Shravani Patil
D15A-38

05/05

Advane Devops
Assignment no: 02

1. Create a REST API with serverless framework

→ creating a REST API with serverless framework is an efficient way to deploy serverless applications that can scale automatically without managing servers.

1. Serverless framework: A powerfull tool that simplifies deployment of serverless applications across various cloud providers such as AWS, Azure & Google cloud.

ii. serverless Architecture: This design model allows developers to build applications without worrying about underlying infrastructure, enabling focus on code & business logic.

iii. Rest API: Representational state transfer is architectural style for designing network applications.

Steps for creating REST API for serverless framework:

1. Install serverless framework:
You start by installing serverless framework CLI, globally using Node package manager (npm). This allows you to manage serverless applications directly from your terminal.

2. create a Node.js serverless project.
A directory is created for your project, where you initialise a serverless service (project). This service will host all your lambda functions, configurations & cloud resources. Using the command 'serverless create' you set up a template for AWS Node.js microservices that will eventually deploy to AWS lambda.

3. Project structure:
The project scaffold creates essential files like handler.js which contains code for lambda function.

4. create a REST API Resource
To In the serverless.yml file you define function that handles HTTP POST requests.

5. Deploy the service:
with the s/s deploy commands, serverless framework packages your applications uploads necessary resources to AWS & set up the infrastructure.

6. Testing the API:
Once deployed you can test REST API using tools like curl or postman by making POST requests to generated API.

7. Storing data in Dyanamo DB
To store submitted candidate data,
you integrate AWS Dyanamo DB as a
database.

8. Adding more functionalities like 'List all
candidates, get candidates by ID.

9. AWS IAM permissions:
You need to ensure that serverless framework
is given right permissions to interact
with AWS resources like dyanamo DB.

10. Monitoring & maintanence
After deployement serverless framework provides
service information like deployed end points
API key, log streams:

Q.2. Case Study on sonarqube.
- create your own profile in sonarqube for
testing project quality
- use sonarcloud to analyse your github code.
- Install sonarlint in your Java intellije
IDE or eclipse ide. or eclipse ide and
analyse your java code.
- Analyse python project with sonarqube.
- Analyse nodejs project with sonarqube.

→ Solution

- create the sonarqube profile for testing project quality.
- open IntelliJ setting, find tools > sonarlint entry & select to open connection wizard.
- ~~enter~~ enter a name for this connection, select sonarcloud or sonarqube.
- choose the authentication method.

a. generate token on sonarqube or sonarcloud

b. username + password: this can be used on sonarqube connection only.

- For sonarcloud only select organisation that you want to connect
- sonarqube & sonarcloud can push notification to developers.
- validate the connection creating by selecting finishing at the end of the wizard.
- save the connection in global setting by clicking OK.

Bind python project to sonarqube

- select sonarlint > Bind project to sonar cloud.
- Choose the correct project from sonarqube.

▣ Analyse the project (Python project)

- Trigger an analysis by going to code Analyse code > sonarlint

Analyse Node.js project

- Make sure your Node.js project is properly configured with sonar-project.properties file or equivalent for the analysis to run.

Q3 At large organisation your centralised operations team may get many repetitive infrastructure requests. You can use terraform modules 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 & managing services in your organisation, allowing teams to efficiently deploy services in compliance with your organisations practices. Terraform cloud can also integrate with ticketing systems like service now to automatically generate new infrastructure requests.

→ self serve infrastructure model with terraform modules:

At a large organisation, implementation a self serve infrastructure model using Terraform can significantly streamline the process of managing infrastructure across different teams. This approach allows products teams to manage their own infra-structure independently while adhering to the organisational standards & best practises.

key aspects of this self-serve model includes:

a. Standardisation through terraform modules
by creating and utilising terraform modules
large-scale organisations can codify their
infrastructure deployment & management
standards. These modules serve as reusable
packages of terraform configurations that
encapsulate common pattern & best practises.

b. Efficient deployment. product teams can
leverage these standardised modules to
quickly deploy services without needing to
reinvent the wheel or wait for the
centralised operation team to handle every
request.

c. Compliance
By using predefined modules, teams ensure
that deployments comply with the organisations
established practises & security guidelines.

d. Automation:
The use of terraform modules promotes automation,
reducing manual intervention & potential
human errors in infrastructure management.

e. version control: with modules stored in version control system like git, teams can track changes, collaborate on improvements & maintain a history of infrastructure configurations

Integration with ticketing system:
Terraform cloud offer integration capabilities that further enhance the self service model.

a Automatic Infrastructure Requests: Automatically generates new infrastructure requests. This automation streamlines the process of submitting & tracking infrastructure changes.

b Centralised Management - By centralised infrastructure management through terraform cloud organisation & maintain better control over who can suggest & approve infrastructure changes.

c Governance - The integration with ticketing systems allows for better governance of infrastructure requests, ensuring that all changes go through proper approval processes before deployment.

Collaborative Infrastructure Management:

a. Team Based performance permissons

b. state & Run history.

c. sensitive information protection

d. Module registory.

By implementing these features & practices, large organizations can effectively leverage Terraform to build a robust, scalable & compliant infrastructure management system that supports both centralised control & decentralised team autonomy.