Contents

[Assignments: 2](#_Toc139831762)

[Task 1: Helm chart to deploy a nginx server on Kubernetes using self-signed certificate. 2](#_Toc139831763)

[Task 2: Designing a Minimal High-Availability Web Server Architecture. 3](#_Toc139831764)

[Mechanism to Handle Infrastructure Failures: 3](#_Toc139831765)

[Load Balancer: 3](#_Toc139831766)

[Web Server Instances: 3](#_Toc139831767)

[Database Replication: 3](#_Toc139831768)

[Task 3: Strategy for Backup, Restore, and Point-in-Time Recovery in Relational Databases 4](#_Toc139831769)

[Example Timeline: 4](#_Toc139831770)

[Task 4: Designing a Minimal CI/CD Pipeline for a Java Spring Boot Application Code Repository 5](#_Toc139831771)

[Branching Strategy: 5](#_Toc139831772)

[Automated Triggering of Pipeline: 5](#_Toc139831773)

[CI Stage: 5](#_Toc139831774)

[Manual Approval Workflow for Deployment: 6](#_Toc139831775)

[CD Stage: 6](#_Toc139831776)

[Production Deployment: 6](#_Toc139831777)

[CICD pipeline sample diagram 6](#_Toc139831778)

[Task 5: Ansible Playbook script to install Prometheus Node Exporter 7](#_Toc139831779)

# Assignments:

# Task 1: Helm chart to deploy a nginx server on Kubernetes using self-signed certificate.

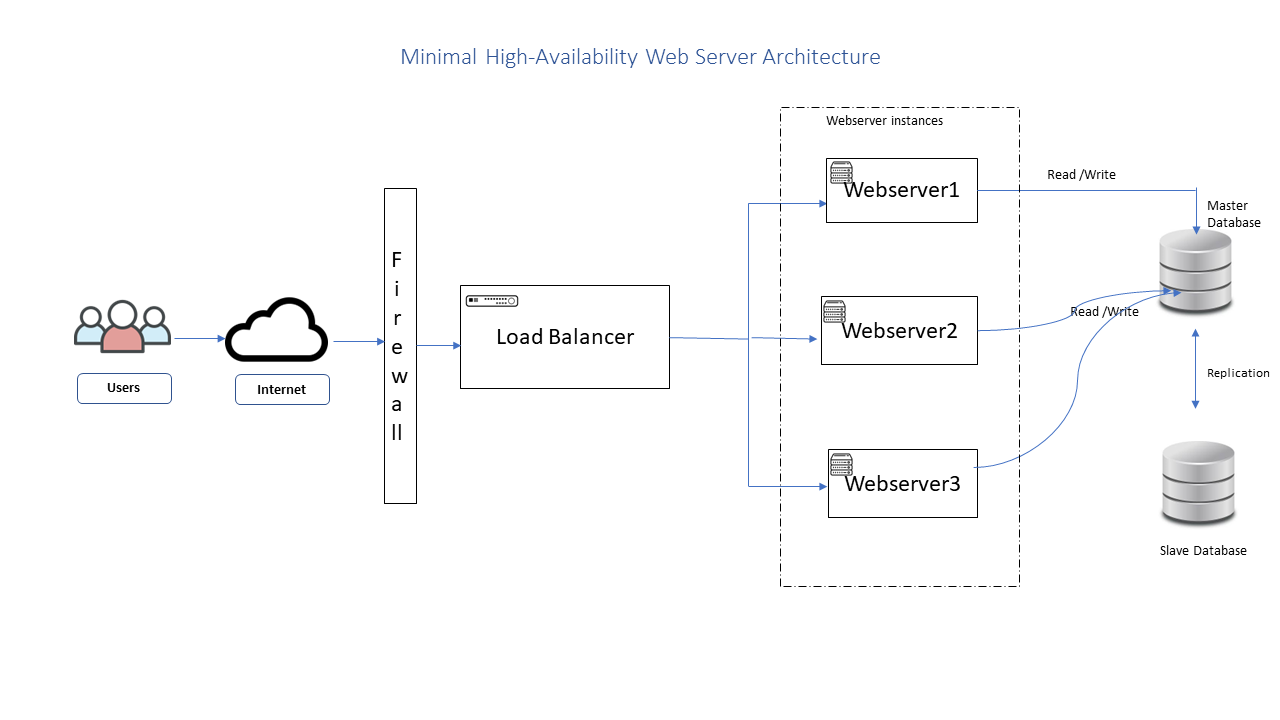
Deployed nginx server on Kubernetes using my self-signed certificate refer the sample screenshot below.

GitHub repo : <https://github.com/r7rajkumar/nginx-chart>

A screenshot of a computer

Description automatically generated

# Task 2: Designing a Minimal High-Availability Web Server Architecture.



## Mechanism to Handle Infrastructure Failures:

Load Balancer:

The Load Balancer continues to monitor the health of the Web Server instances and redirects traffic only to the healthy instances. It isolates any failed or unresponsive instances, ensuring uninterrupted service availability.

Web Server Instances:

Each Web Server instance operates independently, and the Load Balancer manages the traffic distribution. If a Web Server instance becomes unhealthy or fails, the Load Balancer stops routing traffic to that instance. Additional instances can be launched automatically to replace the failed ones, maintaining the desired level of high availability.

Database Replication:

The primary database instance replicates data to the replica instance(s) in real-time or with a certain replication lag. If the primary database fails, one of the replica instances can be promoted to become the new primary, ensuring continuous data availability, and minimizing downtime.

# Task 3: Strategy for Backup, Restore, and Point-in-Time Recovery in Relational Databases

* Determine the Recovery Point Objective (RPO) and Recovery Time Objective (RTO): These define how much data loss and downtime is acceptable in the event of a failure. For point-in-time recovery, we need to decide the granularity of recovery points (e.g., every hour, every 15 minutes).
* Set up regular full backups: Perform periodic full backups of the database. The frequency of full backups depends on the RPO and the size of your database. For example, if RPO is one hour, we may schedule full backups every 24 hours.
* Configure transaction log backups: The transaction log records all changes made to the database. To achieve point-in-time recovery, you need to back up the transaction log at regular intervals. The frequency of transaction log backups depends on the acceptable data loss defined by the RPO. For example, if your RPO is one hour, you may schedule transaction log backups every 15 minutes.
* Store backups offsite: It's crucial to store your backups in a separate location from your primary database server. This ensures that backups are protected in case of physical damage or loss at the primary site.
* Perform incremental backups: To optimize backup size and duration, consider using incremental backups. Incremental backups capture only the changes made since the last full or incremental backup, reducing the backup window.

## Example Timeline:

Let's assume the following timeline to illustrate the backup and restore process:

Day 1, 12:00 PM: Perform a full backup of the database.

Day 2, 12:00 AM: Perform an incremental backup, capturing changes since the last full backup.

Day 2, 3:00 AM: Perform a transaction log backup, capturing changes since the last transaction log backup.

Day 2, 6:00 AM: Perform another transaction log backup

Day 2, 9:00 AM: Database failure occurs.

To restore the database to a point in time before the failure (e.g., Day 2, 8:00 AM), follow these steps:

Restore the latest full backup from Day 1, 12:00 PM.

Apply the incremental backup from Day 2, 12:00 AM.

Apply the transaction log backup from Day 2, 3:00 AM.

Apply the transaction log backup from Day 2, 6:00 AM.

By following these steps, we can restore the database to the desired point in time, allowing you to recover the data as it existed before the failure occurred.

# Task 4: Designing a Minimal CI/CD Pipeline for a Java Spring Boot Application Code Repository

## Branching Strategy:

Main Branch: The main branch, such as "master" or "main," represents the production-ready code. Only stable and tested code should be merged into this branch.

Feature Branches: Developers create feature branches for new features or bug fixes. These branches are created from the main branch and merged back into the main branch after code review and testing.

### Automated Triggering of Pipeline:

Whenever changes are pushed to the feature branches or the main branch, a webhook or a Git trigger in Jenkins detects the changes and triggers the pipeline.

Jenkins fetches the source code from the Git repository.

### CI Stage:

Compile and Build: Jenkins runs the build process using Maven or Gradle by executing the "pom.xml" file to compile and build the Java Spring Boot application.

Run Tests: Automated tests, including unit tests and integration tests, are executed to ensure code quality and functionality.

### Manual Approval Workflow for Deployment:

If the CI stage is successful, the pipeline moves to the deployment stage.

Deployment to Kubernetes is not triggered automatically but requires manual approval.

Jenkins sends a notification or deploys a dedicated environment for the application, such as a staging environment.

The responsible person or team reviews the staging environment, performs any necessary manual tests, and approves the deployment.

### CD Stage:

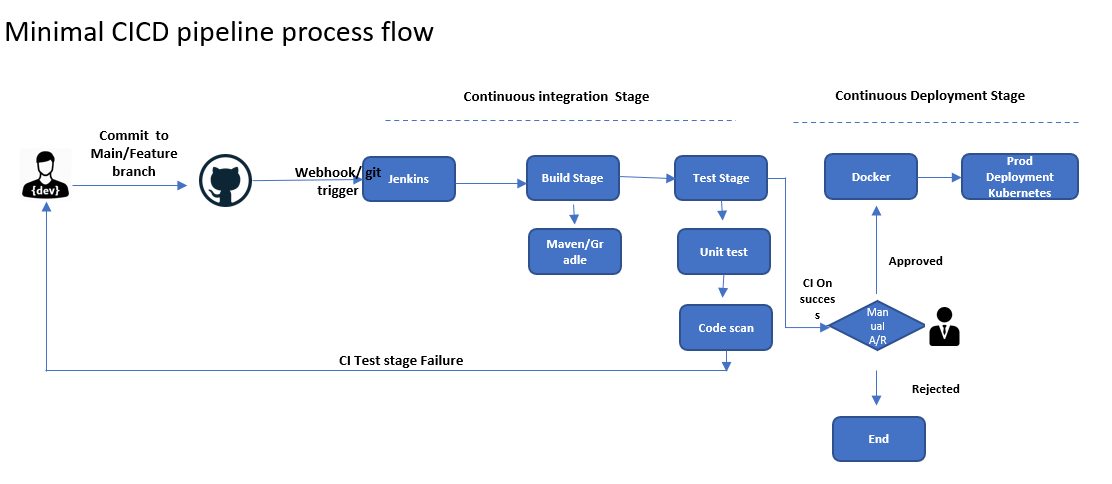
Dockerization: Jenkins builds a Docker image using the provided "Docker file" to containerize the Java Spring Boot application.

Helm Deployment: Jenkins deploys the Docker image to Kubernetes using Helm. The "helm/" directory contains the necessary Helm charts for deploying the application.

### Production Deployment:

After manual approval, Jenkins triggers the deployment of the Docker image to the production environment on Kubernetes using Helm.

### CICD pipeline sample diagram

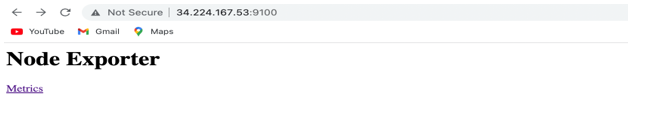


Note: The provided diagram represents a minimal version of the CI/CD process flow. Depending on specific requirements, each stage and approval can be included to create a more comprehensive and tailored CI/CD process diagram.

# Task 5: Ansible Playbook script to install Prometheus Node Exporter

Deployed Prometheus node exporter successfully using ansible playbook. Please refer the sample screenshot below.

GitHub Repo : <https://github.com/r7rajkumar/ansible-prometheus-node-exporter-installation>



A screenshot of a computer

Description automatically generated