Corporate CI/CD DevOps Pipeline Project Report

1. Introduction

This report provides an in-depth analysis of the enterprise-grade CI/CD pipeline infrastructure deployed across multiple virtual machines, prioritizing security, scalability, and automated quality checks. The project integrates multiple DevOps tools and follows the best practices of Continuous Integration (CI) and Continuous Deployment (CD).

2. Infrastructure Layout

The infrastructure uses Infrastructure as Code (IaC) principles with tools like Terraform and Ansible to automate the deployment and management of the environment. The environment consists of the following components:

- Terraform provisions and manages the cloud infrastructure, including the virtual machines for Jenkins, Kubernetes clusters, SonarQube, and monitoring.
- Ansible is used to configure and manage these virtual machines with consistent playbooks. Key playbooks include:
  - Docker installation
  - Jenkins setup
  - Kubernetes setup for both master and worker nodes
  - SonarQube installation

Dedicated Virtual Machines:

1. Jenkins VM:

    - Acts as the pipeline orchestrator, operating on IP 172.200.209.8 with 2 CPU cores and 8 GB RAM for smooth execution under heavy loads.

2. Kubernetes Cluster:

- One master node (IP 20.106.221.225) and two worker nodes (IPs 20.106.222.59, 51.8.89.123) for scalable application workloads, each node provisioned with 4 GB RAM and 2 CPU cores.

3. Monitoring VM:

- Hosts Prometheus for metrics collection, Grafana for visualization, and Alert Manager for intelligent alert routing. It operates with 4 GB RAM and 2 CPU cores (IP 20.109.17.13).

4. SonarQube VM:

- Facilitates continuous code quality analysis and integrates seamlessly with the CI/CD pipeline. It is hosted on IP 13.91.61.42.

3. CI/CD Pipeline Workflow

The CI/CD pipeline has a detailed and automated workflow designed for continuous integration, testing, deployment, and monitoring:

1. Development Phase:

- Developers work on code locally and conduct integration tests before pushing changes to GitHub.

2. Version Control (GitHub):

- Code is versioned using GitHub with enforced branch protection rules and a pull request workflow for code reviews.

3. Jenkins Pipeline:

- Jenkins pulls the latest code from GitHub, orchestrating the build process with parallel execution for faster builds.

4. Build & Test (Maven):

- The application is compiled using Maven, which also runs both unit and integration tests. Detailed test reports are generated.

5. Security Scanning:

- The pipeline includes two key security checks:
  - Trivy Scan for detecting known vulnerabilities in dependencies.
  - SonarQube Analysis for static code analysis and security vulnerability detection.

6. Artifact Management:

- Final build artifacts are stored in Docker Hub, with secure version control maintained for Docker images.

7. Containerization (Docker):

- Docker images are built and tagged, with additional security scanning for container vulnerabilities.

8. Kubernetes Deployment:

- The application is deployed on a Kubernetes cluster using YAML files to describe the desired state, and Kubernetes automatically scales the application as needed.

9. Monitoring Setup:

- Prometheus collects metrics, Grafana provides real-time dashboards, and the system monitors incoming and outgoing traffic patterns for potential bottlenecks.

10. Benchmarking:

- Load testing is performed using ApacheBench, and response times are monitored via Grafana dashboards to assess application performance under significant traffic.

4. Monitoring and Benchmarking

The monitoring setup is divided into three sections:

1. Infrastructure Monitoring:

- Prometheus collects CPU and RAM metrics, which are visualized through Grafana dashboards.

2. Application Monitoring:

- Response times, error rates, and custom metrics are tracked to ensure the smooth operation of the application.

3. Traffic Monitoring:

- Network traffic is analyzed to identify potential performance bottlenecks.

Benchmarking:

The team uses benchmarking tools like ApacheBench to generate application load and monitor how it performs under stress. This enables real-time identification of bottlenecks or areas for optimization.

5. Team Roles

The team members played critical roles in different sections of the project:

- Amr Yasser: Responsible for Jenkins setup and SonarQube integration, ensuring efficient pipeline execution and robust code quality checks.
- Mohammed Elsayed: Managed Kubernetes deployment and monitoring setup using Prometheus and Grafana.

- Tasneem Selim: Worked on infrastructure provisioning and configuration using Terraform and Ansible.

Conclusion

This report highlights the implementation of a highly scalable and secure CI/CD pipeline, using industry-standard tools and best practices for infrastructure management, application deployment, and real-time monitoring. The pipeline ensures efficient code integration, robust testing, and secure application delivery.