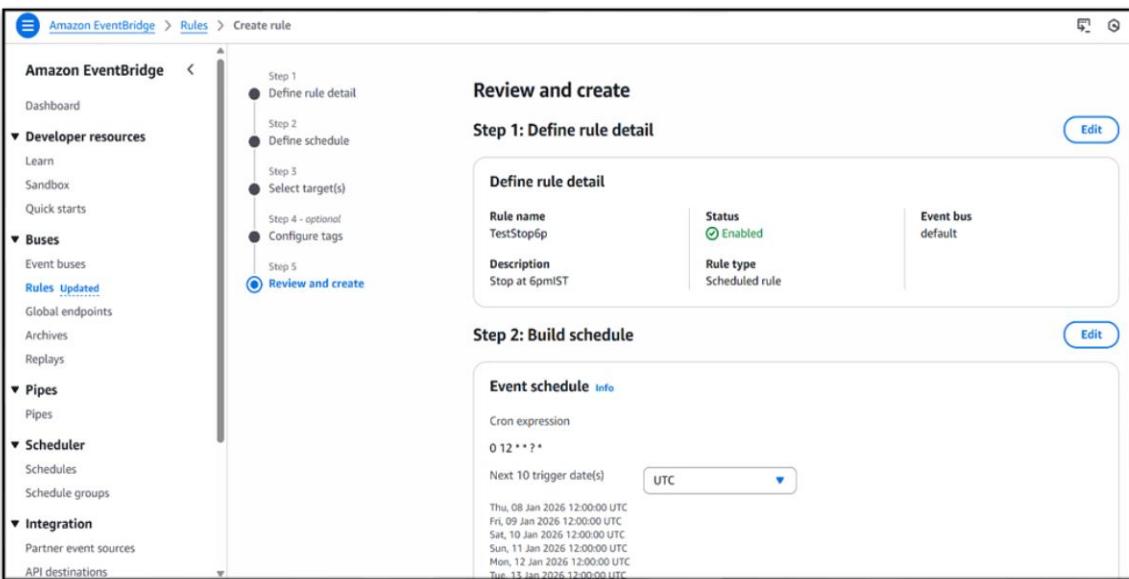
The top screenshot shows the AWS EC2 Instances page. A single instance named "Test" (ID: i-08dd9fe8c781f8445) is listed as "Running". The bottom screenshot shows the Lambda Function Details for "StartEC2Function", indicating it executed successfully with a duration of 4635.00 ms and a memory usage of 128 MB.

EventBridge:

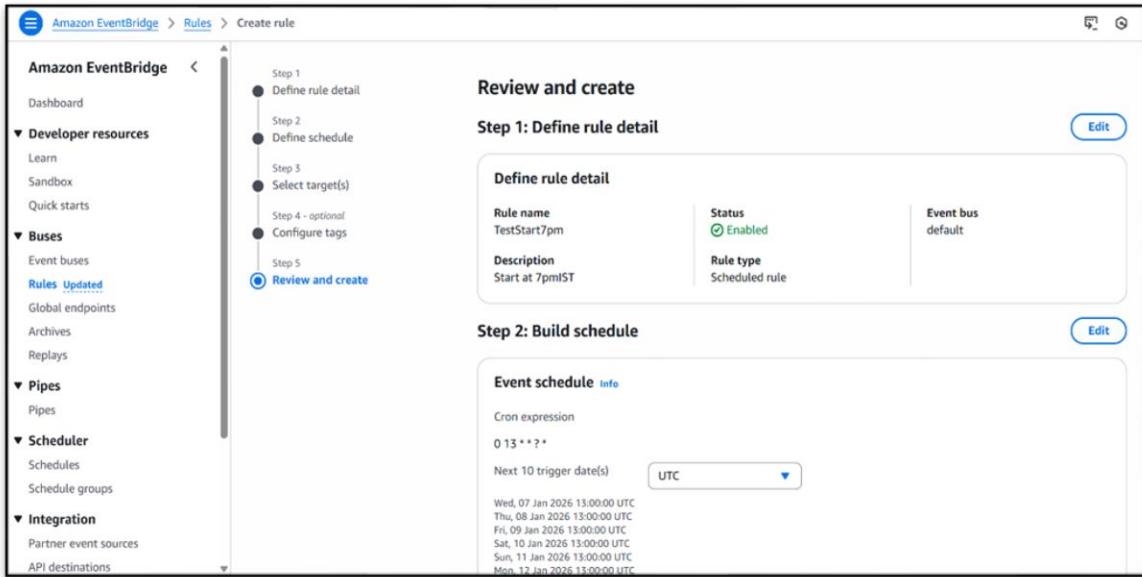
- Navigate to Amazon Eventbridge Console → Rules → Create rule
→ Provide the name and description → Next.
- Define schedule by entering appropriate cron expression → Next.



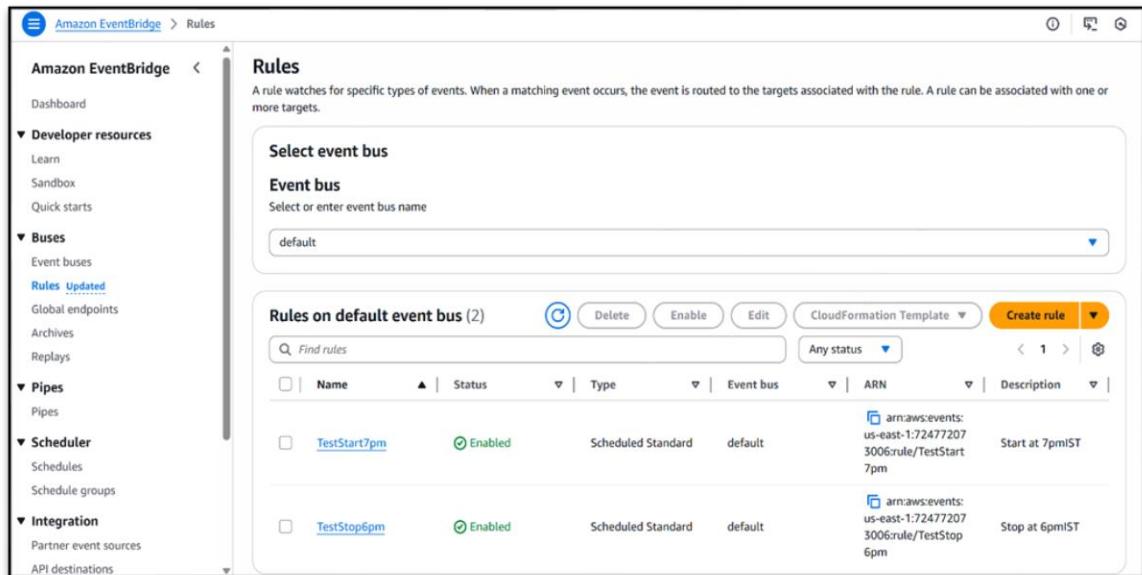
- Select the target type as Lambda Function → Stop EC2 Function
→ Next → Review and Create.



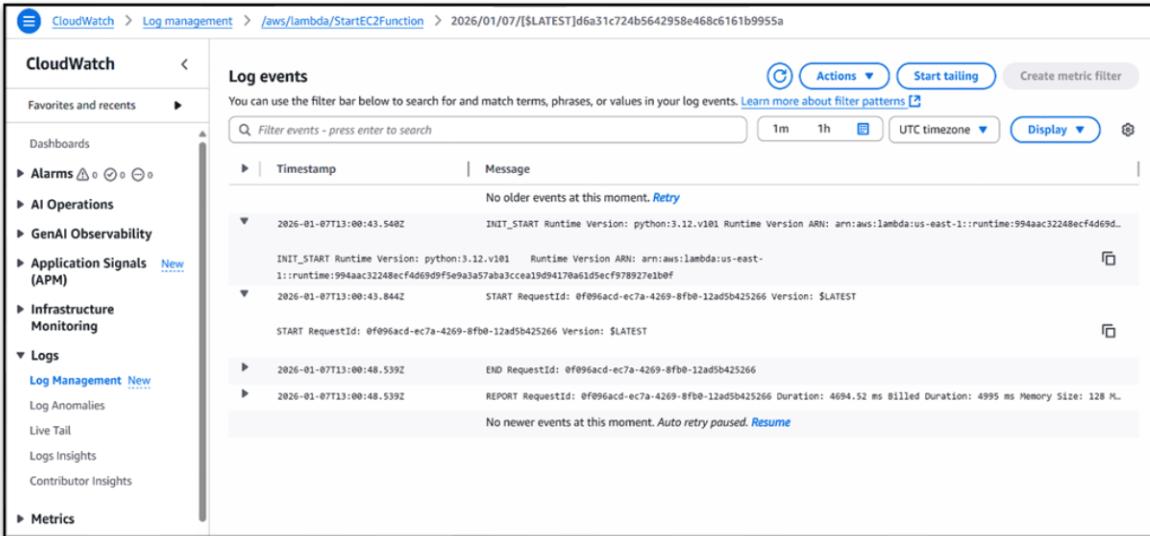
- Follow the same steps and create another rule for Start EC2 Function.



- Both the Start and Stop rule have been successfully created to schedule the EC2.



- Through the log events we could find that instance have stopped and started at the specified time intervals.



- The EC2 instances have been scheduled to stop and start at the specified time using IAM policy, Lambda functions and Eventbridge.

Conclusion:

- This AWS EC2 auto stop/start project successfully implemented serverless automation using Lambda and EventBridge, achieving 75% cost savings by stopping instances during specified times.