Anushka Shahane
D15A (55)

05/10

ADVANCE DEVOPS
ASSIGNMENT 2

Q1) Create a REST API with the serverless Framework

Soln: The serverless Framework helps you deploy applications to cloud providers like AWS using a simplified configuration.

(1) Install serverless framework: Ensure you have Node.js installed, then install the serverless Framework using npm

```
npm install -g serverless
```

(2) Set up AWS credentials: Serverless uses AWS Lambda and API Gateway so configure your AWS Console

```
aws configure
```

Add your AWS Access Key, Secret key Region etc.

(3) Create a New Service: Create a new project using a Node.js template.

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

This creates a basic serverless service with a structure including serverless.yml and handler.js file for code.

(4) Define the REST API in serverless.yml:
Edit the serverless.yml to define the REST API endpoints.
For eg: service: rest-api-service

```
provider:
  name: aws
  runtime: nodejs14.x
functions:
  hello:
    handler: handler.hello
    events:
      - http:
          path: hello
          method: get
```

The handler.js file would contain the logic

(5) Deploy the Service: Deploy the API to AWS Lambda and API gateway using: serverless deploy

This deploys infrastructure and you'll get a URL to access API

(6) Testing: Use the URL provided to access your REST API. You can test it with tools like Postman or simply via your browser.

(2) Case Study for Sonarqube.

Soln:

SonarQube helps to automatically review your code for bugs, vulnerabilities, and code smells. The steps below cover Java, Python and Node.js analysis.

(1) Create a SonarQube Profile:
Go to SonarQube Cloud and create an account You can link this account to your Github profile to analyze code directly from repositories.
Create a project in SonarCloud and connect it with your Github repository.

(2) Analyze Code on SonarCloud:
For your Github repository, configure it with SonarCloud. This can be done using CI pipelines (eg. Github Actions) or manually.

Example of Github Actions YAML for SonarCloud

```
name: SonarCloud
on:
  push:
    branches:
      - main
```

```
jobs:
  sonarcloud:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: SonarCloud Scan
        uses: sonarsource/sonarcloud-github-
                                action@v1.4
        with:
          args: >
            -Dsonar.projectKey=my-project-key
            -Dsonar.organization=my-org
            -Dsonar.host.url=https://sonarcloud.io
```

(3) SonarLint setup in Java IDE:

Install SonarLint in IntelliJ IDEA or Eclipse from the plugin marketplace.

Configure it to link with your Sonar cloud account for continuous quality checks.

Once installed SonarLint will analyze your Java code locally for issues.

(4) Python Project Analysis:

① Create a sonar-project.properties file in the root of your python project.

```
sonar.projectKey = my_python_project
sonar.organization = my_org
sonar.sources = .
```

Run the analysis using SonarQube scanner:
sonar-scanner

(5) Nodejs Project Analysis:

For a Node.js project the steps are similar to Python. Configure a sonar-project.properties file and scan the project using sonar-scanner.

Example sonar-project.properties

```
sonar.projectKey = my-nodejs-project
sonar.sources = .
sonar.exclusions = node_modules/**,**/*.test.js
```

Run sonar-scanner to analyze your Node.js project.

Analyze Results:
Just like with Python, the results of the analysis will be uploaded to Sonar Cloud. The report can be accessed from the dashboard which will highlight issues such as missing semicolons, unused variables and more.

Key features:

- Sonancloud allows you to integrate with Github easily and provides a cloud based dashboard for viewing the quality of your projects.

- SonarLint helps developers fix issues in real time as they write code, making it easier to catch issues before they are committed.

- For Python and Node.js, the sonar-project.properties file is essential for configuring the analysis and the sonar-scanner tool helps run the analysis locally.

3) At a large organization, your centralized operations team get many repetitive infrastructure request. 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 organization, 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.

Soln: The goal of this task is to use Terraform to create a reusable infrastructure model enabling teams to deploy and manage their own resources without involving central operations.

① Understand the self serve infrastructure Model:
In large organizations product teams often requests infrastructure resources from a central ops team. This can be repetitive and time-consuming

By using Terraform you can create reusable modules that product teams can use independently to deploy their own infrastructure based on organization standards.

(2) Creating Terraform Modules:

Modules allow you to reuse Terraform code. Create a module that defines a common infrastructure component, such as an EC2 instance.

Teams can use this module in Terraform configurations:

```
module "ec2" {
  source = "./ec2-instance"
  ami-id = "ami-0abcd1234"
  instance_name = "team-app-instance"
}
```

(3) Automating with Terraform Cloud and Ticketing Systems:

Terraform Cloud allows you to collaborate on infrastructure deployments. It can be integrated with a ticketing system like Service Now to automate infrastructure requests.

This integration ensures that infrastructure is deployed in compliance with the organization security and governance standards.

Key Tools to Use:

① Serverless Framework: for deploying REST API's

② SonarQube/SonarCloud: for code quality analysis.

③ SonarLint in IntelliJ/Eclipse for on-the-fly Java Analysis.

④ Terraform: for infrastructure automation and self-service infrastructure.