Project Report on

# End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring


**Submitted by**

**Apurva Lagad (240844223004)**
**Srushti Bhosale (240844223007)**
**Rohini Khandare (240844223040)**

Under the guidance of

**Mr. Sandeep Walvekar**


**In partial fulfillment of the award of Post Graduate Diploma in**

**IT Infrastructure, Systems and Security**

**(PG-DITISS)**



**Sunbeam Institute of Information Technology,**

**Pune (Maharashtra)**


**PG-DITISS -2024**

# DECLARATION

We declare that this written submission represents our ideas in our own words and where others ideas or words have been included; we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed**.**

Place: Pune

Date: 10.02.2025


| **Apurva Lagad** | **Srushti Bhosale** | **Rohini Khandare** |
|---|---|---|
| **(240844223004)** | **(240844223007)** | **(240844223040)** |

# CERTIFICATE

This is to certify that the project report entitled **"End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring,"** submitted by **Apurva Lagad** is the bonafide work completed under our supervision and guidance in partial fulfillment for the award of Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date: 10.02.2025

**Mr. Sandeep Walvekar**

Guide

**Mr. Vishal Salunkhe**

Course Coordinator

**Mr. Nitin Kudale**

CEO

Sunbeam Institute of Information Technology

Pune (M.S.) – 411057

# CERTIFICATE

This is to certify that the project report entitled **"End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring,"** submitted by **Srushti Bhosale** is the bonafide work completed under our supervision and guidance in partial fulfillment for the award of Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date: 10.02.2025

**Mr. Sandeep Walvekar**

Guide

**Mr. Vishal Salunkhe**

Course Coordinator

**Mr. Nitin Kudale**

CEO

Sunbeam Institute of Information Technology

Pune (M.S.) – 411057

# CERTIFICATE

This is to certify that the project report entitled **"End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring,"** submitted by **Rohini Khandare** is the bonafide work completed under our supervision and guidance in partial fulfillment for the award of Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date: 10.02.2025

**Mr. Sandeep Walvekar**

Guide

**Mr. Vishal Salunkhe**

Course Coordinator

**Mr. Nitin Kudale**

CEO

Sunbeam Institute of Information Technology

Pune (M.S.) – 411057

# APPROVAL CERTIFICATE

This Project II report entitled **"End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring"** by **Apurva Lagad (240844223004)** is approved for Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date: 10.02.2025

Examiner: _____

**(Signature)**

_____

**(Name)**

# APPROVAL CERTIFICATE

This Project II report entitled **"End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring"** by **Srushti Bhosale (240844223007)** is approved for Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date: 10.02.2025

Examiner: _____

**(Signature)**

_____

**(Name)**

# APPROVAL CERTIFICATE

This Project II report entitled **"End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring"** by **Rohini Khandare (240844223040)** is approved for Post Graduate Diploma in IT Infrastructure, Systems and Security (PG-DITISS) of Sunbeam Institute of Information Technology, Pune (M.S.).

Place: Pune

Date: 10.02.2025

Examiner: _____

**(Signature)**

_____

**(Name)**

# CONTENTS


| TITLE | PAGE NO |
|---|---|
| **Declaration** | |
| **Certificate** | |
| **Approval Certificate** | |
| **Abstract** | i |
| **1. INTRODUCTION** | 1 |
| 1.1 Applications | 3 |
| 1.2 Organization and Project Plan | 4 |
| **2. LITERATURE SURVEY** | 5 |
| Paper 1 | 5 |
| Paper 2 | 5 |
| Paper 3 | 5 |
| **3. SYSTEM DEVELOPMENT AND DESIGN** | 6 |
| 3.1 Proposed System | 6 |
| 3.2 Flow Chart | 7 |
| 3.3 Technology used | 8 |
| 3.3.1 Git | 8 |
| 3.3.2 Docker | 9 |
| 3.3.3 Jenkins | 10 |
| 3.3.4 Kubernetes | 11 |
| 3.3.5 Sonarqube | 12 |
| 3.3.6 Trivy | 13 |
| 3.3.7 Prometheus | 14 |
| 3.3.8 Grafana | 15 |
| **4. PROJECT OUTPUT** | 16 |
| **5. CONCLUSION** | 19 |

| | |
|---|---|
| 5.1 Conclusion | 19 |
| 5.2 Future Scope | 19 |
| **REFERENCES** | 20 |

# ABSTRACT

In modern software development, automating the Continuous Integration and Continuous Deployment (CI/CD) process is crucial for improving efficiency, security, and reliability. This project, **End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring**, aims to streamline software delivery by reducing manual intervention and enhancing security measures throughout the development lifecycle.

The CI/CD pipeline was designed and implemented using **Jenkins**, automating key stages such as building, testing, security scanning, and deployment. **Docker** was employed to containerize the application, ensuring consistency across different environments. To strengthen security, **SonarQube** was integrated for static application security testing (SAST), helping to identify vulnerabilities at an early stage. Additionally, **Trivy** was used to perform container security scanning, detecting potential threats in the application images before deployment.

The deployment process was managed using **Kubernetes**, which provides scalability, high availability, and efficient orchestration of containerized applications. To ensure comprehensive observability, **Prometheus** and **Grafana** were implemented for real-time monitoring, performance tracking, and log analysis, allowing proactive identification of issues. Furthermore, **GitHub CI/CD integration** played a key role in maintaining version control and automating workflows, ensuring a seamless development-to-deployment process.

# 1. INTRODUCTION

In today's fast-paced software development environment, automation plays a crucial role in enhancing efficiency, security, and reliability. Traditional software deployment methods are often manual, time-consuming, and prone to errors, making it difficult to ensure consistency and security across different environments. To overcome these challenges, this project, **End-to-End CI/CD Pipeline for Web Application with Docker, Jenkins and Automated Monitoring**, focuses on implementing a fully automated pipeline that streamlines development, testing, security scanning, and deployment while reducing manual intervention.

**Stages of the CI/CD Pipeline:** The project follows a structured CI/CD workflow, consisting of several key stages:

**1.Code Integration and Version Control:**

The development process begins with maintaining the source code in **GitHub**, where version control ensures efficient collaboration among developers. Developers commit changes to a shared repository, triggering the CI/CD pipeline to automate subsequent stages.

**2.Build Automation and Compilation:**

Once code is pushed to the repository, **Jenkins** automatically pulls the latest changes and initiates the build process. This involves compiling the application, resolving dependencies, and generating executable artifacts in a consistent environment. **Docker** is used to containerize the application, ensuring that it runs uniformly across development, testing, and production environments.

**3.Automated Testing:**

To ensure code quality and functionality, the pipeline includes automated testing stages. Unit tests and integration tests are executed to validate new code changes. Automated testing helps detect bugs early in the development cycle, preventing faulty releases.

**4.Static Code Analysis and Security Scanning:**

Security is a top priority in modern software development. **SonarQube** is integrated into the pipeline to perform **Static Application Security Testing (SAST)**, scanning the source code for vulnerabilities, code smells, and potential security risks. Additionally, **Trivy** is used to conduct **container security scanning**, identifying security flaws in the application's Docker images before deployment.

**5.Containerization and Image Deployment:**

After security scanning and testing, the application is packaged into a **Docker container** to maintain consistency across different environments. The container image is then pushed to a container registry, making it accessible for deployment.

**6.Deployment to Kubernetes:**

The application is deployed using **Kubernetes**, ensuring scalability, high availability, and resource efficiency. Kubernetes automates the deployment, scaling, and management of containerized applications across multiple nodes, preventing downtime and improving reliability.

**7.Monitoring and Logging:**

To maintain visibility into application performance and infrastructure health, **Prometheus** and **Grafana** are integrated for real-time monitoring and log analysis. These tools help in tracking resource usage, detecting anomalies, and identifying potential performance bottlenecks before they impact users.

**8.Continuous Deployment and Workflow Automation:**

The pipeline is designed for seamless **Continuous Deployment**, ensuring that approved changes are automatically released to production environments without manual intervention. **GitHub CI/CD integration** plays a vital role in managing version control and orchestrating automated workflows.

### 1.1 Applications

- Enterprise Software Development – Automates the deployment of enterprise applications, ensuring faster release cycles and reduced manual errors.
- DevSecOps Implementation – Integrates security scanning (SonarQube, Trivy) into the pipeline, making security an integral part of software development.
- Cloud-Native Application Deployment – Enables seamless deployment of containerized applications on Kubernetes-based cloud platforms.
- Scalable Microservices Architecture – Supports deploying and managing microservices efficiently with Kubernetes, ensuring high availability.
- Automated Monitoring and Incident Response – Provides real-time performance insights through Prometheus and Grafana, enabling proactive issue resolution.
- Version Control and Collaborative Development – Helps large teams collaborate effectively by integrating GitHub for version control and Jenkins for automation.
- Software Testing and Quality Assurance – Automates testing and code analysis, reducing the risk of deploying defective software.
- IoT and Edge Computing Applications – Supports the deployment of IoT applications by automating software updates for edge devices.
- AI/ML Model Deployment – Facilitates the continuous deployment of machine learning models by automating version control, testing, and scaling.
- Game Development and Deployment – Automates game builds, testing, and deployment to cloud-based gaming platforms.

### 1.2 Organization and Project Plan

| Sr. No. | ACTIVITY | WEEK | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| 1 | Project group formation | | | | |
| 2 | Project work to be started in respective labs | | | | |
| 3 | First review with PPT presentation | | | | |
| 4 | Design Use-Case view as per project | | | | |
| 5 | Design Block diagram as per project | | | | |
| 6 | Second review with PPT presentation | | | | |
| 7 | Selection | | | | |
| 8 | Final review with PPT presentation | | | | |
| 9 | Implementation coding as per project | | | | |
| 10 | Testing, Troubleshooting with different techniques | | | | |
| 11 | Created Soft copy of project and then final hard copy | | | | |

**Table: Activities Details**

# 2. LITERATURE SURVEY

**Paper 1:.Security for Continuous Integration and Continuous Deployment Pipeline"**

**Author:** Sachin Vighe

**Description:** This paper emphasizes the importance of integrating security measures into CI/CD pipelines to address potential vulnerabilities that can arise from rapid deployment processes. It discusses how incorporating automated security testing, enforcing security policies, and ensuring compliance with regulations can help identify and mitigate security threats early in the development lifecycle. The proactive approach outlined aims to reduce the likelihood and impact of security incidents, thereby maintaining customer trust and the integrity of software applications.

**Paper 2: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines"**

**Authors:** Thorsten Rangnau, Remco v. Buijtenen, Frank Fransen, Fatih Turkmen

**Description:** This case study explores the integration of dynamic security testing tools within CI/CD pipelines to enhance the security posture of software systems. It examines the challenges and benefits of incorporating security testing into the continuous integration and delivery processes, providing insights into how automated security measures can be effectively implemented without hindering the agility of DevOps practices.

**Paper 3: Automating Security in a Continuous Integration Pipeline"**

**Authors**: Sohrab Chalishhafshejani, Bao Khanh Pham, and Martin Gilje Jaatun

**Description:** This paper identifies solutions for automating the security phase within a continuous software delivery process. It focuses on integrating security tools into a GitHub repository by utilizing GitHub Actions to create automated vulnerability scanning workflows for software projects. The study emphasizes the importance of embedding security measures directly into the CI pipeline to detect vulnerabilities early and maintain the integrity of the software development lifecycle.

# 3. SYSTEM DEVELOPMENT AND DESIGN

## 3.1 Proposed System

We propose a system where we are setting up the Software Development Life Cycle (SDLC), ensuring a structured and efficient approach to automating the CI/CD pipeline for secure web application deployment. The development began with requirement analysis, identifying key automation and security needs. The system design phase focused on architecting a pipeline using Jenkins for automation, Docker for containerization, and Kubernetes for deployment. The implementation phase involved scripting pipeline workflows, integrating SonarQube and Trivy for security scanning, and setting up monitoring with Prometheus and Grafana. Rigorous testing was conducted, including unit, integration, and security testing, ensuring robustness. Finally, the deployment phase automated the entire CI/CD pipeline, streamlining application releases while maintaining high security and reliability.

### 3.2 Flowchart

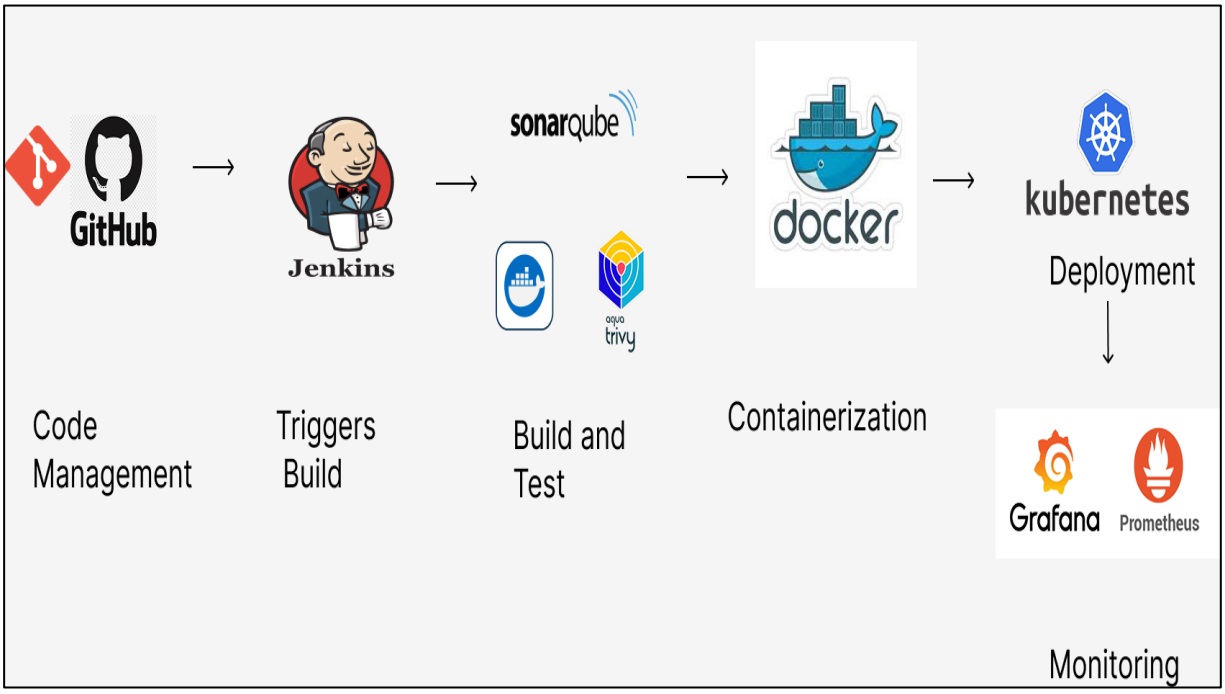


**Figure: Flowchart**

### 3.3 Technology used

#### 3.3.1 Git

Git is a distributed version control system (VCS) designed to manage source code history and facilitate collaborative software development.

**Key features of Git:**

Distributed Architecture: Unlike centralized version control systems, Git is distributed. Each developer has a complete copy of the repository, including its entire history. This allows for offline work, faster operations, and improved resilience.

Branching and Merging: Git makes it easy to create branches, which are separate lines of development. Developers can work on features, bug fixes, or experiments in their own branches without affecting the main codebase. Merging branches back together is relatively simple and allows for collaborative development.

Commit History: Git maintains a detailed history of changes to the codebase. Each change is represented by a commit, which includes information about who made the change, when it was made, and what was changed. This commit history provides a clear view of the evolution of the project.

Fast and Efficient: Git is designed for speed and efficiency. Most operations are local, as the repository resides on the developer's machine. This results in rapid commits, branching, and merging.

Collaboration: Git enables effective collaboration among developers. Multiple developers can work on different branches simultaneously, and changes can be shared by pushing them to a remote repository. Pull requests or merge requests facilitate the process of reviewing and integrating changes from different contributors.

### 3.3.2 Docker

Docker is an open-source platform that allows you to automate the deployment, scaling, and management of applications using containerization. Containers are lightweight, portable, and isolated environments that package an application and its dependencies together.

**Key features of Docker:**

Containerization: Docker allows you to create containers that encapsulate applications and their dependencies, including libraries, runtime, and system tools. Containers ensure consistency between different environments, from development laptops to production servers.

Portability: Docker containers can run consistently across different environments, such as local development machines, testing servers, and cloud-based production environments. This eliminates the "it works on my machine" problem.

Efficiency: Containers share the host operating system's kernel and resources, which makes them more lightweight and efficient compared to traditional virtual machines. This leads to faster startup times and reduced resource consumption.

Version Control: Docker images are versioned, allowing you to track changes to your application and its dependencies over time. This facilitates rollback and ensures that you can recreate previous versions of your application easily.

Docker Images: Docker images are the blueprints for containers. They are created from a set of instructions defined in a Dockerfile, which specifies the base image, environment, configuration, and application code.

### 3.3.3 Jenkins

Jenkins is an open-source automation server that facilitates the continuous integration and continuous delivery (CI/CD) of software projects. It helps automate various tasks related to building, testing, and deploying applications, making the development and release process more efficient and reliable.

**Key features of Jenkins:**

Continuous Integration: Jenkins automates the process of integrating code changes from multiple contributors into a shared repository. It triggers builds whenever code is committed, allowing developers to identify and fix integration issues early.

Automated Builds: Jenkins can automatically build projects from source code repositories. It supports various build tools, languages, and platforms, making it versatile for different types of projects.

Extensibility: Jenkins can be extended through a wide range of plugins that provide additional functionalities. Plugins are available for source code management, build tools, testing frameworks, and deployment options.

Pipeline as Code: Jenkins uses a domain-specific language called Groovy to define build pipelines as code. This enables you to define complex workflows that include build, test, and deployment stages in a version-controlled script.

Continuous Delivery: Jenkins supports continuous delivery by automating the deployment process after successful builds. It can deploy applications to different environments, such as development, staging, and production.

Distributed Builds: Jenkins can distribute builds across multiple machines, allowing for parallel builds and improved build performance. This is particularly useful for large and resource-intensive projects.

### 3.3.4. Kubernetes

Kubernetes (K8s) is a powerful container orchestration platform that automates the deployment, scaling, and management of containerized applications across clusters of machines.

**Key Features:**

Automated Deployment & Scaling: Deploys applications efficiently across multiple nodes. Supports horizontal and vertical scaling to handle increased traffic automatically. Uses ReplicaSets to maintain a specified number of running instances.

Self-Healing Capabilities: Monitors the health of pods and automatically restarts failed containers. Replaces unhealthy nodes and reschedules workloads accordingly.

Load Balancing & Service Discovery: Distributes incoming traffic efficiently across multiple instances of an application. Uses Kubernetes Services to enable automatic service discovery within the cluster.

Multi-Cloud & On-Premises Support: Can run in any environment, including AWS, Google Cloud, Azure, and private data centers.

Secret & Configuration Management: Manages sensitive data (e.g., passwords, API keys) securely with Kubernetes Secrets. Separates application configurations from the code using ConfigMaps.

### 3.3.5. SonarQube

SonarQube is an automated code quality and security analysis tool that detects vulnerabilities, bugs, and code smells in software projects.

**Key Features:**

Static Code Analysis (SAST):Scans source code for security vulnerabilities without executing it . Detects SQL injection, cross-site scripting (XSS), buffer overflows, and other security flaws.

Multi-Language Support: Supports over 25+ programming languages, including Java, Python, JavaScript, C++, and more.

Code Quality & Maintainability: Identifies code duplication, bad practices, and complexity issues. Helps maintain clean, readable, and maintainable code.

Quality Gates & CI/CD Integration: Implements "Quality Gates", which set criteria for code quality before deployment. Easily integrates with Jenkins, GitHub Actions, GitLab CI/CD, and Azure DevOps.

Security Vulnerability Detection: Detects security risks and provides detailed remediation steps. Works with compliance standards like OWASP Top 10 and SANS CWE.

### 3.3.6. Trivy

Trivy is a lightweight and fast container security scanner used to detect vulnerabilities in container images, file systems, and infrastructure-as-code (IaC) configurations.

**Key Features:**

Container Image Scanning: Detects vulnerabilities in Docker and OCI images by scanning installed software packages and dependencies.

Infrastructure as Code (IaC) Security Scanning: Identifies misconfigurations in Kubernetes YAML files, Terraform scripts, and Helm charts. Ensures best security practices for cloud and containerized applications.

Fast and Efficient Scanning: Minimal setup required and provides quick results compared to other security scanners. Works both online (fetching vulnerability databases) and offline (pre-downloaded databases).

CVE (Common Vulnerabilities and Exposures) Detection: Continuously updates and checks against security databases like NVD, RedHat, Debian, Alpine, and GitHub Security Advisories. Integration with CI/CD Pipelines: Works seamlessly with Jenkins, GitHub Actions, GitLab CI, and Kubernetes to ensure security compliance before deployment.

### 3.3.7. Prometheus

Prometheus is an open-source monitoring and alerting tool designed for cloud-native environments. It collects and stores metrics in a time-series database, allowing real-time monitoring of system performance.

**Key Features:**

Multi-Dimensional Data Model: Stores time-series data with labels (key-value pairs) for flexible querying and filtering. Uses PromQL (Prometheus Query Language) for advanced analytics.

Time-Series Data Collection: Captures system and application performance metrics at regular intervals. Stores data in an optimized format for fast querying.

Alerting with Alertmanager: Sends alerts via email, Slack, PagerDuty, or other channels when metrics exceed defined thresholds. Helps detect CPU spikes, memory leaks, and network failures.

Kubernetes & Docker Integration: Works natively with Kubernetes, Docker, and cloud-native applications. Monitors container performance, resource usage, and application health.

Data Retention & Scalability: Can be scaled horizontally with federated Prometheus instances. Stores long-term monitoring data for historical trend analysis.

### 3.3.8. Grafana

Grafana is a real-time visualization and dashboard tool that integrates with Prometheus to display monitoring metrics in a user-friendly and customizable format.

**Key Features:**

Customizable Dashboards: Allows users to create dynamic dashboards with graphs, tables, and heatmaps. Supports templating to make dashboards more interactive.

Multi-Data Source Support: Connects with Prometheus, MySQL, PostgreSQL, Elasticsearch, Loki, InfluxDB, and many more. Allows cross-database queries for better insights.

Real-Time Monitoring & Alerts: Displays real-time system performance metrics like CPU, memory, disk usage, and network traffic. Generates alerts when performance metrics exceed predefined limits.

User Authentication & Access Control: Supports user authentication via LDAP, OAuth, and built-in login mechanisms. Provides role-based access control (RBAC) to limit user permissions. Export & Sharing Features: Enables exporting reports, snapshots, and embedding dashboards into applications. Facilitates collaboration among DevOps teams, developers, and security analysts.

# 4. PROJECT OUTPUT

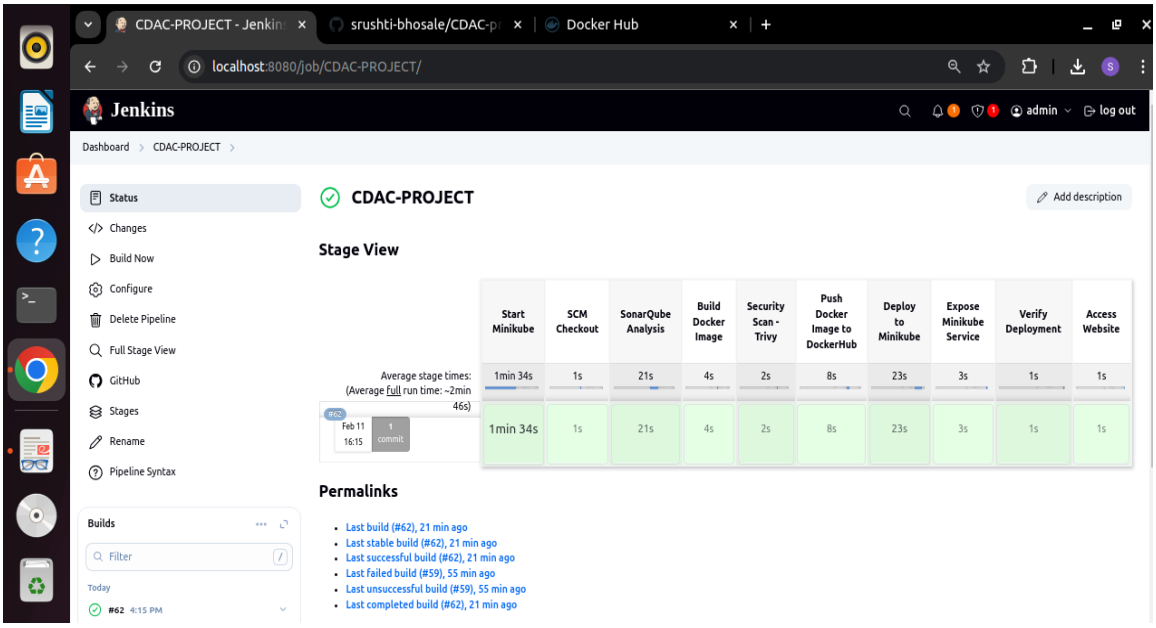


Fig. 4.1 Pipeline Stages

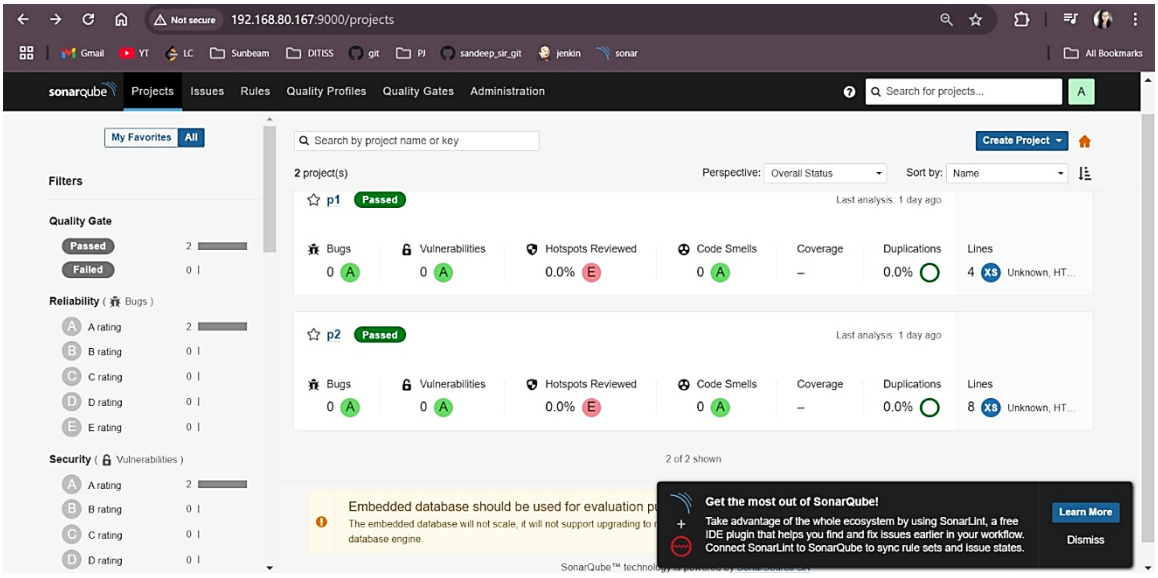


Fig. 4.2.Sonarqube

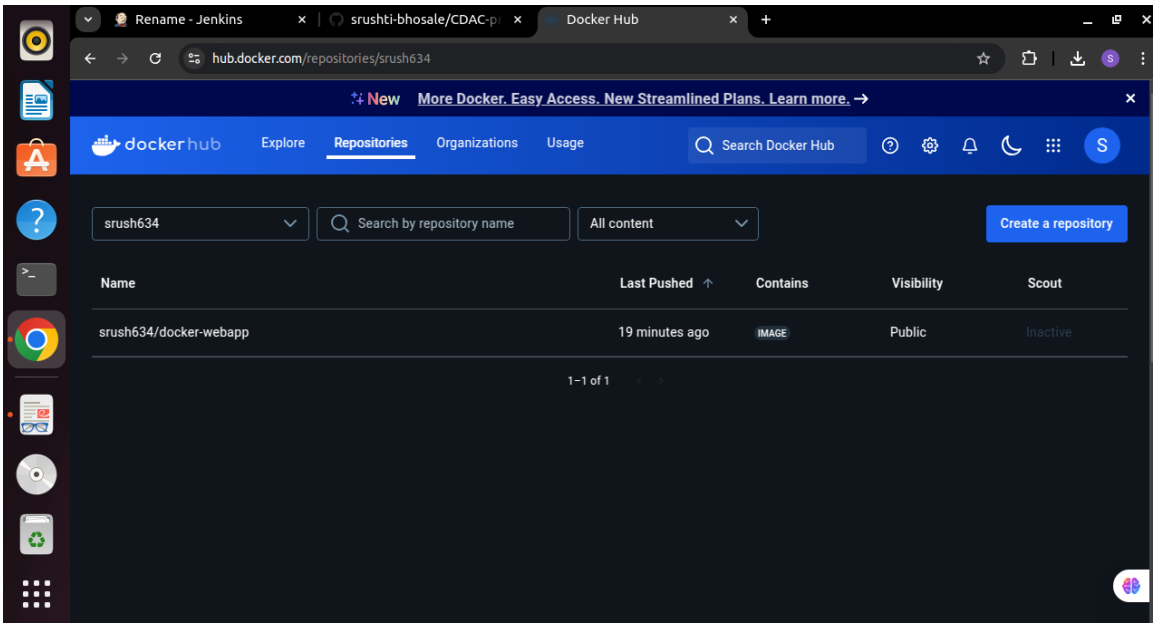


Fig. 4.3 Docker Image

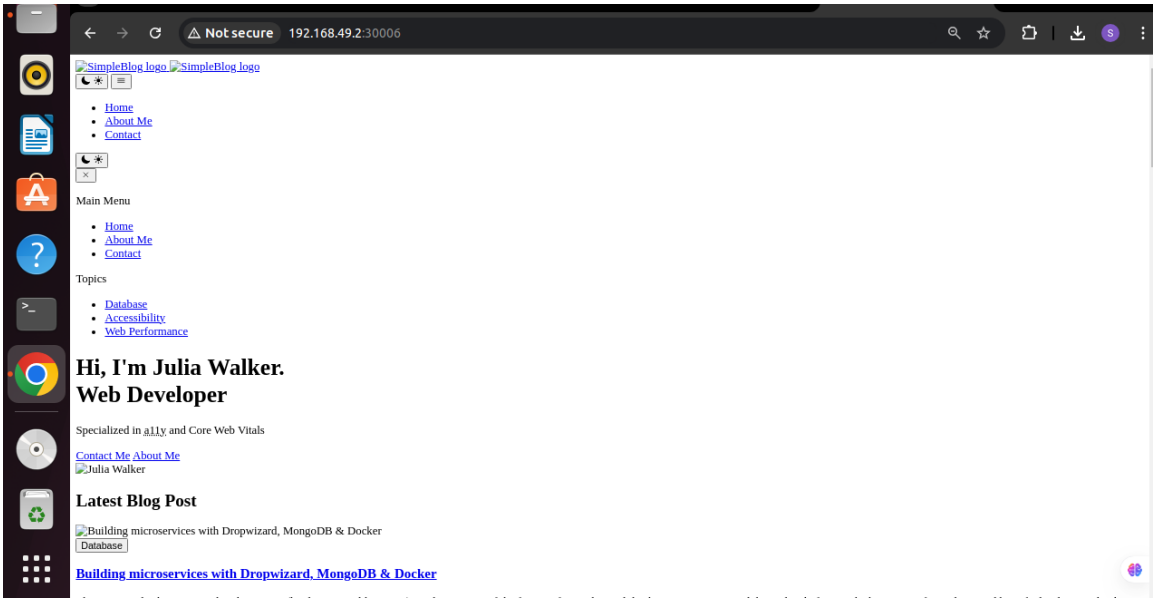


Fig. 4.4 Web Application

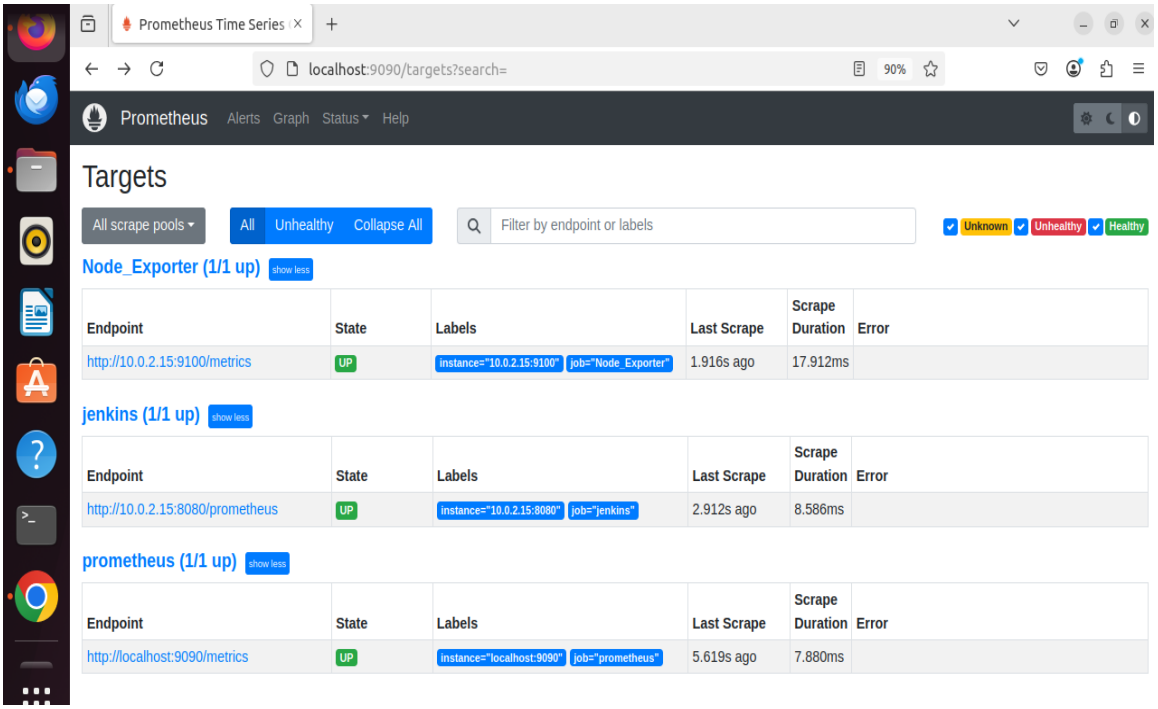


Fig. 4.5 Prometheus

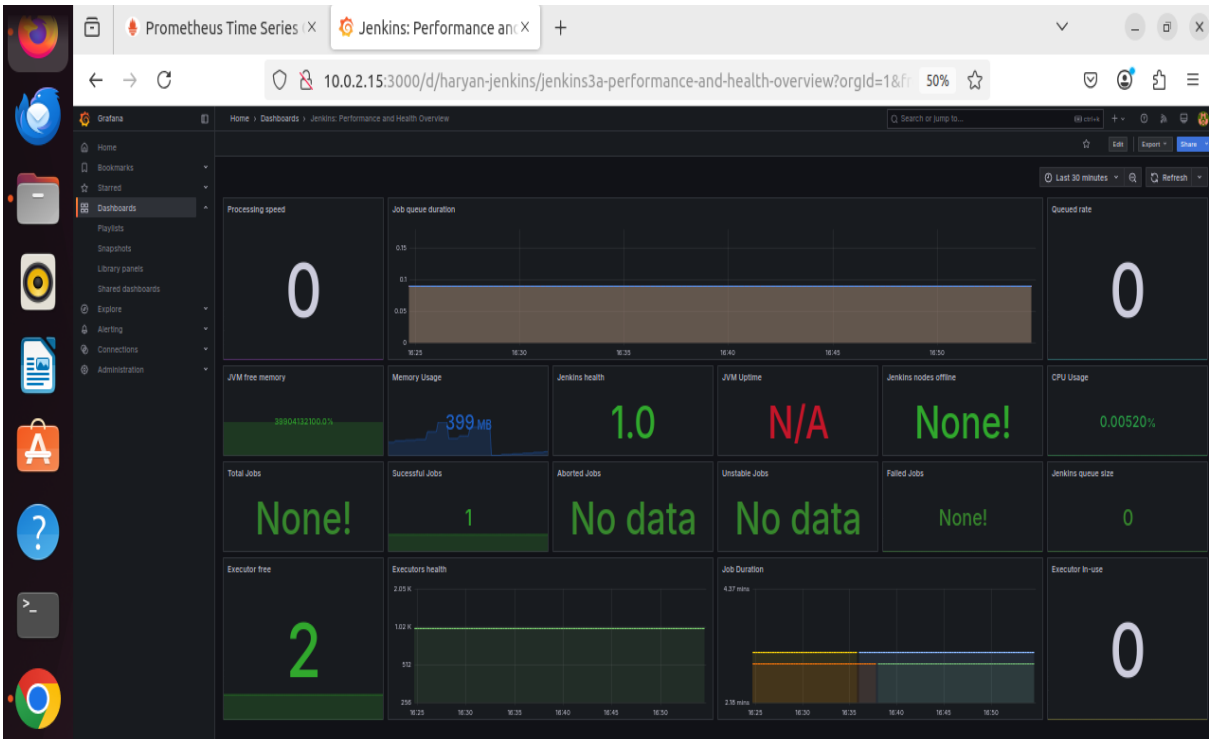


Fig. 4.6.Graphana

# 5. CONCLUSION

## 5.1 Conclusion

Hence, we have successfully implemented an end-to-end CI/CD Pipeline for Web Application Deployment significantly enhancing the efficiency, security, and reliability of software delivery. By integrating Jenkins for automation, Docker for containerization, and Kubernetes for scalable deployments, the pipeline ensures consistency across different environments. SonarQube and Trivy provide robust security scanning, identifying vulnerabilities early in the development lifecycle. Additionally, Prometheus and Grafana enable real-time monitoring and log analysis, ensuring proactive system performance tracking. This project reduces manual effort, minimizes security risks, and accelerates software release cycles, making it an ideal approach for modern DevOps-driven development. By leveraging these tools, organizations can achieve a secure, automated, and highly efficient deployment workflow, ultimately enhancing software quality and user experience.

## 5.2 Future Scope

The future scope of this project includes integrating AI/ML-driven DevOps for intelligent code analysis and resource optimization, enhancing security with Dynamic Application Security Testing (DAST) and Zero Trust policies, and expanding multi-cloud and serverless deployments for better scalability. Further improvements include automated incident response and self-healing mechanisms in Kubernetes, advanced observability with ELK Stack, and predictive analytics for performance optimization. Additionally, adopting GitOps practices with ArgoCD or Flux and enhancing Infrastructure as Code (IaC) with Terraform can further automate deployment processes. These advancements will transform the CI/CD pipeline into a highly automated, secure, and scalable system for modern web applications.

# REFERENCES

**Paper 1:.**Security for Continuous Integration and Continuous Deployment Pipeline"
**Author:** Sachin Vighe

**Paper 2:** A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines"
**Authors:** Thorsten Rangnau, Remco v. Buijtenen, Frank Fransen, Fatih Turkmen

**Paper 3:** Automating Security in a Continuous Integration Pipeline"
**Authors**: Sohrab Chalishhafshejani, Bao Khanh Pham, and Martin Gilje Jaatun