HIGH LEVEL DESIGN - DOCUMENT

PROBLEM STATEMENT➖

Deploy a highly available web application on AWS using Infrastructure as Code and automate the CI/CD pipeline using AWS services.
The project should include the following tasks:

Design a highly available architecture for a web application using AWS services such as EC2, RDS, and Elastic Load Balancer in 2 regions
Use Infrastructure as Code tools such as Terraform, etc to automate the deployment process.
Implement a CI/CD pipeline using AWS services such as CodePipeline, CodeBuild, and CodeDeploy to automate the build and deployment process.
Develop and maintain documentation for the DevOps processes and procedures.

The project should demonstrate your ability to design, implement, and manage a complex AWS environment using modern tools and best practices. It should also showcase your skills in automation, cloud computing, and security.

INTRODUCTION ➖

This document presents a high-level design for a resilient and highly available web architecture deployed on Amazon Web Services (AWS). The architecture is thoughtfully designed to optimize performance, ensure scalability, and guarantee reliability. Key elements of this architecture include a Virtual Private Cloud (VPC) spanning two availability zones (ap-south-1a and ap-south-1b), emphasizing a commitment to high availability and fault tolerance.

In conjunction with the infrastructure design, a robust Continuous Integration and Continuous Deployment (CI/CD) pipeline is established to automate the software delivery process. At the core of this automation is AWS CodePipeline, a comprehensive solution that orchestrates four essential stages: Code Commit,Code Quality Analysis, Code Build, and Code Deploy. This CI/CD pipeline not only streamlines the workflow but also contributes to the efficiency, consistency, and reliability of the development and deployment lifecycle within the AWS environment. Together, these architectural components create a resilient foundation for hosting web applications while promoting automation and best practices in software delivery.

A BRIEF DESCRIPTION TO SOLUTION ➖

1. VPC Configuration ➖ The architecture begins with the creation of a VPC, strategically divided into two availability zones to mitigate the impact of failures in a specific zone. Each availability zone features a well-defined structure with a public subnet associated with a route table and a private subnet linked to another route table. This segmentation enables fine-grained control over the network traffic.

2. Database Layer ➖Within the private subnets, a MySQL RDS instance is deployed, allowing only internal VPC traffic. This design ensures that the database is accessible exclusively within the VPC, enhancing security and restricting external access.

3. Auto Scaling for High Availability ➖ To further fortify high availability, an Auto Scaling Group is established within the public subnets. This group dynamically adjusts the number of EC2 instances based on traffic and demand, distributing instances across multiple availability zones to enhance fault tolerance.

4. Load Balancing and Traffic Routing ➖ An Application Load Balancer (ALB) takes center stage, listening on port 80 to efficiently distribute incoming traffic. The ALB seamlessly redirects this traffic to the respective EC2 instances within the Auto Scaling Group, where the web application is hosted on port 8080.

5. Route53, Gateway and User Access ➖ User access is facilitated through an Route53 DNS which redirect traffic to Internet Gateway, allowing external users to connect to the VPC. Incoming traffic is then intelligently routed to the ALB, initiating the seamless flow into the web architecture.

6. Code Commit Stage ➖ The pipeline begins with the Code Commit stage, leveraging AWS CodeCommit as the source repository. Within this stage, a repository has been established to host the web application source code. Developers can seamlessly collaborate, version, and store their codebase in this secure and scalable repository.

7. Code Quality Analysis Stage ➖ SonarQube evaluates web app code for vulnerabilities, bugs, and adherence to standards. This ensures compiled code meets quality benchmarks, enhancing reliability. Insights from SonarQube guide continuous improvement, fostering a high-quality development process.

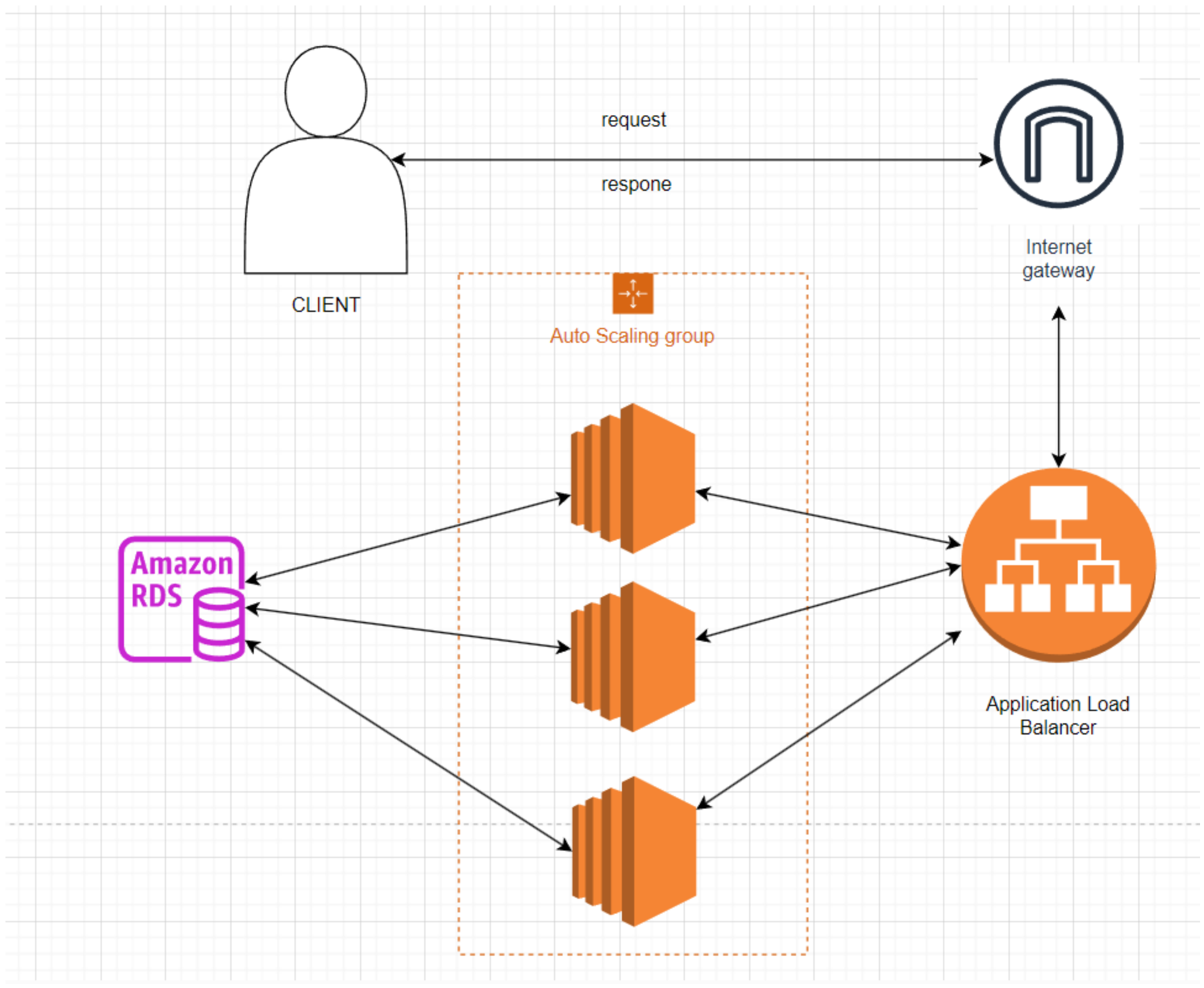
8. Code Build Stage ➖ The subsequent stage in the pipeline is the Code Build phase, where the Java application is built using Maven. CodeBuild provides a flexible and scalable environment for compiling source code, running tests, generating artifacts and store it into S3 bucket . The build process ensures that the application is ready for deployment and adheres to quality standards.

9. Code Deploy Stage ➖ The final leg of the pipeline is the Code Deploy stage, orchestrating the deployment of the artifact into the EC2 instances within the Auto Scaling Group. This stage automates the deployment process, minimizing downtime and ensuring consistency across the environment. CodeDeploy allows for controlled and phased deployments, enabling thorough testing before reaching full production.

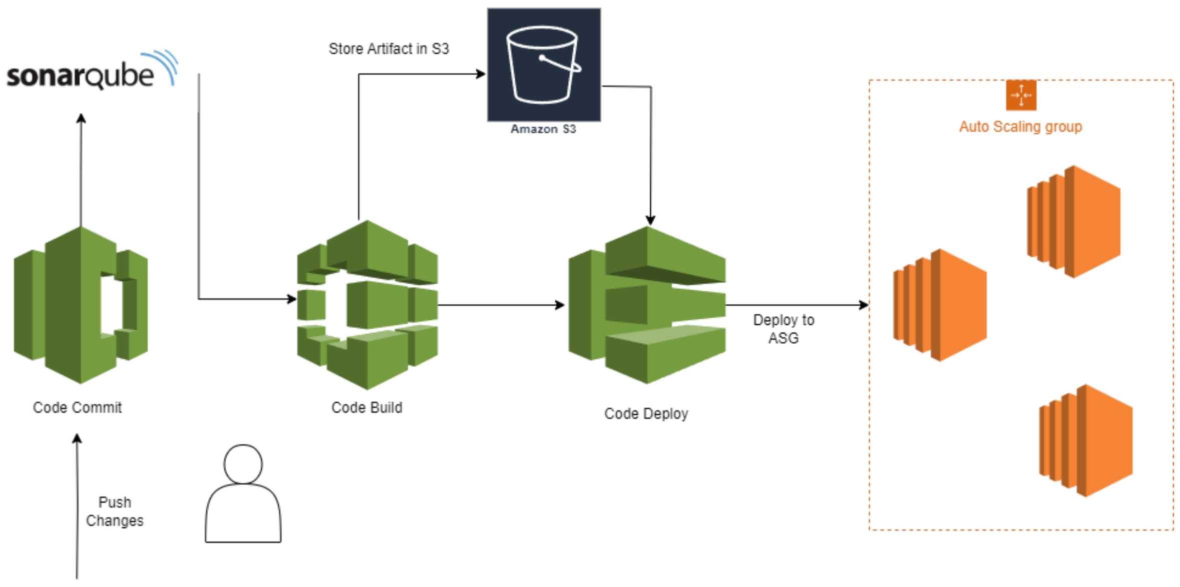
10. Containerization ➖ Containerization of web applications gives portability, isolation, less dependencies issues, and consistency. In this project docker is used to containerize the web application and then deployed into ec2 instances.

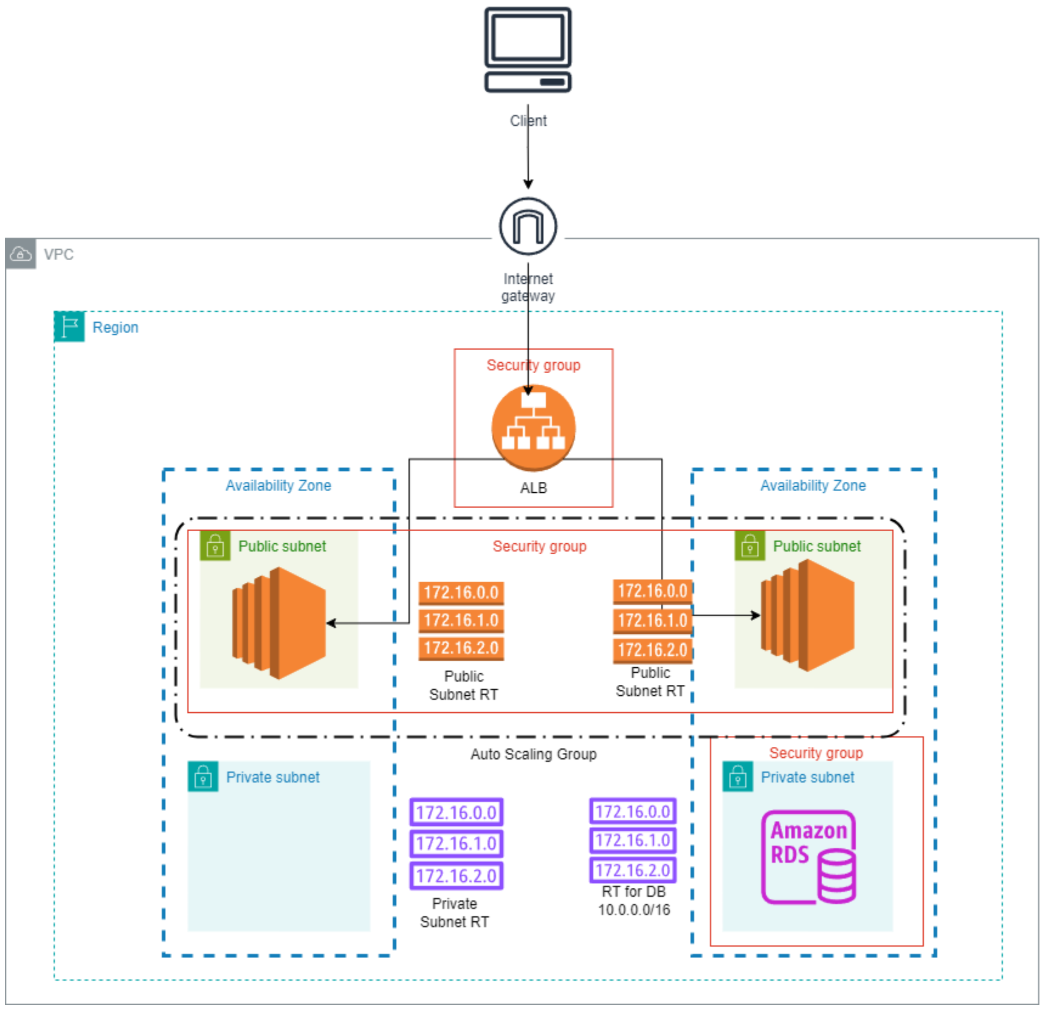
ARCHITECTURE DIAGRAMS OF VPC AND CODE-PIPELINE.

1. Data / Network Flow Diagram



2. Architecture of Code-Pipeline and VPC.


sonarqube
Store Artifact in S3
Amazon S3
Auto Scaling group
Code Commit
Code Build
Code Deploy
Deploy to ASG
Push Changes


Client
Internet gateway
VPC
Region
Security group
ALB
Availability Zone
Availability Zone
Public subnet
Security group
Public subnet
172.16.0.0
172.16.1.0
172.16.2.0
Public Subnet RT
172.16.0.0
172.16.1.0
172.16.2.0
Public Subnet RT
Auto Scaling Group
Security group
Private subnet
Private subnet
172.16.0.0
172.16.1.0
172.16.2.0
Private Subnet RT
172.16.0.0
172.16.1.0
172.16.2.0
RT for DB 10.0.0.0/16
Amazon RDS