Vivekanand Education Society's

Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai,, Approved by AICTE & Recognized by Govt. of Maharashtra
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

A.Y. 2024-25

Advance DevOps Lab
Assignment 02

Aim:

- Deploying AWS Infrastructure Using Terraform: A Hands-On Approach with S3, SQS, and Lambda Integration

Roll No.	32
Name	Jeetu Mamtora
Class	D15B
Subject	Advance DevOps Lab
LO Mapped	LO1: To understand the fundamentals of Cloud Computing and be fully proficient with Cloud based DevOps solution deployment options to meet your business requirements.
Grade:	

- **AIM :** To develop a website and host it on your local machine on a VM Reference and hostinga static website on Amazon S3 (AWS).
- **THEORY :**
  Terraform is an open-source infrastructure as code (IaC) tool that allows users to define and provision infrastructure using a declarative configuration language. It enables the management of cloud resources in a version-controlled manner, making deployments consistent and repeatable. In this experiment, we will focus on integrating three key AWS services: Amazon S3 (Simple Storage Service), Amazon SQS (Simple Queue Service), and AWS Lambda.
- **Amazon S3** is a scalable object storage service designed for data backup, archiving, and analytics. It allows users to store and retrieve any amount of data from anywhere on the web.
- **Amazon SQS** is a fully managed message queuing service that enables decoupling and scaling microservices, distributed systems, and serverless applications. It provides reliable and secure message transmission.
- **AWS Lambda** is a serverless compute service that runs code in response to events and automatically manages the underlying compute resources. It is commonly used to process data in real-time, such as files uploaded to S3 or messages received from SQS.
  In this experiment, we will set up an S3 bucket to store files, configure an SQS queue to handle messages related to the uploaded files, and create a Lambda function that processes the files whenever they are uploaded to the S3 bucket.

1. Install Terraform

First, you need Terraform installed on your machine.

- Go to the Terraform download page.
- Download and install it according to your operating system (Windows, macOS, or Linux).
- After installation, you can verify it by typing this command in your terminal:

bash

Copy code

```
terraform -v
```

If installed correctly, it should show the version number.

2. Set Up AWS Credentials

To connect Terraform with AWS, you need to configure AWS credentials. If you haven't already, follow these steps:

1. **Install AWS CLI** (Command Line Interface):
   - Follow the instructions at AWS CLI Installation.
2. **Configure AWS CLI**:
   - Once the CLI is installed, run the command:

```
aws configure
```

   - You'll be prompted for your AWS **Access Key**, **Secret Key**, **Region**, and **output format**.
   - If you don't have these keys, you can create an IAM user with programmatic access in the AWS Console under **IAM > Users**.

3. Create a New Directory for Your Project

Now, let's create a directory where you'll write your Terraform configuration and Lambda code:

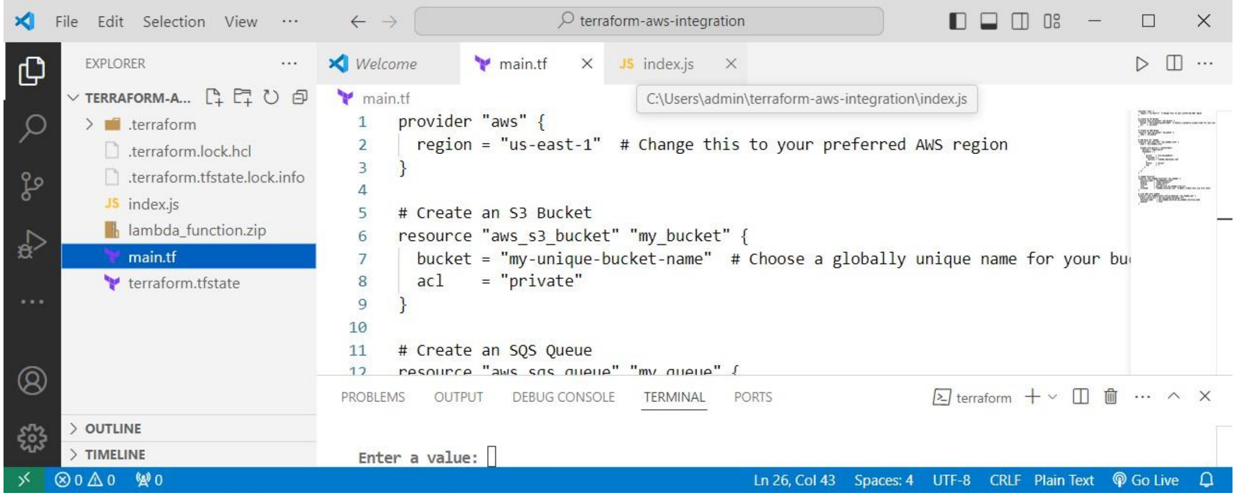
```
mkdir terraform-aws-integration
cd terraform-aws-integration
```

4. Write the Terraform Configuration File (main.tf)

In this step, we define the AWS services (S3, SQS, Lambda) we want to create. In the same directory, create a new file called main.tf with the following content:

```
provider "aws" {
  region = "us-east-1"  # Change this to your preferred AWS region
}
```

```
# Create an S3 Bucket
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-bucket-name"  # Choose a globally unique name for your bucket
  acl    = "private"
}

# Create an SQS Queue
resource "aws_sqs_queue" "my_queue" {
  name = "my-queue"
}

# IAM Role for Lambda
resource "aws_iam_role" "my_lambda_role" {
  name = "my_lambda_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action    = "sts:AssumeRole"
        Principal = {
          Service = "lambda.amazonaws.com"
        }
        Effect    = "Allow"
        Sid       = ""
      },
    ]
  })
}

# Lambda Function
resource "aws_lambda_function" "my_lambda" {
  function_name = "myLambdaFunction"
  handler       = "index.handler"
  runtime       = "nodejs14.x"
  role          = aws_iam_role.my_lambda_role.arn
  filename      = "lambda_function.zip"  # We’ll create this zip file later
}

# Link SQS with Lambda
resource "aws_lambda_event_source_mapping" "my_lambda_sqs" {
  event_source_arn = aws_sqs_queue.my_queue.arn
  function_name    = aws_lambda_function.my_lambda.function_name
  enabled          = true
}
```



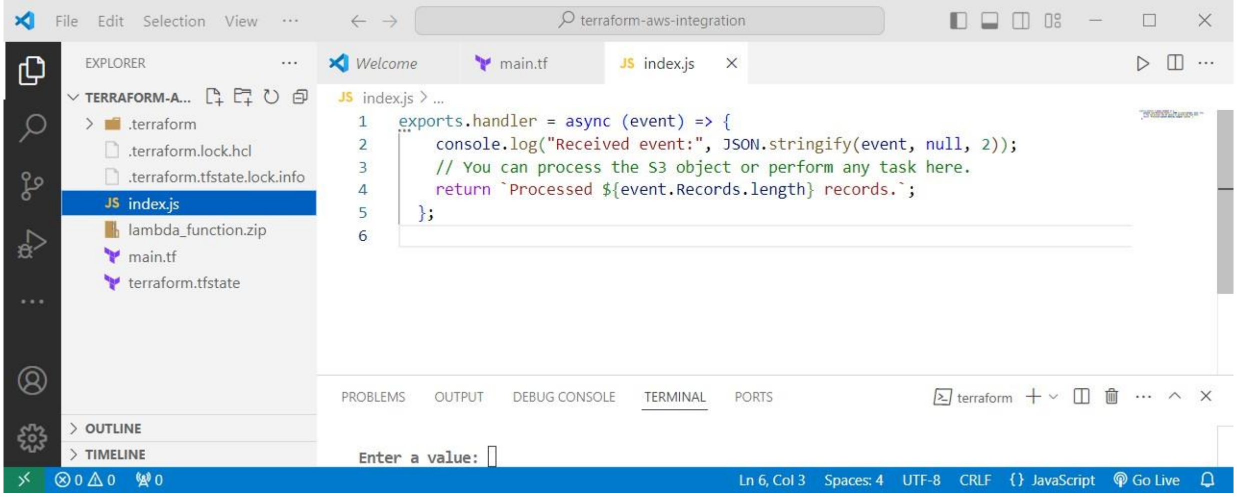
```
provider "aws" {
  region = "us-east-1"  # Change this to your preferred AWS region
}

# Create an S3 Bucket
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-bucket-name"  # Choose a globally unique name for your bu
  acl    = "private"
}

# Create an SQS Queue
resource "aws_sqs_queue" "my_queue" {
```

```
Enter a value:
```

5. Create the Lambda Function Code

You need to create a small program that will be triggered when a file is uploaded. In the same directory, create a file called index.js with this content:

```
exports.handler = async (event) => {
  console.log("Received event:", JSON.stringify(event, null, 2));
  // You can process the S3 object or perform any task here.
  return `Processed ${event.Records.length} records.`;
};
```



```
exports.handler = async (event) => {
    console.log("Received event:", JSON.stringify(event, null, 2));
    // You can process the S3 object or perform any task here.
    return `Processed ${event.Records.length} records.`;
};
```

```
Enter a value:
```

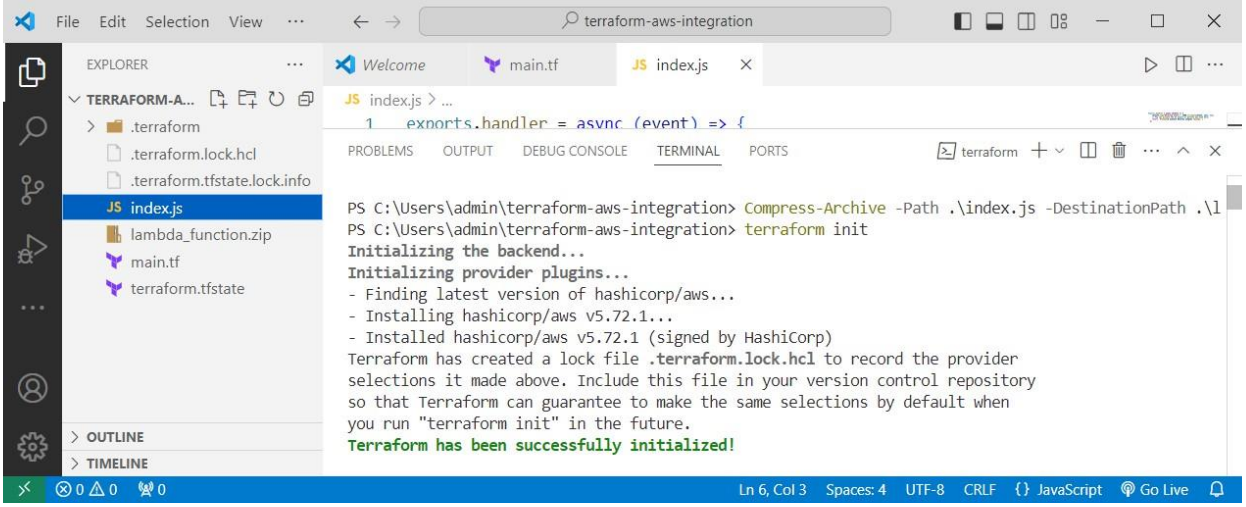
Now, **zip** this file so it can be uploaded to AWS Lambda:

```
Compress-Archive -Path .\index.js -DestinationPath .\lambda_function.zip
```

6. Initialize Terraform

Run this command to download necessary plugins and initialize your Terraform project:

```
terraform init
```



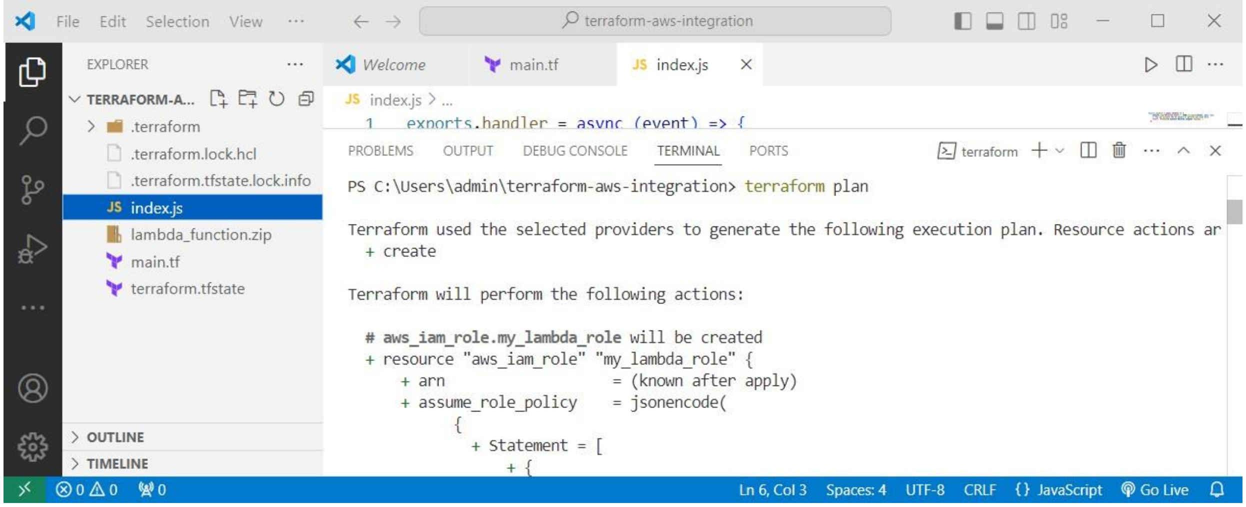
```
exports.handler = async (event) => {
```

```
PS C:\Users\admin\terraform-aws-integration> Compress-Archive -Path .\index.js -DestinationPath .\l
PS C:\Users\admin\terraform-aws-integration> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
Terraform has been successfully initialized!
```

7. Review the Execution Plan

Before creating the resources, check what Terraform plans to do:

terraform plan



```
exports.handler = async (event) => {
```

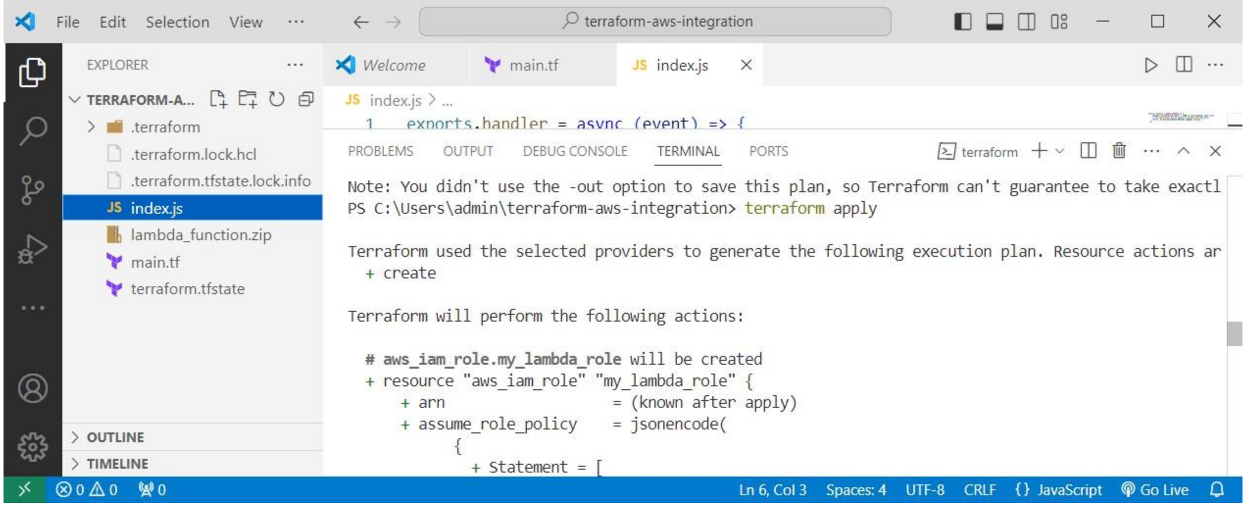
```
PS C:\Users\admin\terraform-aws-integration> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions ar
  + create

Terraform will perform the following actions:

  # aws_iam_role.my_lambda_role will be created
  + resource "aws_iam_role" "my_lambda_role" {
      + arn                   = (known after apply)
      + assume_role_policy    = jsonencode(
            {
              + Statement = [
                  + {
```

8. Apply the Configuration

If the plan looks correct, run this command to create the resources in AWS:

terraform apply

You will be asked to type yes to confirm.



```
exports.handler = async (event) => {
```

```
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactl
PS C:\Users\admin\terraform-aws-integration> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions ar
  + create

Terraform will perform the following actions:

  # aws_iam_role.my_lambda_role will be created
  + resource "aws_iam_role" "my_lambda_role" {
      + arn                   = (known after apply)
      + assume_role_policy    = jsonencode(
            {
              + Statement = [
```

9. Testing the Setup

Once the resources are created:

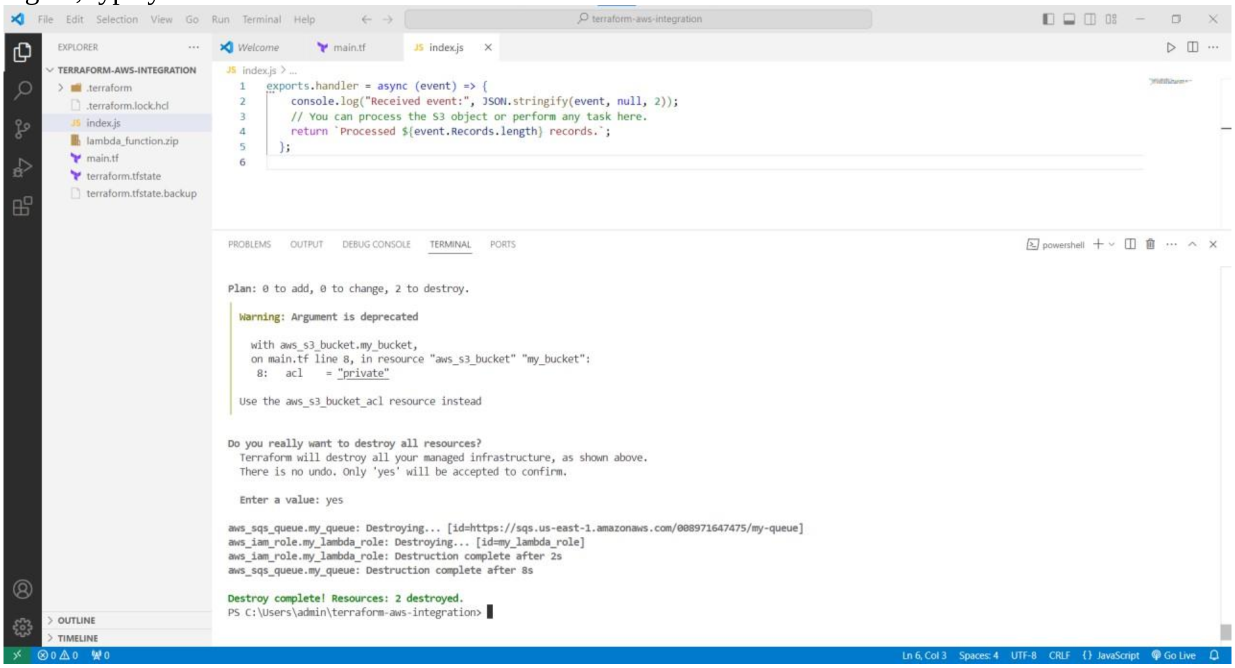
1. Upload a file to your S3 bucket (you can do this via the AWS Management Console or CLI).
2. Check your SQS queue – a message will be sent indicating the file upload.
3. Go to AWS Lambda (in the AWS Console) and view the logs in CloudWatch to see if the Lambda function processed the file.

10. Clean Up Resources

When you are done with the experiment, you can remove all the created resources using this command:

```
terraform destroy
```

Again, type yes to confirm.



```
exports.handler = async (event) => {
    console.log("Received event:", JSON.stringify(event, null, 2));
    // You can process the S3 object or perform any task here.
    return `Processed ${event.Records.length} records.`;
  };

```

```
Plan: 0 to add, 0 to change, 2 to destroy.

  Warning: Argument is deprecated

    with aws_s3_bucket.my_bucket,
    on main.tf line 8, in resource "aws_s3_bucket" "my_bucket":
     8:   acl    = "private"

  Use the aws_s3_bucket_acl resource instead

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_sqs_queue.my_queue: Destroying... [id=https://sqs.us-east-1.amazonaws.com/008971647475/my-queue]
aws_iam_role.my_lambda_role: Destroying... [id=my_lambda_role]
aws_iam_role.my_lambda_role: Destruction complete after 2s
aws_sqs_queue.my_queue: Destruction complete after 8s

Destroy complete! Resources: 2 destroyed.
PS C:\Users\admin\terraform-aws-integration>
```

Conclusion

By following these steps, you have successfully deployed a basic AWS infrastructure using Terraform that integrates S3, SQS, and Lambda. This setup allows for file uploads to S3, with messages sent to SQS and processed by a Lambda function, demonstrating a scalable and serverless architecture.