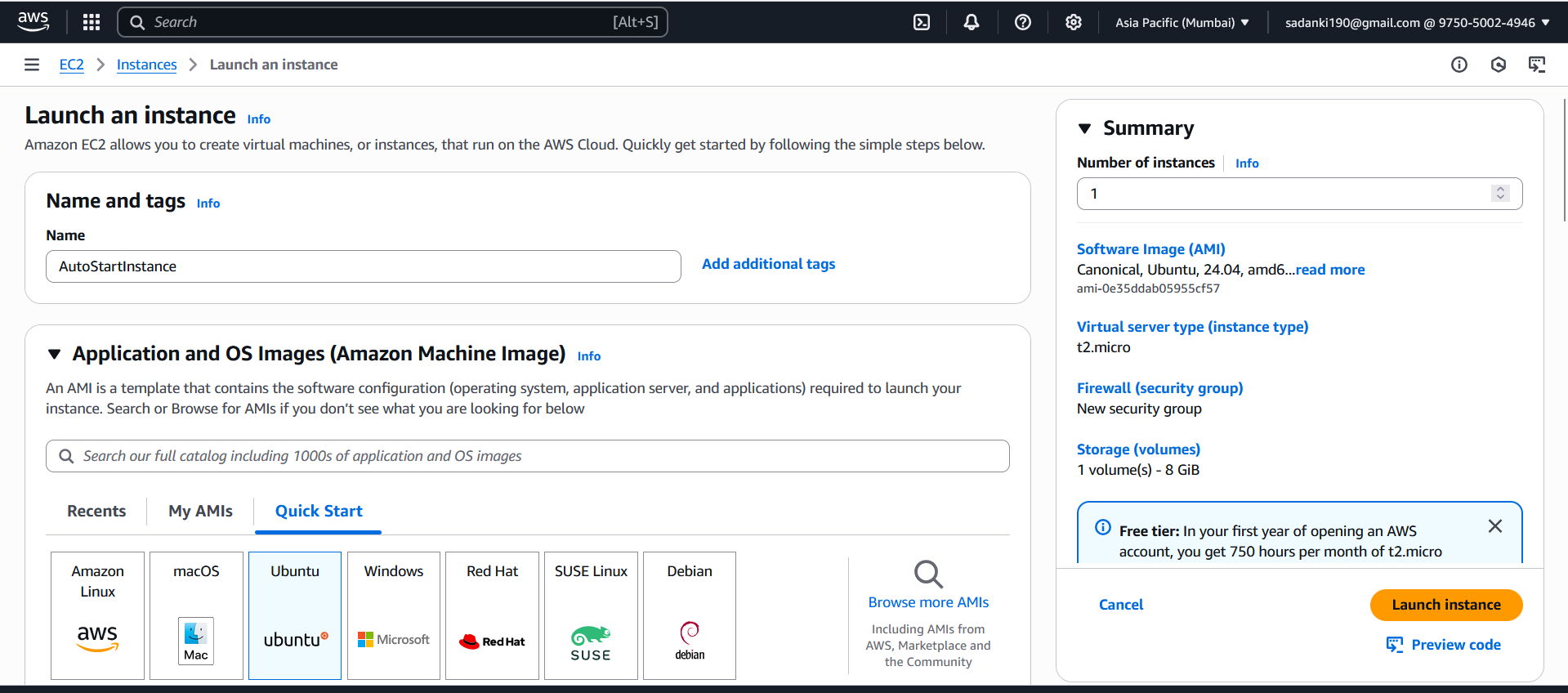
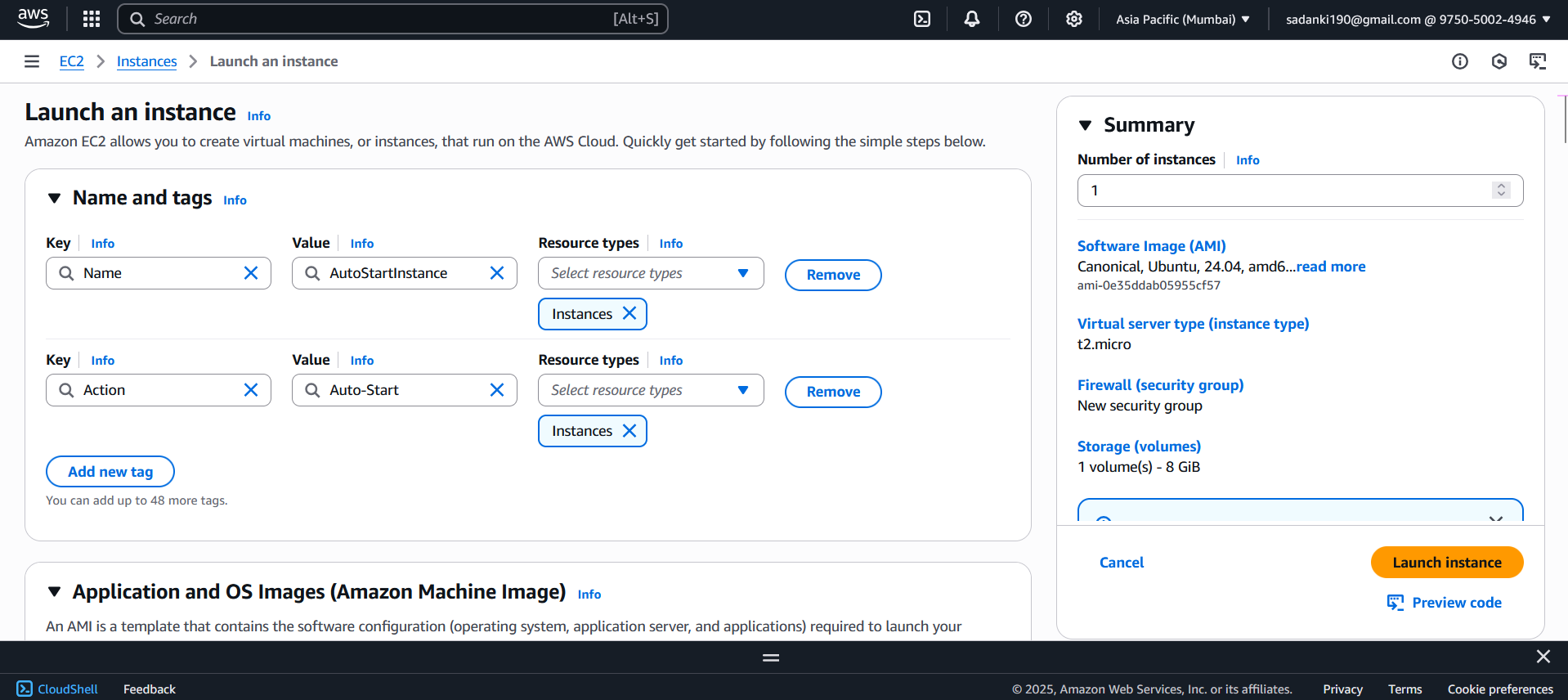
**Assignment 1:**

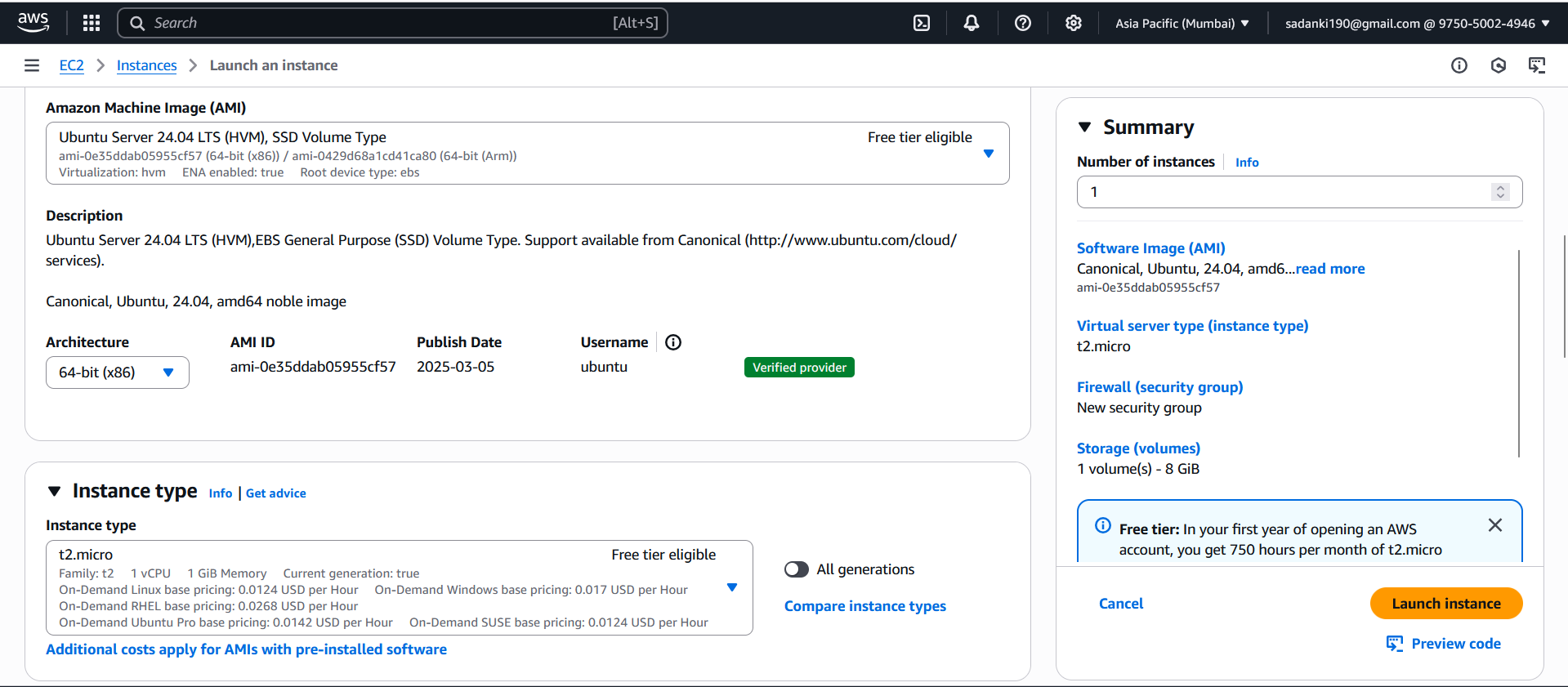
1. EC2 Setup

1. Go to EC2 Dashboard → Launch Instance.
2. Create two t2.micro instances:
   * Use Amazon Linux 2 AMI (free tier).
   * Name them AutoStopInstance and AutoStartInstance for clarity.
3. Tag them:
   * First instance: Key = Action, Value = Auto-Stop
   * Second instance: Key = Action, Value = Auto-Start

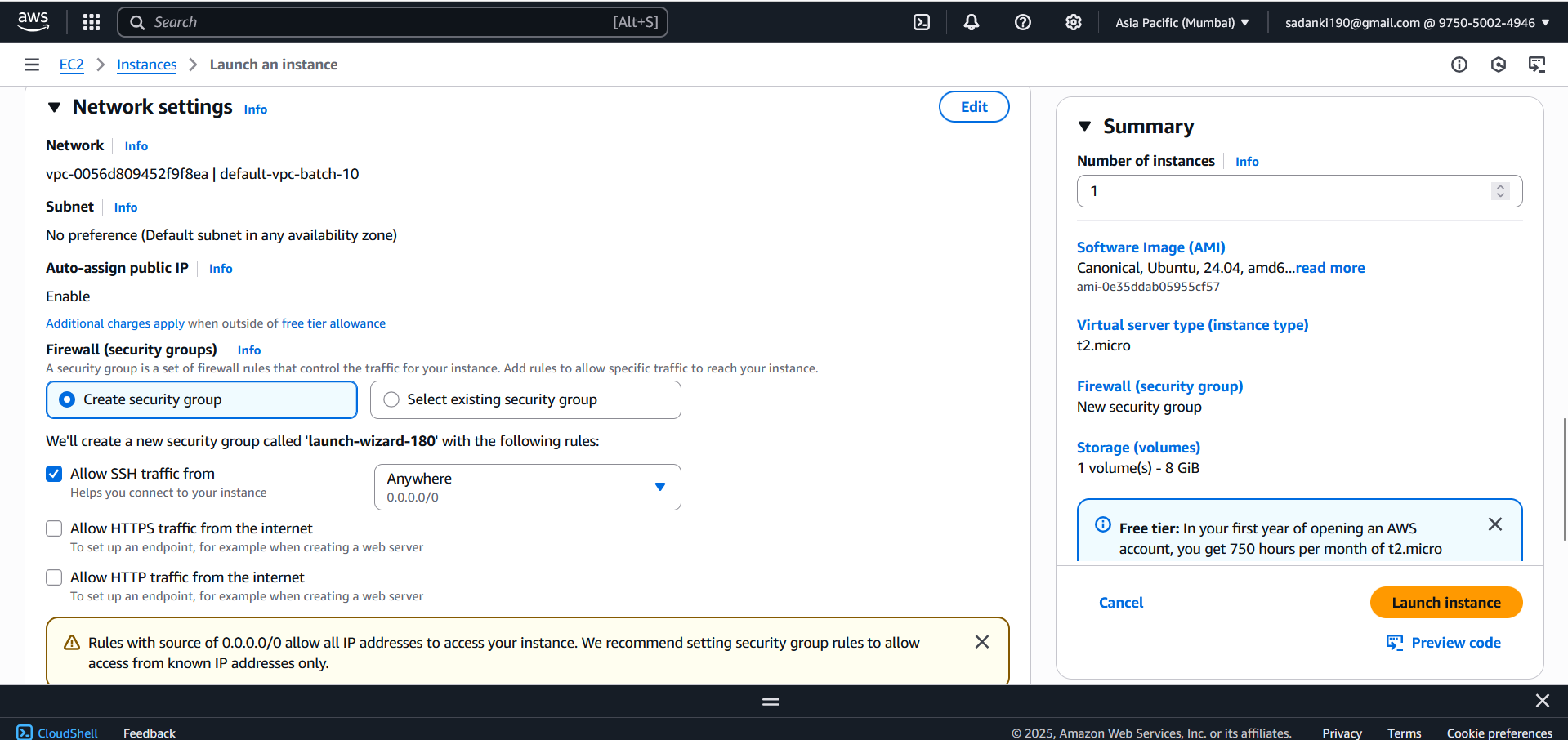
AutoStartInstance

****

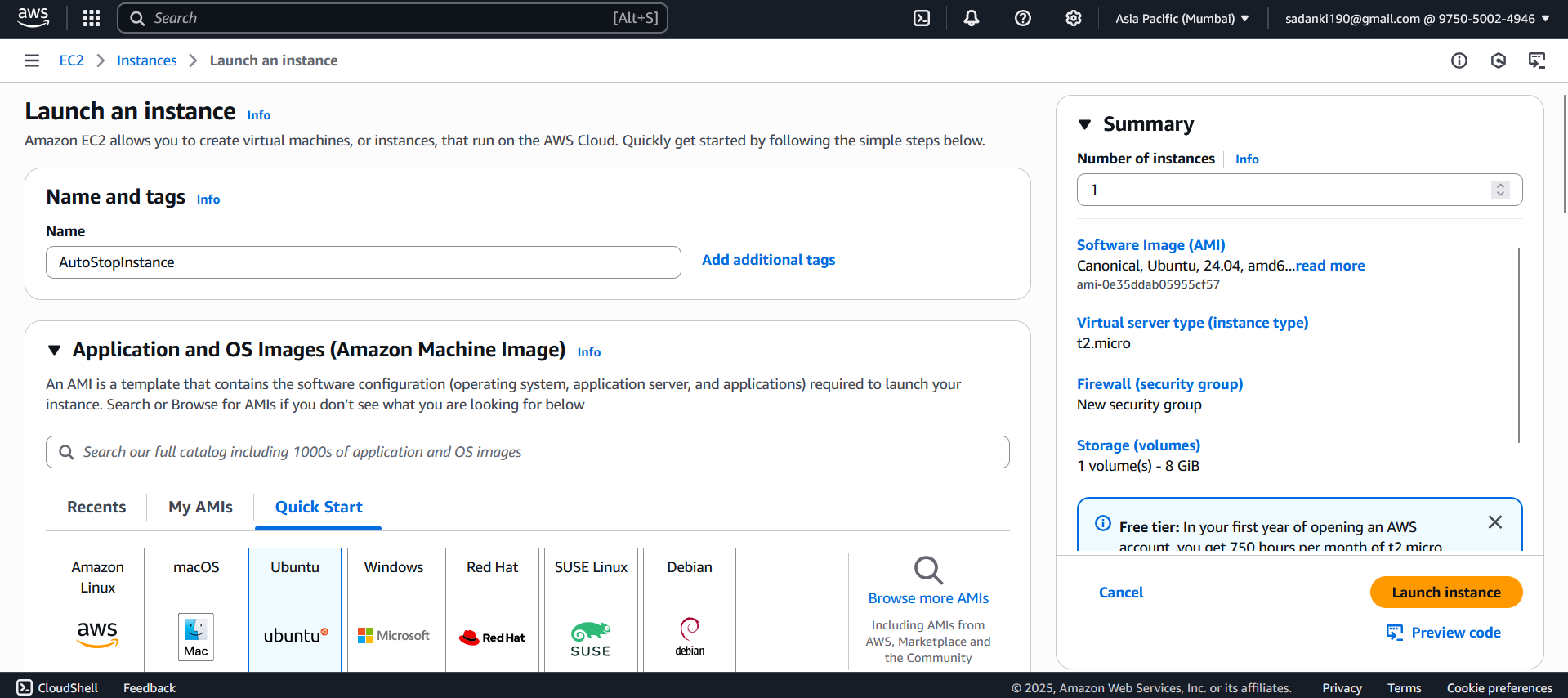
****

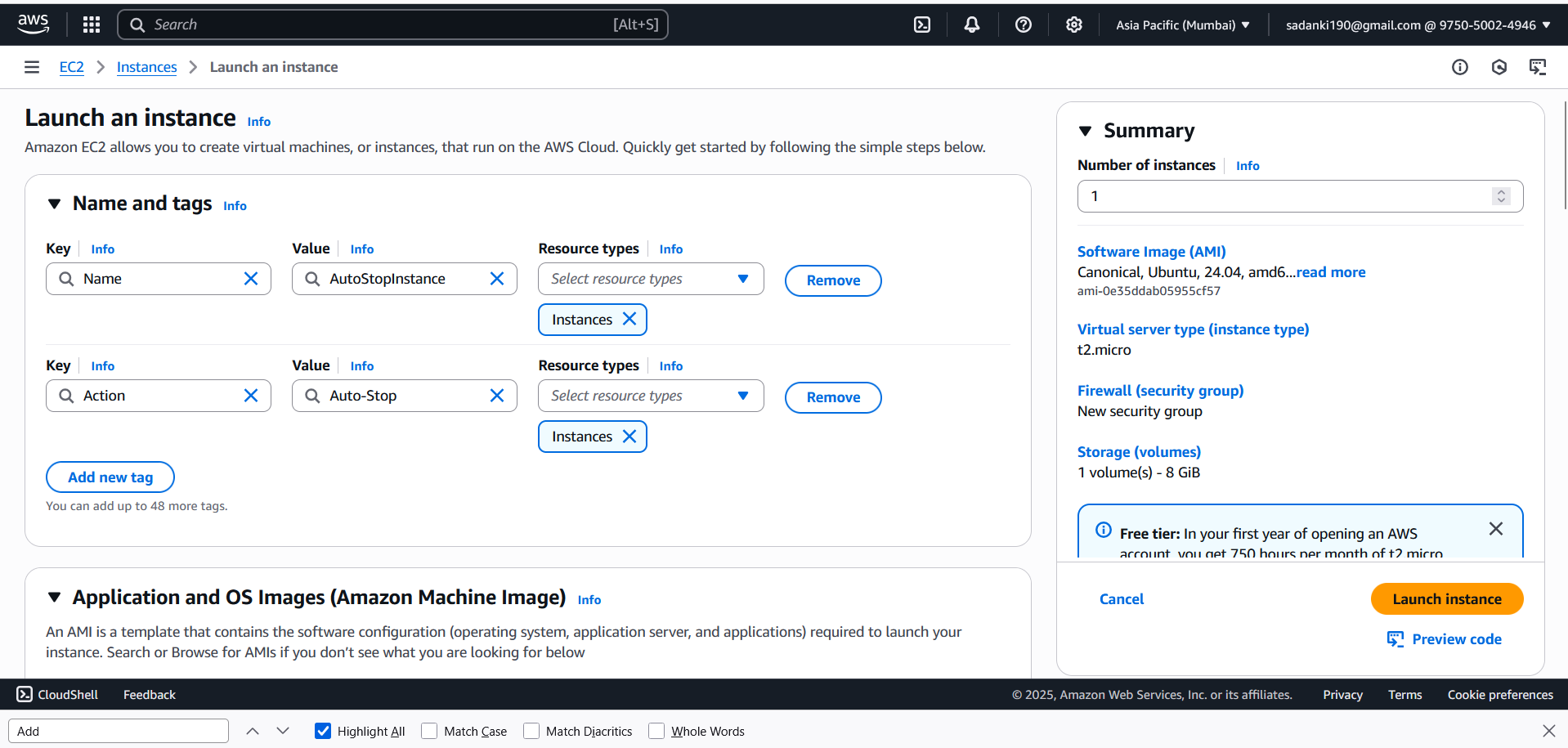
****

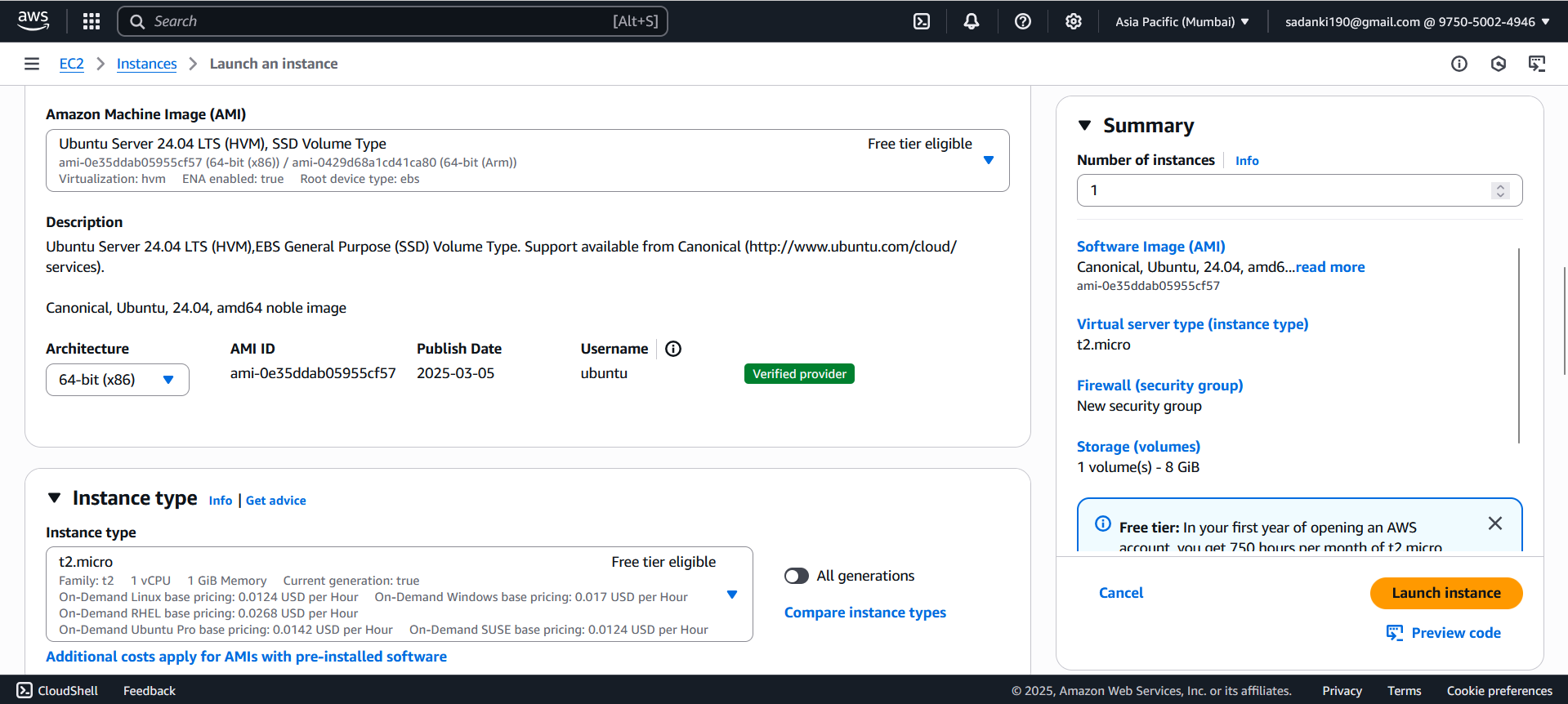


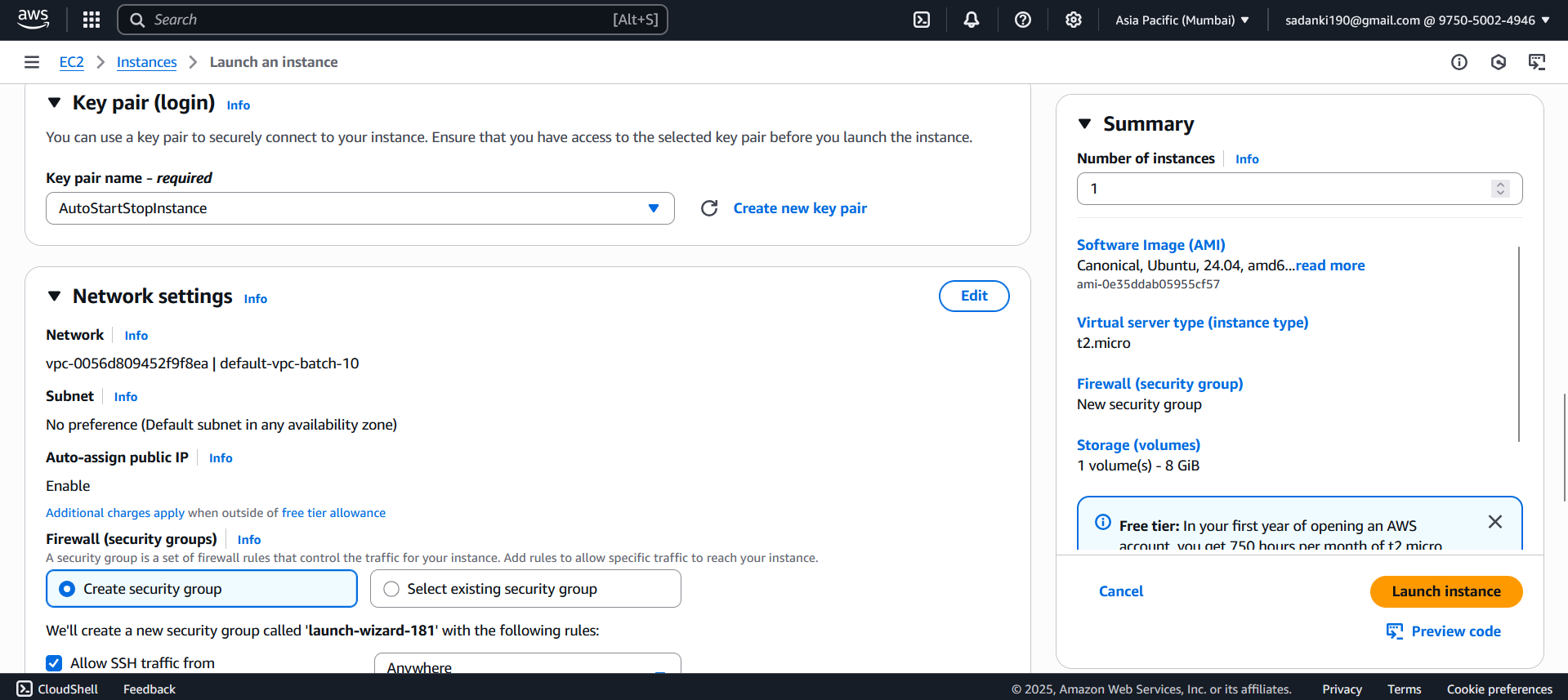


AutoStopInstance:

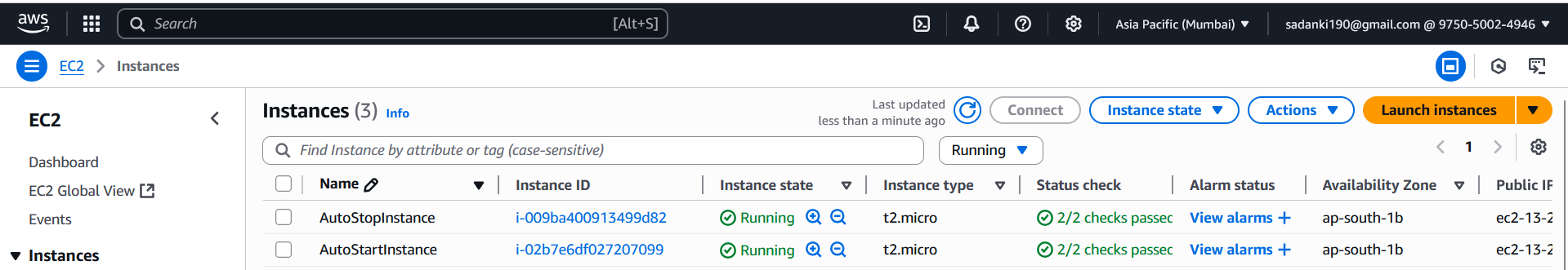






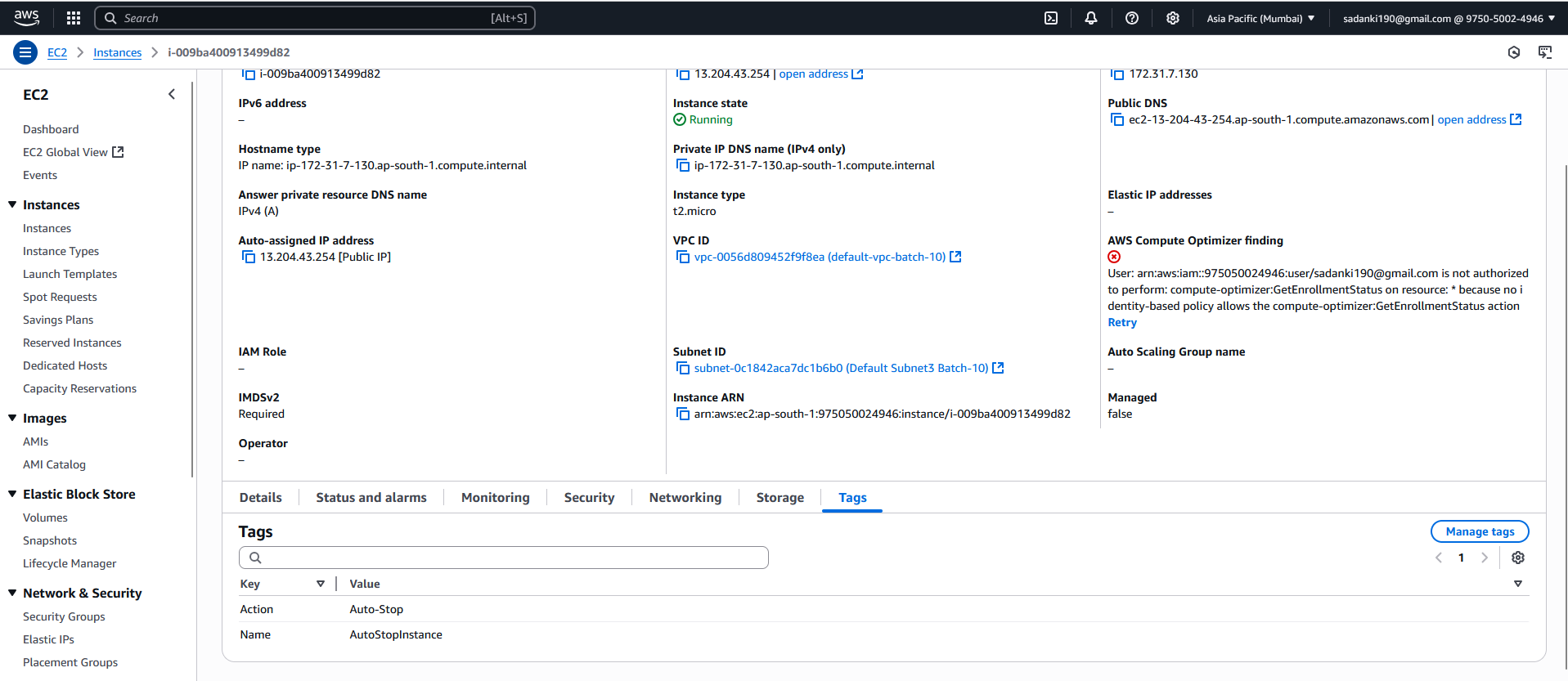


Both the instances are now running.

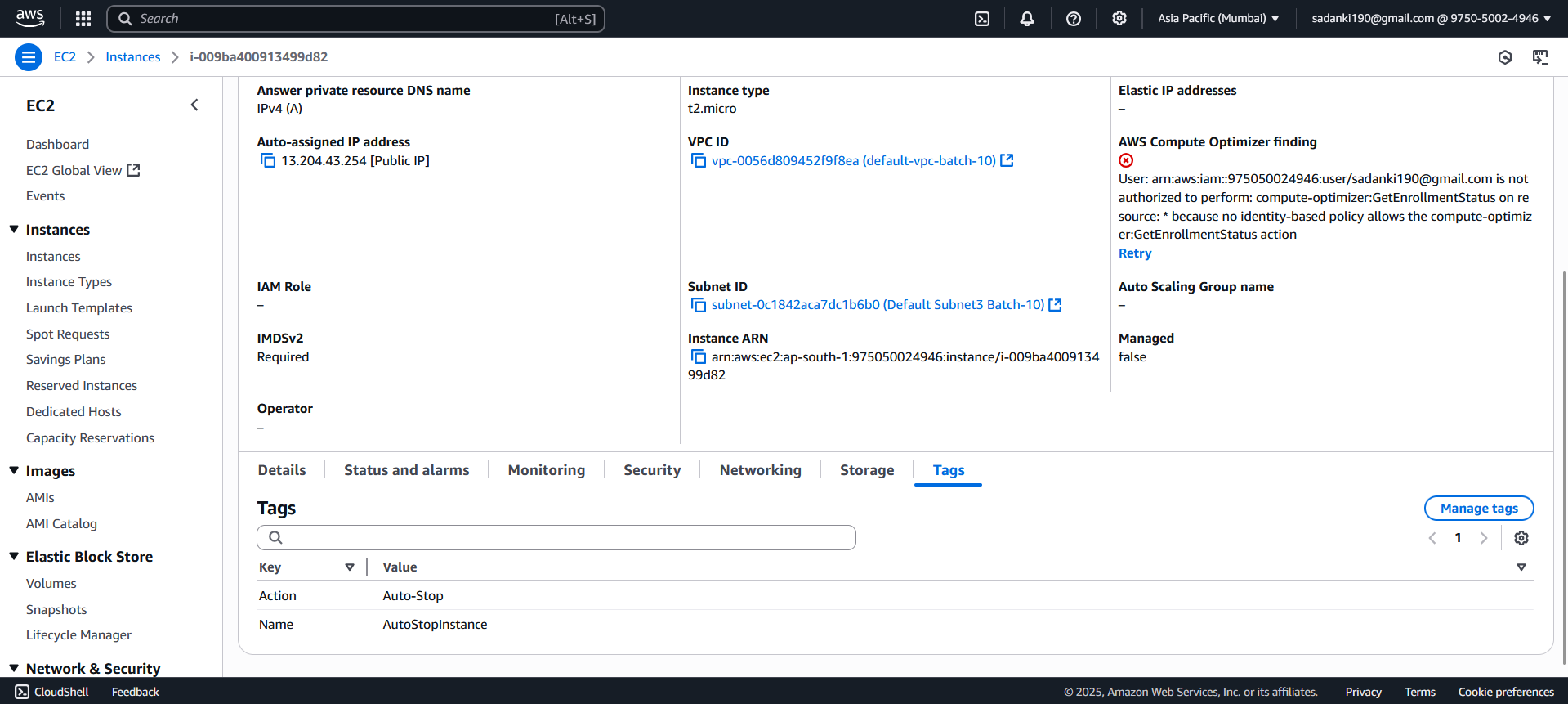


**Confirm Tags and Status**

1. Go to **EC2 Dashboard → Instances**.
2. Make sure you see:
   * AutoStartInstance is **running** and has tags:
     + Name = AutoStartInstance

Action = Auto-Start 

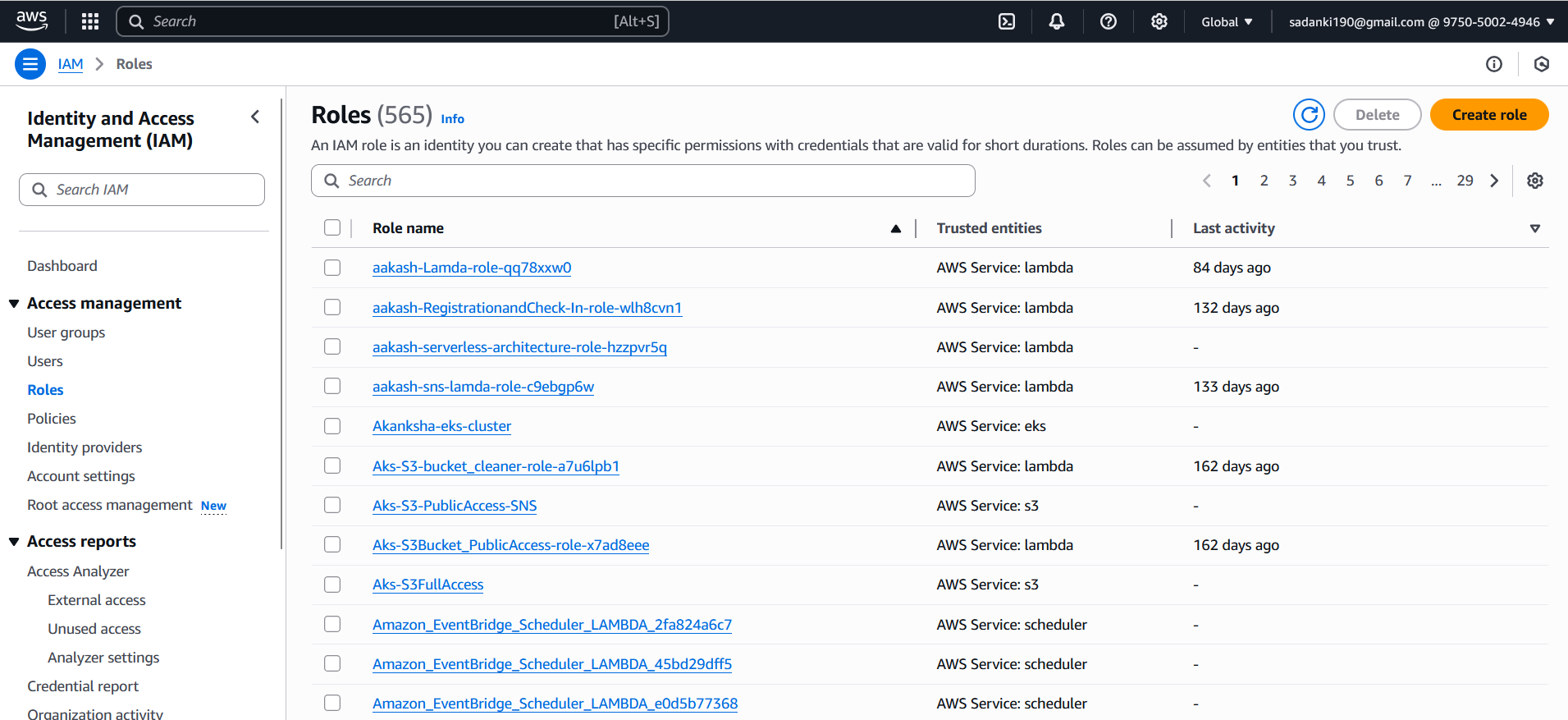
* + AutoStopInstance is **running** and has tags:
    - Name = AutoStopInstance
    - Action = Auto-Stop



**2. IAM Role Creation for Lambda**

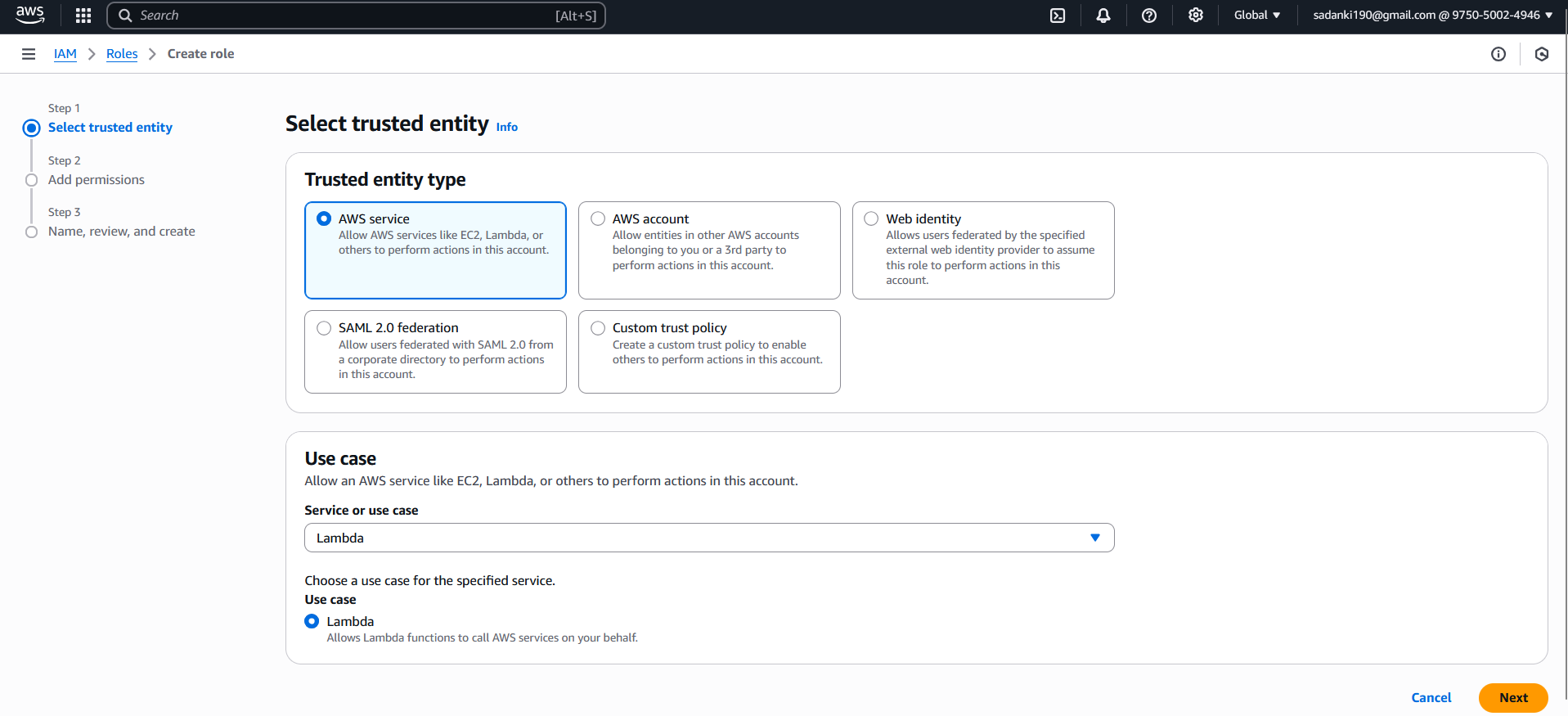
**2.1: Go to IAM Console**

1. In the sidebar, click **Roles** and Click the **Create role** button



**2.2: Select Trusted Entity**

1. Choose **Trusted entity type** as: AWS service
2. Choose **Use case**: Select **Lambda**
3. Click **Next**

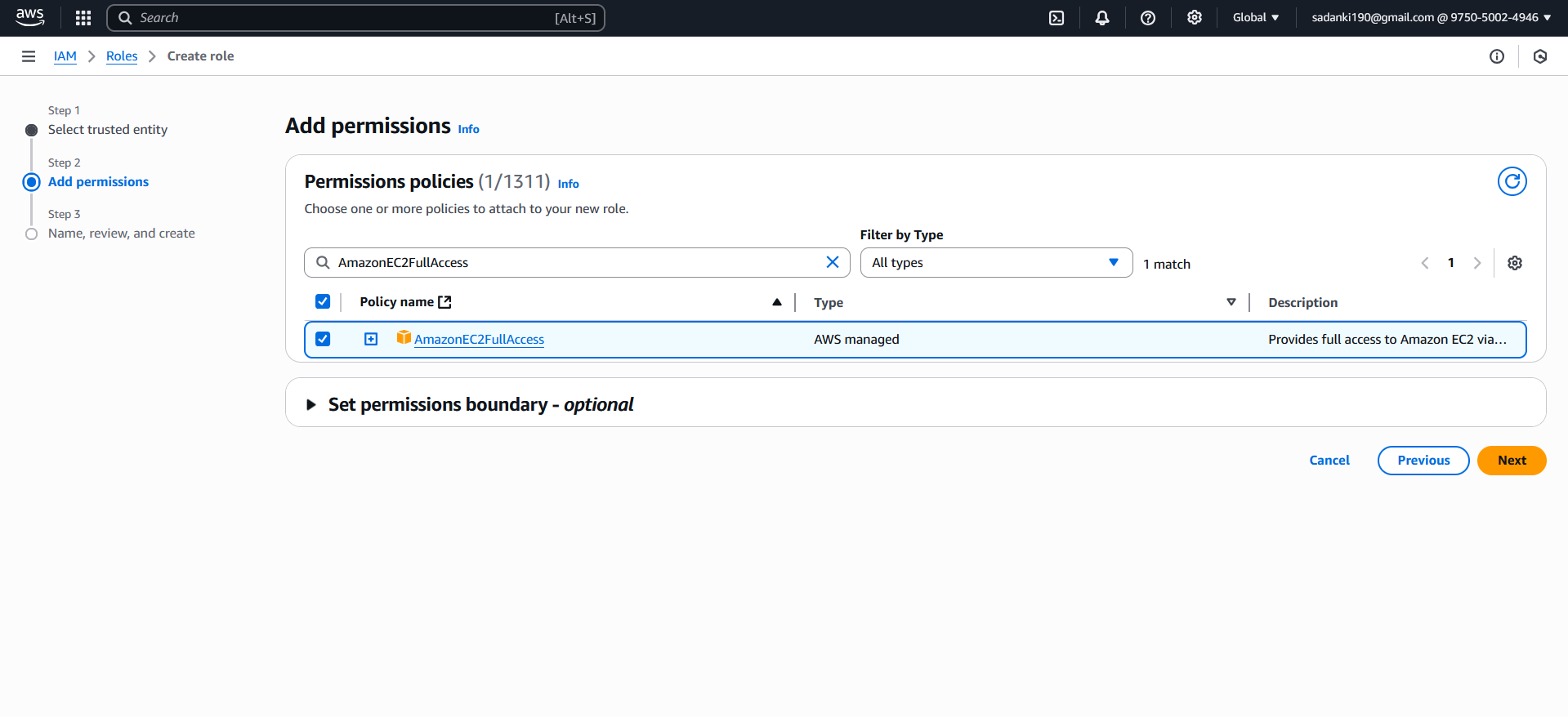


**2.3: Attach Permissions Policy**

1 In the search box, type: AmazonEC2FullAccess

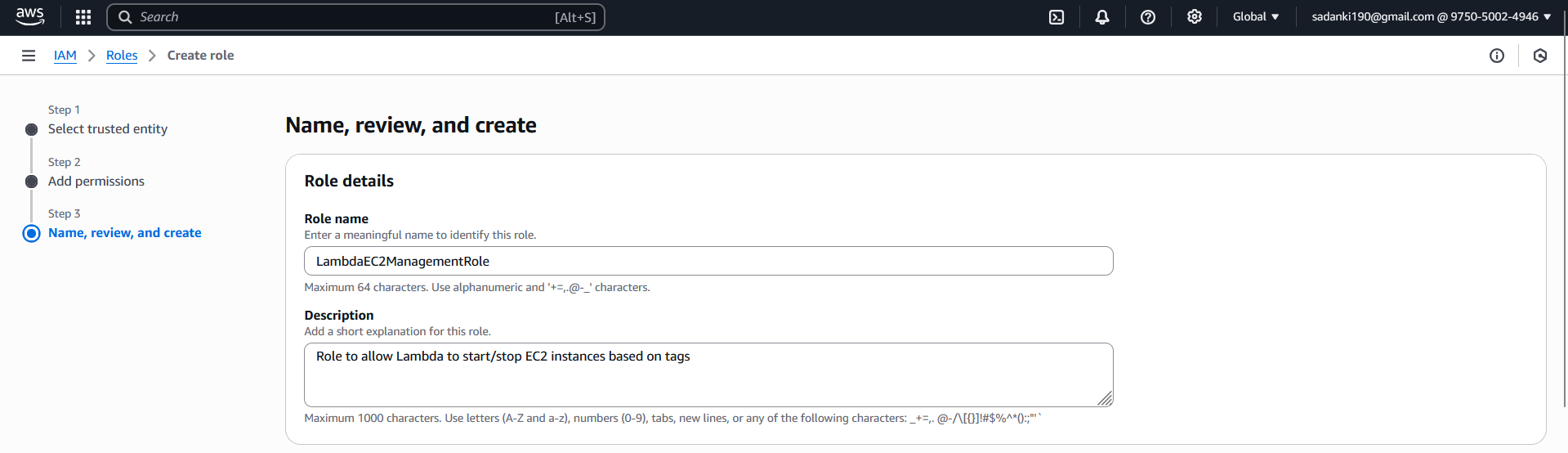
2 Check the box next to **AmazonEC2FullAccess**

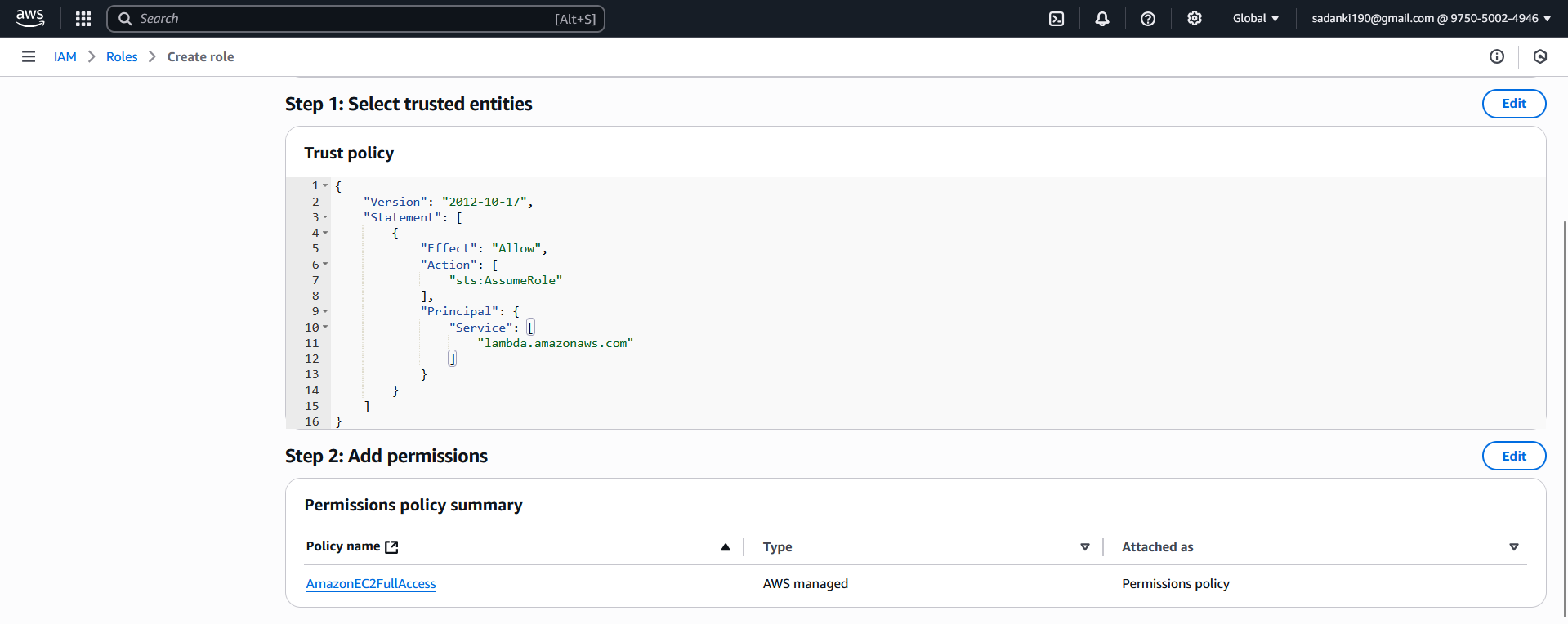
**3 Click Next**

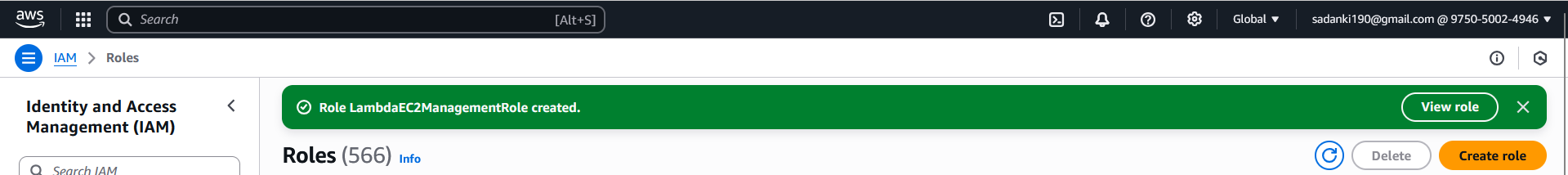
****

2.4: Name and Create Role:

1. **Role name**: LambdaEC2ManagementRole
2. Add a description: Role to allow Lambda to start/stop EC2 instances based on tags
3. Click **Create role**



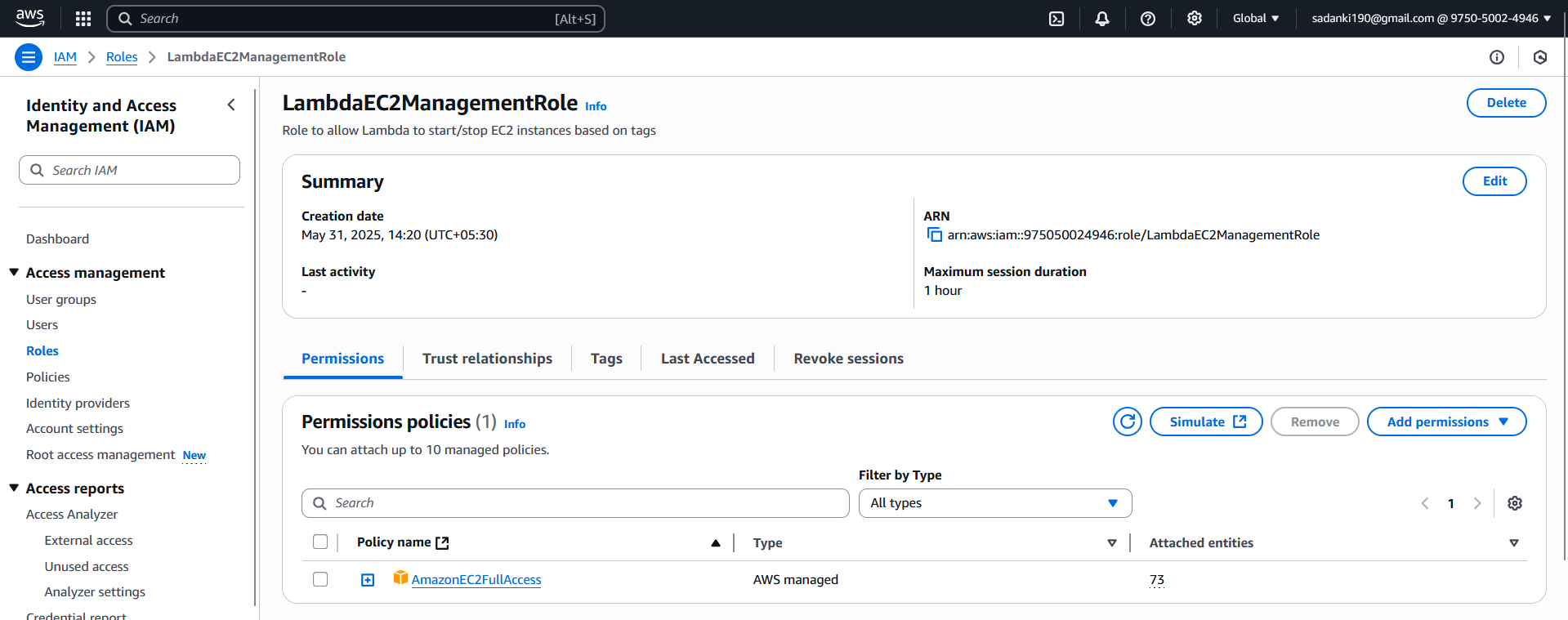


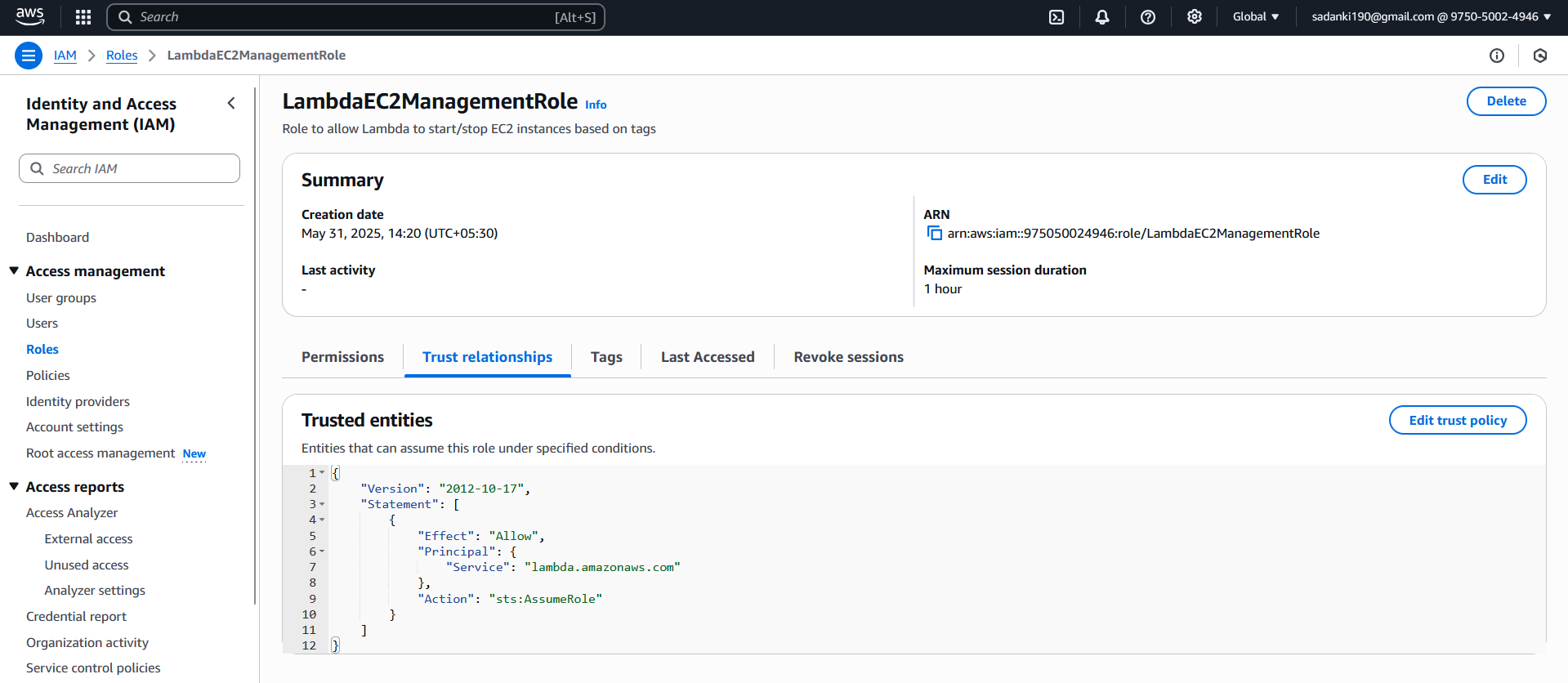


Verify the Role:

After creating:

1. You’ll be taken to the role's summary page.
2. Under **Permissions**, you should see: AmazonEC2FullAccess
3. Under **Trusted entities**, you should see: Service: lambda.amazonaws.com





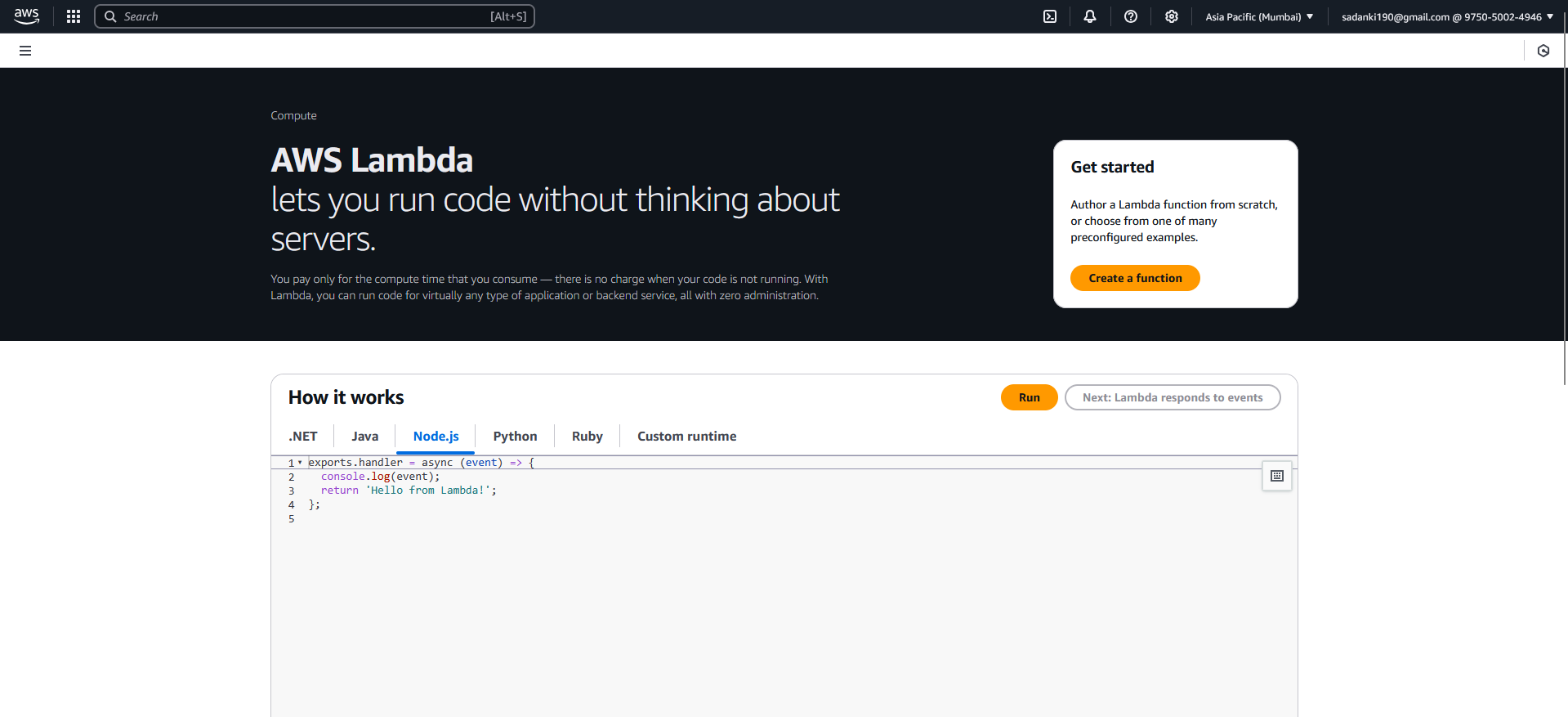
Step 3: Create the Lambda Function to Start/Stop EC2 Instances:

This Lambda function will:

* Start EC2 instances with tag Action = Auto-Start
* Stop EC2 instances with tag Action = Auto-Stop

**3.1: Go to AWS Lambda Console**

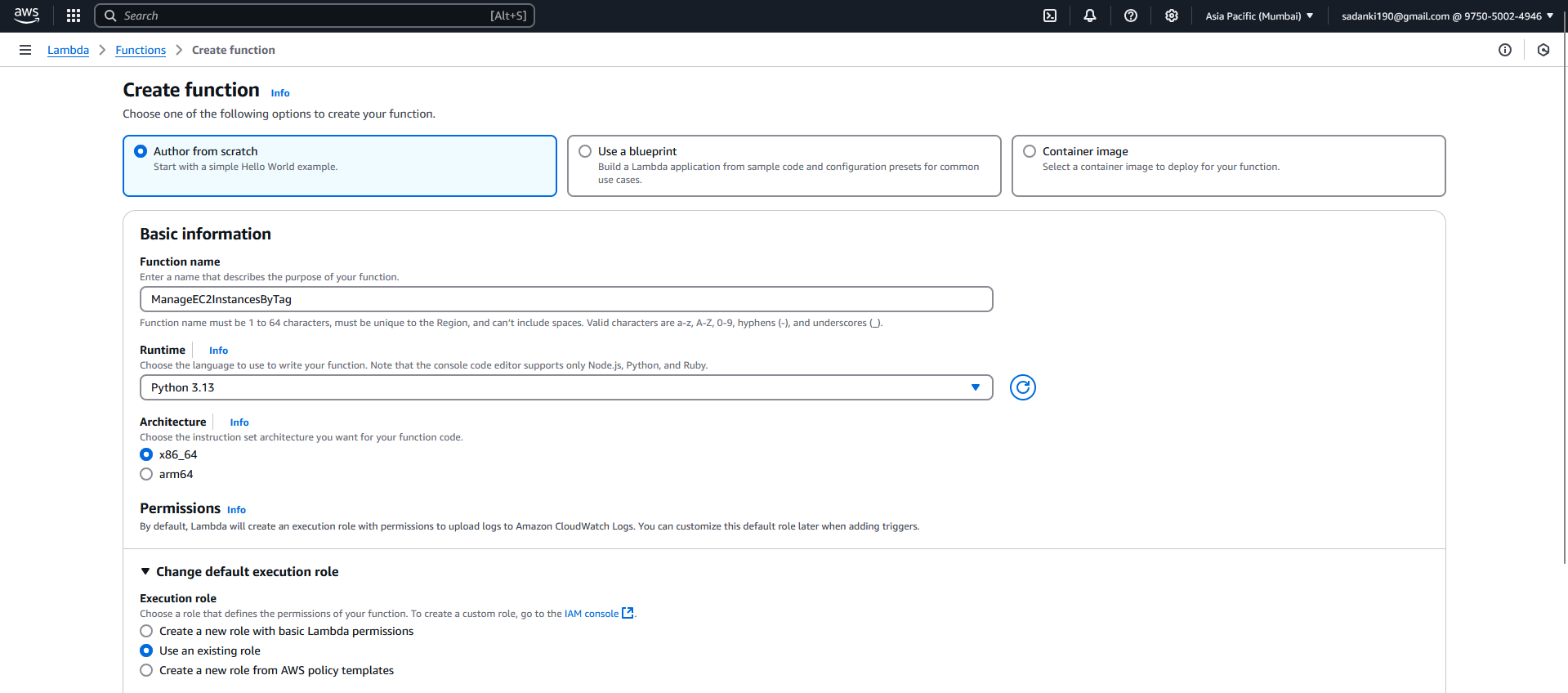
1. Open <https://console.aws.amazon.com/lambda>
2. Click **Create function**

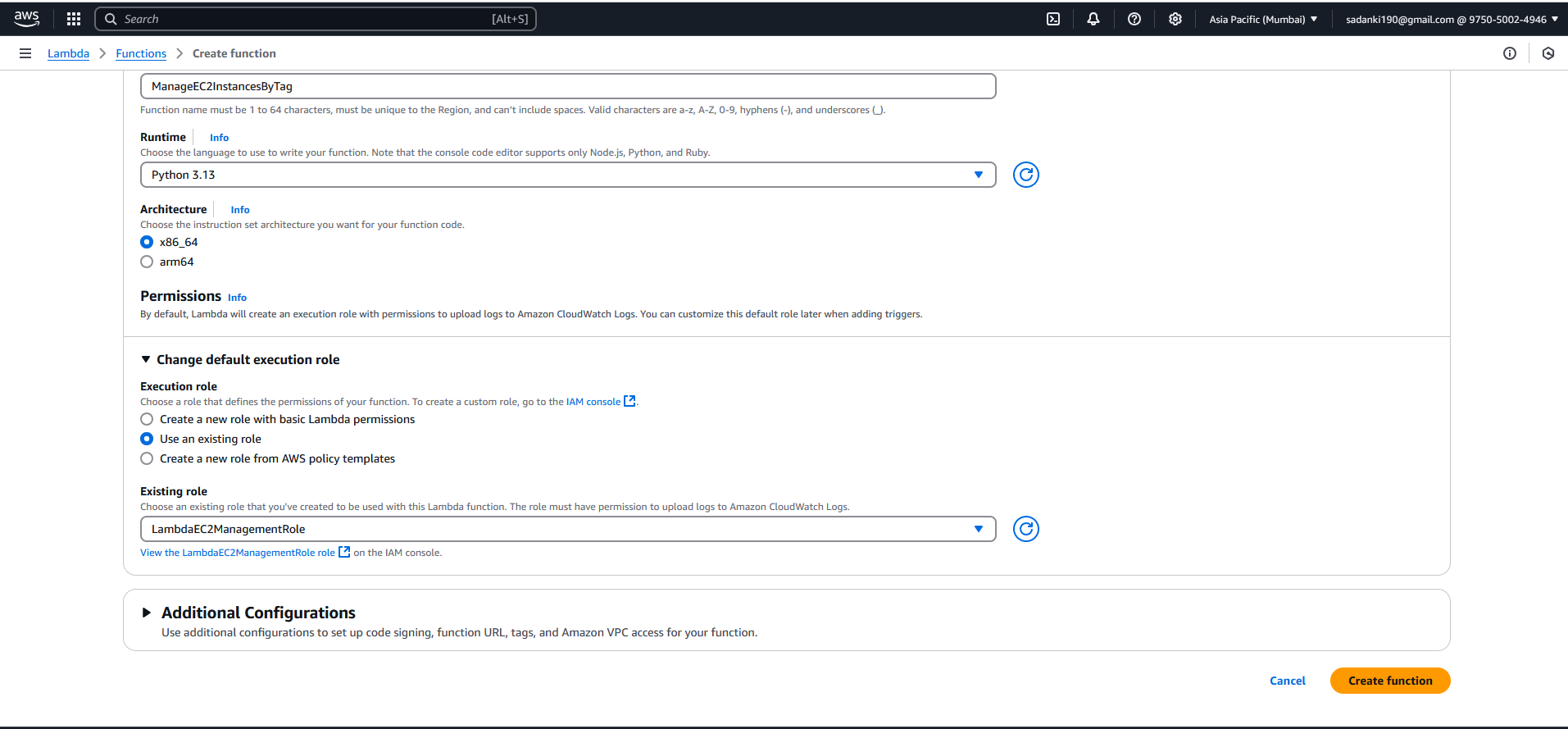


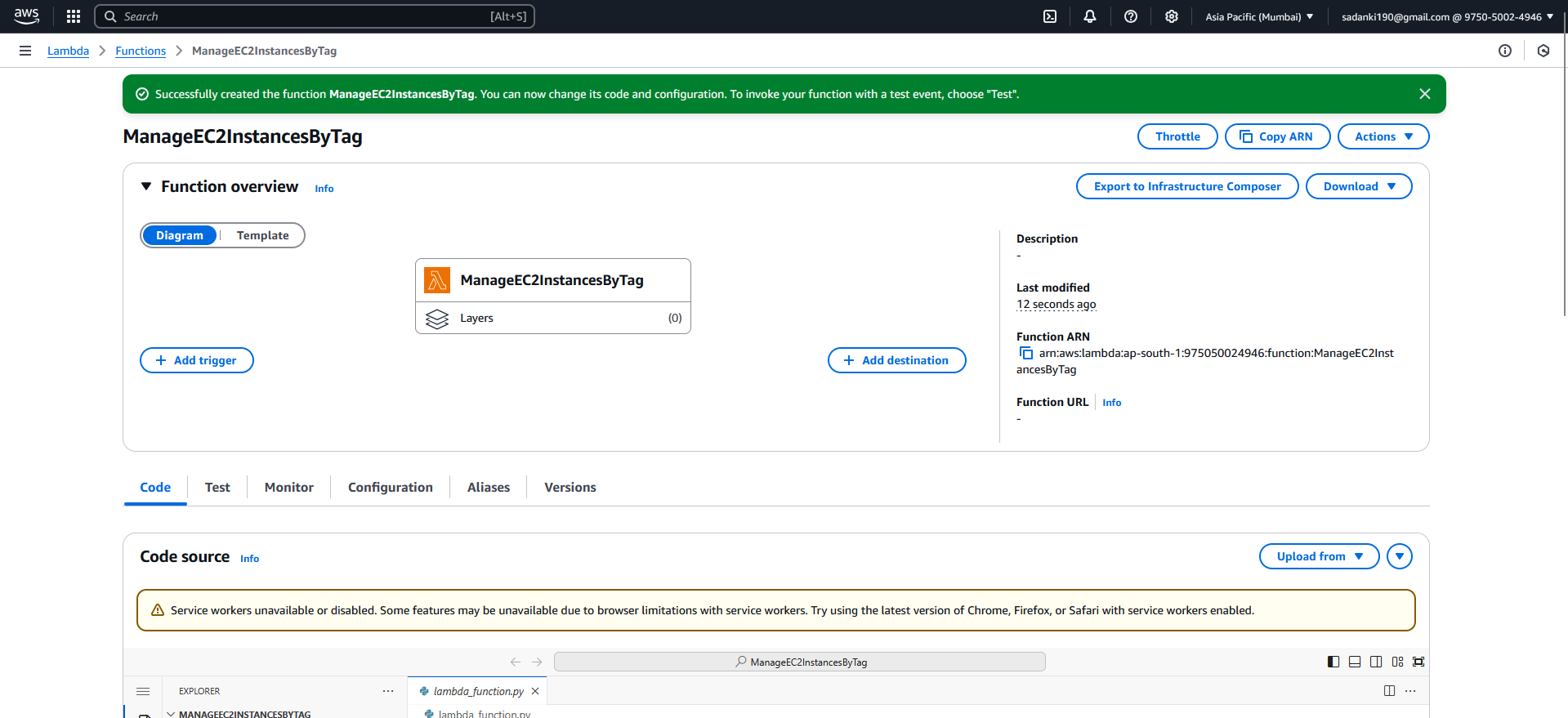
3.2: Configure Function:

| **Field** | **Value** |
| --- | --- |
| Function name | ManageEC2InstancesByTag |
| Runtime | Python 3.12 (or 3.10 / 3.9) |
| Execution role | Use an existing role |
| Existing role | LambdaEC2ManagementRole (select the role you created) |

Click **Create function**



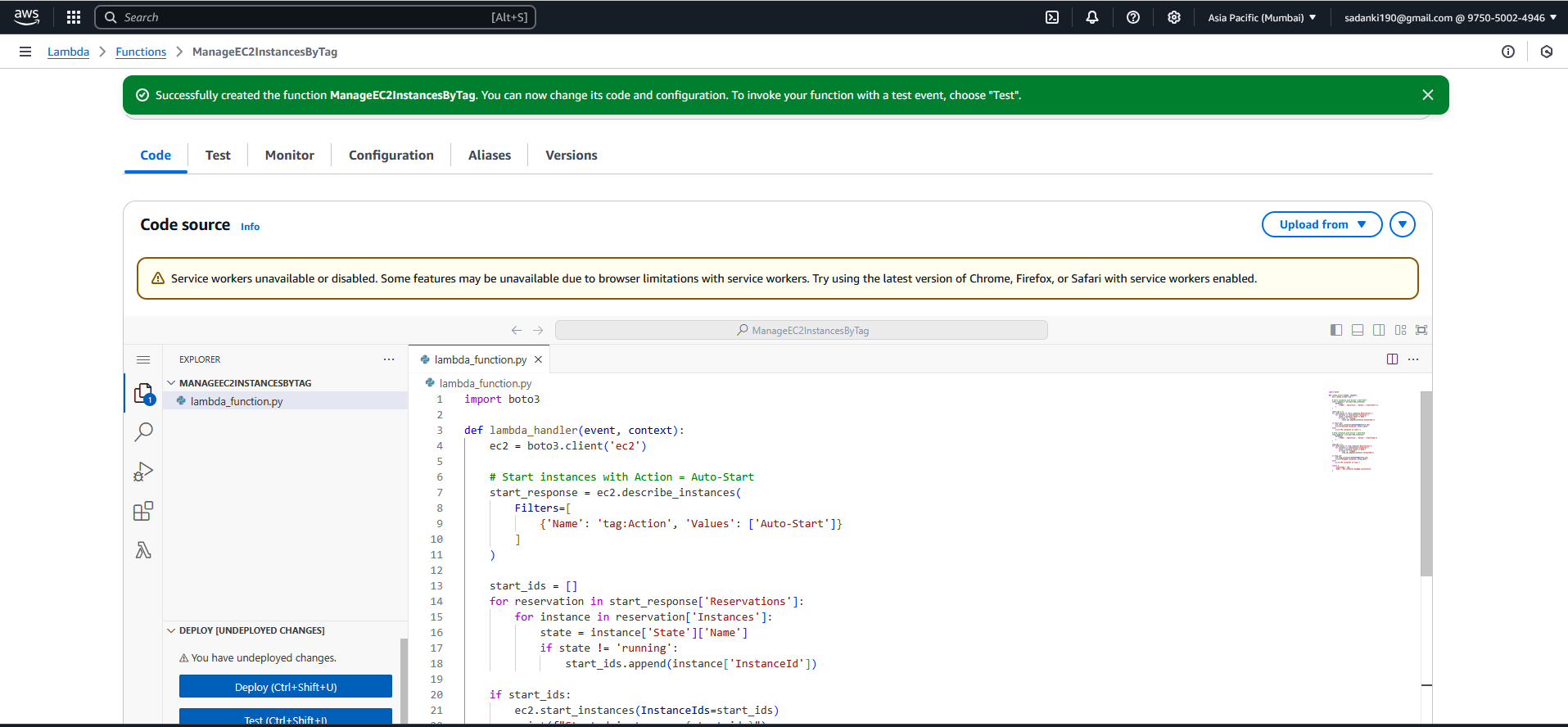


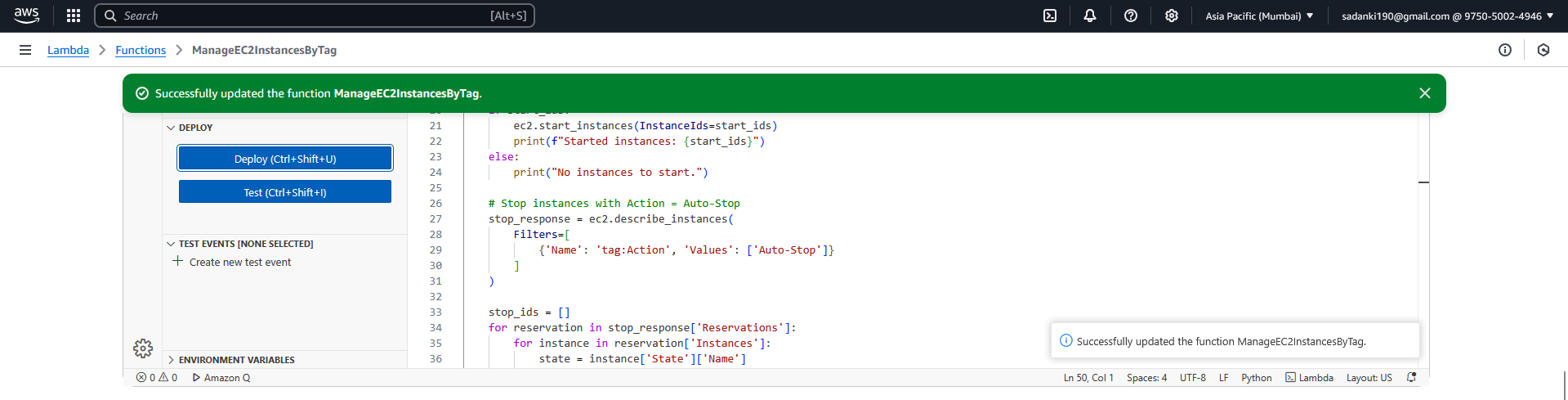


**3.3: Add the Code**

1. Scroll down to the **Code source** editor
2. Replace the default code with this:
3. import boto3
4. def lambda\_handler(event, context):
5. ec2 = boto3.client('ec2', region\_name='ap-south-1')
6. # Get all instances
7. instances = ec2.describe\_instances()
8. auto\_start\_ids = []
9. auto\_stop\_ids = []
10. for reservation in instances['Reservations']:
11. for instance in reservation['Instances']:
12. tags = {tag['Key']: tag['Value'] for tag in instance.get('Tags', [])}
13. instance\_id = instance['InstanceId']
14. if tags.get('Action') == 'Auto-Start':
15. ec2.start\_instances(InstanceIds=[instance\_id])
16. auto\_start\_ids.append(instance\_id)
17. elif tags.get('Action') == 'Auto-Stop':
18. ec2.stop\_instances(InstanceIds=[instance\_id])
19. auto\_stop\_ids.append(instance\_id)
20. return {
21. 'statusCode': 200,
22. 'body': {
23. 'message': 'EC2 instances managed successfully',
24. 'started\_instances': auto\_start\_ids,
25. 'stopped\_instances': auto\_stop\_ids
26. }
27. }

Click **Deploy** after pasting the code.





3.4: Test the Lambda Function

Click the **Test** tab (next to Code).

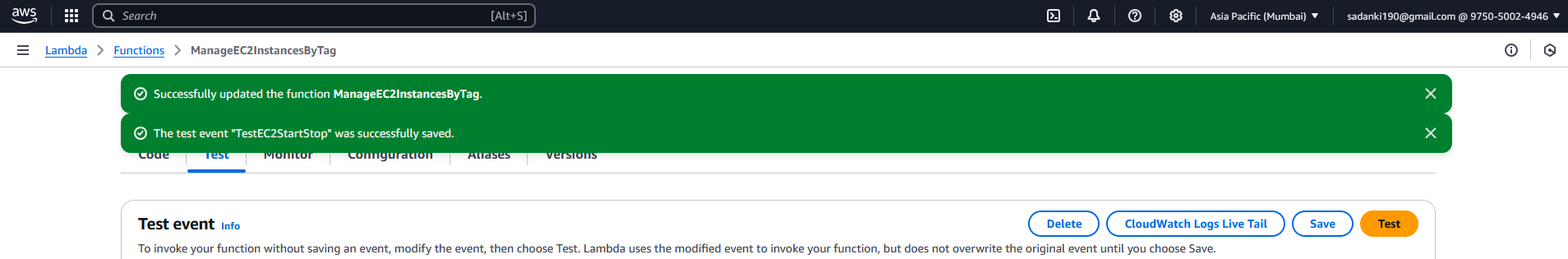
Configure a test event:

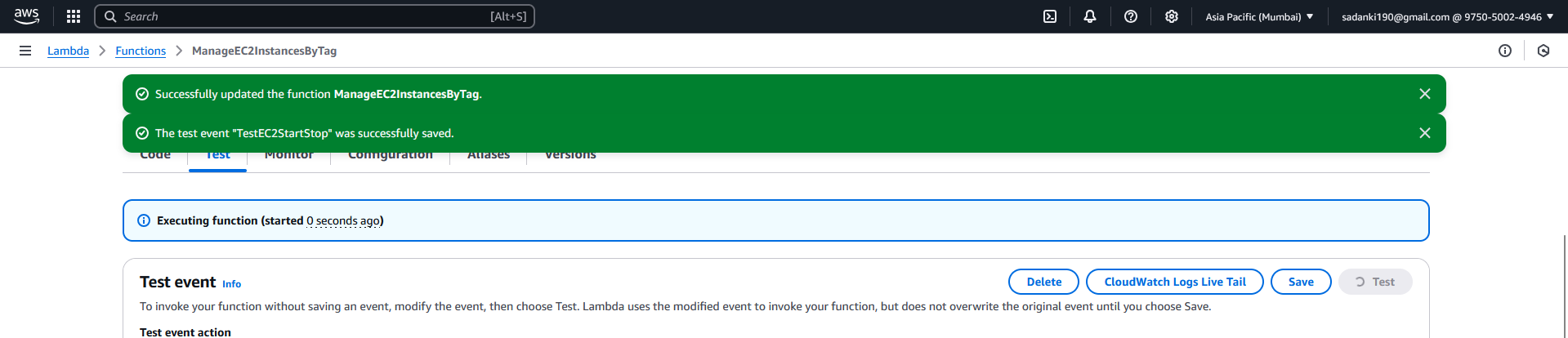
* Event name: TestEC2StartStop
* Event sharing setting: Private
* Leave the default JSON as-is:

{}

Click **Save and Test**

****

****

****

**What to Expect:**

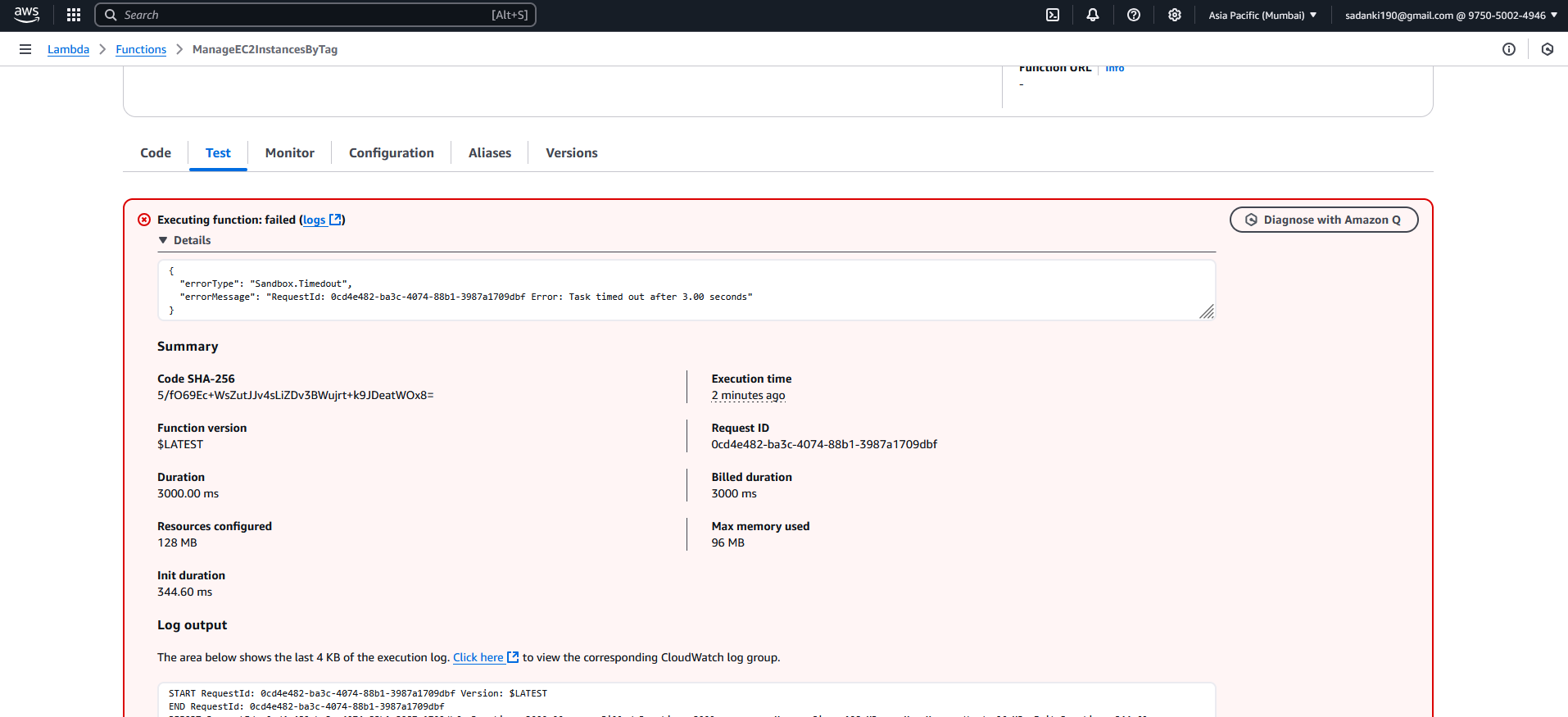
* If AutoStartInstance was stopped → it will **start**
* If AutoStopInstance was running → it will **stop**
* Logs will show which instances were started/stopped

Go to **EC2 → Instances** and verify the state changes. This may take 30–60 seconds

If you see error message:

"errorType": "Sandbox.Timedout",

"errorMessage": "Task timed out after 3.00 seconds"



means your Lambda function **exceeded the default timeout**, which is **3 seconds** — this is common when starting/stopping EC2 instances, as these actions take a few seconds.

Fix: Increase the Lambda Timeout:

**Step-by-Step:**

1. Go to your Lambda function in the AWS console  
   <https://console.aws.amazon.com/lambda>
2. On the left, click **Configuration**
3. Then click **General configuration**
4. Click **Edit**
5. Under **Timeout**, change it from 3 seconds to **30 seconds** (or even 1 minute if you'd like extra buffer)
6. Click **Save**

**Now Try Again**

After updating the timeout:

* Go to the **Test** tab
* Click **Test** again

You should now see a **success response** like:

{

"statusCode": 200,

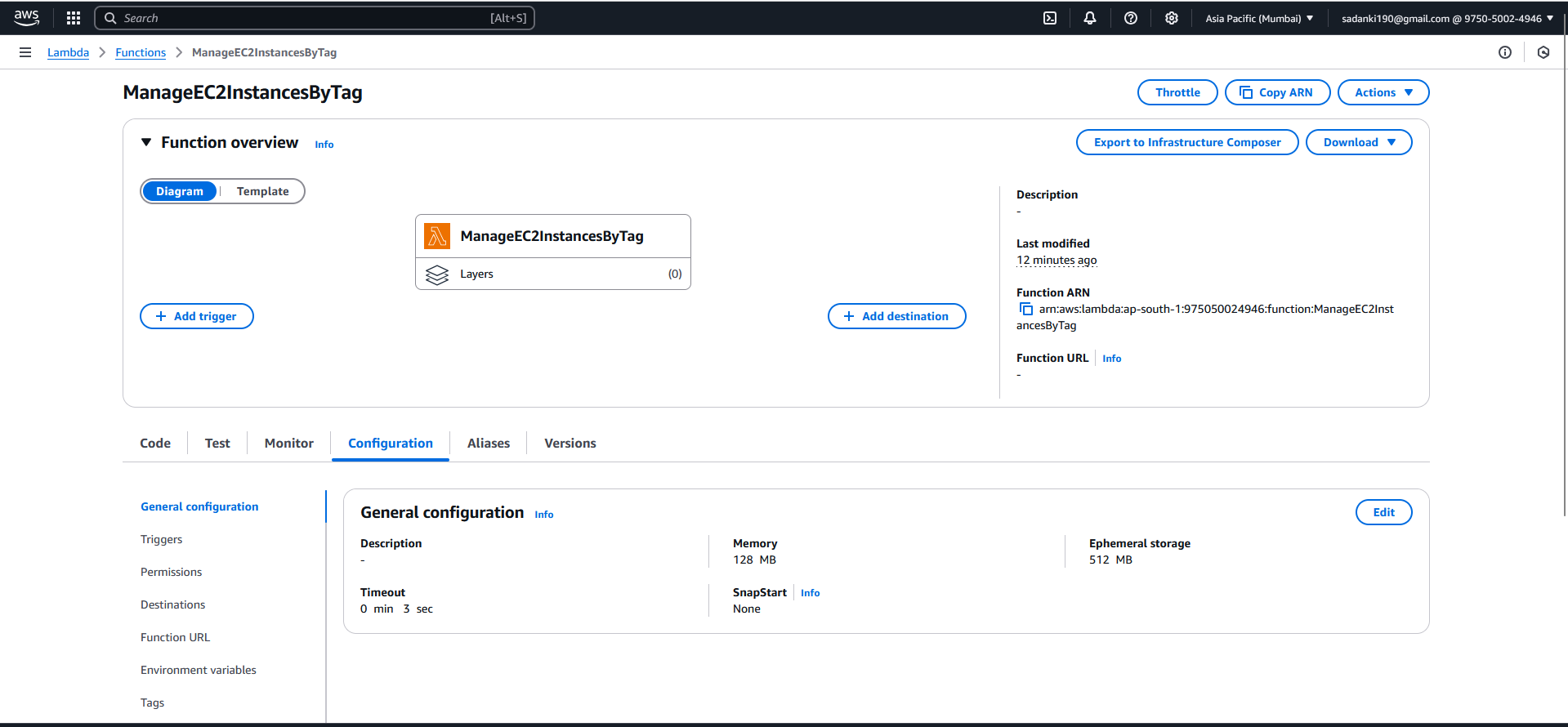
"body": "EC2 instances managed successfully"

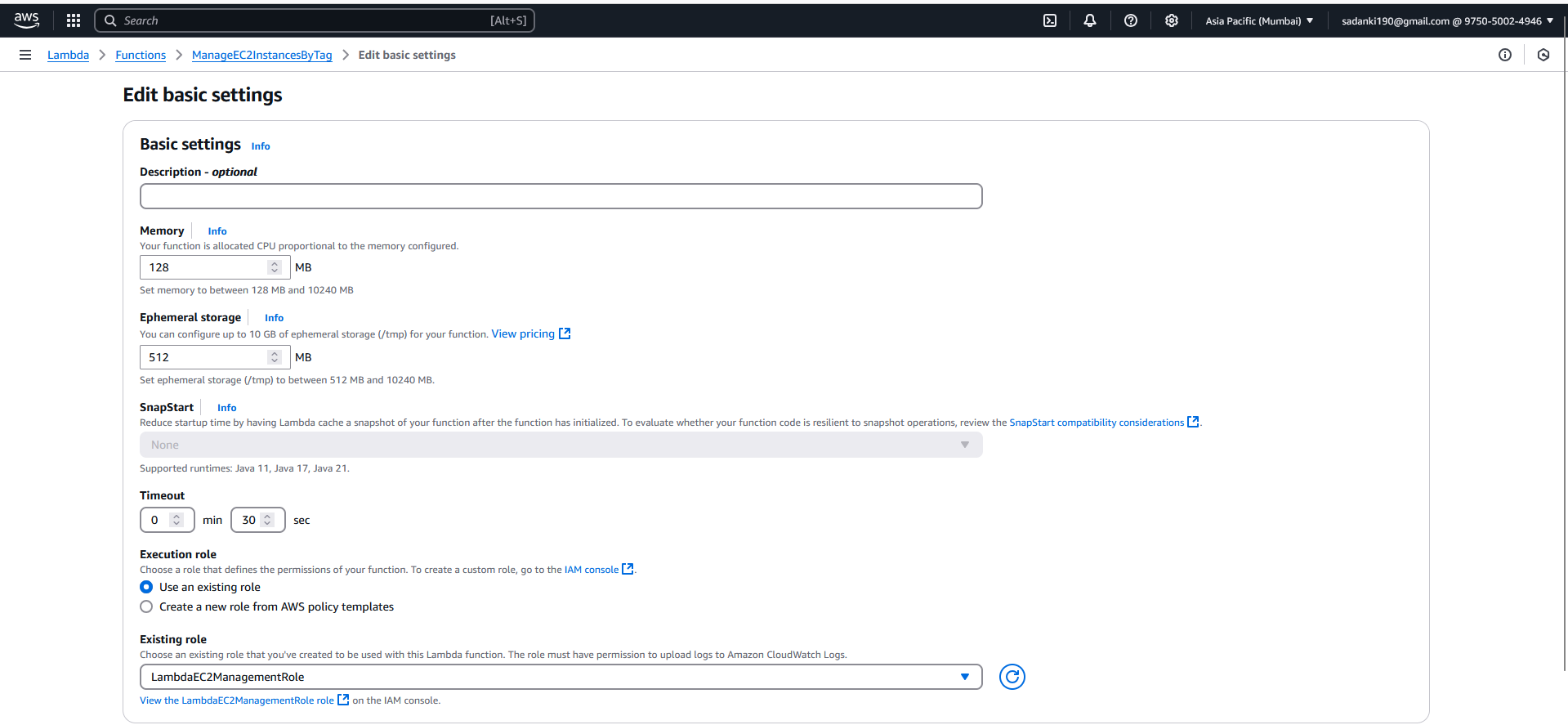
}

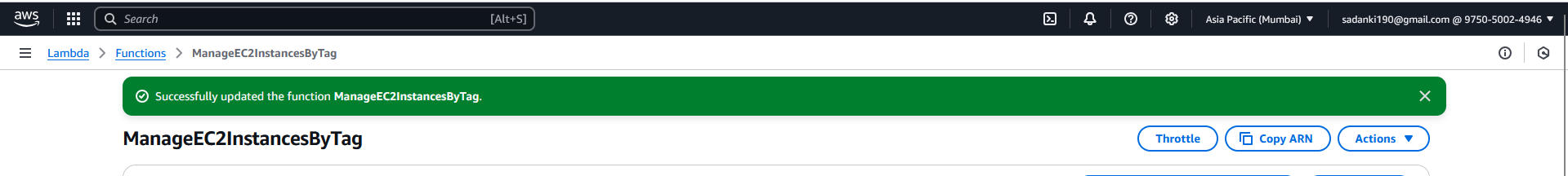
And in the logs:

Started instances: ['i-xxxxxxx']

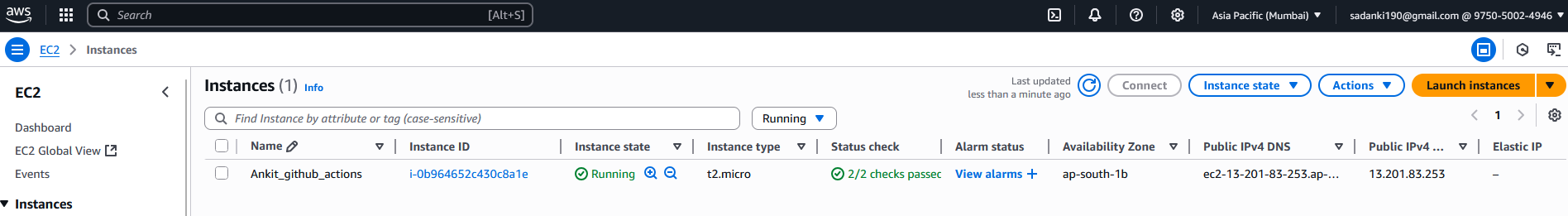
Stopped instances: ['i-yyyyyyy']



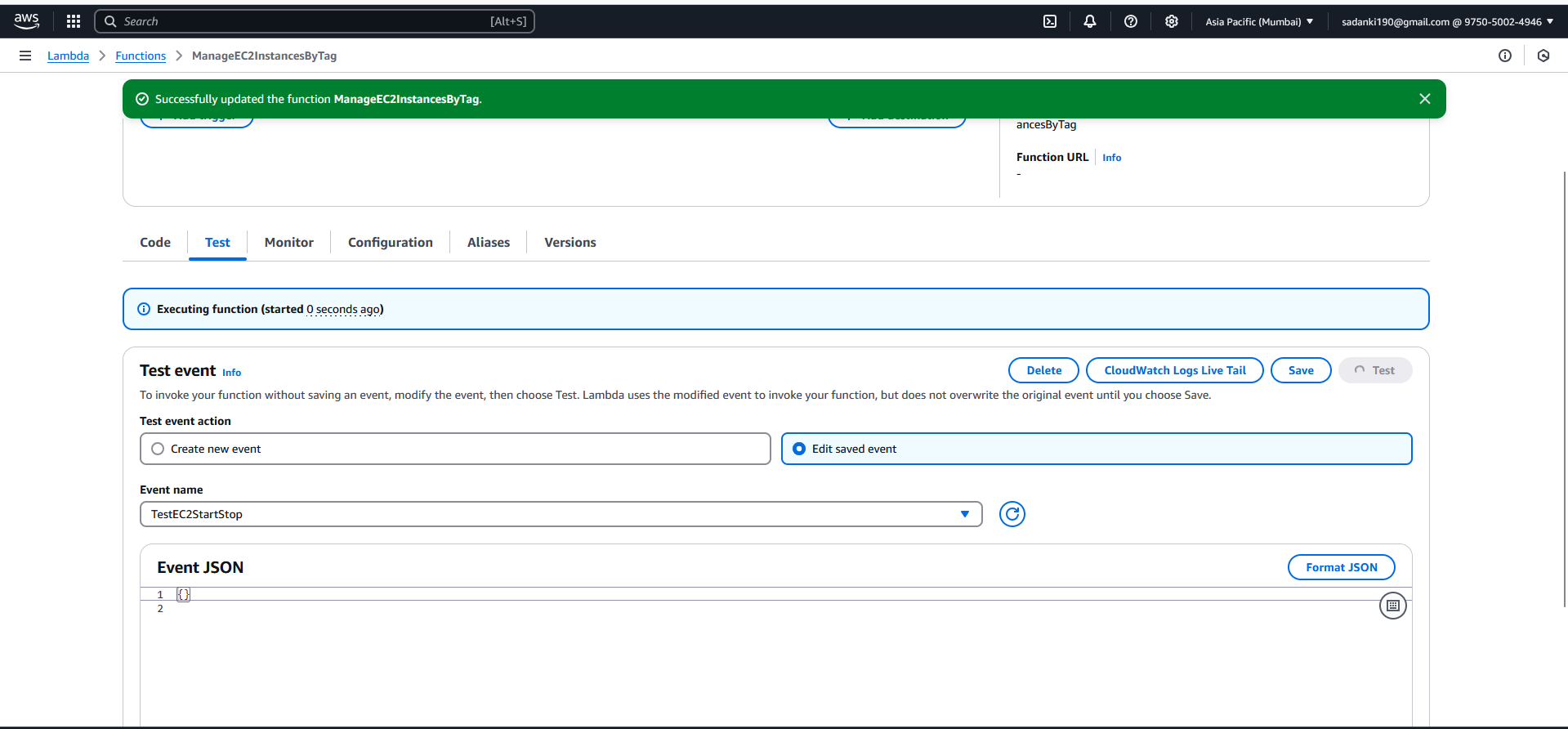


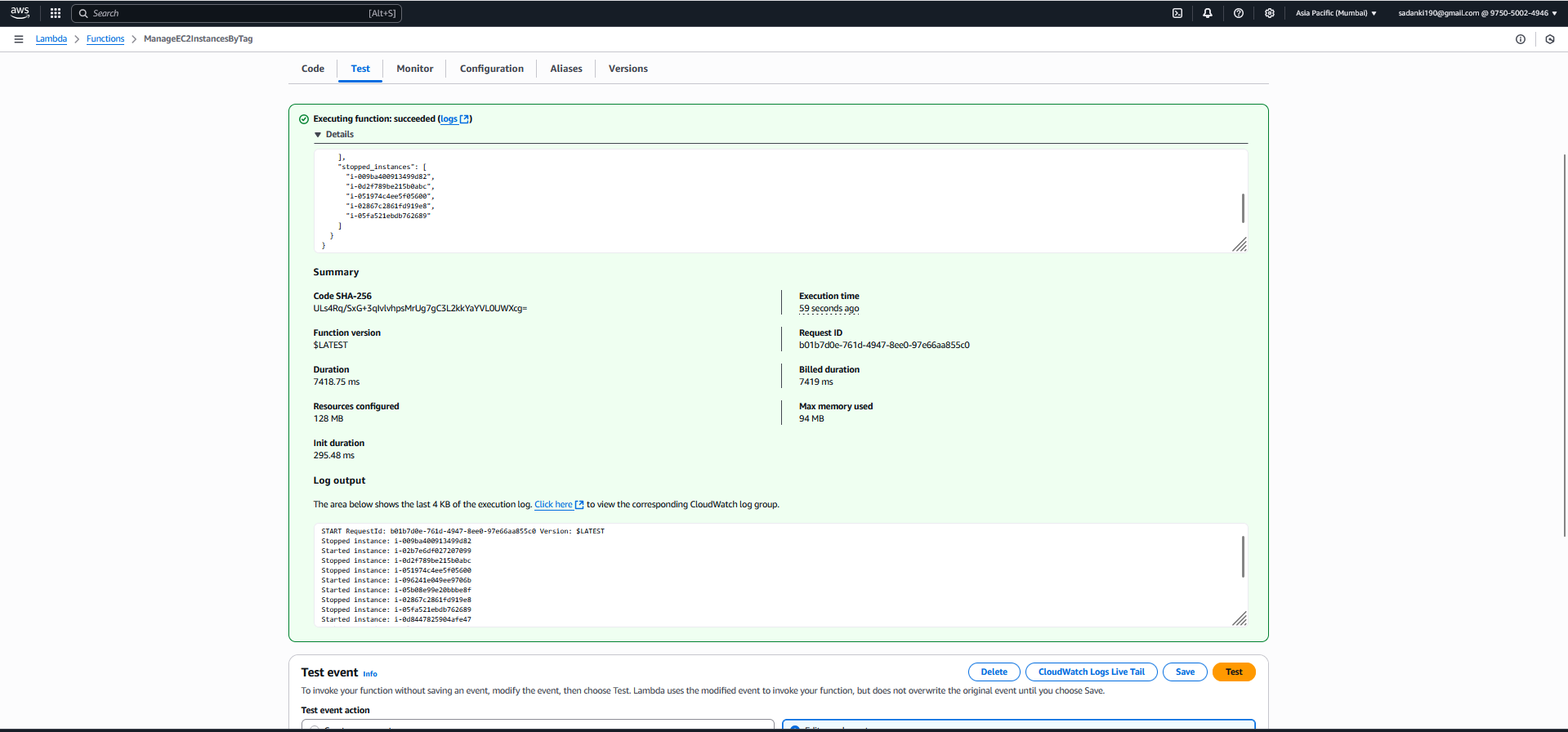


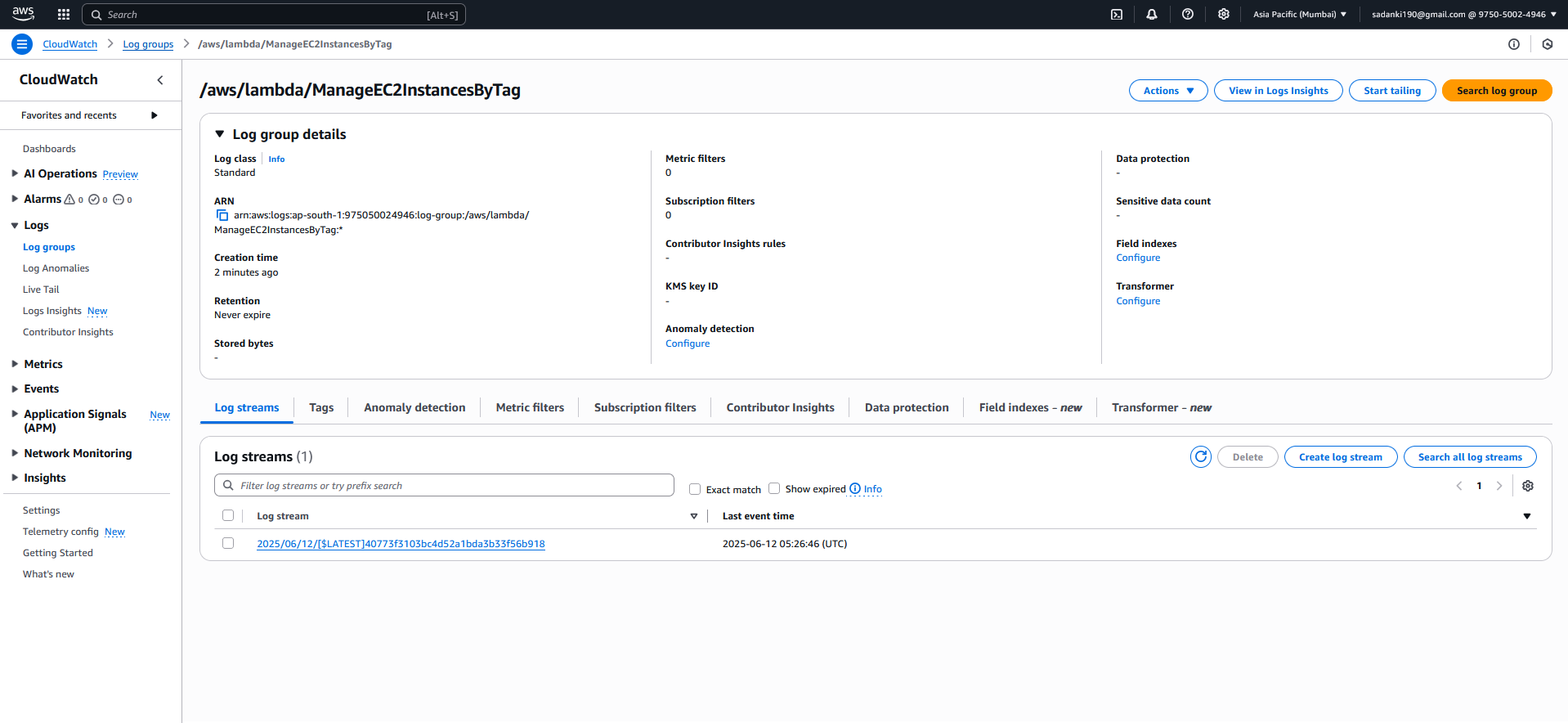
Currently both the instances are stopped and now let’s run the test in Lambda.

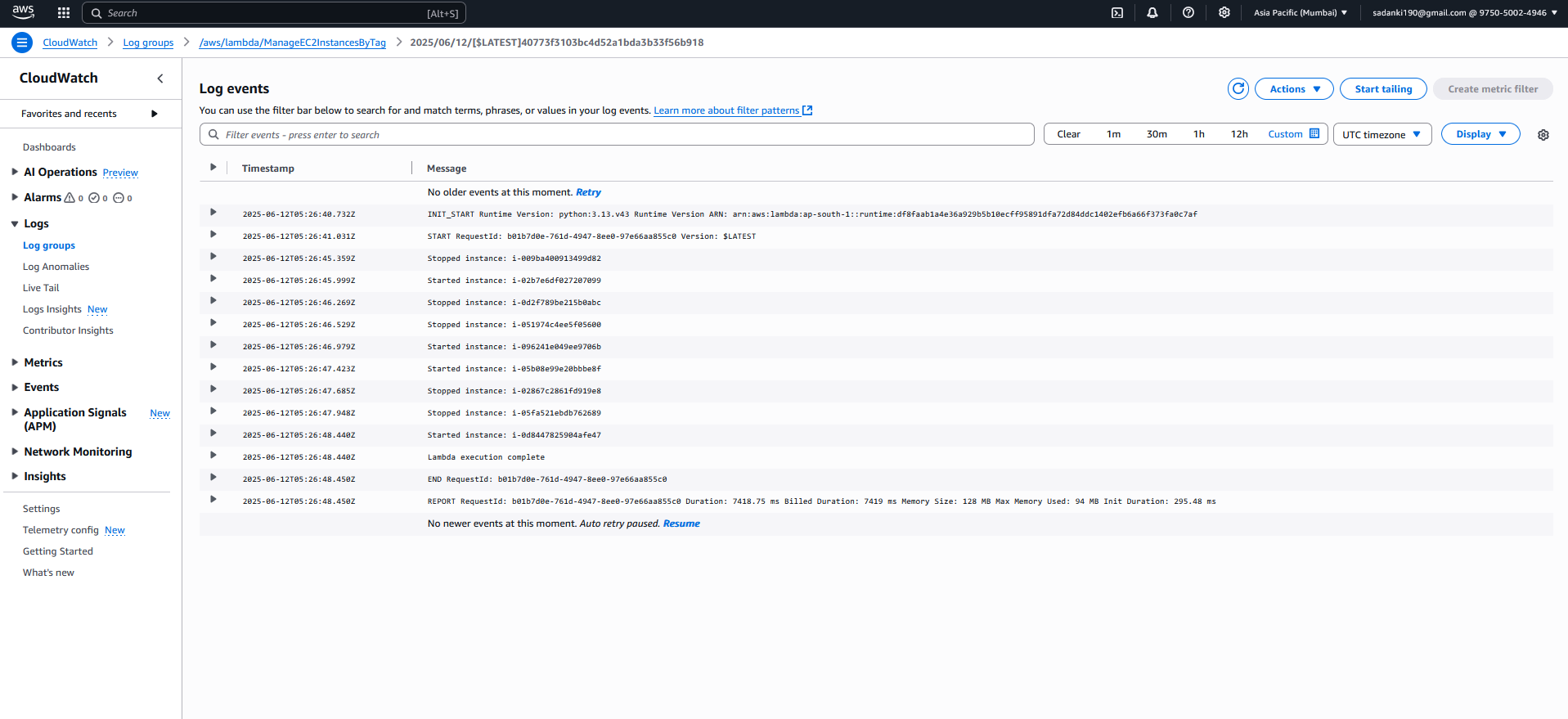


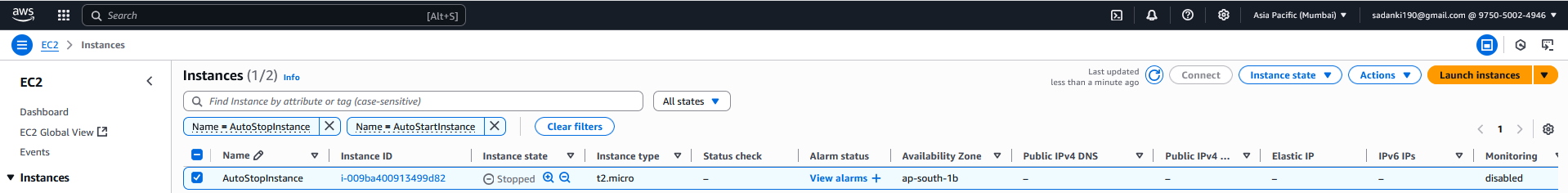
After clicking TEST:











This output means your Lambda function executed successfully and managed your EC2 instances as intended.

Response body:

{

"statusCode": 200,

"body": {

"message": "EC2 instances managed successfully",

"started\_instances": [

"i-02b7e6df027207099",

"i-096241e049ee9706b",

"i-05b08e99e20bbbe8f",

"i-0d8447825904afe47"

],

"stopped\_instances": [

"i-009ba400913499d82",

"i-0d2f789be215b0abc",

"i-051974c4ee5f05600",

"i-02867c2861fd919e8",

"i-05fa521ebdb762689"

]

}

}

"message": "EC2 instances managed successfully" — confirms the function completed without errors.

"started\_instances": ["i-02b7e6df027207099"] — this instance (tagged Auto-Start) was **started** by the Lambda function.

"stopped\_instances": ["i-009ba400913499d82"] — this instance (tagged Auto-Stop) was **stopped** by the Lambda function.

Logs:

START RequestId: 0a57afe9-7f93-4da0-b313-08f1ac5a280d Version: $LATEST

END RequestId: 0a57afe9-7f93-4da0-b313-08f1ac5a280d

REPORT RequestId: 0a57afe9-7f93-4da0-b313-08f1ac5a280d Duration: 1514.16 ms Billed Duration: 1515 ms Memory Size: 128 MB Max Memory Used: 95 MB

START and END mark the start and finish of your Lambda execution.

REPORT shows how long the function ran (about 1.5 seconds), the memory used, and billing info.

**Summary:**

Lambda function detected that:

* The instance tagged for **Auto-Start** was stopped, so it started it.
* The instance tagged for **Auto-Stop** was running, so it stopped it.

Everything ran smoothly without errors.

**Assignment 1: Automated Instance Management Using AWS Lambda and Boto3** — including its **purpose**, **benefits**, and **limitations**.

**Assignment Overview**

**Title:** Automated Instance Management Using AWS Lambda and Boto3

**Objective:**  
To automatically **start or stop EC2 instances** based on tags using **AWS Lambda** and the **Boto3** SDK for Python.

**Why do we need this project?**

In many cloud-based environments, EC2 instances are left running even when they’re not needed — overnight, during weekends, or off-hours. This causes **unnecessary costs**.

**The solution:**

This project automates EC2 instance management based on **tags** (like Auto-Start or Auto-Stop) using serverless Lambda functions.

You **save money** and **increase efficiency** without relying on manual intervention or human error.

**Use Cases**

| **Use Case** | **Description** |
| --- | --- |
| Dev/Test Environments | Automatically stop development servers at night and start them in the morning. |
| Academic Projects | Helps students learn automation, serverless, and AWS SDK (Boto3). |
| Cost Optimization | Reduce AWS bills by stopping unused resources. |
| Security | Reduce attack surface by not keeping unnecessary instances running. |

**Advantages**

| **Advantage** | **Explanation** |
| --- | --- |
| **Automation** | Fully automates instance management — no human action needed. |
| **Cost Savings** | Prevents accidental billing by stopping unused instances. |
| **Serverless** | Uses AWS Lambda — no need to manage extra infrastructure. |
| **Permission Controlled** | Uses IAM roles for secure and controlled access to EC2. |
| **Tag-Based Flexibility** | Easy to extend — just tag more instances to include them. |
| **Hands-On Learning** | Teaches key AWS services: EC2, Lambda, IAM, and Boto3. |

**Disadvantages / Limitations**

| **Limitation** | **Explanation** |
| --- | --- |
| **Delay in Scheduling** | Lambda must be triggered on a schedule or manually. Real-time control may require more complexity. |
| **Security Risks if Over-Permissioned** | Using AmazonEC2FullAccess is insecure in production (least-privilege is better). |
| **Tagging Errors** | Mistagged instances won’t be managed correctly. |
| **Limited Custom Logic** | More advanced logic (e.g., conditional start/stop) would need extra development. |

**What you learn**

* **Python with Boto3** to interact with AWS programmatically.
* **AWS Lambda** for serverless automation.
* **IAM Roles** and permission management.
* **EC2 Management** and instance tagging best practices.
* **Testing and Debugging AWS functions** via logs and CloudWatch.

**Summary**

This assignment simulates a **real-world DevOps use case** and teaches how to build **intelligent cloud automation**. While basic, it's a **foundational project** in understanding how to reduce cloud costs and improve system operations using AWS services.