05

Assignment no: 1

create a REST API with the serverless framework.

1) install nodejs from official website, then verify its installation by node -v and npm -v.

2) install the serverless framework globally.
npm install -g serverless.

3) 1) go to AWS account. then go to IAM service.

4) click on users on left sidebar

5) click on add user

6) enter a username here I used serverless-user.

7) select programmatic access to create access key for user.

8) click on next: permissions.

9) assign AWSlambdaFullAccess, AmazonAPIGatewayAdministrator alternatively you can also create a custom policy. for IAM access.

10) click Next: Tags, then Next: review and finally create user.

11) download access key and secret key and save this credentials. Then on command prompt, write "aws configure" and given this access, secret key.

5) create a new serverless service.

1) create a new service

serverless create -template aws-nodejs --path my-rest-api
cd rest-api or serverless then select your app.

2) install dependensies - npm init -y
npm install

6) update serverless.yml.

service: my-rest-api

provider:

name: aws

```
  runtime: nodejs20.x
functions:
  getUsers:
    handler: handler.getUsers
    events:
      - http:
          path: users
          method: get
  getUsers:
    handler: handler.getuser
    events:
      - http:
          path: users/{id}
          method: get
  createUser:
    handler: handler.createUser
    events:
      - http:
          path: users
          method: post
  updateUser:
    handler: handler.updateUser
    events:
      - http:
          path: users/{id}
          method: put
  deleteuser:
    handler: handler.deleteuser
    events:
```

- http:
    path: users/{id}
    method: delete.

Here, my-rest-api is name of your service. Here provider specifies AWS as the provider and the node.js runtime. Then I defined five lambda functions for handling CRUD operations on users.

1) Create handler functions. I have created handler.js file

```
const users = [{id:1, name: 'John Doe'},
               {id:2, name: 'Jane Doe'}];

module.exports.getUsers = async (event) => {
  return {
    statusCode: 200,
    body: JSON.stringify(users),
  };
};

module.exports.getUser = async (event) => {
  const userId = parseInt(event.pathParameters.id);
  const user = users.find(u => u.id === userId);
  if (user) {
    return {
      statusCode: 200,
      body: JSON.stringify(user),
    };
  }
  return {
    statusCode: 404,
    body: JSON.stringify({message: 'User not found'}),
  };
};
```

```
module.exports.createUser = async (event) => {
  const newUser = JSON.parse(event.body);
  newUser.id = users.length + 1;
  users.push(newUser);
  return {
    statusCode: 201,
    body: JSON.stringify(newUser),
  };
};
module.exports.updateUser = async(event) => {
  const userId = parseInt(event.pathParameters.id);
  const userIndex = users.findIndex(u => u.id === userId);
  if (userIndex > -1) {
    const updatedUser = JSON.parse(event.body);
    users[userIndex].name = updatedUser.name;
    return {
      statusCode: 200,
      body: JSON.stringify(users[userIndex]),
    };
  }
  return {
    statusCode: 404,
    body: JSON.stringify({message: 'User not found'})
  };
};
module.exports.deleteUser = async(event) => {
  const userId = parseInt(event.pathParameters.id);
  const userIndex = users.findIndex(u => u.id === userId);
```

```
if (userIndex > -1) {
  users.splice(userIndex, 1);
  return {
    statusCode: 204,
    body: null,
  };
}
return {
  statusCode: 404,
  body: JSON.stringify({ message: 'User not found' }),
};
};
```

8) Deploy your api by command serverless deploy. abv, then deploy by using serverless deploy --debug command to get detailed info.

9) then get info by using serverless info command which will give URI (by) in the format "https://bzprjxthfl.execute-api.us-east-1.amazonaws.com/dev".

10) then perform all get, past commands to perform CRUD operations.

11) To get info:

Invoke-WebRequest -Uri "https://bzprjxthfl.execute-api.us-east-1.amazonaws.com/dev/users" - Method GET

→ status code : 200

status description : OK

content : [{"id":1,"name":"John Doe"}, {"id":2,"name": "Jane Doe"}]

RawContent : HTTP/1.1 200 OK

connection: keep-alive
X-Amzn-Trace-Id: Root=1-66fcF2ab-1@1a214
00fa46dad9ed6; Parent=695b0630597c7e
Sampled=0; Lineage=1,880d3121:0
x-amzn-RequestId: 5d14c90e-84e9-4044-9
Forms: {}
Headers: {[Connection, keep-alive] [X-Amzn-Trace-Id,
Root=1-66ffcF2ab-1b1e1a214@fo0fa46dad9ed6]}
Images: {}
Inputfields: {}
Links: {}
ParsedHtml: *golang.org/x/net/html.HTMLDocument
RawContentLength: 55

Q 2. case study for sonarqube.

1) create ur own profile in sonarqube for testing project

2) use sonarqube to analyze your Github code.

3) Install sonarlint in ur Java Intelli or eclipse and analyze the code.

4) Analyze python project with sonarqube

5) Analyze nodejs project with sonarqube.

Sonarqube is an opensource platform used for continuous inspection of code quality. It helps developers by detecting bugs, code smells and security vulnerabilities, thus ensuring maintainable and reliable code. sonarqube integrates with popular CI/CD pipelines and supports multiple programming languages.

The purpose of this case study is to demonstrate how sonarqube can be implemented in a software development lifecycle to improve code quality, maintain security, and facilitate continous integration.

step1: create a sonarqube profile for Testing project Quality

1) download and install sonarqube from official website for docker container

2) access sonarqube through http://localhost:9000 by running the docker container.

3) login to sonarqube by username "admin" and password "admin".

4) then click on your profile. then click on Quality profiles from top menu.

5) click on create to setup a new profile of your project

6) choose the language (eg. Java, Node, Python)

7) customize profile by adding or removing rules.

8) then click on creating a local project and then go to your account and create a token for your project

9) In your project create, sonar-project.properties, which will

have key, name, hosting url and login of your project.
10) then run analysis by running "sonar-scanner".

step 2: Analyze Github code using SonarCloud.
1) go to your github account.
2) then create a new github project or fork an existing repository to analyze.
3) then create a file named sonar-scanner.properties file and write sonar.projectkey = "project_key"
sonar.projectName = "project_name"
sonar.host.url = http://localhost:9000
sonar.login = your_token
all these information.
4) then start your jenkins at localhost:8080.
5) then create creat a new item ie a freestyle project
6) then go to manage plugins and install sonar scanner.
6) After installation, go to configuration system and give the SonarQube URL and authorization-token.
8) Then again go back to your project. go to configure section.
9) under build environment, select SonarQube scanner environme
10) In build section, add an Execute SonarQube scanner build step.
11) Ensure the sonar-project.properties file is in your project and provide the project key, login token and othe required details
12) then set up source code management (SCM) configuration to fecth your code from github repository.
13) the run jenkins job by building. After jenkins build is successful, then go to sonarqube. there you will get complete ana

step 4: Install sonarlint in Intellij or Eclispe and analyze Java code.

1) 1) for Intellij : open file → settings → Plugins, search for sonarlint and click install.

2) for Eclipse: go to Help → Eclipse Marketplace, search for sonarlint and click Install.

2) Navigate to settings → Tools → sonarlint.

3) open your java project, and sonarlint will automatically analyze the code in real-time.

4) fix any issues such as bugs, vulnerabilities or code smells highlighted by sonarlint.

steps: Analyze a project Python with sonarqube

1) create a python project. then create a new file in your project named sonar-project.properties.

2) sonar.language = py
   sonar.sources = src
   sonar.python.version = 3.x

3) update this in the "sonar-project.properties" file.

4) then go to sonarqube at localhost:9000.

5) then make sure that python in enabled in your sonarqube Add the sonarqube plugin if it's not enabled.

6) the navigate to your project and run sonar-scanner.

7) view analysis report on the sonarqube dashboard.

step 6: Analyze Node.js project with sonarqube.

1) Create a Nodejs project

2) then create sonarqube properties file in this project by name "sonar-project.properties"

3) inside this file write the following information:

sonar.projectKey = node-project

sonar.sources = src

sonar.language = js

sonar.host.url = https://localhost:9000

sonar.login = your_token.

4) ensure sonarqube supports Javascript/Node.js by enabling the sonarJS plugin.

5) install the sonarscanner globally using:

npm install -g sonarqube-scanner.

6) In the project root directory, run sonarqube-scanner.

7) Then go to sonarqube this will give review ie detailed report on the sonarqube dashboard.

3. At a large organizations, your centralized operations team may get many repetitive infrastructure requests. You can use Terraform to build a "self-serve" 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.

→ 1) Terraform enables the use of Infrastructure as a code (IaC), where infrastructure is defined and provisioned through declarative code. This allows teams to automate the process of setting up, scaling and maintaining infrastructure across different environments.

2) By creating reusable terraform modules, organizations can ensure infrastructure is deployed according to predefined standards, reducing manual errors and maintaining consistensy.

3) self-serve infrastructure allows individual product teams to autonomously manage their infrastructure by using pre-defined Terraform modules. This gives them flexibility in deploying and managing infrastructure. 4) modules in Terraform are reusable building blocks consist of EC2 instances, VPCs, load balancers, etc.

5) Terraform cloud can automate the provisioning and management of infrastructure. It integrates with ticketing systems like ServiceNow, where teams can submit requests for infrastructure changes or new environments.

Steps:

1) create an Ec2 instance of t2.micro of name terrafor on ubuntu OS and also create a new key and use it.
2) then go to IAM user.
3) then click on user on left sidebar.
4) there create a IAM user. then assign a policy name "AmazonEC2FullAccess". to that user.
5) then click on next and create the user.
6) Then open a powershell then go to the directory where key is stored. Here I go to downloads where m key named "serverkey.pem" stored.
7) then connect to instance by following command: "ssh -i "serverkey.pem" ubuntu@ec2-98-83-119-211.com -1.amazonaws.com".
8) then update the package by "sudo apt update".
9) then install terraform on Ec2 instance by sudo apt install -y unzip wget.
10) wget https://releases.hashicorp.com/terraform/1.5.7/terr -1.5.7_linux_amd64.zip
11) then unzip the folder by "unzip terraform_1.5.7_linux_a zip".
12) then move to terraform folder :"sudo mv terraform /u /local/bin/"
13) After the installation, verify its version and installatio by "terraform -- version". it given output Terraform v1.5. on linux_amd64.
14) Now, create terraform_project directory by "mkdir terraform_project". then go inside this directory by "cd terraform_project".

15) inside this terraform-project create a file named "main.tf". Here main.tf defines the infrastructure resource. ie Here we have used Ec2 instance.

16) then to make changes in that file write command "nano main.tf"

17) provider "aws" {
    region = "us-east-1"
    access_key = "AKIAU MTWLD6RYBNVNDTZ"
    secret_key = "3qJLor2lbPIuuPalgOTPW6Lj6lEiVmCWRAEmOGoN"
}
resource "aws_instance" "example" {
    ami = "ami-0866a3c8686eaeeba"
    instance_type = "t2.micro"
    tags = {
        Name = "TerraformExample"
    }
}

write these in main.tf and then save that file.

18) then initialize terraform by "terraform init" command.

19) then before applying terraform check whether any changes required or not by using "terraform plan".

20) After that, apply by using "terraform apply".

Thus By following these steps, a large organization can successfully implement a self-serve infrastructure model using terraform, providing autonomy to product teams while ensuring compliance with organizational standards. It provides efficiency, standardization, compliance, scalability and automation.