**Lab Assignment 3**

**Ashutosh Gangurde 7031**

**Q1  
Create a Lambda function which will display “This is server” message if the code sent is 001. “This is node1” if code sent is 002 and “This is a router” if the code sent is 003. For any other code it will display “Invalid code” message.**

**Create a test event with code as 002. and display the message.**

**Step 1: Set Up Lambda Function**

1. **Sign in to the AWS Management Console** and go to the **Lambda** service.
2. Click **Create function**.
3. Choose **Author from scratch**.
4. Set the function name,
5. Choose the **Runtime** ( Python 3.9).
6. Click **Create function**.

**Step 2: Write the Lambda Function Code**

**import json**

def lambda\_handler(event, context):

    code = event.get("code", "")

    if code == "001":

        message = "This is server"

    elif code == "002":

        message = "This is node1"

    elif code == "003":

        message = "This is a router"

    else:

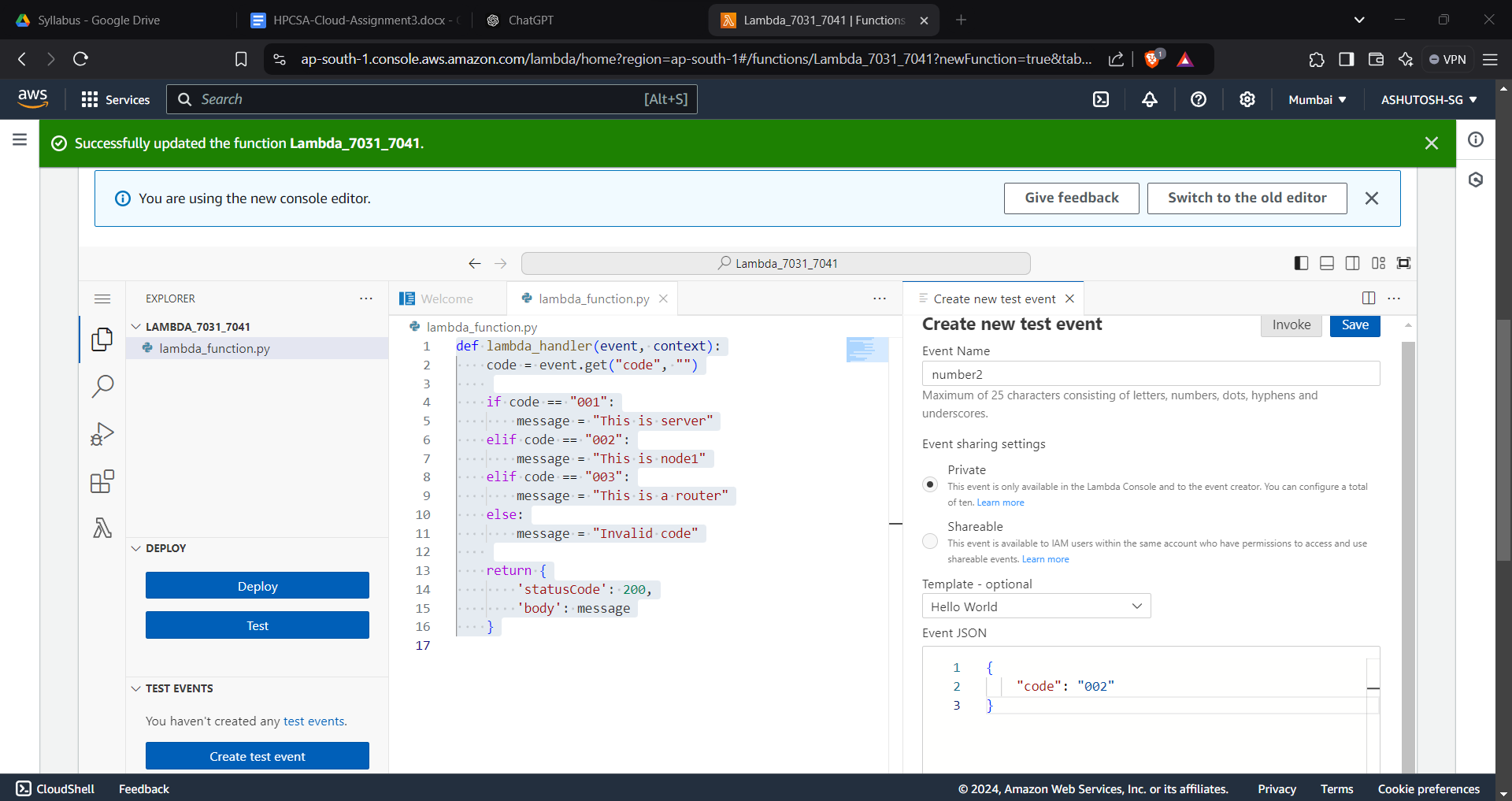
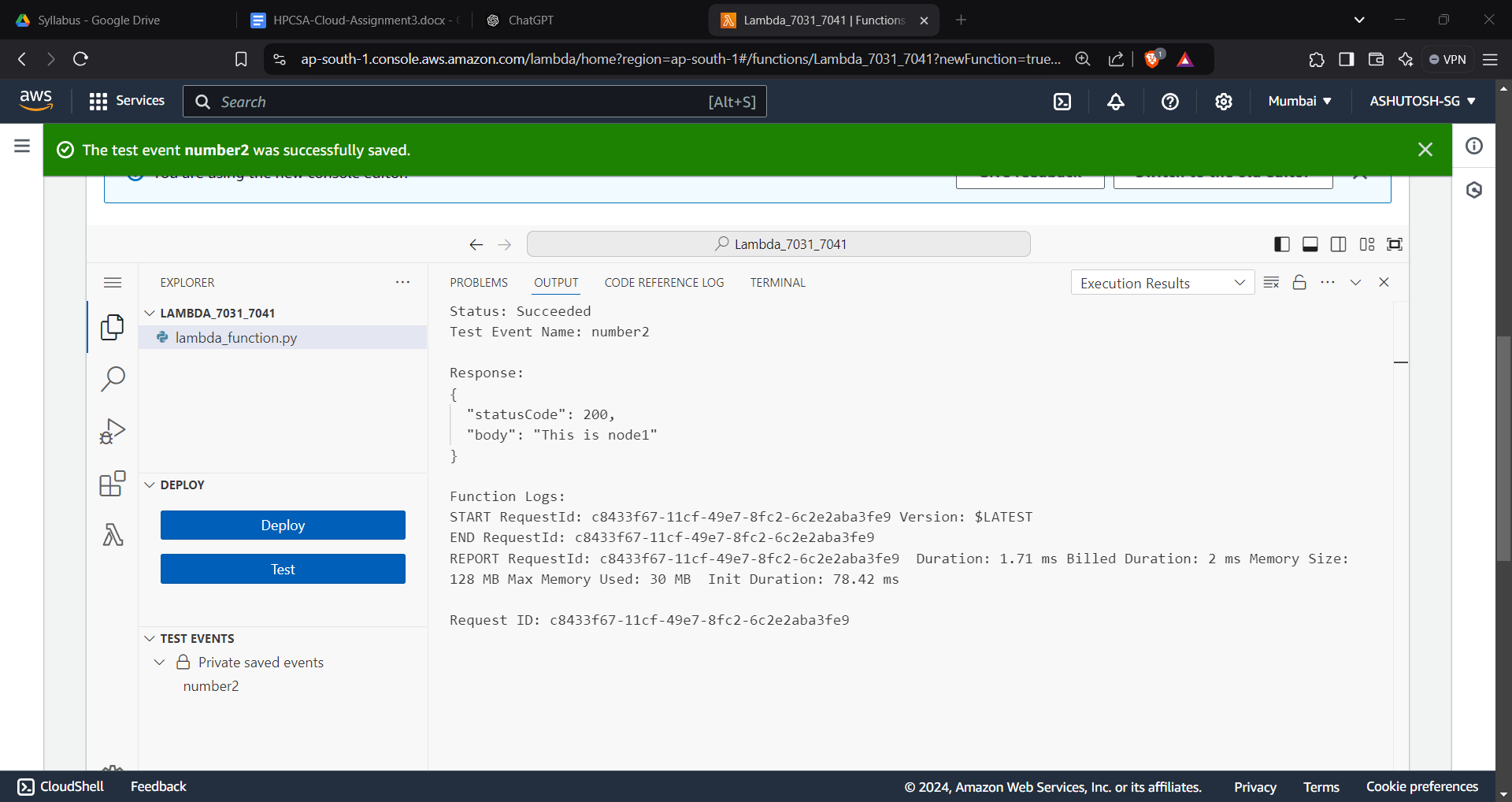
        message = "Invalid code"

    return {

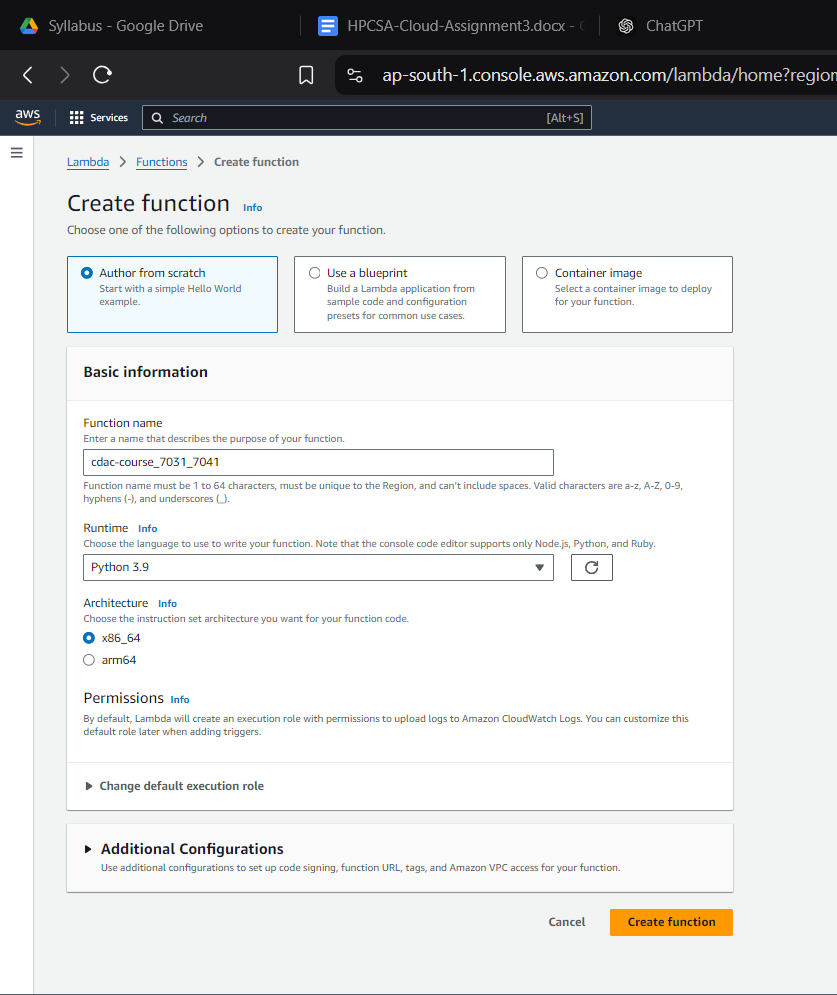
        'statusCode': 200,

        'body': message

    }

 **Test Event** name = number 2 **add Json**{ "code": "002" }  
  
**Q2**  
Create a Lambda function with URL. In the URL user will define course=. If the course=hpcsa then display “Welcome to HPCSA course” message. Similarly it should display messages for ditiss, dbda course.

Access function URL and display the output for all courses.

  
   
  
**def lambda\_handler(event, context):**

**# Get the 'course' parameter from the query string parameters in the event**

**course = event.get("queryStringParameters", {}).get("course", "")**

**# Determine the message based on the course value**

**if course == "hpcsa":**

**message = "Welcome to HPCSA course"**

**elif course == "ditiss":**

**message = "Welcome to DITISS course"**

**elif course == "dbda":**

**message = "Welcome to DBDA course"**

**else:**

**message = "Invalid course"**

**# Return the message with a status code of 200**

**return {**

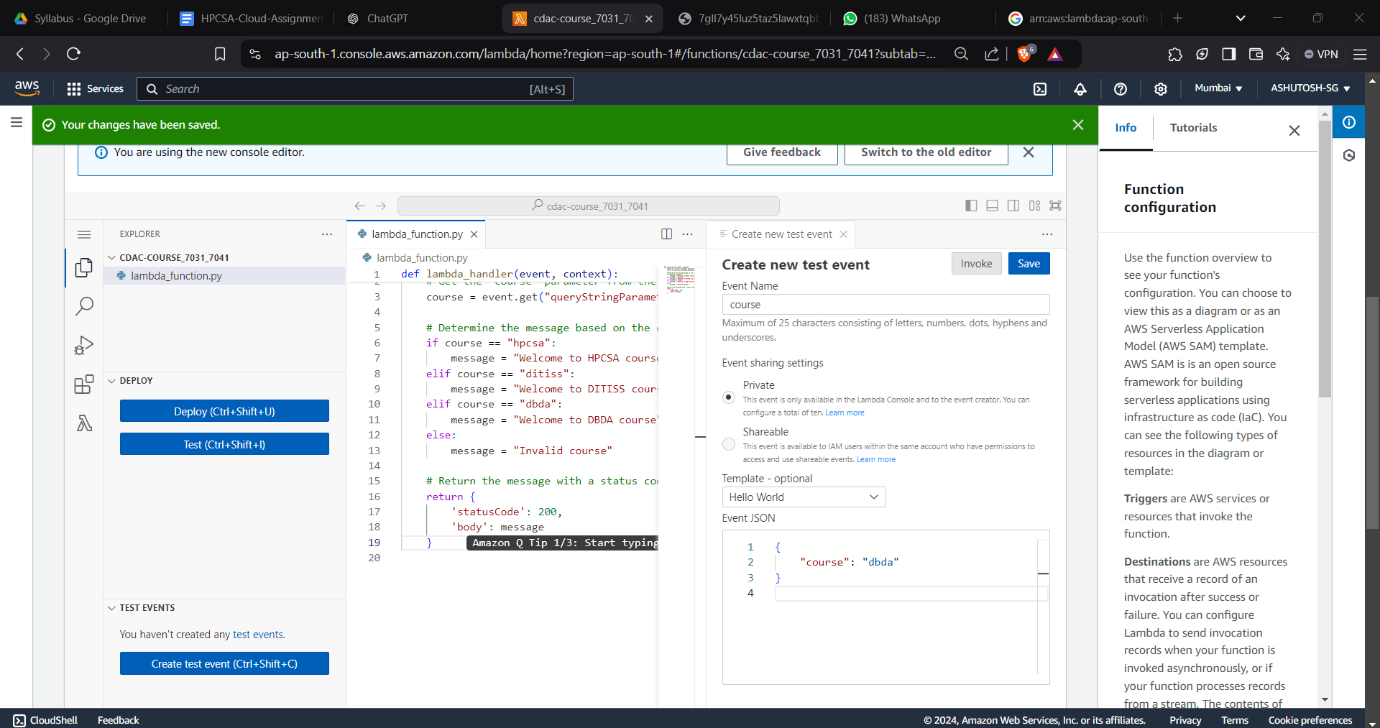
**'statusCode': 200,**

**'body': message**

**}**

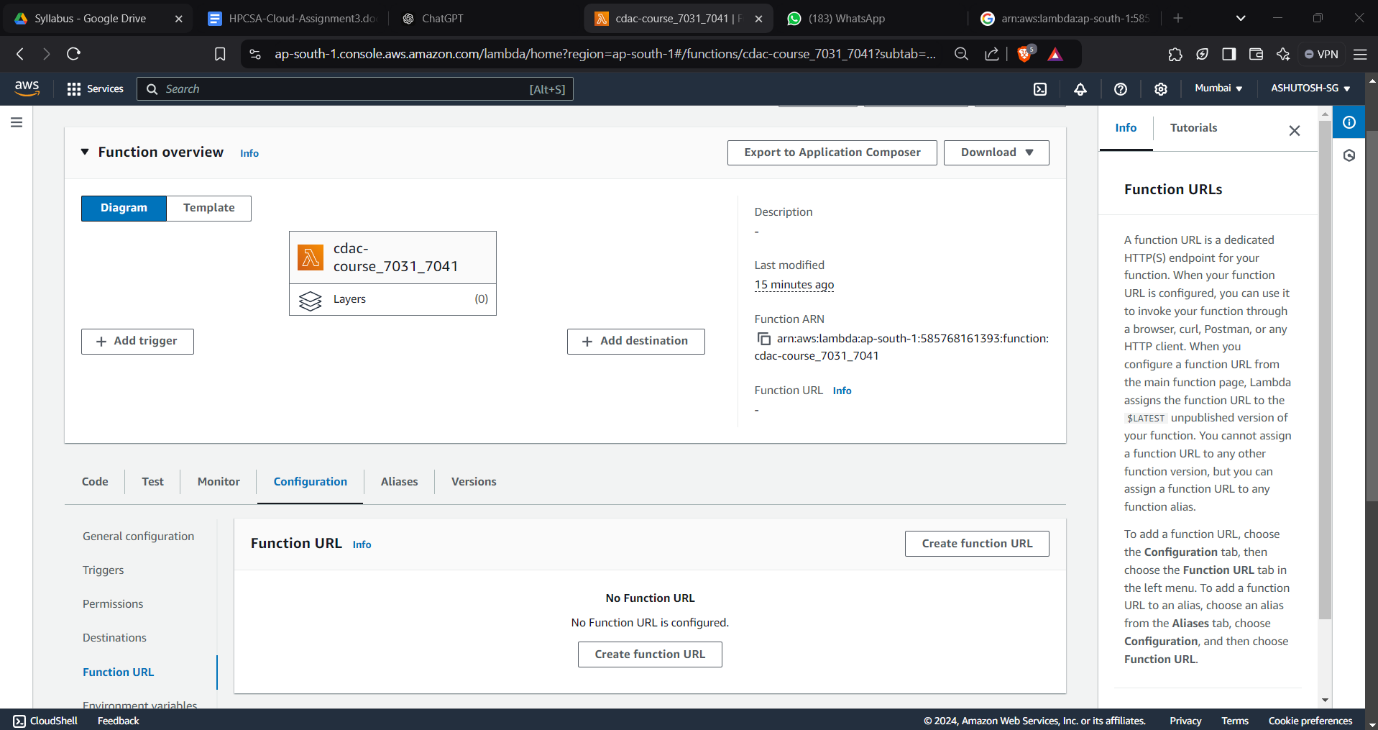
**Set event**

**{   "course": "dbda" }**

****

**Step 3: Configure a Lambda Function URL**

1. In the Lambda function console, scroll to Function URL.
2. Click Create function URL.
3. Set Auth type to None (for open access).
4. Click Create function URL.
5. Copy the function URL provided.



**Step 4: Test the Lambda Function URL**

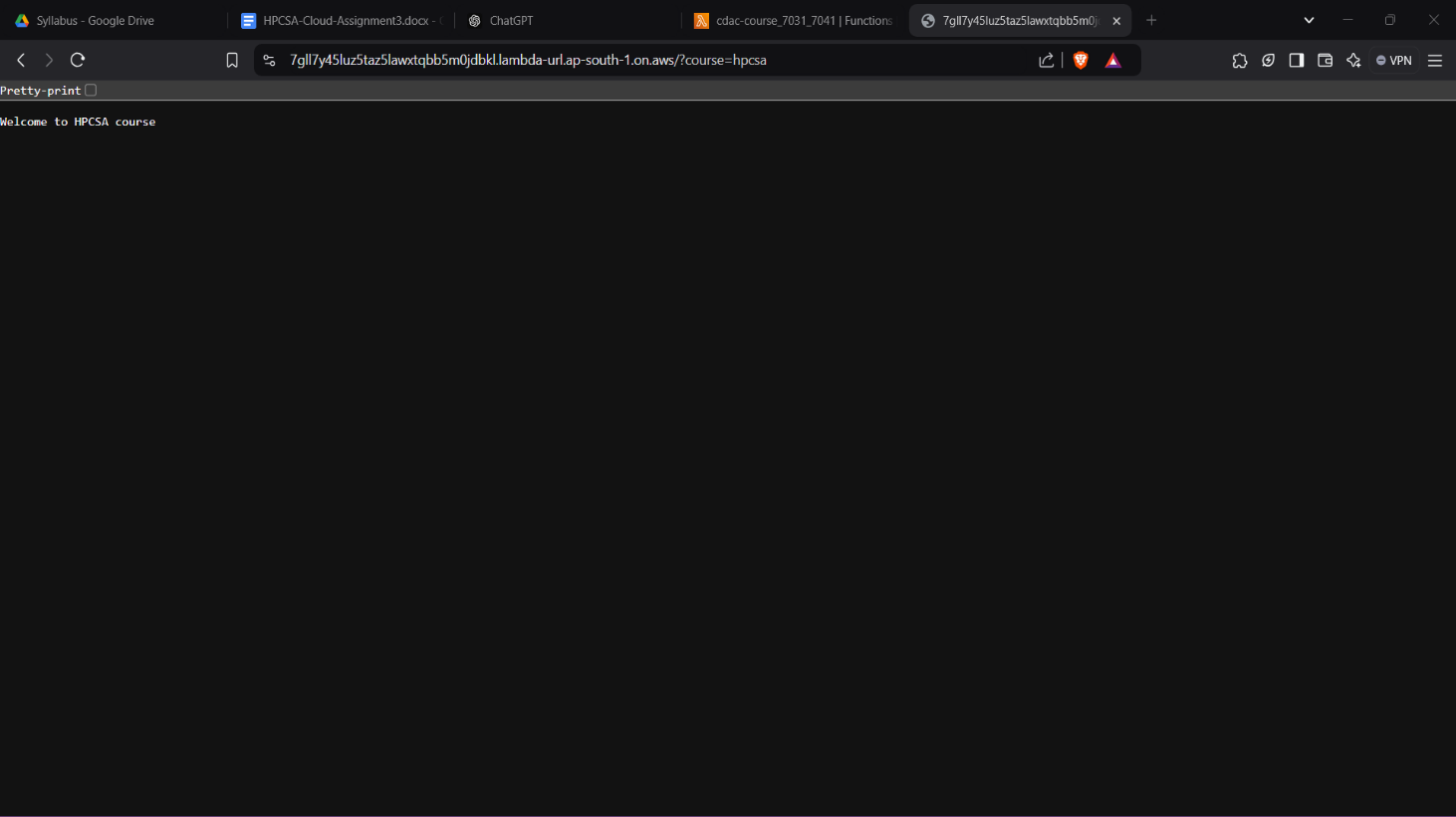
**In Browser:**

https://<your-function-url>?course=hpcsa

<https://7gll7y45luz5taz5lawxtqbb5m0jdbkl.lambda-url.ap-south-1.on.aws/?course=hpcsa>

<https://7gll7y45luz5taz5lawxtqbb5m0jdbkl.lambda-url.ap-south-1.on.aws/?course=ditiss>

<https://7gll7y45luz5taz5lawxtqbb5m0jdbkl.lambda-url.ap-south-1.on.aws/?course=dbda>



Q3

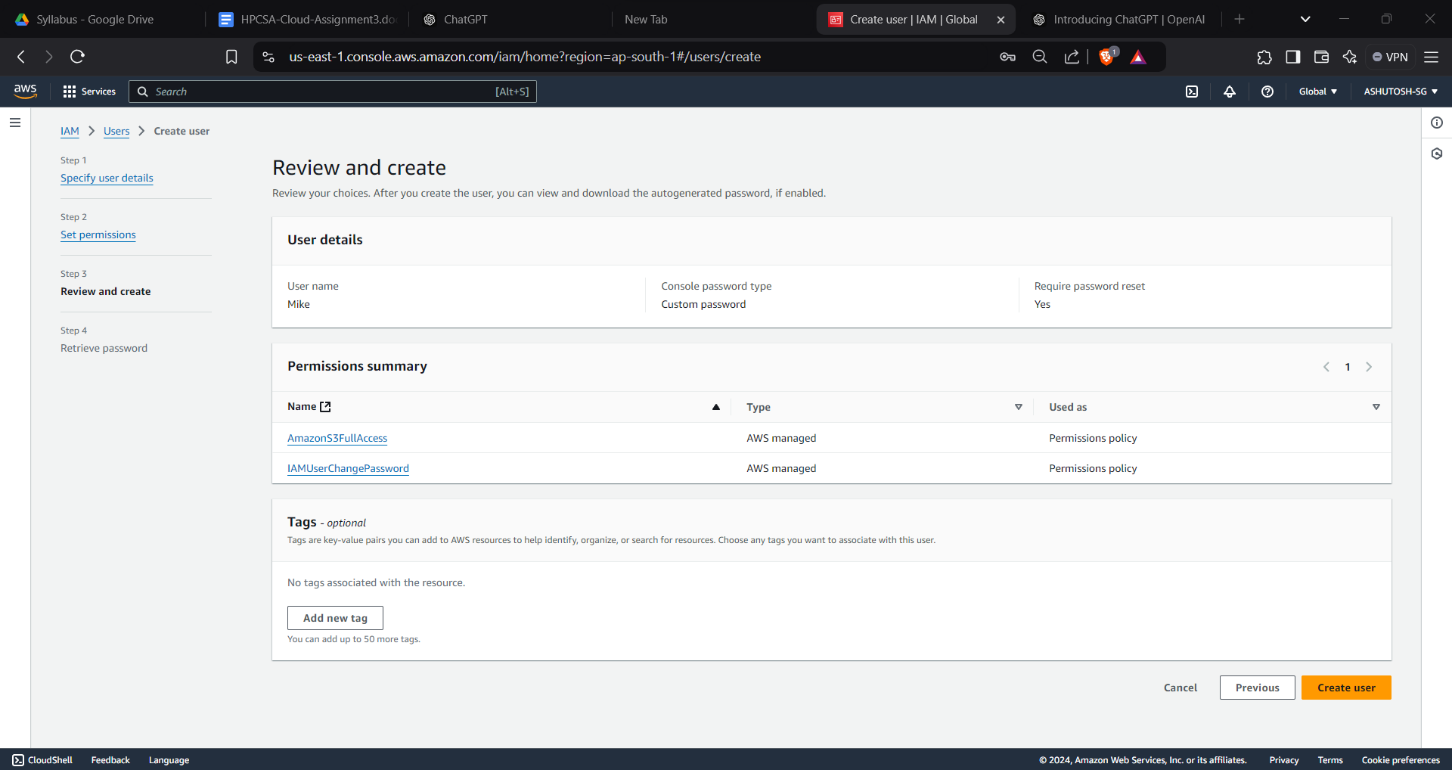
**Create a user Mike in AWS IAM and give that user S3 full access. Create another user by name SAM and give that user S3 read only permission. Create a S3 bucket. Then login by both users and check if they can access this bucket. Which user can upload files to the bucket?**

**Step 1: Create IAM Users in AWS Console**

1.1 Create User Mike (Full Access to S3)

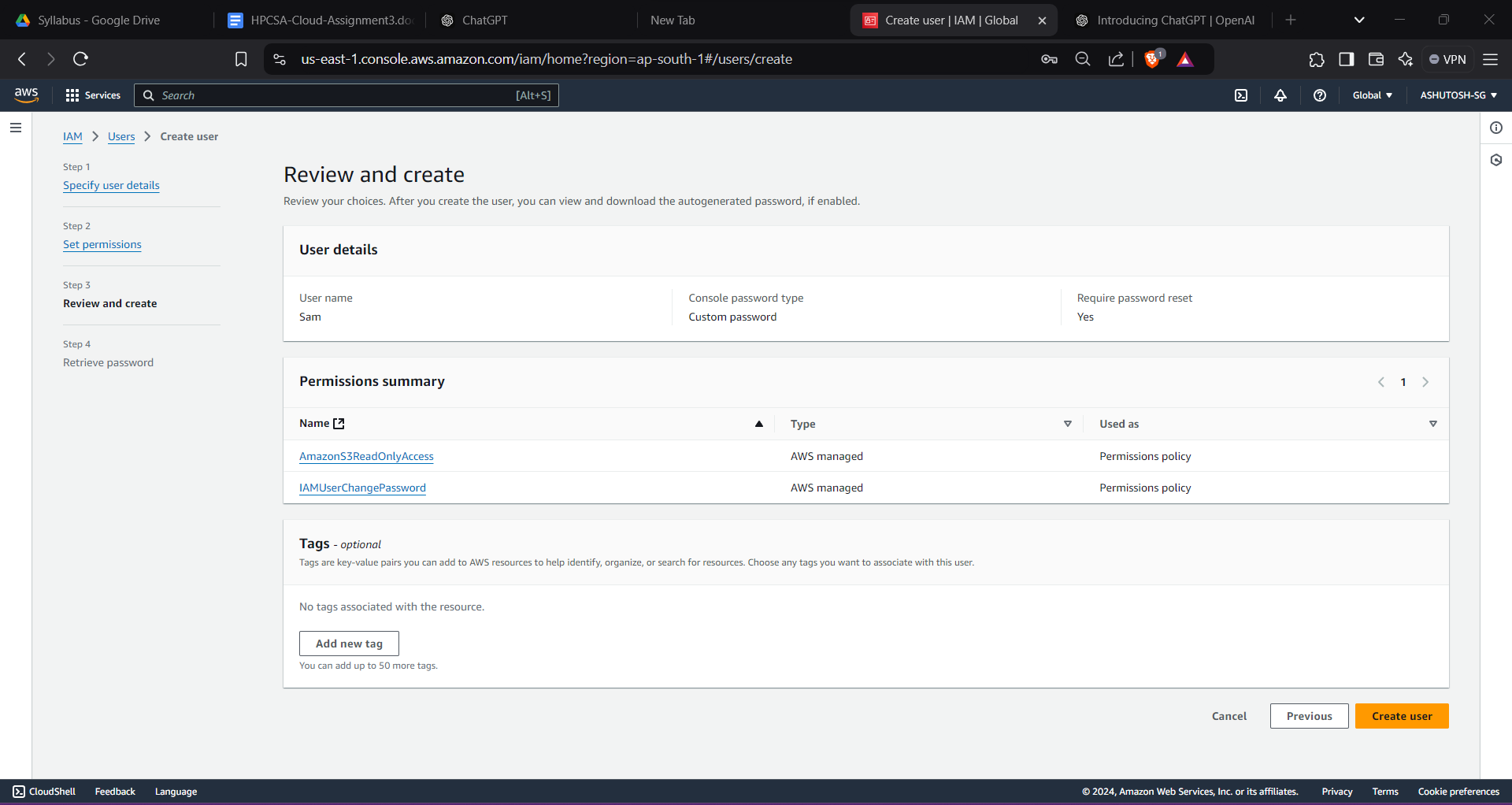
1. Sign in to the AWS Management Console as an administrator.
2. Go to IAM (Identity and Access Management) service.
3. In the IAM Dashboard, click on Users from the left sidebar.
4. Click on Add user.
5. Enter Mike as the user name.
6. Select Programmatic access and AWS Management Console access (if you want them to use both CLI and Console).
7. Set a password for Mike (or use auto-generated).
8. Click Next: Permissions.

* Choose Attach policies directly.
* Search for and select the AmazonS3FullAccess policy (this gives Mike full access to all S3 buckets).
* Click Next: Tags (optional).
* Click Next: Review, then click Create user.

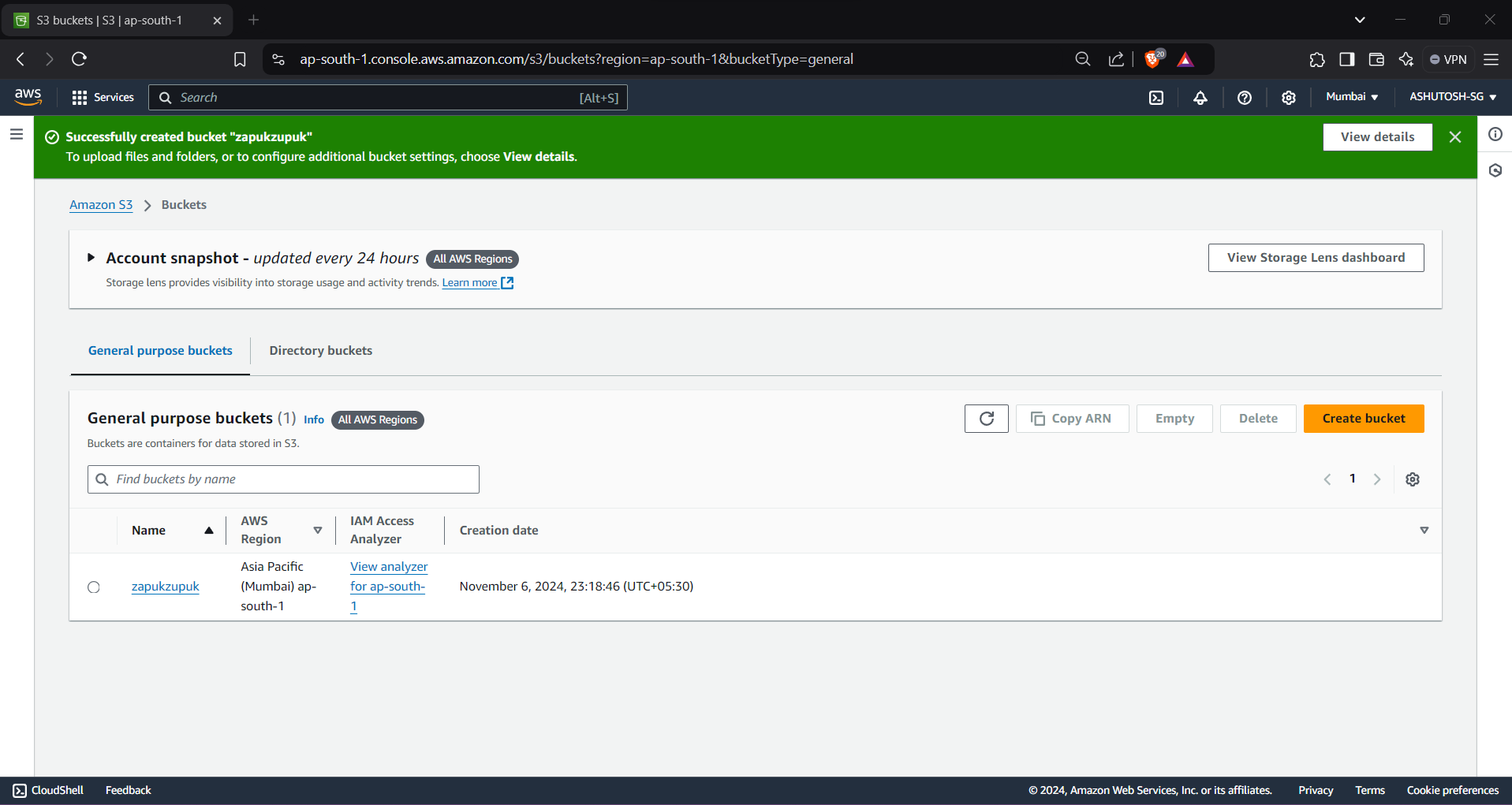
**  
  
  
1.2 Create User Sam (S3 Read-Only Access)**

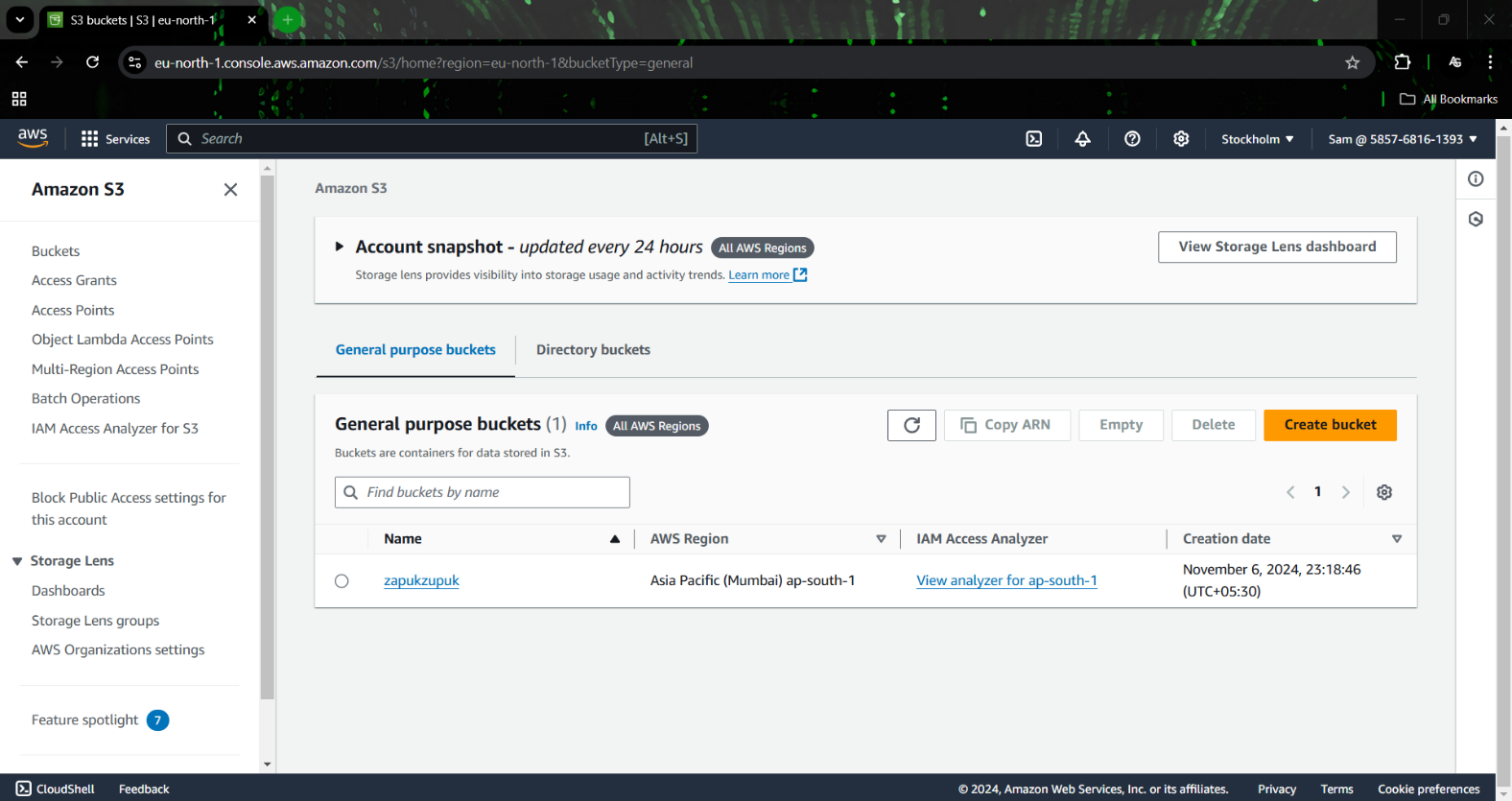
1. Go back to the IAM Dashboard.
2. Click on Add user again.
3. Enter Sam as the user name.
4. Select Programmatic access and AWS Management Console access.
5. Set a password for Sam (or use auto-generated).
6. Click Next: Permissions.

* Choose Attach policies directly.
* Search for and select the AmazonS3ReadOnlyAccess policy (this gives Sam read-only access to all S3 buckets).
* Click Next: Tags (optional).
* Click Next: Review, then click Create user.

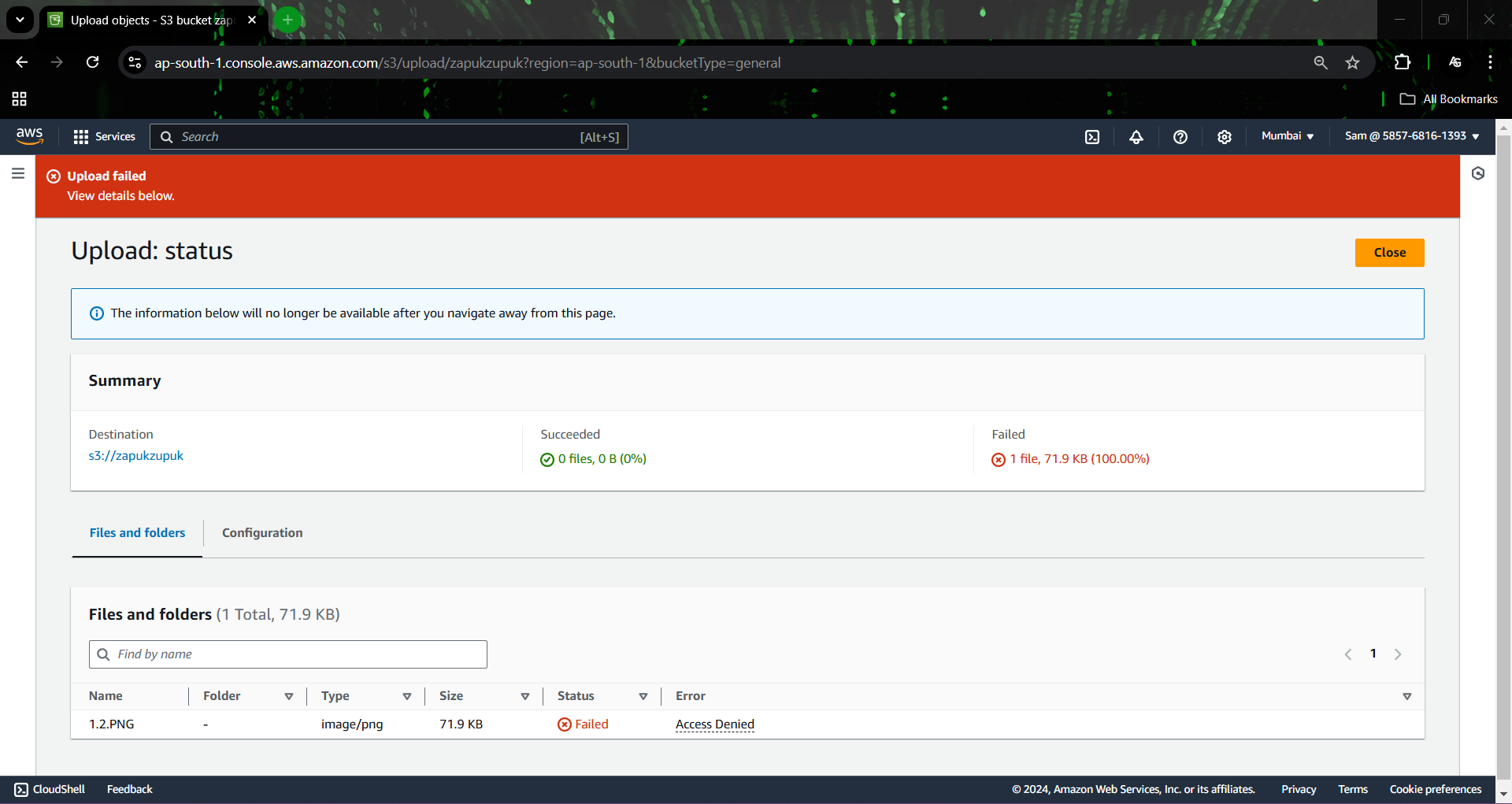
**  
  
Step 2: Create an S3 Bucket**

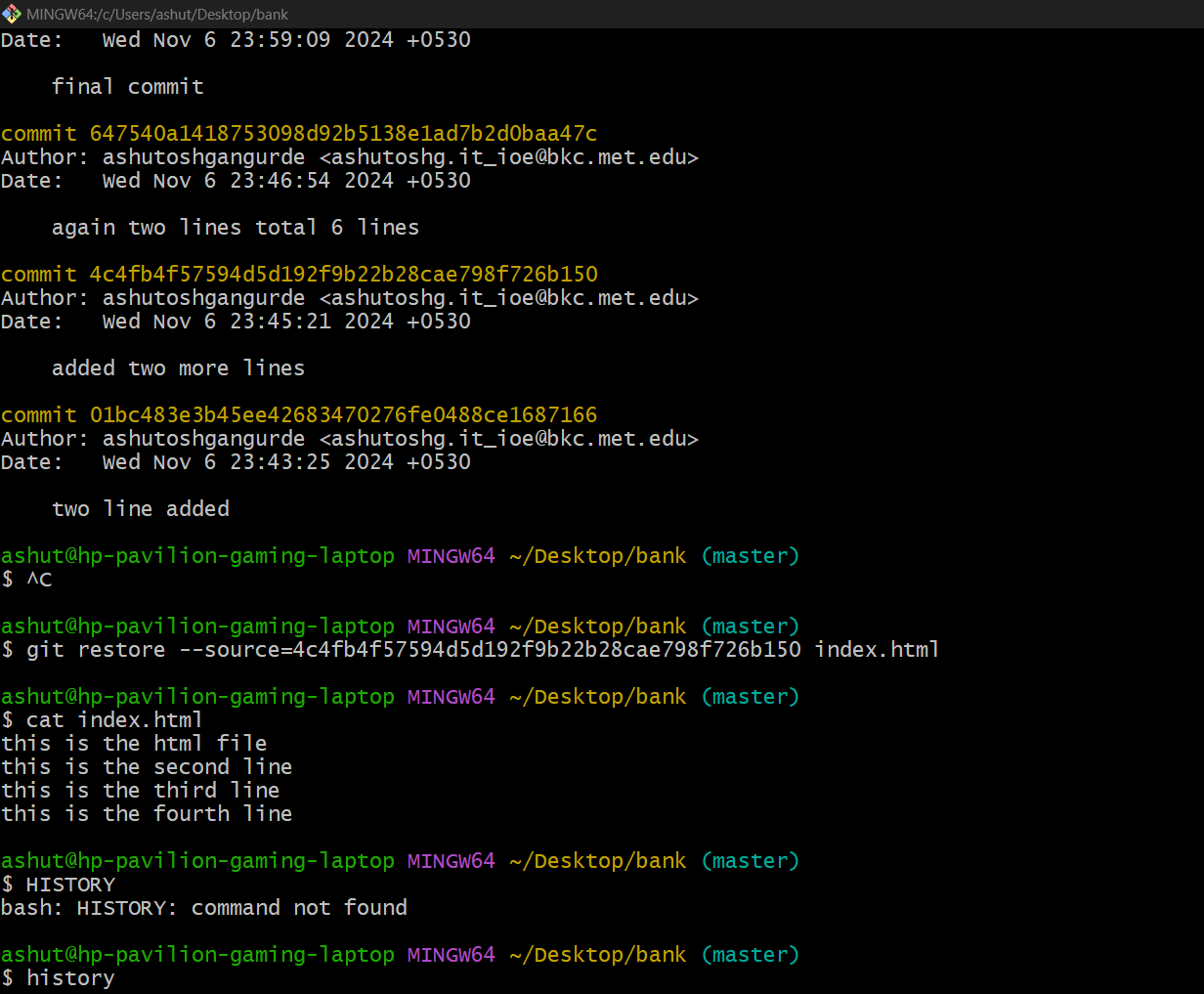
1. Go to the S3 Console.
2. Click Create bucket.
3. Enter a unique name for your bucket (e.g., my-bucket-unique-name-2024).
4. Choose a region and configure other settings as needed.
5. Click Create.

**  
3.2 Log in as Sam (S3 Read-Only Access)**

1. Go to the AWS login page for Sam: https://aws.amazon.com/console/ and log in with Sam’s credentials.
2. In the AWS Management Console, go to S3.
3. Sam will be able to view the bucket and list the contents of the bucket, but they cannot upload files because they have only read-only access  
     
   

**Step 4: Verify Permissions**

* Mike (with S3 Full Access) can upload files to the bucket.
* Sam (with S3 Read-Only Access) can only view the bucket contents but cannot upload files. If Sam tries to upload a file, they will receive an Access Denied error.  
    
  

**Q4**  
Create a local git repository. Create a file by name index.html. Type 2 lines in it. Perform commit. Modify index.html file and add 2 more lines. Perform commit. Modify the file again and add 2 more line. Restore the earlier verison of the file with 4 lines. Modify file again and 3 lines. Perform commit. Restore 4 line version of the file.  
  
  
  
  
  
  
  
  
  
  
mkdir bank # Create a directory named "bank"

cd bank # Change into the "bank" directory

git init # Initialize a Git repository

nano index.html # Create the file index.html and add two lines

git add index.html # Stage the file for commit

git commit -m "two lines added" # Commit the changes with a message

nano index.html # Modify the file and add two more lines

git add -A # Stage the changes

git commit -m "added two more lines" # Commit the changes

nano index.html # Modify the file and add two more lines

git add -A # Stage the changes

git commit -m "again two lines total 6 lines" # Commit the changes

git restore index.html # Restore the file to the previous commit (4 lines)

cat index.html # View the contents of index.html

git log # View the commit history

git revert 4c4fb4f57594d5d192f9b22b28cae798f726b150 # Revert to a previous commit (6 lines)

cat index.html # View the contents after reverting

git revert --abort # Abort the revert operation

nano index.html # Modify the file and add 3 more lines

git add -A # Stage the changes

git commit -m "final commit" # Commit the final changes

git restore --source=4c4fb4f57594d5d192f9b22b28cae798f726b150 index.html # Restore the file to a specific commit

cat index.html # View the restored file contents