Low-Level Design (LLD) for Microservices E-Commerce Platform

Project Overview

The goal is to create an automated DevOps pipeline for a microservices-based e-commerce application, implementing cloud-native principles. The project includes containerization, orchestration, CI/CD, secrets management, and infrastructure automation using AWS and various DevOps tools.

Tools and Technologies

- **Source Control**: GitHub
- **CI/CD**: Jenkins
- **Containerization**: Docker
- **Container Orchestration**: Kubernetes (EKS)
- **Infrastructure as Code**: Terraform
- **Configuration Management**: Ansible
- **Code Quality & Security**: SonarQube
- **Secrets Management**: Vault
- **Cloud**: AWS (VPC, EC2, RDS, S3, EKS, IAM).

Scope and Tasks

1. Infrastructure Setup (Terraform)

- **Provision a VPC**:
    - Set up subnets, NAT Gateway, and Internet Gateway in the AWS region.
- **Create EC2 instances**:
    - For Jenkins, Vault, and SonarQube to provide CI/CD and secrets management functionalities.
- **Provision EKS Cluster**:
    - Deploy an EKS cluster for Kubernetes orchestration.
    - Define worker node groups and use Auto Scaling for worker node management.
- **Create RDS PostgreSQL Instance**:

    - Provision a managed PostgreSQL instance for storing application data.
- **Set up S3 and DynamoDB**:
    - Use Amazon S3 and DynamoDB for storing Terraform state securely (use encryption, versioning, and access control).

2. Configuration (Ansible)

- **Jenkins Configuration**:
    - Install Jenkins and required plugins using Ansible.
    - Configure Jenkins for connecting with GitHub and ECR.
- **SonarQube Setup**:
    - Install and configure SonarQube for code quality and security analysis.
    - Set up a PostgreSQL database for SonarQube's backend storage.
- **Vault Configuration**:
    - Install Vault on EC2 or Kubernetes using Helm.
    - Enable the KV v2 secrets engine to manage secrets securely.
- **Jenkins Vault Integration**:
    - Ensure Jenkins can securely retrieve secrets from Vault (e.g., database credentials, API keys).

3. CI/CD Pipeline (Jenkins + GitHub)

- **Pipeline Trigger**:
    - Set up Jenkins pipeline to trigger on GitHub push to the repository.
- **Unit Tests and SonarQube Analysis**:
    - Run unit tests (using PHPUnit or another testing framework).
    - Run SonarQube static code analysis to ensure code quality and security.
- **Docker Build & Push to ECR**:
    - Build Docker images using Jenkins.
    - Push the images to Amazon ECR for container storage.
- **Secrets Retrieval from Vault**:

    - Retrieve AWS credentials and other sensitive data from Vault to authenticate with AWS services.

- **Deploy to EKS**:
    - Use Helm to deploy the services to EKS.
    - Create Kubernetes resources like Deployments, Services, and ConfigMaps using Helm charts.
    - Make sure services are deployed to the correct namespaces and RBAC rules are applied.

---

4. Kubernetes Setup

- **Helm for Deployment**:
    - Use Helm charts to deploy each microservice on Kubernetes. Each service should be in a separate Helm chart.
- **Namespace Organization**:
    - Organize Kubernetes resources into separate namespaces for isolation and scalability (e.g., dev, staging, production).
- **RBAC & Network Policies**:
    - Implement Role-Based Access Control (RBAC) to manage user permissions for Kubernetes resources.
    - Set up Network Policies to secure pod communication between services.
- **Horizontal Pod Autoscaling (HPA)**:
    - Enable HPA for microservices to scale based on CPU or memory usage.
    - Ensure that appropriate resource requests and limits are set for each microservice.

---

5. Security Implementation

- **IAM Roles with Least Privilege**:

    - Create IAM roles with the least privilege for Jenkins, EKS nodes, and EC2 instances.
    - Use IAM roles for service accounts (IRSA) to allow Kubernetes services to access AWS resources securely.
- **Secrets Management via Vault**:
    - Store sensitive information like database credentials, API keys, and other secrets in Vault.
    - Ensure no hardcoded credentials in application code or configuration files.
- **SonarQube Integration**:
    - Integrate SonarQube into the CI/CD pipeline for continuous code quality enforcement.
    - Use SonarQube for static code analysis and vulnerability detection