Sneha Patra
D15A '41'

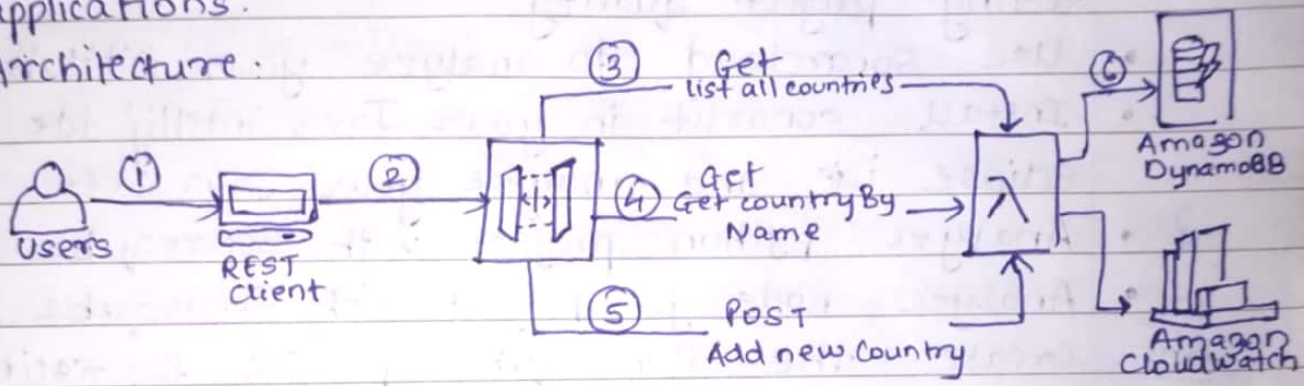
Advance DevOps

Assignment No: 2

Q1] Create a REST API with the Serverless framework.

Ans. Aws is a popular cloud provider that offers a myriad of services to support serverless architectures One of the most common use cases involves the combination of Lambda, API Gateway, and DynamoDB to build scalable, efficient, and cost-effective applications.

Architecture:



Setting Up the Environment.

- To install the Serverless framework, you need to have Node.js installed on your machine. Once you have installed the serverless framework you can create a new serverless project using a template Here we will use aws-node.js template
- The handler.js file is where we will write our Lambda function code.
- The serverless.yml file is where we will define our infrastructure. In this we will define DynamoDB table to store Data
- Lambda function will be triggered by different endpoints of our API.
- After defining the serverless.yml file and

all of the functions, we can deploy our serverless application using following commands SLS deploy
- After the deployment is complete, the serverless framework will output the URLs of API endpoint.

Q2] Case Study for Sonarqube.
- Create your own profile in sonarqube for testing project quality
- Use Sonarcloud to analyze your GitHub code
- Install sonarlit in your Java intellij ide or eclipse ide and analyze your Java code
- Analyze python project with Sonarqube
- Analyze node js project with Sonarqube.

Ans: Create the Sonarqube profile for testing project quality
- Then open Intelij setting, find Tools > Sonarlint entry and select + to open the connection wizard
- Enter a name for this connection, select SonarCloud or SonarQube.
- Choose the authentication method.

(a) Generate token on sonarqube or SonarCloud
(b) Now add username + Password: This can be used on Sonarqube connection only
- for SonarCloud only select organisation that you want to connect to.
- SonarQube and SonarCloud can push notification to developers.
- Validate the connection creating by selecting

Finishing at the end of the wizard and save the connection in global setting by clicking OK.

→ BIND Python Project to Sonarqube

- Select SonarLint > Bind project to SonarQube
- → choose the correct project from SonarQube
- → Analyze the project (Python Project)
- Trigger an analysis by going to Code > Analyze code > SonarLint.

→ Analyze Noje.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 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 product teams manage their own infrastructure independently. You can create and use Terraform modules that codify the standards for deploying and manging 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.

Sol^n At a large organization, implementing a self-serve infrastructure model using Terraform can significantly streamline the process of maniging infrastructure across different teams. This approach allows product teams to manage their own infrastructure independently while adhering to organizational standards and best practices.

Key aspects of this self-serve model include:

1.] Standardization through Terraform Modules.
By creating and utilizing Terraform modules, organizations can codify their infrastructure deployment and management standards. These modules serve as reusable packages of Terraform configurations that encapsulate common patterns and best practices.

2.] Efficient Deployment:
Product team can leverage these standardized modules to quickly deploy services without needing to reinvent the wheel or wait for the centralized operations team to handle every request.

3.] Compliance
By using predefined modules, teams ensure that their deployments comply with the organization's established practices and security guidelines.

4.] Automation.
The use of Terraform modules promotes

automation, reducing manual intervention and potential human errors in infrastructure management.

5.] Version Control.

With modules stored in version control systems like Git, teams can track changes, collaborate on improvements, and maintain a history of infrastructure configuration.

Integration with Ticketing Systems

Terraform Cloud offers integration capabilities that further enhance the self-serve model:

1.] Automatic Infrastructure Requests.

Terraform Cloud can integrate with ticketing systems like ServiceNow to automatically generate new infrastructure requests. This automation streamlines the process of submitting and tracking infrastructure changes.

2.] Centralized Management.

By centralizing infrastructure management through Terraform Cloud, organizations can maintain better control over who can request and approve infrastructure changes.

3.] 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.

As organizations grow and adopt Terraform at scale, they often move towards a collabrative infrastructure management approach:

1.] Team-Based Permissions and Team-Based Implementing
2.] State and Run History.
3.] Sensitive information protection.
4.] Module Registry.

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