Assignment No. 2

05/05

Q.1. Create a REST API with the Serverless framework.

=>

The serverless framework is an open-source framework that simplifies the development, deployment, and management of serverless applications. It allows developers to build applications using cloud services without managing server infrastructure.

Steps to deploy REST API server on serverless framework

1. Install serverless node package using npm install -g serverless.
2. Create a serverless.yml configuration file with appropriate settings to define parameters required by REST API
3. Create a new file in the same directory as handler.js with the REST API code that server would execute and return output on calling it.
4. In the same directory where the configuration file was created, initialize the serverless using command serverless.
5. Upon initializing serverless, we would be asked to Sign In. Sign In if you have account or Sign Up.
6. After successfully, registration we have to either select creation of New App or selection of new App
7. Now we require serverless to connect to our AWS account. Create a new AWS IAM Role for serverless so that Serverless can manage deployment on our AWS account.

Note :- Ensure you have created credentials file where App-Id and secret access key is stored before proceeding

(8) Now our Serverless service is ready to deploy, use serverless deploy.
(9) After successful deployment, we will get a URL of serverless REST Api Endpoint.
(10) Visit the URL given to access system information of server.

Q2. Case study for Sonarqube.

a) Create your own profile in Sonarqube for testing of project quality.

=>

(1) Create a local project in Sonarqube dashboard.
(2) Create custom quality profile with custom quality checks.
(3) Select language and name of quality profile.
(4) Activate security checks according to your project.
(5) Assign created quality profile to project by going to the projects section and select the project we just created. Then go to project settings situated at right top corner and select quality profiles.
(6) Open jenkins dashboard and create new item.
(7) Name the project and select item type as pipeline.
(8) Go to pipeline script section and paste the following code save it. Here we are analyzing the python project named as requests, a simple HTTP library.
(9) Click on build and see the output in console.
(10) After completion of build open Sonarqube to check the status of project whether it passed our quality profiles or not.

b) Use Sonarcloud to analyze your GitHub code

->

(1) sing up on Sonar cloud (https://sonarcloud.io/login)
(2) Create a new project in Sonar cloud
(3) fork the repo of your choice into your github account.
(4) click on "Import an organization from GitHub")
(5) Give access permission to repository you want to analyse
(6) A summary will be shown, choose free plan and click on "create organization".
(7) select the repositories under created organization and set up them.
(8) select analysis method. after which analysis will be started. Wait for sometime until analysis gets completed
(9) We can see any issue in detail by clicking on it

c) Install sonarlint in your Java Intellij ide or eclipse ide and analyse your Java code.

->
(1) Download and install Eclipse IDE and open It
(2) Go to the Help menu and select Eclipse Marketplace
(3) In the eclipse marketplace, search for Sonarlint
(4) In the search results, find SonarLint and click the Install button. Restart Eclipse IDE after installing Sonarlint
(5) Create a new Java project.
(6) Name your project and finish setting up
(7) Create a new sample java file (e.g. Calculator.java) in the newly created project.
(8) Run manual analysis: you can trigger a manual code analysis by right-clicking on the project, folder or file in the Project Explorer and selecting SonarLint > Analyse

(9) See results of analysis in bottom tab
(10) Correct error if got any. The name and source of error is given in results after analysis

D) Analyze python project with sonarqube
=> (1) Clone the project repository of your ~~code~~ choice (e.g. https://github.com/pallets/flask.git)
(2) Create the sonar-project.properties file in the root of the flask project directory
(3) Before running the analysis, install the required dependencies of the flask project. The requirement-dev.txt file contains development dependencies, including testing tools.

```
pip install -r requirements/dev.txt
```

(4) Now we have to create a global analysis token, to do so, go to My Account then go to security tab
(5) Name your token and select type as Global Analysis Token and click on generate token
(6) A new token will be generated, copy it in clipboard
(7) Now we would use sonar scanner CLI to analyse our project using below command.

```
sonar-scanner \
-Dsonar.projectKey=flask-project \
-Dsonar.sources=. \
-Dsonar.host=http://localhost:9000 \
-Dsonar.login=<Token>
```

Note:- Replace -Dsonar.login with your own generated token.
(8) open the project dashboard using provided URL in output OR Look in projects section of Sonarqube dashboard

(9) In the project dashboard we can see all issues in our python code

E) Analyze nodejs project with sonarqube

=> (1) clone the Node.js project you want to analyze. for this example, let's use Express repository.

> git clone https://github.com/expressjs/express.git

> cd express

(2) In the root directory of your Node.js project, create a sonar-project.properties file to configure the analysis.

(3) Use the previously generated globally Analysis token to authenticate SonarScanner.

(4) Now we would use Sonar Scanner CLI to analyze our project using below command

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

(5) Go to project dashboard via link in the output or directly open SonarQube dashboard to see issues in our Nodejs project.

Q.3. At a large organisation, 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 organization, allowing teams to efficiently deploy services in compliance with your organization's practices. Terraform cloud can also integrate with ticketing systems like Service Now to automatically generate new infrastructure requests.

=>

(1) Create a parent directory terraform_modules, inside it create S3_bucket directory
(2) Create 4 files named main.tf, variables.tf, output.tf and terraform.tfvars with respective content in it.
(3) After Creating above files, Now initialize git repository in S3-bucket directory
(4) > git init
> git add .
> git commit -m "Add S3 bucket Terraform config"
Create empty repository on Github and push the changes to it.
> git remote add origin <REPO-LINK>
> git branch -M main
> git push -u origin main
(5) Create a Terraform cloud Account on app.terraform.io
(6) Now, Create organization by giving name and email.
(7) In your organization, Create a new workspace that will be linked to your Terraform code repository.
(8) Authorize Terraform cloud to access your Github account
(9) Install Terraform cloud as Github App so it can see and manage changes in github repository.
(10) Select repository from which terraform config files will be looked from.

(11) Review final settings and click on Create.
(12) Give tag name and bucketname which would be created in your AWS account and click on save variables and then click on start new plan.
(13) ensure bucket name is Globally unique
(14) Give run name and select run type.
(15) Generate Access keys for AWS account.
(16) Go to users, then either create new user or select existing user and copy App-Id and secret Access key
(17) Now go to settings -> Variable sets and click on create variable set
(18) Name the variable set and set scope to apply globally
(19) Select variable category as Environment variable. Enter key name as AWS_ACCESS_KEY_ID and value as your secret key which you have get from AWS account and mark it is as sensitive
(20) Similarly add AWS_DEFAULT_REGION and AWS_SECRETACCESS_key
(21) Go to workspaces on Terraform cloud and select S3-bucket-terraform and click on New Run situated at top right corner.
(22) click on confirm and apply the plan. Add comment to confirm applying plan
(23) After terraform cloud finishes applying plan, we can go to our AWS account and check if the Bucket was created or not