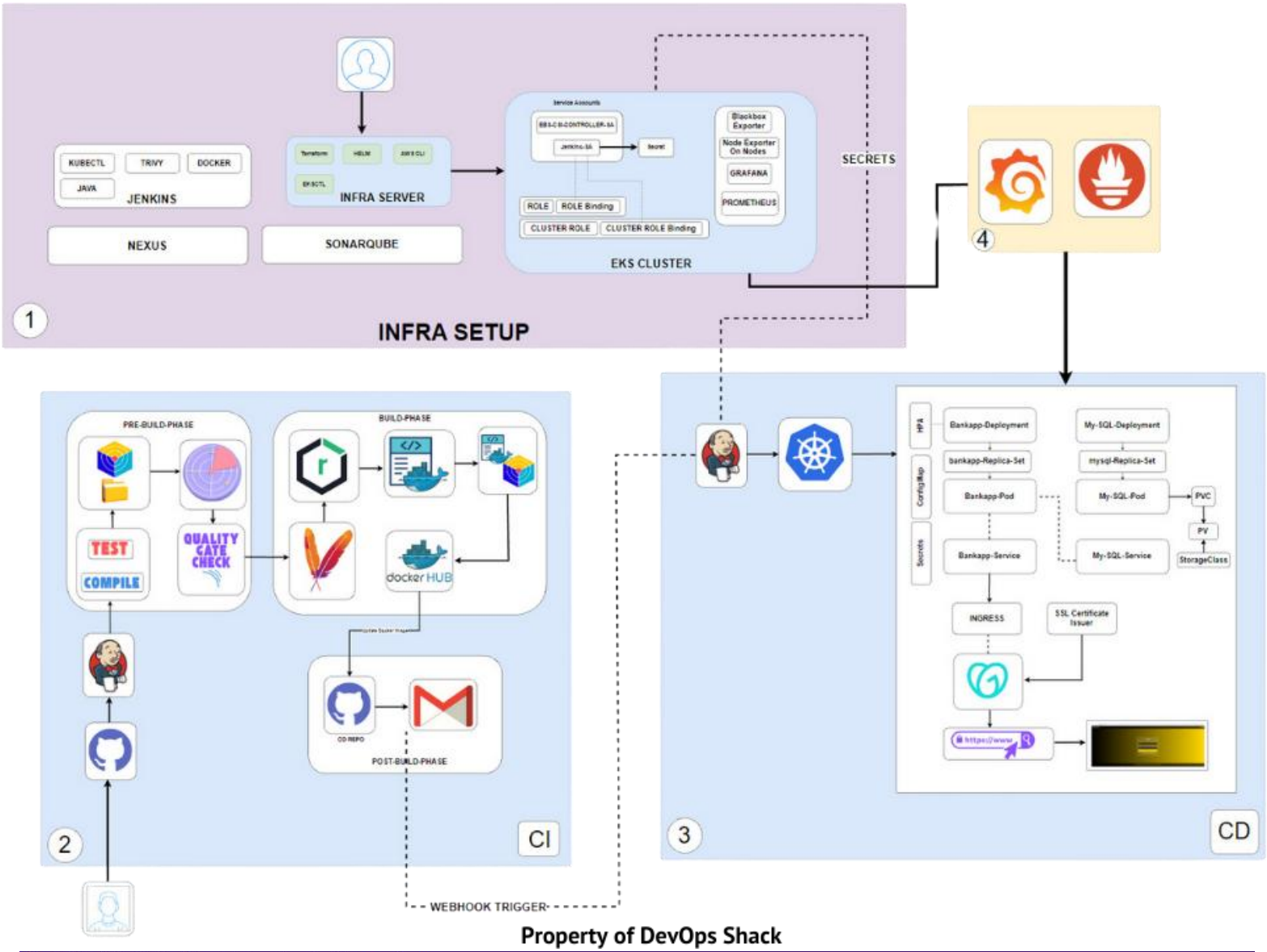
DevOps Shack

🚀 Ultimate Corporate Mega DevOps Project 🚀




```
ubuntu@ip-172-31-32-132:~$ kubectl get all -n webapps
NAME                          READY   STATUS    RESTARTS      AGE
pod/bankapp-589f97bb98-kkmbb  1/1     Running   0             5d4h
pod/bankapp-589f97bb98-qv4pj  1/1     Running   1 (5d4h ago)  5d4h
pod/mysql-6ffd887848-gwd6h    1/1     Running   0             5d4h

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/bankapp-service ClusterIP   172.20.223.171   <none>        80/TCP     5d4h
service/mysql-service   ClusterIP   172.20.102.169   <none>        3306/TCP   5d4h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/bankapp 2/2     2            2           5d4h
deployment.apps/mysql   1/1     1            1           5d4h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/bankapp-589f97bb98 2         2         2       5d4h
replicaset.apps/mysql-6ffd887848   1         1         1       5d4h

NAME                                        REFERENCE              TARGETS                                        MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/bankapp-hpa   Deployment/bankapp   cpu: <unknown>/50%, memory: <unknown>/70%   2         5         2          5d4h
ubuntu@ip-172-31-32-132:~$ kubectl get sc
NAME     PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
ebs-sc   ebs.csi.aws.com         Retain          WaitForFirstConsumer   false                  5d4h
gp2      kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false                  5d20h
ubuntu@ip-172-31-32-132:~$ kubectl get pvc -n webapps
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mysql-pvc   Bound    pvc-9b1c0f94-be63-4640-9240-bff3541269ad   5Gi        RWO            ebs-sc         <unset>                 5d4h
ubuntu@ip-172-31-32-132:~$ kubectl get ingress -n webapps
NAME              CLASS   HOSTS              ADDRESS                                                                   PORTS     AGE
bankapp-ingress   nginx   www.shackverse.co   a30cd54a38bf8421292cfb8f09f92fa0-149236318.ap-south-1.elb.amazonaws.com   80, 443   5d4h
ubuntu@ip-172-31-32-132:~$ kubectl get certificates -n webapps
NAME                READY   SECRET              AGE
shackverse-co-tls   True    shackverse-co-tls   5d4h
ubuntu@ip-172-31-32-132:~$
```

Introduction: The Ultimate Corporate Mega DevOps Project

In today's fast-paced tech world, delivering high-quality software rapidly and reliably is paramount. The **Ultimate Corporate Mega DevOps Project** demonstrates a comprehensive enterprise-level DevOps implementation, integrating modern tools and best practices to streamline the software development lifecycle. This project is designed to mimic real-world workflows, ensuring automation, scalability, and monitoring across all stages.

The project is structured into **four distinct phases**, each playing a vital role in achieving a robust and production-ready system:

1. **Phase-1: Infrastructure Setup** 🏗️
   Establishes a scalable and secure cloud infrastructure using **Amazon EKS**, with all necessary tools like **Jenkins**, **SonarQube**, **Nexus**, and monitoring solutions like **Prometheus** and **Grafana** integrated within the cluster.

2. **Phase-2: Git Repository Setup** 📂
   Implements a clear version control strategy by organizing the codebase into three repositories:
   - CI Project Repository (source code and CI pipelines).
   - CD Project Repository (Kubernetes deployment manifests).
   - Terraform Repository (infrastructure as code for EKS setup).

3. **Phase-3: CI/CD Pipelines** 🛠️
   Automates the application lifecycle with robust **CI pipelines** for building, testing, and artifact management, and **CD pipelines** for deploying applications to Kubernetes clusters.

4. **Phase-4: Monitoring & Observability** 📊
   Implements real-time monitoring using **Prometheus**, **Grafana**, and **Node Exporter** to track both infrastructure and application health, ensuring proactive issue detection and resolution.

This phased approach ensures a seamless integration of DevOps practices, enabling continuous delivery, enhanced security, and reliable monitoring. Each phase builds on the previous, culminating in a fully operational and production-grade system. Let's dive into the details of each phase! 🚀

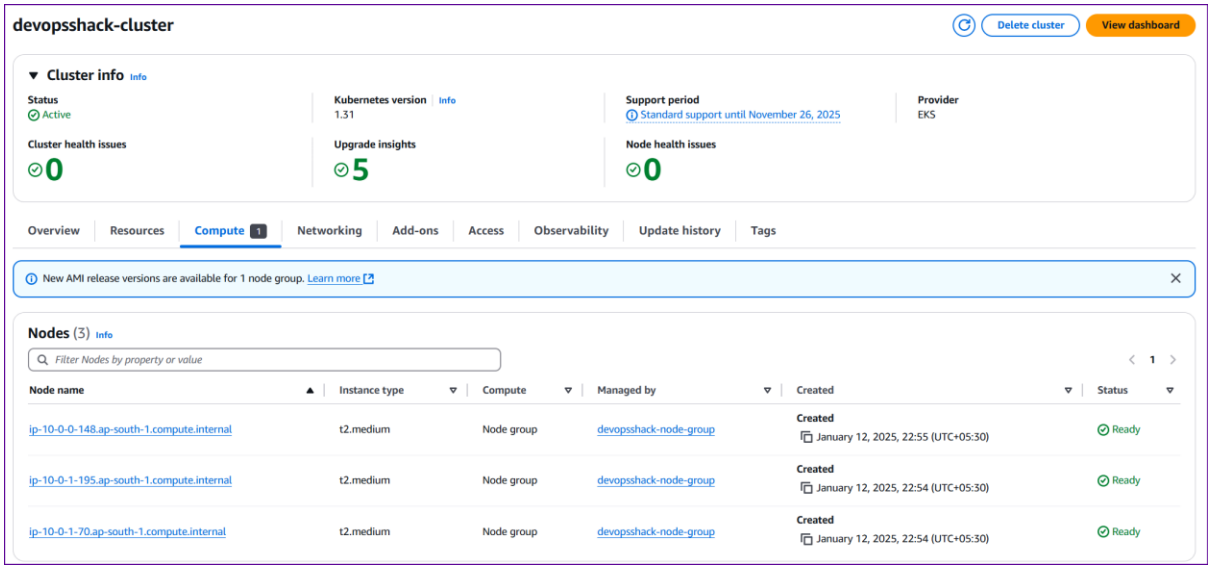
This project is designed to integrate **DevOps best practices**, including infrastructure automation, CI/CD pipelines, application monitoring, and security. Below is an in-depth explanation of the project, phase by phase.

Phase-1: Infrastructure Setup

Infrastructure setup is the foundation for this project, creating a robust, scalable, and secure environment to host applications and manage workflows effectively.

EKS Cluster Setup

Amazon Elastic Kubernetes Service (**EKS**) is used as the orchestration platform to run and manage containerized workloads.



1. Service Accounts

Service accounts are essential for enabling secure communication between applications and cluster resources:

- **Jenkins Service Account**:
  - Provides Jenkins pipelines with access to Kubernetes resources like:
    - Deployments, Services, and Pods (Namespace-scoped).
    - PersistentVolumeClaims and ConfigMaps (Cluster-scoped).
  - Configured using **Role-Based Access Control (RBAC)** for secure permissions.
  - Enables automated deployment and resource management.
- **EBS Service Account**:
  - Facilitates integration with **AWS Elastic Block Store (EBS)** for creating persistent volumes dynamically.
  - Essential for applications requiring storage (e.g., databases).

2. RBAC (Role-Based Access Control) Setup 🔐

RBAC ensures that each service or application only accesses resources it's authorized for:

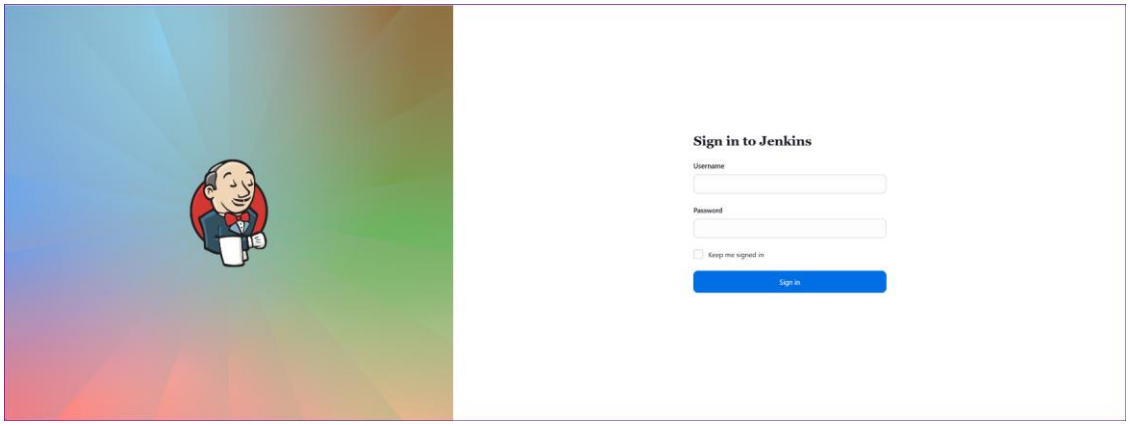
- **Namespace-Scoped Roles**:
  - Role definitions allow access to resources like pods, deployments, and secrets within a specific namespace (e.g., webapps).
- **Cluster-Scoped Roles**:
  - Roles manage cluster-wide resources, such as nodes and PersistentVolumeClaims.
  - These roles are critical for managing dynamic storage provisioning and cluster-level deployments.

Tool Installation & Configuration 🛠️

This project uses several critical tools deployed within the EKS cluster to handle CI/CD, artifact management, monitoring, and security.

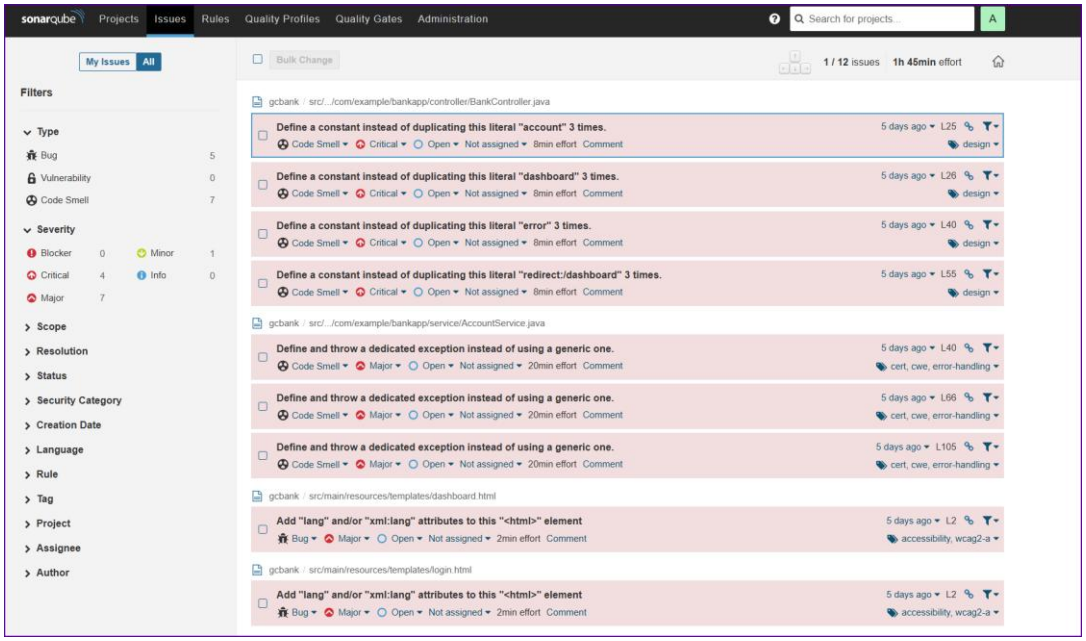
1. Jenkins:

- **Why Jenkins?**
  - Automates CI/CD pipelines.
  - Provides robust integration with Kubernetes and Docker.
- **Configuration Details**:
  - Installed with the following tools:
    - **Trivy**: Scans for vulnerabilities in file systems and Docker images.
    - **Helm**: Deploys and manages Kubernetes applications.
    - **Docker**: Builds and pushes container images to the registry.
  - Service account credentials are injected into Jenkins to enable Kubernetes interactions.



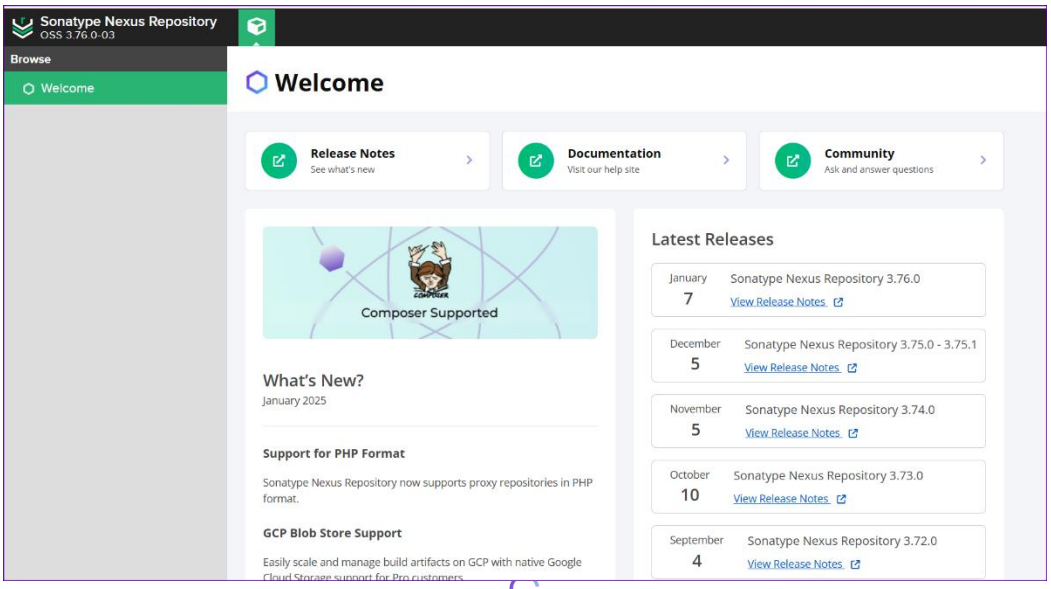
2. SonarQube:

- A leading tool for **static code analysis**:
  - Identifies code smells, bugs, and vulnerabilities.
  - Provides detailed reports for developers to improve code quality.
- Deployed in EKS with persistent storage provided by **EBS volumes**.



3. Nexus Repository Manager:

- A centralized artifact repository used to:
  - Store build artifacts (e.g., JAR/WAR files).
  - Enable version control for artifacts.
- Configured with credentials for integration with Jenkins.



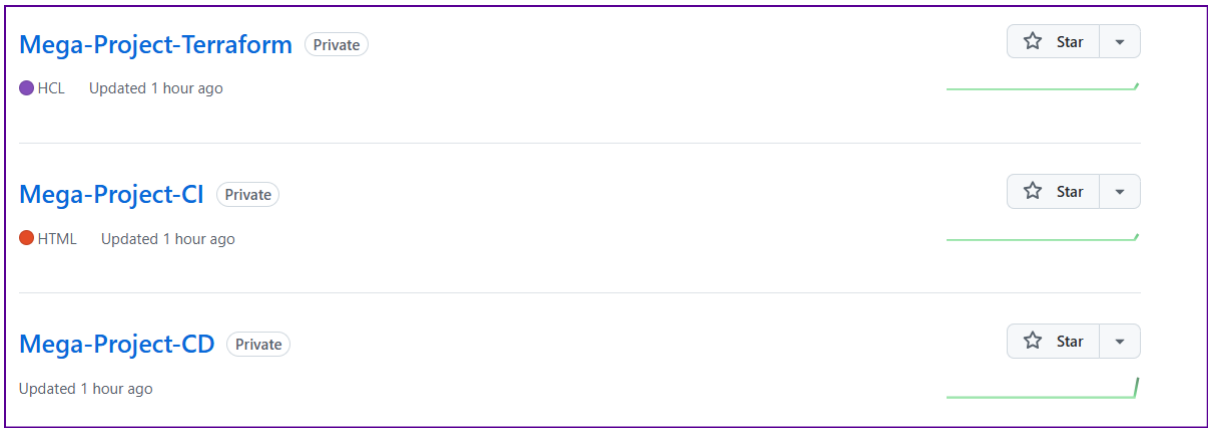
4. Monitoring Stack:

- **Prometheus**:
  - Collects time-series metrics from nodes, pods, and application endpoints.
  - Acts as the primary data source for monitoring.
- **Grafana**:
  - Provides dashboards for visualizing metrics collected by Prometheus.
  - Dashboards include:
    - Node health (CPU, memory, disk usage).
    - Application pod performance.
    - Cluster resource utilization.
- **Node Exporter**:
  - Monitors worker nodes' resource usage.
- **Kube-state-metrics**:
  - Monitors the health and state of Kubernetes objects such as:
    - Deployments, Pods, Services.

📂 Phase-2: Git Repository Setup 📂

Effective organization and version control are crucial for managing multiple aspects of this project. Three separate repositories are created:

1. **CI Project Repository**:
    - Stores:
        - Application source code.
        - Jenkins pipeline scripts for building, testing, and packaging the application.
    - Versioned for traceability of code and pipeline changes.
2. **CD Project Repository**:
    - Contains:
        - Kubernetes manifests for deploying the application.
        - Horizontal Pod Autoscaler (**HPA**) configurations.
    - Managed separately to decouple CI and deployment concerns.
3. **Terraform Repository**:
    - Automates infrastructure creation using **Infrastructure as Code (IaC)**.
    - Resources managed include:
        - **EKS Cluster**.
        - Networking components (VPC, subnets, route tables).
        - IAM roles and policies for Kubernetes integrations.



🛠️ Phase-3: CI/CD Pipelines 🛠️

This phase automates the build, test, and deployment process, ensuring rapid and reliable delivery of application changes.

CI Pipeline: Making the Application Deployment-Ready 🚦

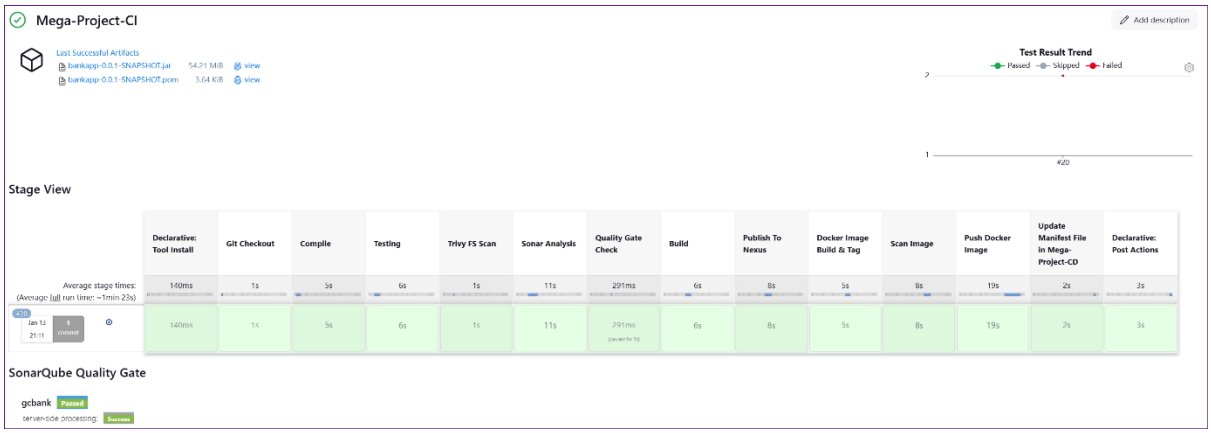
The **Continuous Integration (CI)** pipeline includes multiple automated steps to validate and prepare the application for deployment.

Pipeline Stages (Detailed):

1. **Git Checkout**:
    - Clones the source code from the CI repository (main branch).
2. **Compile**:
    - Uses **Maven** to compile the Java application.
3. **Testing**:
    - Executes unit tests using Maven's testing framework.
    - Ensures application functionality before proceeding.
4. **Trivy File System Scan**:
    - Scans the project directory for known vulnerabilities.
    - Generates a report (fs-report.html) for review.
5. **SonarQube Analysis**:
    - Analyzes code for:
        - Bugs, Code Smells, and Vulnerabilities.
    - Waits for **Quality Gate** approval to ensure high-quality code before proceeding.
6. **Build**:
    - Packages the application as a deployable artifact (e.g., JAR or WAR file).
7. **Publish to Nexus**:
    - Uploads the artifact to the Nexus repository for centralized storage.
8. **Docker Image Build & Tag**:
    - Builds a Docker image of the application.

- Tags the image using the build number for unique versioning.

9. **Image Security Scan**:
   - Scans the Docker image for vulnerabilities using Trivy.
   - Generates a report (image-report.html).
10. **Push Docker Image**:
    - Pushes the Docker image to a DockerHub repository.
11. **Update Manifest File**:
    - Updates the Kubernetes deployment manifest in the CD repository.
    - Commits and pushes the updated file to the main branch.



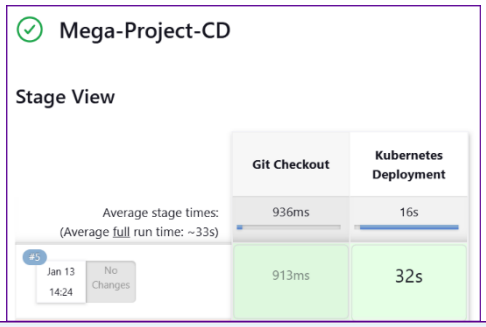
CD Pipeline: Deploying the Application 🚀

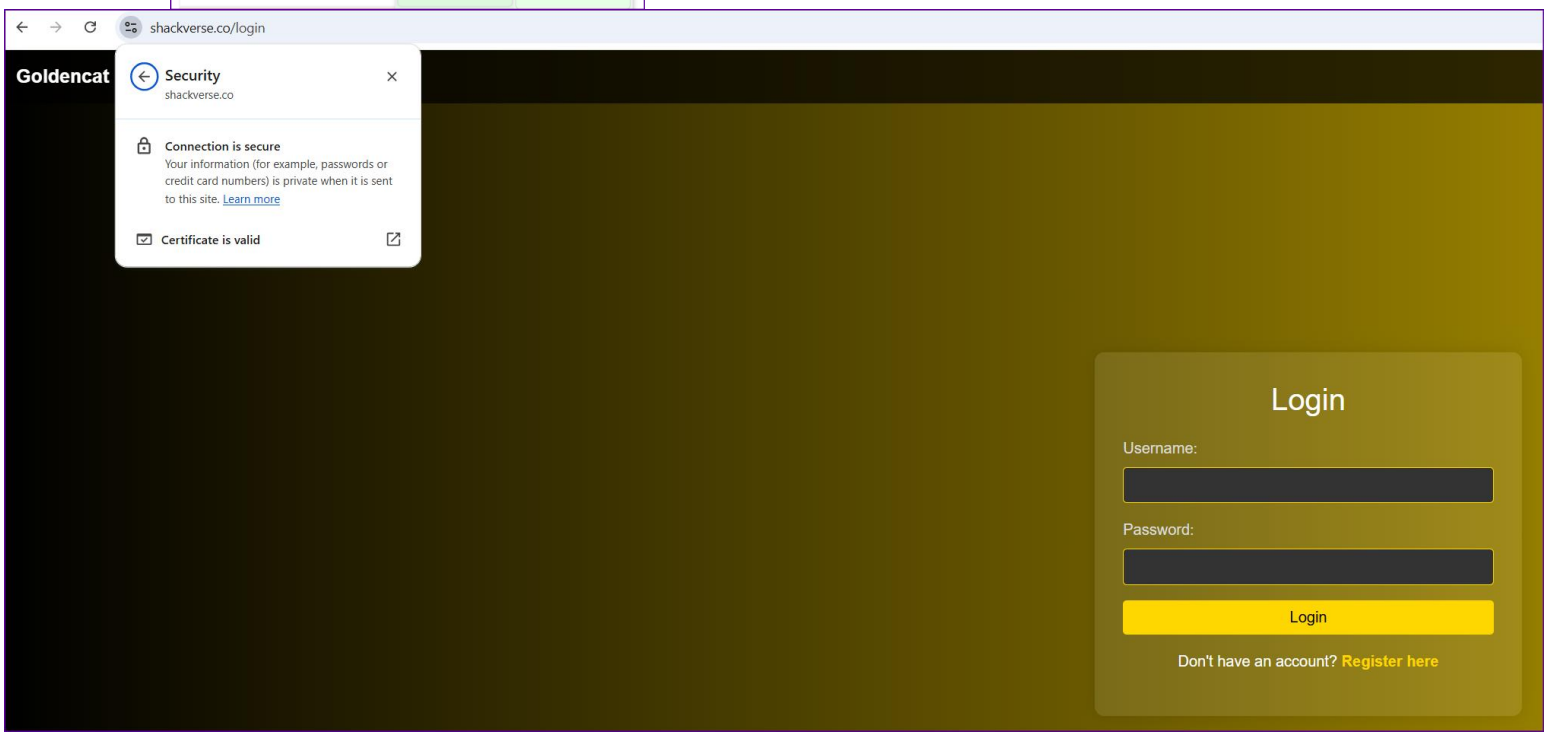
The **Continuous Deployment (CD)** pipeline automates the application deployment process.

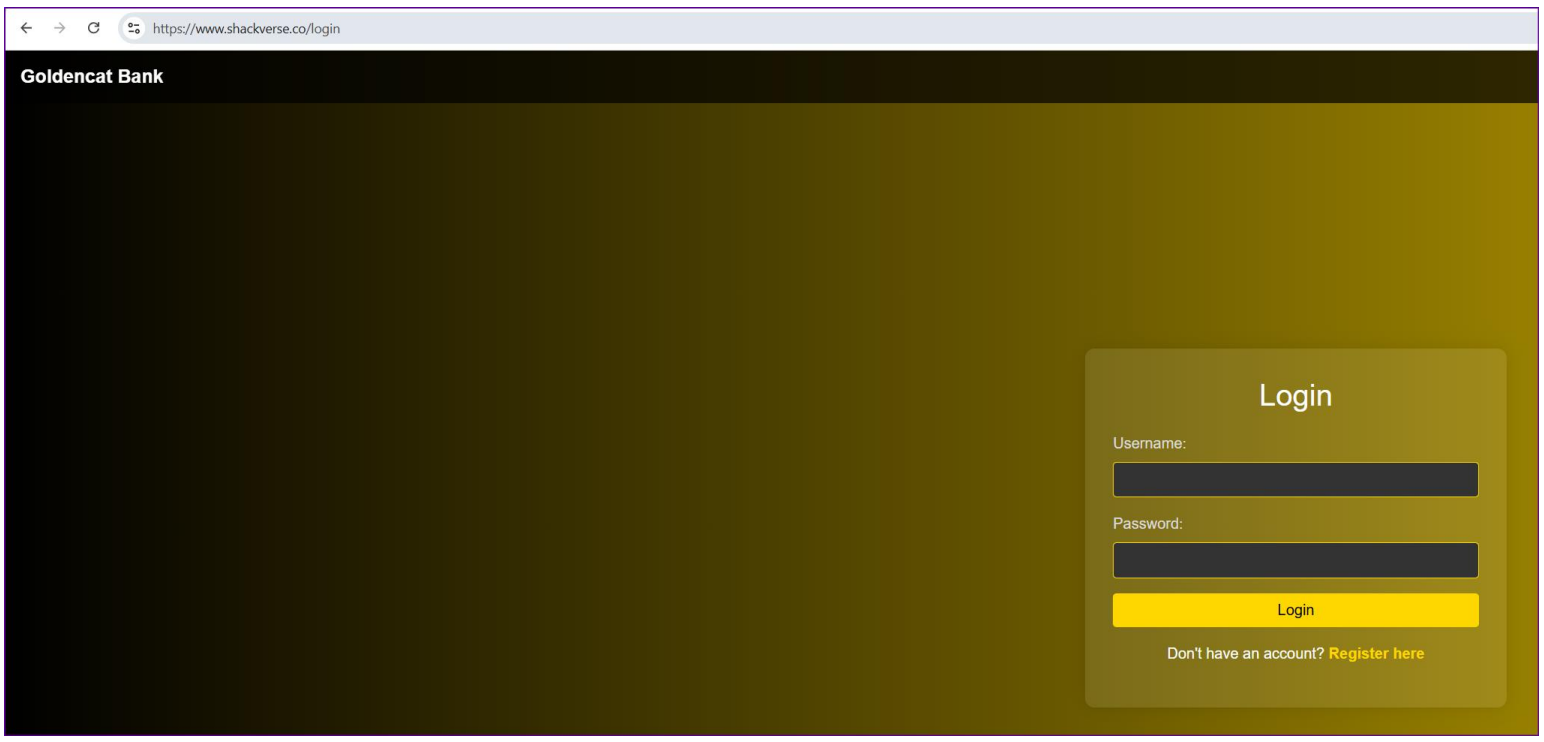
Pipeline Stages (Detailed):

1. **Git Checkout**:
   - Clones the CD repository containing deployment manifests.
2. **Kubernetes Deployment**:
   - Applies updated Kubernetes manifests (manifest.yaml).
   - Deploys the application to the **webapps** namespace.
   - Configures **Horizontal Pod Autoscaler (HPA)**:
     - Scales pods based on CPU/memory usage.
   - Validates deployment by:
     - Listing pods (kubectl get pods).

- Verifying service exposure (kubectl get service).







📊 Phase-4: Monitoring & Observability 📊

Monitoring is a critical aspect of maintaining application performance and reliability.

Worker Node Monitoring:

- **Node Exporter**:
    - Collects metrics on CPU, memory, and disk usage from Kubernetes worker nodes.
    - Sends metrics to **Prometheus** for aggregation.

Application Monitoring:

- **Kube-state-metrics**:
    - Provides detailed insights into Kubernetes objects such as:
        - Pod health, Deployment status, Resource consumption.
    - Ensures the application is running as expected.

Grafana Dashboards:

- Dashboards are created to visualize metrics, including:
    - **Node-Level Metrics**:
        - CPU, Memory, Disk Usage.
    - **Application-Level Metrics**:
        - Pod resource consumption.
        - Request/response rates.
    - **Cluster-Level Metrics**:
        - Resource allocation for namespaces and deployments.

Highlights & Benefits 🌟

1. **Security First**:
    - Integrated tools like **Trivy** and **SonarQube** ensure code and container security.
2. **Scalability**:
    - The use of **Horizontal Pod Autoscalers** allows the application to handle varying loads seamlessly.
3. **Automation**:

- CI/CD pipelines automate repetitive tasks, reducing manual errors and improving efficiency.

4. **Real-Time Monitoring**:
   - Prometheus and Grafana provide immediate insights into system health and performance.
5. **Version Control**:
   - Git repositories ensure traceability and collaboration for both code and deployment configurations.