Vivekanand Education Society's
Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Advance DevOps Lab

Experiment No.	Assignment-2
Title.	
Roll No.	51
Name	Krushikesh Shelar
Class	D15 C
Subject	Advance DevOps
Lab Outcome	LO6:To engineer a composition of nano services using AWS Lambda and Step Functions with the Serverless Framework
Signature:	
Grade:	5

05/05

Advance Devops Assignment 2

Create a REST API with the Serverless framework

Steps to create a REST API with serverless framework

Install serverless framework globally using the following command on the terminal :-

npm install -i serverless

This command installs the serverless framework on your machine globally using npm. It allows you to create, manage and deploy serverless applications across various cloud providers, including AWS.

Create a new Services with AWS Node.js Template

serverless create --template aws-nodejs --path rest-api

This command initialises a new Serverless service called rest-api. It create a folder containing basic files and a template specifically configured for building serverless application using Node.js on AWS Lambda

Navigate to the project directory:

cd rest-api

This command changes directory into the newly created project directory to manage files and configurations specific to our service.

Initialize Node.js project and install dependencies.

npm init -y

npm install express serverless-http

The express dependencies builds the REST API and Serverless http integrate express with Aws Lambda.

5.] Edit the serverless yml file to include

```
service: rest-api
provider:
  name: aws
  runtime: nodejs14.x
  stage: dev
  region: us-east-1
functions:
  app:
    handler: handler.app
    event:
      - http:
          path: /
          method: any
```

This configuration specifies the service name, AWS provider setting and deploys the Lambda function with HTTP event trigger.

6.] Edit handler.js to add the express app

```
const express = require('express')
const serverless = require('serverless-http');
const app = express();
app.get('/helloworld', (req, res) => res.json({ message: 'Hello world' }));
module.exports.app = serverless(app);
```

This creates a simple Express app with a single route /hello world and exports it in a Lambda-compatible format.

Deploy the service:

serverless deploy

Deploys the API to AWS setting up resources like Lambda and API Gateway. A URL is generated for testing

Test the deployed API:

curl https://<api-id>.execute-api.<region>.amazonaws.com/dev/helloworld

Using above command, it returns a JSON message {"message": "Hello World"}.

Redeploy after updates:

serverless deploy

After modifying the code, redeploy it to update the API with our changes.

Remove the service

serverless remove

The above command removes all AWS resources associated with the API, ensuring that there are no charges for unused services.

Q2] Case study for sonarqube.

- Create your own profile in sonarqube for testing project quality.
- Use SonarCloud to analyze your Github code
- Install SonarLint in your Java IntelliJ IDE or Eclipse IDE and analyse your Java code.
- Analyze Python Project with sonarqube
- Analyze Node.js project with sonarqube.
- ~~Analyze~~

→

- Create your own profile in sonarqube

1] Download and install sonarqube from the official website. Unzip the file and start the server by running
./bin/windows-x86-64/StartSonar.bat.
This launches sonarqube locally and can be accessed at http://localhost:9000

2] Log in to SonarQube using the default credentials (username: admin, password: admin). After logging in, change the password.

3] Navigate to Project tab, click on 'Create New Project' assign a project key and name and generate a project token.

- Use SonarCloud to analyze your Github code:

1] Signup for SonarCloud from the official website using your Github account.

In SonarCloud, under Projects > Create Project, choose your Github repository and grant SonarCloud access to it.
Add a sonar project properties file in the root of your repository with the following code:
Sonar.projectKey = <your_project_key>
sonar.organisation = <your-organisation>
sonar.host.url = https://sonarcloud.io

Use SonarScanner to analyze the code by running the following command: sonar-scanner
This uploads analysis results from your local machines to SonarCloud.

Install SonarLint in your Java IntelliJ or Eclipse IDE and analyze your java code.
Install sonarlint by going onto intellij or Eclipse going to Plugin's Marketplace and search for Sonarlint
Install and restart your IDE.
In the IDE, configure SonarLint by linking it to your SonarQube or SonarCloud project to sync the rules and profiles
Open a Java project and use SonarLint to analyze it. It will display issues, directly in the IDE while coding. SonarLint provides real time feedback on code quality based on SonarQube rules.

Analyze python project with SonarQube:-

1.] Set up a python code in a project and ensure that Sonarqube is running locally.

2.] Download and continue Sonar Scanner from its official website and in the sonar-project.properties file edit to include the following.

```
sonar.projectKey=python_project
sonar.language=py
sonar.sources=.
```

3.] Run the analyses of the project by executing the following from the project directory. Sonar scanner the results will be pushed to your local SonarQube server and the analysis is visible on the dashboard.

Analyse Node.js project with SonarQube.

1.] Set up a Node.js project.

2.] In SonarQube, ensure that all JS/TS plugins have been install

3.] Create a sonar-project.properties file in your project root and include sonar.projectKey=node_project sonar.language=js

4.] Run the analysis of the project by executing the Sonar-Scanner command.

SonarQube will analyze the Node.js project and show results on the dashboard, highlighting code quality bug and vulnerabilities.

At a large organization, your centralized operations team may get many repetitive infrastructure requests. You can use Terraform to build a "self-serve" infrastructure model that lets product teams manage their own infrastructure independently. You can create and use Terraform modules that codify the standards for deploying and managing services in your organisation, allowing teams to efficiently deploy services in compliance with your organization's practices. Terraform Cloud can also integrate with ticketing systems like ServiceNow to automatically generate new infrastructure requests.

1) Self Serve Infrastructure with Terraform:
- By using Terraform you can build reusable modules that encapsulate the best practices and compliance standards of your organization. These modules can be made available to various product teams, empowering them to provision infrastructure themselves while following the pre-defined standards.
- This reduces the bottleneck that a centralized operations team may face, as the product teams no longer need to await for manual approval or setup. Instead they can handle their own infrastructure needs within the boundaries set by the organization.

2) Terraform Modules for Standardization:
- Terraform modules are reusable templates that simplify the process of deploying infrastructure by codifying the standards for managing resources. Product teams can leverage these modules

to deploy resources that comply with organizational policies ensuring consistency and compliance across different environments.

- This helps maintain security, performance, and cost efficiency, as teams are not manually configuring infrastructure from scratch.

2] Integration with Ticketing Systems like ServiceNow

- Terraform cloud or Enterprise can integrate with tools like ServiceNow, automating the process of generating new infrastructure requests.
- For instance, when a product team submits a ticket for infrastructure resources, Terraform Cloud can automatically handle provisioning based on pre-approved templates reducing the needs for manual intervention from the operations team.

This model optimizes operational efficiency by delegating infrastructure management to product teams while still maintaining control over the infrastructure standards, and automating the request process through integrations.