05

Create a REST API with the serverless Framework.

To create a REST API with the serverless Framework in theory, here's an overview of the process:

1) Serverless Framework Setup

The serverless framework is used to build serverless applications and easily manage cloud resources, such as AWS Lambda. First, you set up the framework on your machine. The framework automates the process of deploying your functions to the cloud.

2) Configure the API: The core configuration happens in the serverless.yml file. This file is used to define the service provider (like AWS), runtime environment (e.g., Node.js) and the API's functions. Each function corresponds to a specific endpoint and HTTP method (GET, POST, etc) For example, you might define one function that handles retrieves data with a get request and another that creates data with a POST request.

3) Event driven functions: Serverless uses an event-driven model where each function is triggered by an event. For a REST API, the events are HTTP requests.

4) Deployment: With a simple serverless deploy command, your API is deployed to the cloud. The serverless Framework automatically sets up API Gateway and AWS Lambda, scaling resources based on demand.

5) Testing: After deployment the framework provides.

you with the live URL of the API endpoints. You test these end-points using tools like POSTMAN curl, confirming that each HTTP request triggers the correct Lambda function.

In summary, the serverless framework allows to build, deploy, and manage REST API's efficiently abstracting away the complexity of managing se and cloud infrastructure.

Q.2) Case study for Sonarqube:

- Create ur own profile in sonarqube for testing project
- Use Sonarqube to analyze your Github code.
- Install sonarlint in ur Java intellij ide or eclipse and analyze Java code.
- Analyze python project with sonarqube
- Analyze node js project with project

→ In this case study, the objective is to improve cod quality using Sonarqube, Sonarcloud and Sonarlint f different programming languages and environments

First, Sonarqube is installed on a local machine or server to create a custom quality profile. This pro defines specific rules and quality gates for testing project quality. Developers can customize metrics like code duplication and complexity to match their proj needs. Once configured, the project code is analy by running the Sonarqube scanner, which generates detailed reports on the quality dashboard.

Next, Sonarcloud is used for Github integration. By linking Github repositories to SonarCloud, automatic analysis is triggered on every push to the repository. This integration helps track code quality over time, with insights available directly in the Sonarcloud dashboard

In IDE environments, Sonarlint is installed to provide real-time feedback while coding. It helps Java developers detect issues, bugs, or code smells as they write code, ensuring immediate fixes.

Lastly, SonarQube can be used to analyze python and Node.js projects by configuring the sonar-project.properties file and using predefined or customized rules. This provides an in-depth analysis of code quality and helps maintain high standards for various programming quality.

5 At a large organization, your centralized operations team may get many repetitive infrastructure requests. You can use Terraform to build a "self-serve" infrastructure model that lets products 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.

→ Terraform "self-serve" Infrastructure model

① Terraform modules for self-serve infrastruct

• Create Terraform modules that codify the standards for deploying common resources lik VPC's, EC2 instances, and S3 buckets.

Example module for an EC2 instance:

ec2-module/main.tf:

```
variable "instance-type" {
    default = "t2.micro"
}

resource "aws_instance" "example" {
   ami = "ami-12345678"
   instance.type = var instance_type
   tags = {
           Name = "example-instance"
         }
  }
```

ec2-module/outputs.tf:

```
output "instance_id" {
   value = aws_instance.example.id
  }
```

Teams can now use this module to deploy EC2 instances with.

```
module "ec2" {
    source = "./ec-2-module"
    instance-type = t2 medium"
  }
```

② Terraform cloud integration with service Now

- You can integrate Terraform Cloud with service Now, to automate the infrastructure request process

- Using Terraform's API-driven approach, Service Now can trigger Terraform runs based on ticket approval, automating resource deployment.

Example workflow

① A product team submits a request in serviceNow for new infrastructure.

② The request triggers a Terraform cloud updates the ServiceNow ticket with the status and resource details.

③ Creating Terraform modules for teams. Define reusable modules for commonly repeated resources like

① Networking (VPC, subnets)
② Compute (EC2, autoscaling groups).

③ Storage (S3, RPS)
④ IAM Roles / Policies.

By doing this, teams can manage their own in-ture while maintaining compliance with organizational standards