ASSIGNMENT - 2

1] Create REST API with the serverless framework.

Steps to create a REST API with serverless frame-work:

1] Install serverless Framework globally using the following command on the terminal:-

npm install -g serverless.

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

2] Create a new service with AWS Node.js's template:

Serverless create - template aws- node.js --path rest-ap

This command initialises a new serverless ser called rest-api. It creates a folder containing basic files and a template specifically configure for building serverless applications using Node.js on AWS lambda.

3] 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 servi

4] Initialize Node.js's project and install dependencies:-

npm init -y

npm install express serverless-http

The express dependency builds the REST API and serverless-http integrate Express with AWS lambda.

5] Edit the serverless cmd file to include

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

This configuration specifies the service name, provider settings and defines the Lambda function with HTTP event triggers.

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

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

This creates a simple express app with a single route /helloworld and exports it in Lambda-compatible format.

7] Deploy the service:
serverless deploy
Deploys the API to AWS, setting up resources like lambda and API Gateway is generated for testing.
8] Test the deployed API:
curl https://<api-id>.execute-api.<region>.amazonaws.com/dev/helloworld
Using above command, it returns JSON msg:
{"message": "Hello world!!!"}.
9] Redeploy after updates:
serverless deploy.
After modifying the code, redeploy it to update the API with our changes.
10] 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.

1] Case study for Sonarqube:
Create your profile in Sonarqube for testing project quality.
Use Sonarcloud to analyse your Github code.
Install Sonarlint in your Java IntelliJ IDE or Eclipse IDE and analyse your Java code.
Analyse python project with Sonarqube.
Analyse Nodejs project with Sonarqube.

- Create your own profile in sonarqube:
1] Download and install sonarqube from the official website. Unzip the file & start the server by running: ./bin/windows-x86-64/StartSonar.
This launches Sonarqube locally and can be accessed at http://localhost:9000.
2] Log into Sonarqube using the default credentials (user-name: admin. password: admin). After logging in change the password.
3] Navigate to Projects tab, click on. Create and assign a project key and name and generate a project token.

- Use Sonar Cloud to analyse your Github
1] Sign up for SonarCloud from the official website using your GitHub account.
2] In SonarCloud under Projects > Create Project. Choose your GitHub repository and grant SonarCloud access to it.
3] 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
4] Use Sonar scanner to analyse the code by running: sonar-scanner.

Install Sonar Linky in your Java IntelliJ or Eclipse IDE & analyse your Java code:-

1] Install SonarLint by going onto IntelliJ or Eclipse, going to plugins/ Marketplace and searches for SonarLint.

2] In the IDE, configure SonarLint by linking it to your SonarQube or SonarCloud project to sync the rules and profiles.

3] Open a Java Project and SonarLint to analyse it. It will display issues directly in the IDE while coding. SonarLint provides real-time feedback on code quality based on SonarQube rules.

Analyse python project with SonarQube:-

1] Set up a Python code in a project & ensure that SonarQube is running locally.

2] Download and configure 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 analysis 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 | Type plugins have been installed. Plugins can be from the Marketplace tab in SonarQube.

3] Create a sonar-project.properties file in your project sonar root to include

sonar.projectKey=node-project
sonar.language=js
sonar.sources=

4] Run the analysis at the project by executing the 'sonar-scanner' command.

SonarQube will analyse the Node.js project show results on the dashboard, highlighting code quality, bugs & vulnerabilities.

Q3] At a large organisation your centralised operations team may get many repetitive infrastructure requests. You can use Terraform to build a self-service 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.

Creating a self-service infrastructure model using Terraform for a large organisation involves the following steps:

Step 1: Define Infrastructure Standards:
Establish clear standards and best practices for infrastructure deployment including naming conventions, resource types, tagging policies and security compliance. This foundation ensures consistency across the organisation.

Step 2: Create Terraform Modules:-
Develop reusable Terraform modules based on your organisation's standards. Each module defines resource and configurations, allowing teams to deploy infrastructure efficiently and consistently.

Step 3: Set up Terraform Cloud or Enterprise:-
Use Terraform Cloud or Enterprise for centralised management of configurations and state files, enabling collaboration and access control for infrastructure changes.

Step 4: Configure version control:
Integrate Terraform modules with version control (eg- GitHub). This tracks changes, facilitates collaboration and ensures proper versioning for updates & compatibility.

Step 5: Integrate with Service Now or other ticketing systems:-
Integrate with systems like service-Now or other ticketing systems.

Step 6: Provide Documentation & Training:
Create documentation & training for using Terraform modules & submitting requests. helping teams understand & follow best practices.

Step 7: Monitor & support:
Monitor the usage of the self-service module & provide ongoing support to users. Gather feedback helps identify pain points & areas for improvement, ensuring that the infrastructure remains compliant and efficient.

Step 8: Iterate & Improve:
Regularly review and update Terraform modules, documentation & policies based on feedback & changing organisational needs. Continuous iteration enhances efficiency, security, compliance.

By following above steps, organisations can empower product teams to manage their own infrastructure through a standardised Terraform approach