Task 1: AWS

1. Create an S3 Bucket

a. Sign in to AWS Console

- Go to the **AWS Management Console**.
- Navigate to the **S3** service from the Services menu.

b. Create a New S3 Bucket

- Click on the **"Create bucket"** button.
- My bucket name is ragini-devops-bucket.
- Select the AWS Mumbai **region Which is** closest to me.

Important Settings:

- **Uncheck** "Block all public access" under "Bucket settings for Block Public Access."
- Acknowledge the warning about making the bucket public by checking the box.
- Click **"Create bucket"**.

c. Enable Static Website Hosting

- Click on the bucket name Which created
- Go to the **Properties** tab.
- Scroll down to **Static website hosting**.
- Click **Edit**.
- Enable **Static website hosting**:

    - Select **"Host a static website"**.
    - Specify **index document** (example: `index.html`).
    - (Optional) Specify **error document** (example: `error.html`).
- Click **Save changes**.

d. Upload Your Website Files

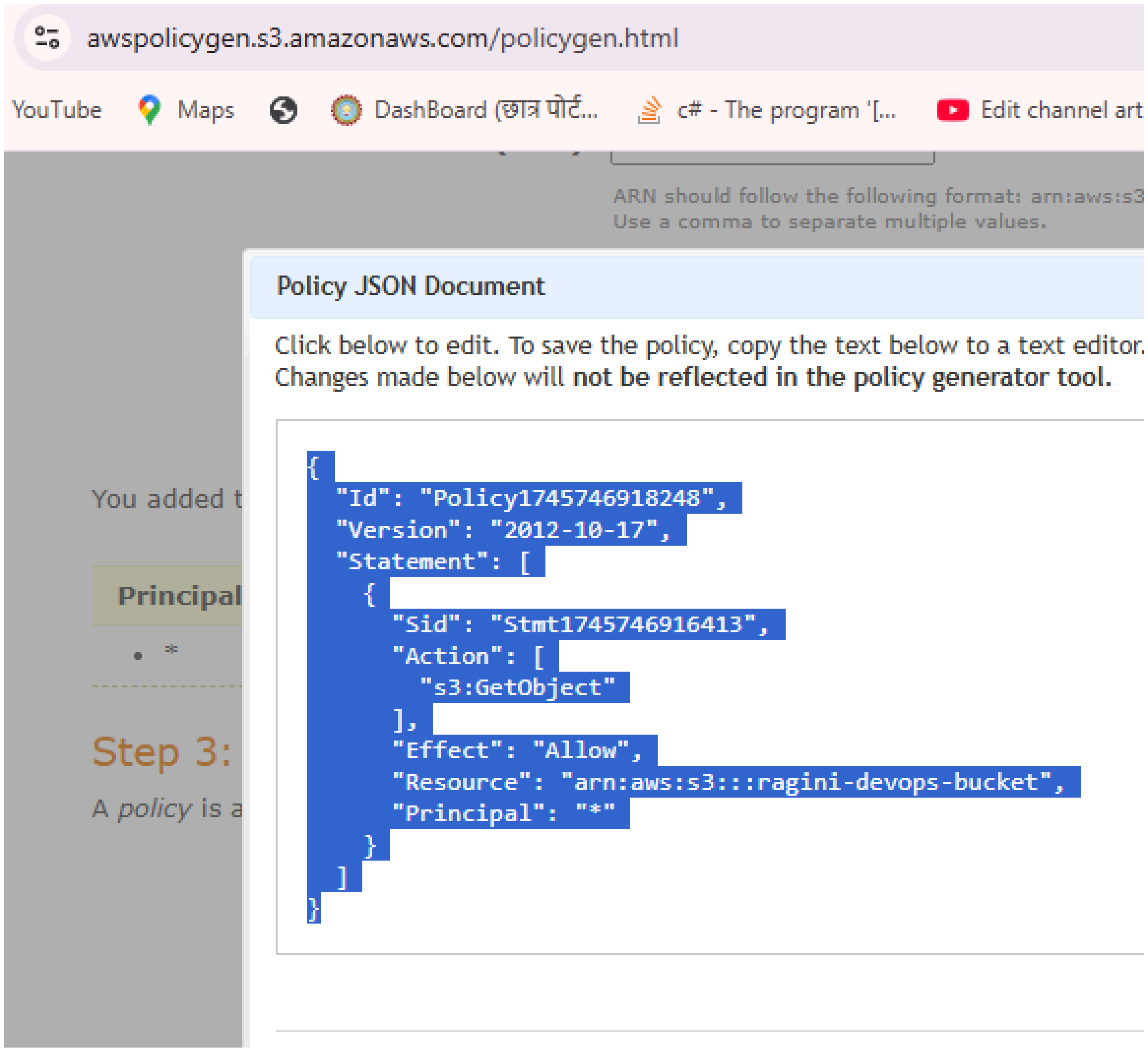
- Go to the **Objects** tab inside your bucket.
- Click **Upload** → **Add Files** → Choose your `index.html` file and index.css and image1,image2
- Click **Upload**.

e. Set Bucket Policy for Public Access

- In **Permissions** tab.
- Under **Bucket Policy**, click **Edit**.
- I used policy generator to create json format policyPaste the following policy to allow public read access:

json
CopyEdit

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::yourname-static-website-bucket/*"
    }
  ]
}
```



ARN should follow the following format: arn:aws:s3
Use a comma to separate multiple values.

Policy JSON Document

Click below to edit. To save the policy, copy the text below to a text editor.
Changes made below will **not be reflected in the policy generator tool.**

```
{
  "Id": "Policy1745746918248",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1745746916413",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::ragini-devops-bucket",
      "Principal": "*"
    }
  ]
}
```

- Click **Save changes**.

f. Test the Static Website

- In the **Properties** tab.
- In the Static Website Hosting section, copy the **Bucket website endpoint**.
- Paste it into your browser — you should see your HTML page live!

2. Set Up an EC2 Instance:

- Connected via SSH using key pair authentication.

- Installed Apache web server

```
 # Update package list
    sudo apt update


    # Install Apache2

    sudo apt install apache2 -y


    # Start Apache service

    sudo systemctl start apache2


    # Enable Apache to start on boot

    sudo systemctl enable apache2
```

.

- Created and hosted a simple HTML page.

```
    echo "<html><body><h1>Hello this is DevOps interview
project!</h1></body></html>

    " | sudo tee /var/www/html/index.html
```

- Website accessible at:

```
    http://13.234.240.58/
```

3. Configure Security Group:

Here are the steps to implement this configuration:

1. **Go to the EC2 Console** and click on **Security Groups**.
2. **Select the Security Group** associated with your EC2 instance.
3. In the **Inbound Rules** tab, click **Edit inbound rules**.
4. **Add Rule**:
    - **Type**: HTTP (or manually set port 80).
    - **Protocol**: TCP.
    - **Port Range**: 80.
    - **Source**: Select **My IP** to automatically fill in your public IP, or manually enter your IP address in CIDR format, e.g., `203.0.113.5/32` (this is a single IP).
5. **Save Rules**.

After doing this, only your specific IP will be able to access your EC2 instance via HTTP. All other IPs will be blocked.

1. **Log into AWS Management Console**:
    - Go to the **AWS Lambda** service.
2. **Create a Lambda Function**:
    - Click on **Create function**.
    - Choose **Author from scratch**.
    - **Function name**: Name your function (`S3EventLogger`).
    - **Runtime**: Select **Python 3.9**

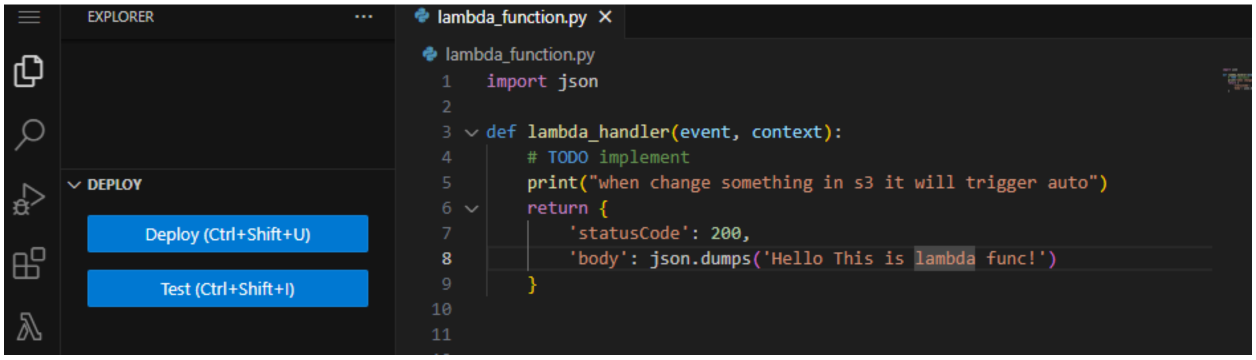
- **Role**: Select an **IAM role**. If you don't have an appropriate role, you can create one:
    - Choose **Create a new role with basic Lambda permissions**.
    - Lambda needs permission to write logs to **CloudWatch**, so choose the **AWSLambdaBasicExecutionRole** policy.
    - Optionally, choose **AmazonS3ReadOnlyAccess** if you want the Lambda function to access the S3 event details.

3. **Create Function**:
    - After filling in the details, click **Create function**.

Step 2: Added the default Lambda Function Code

1. Once the function is created, scroll down to the **Function code** section.
2. Replace the default code with the following Python code:

python



```python
import json

def lambda_handler(event, context):
    # TODO implement
    print("when change something in s3 it will trigger auto")
    return {
        'statusCode': 200,
        'body': json.dumps('Hello This is lambda func!')
    }
```

3. Click **Deploy** to save the function.

Step 3: Create an S3 Event Trigger

1. **Go to the S3 Console**:
    - Navigate to the **S3 Console** and select the bucket you want to use.
2. **Set up an Event Notification**:
    - Go to the **Properties** tab of the bucket.
    - Scroll down to **Event notifications** and click **Create event notification**.
    - **Event types**: Select **All object create events** (or a specific type, like `ObjectCreated`).
    - **Destination**: Choose **Lambda function**.
    - **Lambda function**: Select the Lambda function you created (`S3EventLogger`).
3. **Save** the event notification.

Step 4: Configure Permissions for S3 to Trigger Lambda

1. **Go to IAM Console**:
    - In the **IAM Console**, select the Lambda execution role that was created when you made the Lambda function.
2. **Attach Policy to Lambda Execution Role**:
    - Add a policy to the Lambda execution role that allows **S3** to trigger the Lambda function. Use the following example policy:

json

CopyEdit

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource":
"arn:aws:lambda:REGION:ACCOUNT_ID:function:S3EventLogger",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:s3:::YOUR_BUCKET_NAME"
        }
      }
    }
  ]
}
```

3. **Attach the policy** to the Lambda execution role.

Step 5: Test the Lambda Function

1. **Upload a file** to your S3 bucket (or perform the action that triggers the event, such as object creation).
2. **Go to CloudWatch Logs**:
   - Navigate to the **CloudWatch Console**.
   - Under **Logs**, find the log group that corresponds to your Lambda function.

    - Check the logs for the Lambda function to verify that the event details (like the bucket name, object key, and event time) were logged.

Step 6: Monitor Logs

- You can now monitor the Lambda function's logs in **CloudWatch Logs** whenever a new object is created in the S3 bucket.
- Each event (e.g., object creation) will be logged with details such as:
    - **Event Time**: When the event happened.
    - **Bucket Name**: The S3 bucket name.
    - **Object Key**: The object key (filename) that was created.

Summary of Actions:

1. **Create Lambda function** with Python code to log event details.
2. **Create an S3 event notification** to trigger the Lambda function on object creation.
3. **Attach the appropriate permissions** for S3 to trigger Lambda.
4. **Deploy and test** the Lambda function.