Name: Shivprasad Hande Class DLOC Roll no 14

Advance Devops Assignment 2

Page No.
Date

04

Create a REST API with serverless framework.

creating a REST API with the serverless Framework involves several steps.

1) Set up your Environment
- Install Nodejs : make sure you have Node.js install on your machine

- Install the serverless Framework
npm install -g serverless.

2) Create a New Service
- Create a new serverless service.
serverless create --template
aws-nodejs --path my-service
cd my-service.

3) Configure:
- Edit the serverless.yml file to define your API and functions. Here's simple example

4) Create the function:
- In handler.js implement your function

```
module.exports.hello = async (event)
=> { return {
        statusCode: 200,
        body: JSON.stringify
        { message: 'hello'
    }
}
}
```

5) Deploy your service

↳ Deploy your service on AWS
→ serverless deploy

This command packages your application and deploys you to AWS Lambda.

6) Test your API:

After deployment, you'll see an endpoint URL in the output. You can test it using curl your browser.

Ex → curl
https://your-api-id.../dev/hello.

7) monitor and manage

you can use Serverless Dashboard for logs and managing your deployments.

Page No.
Date

Case study - Implementing SonarQube for code quality analysis and testing

Overview

The case study explores the implementation of SonarQube in a software development project to enhance the code quality, maintainability and testing practices. The project involves a web application developed using Java & Springboot.

Objectives

1. Improve code quality
2. Ensures code coverage
3. Streamline development process.

Implementation steps

1. Setting up SonarQube
   - Installation: Installed SonarQube on a dedicated server, using Docker for easy management
   - Configuration: Configured SonarQube with the database and the backend
   - Access: Create user accounts for developers and project managers.
2. Integrating SonarQube with the project
   - SonarScanner Installation: Installed SonarScanner in the project environment.
   - Configuration File: Added a sonar-project.properties file to the project settings.

3) Running Analysis
- Initial Scan: Executed the first scan using the command
sonar-scanner.
- Analysis Results: Reviewed the dashboard, with highlighted:
Code Smells: 150 instance detected.
Bugs: 5 critical bugs.

4) Addressing Issues
- Refactoring: Developers motivated and fixed high severity bugs and vulnerabilities.
- Code Rules: Introduced mandatory code reviews for code smells before merging changes.
- Test Coverage: Increased unit test coverage to 80% by adding new tests and improving existing ones.

5) CI/CD Integration
- Pipeline Configuration: Integrated SonarQube analysis into CI/CD pipeline using Jenkins.
• Added SonarQube analysis as a build step.
• Configured build failure criteria based on quality gates (e.g. no new critical issues).

6) Continuous Monitoring and Feedback
- Regular Scans: Scheduled regular scans in deployment to monitor code quality.
- Dashboards: Utilized SonarQube dashboard to project status, making it accessible for team.

- Team training: conducted workshops to educate the team on code quality practices and bring them upto speed.

Results:
- Improved code quality
- Higher test coverage
- Faster development cycle.
- Increased team morale

In a large organization, your centralized operations team may get many infrastructure request. You can leverage Terraform modules that you can write and use Terraform modules that codify the standards for deploying and managing the service in your organization, allowing teams to deploy services in compliance with your organizational policies. Terraform cloud can also integrate with ticketing systems like ServiceNow.

To solve this problem, the goal is to design a self-service infrastructure model using terraform for a large organization to streamline infrastructure management.

1) Define the problem.
- Centralized teams open ticket requesting infrastructure services.

2) Leverage Terraform for self service infrastructure.
Terraform is a tool for automating infrastructure deployment. It allows teams to define infrastructure as code (IaC) to implement a self service model.