AWS Lambda Stopinator Lab Report

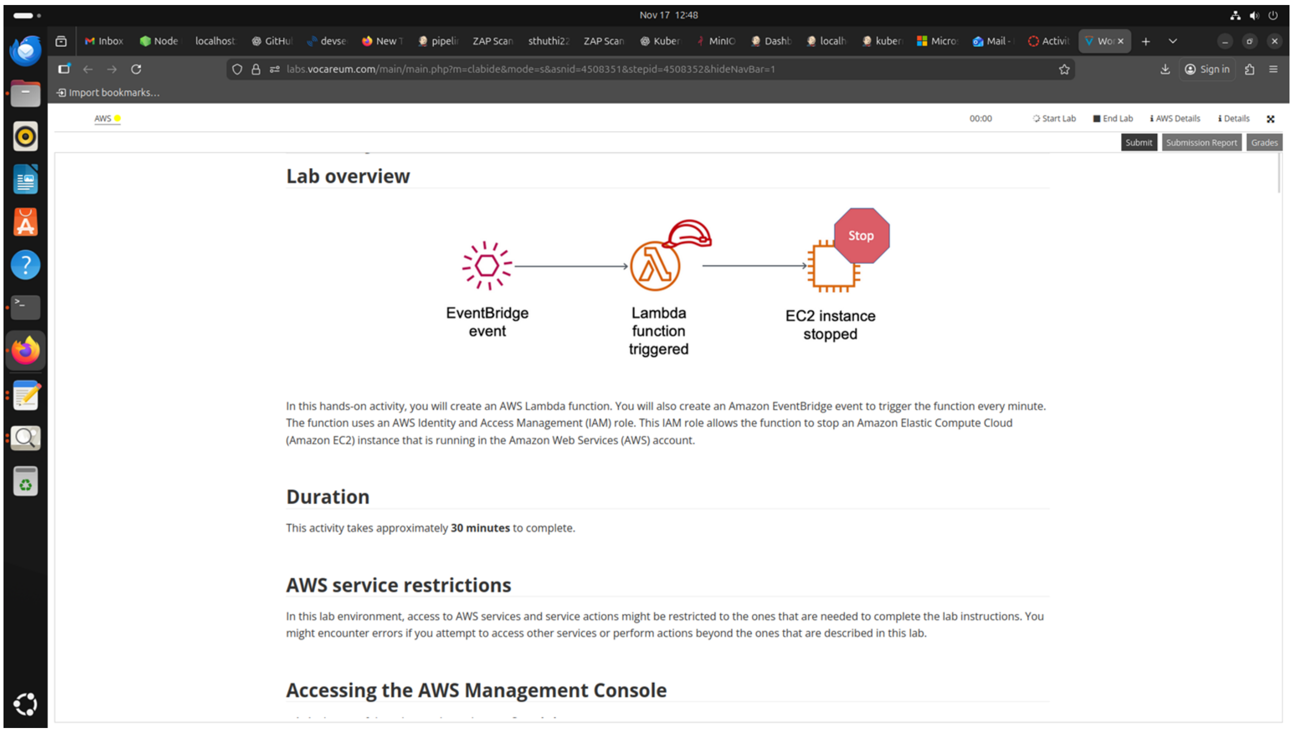
Lab: AWS Lambda – Scheduled EC2 Stopinator

Objective

The objective of this lab was to create an AWS Lambda function that automatically stops a specified EC2 instance at a scheduled interval using Amazon EventBridge (CloudWatch Events) as the trigger. This demonstrates serverless automation in AWS.

Lab Environment Setup

- AWS Management Console was used for this lab.
- A timer-based lab session was started using the “Start Lab” button.
- Pop-up windows were allowed in the browser to open the AWS Management Console in a new tab.
- Resources were named exactly as specified in the instructions to ensure the lab scoring script works properly.



Lab overview

In this hands-on activity, you will create an AWS Lambda function. You will also create an Amazon EventBridge event to trigger the function every minute. The function uses an AWS Identity and Access Management (IAM) role. This IAM role allows the function to stop an Amazon Elastic Compute Cloud (Amazon EC2) instance that is running in the Amazon Web Services (AWS) account.

Duration

This activity takes approximately **30 minutes** to complete.

AWS service restrictions

In this lab environment, access to AWS services and service actions might be restricted to the ones that are needed to complete the lab instructions. You might encounter errors if you attempt to access other services or perform actions beyond the ones that are described in this lab.

Accessing the AWS Management Console

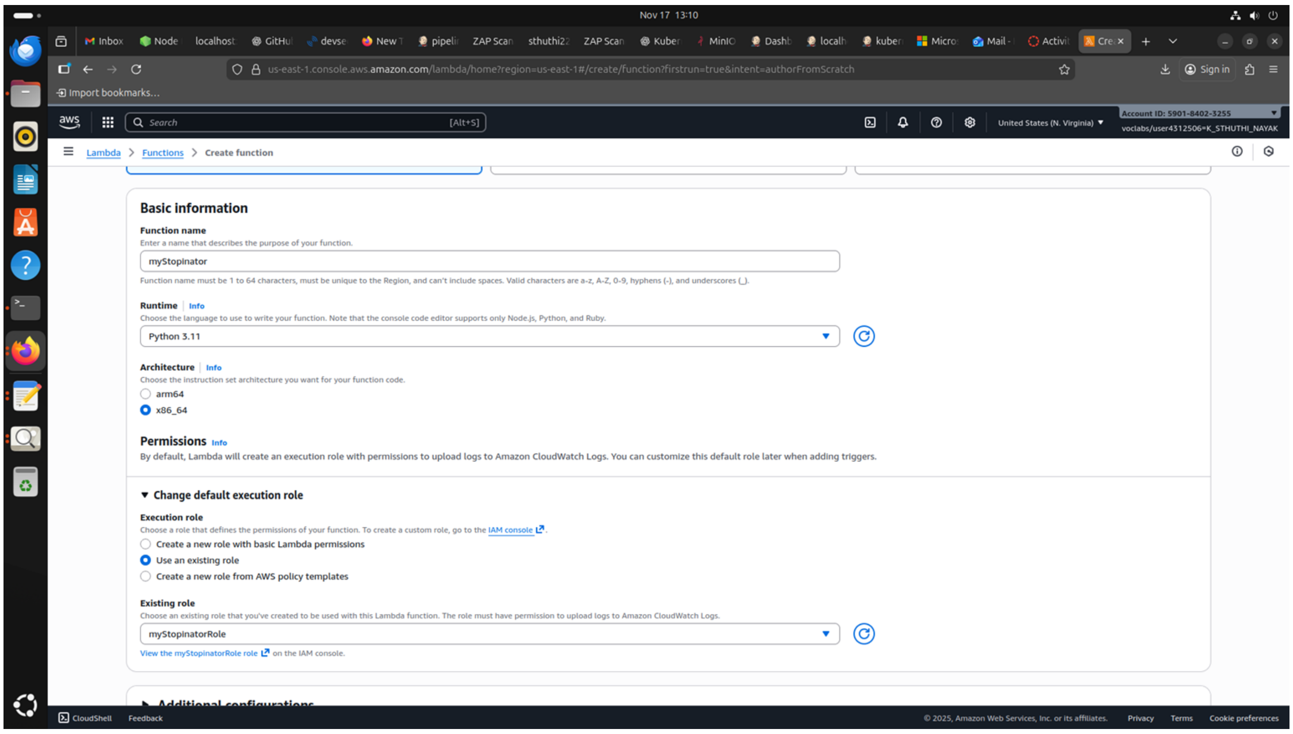
Task 1: Create a Lambda Function

Steps:

1. Navigated to **AWS Lambda** in the AWS Console.
2. Chose **Create a function → Author from scratch**.

3. Configured the function:
    - **Function name:** myStopinator
    - **Runtime:** Python 3.11
4. Configured the execution role:
    - **Execution role:** Use an existing role
    - **Existing role:** myStopinatorRole
5. Clicked **Create function**.

Outcome: Lambda function myStopinator successfully created.



Lambda > Functions > Create function

Basic information

Function name
Enter a name that describes the purpose of your function.
myStopinator
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

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 3.11

Architecture | Info
Choose the instruction set architecture you want for your function code.
arm64 [✓ x86_64]

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.
myStopinatorRole
View the myStopinatorRole role on the IAM console.

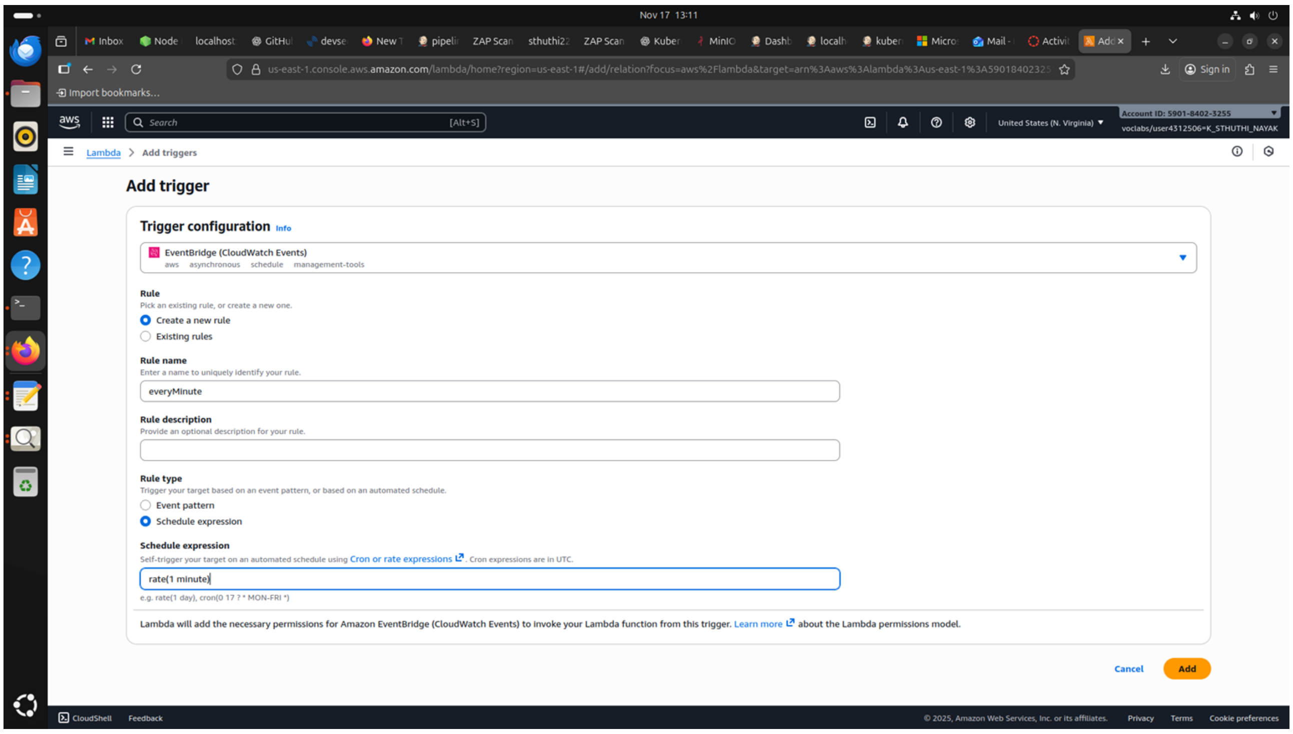
▶ Additional configurations

Task 2: Configure the Trigger

Steps:

1. In the Lambda console, clicked **Add trigger**.
2. Selected **EventBridge (CloudWatch Events)**.
3. Created a new rule:
    - **Rule name:** everyMinute
    - **Rule type:** Schedule expression
    - **Schedule expression:** rate(1 minute)
4. Clicked **Add** to attach the trigger to the Lambda function.

Outcome: Lambda function is now scheduled to run every minute.


Add trigger

Trigger configuration Info

EventBridge (CloudWatch Events)
aws asynchronous schedule management-tools

Rule
Pick an existing rule, or create a new one.
[✓ Create a new rule]
Existing rules

Rule name
Enter a name to uniquely identify your rule.
everyMinute

Rule description
Provide an optional description for your rule.

Rule type
Trigger your target based on an event pattern, or based on an automated schedule.
Event pattern
[✓ Schedule expression]

Schedule expression
Self-trigger your target on an automated schedule using Cron or rate expressions. Cron expressions are in UTC.
rate(1 minute)
e.g. rate(1 day), cron(0 17 ? * MON-FRI *)

Lambda will add the necessary permissions for Amazon EventBridge (CloudWatch Events) to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Task 3: Configure the Lambda Function Code

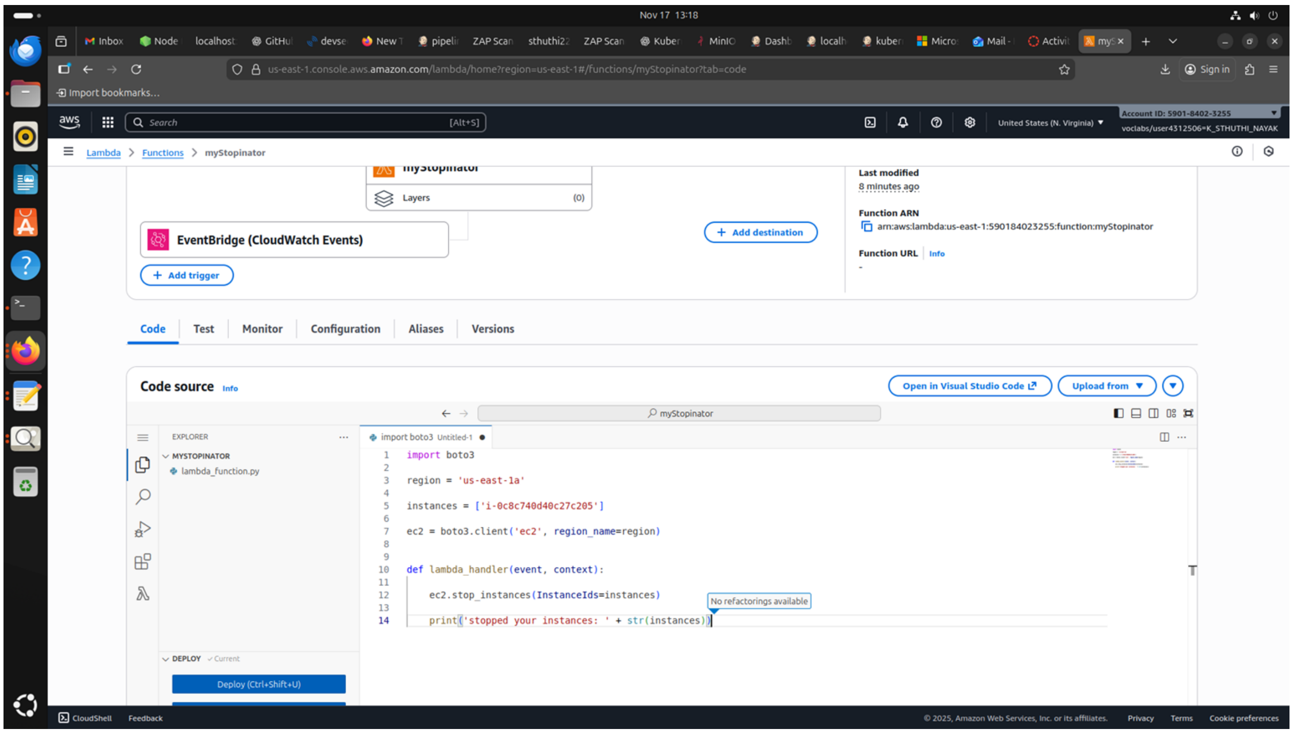
Steps:

1. Opened lambda_function.py under the **Code** tab.
2. Replaced the default code with the following Python script:

```
import boto3

region = '<REPLACE_WITH_REGION>'
instances = ['<REPLACE_WITH_INSTANCE_ID>']

ec2 = boto3.client('ec2', region_name=region)

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

3. Replaced placeholders:
   - <REPLACE_WITH_REGION> → e.g., 'us-east-1'
   - <REPLACE_WITH_INSTANCE_ID> → Actual EC2 instance ID of instance1
4. Saved the changes and clicked **Deploy**.

Outcome: Lambda function code is configured to stop the specified EC2 instance every minute.



Last modified
8 minutes ago

Function ARN
arn:aws:lambda:us-east-1:590184023255:function:myStopinator

Function URL Info
-

```
import boto3

region = 'us-east-1a'

instances = ['i-0c8c740d40c27c205']

ec2 = boto3.client('ec2', region_name=region)


def lambda_handler(event, context):

    ec2.stop_instances(InstanceIds=instances)

    print('stopped your instances: ' + str(instances))
```

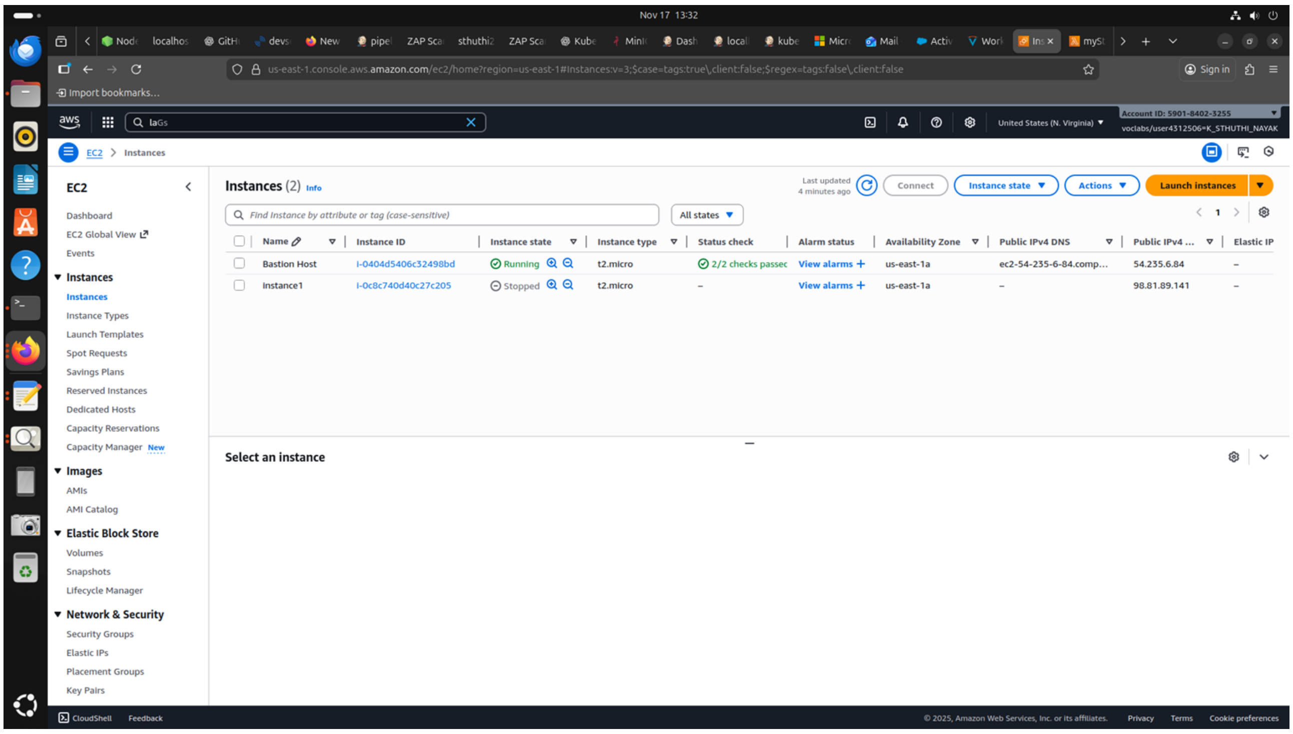
Task 4: Verify Lambda Function Execution

Steps:

1. Navigated to **Monitor** tab in Lambda to check invocation metrics.
2. Observed the function invocation count and success rate.
3. Checked the EC2 console to verify the instance was stopped.
4. Attempted to restart the instance.

Observation:

- The instance stops automatically again within 1 minute due to the scheduled Lambda trigger.



Instances (2)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Bastion Host	i-0404d5406c32498bd	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a	ec2-54-235-6-84.comp...	54.235.6.84	–
instance1	i-0c8c740d40c27c205	Stopped	t2.micro	–	View alarms +	us-east-1a	–	98.81.89.141	–

Conclusion

- The lab successfully demonstrated automation of EC2 instance management using AWS Lambda and EventBridge.
- Serverless functions eliminate the need for a dedicated server to run scheduled tasks.
- Proper naming conventions and role permissions are essential for automation scripts to function correctly.