

Name: Aditya Nagesh Rao Rane

Class: D15C / Bath B

Roll No: 44

Assignment 2

05/05

Page No.

12

Date:

07/10/24

Q1] Create a REST API with Serverless Framework

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

```
npm install -g serverless
```

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

b) Create a new service with AWS Node.js template

```
serverless create --template aws-node.js --path rest-api
```

This command initialises a new serverless service called rest-api including basic files & a template configured w/ Node.js & AWS Lambda

c) Navigate to project directory `cd rest-api`

d) Initialize Node.js project & install all the dependencies

```
npm init -y
```

```
npm install express serverless-http
```

It builds the REST API & serverless-http integrates with AWS Lambda

© Edit the serverless.yml file:-

```
service: rest-api
```

```
provider:
```

```
  name: aws
```

```
  runtime: node.js 14.8
```

```
  stage: dev
```

```
  region: us-east-1
```

```
functions:
```

```
  app:
```

```
    handler: handler.app
```

```
  events:
```

```
    - http:
```

```
      path: /
```

```
      method: any
```

This configuration specifies service name, AWS provider settings & defines the Lambda function with HTTP event trigger

f) Edit handler.js to add express app.

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

g) Deploy the service :- `serverless deploy`

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

h) Test the deployed API :-

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

i) To remove the service → `serverless remove`

It removes all AWS resources associated with API, ensuring that there are no charges for unused services

28/10/24

Amazon API Gateway is a managed service enabling developers to define REST API or WebSocket API endpoints & connect them with back end. It handles authentication, access control, monitoring & tracing of API requests. API Gateway integrates with AWS services like Lambda, SNS, IAM & Cognito Identity Pools.

API Gateway sits between backend services & API users, routing HTTP requests to corresponding backends. It provides tools to manage API definition & end-points mapping. It generates API references from definitions & make them available as documentation. API Gateway ties together serverless functions & API definitions, enabling truly serverless web applications.

- a) AWS Lambda: run Lambda functions to generate HTTP responses
- b) AWS SNS: publish notifications when endpoints are accessed
- c) Amazon Cognito: provide authentication & authorization.

Benefits :-

- a) Simplified API management
- b) Enhanced security through authentication & authorization
- c) Improved scalability & reliability
- d) Integration with AWS services
- e) Reduced administrative burden

Drawbacks :-

a) Added latency → It can introduce additional latency, potentially impacting application performance

b) Limited fine-tuning capabilities → The performance parameter's can't be customized

① Aws Lambda → Create Function with
Runtime = Node.js 18x

② Add a Trigger → Trigger Configuration ≡ API Gateway
In Intent → Select "REST API" & Security as
"Open".

③ Name the api as "adityaxoxoxox".

Q2] ① Create your own profile in SonarQube for Testing :-

a) Install SonarQube locally using Docker

```
docker run -d --name sonarqube -e
SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p
9000:9000 sonarqube:latest
```

b) Run sonarqube by `http://localhost:9000`. Login with username & password

c) Create a New Profile via Quality Profiles section. After creating a project, run sonarqube scanner which will analyze it

```
sonar-scanner \
-Dsonar.projectKey=<key> \
-Dsonar.sources=. \
-Dsonar.host.url=http://localhost:9000
-Dsonar.login=<token>
```

② Analyze your code via SonarCloud :-

1) SonarCloud is a cloud-based version of SonarQube. After creating your account, go to My Projects & select Analyze new Project.

2) Choose the github repository. The code will be analyzed, & results will be displayed on SonarCloud dashboard.

3) We can get feedback on code quality, vulnerabilities & code smells directly on SonarCloud interface.

③ Install SonarLint in IntelliJ & analyze Java Code :-

1) SonarLint is an IDE plugin that helps analyze code quality issues directly as we code.

2) It begins with Installation of SonarLint plugin. Once configured, you can bind it to a specific SonarQube. SonarLint will automatically start analyzing the code as we type.

d) Analyze a Python Project

- ① Ensure that SonarQube is running.
- ② Add SonarPython plugin in SonarQube in Manage Jenkins → Configure
- ③ Configure sonar-project.properties

```
sonar.projectKey=mypythonproject  
sonar.sources=
```

```
sonar.language=py
```

```
sonar.host.url=http://localhost:9000
```

```
sonar.login=<your-token>
```

- ④ At last run your pipeline & build it to check its console out.

e) Analyze a Node.js Project.

a) Build a pipeline in Jenkins.

b) Configure your sonar-project.properties

```
sonar.projectKey=my_nodejs_project  
sonar.sources=
```

```
sonar.language=js
```

```
sonar.host.url=http://localhost:9000
```

```
sonar.login=<your-token>
```


Q3] In organizations, managing repetitive infrastructure requests can strain centralized operations teams, slowing down process. Adopting a self-serve infrastructure model using Terraform decentralizes this responsibility, empowering product team to manage their own infrastructure.

Terraform, a leading Infrastructure as a Code (IaC) tool, enables organizations to automate & manage infrastructure using declarative configuration files which reduces manual errors, improves operational efficiency & making it scalable. Central to a self-serve model are reusable, version-controlled Terraform modules, which allow to standardize infrastructure deployments. Eg:- A Kubernetes module may include RBAC which can be parameterized to allow customization without sacrificing compliance.

Terraform Cloud offers secure, centralized state management, preventing from overwriting each other's changes. Sentinel, Terraform Cloud's policy-as-code framework, enables organization to enforce governance policies & integrate them into CI/CD pipelines. Terraform Cloud's Integration with ticketing systems like ServiceNow streamlines infrastructure provisioning by automating workflows & approvals, reducing manual intervention while maintaining compliance. It also supports automated workflows like GitLab cost estimation features allow teams to forecast expenses, while multi-cloud & hybrid-cloud support ensures scalability across diverse

environments. Security is embedded through IAM policies, state encryption & integrations with secrets management tools, ensuring adherence to frameworks like SOC 2 & HIPAA.

Terraform's drift detection ensures that infrastructure remains aligned with desired state, automatically identifying & correcting any manual changes. Implementing a self-serve infrastructure model with Terraform enables large organizations to manage their infrastructure with greater autonomy, efficiency & control. By leveraging Terraform, teams can deploy infrastructure that adheres to organization's security with operational bottlenecks, thus making Terraform an ideal tool for modern infrastructure management in complex organizations.

Terraform's uses a graph based planning algorithm to optimize infrastructure deployments, minimizing dependencies & conflicts. Terraform's filter resources capabilities allow for targeted resource management, enabling selective deployment & management. Also Terraform's state dependency locking mechanism ensures consistent provider versions across deployments.

Case Study : Airbnb

Airbnb uses Terraform to manage their complex infrastructure across multiple cloud providers, including AWS, GCP & Azure.

Terraform Features Used →

- a) Modules = To standardize infrastructure configs
- b) Workspace = Managing different environments (eg:- dev, staging, prod)
- c) State Management = to track infrastructure changes
- d) PoC = for fine-grained policy management
- e) Terraform Enterprise = Collaboration, Security & governance.

