# AWS Training – assignments

* IAM

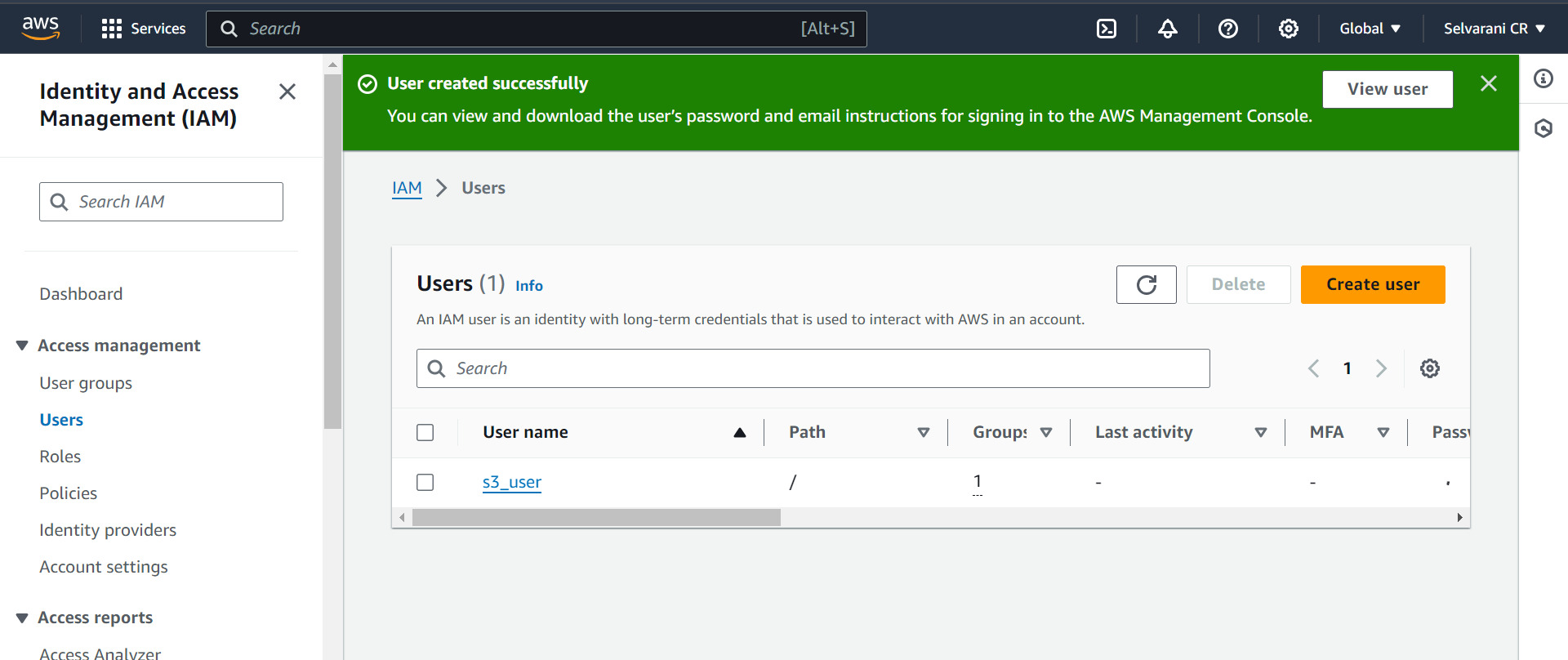
1. Go to the AWS Management Console and open the IAM service.
2. In the left navigation pane, click on "Users".
3. Click "Add user".
4. Enter a username.
5. Select "Programmatic access" for AWS Management Console access.
6. Click "Next: Permissions".

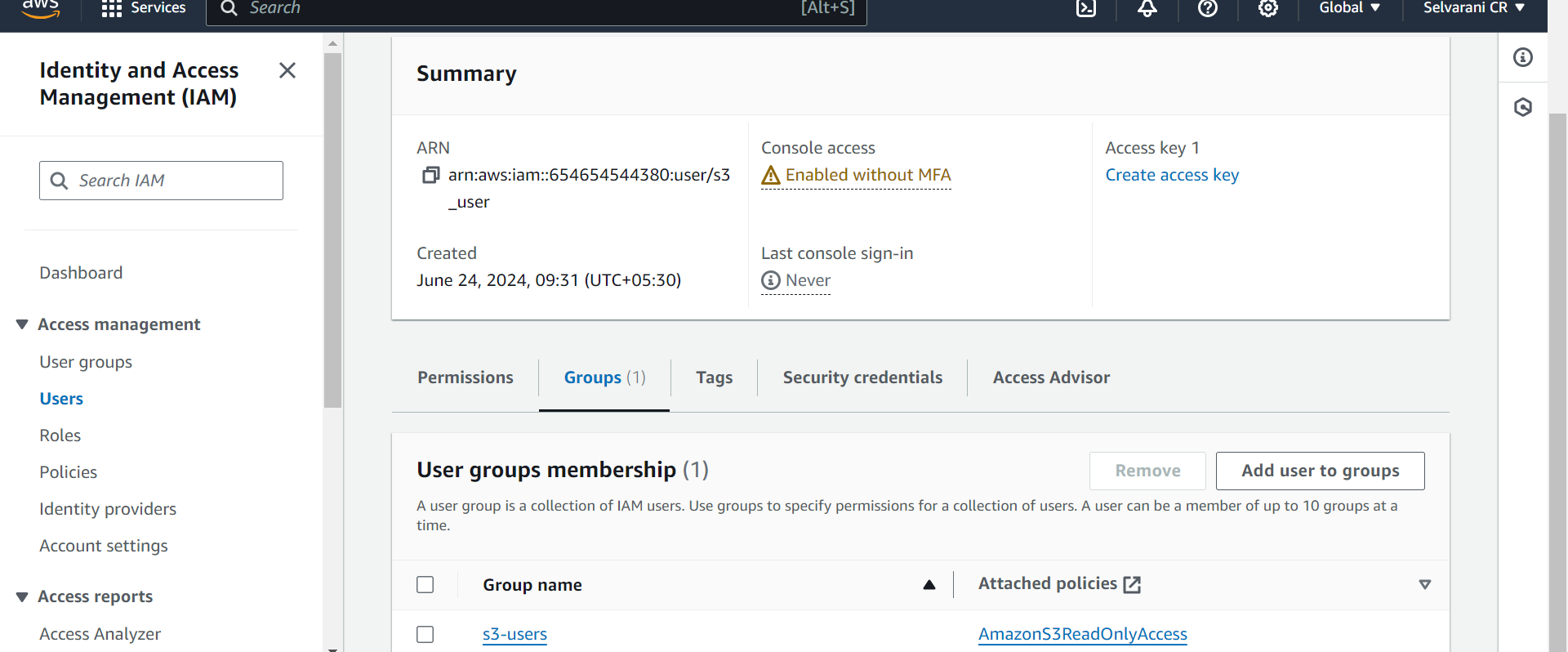
**Create a Group and Attach Policy:**

1. On the "Add group" page, click "Create group".
2. Enter a group name.

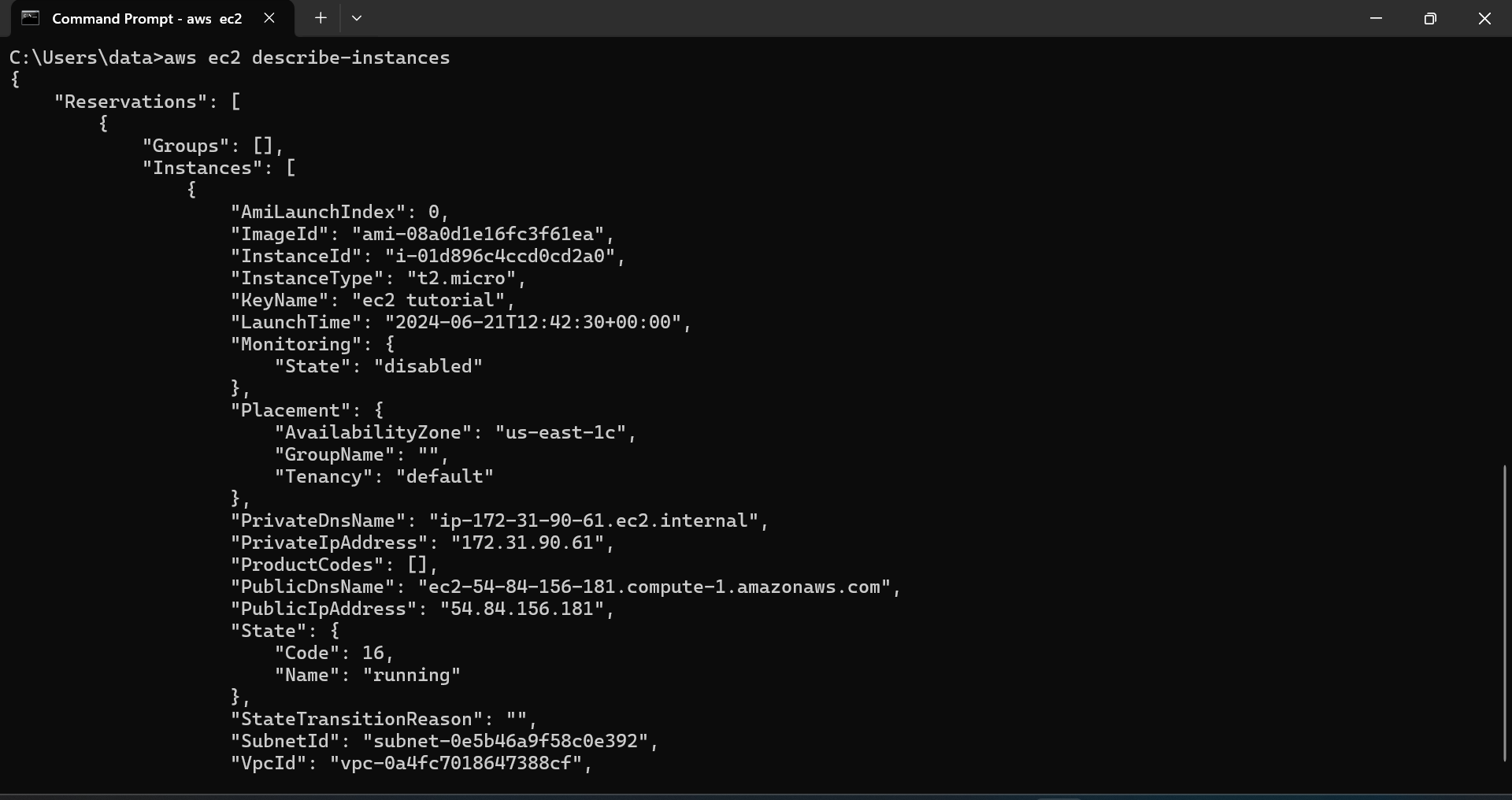
* Add s3readonly policy to the group.

10. Add the user to that group.



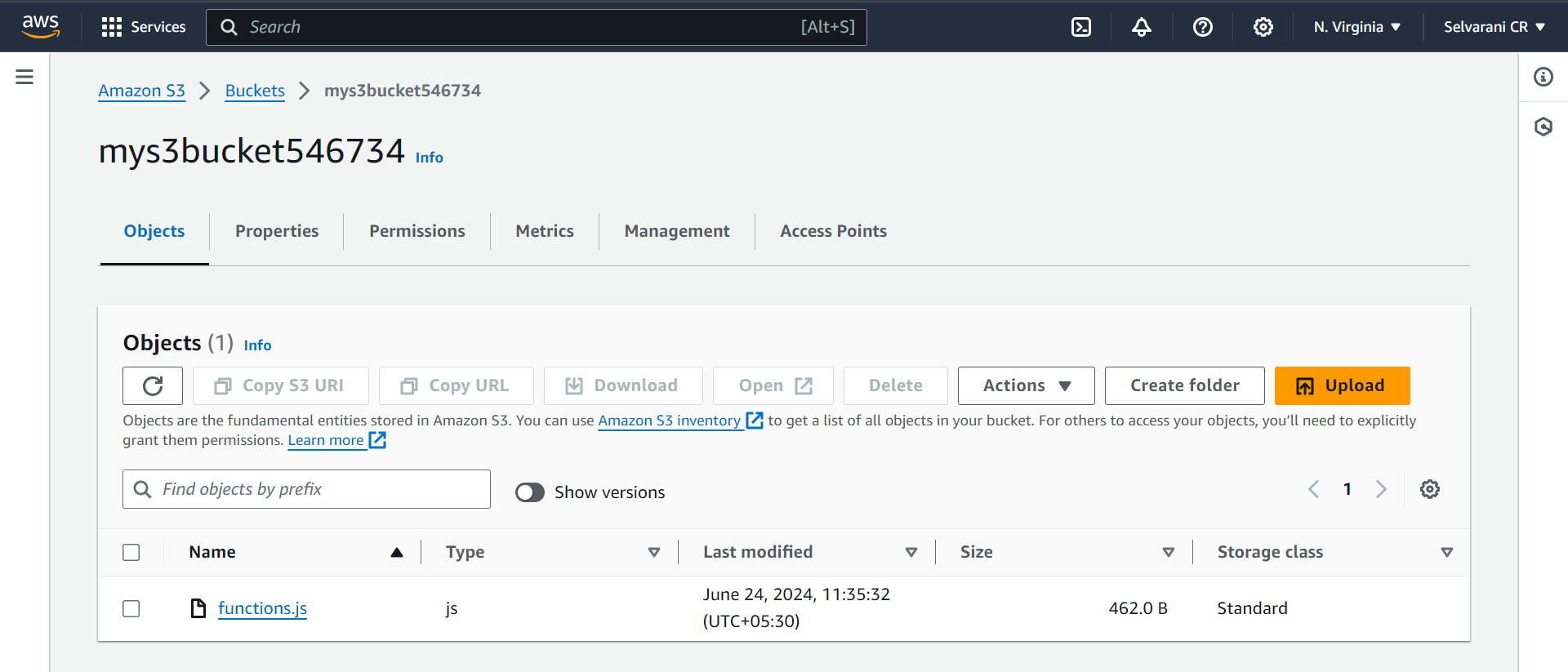
* 
* **Create a role and Assume it :**
* Create IAM Role
* Log in to AWS Management Console.
* Navigate to IAM console and click on Roles.
* Click Create role.
* Select AWS service as the type of trusted entity.
* Choose EC2 as the service that will use this role.
* Attach policy AmazonEC2ReadOnlyAccess.
* Review and create the role.
* Open command line prompt, and configure the user.
* Then assume the role using this command  
  aws sts assume-role --role-arn arn:aws:iam::654654544380:role/ec2instance\_read\_access --role-session-name ec2session
* Set the environment variables using the credentials given.
* Then list the ec2 instances using this command   
  aws ec2 describe-instances.

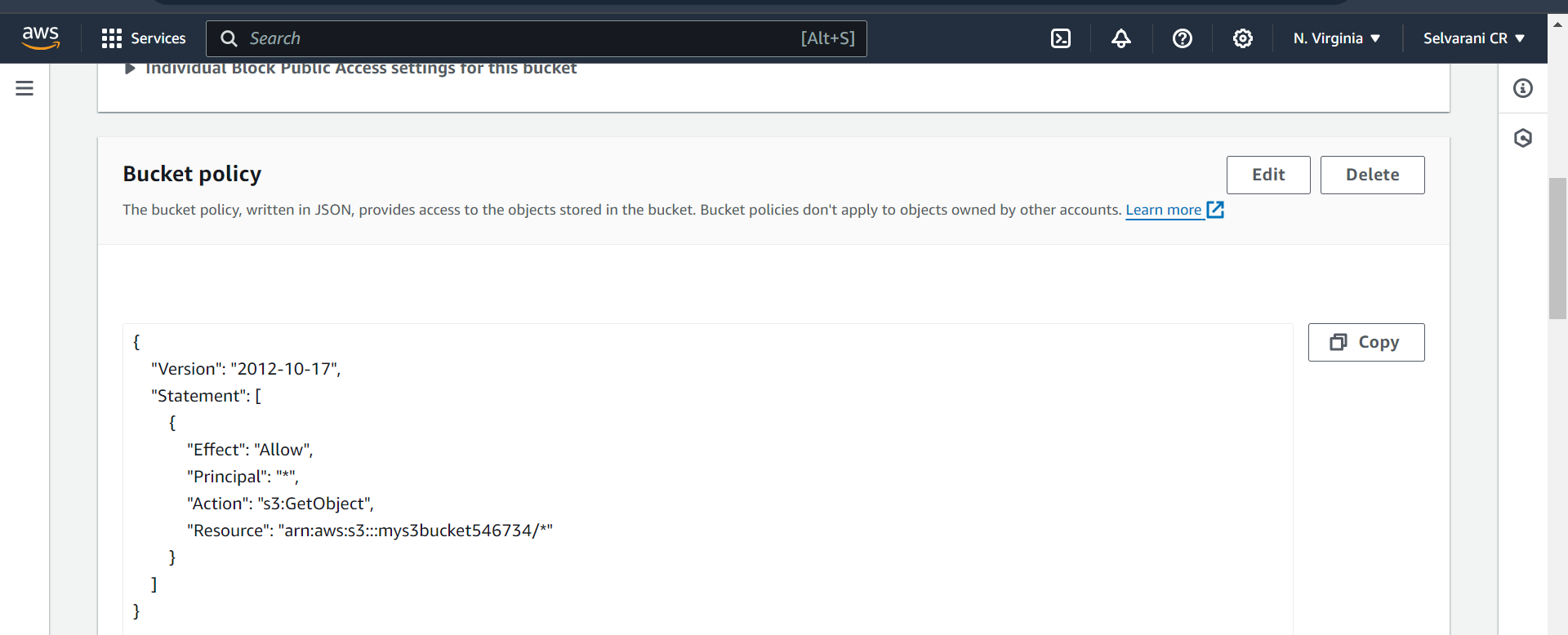


* 
* **Create an account b and access the ec2 instances of s3\_user from this account:**
* Log in with appropriate credentials for s3\_user.
* In the AWS Management Console, go to the IAM service.
* Create IAM Role
* Click on Roles in the left-hand navigation pane.
* Click Create role.
* Choose the trusted entity
* Select Another AWS account as the type of trusted entity.
* Enter Account B's AWS account ID.
* Attach Permissions.
* Attach the policy AmazonEC2ReadOnlyAccess
* Review and create the role, note the ARN of the role.
* Open command line and using   
  aws configure – configure using account b credentials.
* Assume the IAM role from s3\_user in account b  
  aws sts assume-role –role-arn arn:aws:iam::654654544380:role/ec2\_read\_only\_for\_accountb –role-session-name ec2-session
* Set the environment variables.
* List the ec2 instances using this command  
  aws ec2 describe-instances –region us-east-1.

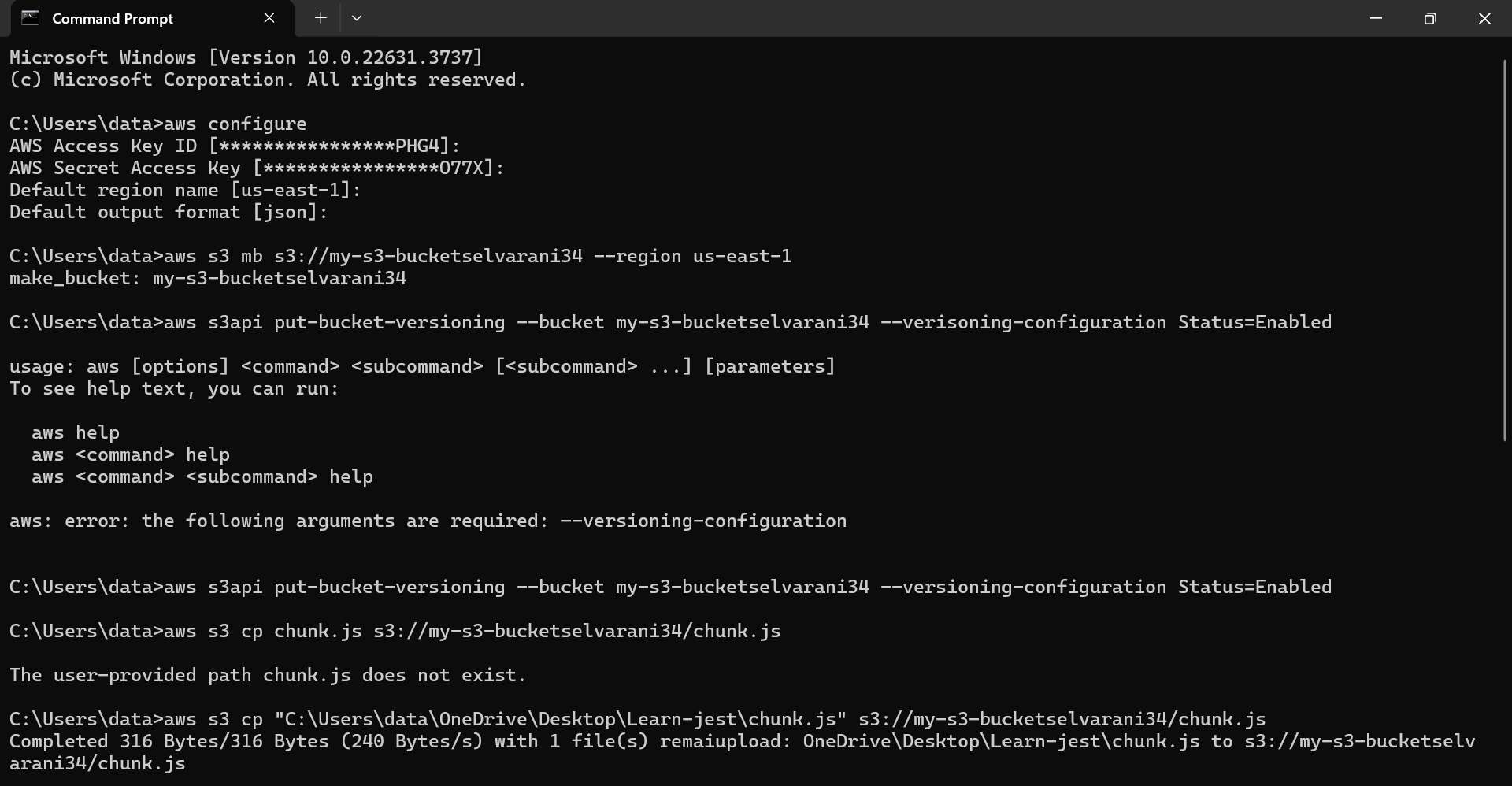
S3

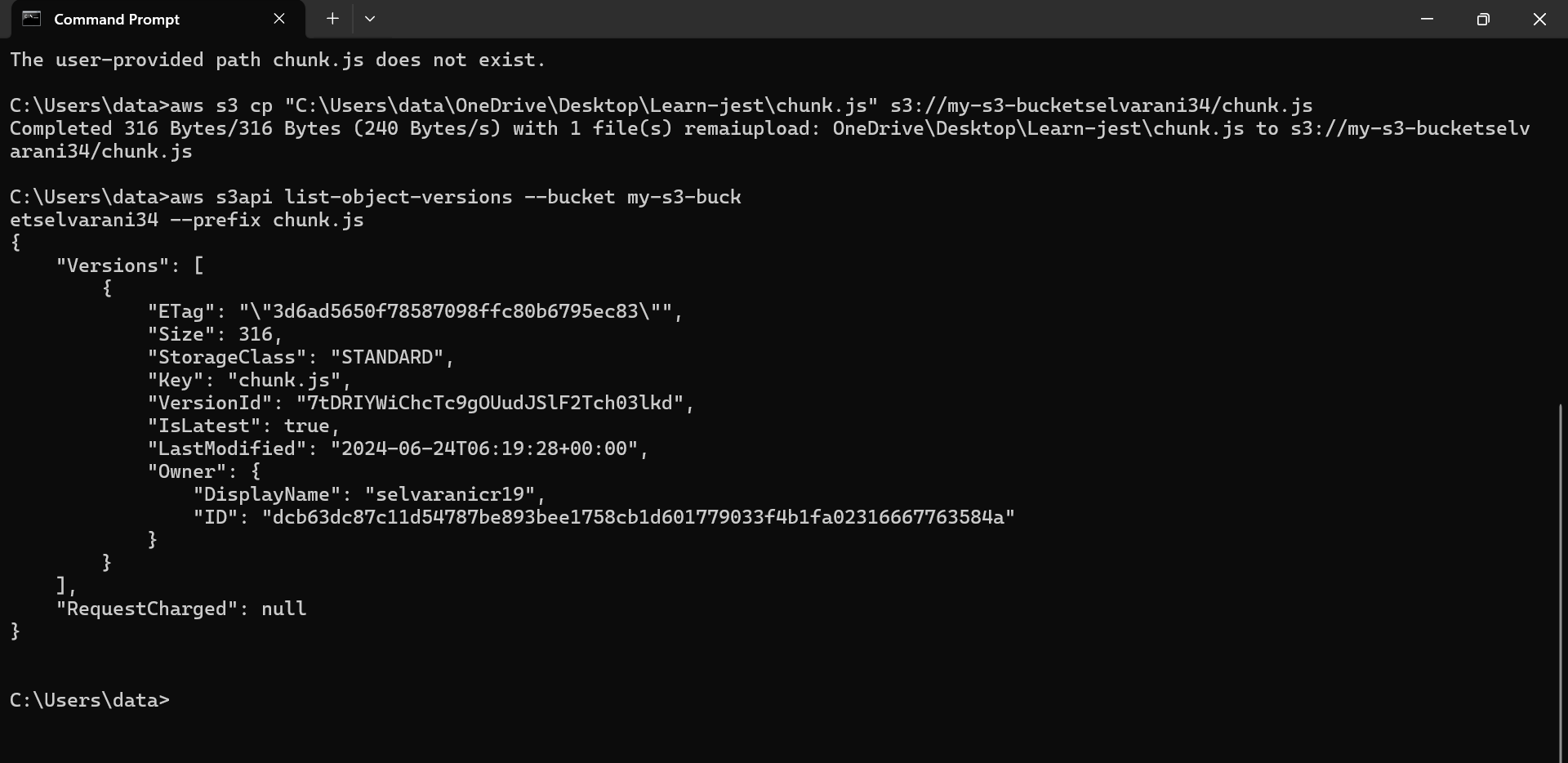
* **Create an S3 bucket and put objects, add versioning and access policy:**
* Navigate to the S3 service.
* Click on Create bucket.
* Enter a unique bucket name and choose a region.
* Click Create to create the bucket.
* **Upload Objects to the Bucket:**
* Select the bucket you created.
* Click Upload to upload files from your local machine to the bucket.
* Select files and upload them.
* **Enable Versioning:**
* In the S3 Management Console, select the bucket you created earlier.
* Go to the Properties tab.
* Click on Object Versioning.
* Click Enable versioning.
* Access policies in S3 define who can access your bucket and how they can access it.
* **Creating a Bucket Policy:**
* Navigate to the bucket in the S3 Management Console.
* Go to the Permissions tab.
* Click on Bucket Policy and add the policy in json format and save.





AWS CLI and boto3

Create s3 bucket, enable versioning, upload files and lists files with versions using cli  




CloudFormation

* Define the template version and description for the template.
* Parameters – BucketName, EnableVersioning, EnableAcceleration.
* Resources - Type: AWS::S3::Bucket
* Properties:
* BucketName: Uses the BucketName parameter value.
* Tags: Adds a tag with the key Name and value YourName.
* VersioningConfiguration: Sets the versioning status based on the EnableVersioning parameter.
* AccelerateConfiguration: Sets the acceleration status based on the EnableAcceleration parameter.

AWSTemplateFormatVersion: '2010-09-09'

Description: 'AWS CloudFormation template to create S3 bucket'

Parameters:

BucketName:

Description: 'Name of the bucket.'

Type: String

Default: ''

EnableVersioning:

Description: 'Enable versioning for the S3 bucket.'

Type: String

Default: 'Enabled'

AllowedValues:

- Enabled

- Disabled

EnableAcceleration:

Description: 'Enable Transfer Acceleration for the S3 bucket.'

Type: String

Default: 'Enabled'

AllowedValues:

- Enabled

- Suspended

Resources:

MyS3Bucket:

Type: 'AWS::S3::Bucket'

Properties:

BucketName: !Ref BucketName

Tags:

- Key: Selvarani

Value: MyS3Bucket

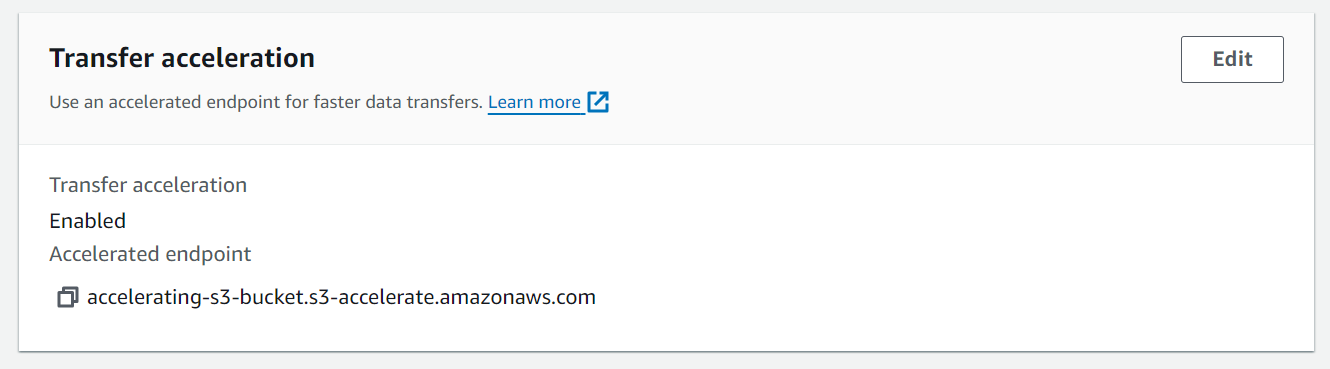
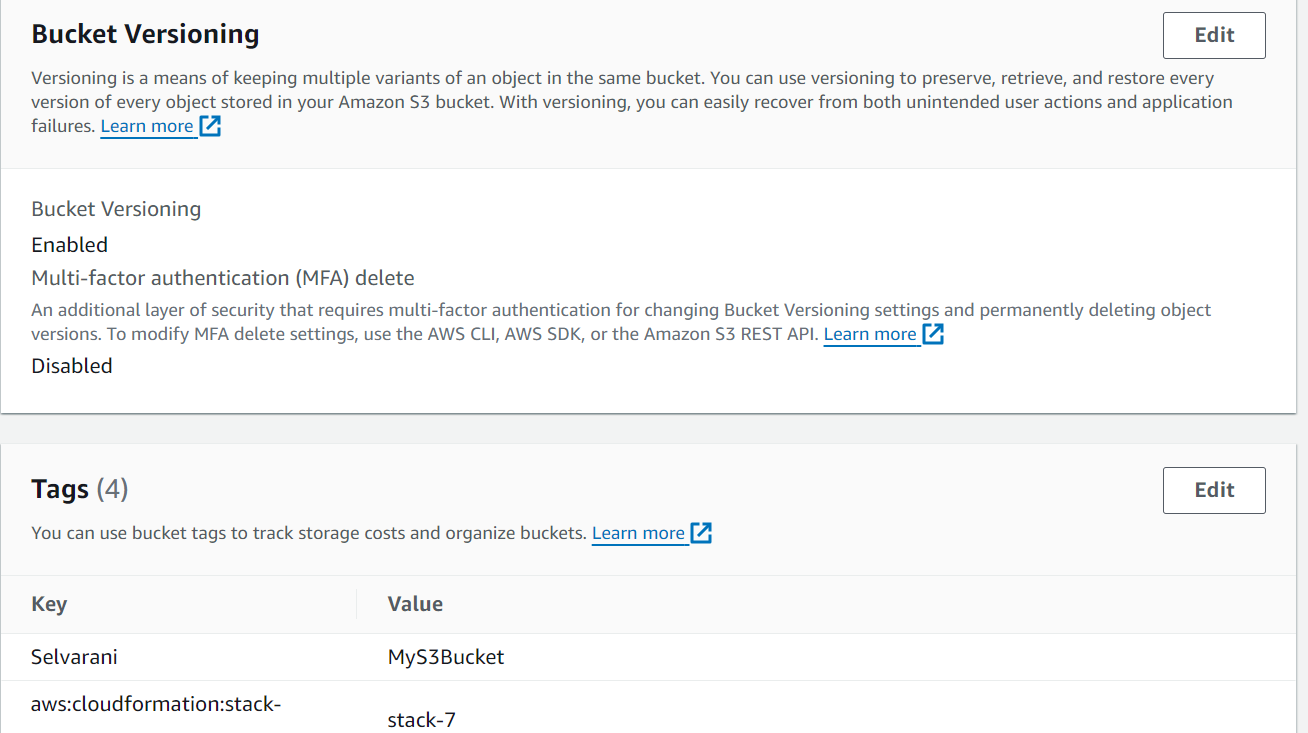
VersioningConfiguration:

Status: !Ref EnableVersioning

AccelerateConfiguration:

AccelerationStatus: !Ref EnableAcceleration

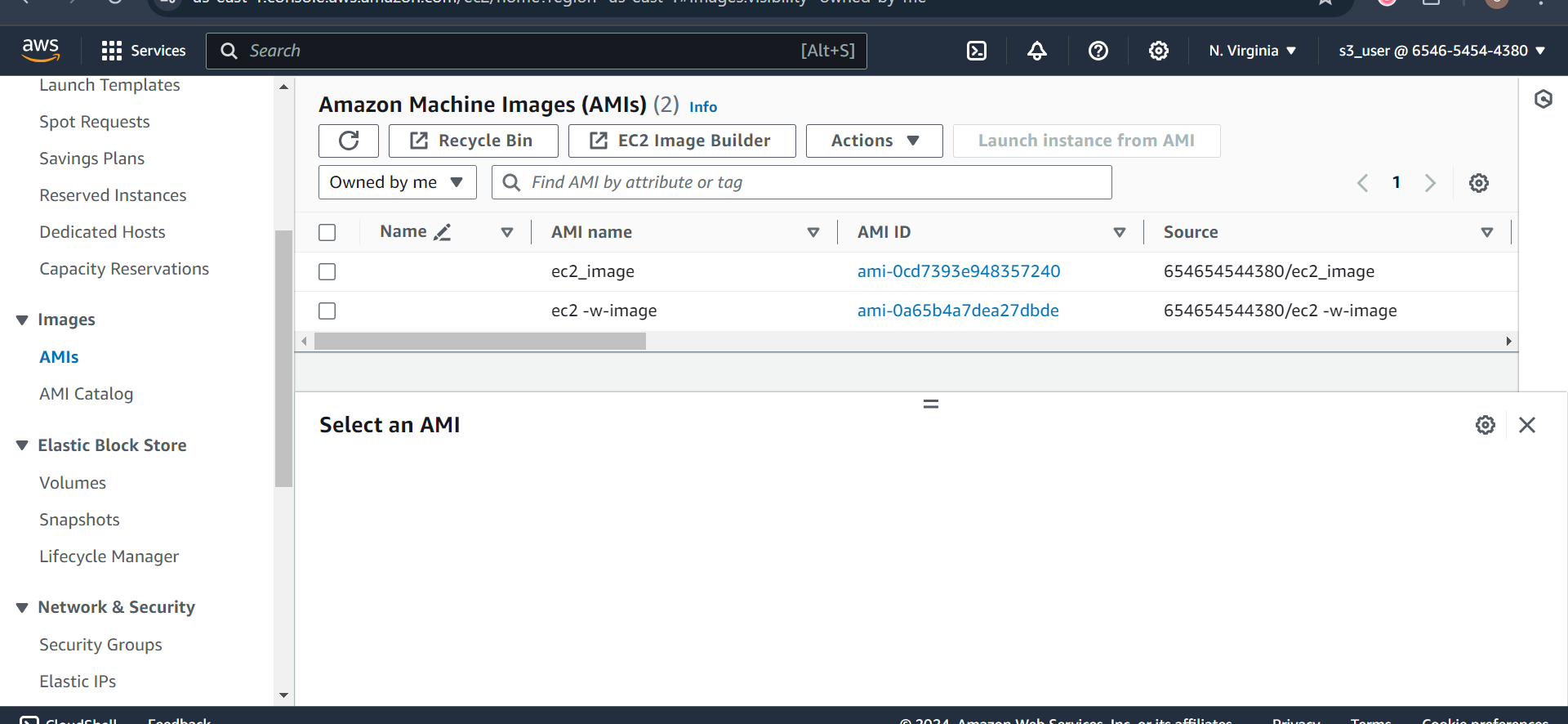
* In the CloudFormation dashboard, click on the "Create stack" button.
* In the specify template section, select upload template file and upload the file, click next.
* Specify stack name and bucket name, other parameters are given default value if not specified.
* Review and create stack.
* Monitor stack creation and once complete check the s3 bucket if created and parameters are correctly assigned.



EC2

Launch ec2 instances for both OS, linux and windows

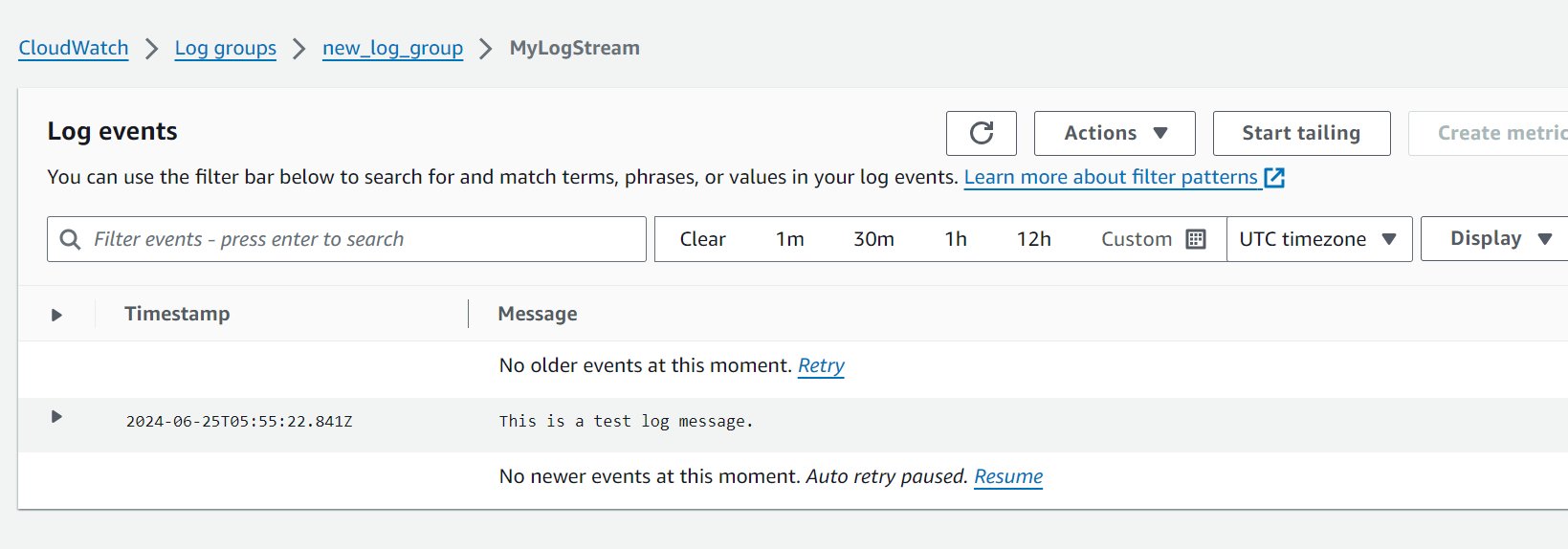
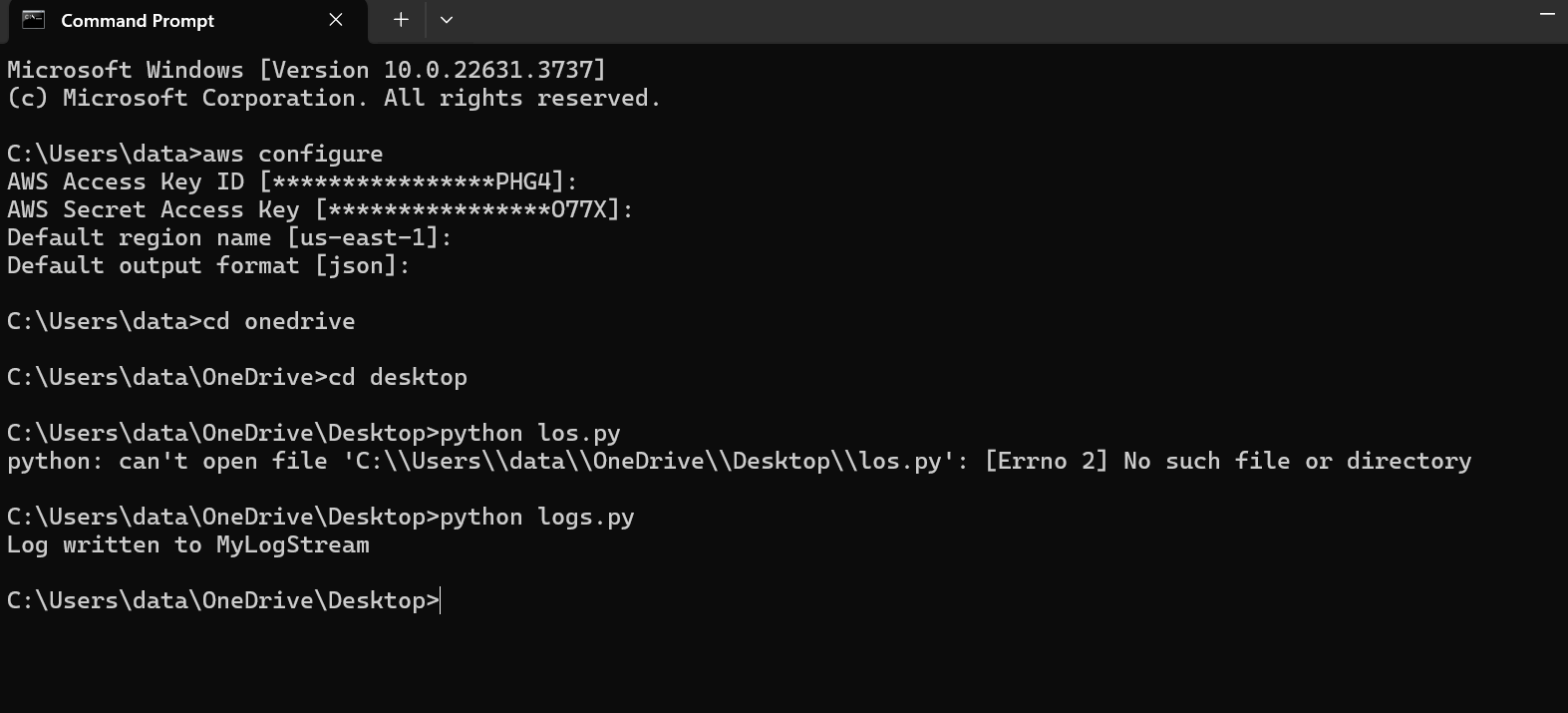
* Navigate to EC2 Dashboard:
* Click on "Services" in the top left corner and select "EC2" under "Compute".
* Launch Instance:
* Click on "Launch Instance".
* Select "Amazon Linux 2 AMI".
* Choose an instance type (e.g., t2.micro for free tier eligibility).
* Configure instance details (subnet, auto-assign public IP).
* Configure security group rules (SSH for Linux access).
* **Launch Windows EC2 Instance:**
* Navigate to EC2 Dashboard:
* Click on "Launch Instance".
* Select a Windows Server AMI (e.g., Microsoft Windows Server 2019 Base).
* Choose an instance type suitable for your needs.
* Configure instance details (subnet, auto-assign public IP).
* Configure security group rules (RDP for Windows access).
* To create an image, select the instance, actions>image and templates > create image.
* Name the image and click create.



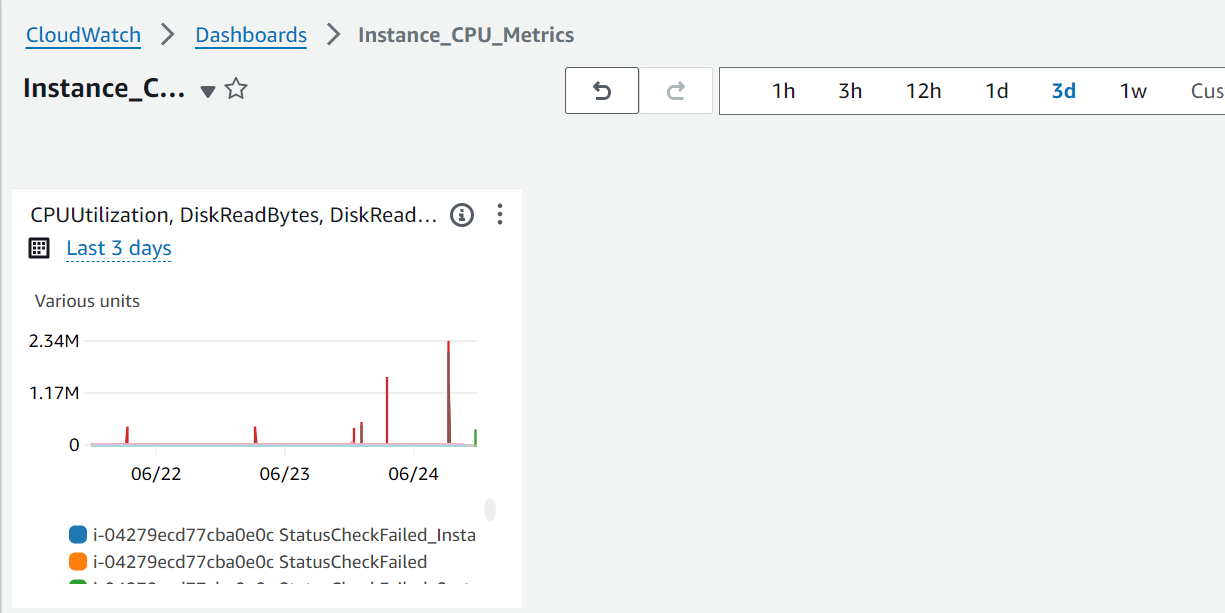
CloudWatch

**Create a log group, using python scripts write logs to the group and create a dashboard for monitoring instance cpu metrics.**

* + In the CloudWatch console, click on "Log groups" in the left sidebar.
  + Click on the "Create log group" button.
  + Enter a name for your log group (new\_log\_group)
  + Click on "Create log group".
  + Click on create log stream and name your log stream (MyLogStream).
  + Write the python script to write logs and run it.



* + In the cloudwatch console, click on "Dashboards" in the left sidebar.
  + Click on "Create dashboard".
  + Enter a name for your dashboard (Instance\_CPU\_Metrics) and click "Create dashboard".
  + Click on "Add widget".
  + Choose "Line" or "Number" widget type.
  + Click on "Configure".
  + Select CPU Utilization Metric:
  + In the "Metrics" tab, click on "EC2".
  + Click on "Per-Instance Metrics".
  + Select the checkbox next to the CPUUtilization metric for your instance.
  + Configure Widget Options:
  + Set the Period to your preferred interval.
  + Set the Statistic to Average.
  + Click on "Create widget" to add it to your dashboard.
  + Once all widgets are added, click "Save dashboard".



Lambda Functions

* + In the AWS Management Console, type "Lambda" in the search bar and select "Lambda" from the dropdown.
  + Click on the "Create function" button.
  + Choose "Author from scratch".
  + Configure the Function:
  + Function name: Country\_time
  + Runtime: Python 3.9 (or any other supported version)
  + Role: Choose "Create a new role with basic Lambda permissions".
  + Click "Create function" to create the Lambda function.
  + In the Function Code Section:
  + Scroll down to the "Function code" section.
  + Choose "Code" and add the code in the inline editor:

import json

from datetime import datetime

from zoneinfo import ZoneInfo

def lambda\_handler(event, context):

country = event['country'].lower()

timezones = {

'india': 'Asia/Kolkata',

'singapore': 'Asia/Singapore',

'australia': 'Australia/Sydney',

'new york usa': 'America/New\_York'

}

if country not in timezones:

return {

'statusCode': 400,

'body': json.dumps(f"Invalid country: {country}")

}

timezone = ZoneInfo(timezones[country])

country\_time = datetime.now(timezone).strftime('%Y-%m-%d %H:%M:%S')

return {

'statusCode': 200,

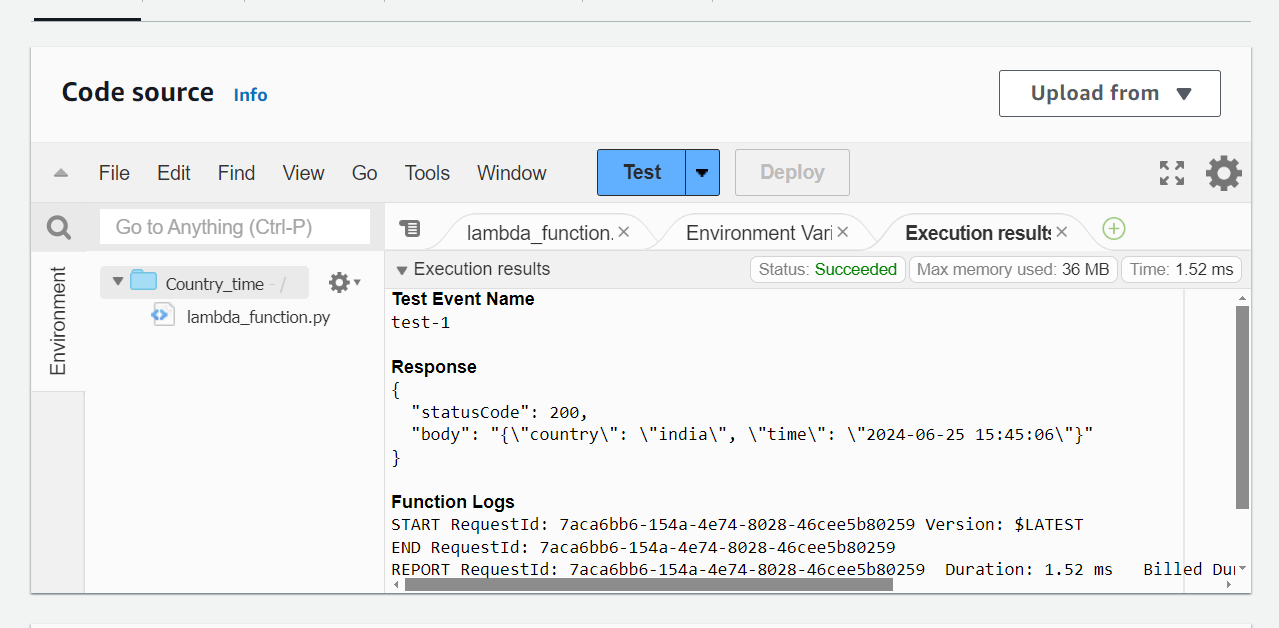
'body': json.dumps({'country': country, 'time': country\_time})

}

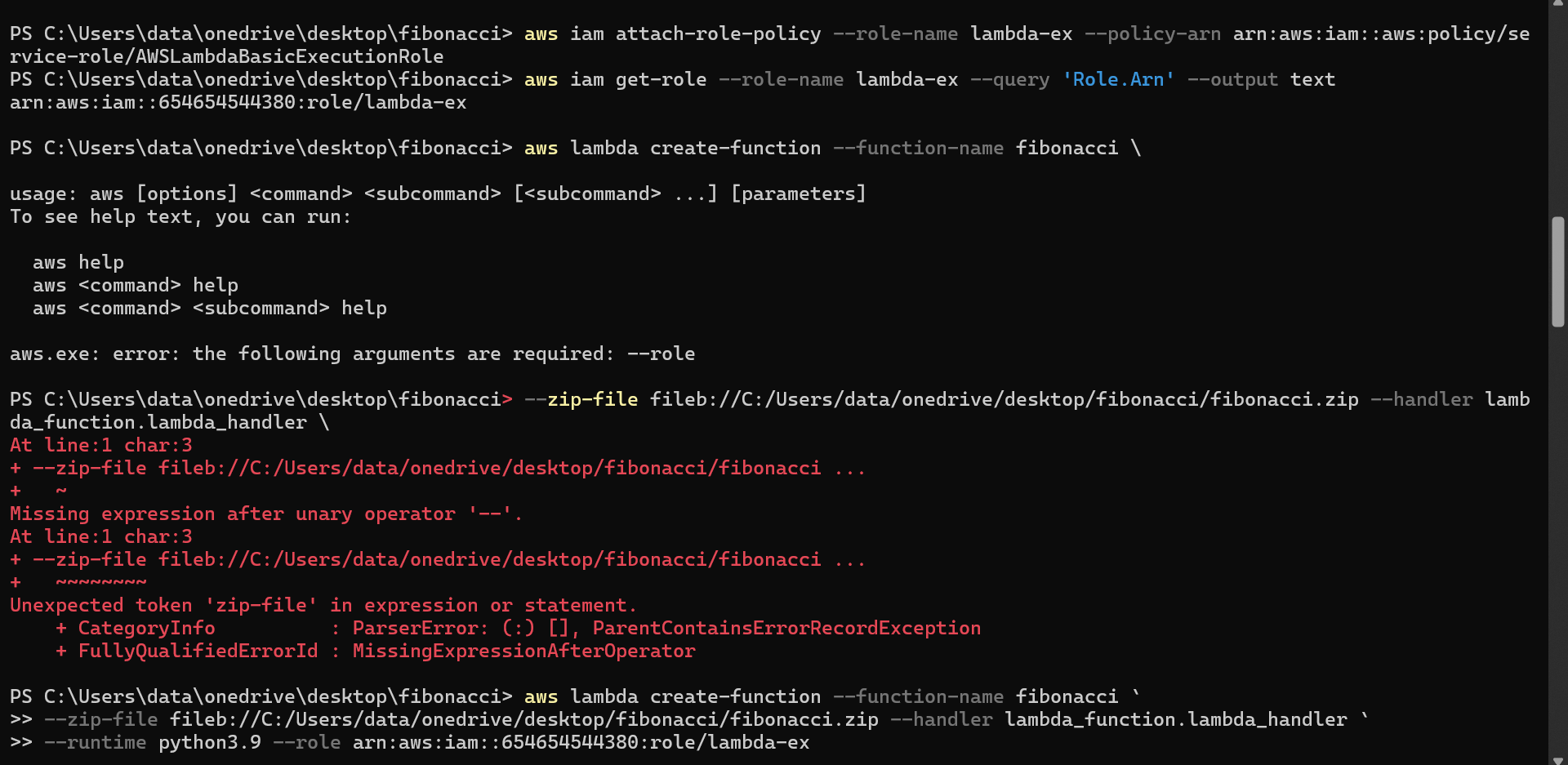
* + Click on "Test" in the AWS Lambda console.
  + Configure a new test event with the following JSON:  
    {

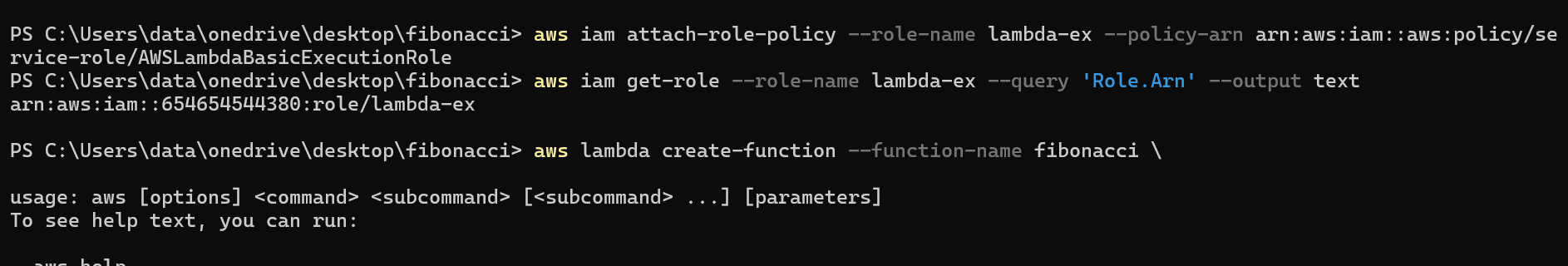
"country": "india"   
}

* + Name the test event and save.
  + Click the "Test" button to run the function with the test event.
  + Check the execution result. You should see the current time in India.



**2. Write a lambda function which checks whether the given number is a fibonacci or not. The input number to check will always be less than 300.**

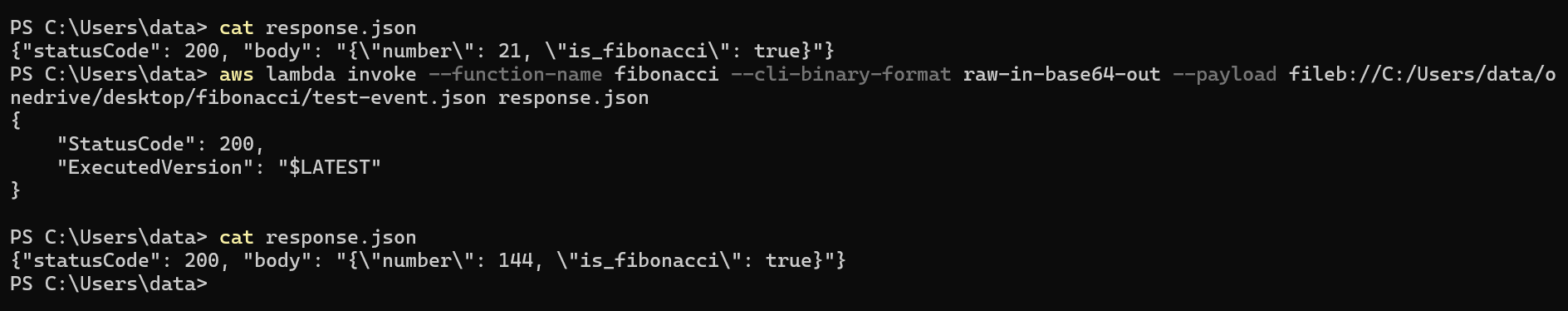
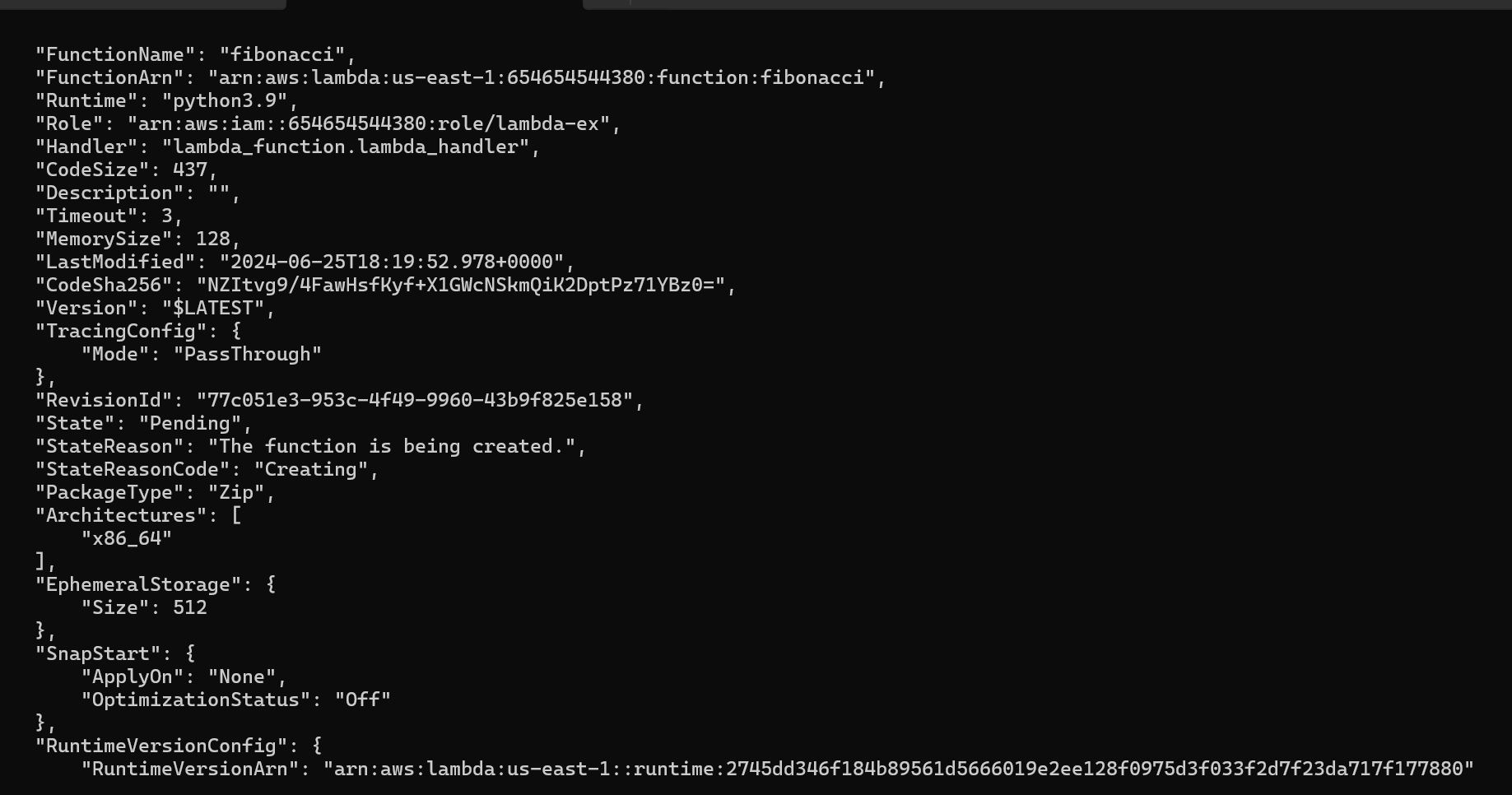
* + Create a Python script (lambda\_function.py) with the Lambda function code.
  + Open AWS CLI.  
    



* + aws lambda create-function --function-name fibonacci `

>> --zip-file fileb://C:/Users/data/onedrive/desktop/fibonacci/fibonacci.zip --handler lambda\_function.lambda\_handler `

>> --runtime python3.9 --role arn:aws:iam::654654544380:role/lambda-ex



DynamoDB

* + In the aws console, navigate to DynamoDB.
  + Click on Create table.
  + Set Table name to GenericFoodTable.
  + Set Partition key to food\_name (String).
  + Set Sort key to scientific\_name (String).
  + Set another attribute to group (String).
  + Click Create.
  + navigate to the Items tab.
  + Click Create item.
  + Enter values for food\_name, scientific\_name and group for 10 items.
  + **Create a Lambda Function:**
  + Navigate to AWS Lambda in the AWS Management Console.
  + Click Create function.
  + Choose Author from scratch.
  + Enter a function name (GenericFood).
  + Choose the runtime (e.g., Python 3.12).
  + Click Create function.
  + Go to the function’s role in IAM console and attach the AmazonDynamoDBFullAccess policy to the Lambda execution role.
  + Go to the function and add the function and deploy.
  + import json

import boto3

from boto3.dynamodb.conditions import Key

def lambda\_handler(event, context):

food\_name = event['food\_name'].lower()

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('GenericFoodTable')

response = table.query(

KeyConditionExpression=Key('food\_name').eq(food\_name)

)

scientific\_names = [item['scientific\_name'] for item inresponse['Items']]

return {

'statusCode': 200,

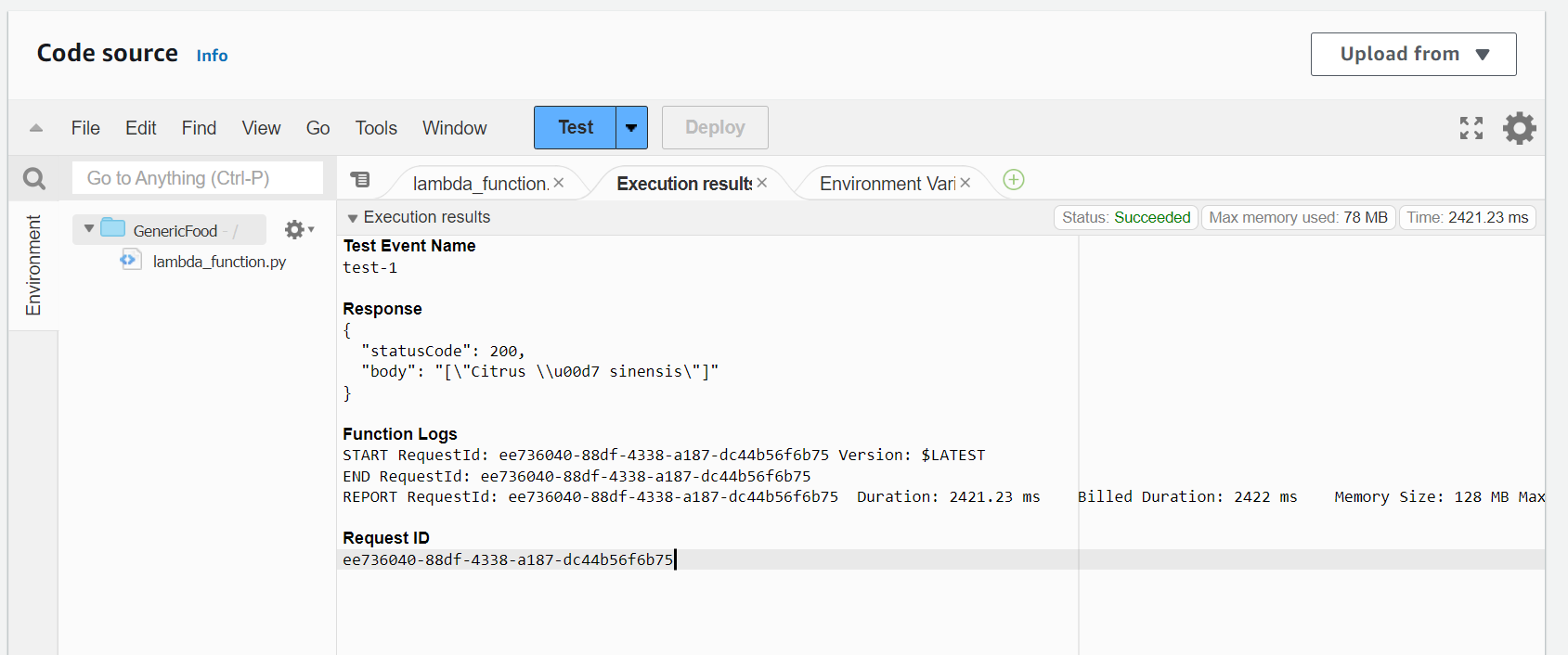
'body': json.dumps(scientific\_names)

}

* + - Name the test and add the json content  
      {

“food\_name”: “orange”

}



* + - **Listing the items based on group:**
    - Repeat the same steps for creating the lambda function
    - Add the code and deploy it.  
      import json

import boto3

from boto3.dynamodb.conditions import Key

def lambda\_handler(event, context):

group = event['group'].lower()

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('GenericFoodTable')

response = table.scan(

FilterExpression=Key('group').eq(group)

)

food\_names = [item['food\_name'] for item in response['Items']]

return {

'statusCode': 200,

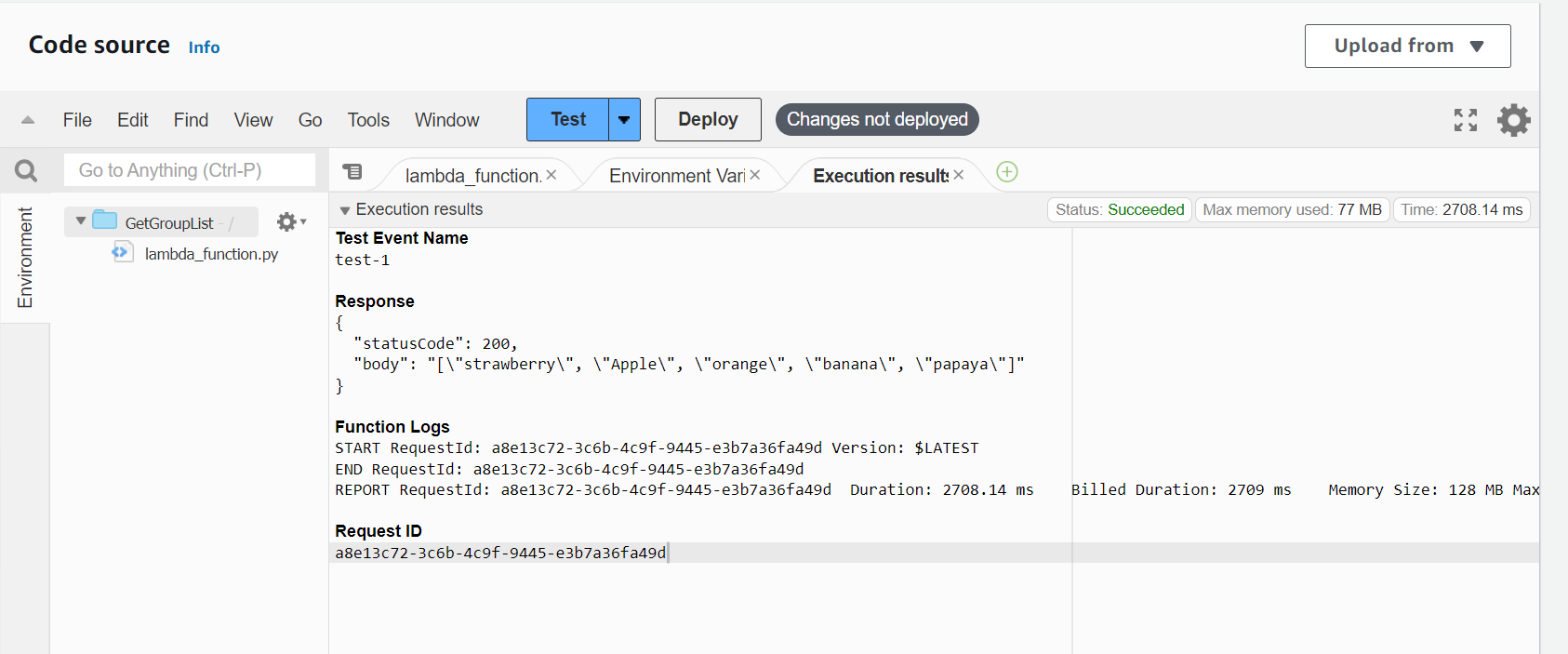
'body': json.dumps(food\_names)

}

* + - Name the test and add the json content  
      {

“group”: “fruit”

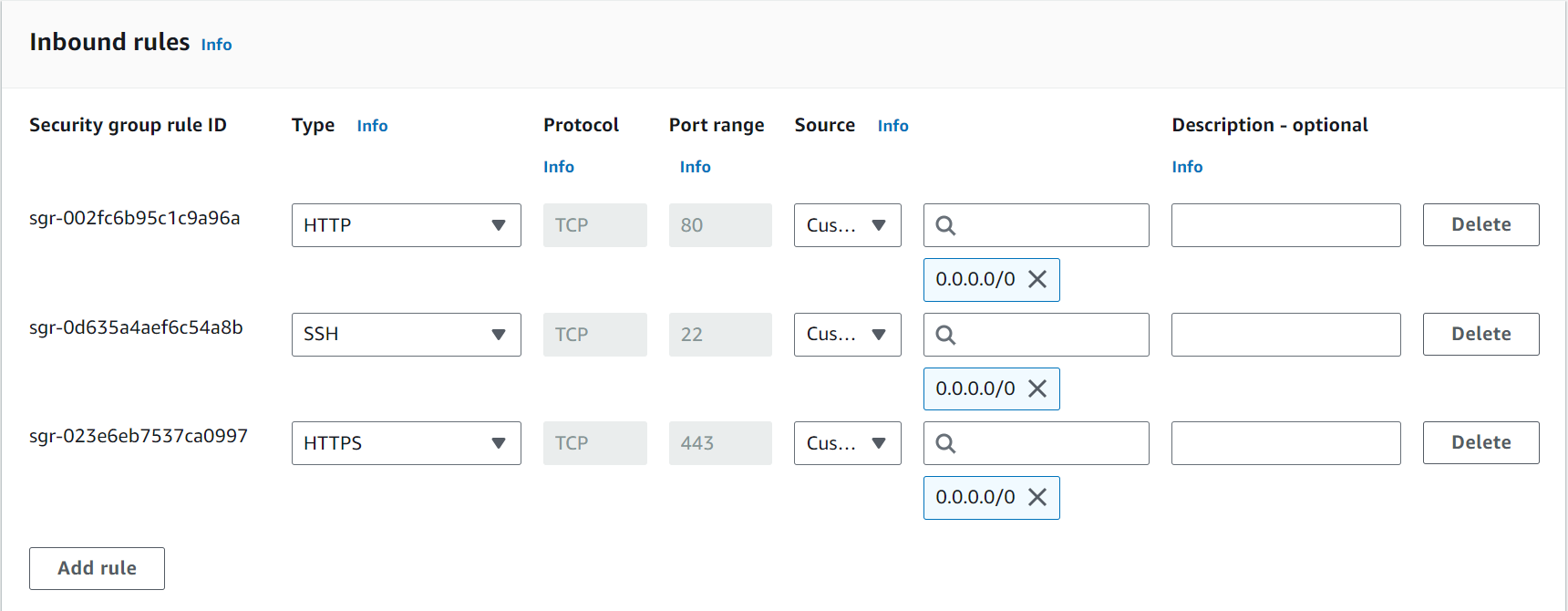
}

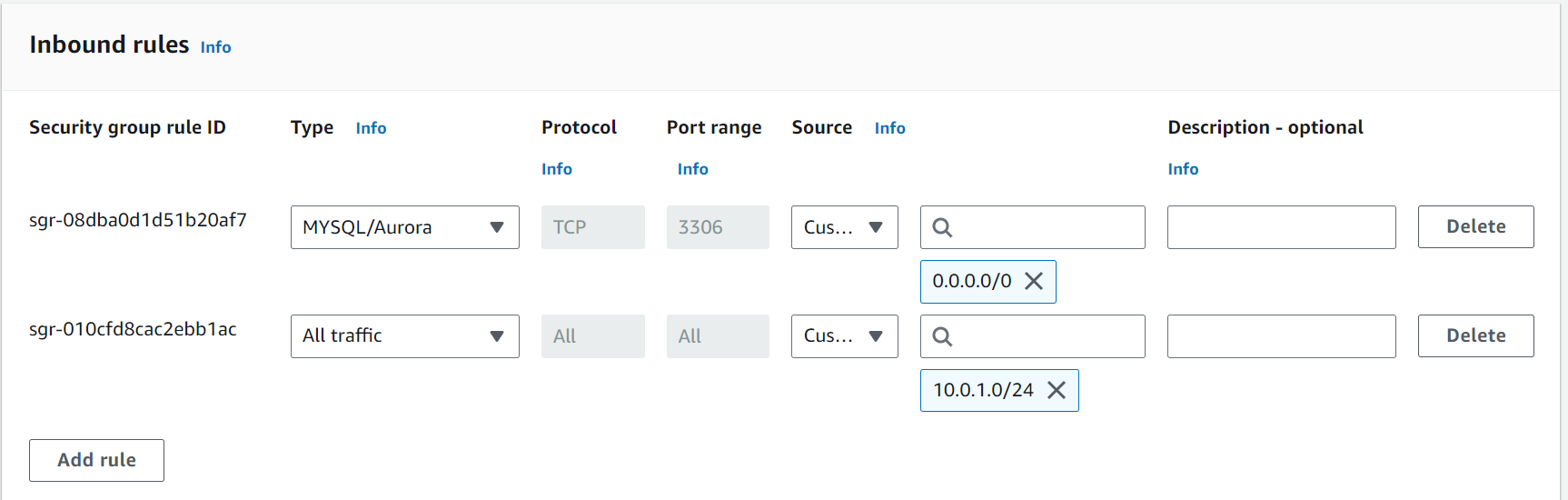


VPC

**Create a VPC**

* + Open the AWS Management Console.
  + Navigate to the VPC Dashboard, then click Create VPC.
  + Enter the details,
  + Name tag: myvpc-2
  + IPv4 CIDR block: 10.0.0.0/16
  + IPv6 CIDR block: No IPv6 CIDR Block
  + Tenancy: Default
  + Click Create VPC.
  + **Create Subnets**
  + Navigate to Subnets on the left-hand side, then click Create subnet.
  + Enter the following details:
  + Name tag: PublicSubnet
  + VPC: myvpc-2
  + Availability Zone: us-east-1a
  + IPv4 CIDR block: 10.0.1.0/24
  + Click Create subnet.
  + Follow the same steps for creating the private subnet.
  + Navigate to Internet Gateways on the left-hand side, then click Create internet gateway.
  + Enter the following details:
  + Name tag: MyIGW
  + Click Create internet gateway.
  + Select the newly created MyIGW.
  + Click Actions and select Attach to VPC.
  + Choose myvpc-2 and click Attach internet gateway.
  + **Create a Public Route Table**
  + Navigate to Route Tables, then click Create route table.
  + Enter the following details:
  + Name tag: PublicRouteTable
  + VPC: myvpc-2
  + Click Create route table.
  + Select the newly created PublicRouteTable.
  + Click on the Routes tab, then click Edit routes.
  + Add the following route:
  + Destination: 0.0.0.0/0
  + Target: igw-<id> select igw that created.
  + Click Save routes.
  + Click on the Subnet associations tab of the PublicRouteTable.
  + Click Edit subnet associations.
  + Select PublicSubnet and click Save associations.
  + Follow the same steps for creating the private route table and associate it with private subnet.
  + Configure Security Groups
  + Navigate to Security Groups, then click Create security group.
  + Enter the following details:
  + Security group name: PublicSG
  + Description: Security group for public subnet
  + VPC: myvpc-2
  + Click Create security group.
  + Select the newly created PublicSG.
  + Click on the Inbound rules tab, then click Edit inbound rules.
  + Add the following rules:

Follow the same steps for creating private security group and edit the inbound rules as following:



**Launch an Instance in the Public Subnet (phpMyAdmin)**

* + Navigate to Instances on the left-hand side, then click Launch instance.
  + Enter the following details:
  + Name: phpMyAdminInstance
  + AMI: Choose an appropriate AMI, e.g., Amazon Linux 2
  + Instance type: t2.micro
  + Key pair: Select key pair (ec2instance)
  + Network settings:
  + Network: myvpc-2
  + Subnet: PublicSubnet
  + Auto-assign Public IP: Enable
  + Security group: PublicSG
  + Click Launch instance.
  + Connect to the instance via SSH and install phpmyadmin  
    sudo yum update -y

sudo yum install -y httpd

sudo systemctl start httpd

sudo systemctl enable httpd

sudo amazon-linux-extras install -y php7.4

sudo yum install -y php-mbstring php-zip php-gd php-json php-xml

cd /var/www/html

wget <https://www.phpmyadmin.net/downloads/phpMyAdmin-latest-all-languages.tar.gz>

tar -xzf phpMyAdmin-latest-all-languages.tar.gz

mv phpMyAdmin-\*-all-languages phpmyadmin

rm phpMyAdmin-latest-all-languages.tar.gz

sudo systemctl restart httpd

* + - Follow the same steps for creating instance and install mysql  
      sudo yum update -y

sudo yum install -y mysql-server

sudo systemctl start mysqld

sudo systemctl enable mysqld

sudo mysql\_secure\_installation

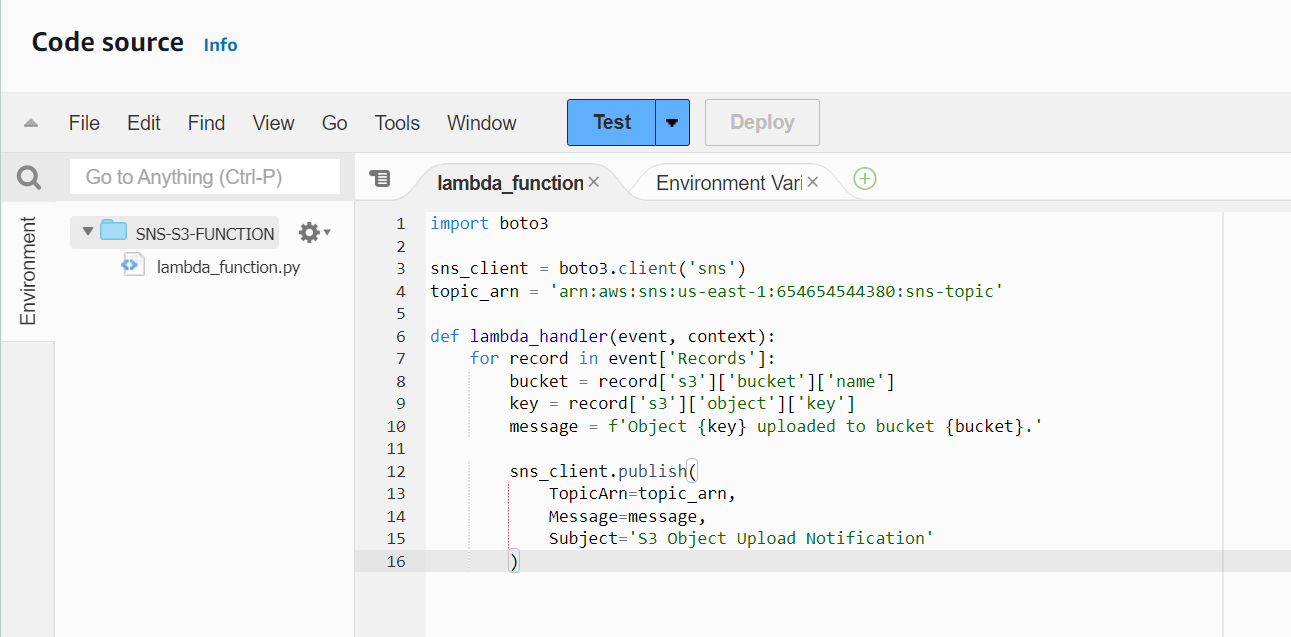
* + - Open the phpMyAdmin configuration file on the public instance.  
      sudo nano /var/www/html/phpmyadmin/config.inc.php

$cfg['Servers'][$i]['host'] = '10.0.2.10'; $cfg['Servers'][$i]['user'] = 'root';

$cfg['Servers'][$i]['password'] = '&bLAk55$';

$cfg['Servers'][$i]['auth\_type'] = 'cookie';

SNS

* + From the AWS Management Console, go to the S3 service.
  + Create a Bucket:
  + Click on the "Create bucket" button.
  + Enter a globally unique name for your bucket (sns-bucket).
  + Select the region and click "Create bucket".
  + From the AWS Management Console, go to the SNS service.
  + Create a Topic:
  + Click on "Create topic".
  + Enter a name for your topic (sns-topic).
  + Click "Create topic".
  + Select the topic you just created.
  + Click on "Create subscription".
  + Choose "Email" as the protocol.
  + Enter the email address.
  + Click "Create subscription".
  + From the AWS Management Console, go to the Lambda service.
  + Create a Lambda Function:
  + Click on "Create function".
  + Choose "Author from scratch".
  + Enter a name for your function (SNS-S3-FUCNTION).
  + Select a runtime (Python 3.12).
  + Under "Permissions", choose "Create a new role with basic Lambda permissions".
  + Click "Create function".
  + Add S3 Trigger:
  + In the Lambda function designer, click on "Add trigger".
  + Choose "S3" as the trigger type.
  + Select the S3 bucket you created earlier.
  + Set the event type to "Object Created (All)" or specific to your needs.
  + Click "Add".
  + Edit the function editor and deploy the function.  
    
  + Test the function by uploading an object to your S3 bucket.
  + Check CloudWatch Logs for logs from your Lambda function.
  + Check the mail for the notification.  
    