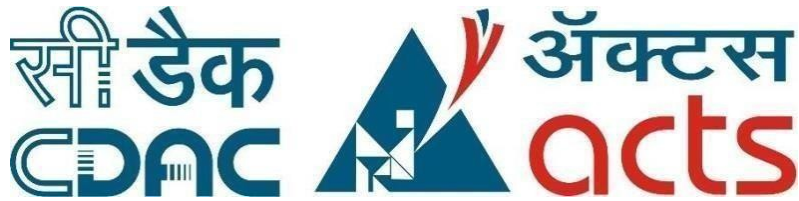


Project Report On
End to end deployment of Tier 3 application using
CI/CD



Submitted
In partial fulfilment
For the award of the Degree of
Post Graduate Diploma in IT Infrastructure System & Security
(C-DAC, ACTS (Pune))

Guided by:

Mr. Suyash Kulkarni

Submitted by:

Prathamesh Ankush Jadhav	PRN: 240840123040
Anisha Patil	PRN: 250240123003
Monali Nishad	PRN: 250240123025
Salunke Bhushan Laxman	PRN: 250240123038
Tulsi Santosh Warke	PRN: 250240123048

Centre of Development of Advanced Computing (C-DAC)
ACTS (Pune – 411008)



CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Prathamesh Ankush Jadhav	PRN: 240840123040
Anisha Patil	PRN: 250240123003
Monali Nishad	PRN: 250240123025
Salunke Bhushan Laxman	PRN: 250240123038
Tulsi Santosh Warke	PRN: 250240123048

Have successfully completed their project on

**“End to end deployment of Tier 3 application using
CI/CD”**

Under the guidance of

Mr. Suyash Kulkarni

Project Guide

Mr. Suyash Kulkarni

Course Coordinator

Ms. Divya Patel

HOD ACTS

Mr. Gaur Sunder

ACKNOWLEDGEMENT

This is to acknowledge our indebtedness to our Project Guide, **Mr. Suyash Kulkarni**, CDAC ACTS, Pune for his constant guidance and helpful suggestions for preparing this project **End to end deployment of tier 3 application using CI/CD**.

We express our deep gratitude towards his inspiration, personal involvement, constructive criticism that he provided us along with technical guidance during the course of this project.

We take the opportunity to thank head of department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment of our overall development.

We express sincere thanks to **Mr. Soumyakant Behera**, Process Owner, for their kind cooperation and their extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude towards **Mrs. Risha P.R.** (Program Head) and **Ms. Divya Patel** (Course Coordinator, PG-DITISS) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS** Pune, which provided us this opportunity to carry out this prestigious Project and enhance our learning in various technical fields.

Prathamesh Ankush Jadhav	PRN: 240840123040
Anisha Patil	PRN: 250240123003
Monali Nishad	PRN: 250240123025
Salunke Bhushan Laxman	PRN: 250240123038
Tulsi Santosh Warke	PRN: 250240123048

ABSTRACT

This project focuses on building a secure, automated, and scalable deployment pipeline for a Tier-3 application using CI/CD practices and container orchestration. The application is containerized using Docker and deployed on a local Kubernetes cluster managed through Minikube. Jenkins is employed to automate the continuous integration and deployment pipeline, while Git is used for source code management. Code quality is ensured through SonarQube, and security vulnerabilities in dependencies are identified early using OWASP Dependency-Check. Grafana and Prometheus are integrated into the Kubernetes environment for monitoring and visualization, providing insights into application performance and resource utilization. ArgoCD enables GitOps-based deployment, offering declarative and synchronized application delivery to Kubernetes. This setup streamlines the development lifecycle by integrating automation, security, and orchestration, aligning with modern DevSecOps principles and enhancing operational efficiency.

TABLE OF CONTENTS

S. No.	Title	Page. No.
1	Introduction and Overview of Project	
1.1	Purpose	1-2
1.2	Aims and Objective	
2	Prerequisites	
2.1	Hardware and Software Requirements	3-4
2.2	Jenkins, Docker, SonarQube, OWASP Dependency-Check etc.	
3	Overall Description	
3.1	Introduction	7-9
3.2	Data flow diagram	
3.3	Architecture	
4	CI/CD Lab Setup	10-14
5	Working	
5.1	Git	15-18
5.2	Jenkins	
5.3	Docker	
5.4	SonarQube	
5.5	OWASP	
5.6	ArgoCD-GitOps Deployment	
5.7	Prometheus Monitoring	
5.8	Grafanna	
6	Implementation and Output	19-28
7	Conclusion and Future Scope	29-30
8	References	31

1.INTRODUCTION AND OVERVIEW OF PROJECT

1.1 PURPOSE

The purpose of the project titled “**End to End Deployment of Tier 3 Application using CI/CD**” is to establish a secure, automated, and scalable deployment pipeline for a Tier-3 application. By leveraging Continuous Integration and Continuous Deployment (CI/CD) practices along with container orchestration, the project aims to enhance the efficiency and reliability of the application deployment process while ensuring high standards of security and performance monitoring.

1.2 AIM AND OBJECTIVE

- **Aim:** The primary aim of this project is to create a comprehensive deployment pipeline that automates the build, test, and deployment processes for a Tier-3 application, utilizing modern tools and practices to ensure security, scalability, and operational efficiency.
- **Objective:** The project seeks to achieve the following objectives:
 - **Automated Deployment Pipeline:** Build a fully automated CI/CD pipeline using **Jenkins** to facilitate continuous integration and deployment of the Tier-3 application.
 - **Containerization:** Utilize **Docker** to containerize the application, ensuring consistency across development, testing, and production environments.
 - **Kubernetes Orchestration:** Deploy the application on a local **Kubernetes** cluster managed through **Minikube**, enabling efficient resource management and scaling.
 - **Source Code Management:** Implement **Git** for effective source code management, allowing for version control and collaboration among development teams.
 - **Code Quality Assurance:** Integrate **SonarQube** into the pipeline to ensure high code quality through static code analysis and early detection of issues.
 - **Security Vulnerability Detection:** Use **OWASP Dependency-Check** to identify and address security vulnerabilities in application dependencies early in the development lifecycle.
 - **Monitoring and Visualization:** Integrate **Grafana** and **Prometheus** for realtime monitoring and visualization of application performance and resource utilization, providing insights for optimization.

- **GitOps Deployment:** Employ **ArgoCD** for GitOps-based deployment, ensuring declarative and synchronized application delivery to the Kubernetes environment.
- **Alignment with DevSecOps Principles:** Streamline the development lifecycle by integrating automation, security, and orchestration, aligning with modern **DevSecOps** principles to enhance operational efficiency.

By achieving these objectives, the project aims to create a robust and efficient deployment pipeline that supports the continuous delivery of high-quality software while maintaining security and performance standards.

2.PREREQUISITES

2.1 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

1. **Machine 1** (Jenkins, Docker, SonarQube, OWASP Dependency-Check):

- **RAM:** At least **4 GB**
- **CPU:** **2 cores**
- **Disk Space:** **20 GB**

2. **Machine 2** (Kubernetes, ArgoCD, Prometheus, Grafana):

- **RAM:** At least **8 GB**
- **CPU:** **2 cores**
- **Disk Space:** **40 GB**

If using cloud infrastructure, ensure the machines meet the recommended specifications for the respective services.

Software Requirements:

- **Operating System:** Preferably **Ubuntu (18.04 or higher)** or any Linux-based distribution.
- **Docker:** For containerizing the Spring Boot microservice and running containers in both environments.
- **Kubernetes Cluster:** Either **Minikube** (for local setups) or cloud-based Kubernetes services (like **AWS EKS, Google GKE, Azure AKS**).
- **Helm:** A package manager for Kubernetes that simplifies the deployment of Prometheus and Grafana.
- **Jenkins:** For automating the CI/CD pipeline.
- **ArgoCD:** For GitOps-style deployment in Kubernetes.
- **Prometheus:** For monitoring the application and infrastructure.
- **Grafana:** For visualizing metrics collected by Prometheus.

2.2 TOOLS AND TECHNOLOGIES

Ensure you have access to and install the following tools and technologies:

A) CI/CD and Deployment Tools:

- **Jenkins:**

- Must be installed and configured on Machine 1.
- **Plugins:** Install required Jenkins plugins like **GitHub**, **Docker Pipeline**, **SonarQube Scanner**, **Email Extension**, and **OWASP Dependency-Check**.
- **Jenkinsfile:** Write a Jenkins pipeline script that integrates with all other tools.

- **Docker Hub/Private Registry:** For pushing and pulling Docker images.

- **ArgoCD:** ○ Must be installed and running on Machine 2. ○ You need access to a Kubernetes cluster for ArgoCD to manage deployments.

- Access to your GitHub repository to set up GitOps deployment.

- **Kubernetes:** ○ Kubernetes cluster must be set up on Machine 2 (via Minikube or cloud-based Kubernetes).

- Configure **kubectl** on Machine 1 to interact with the Kubernetes cluster. ○

Helm must be installed for deploying Prometheus and Grafana.

- **Prometheus and Grafana:** ○ Install Helm on Machine 2 to deploy and configure Prometheus and Grafana.

B) Security and Code Quality Tools:

- **SonarQube:** ○ Must be installed on Machine 1.

- Required for static code analysis and ensuring code quality.

- **OWASP Dependency-Check:**

- Install the OWASP Dependency-Check plugin in Jenkins to scan for known vulnerabilities in dependencies.

C) Networking and Access Control

- **Machine 1 to Machine 2 Communication:** Ensure that Machine 1 (where Jenkins runs) can communicate with Machine 2 (where Kubernetes and ArgoCD are deployed). This may require configuring networking and firewall rules, especially if they are on different networks.
- **GitHub Access:** Ensure that your GitHub repository is set up and accessible for integration with Jenkins (via webhooks) and ArgoCD. A **GitHub Personal Access Token (PAT)** may be required for authentication between GitHub, Jenkins, and ArgoCD.
- **Docker Hub Access:** You need an account on Docker Hub or a private Docker registry to store and pull Docker images. Ensure your Jenkins pipeline has the credentials to push images to Docker Hub.

D) Basic Configuration Knowledge

- **Git:** Knowledge of Git and GitHub for repository management. Be able to set up and manage webhooks to trigger Jenkins pipelines.
- **Jenkins Pipeline:** Basic understanding of Jenkins pipelines and Jenkinsfile for automating build, test, and deployment processes.
- **Docker:** Understanding how to write a Dockerfile to containerize the application. Knowledge of Docker Compose (optional) for local testing of multi-container setups.
- **Kubernetes:** Basic knowledge of Kubernetes and how to deploy applications on Kubernetes using kubectl. Familiarity with Kubernetes YAML files for deploying applications. Ability to manage namespaces, pods, deployments, services, and ingress in Kubernetes.
- **ArgoCD:** Basic understanding of how ArgoCD works with GitOps to manage the deployment lifecycle. Setting up ArgoCD Applications and Syncing to deploy services automatically.
- **Prometheus & Grafana:** Basic knowledge of how to install and configure Prometheus for monitoring. Understanding

E) Access and Credentials

- **GitHub Credentials:**

- Access to a GitHub repository for storing the source code.
- GitHub token or SSH key for Jenkins integration (to pull code).

- **Docker Hub Credentials:**

- Access to a Docker Hub account to push and pull images.
- Docker registry credentials configured in Jenkins to allow image push from Jenkins.

- **Kubernetes and ArgoCD Credentials:**

- A kubeconfig file to interact with the Kubernetes cluster.
- Ensure Jenkins has the right permissions to trigger deployments on ArgoCD using the ArgoCD API or Webhooks.

F) Security Considerations

- **SonarQube:** Configure SonarQube authentication and set up quality gates for the build to ensure code quality and security.
- **OWASP Dependency-Check:** Ensure OWASP Dependency-Check is set up correctly in Jenkins for vulnerability scanning of project dependencies.
- **Prometheus Alerts:** Configure alert rules in Prometheus for any infrastructure or application issues (e.g., high CPU or memory usage) to proactively monitor system health.
- **Email Notification:** Set up email alerts for Jenkins, Prometheus, and Grafana to notify the team in case of build failures or application issues, ensuring timely responses to critical events.

3. OVERALL DESCRIPTION

3.1 INTRODUCTION:

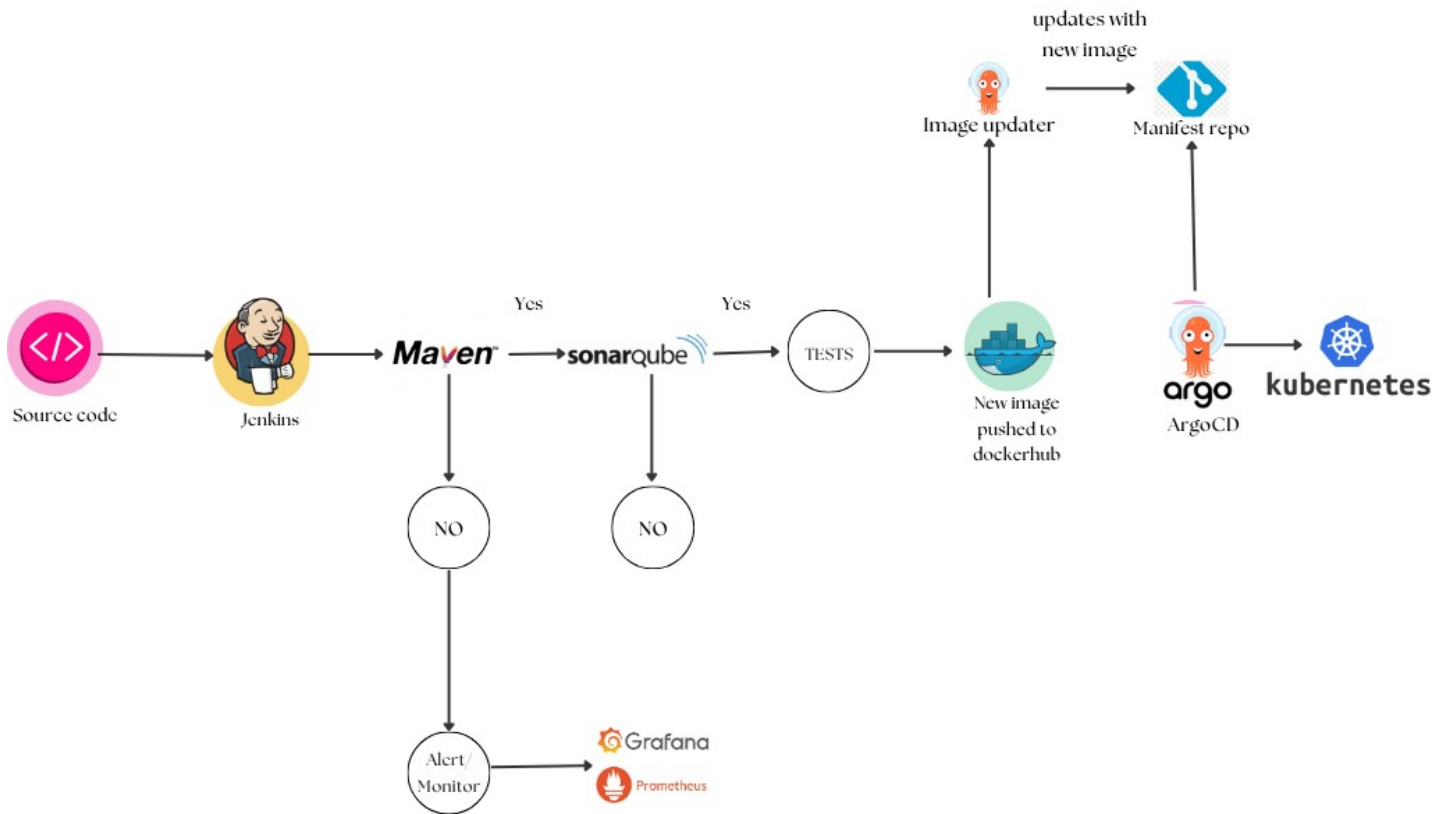
In today's fast-paced software development landscape, the demand for rapid, reliable, and secure application deployment has surged. Traditional deployment methods, often reliant on manual configurations, can lead to errors, inefficiencies, and security vulnerabilities. To overcome these challenges, organizations are increasingly adopting Continuous Integration and Continuous Deployment (CI/CD) practices, which automate the entire software delivery pipeline.

This project, titled “Automated CI/CD with Security, Monitoring & GitOps Deployment,” aims to design and implement a fully automated CI/CD pipeline specifically for deploying a Spring Boot microservice within a Kubernetes environment. The solution leverages Jenkins for automation, ArgoCD for GitOpsbased deployment, and Docker for containerization, ensuring smooth and consistent deployments.

To bolster security and enhance code quality, the pipeline integrates several essential tools, including SonarQube for static code analysis, OWASP Dependency-Check for security vulnerability scanning, and Trivy for container security assessments. Furthermore, Prometheus and Grafana are utilized for real-time monitoring, continuously tracking application performance and health metrics.

Automated workflows are triggered through GitHub and Jenkins webhooks, facilitating seamless deployments upon code changes. Additionally, email notifications are integrated to provide proactive alerts for pipeline events. By embracing GitOps principles with ArgoCD, this project ensures a declarative, version-controlled, and automated approach to infrastructure management, paving the way for more efficient and secure software delivery.

3.2 DATA FLOW DIAGRAM



The code changes automatically trigger a Jenkins pipeline that builds, tests, and analyzes the project. Upon successful completion of the build process, a Docker image is created and subsequently deployed to a Kubernetes cluster using Argo CD for continuous delivery. In the event of build failures, detailed reports are generated to provide insights into the issues encountered, facilitating quicker resolution and improving overall development efficiency.

3.3 ARCHITECTURE

The architecture of the system is designed to facilitate continuous integration and continuous deployment (CI/CD) through a series of automated processes. Here's a breakdown of the key components:

Continuous Integration with Jenkins

- **Triggering Builds:** Code changes in the repository automatically trigger Jenkins to initiate the build process.
- **Build and Test:** Jenkins compiles the code, runs unit tests, and performs static code analysis to ensure code quality.
- **Failure Reporting:** If the build or tests fail, Jenkins generates detailed reports that highlight the issues, allowing developers to address them promptly.

Docker Image Creation

- **Successful Builds:** Upon successful completion of the build and tests, Jenkins creates a Docker image of the application.
- **Versioning:** Each Docker image is tagged with a version number, ensuring that deployments can be tracked and managed effectively.

Deployment to Kubernetes

- **Argo CD Integration:** The Docker image is deployed to a Kubernetes cluster using Argo CD, which manages the deployment process and ensures that the application is running in the desired state.
- **Continuous Delivery:** Argo CD continuously monitors the application and automatically applies updates when new images are available, facilitating seamless updates and rollbacks.

Monitoring and Reporting

- **Failure Reports:** In case of build failures, Jenkins generates comprehensive reports that include logs and error messages, aiding in troubleshooting.
- **Performance Monitoring:** The deployed application is monitored for performance and reliability, ensuring that any issues are detected and addressed quickly.

4. CI/CD LAB SETUP

To set up CI/CD pipeline with Jenkins, ArgoCD, and Kubernetes on two local machines, we will follow these steps:

Machine 1 (Primary Machine) will host:

- Jenkins for CI/CD pipeline orchestration
- Docker for building container images
- SonarQube for static code analysis
- OWASP Dependency-Check for security scanning

Machine 2 (Secondary Machine) will host:

- Kubernetes Cluster (using Minikube or Docker Desktop's Kubernetes)
- ArgoCD for GitOps deployment
- Prometheus and Grafana for monitoring
- Docker (for pushing and pulling container images)
- 1. Machine 1: Jenkins, SonarQube, and Docker Setup
- 1.1 Install Docker on Machine 1
- First, install Docker on Machine 1 (which will build and push the Docker image).

Commands for Ubuntu:

```
bash Copy sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER # Adds user to the docker group Test
```

Docker installation:

```
bash Copy
docker --version
```

1.2 Install Jenkins on Machine 1

Install Jenkins on Machine 1, which will handle the CI pipeline.

Commands for Ubuntu:


```
bash Copy sudo apt update sudo apt install -y openjdk-11-jdk wget -q -O -  
https://pkg.jenkins.io/keys/jenkins.io.key | sudo apt-key add - sudo sh -c  
'echo deb http://pkg.jenkins.io/debian-stable/ /' >  
/etc/apt/sources.list.d/jenkins.list  
sudo apt update sudo apt install -  
y jenkins sudo systemctl start  
jenkins sudo systemctl enable  
jenkins  
Access Jenkins via http://<Machine1-IP>:8080.
```

1.3 Install Jenkins Plugins

Log in to Jenkins and install the following plugins:

GitHub Plugin (to integrate with GitHub repositories)

Docker Pipeline (for Docker image build)

SonarQube Scanner for Jenkins (for SonarQube integration)

OWASP Dependency-Check Plugin (for security scanning)

Email Extension Plugin (for email notifications)

Go to Manage Jenkins → Manage Plugins → Available tab, search, and install these plugins.

1.4 Install and Configure SonarQube on Machine 1 Install

SonarQube locally on Machine 1.

Commands for Ubuntu:

```
bash Copy sudo apt update sudo  
apt install -y openjdk-11-jdk
```

```
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.x.x.zip
```

```
unzip sonarqube-9.x.x.zip
```

```
cd sonarqube-9.x.x/bin/linux-x86-64
```

```
./sonar.sh start
```

Access SonarQube at <http://<Machine1-IP>:9000> to configure the project.

1.5 Configure Jenkins Pipeline

Create a Jenkinsfile in your GitHub repository to define the stages for:

Pulling the code from GitHub

Building the Docker image

Running SonarQube analysis

Pushing the Docker image to Docker Hub

Running OWASP Dependency-Check for security scanning
Triggering ArgoCD to deploy (via webhook)

2. Machine 2: Kubernetes, ArgoCD, Prometheus, and Grafana Setup

2.1 Install Docker on Machine 2

Install Docker on Machine 2 for container management.

Commands for Ubuntu:

```
bash Copy sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

Test Docker installation:

```
bash Copy
docker --version
```

2.2 Install Minikube for Kubernetes (Optional) Commands for Ubuntu:

```
bash Copy sudo apt update sudo apt install -y curl wget apt-transport-https
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64 sudo install minikube-linux-amd64 /usr/local/bin/minikube
minikube start
```

Verify Kubernetes is running with:

```
bash Copy
kubectl get nodes
```

2.3 Install ArgoCD on Machine 2 ArgoCD is the GitOps tool used to automate the deployment of your microservice to Kubernetes.

Commands to Install ArgoCD in Kubernetes (Machine 2):

```
bash
Copy kubectl create namespace argocd kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-
cd/stable/manifests/install.yaml
```

kubectl port-forward svc/argocd-server -n argocd 8080:80 Access
ArgoCD UI via <http://localhost:8080> (default login:
admin/<password>).

2.4 Install Prometheus and Grafana on Kubernetes (Machine 2)

To monitor the application, install Prometheus and Grafana in the Kubernetes cluster.

Commands to Install Prometheus using Helm:

```
bash Copy helm repo add prometheus-community  
https://prometheus- community.github.io/helm-charts  
helm repo update  
helm install prometheus prometheus-community/prometheus
```

Commands to Install Grafana using Helm:

```
bash Copy helm repo add grafana  
https://grafana.github.io/helm-charts helm repo update  
helm install grafana grafana/grafana  
Access Grafana via the port-forwarded service on http://localhost:3000 (default login:  
admin/admin).
```

2.5 Configure ArgoCD Application for GitOps

In ArgoCD, create an Application that points to your GitHub repository (or a separate repository containing your Kubernetes deployment YAML files). This will allow ArgoCD to deploy your Spring Boot microservice automatically whenever changes are made to the repository.

2.6 Set Up Webhook Trigger from Jenkins to ArgoCD

In Jenkins, configure a Webhook trigger to notify ArgoCD whenever a new image is pushed to Docker Hub or changes are made to the deployment files in GitHub.

3. Integrating Jenkins and ArgoCD

3.1 Configure Jenkins to Trigger ArgoCD

Use Jenkins to trigger deployments to ArgoCD via webhooks:

In Jenkins, after a successful build and push to Docker Hub, set up a webhook to notify ArgoCD.

You can configure this through the ArgoCD CLI or by creating a job in Jenkins that calls ArgoCD's REST API to synchronize the application.

3.2 Configure Prometheus and Grafana Alerts

Set up Prometheus to monitor critical application metrics (e.g., CPU, memory usage) and configure Grafana to visualize the data. Set up alerts in Prometheus or Grafana to send notifications (e.g., email)

4. Final Steps Push Code to GitHub: Push the Spring Boot application code and Docker-related files to your GitHub repository.

Jenkins CI Pipeline: Configure Jenkins to automatically build and test the code, run SonarQube analysis, build the Docker image, and push the image to Docker Hub.

GitOps Deployment via ArgoCD: ArgoCD will automatically deploy the application to the Kubernetes cluster when it detects changes in the Git repository.

Monitoring with Prometheus and Grafana: Monitor the application's health and performance using Prometheus and Grafana, ensuring real-time alerts and proactive issue tracking.

5. WORKING

5.1 GitHub - Source Code Management

Workflow:

Developer pushes code changes: Developers push their code to a GitHub repository, which holds the source code of the Spring Boot microservice.

GitHub triggers Jenkins pipeline: GitHub has a webhook configured to notify Jenkins whenever code changes (push events) happen in the repository. The webhook triggers the Jenkins build process automatically.

Role in the CI/CD Pipeline:

Stores the source code of the application.

Acts as the central version control system that integrates with Jenkins to start the CI pipeline.

5.2 Jenkins

Jenkins listens for code changes: Jenkins continuously listens for new pushes or changes in the GitHub repository through the configured webhook.

Build and Test: Once code changes are detected, Jenkins pulls the latest code, triggers the build process (using Maven/Gradle), and runs unit tests.

Jenkins uses a Jenkins file to define the build pipeline stages.

Code Analysis:

SonarQube performs static code analysis to ensure code quality by detecting bugs, vulnerabilities, and code smells.

OWASP Dependency-Check scans project dependencies to detect known vulnerabilities in the libraries used by the Spring Boot application.

Dockerization: Jenkins then builds a Docker image of the Spring Boot application using the Dockerfile. The Docker image is then tagged with a version number and pushed to a Docker registry (Docker Hub in this case).

Push Image to Docker Hub: Once the Docker image is built and tagged, Jenkins pushes the image to Docker Hub, where it can be pulled by Kubernetes for deployment.

Role in the CI/CD Pipeline:

- Automates the build, test, and code quality checks.
- Integrates with Docker to create container images.
- Runs security scans on the code and dependencies.
- Pushes Docker images to a registry for deployment.

5.3 Docker – Containerization Workflow:

Docker Image Build: Jenkins uses Docker to create a containerized version of the Spring Boot application. The Dockerfile is used to define how the application should be packaged and run in a container.

Docker Push: After the image is built, it is pushed to a Docker registry (Docker Hub), where it will be stored and pulled by Kubernetes when needed for Deployment.

Role in the CI/CD Pipeline:

Containerizes the Spring Boot application to ensure consistency across different environments (dev, test, prod).

Ensures that the application can be deployed consistently in any environment (Kubernetes).

5.4 Kubernetes – Container Orchestration

Workflow:

Minikube or Cloud Kubernetes Cluster: If you're using Minikube or any cloud-based Kubernetes service (e.g., AWS EKS, Azure AKS), Kubernetes acts as the orchestrator for the deployment of the application.

Kubernetes Deploys Docker Containers: Kubernetes pulls the latest Docker image from Docker Hub (or any other registry) and creates containers based on it.

Pods and Services: Kubernetes creates pods to host the application and exposes the application using a Service for internal or external access.

Scaling and Self-healing: Kubernetes manages the scaling of the application and ensures high availability. If a pod fails, Kubernetes automatically restarts it.

Role in the CI/CD Pipeline:

Orchestrates the deployment of Docker containers to the cluster.

Handles scaling, load balancing, and self-healing of the application.

5.5 ArgoCD - GitOps Deployment

Workflow:

ArgoCD Syncs with GitHub: ArgoCD continuously monitors a Git repository (which contains Kubernetes deployment YAML files) for changes. When a change is detected, ArgoCD automatically triggers a deployment.

GitOps Workflow: The YAML files in the Git repository define the Kubernetes resources, such as Deployments, Services, and Ingresses. Any changes in these files are detected by ArgoCD, and the corresponding deployment or configuration is automatically applied to the Kubernetes cluster.

Automatic Deployment: ArgoCD uses the GitOps approach where it ensures that the cluster state matches the desired state defined in the Git repository. If a developer modifies the Kubernetes configuration in Git, ArgoCD syncs and updates the application automatically.

Role in the CI/CD Pipeline:

Automates the deployment of application changes using GitOps.

Ensures that the state of the Kubernetes cluster is always in sync with the repository.

5.6 SonarQube - Static Code Analysis

Workflow:

Code Analysis: As part of the Jenkins pipeline, SonarQube analyzes the source code of the Spring Boot application. It looks for issues such as:

- **Code Smells:** Unnecessary complexity, bad practices.
- **Bugs:** Potential logical issues in the code.
- **Vulnerabilities:** Security issues that could lead to breaches.
- **Test Coverage:** Ensures that unit tests are sufficiently covering the code.

SonarQube Results: Once the analysis is complete, the results are displayed in the SonarQube dashboard, allowing developers to review and fix issues.

Role in the CI/CD Pipeline:

Ensures code quality by providing detailed analysis and reports.

Helps prevent bugs and security vulnerabilities from reaching production.

5.7 OWASP Dependency-Check - Security

Scanning

Dependency Scanning: OWASP Dependency-Check runs as part of the Jenkins build process. It scans the project dependencies for any known vulnerabilities listed in public databases (e.g., NVD - National Vulnerability Database).

Reporting: After the scan, it generates a report indicating whether any of the libraries or dependencies used by the Spring Boot application have known security vulnerabilities.

Role in the CI/CD Pipeline:

Detects vulnerabilities in the project dependencies before deployment. Provides a report that helps developers address security issues in third-party libraries.

5.7 Prometheus – Monitoring Workflow:

Collect Metrics: Prometheus is configured to monitor the application and collect metrics from the running Spring Boot application, Kubernetes cluster, and other infrastructure components (e.g., memory usage, CPU utilization, response times).

Time Series Database: Prometheus stores these metrics as time series data and makes them available for querying.

Alerting: Prometheus can be configured with alert rules to notify stakeholders when predefined thresholds are breached (e.g., CPU usage exceeds 90%).

Role in the CI/CD Pipeline:

Monitors the health and performance of the deployed application and infrastructure. Sends alerts in case of issues, allowing for proactive management.

5.8 Grafana - Visualization and Alerts

Workflow:

Query Prometheus Data: Grafana integrates with Prometheus to query the collected metrics and visualize them in real-time on customizable dashboards.

Create Dashboards: Grafana users can create dashboards that display vital application metrics, such as response time, error rates, and resource utilization.

Alerting: Grafana can also configure alerts based on the data it receives from Prometheus, notifying users via email, Slack, or other channels when something goes wrong.

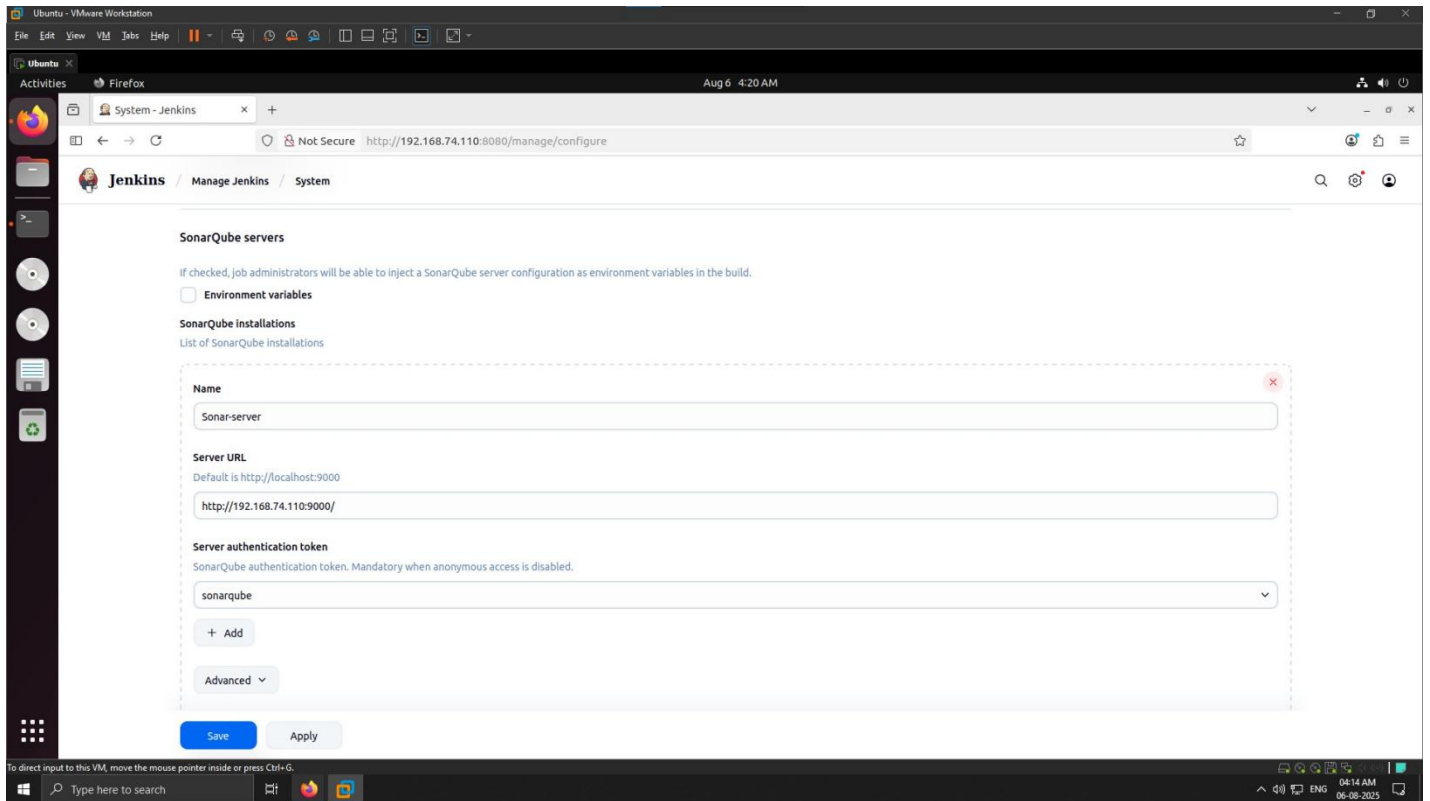
Role in the CI/CD Pipeline:

Provides real-time visualization of application and infrastructure metrics.

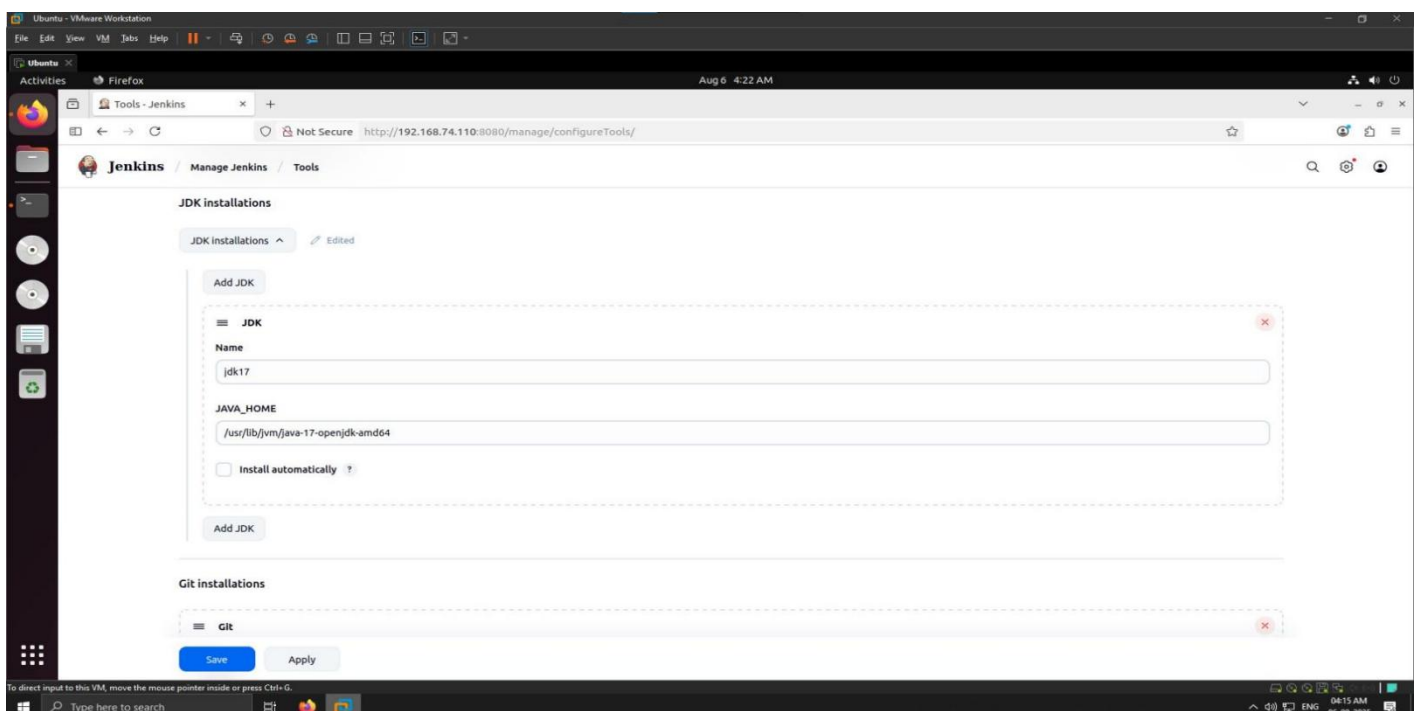
Enhances observability and proactive monitoring with alerting.

IMPLEMENTATION & OUTPUT

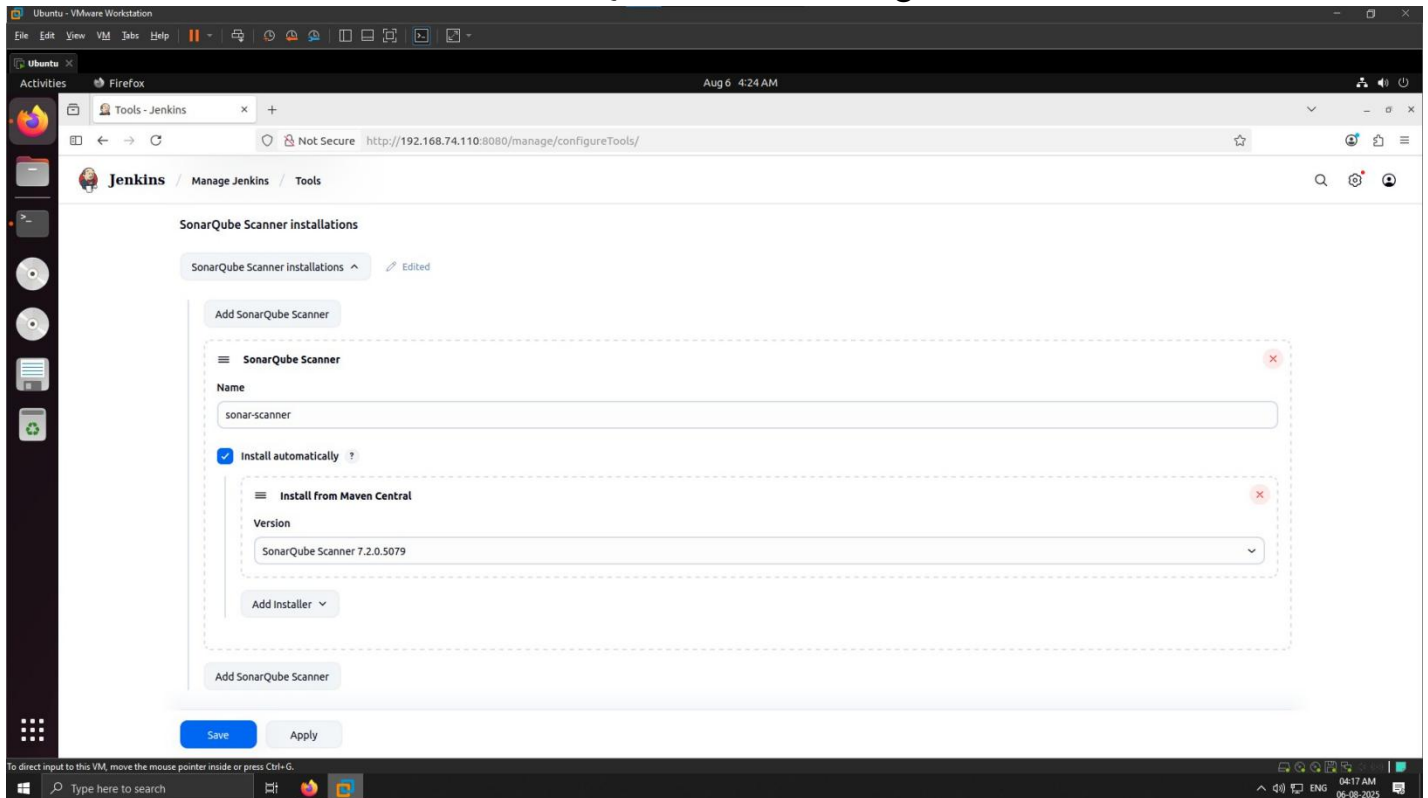
Configuration sonar-server



JDK Configuration

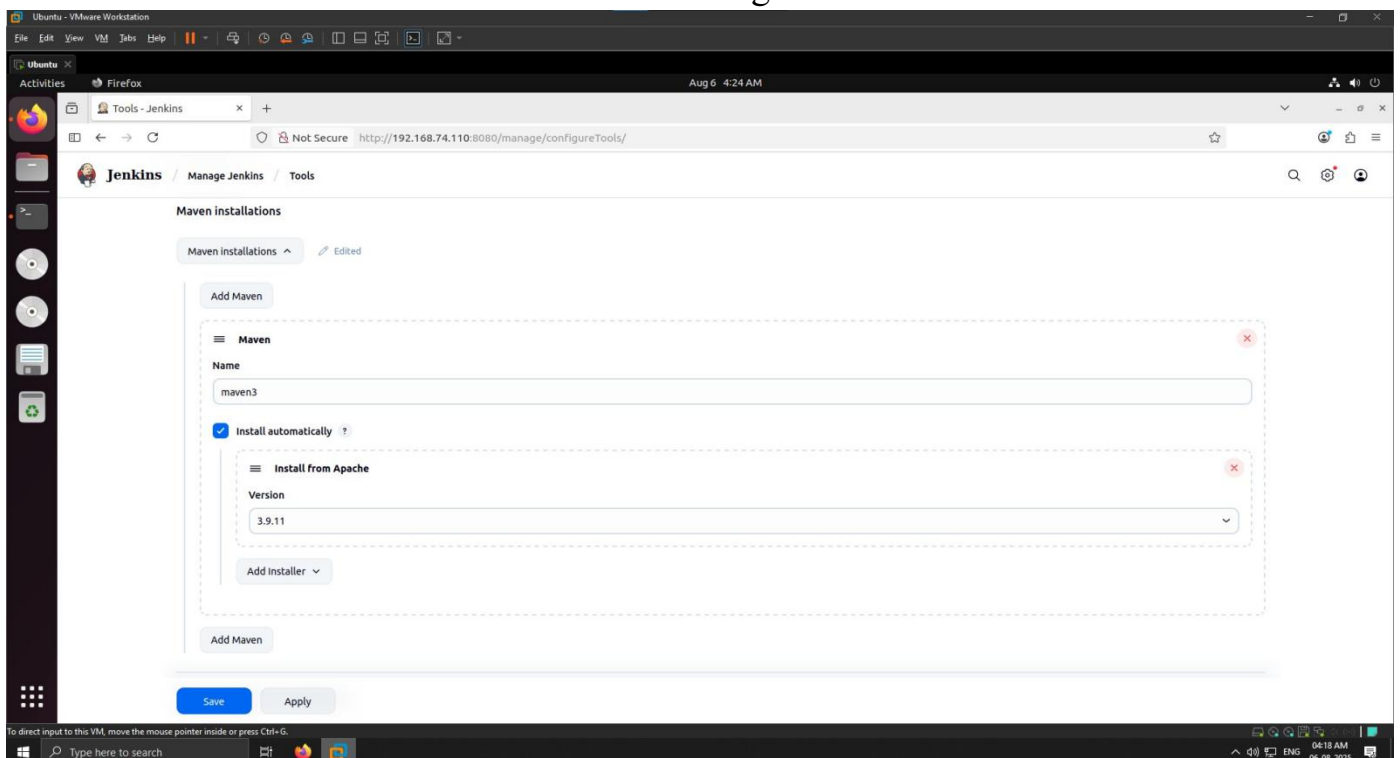


SonarQube Scanner Configuration



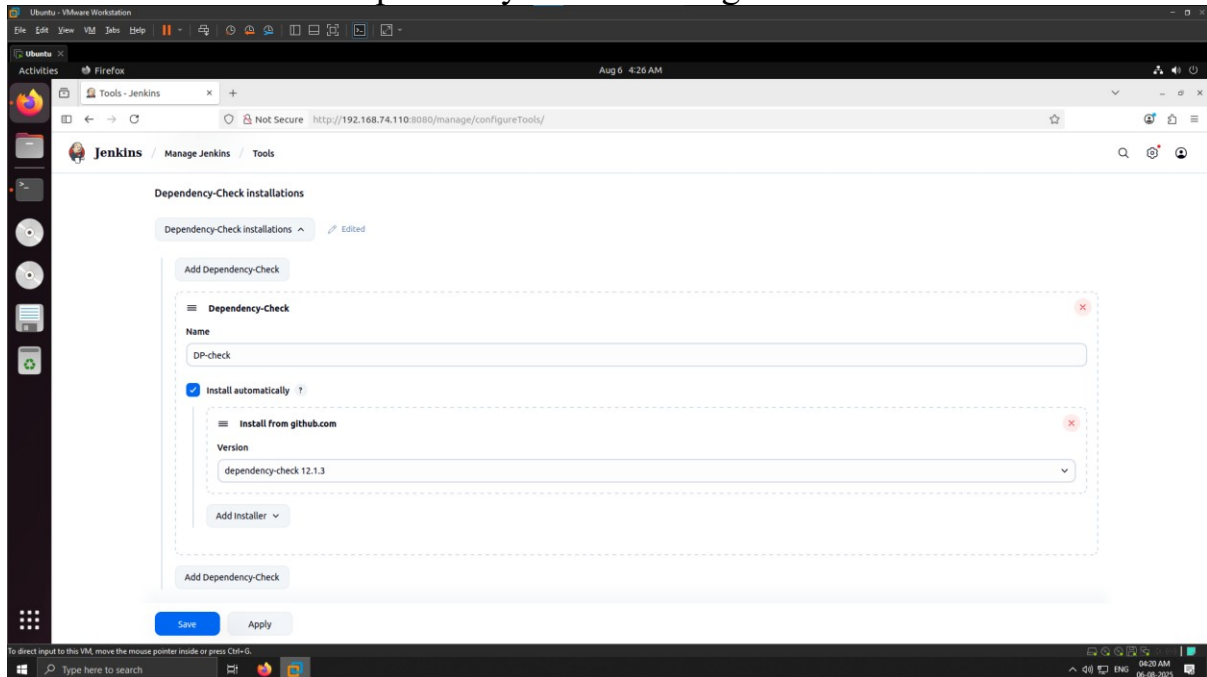
The screenshot shows the Jenkins web interface in a Firefox browser window. The address bar displays the URL `http://192.168.74.110:8080/manage/configureTools/`. The page title is "Jenkins / Manage Jenkins / Tools". The main section is titled "SonarQube Scanner installations". It features a dashed box for adding a new scanner. Inside this box, the "Name" field is set to "sonar-scanner". The "Install automatically" checkbox is checked. Below this, there is a sub-section titled "Install from Maven Central" with a "Version" dropdown menu set to "SonarQube Scanner 7.2.0.5079". At the bottom of the dashed box is an "Add installer" button. Below the dashed box is another "Add SonarQube Scanner" button. At the very bottom of the configuration area are "Save" and "Apply" buttons. The browser's status bar at the bottom shows the time as 04:17 AM on 06-08-2023.

Maven Configuration

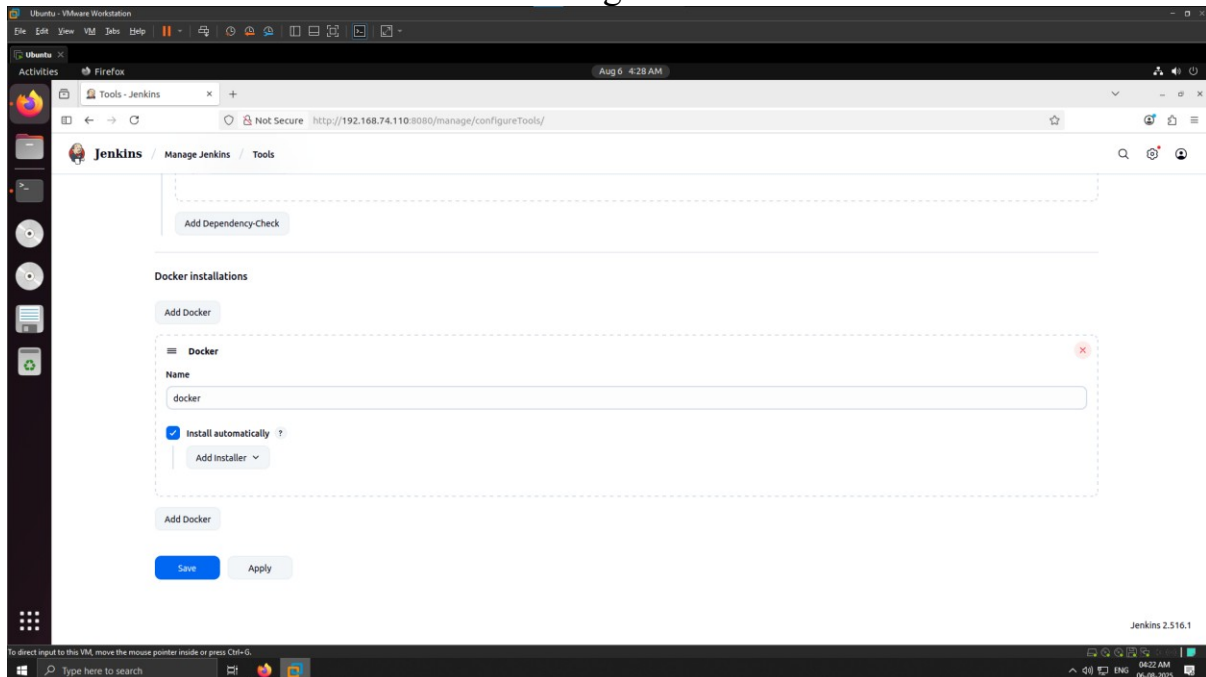


The screenshot shows the Jenkins web interface in a Firefox browser window. The address bar displays the URL `http://192.168.74.110:8080/manage/configureTools/`. The page title is "Jenkins / Manage Jenkins / Tools". The main section is titled "Maven installations". It features a dashed box for adding a new Maven installation. Inside this box, the "Name" field is set to "maven3". The "Install automatically" checkbox is checked. Below this, there is a sub-section titled "Install from Apache" with a "Version" dropdown menu set to "3.9.11". At the bottom of the dashed box is an "Add installer" button. Below the dashed box is another "Add Maven" button. At the very bottom of the configuration area are "Save" and "Apply" buttons. The browser's status bar at the bottom shows the time as 04:18 AM on 06-08-2023.

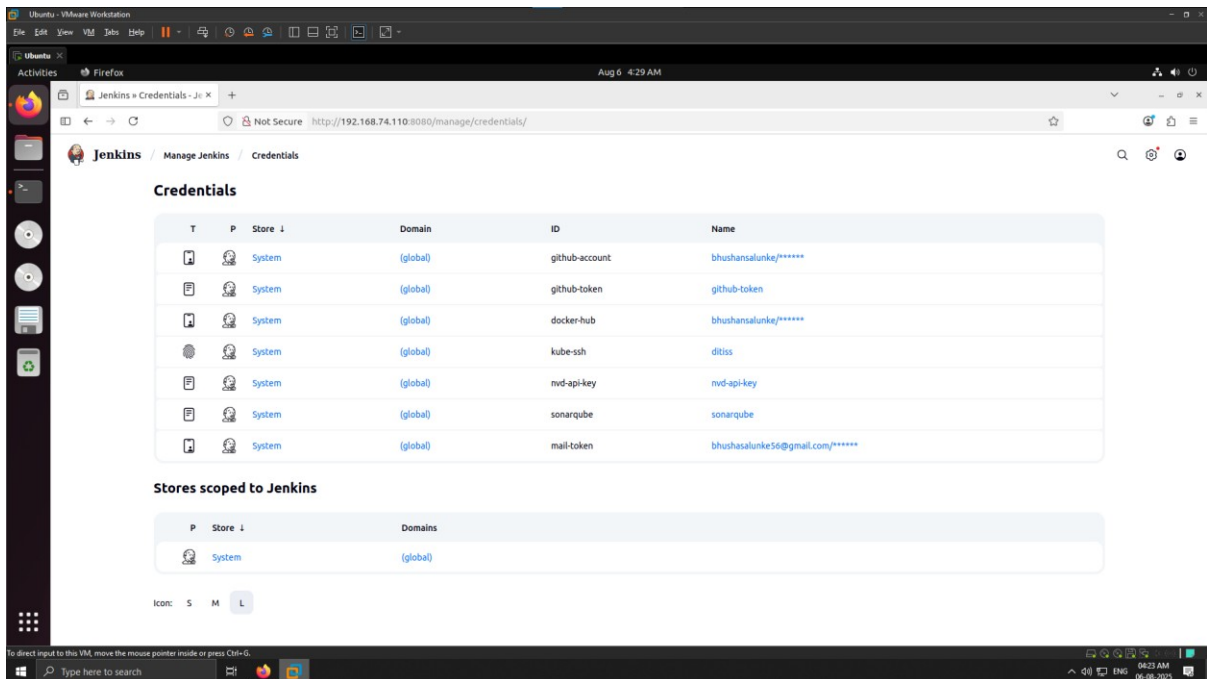
Dependency-Check Configuration



Docker Configuration



Credentials



The screenshot shows the Jenkins web interface in a Firefox browser. The address bar displays the URL `http://192.168.74.110:8080/manage/credentials/`. The page title is "Jenkins - Manage Jenkins - Credentials". The main content area is titled "Credentials" and contains a table with the following data:

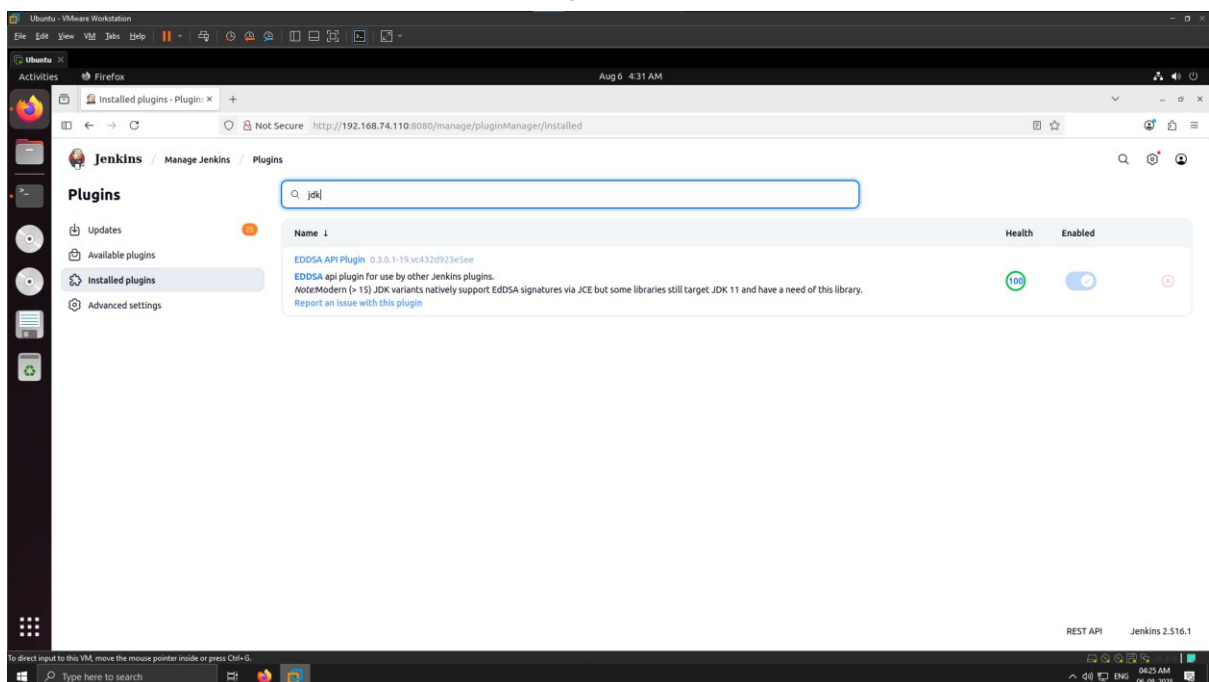
T	P	Store	Domain	ID	Name
		System	(global)	github-account	bhushansalunke/*****
		System	(global)	github-token	github-token
		System	(global)	docker-hub	bhushansalunke/*****
		System	(global)	kube-ssh	ditiss
		System	(global)	mvd-api-key	mvd-api-key
		System	(global)	sonarqube	sonarqube
		System	(global)	mail-token	bhushansalunke56@gmail.com/*****

Below the table, there is a section titled "Stores scoped to Jenkins" which shows a single entry:

P	Store	Domains
	System	(global)

The bottom of the screen shows the Ubuntu desktop environment with the taskbar and system tray.

Plugins JDK



The screenshot shows the Jenkins web interface in a Firefox browser. The address bar displays the URL `http://192.168.74.110:8080/manage/pluginManager/installed`. The page title is "Jenkins - Manage Jenkins - Plugins". The main content area is titled "Plugins" and contains a search bar with the text "jdk". Below the search bar, there is a table with the following data:

Name	Health	Enabled
EDDSA API Plugin 0.3.0-1-19.vc432d923e5ee EDDSA api plugin for use by other Jenkins plugins. Note: Modern (> 15) JDK variants natively support EdDSA signatures via JCE but some libraries still target JDK 11 and have a need of this library. Report an issue with this plugin		

The bottom of the screen shows the Ubuntu desktop environment with the taskbar and system tray.

Plugins Docker

The screenshot shows the Jenkins web interface in a Firefox browser. The 'Plugins' tab is selected, and a search bar contains the text 'docker'. The table below lists the installed Docker-related plugins.

Name	Health	Enabled
Docker Commons Plugin 457.v0f62a_94f11a_3 Provides the common shared functionality for various Docker-related plugins. Report an issue with this plugin	100	<input checked="" type="checkbox"/>
Docker Pipeline 621.va_73f881d9232 Build and use Docker containers from pipelines. Report an issue with this plugin	97	<input checked="" type="checkbox"/>

REST API Jenkins 2.516.1

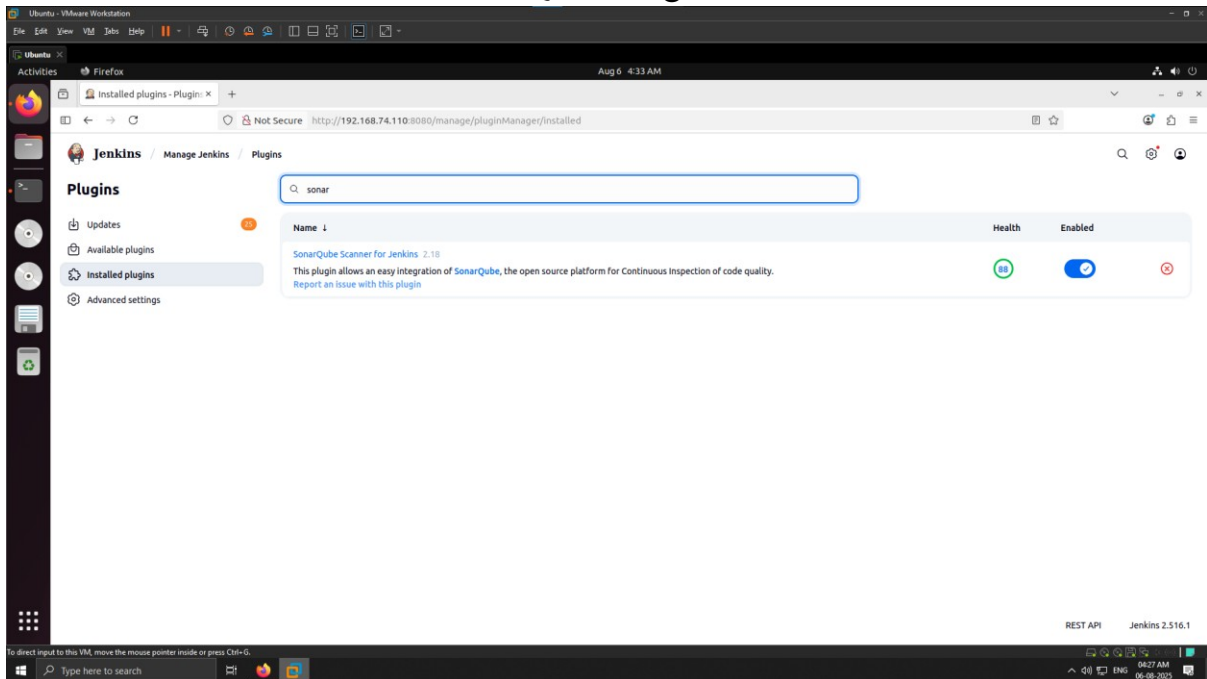
OWASP Plugins

The screenshot shows the Jenkins web interface in a Firefox browser. The 'Plugins' tab is selected, and a search bar contains the text 'owasp'. The table below lists the installed OWASP-related plugins.

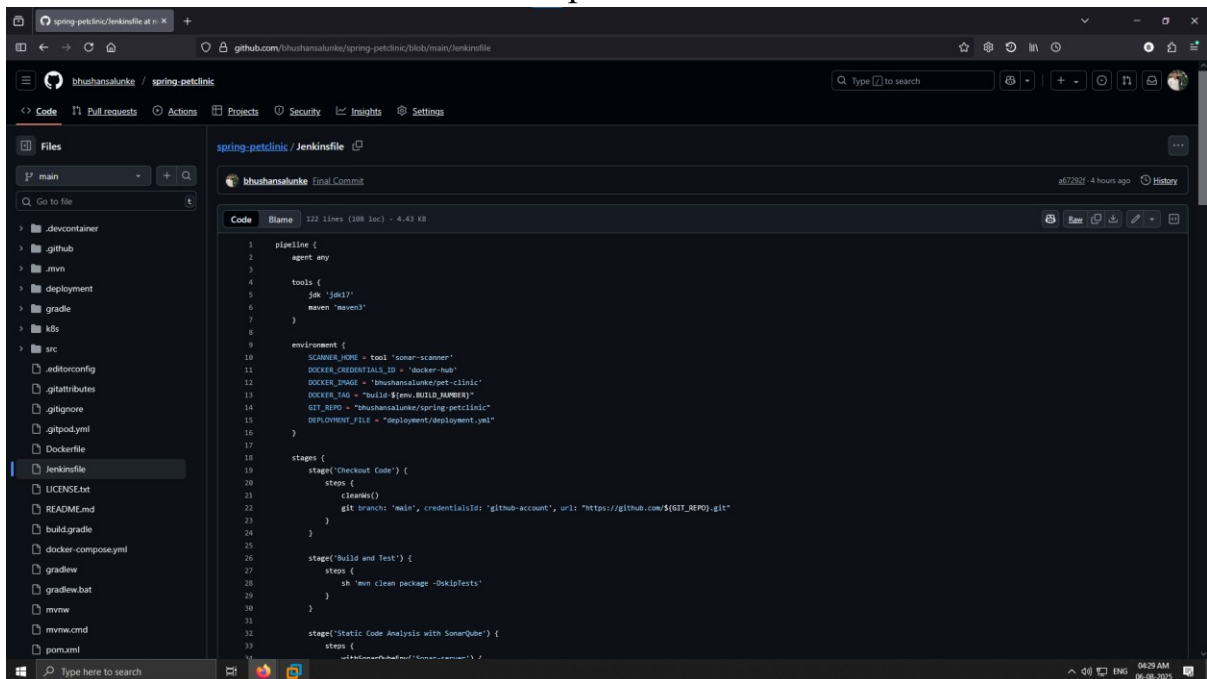
Name	Health	Enabled
OWASP Dependency-Check Plugin 5.6.1 This plug-in can independently execute a Dependency-Check analysis and visualize results. Dependency-Check is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities. Report an issue with this plugin	100	<input checked="" type="checkbox"/>
OWASP Markup Formatter Plugin 173.v680e3a_b_69f73 Uses the OWASP Java HTML Sanitizer to allow safe-seeming HTML markup to be entered in project descriptions and the like. Report an issue with this plugin	99	<input checked="" type="checkbox"/>
Timestampmer 1.30 Adds timestamps to the Console Output Report an issue with this plugin	100	<input checked="" type="checkbox"/>

REST API Jenkins 2.516.1

SonarQube Plugins



Pipeline



```

12 stage('Static Code Analysis with SonarQube') {
13     steps {
14         withSonarQubeEnv('Sonar-server') {
15             sh '''
16                 ${SCANNER_HOME}/bin/sonar-scanner \
17                 -Dsonar.projectKey=petclinic \
18                 -Dsonar.projectName=petclinic \
19                 -Dsonar.sources=. \
20                 -Dsonar.java.binaries=target/classes
21             '''
22         }
23     }
24 }
25
26 stage('Dependency Vulnerability Scan') {
27     steps {
28         withCredentials([string(credentialsId: 'mud-api-key', variable: 'MUD_API_KEY')]) {
29             dependencyCheckAdditionalArguments: '--scan --format HTML --out . --mdapikey "${MUD_API_KEY}" --mdValidForHours 87600 --data /var/lib/jenkins/odc-data',
30             odcInstallations: 'DP-check'
31         }
32         dependencyCheckPublisher pattern: '**/dependency-check-report.html'
33     }
34 }
35
36 stage('Docker Build and Push') {
37     steps {
38         withCredentials([usernamePassword(credentialsId: 'DOCKER_CREDENTIALS_ID', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
39             sh '''
40                 echo "${DOCKER_PASS}" | docker login -u "${DOCKER_USER}" --password-stdin
41                 docker build -t ${DOCKER_IMAGE}:${DOCKER_TAG} .
42                 docker push ${DOCKER_IMAGE}:${DOCKER_TAG}
43                 docker logout
44             '''
45         }
46     }
47 }
48
49 stage('Container Vulnerability Scan - Trivy') {
50     steps {
51         sh '''
52             trivy image --exit-code 0 --severity MEDIUM,CRITICAL ${DOCKER_IMAGE}:${DOCKER_TAG} > trivy-report.txt
53             archiveArtifacts artifacts: 'trivy-report.txt', allowEmptyArchive: true
54         '''
55     }
56 }

```

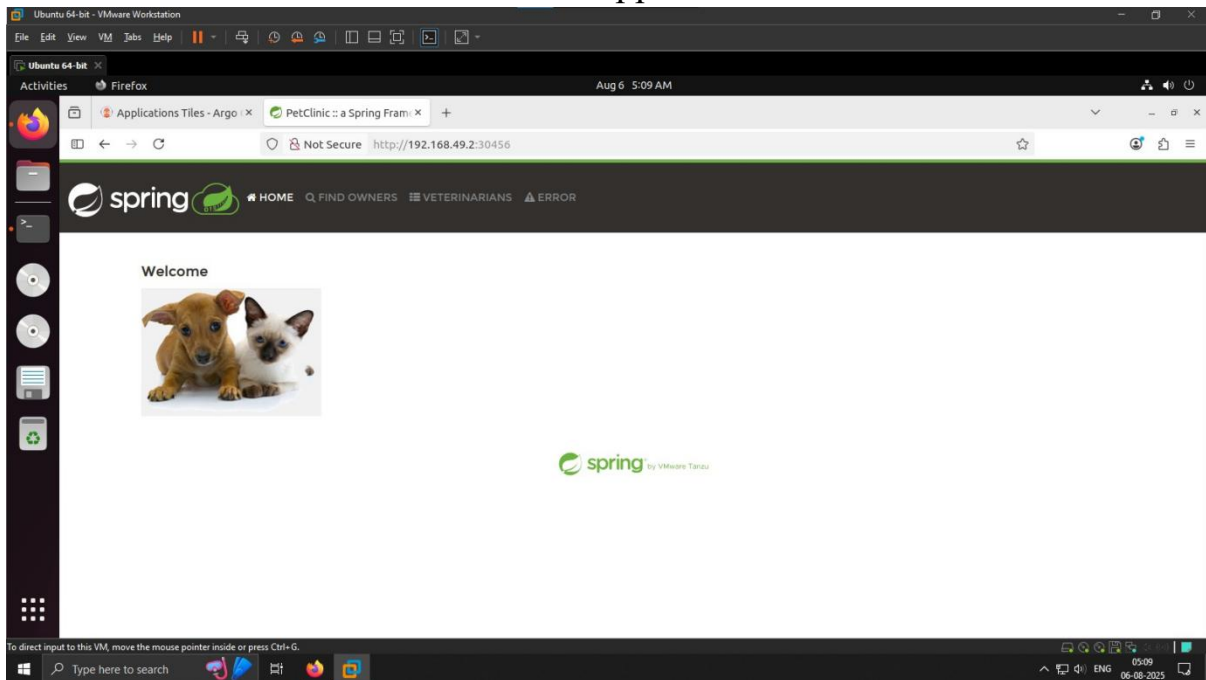
```

73     archiveArtifacts artifacts: 'trivy-report.txt', allowEmptyArchive: true
74 }
75
76 }
77
78
79 stage('Update Deployment File') {
80     steps {
81         withCredentials([string(credentialsId: 'github-token', variable: 'GITHUB_TOKEN')]) {
82             sh '''
83                 git config --global user.email "bhushansalunke5@gmail.com"
84                 git config --global user.name "bhushansalunke"
85                 sed -i -e "s|image: ${DOCKER_IMAGE}:${DOCKER_TAG}|${DEPLOYMENT_FILE}|g"
86                 git add ${DEPLOYMENT_FILE}
87                 git commit -m "Update deployment image to ${DOCKER_TAG} || echo 'No changes to commit'"
88                 git push https://$GITHUB_TOKEN@github.com:$GITHUB_REPO.git HEAD:main
89             '''
90         }
91     }
92 }
93
94 }
95
96
97 post {
98     success {
99         email {
100             subject: "SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
101             body: """opsBuild succeeded./p
102             <a href="${env.BUILD_URL}">View Build</a>/p
103             <p>See attached Dependency-Check and Trivy reports.</p>""",
104             mimeType: 'text/html',
105             to: 'bhushansalunke5@gmail.com',
106             attachmentsPattern: '**/dependency-check-report.html, **/trivy-report.txt'
107         }
108     }
109
110     failure {
111         email {
112             subject: "FAILURE: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
113             body: """opsBuild failed.</p>
114             <a href="${env.BUILD_URL}">View Build</a>/p
115             <p>See attached reports for error details.</p>""",

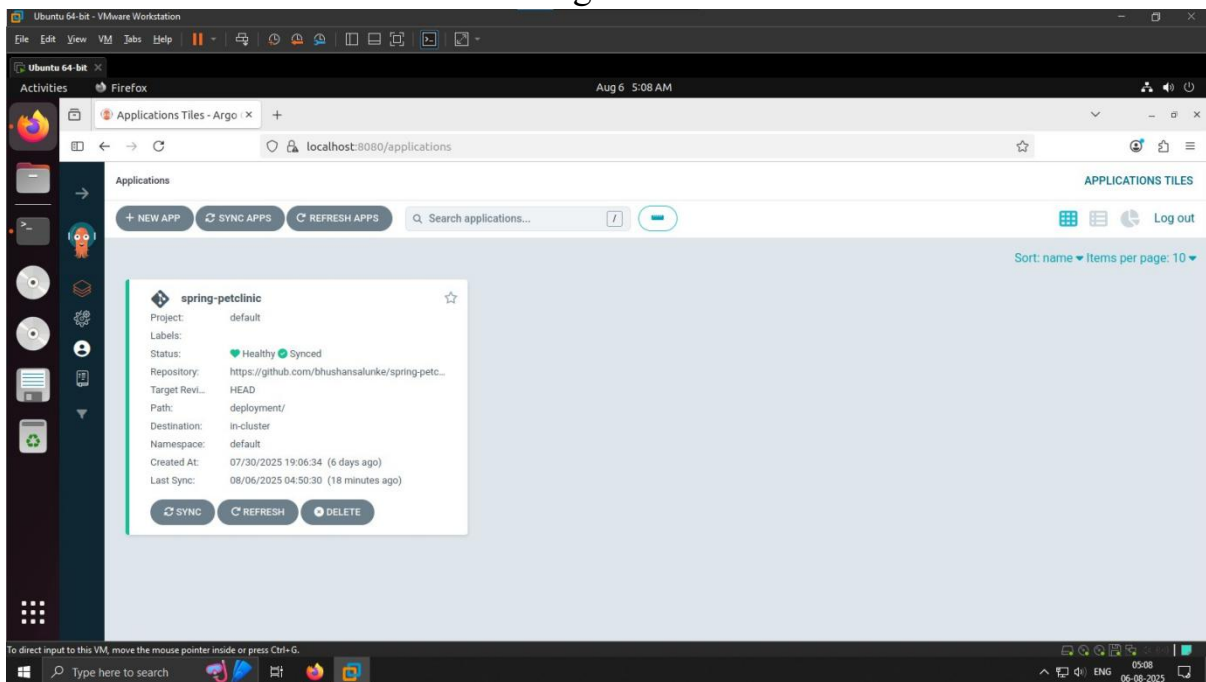
```



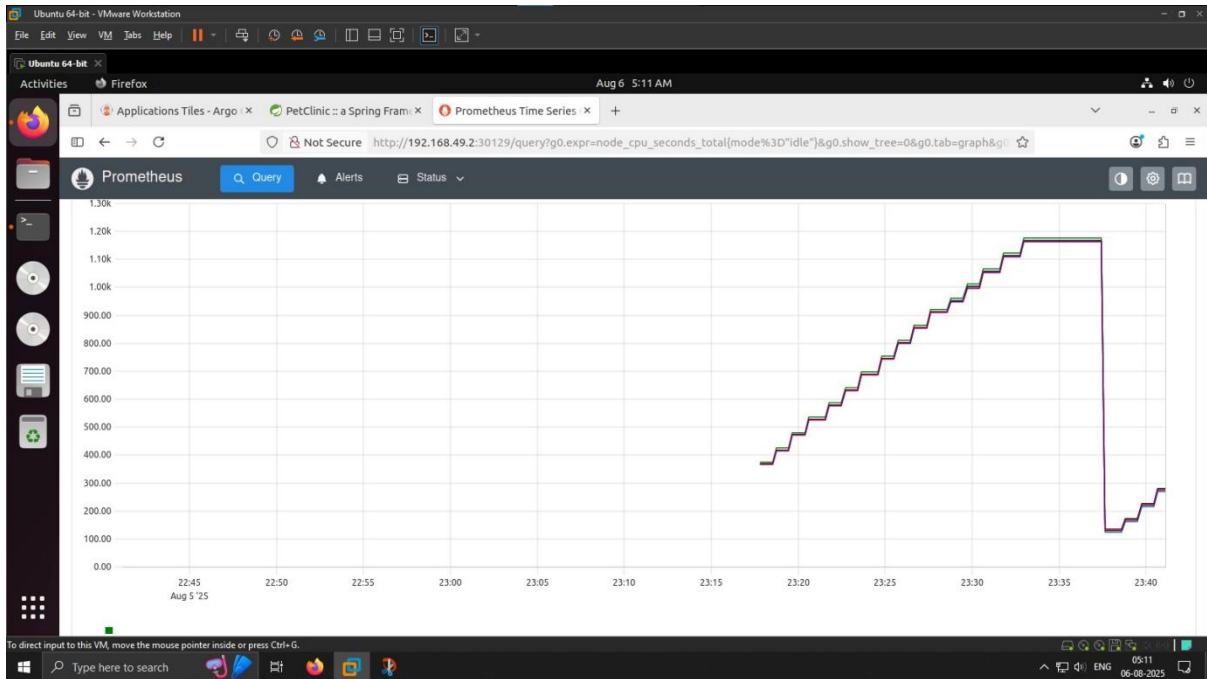

Pet-Clinic Application



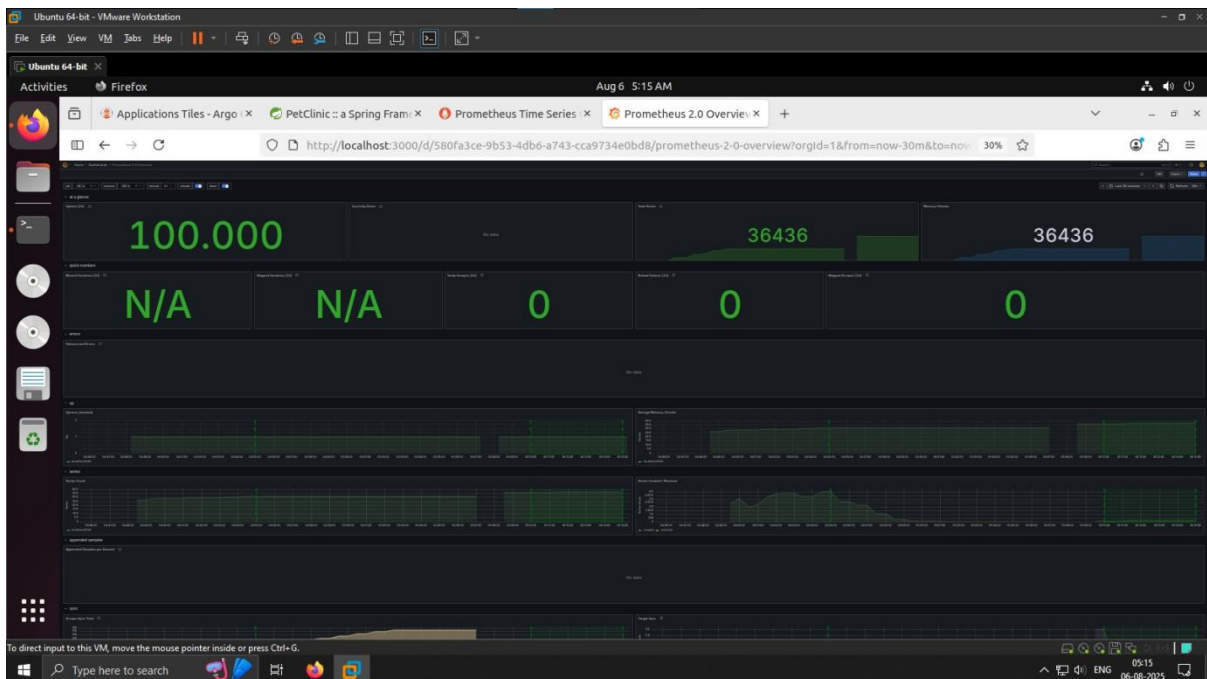
ArgoCD



Prometheus



Grafana



CONCLUSION

In this project, we successfully designed and implemented a fully automated **Continuous Integration and Continuous Deployment (CI/CD)** pipeline tailored for a **Tier 3 application** using **Jenkins**, **ArgoCD**, and **Kubernetes** to deploy a **Spring Boot microservice**. By integrating essential tools such as **SonarQube** for static code analysis, **OWASP Dependency-Check** for security vulnerability scanning, and **Docker** for containerization, we streamlined the software development process, ensuring that only high-quality, secure code is deployed.

The use of **ArgoCD** facilitated a seamless **GitOps** deployment process, where every change in the Git repository was automatically reflected in the Kubernetes cluster. This ensured that the desired state of the application was always in sync with the Git repository, significantly reducing the complexity of managing deployments and enhancing the overall workflow's efficiency and reliability.

Additionally, we incorporated **Prometheus** for real-time monitoring and **Grafana** for data visualization, allowing us to monitor the health and performance of the application in production. This proactive monitoring enabled timely issue resolution, ensuring high availability and performance of the deployed microservice. The integration of email notifications further enhanced team communication, ensuring that any issues during the build, deployment, or monitoring processes were quickly addressed.

This project has successfully demonstrated the power of automation, security, and observability in modern software development practices. By adopting a **DevSecOps** mindset, we integrated security checks directly into the pipeline, ensuring that vulnerabilities were detected and addressed before reaching production. This approach not only improved the overall security posture of the application but also aligned with industry best practices for secure software delivery.

Overall, this automated CI/CD pipeline has streamlined development, improved code quality, enhanced security, and enabled efficient and reliable deployments, making it a robust solution for modern software development teams. The combination of these tools has empowered the team to focus on innovation while ensuring that the deployment process remains seamless and scalable.

FUTURE SCOPE

Kube-bench Integration

Use Kube-bench to check Kubernetes cluster security as per CIS benchmarks.

Email & SMS Alerts

Set up alerts for build failures and vulnerabilities for faster response.

Cloud Kubernetes (EKS/GKE)

Shift from Minikube to cloud-based Kubernetes for better scalability and performance.

RBAC Implementation

Use Role-Based Access Control to allow only authorized access to cluster resources.

Secret Management

Securely manage secrets like API keys using tools like HashiCorp Vault.

Auto Rollbacks & Blue-Green Deployment

Enable safe deployments with blue-green strategy and automatic rollback on failures.

REFERENCES

1. <https://docs.github.com/en>
2. <https://www.jenkins.io/doc/>
3. <https://docs.docker.com/docker-hub>
4. <https://owasp.org/site-documentation/>
5. <https://docs.sonarsource.com/sonarqube-server/9.9/>
6. <https://prometheus.io/docs/>
7. <https://grafana.com/docs/>
8. <https://argo-cd.readthedocs.io/>