Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Advance Devops Practical Examination: AWS Case Study Assignment

Topic : Serverless Image Processing Workflow

1. Introduction

Concepts Used: AWS Lambda, S3, and CodePipeline.

Problem Statement: "Create a serverless workflow that triggers an AWS Lambda function when a new image is uploaded to an S3 bucket. Use CodePipeline to automate the deployment of the Lambda function."

Tasks:

- Create a Lambda function in Python that logs and processes an image when uploaded to a specific S3 bucket.
- Set up AWS CodePipeline to automatically deploy updates to the Lambda function.
- Upload a sample image to S3 and verify that the Lambda function is triggered and logs the event.

Case Study Overview:
The case study involves building a **serverless image processing workflow** using AWS services such as Lambda, S3, and CodePipeline. The goal is to create a serverless workflow that automatically triggers a Lambda function when a new image is uploaded to an S3 bucket. Additionally, the project sets up CodePipeline to automate the deployment of Lambda function updates.

Key Feature and Application:
The case study demonstrates how AWS Lambda can process events triggered by S3 uploads, facilitating **real-time image processing** without manual intervention. The project also highlights how CodePipeline automates Lambda function deployment, ensuring **continuous integration and continuous delivery (CI/CD)**.

Guidelines & Best Practices

1. **Use Least Privilege:** Restrict IAM roles to necessary permissions only.
2. **Modular Code:** Keep Lambda functions focused and maintainable.
3. **Efficient Triggers:** Set specific S3 event types to avoid redundant triggers.
4. **Error Handling:** Implement logging and error management in Lambda using CloudWatch.

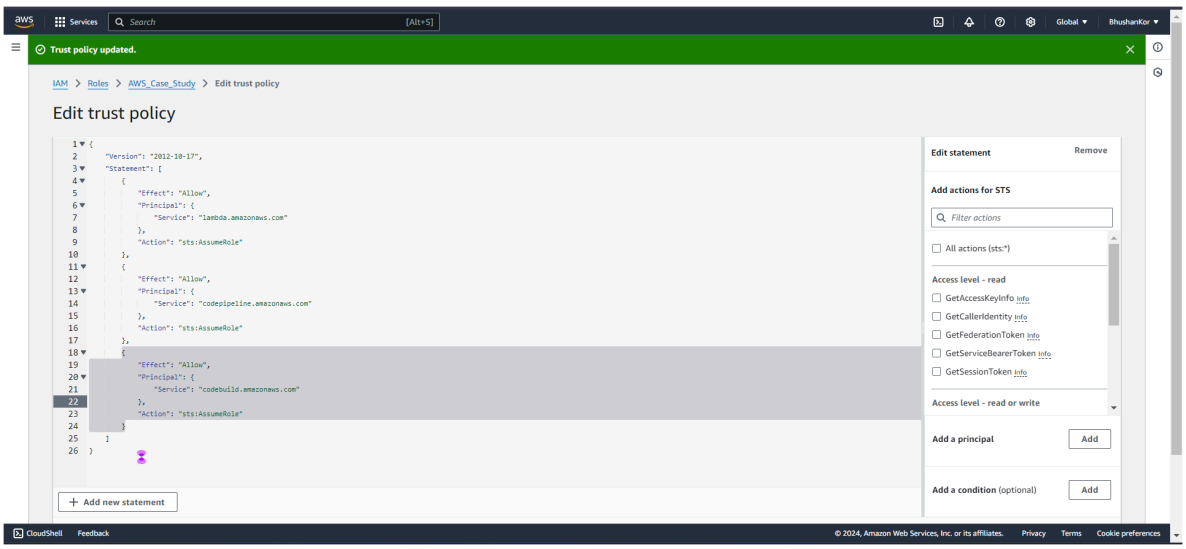
5. **Automate CI/CD:** Use CodePipeline and CodeBuild to update Lambda automatically.
6. **Optimize Costs:** Use AWS Free Tier and manage resource use smartly.

2. Step-by-Step Explanation

1. Create an IAM Role :

- Log in to the **AWS Management Console**.
- Go to the **IAM Console** in AWS.
- Click on **Roles** and choose **Create Role**.
- Select **AWS service** as the trusted entity type, and choose **Lambda** as the use case.
- Attach policies such as:
    - **AmazonS3FullAccess**
    - **AmazonCodeBuildAdminAcess**
    - **AmazonCodePipeline_FullAccess**
    - **AWSLambda_FullAccess**
    - **CloudWatchLogsFullAccess**
- Give the role a name: **AWS_Case_Study** and create it.
- In Trust Realtionship/Policy attach below policy in Statement.

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "codepipeline.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
},
{
    "Effect": "Allow",
    "Principal": {
        "Service": "codebuild.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
}
```

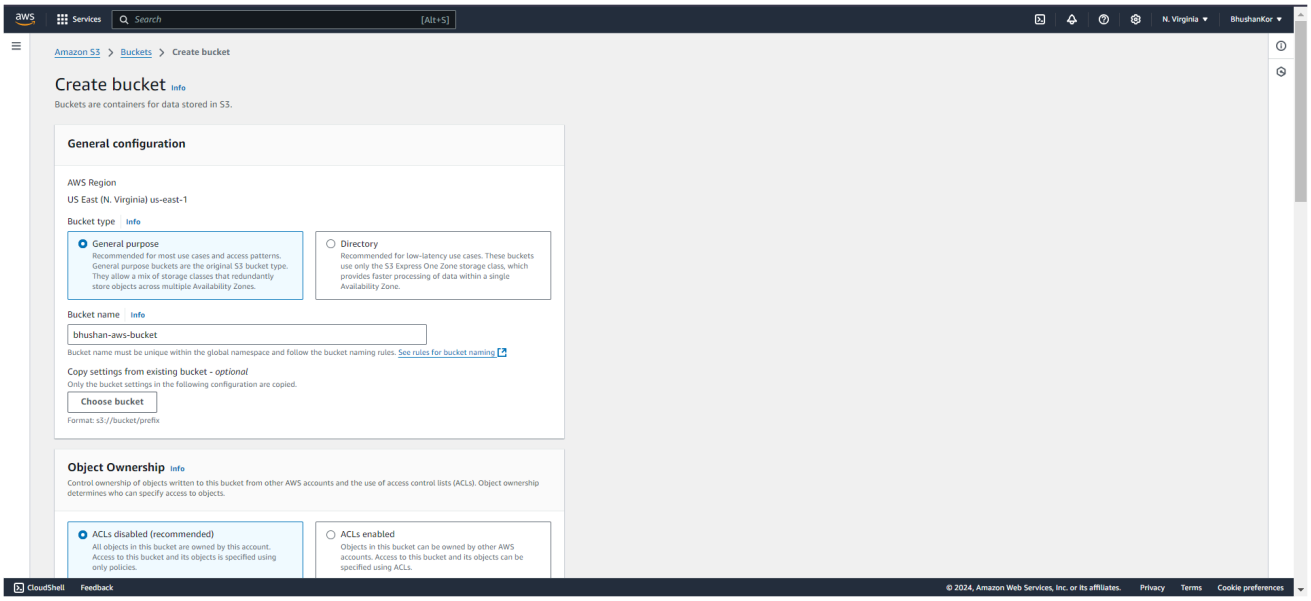


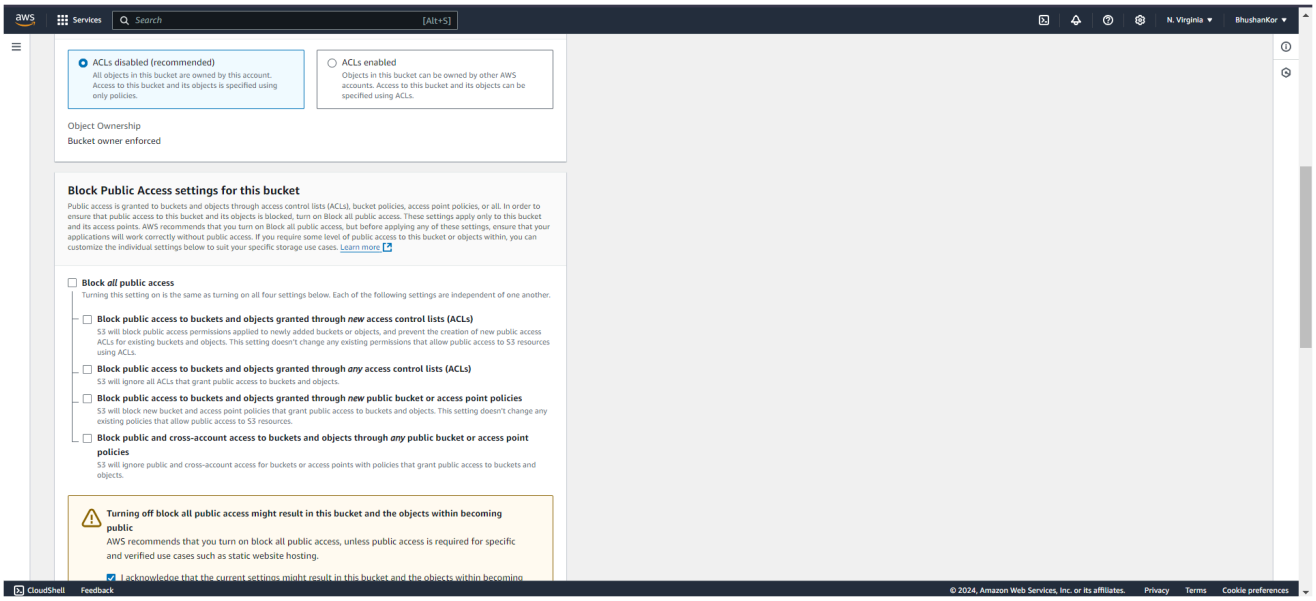
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

2. Create an S3 Bucket:

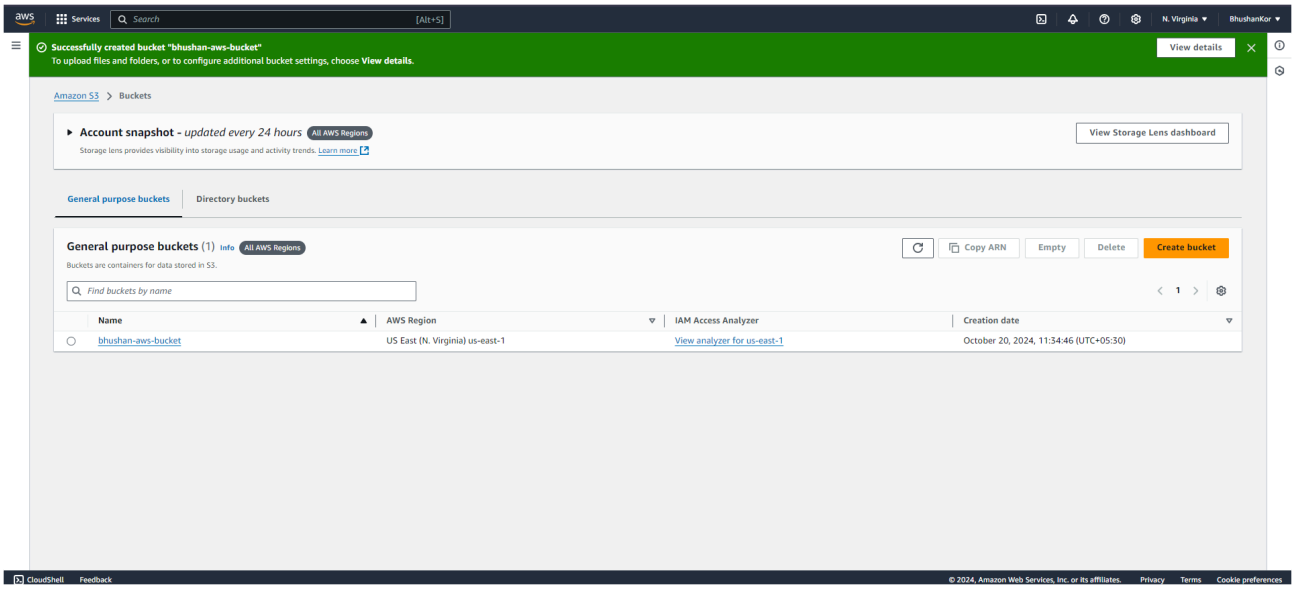
- Navigate to the **S3** service.
- Click on **Create Bucket**.
    - Bucket Type: **General Purpose.**
    - Provide a unique bucket name. **(bhushan-aws-bucket)**
    - **Uncheck the Block all public access option.**
    - Keep Rest of the things to default.
- You can upload a test image to ensure the bucket is working properly.





Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



3. Create a Lambda Function:

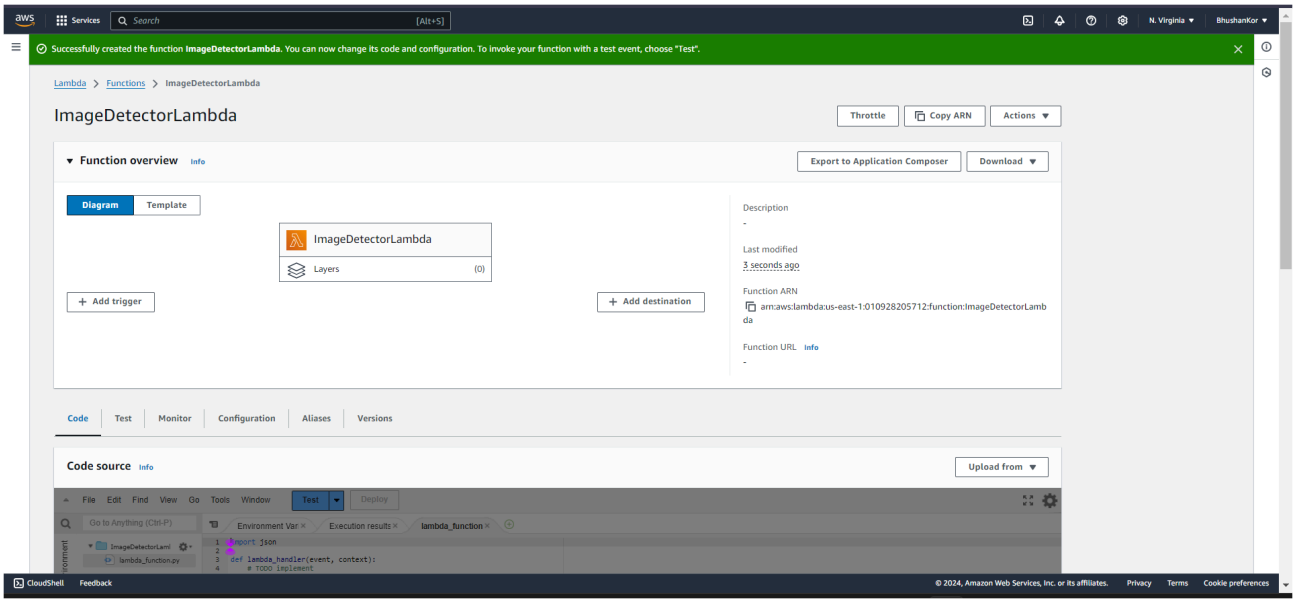
- Navigate to the **Lambda Console**.
- Click **Create Function**.
  - Choose **Author from scratch**.
  - Provide a function name **ImageDetectorLambda**.
  - Choose a free-tier eligible runtime **Python 3.12**.
  - Assign the **IAM role** created earlier **AWS_Case_Study**.
  - Keep Rest of things to default.
- In the **Function Code** section, add the logic to log when the image is uploaded to S3.

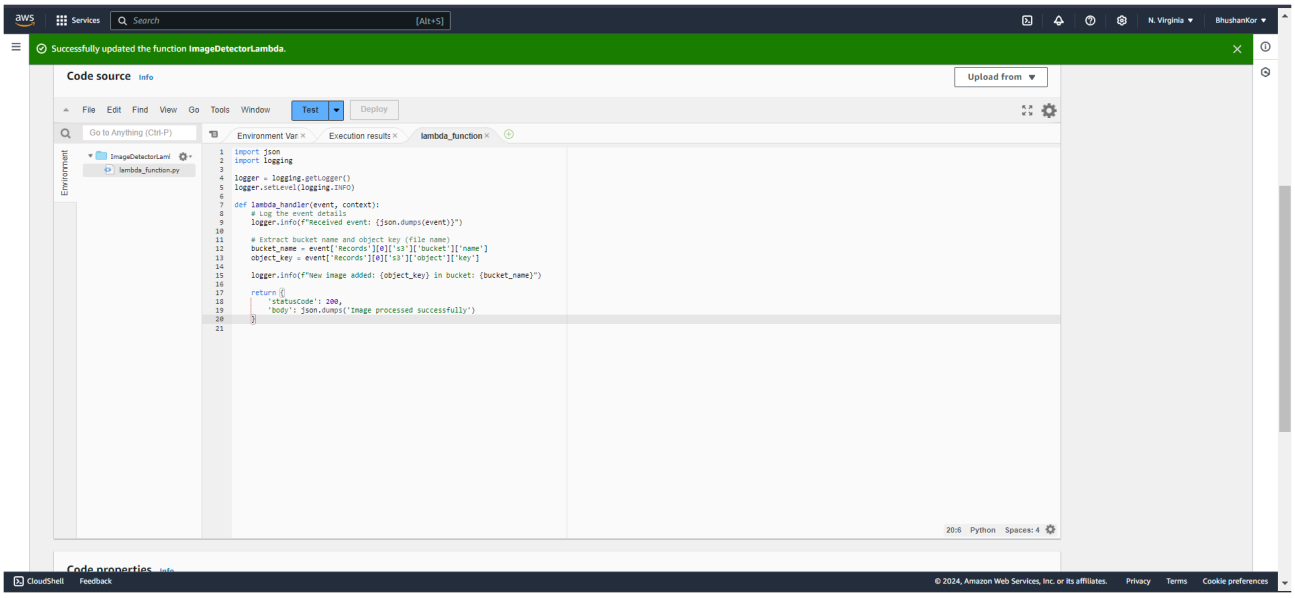
  Code:

```
import json
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    # Log the event details
    logger.info(f"Received event: {json.dumps(event)}")
    # Extract bucket name and object key (file name)
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    object_key = event['Records'][0]['s3']['object']['key']
    logger.info(f"New image added: {object_key} in bucket: {bucket_name}")
    return {
        'statusCode': 200,
```

```
        'body': json.dumps('Image processed successfully.')
    }
```

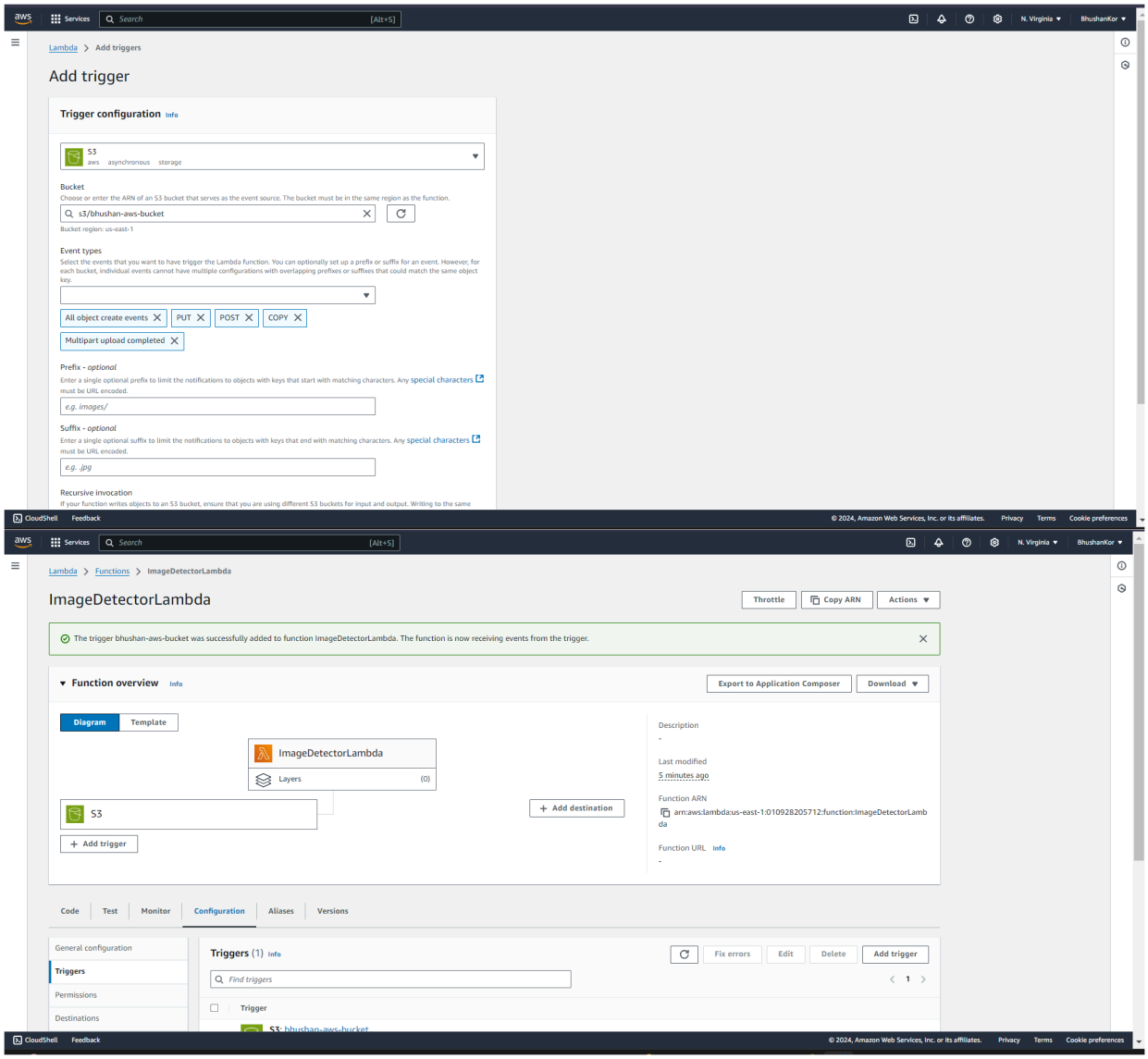
- Now Add the **Trigger** (Click on Trigger).
  - In Trigger Configuration Select S3.
  - Select Bucket which we have created.**(bhushan-aws-bucket)**
  - Select Event Types
    - All object create events
    - PUT
    - POST
    - COPY
    - Multipart upload completed



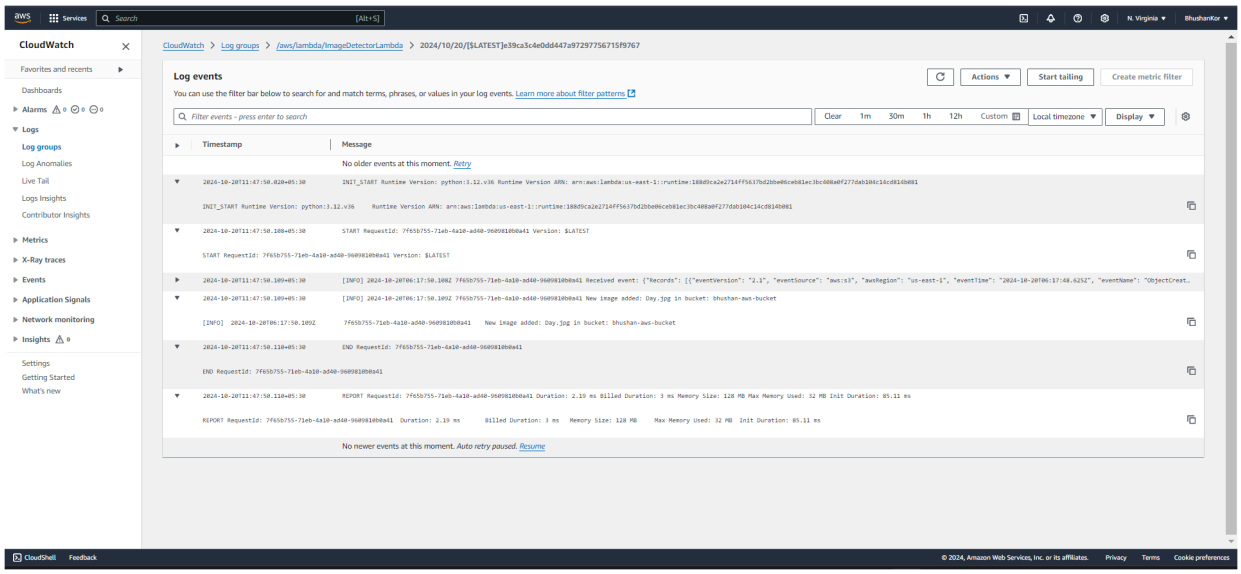


Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



- Now Let's test that Lambda is properly working by uploading image and checking logs.



Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

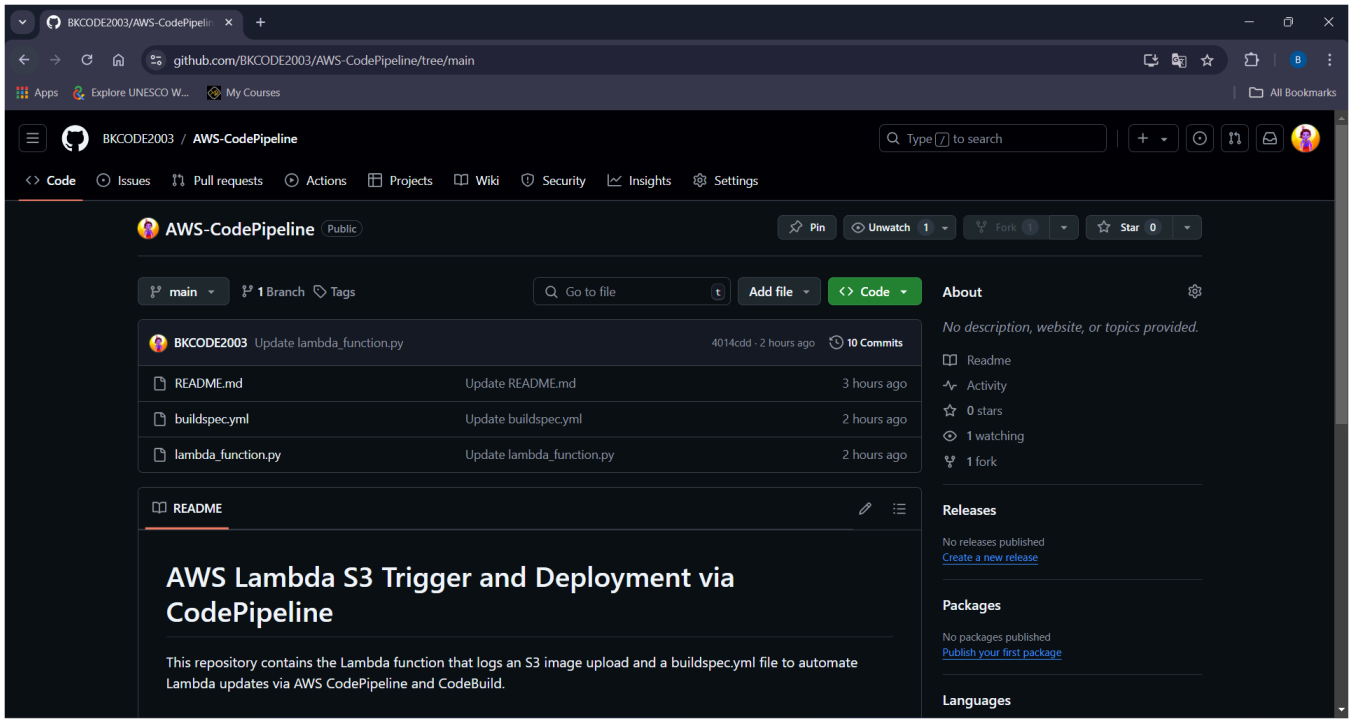
4. Create a Github Repository:

- Create Repositorywith name **AWS-CodePipeline.**
- Add the file **buildspec.yml** and **lambda_function.py**.

buidspec.yml

```
version: 0.2
phases:
  install:
    commands:
      - pip install awscli  # Ensure AWS CLI is available
  build:
    commands:
      - echo "Packaging Lambda function..."
      - zip -r lambda_function.zip .
      - echo "Updating Lambda function in AWS..."
      - aws lambda update-function-code --function-name ImageDetectorLambda --zip-file
fileb://lambda_function.zip
```

lambda_function.py Same as in step 3



5. Create a CodePipeline:

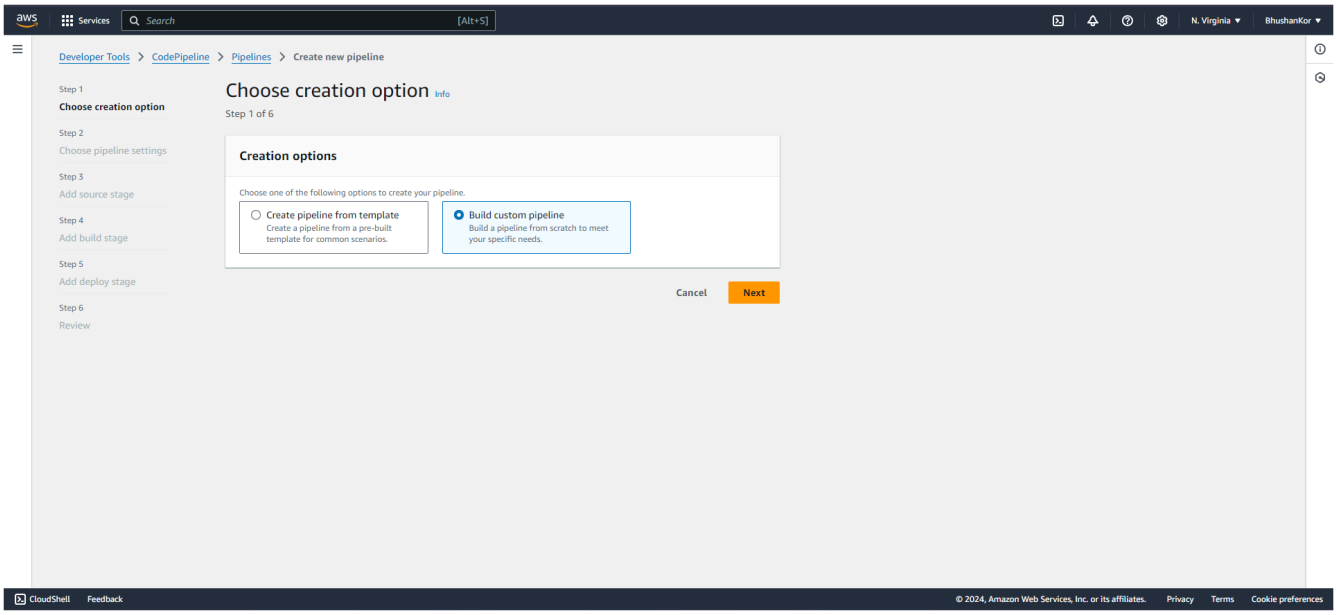
- Go to **AWS CodePipeline** in the AWS Management Console and create a new pipeline.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

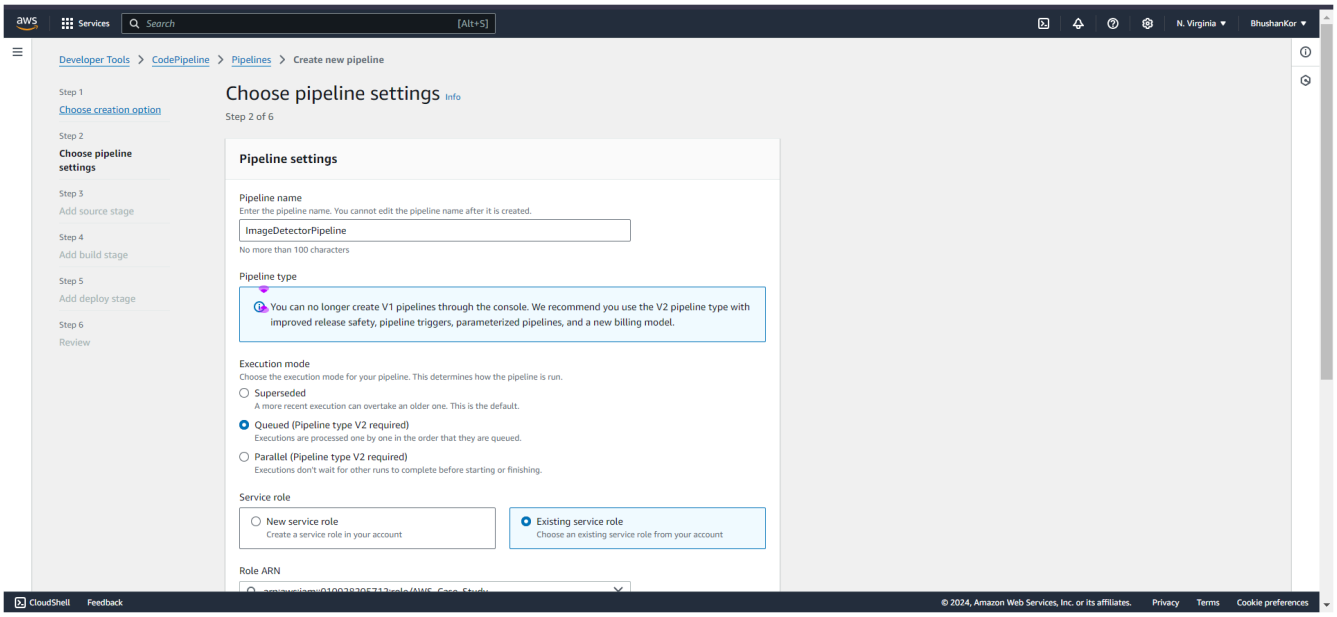
Step 1:Choose Creation option.

- In creation option Select **Build Custom pipeline.**



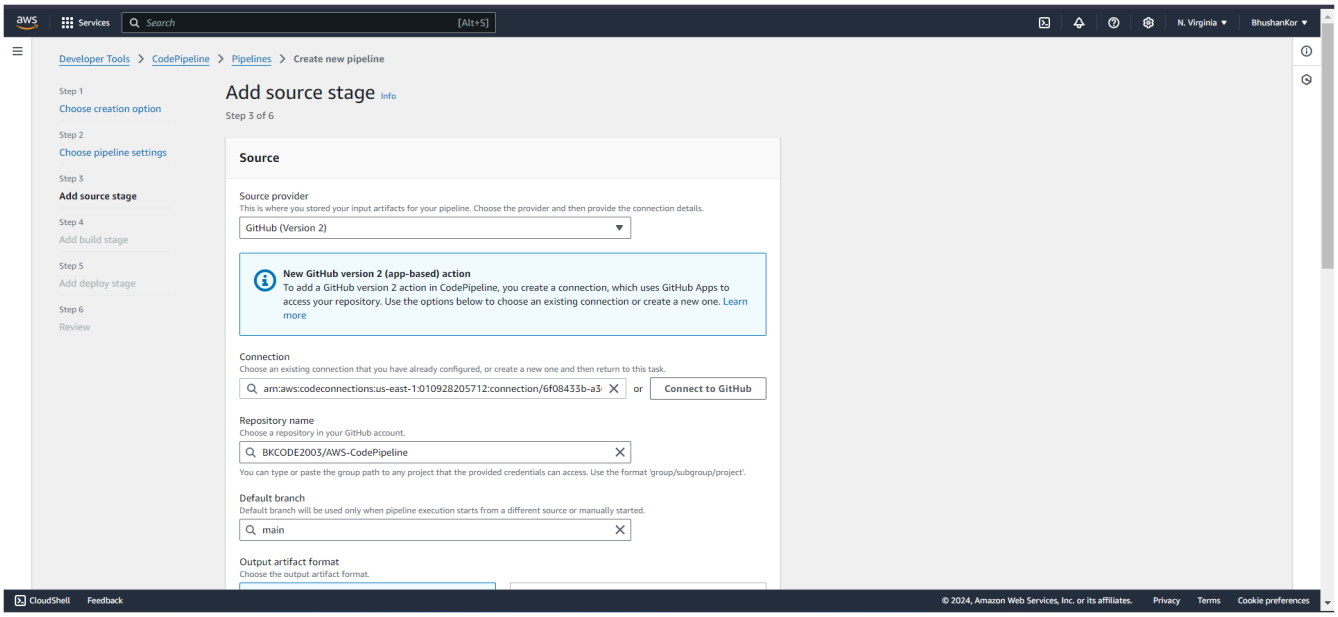
Step 2:Choose Pipeline Settings.

- Pipeline Name: **ImageDetectorPipeline** .
- Select Existing Role : **AWS_Case_Study** .
- Keep Rest all to default.



Step 3: Add Source Storage.

- Source provider: **GitHub (Version 2)** .
- Connect Your account where you have created the repository. **(BKCODE2003)**
- Select Repository. **(BKCODE2003/AWS-CodePipeline)**
- Select Branch to **main**.
- In Trigger Just add **main** in include.
- Keep Rest all to default.

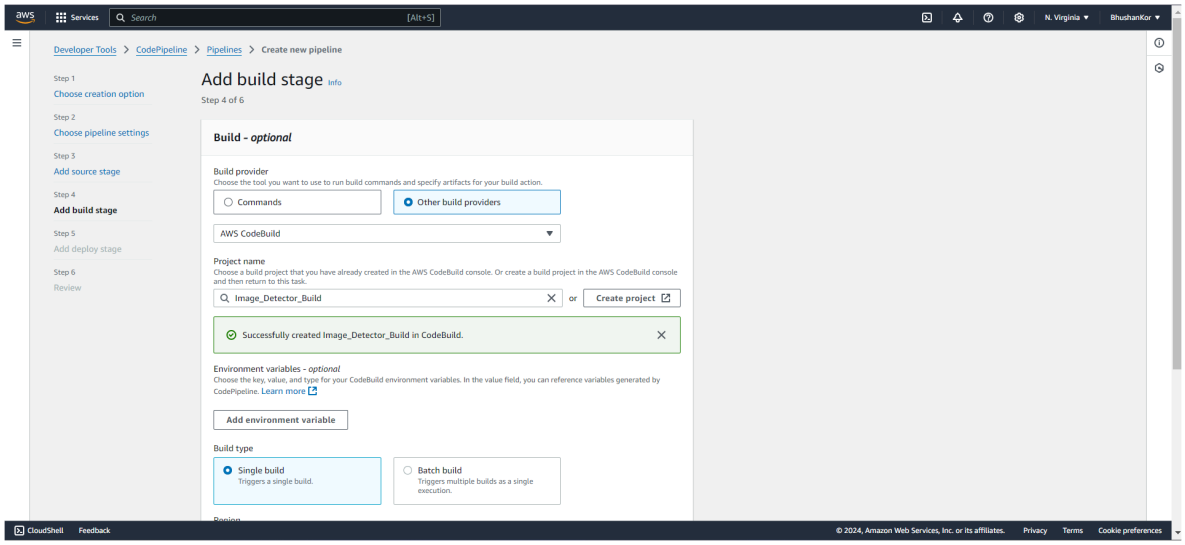


Step 4: Add Build Stage.

- Select **Other build Provider**.
- Select **AWS CodeBuild**.
  - Now Click on **Create Project**.
  - Project name: **Image_Detector_Build**
  - Enable public access.
  - Select Existing Role : **AWS_Case_Study** On 2 Places.
  - Buildspec: **Use a buildespec file** .
  - Buildspec name: **buildspec.yml** .
  - Keep Rest all to Default.
- Select Created Project.
- Keep Rest all to default.

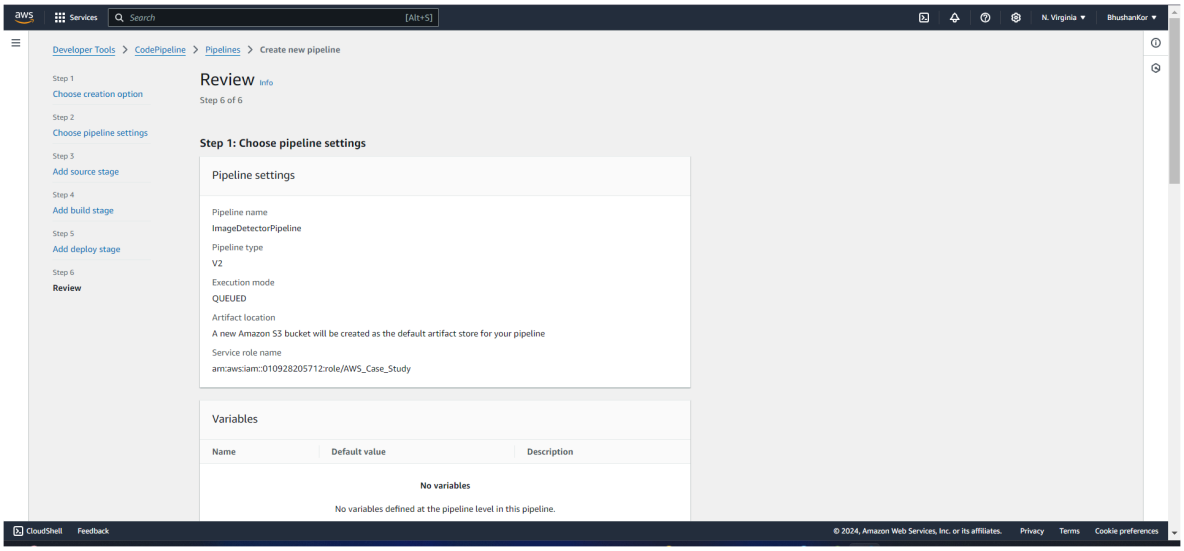
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

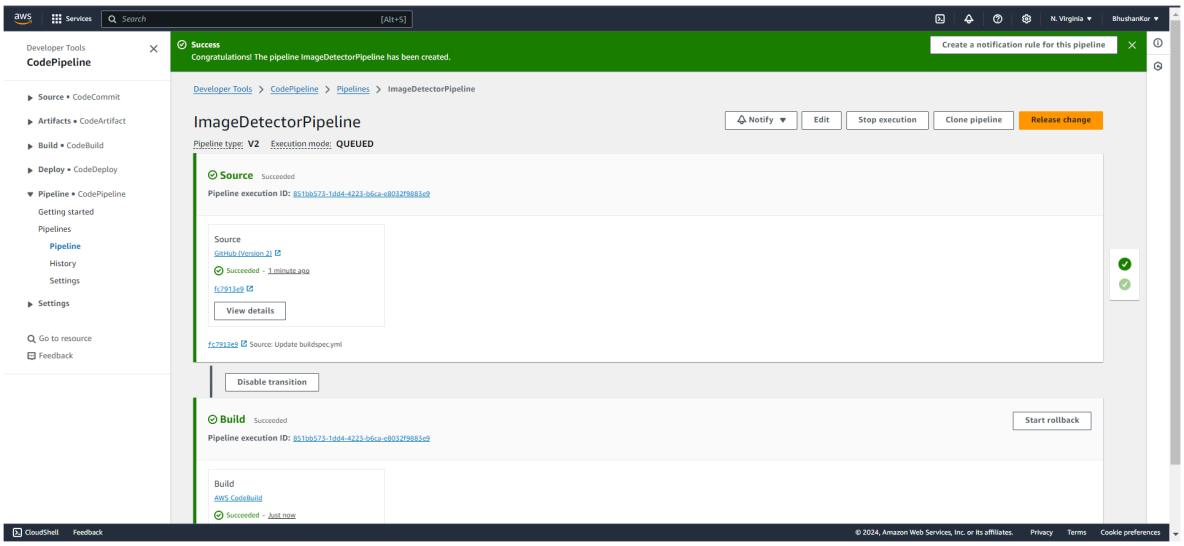
Division: D15C **Roll No: 28**



Step 5: Skip The Deploy Stage.

Step 6: Review the Pipeline and Click on Create.





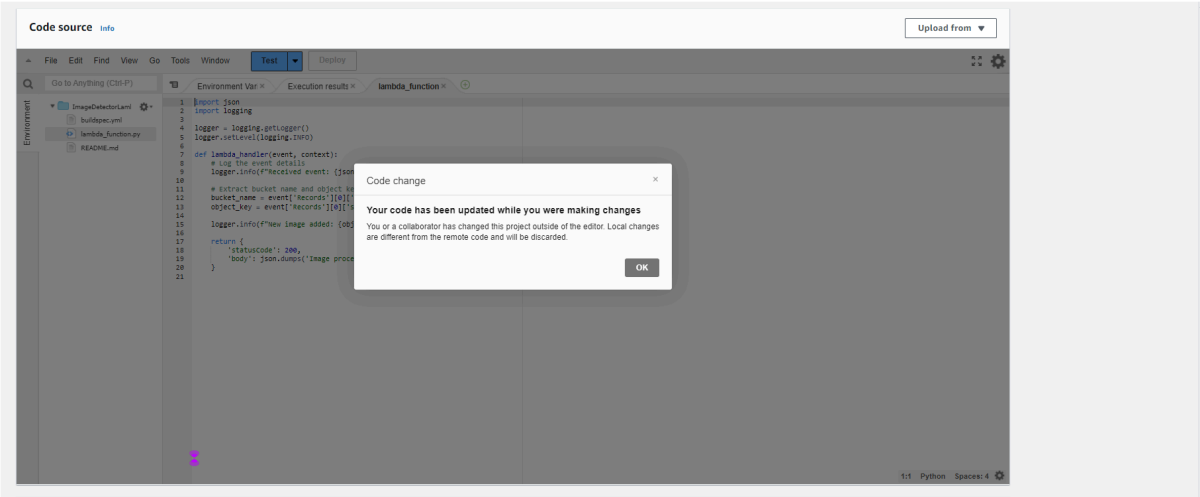
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

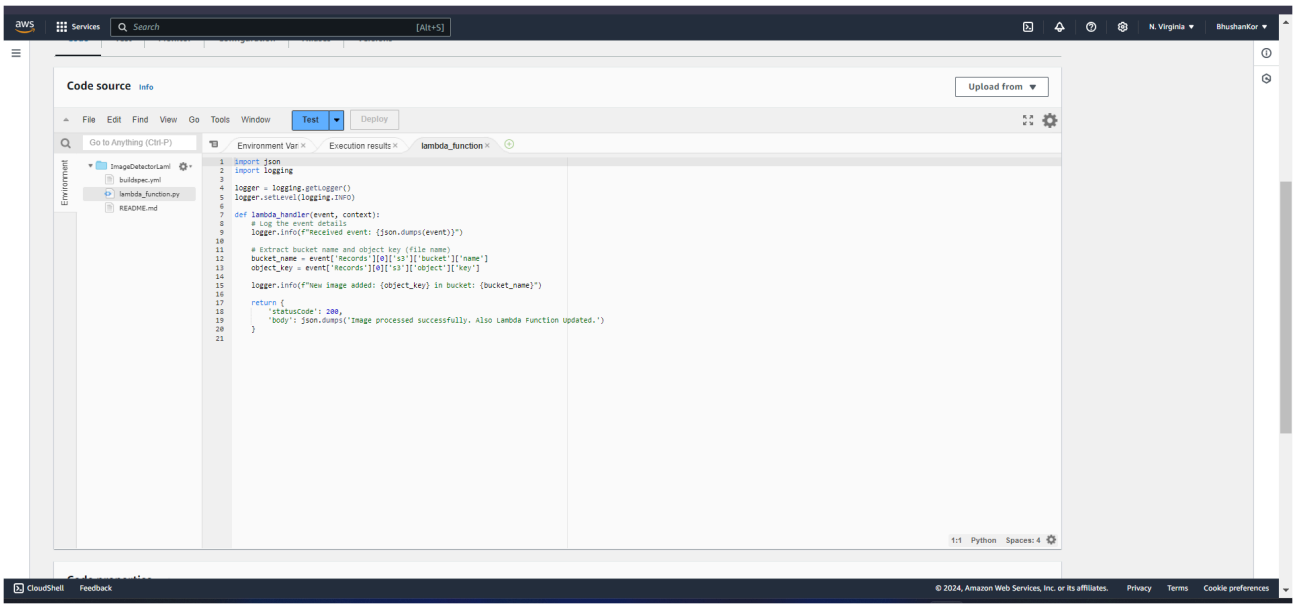
Division: D15C **Roll No: 28**

6.Test the Pipeline:

- Push/Update code to your GitHub repository.
- CodePipeline will trigger the CodeBuild project, which will package the code and update the Lambda function using the AWS CLI command aws lambda update-function-code.

Lambda After push of new code in repository .





Conclusion: In this case study, we successfully implemented a serverless image processing workflow using AWS Lambda, S3, and CodePipeline. The solution demonstrated how Lambda functions can be triggered by S3 events, allowing real-time image processing without manual intervention. By leveraging CodePipeline, we automated the deployment of updates to the Lambda function, ensuring a streamlined CI/CD process. Testing confirmed that the workflow functions as expected, with images uploaded to S3 triggering the Lambda function, which logs the event and processes the image.