



**INSTITUTE FOR ADVANCED COMPUTING**

**AND**

**SOFTWARE DEVELOPMENT,**

**AKURDI, PUNE**

DOCUMENTATION ON

# " A PRODUCTION-GRADE DEVSECOPS KUBERNETES DEPLOYMENT WITH CONTINUOUS SECURITY MONITORING"

PG-DITISS August 2025

SUBMITTED BY:

**GROUP NO: 21**

**PRATIK MUTHAL (258426)**

**TEJAS GAYKHE (258442)**

**MRS. SUSHMA HATTARKI**

**PROJECT GUIDE**

**MR. ANIL SHARMA**

**CENTRE CO-ORDINATOR**

# ABSTRACT

In recent years, cloud-native technologies and DevOps automation have transformed the way modern applications are developed, deployed, and monitored. With the rapid growth of streaming platforms such as Disney+ Hotstar, scalable and secure deployment architectures are required to handle millions of concurrent users. This project presents the design and implementation of a Disney+ Hotstar Clone Application deployed on Kubernetes with an integrated DevSecOps CI/CD pipeline and monitoring framework.

The proposed system leverages Docker containerization, Kubernetes orchestration, Jenkins automation, SonarQube code quality analysis, Trivy vulnerability scanning, OWASP Dependency Check, Terraform infrastructure provisioning, and Prometheus–Grafana monitoring. The CI/CD pipeline automates the entire software delivery lifecycle, including code build, security testing, containerization, and deployment. Kubernetes ensures scalability, high availability, and self-healing capabilities.

The monitoring module provides real-time metrics, alerting, and visualization to ensure system reliability. The project demonstrates a production-grade DevSecOps pipeline with infrastructure automation and cloud-native deployment, serving as a blueprint for enterprise-level cloud application delivery.

# TABLE OF CONTENTS

# LIST OF ABBREVIATIONS

| Sr.no | Abbreviation | Full Form |
|---|---|---|
| 1 | **CI/CD** | Continuous Integration / Continuous Deployment |
| 2 | **DevOps** | Development and Operations |
| 3 | **DevSecOps** | Development, Security, and Operations |
| 4 | **VM** | Virtual Machine |
| 5 | **EC2** | Elastic Compute Cloud |
| 6 | **AWS** | Amazon Web Services |
| 7 | **EKS** | Elastic Kubernetes Service |
| 8 | **API** | Application Programming Interface |
| 9 | **YAML** | Yet Another Markup Language |
| 10 | **IaC** | Infrastructure as Code |
| 11 | **CLI** | Command Line Interface |
| 12 | **SSH** | Secure Shell |
| 13 | **SSL** | Secure Sockets Layer |
| 14 | **TLS** | Transport Layer Security |

| Sr.no | Abbreviation | Full Form |
|---|---|---|
| 15 | **HTTP** | Hypertext Transfer Protocol |
| 16 | **HTTPS** | Hypertext Transfer Protocol Secure |
| 17 | **SAST** | Static Application Security Testing |
| 18 | **DAST** | Dynamic Application Security Testing |
| 19 | **CVE** | Common Vulnerabilities and Exposures |
| 20 | **NVD** | National Vulnerability Database |
| 21 | **CVSS** | Common Vulnerability Scoring System |
| 22 | **K8s** | Kubernetes |
| 23 | **LB** | Load Balancer |
| 24 | **GUI** | Graphical User Interface |
| 25 | **IP** | Internet Protocol |
| 26 | **DNS** | Domain Name System |
| 27 | **SMTP** | Simple Mail Transfer Protocol |
| 28 | **CDN** | Content Delivery Network |
| 29 | **RBAC** | Role-Based Access Control |

| Sr.no | Abbreviation | Full Form |
|---|---|---|
| 30 | **JSON** | JavaScript Object Notation |
| 31 | **UI** | User Interface |
| 32 | **UX** | User Experience |
| 33 | **PoC** | Proof of Concept |
| 34 | **IDE** | Integrated Development Environment |
| 35 | **SaaS** | Software as a Service |
| 36 | **PaaS** | Platform as a Service |
| 37 | **IaaS** | Infrastructure as a Service |

# LIST OF FIGURES

# 1. INTRODUCTION

The rapid adoption of cloud computing and microservices architecture has significantly increased the complexity of deploying and managing large-scale applications. Streaming platforms like Disney+ Hotstar require high scalability, fault tolerance, and security to provide uninterrupted service to users worldwide.

Traditional deployment methods are inefficient, manual, and error-prone. DevOps and cloud-native technologies provide automation, scalability, and continuous integration and delivery mechanisms to overcome these limitations.

This project focuses on deploying a Disney+ Hotstar Clone Application using modern DevOps practices and Kubernetes orchestration. The system integrates CI/CD automation, security testing, containerization, cloud deployment, and monitoring to achieve a scalable and secure application environment.

## 1.1 Problem Statement

Organizations face several challenges in deploying modern applications:

1. Manual deployment processes lead to configuration errors and downtime.
2. Lack of automated security testing increases vulnerability risks.
3. Scaling applications manually is inefficient and costly.
4. Monitoring distributed systems is complex without centralized tools.
5. Infrastructure provisioning is time-consuming without automation.

The problem addressed in this project is to design an automated, scalable, and secure deployment architecture using DevSecOps principles and Kubernetes orchestration.

# 2. LITERATURE SURVEY

Modern application deployment has evolved significantly from traditional monolithic architectures to microservices-based and containerized environments. In earlier systems, applications were deployed on physical servers or virtual machines, which resulted in scalability issues, high maintenance costs, and slow deployment cycles. With the introduction of cloud computing and DevOps practices, organizations are now adopting automated deployment pipelines and container orchestration platforms to improve efficiency and reliability.

Docker is one of the most popular containerization platforms used to package applications and their dependencies into lightweight, portable containers. It ensures consistency across different environments, such as development, testing, and production. Kubernetes is an advanced container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides features such as self-healing, automatic scaling, rolling updates, and service discovery, making it suitable for large-scale cloud-native applications.

Continuous Integration and Continuous Deployment (CI/CD) pipelines play a crucial role in modern software development. Jenkins is a widely used open-source automation server that helps in building, testing, and deploying applications automatically. It supports integration with various tools and plugins, enabling organizations to implement DevOps and DevSecOps practices effectively.

Code quality and security analysis are essential components of modern software development. SonarQube is a static code analysis tool that identifies bugs, vulnerabilities, and code smells in source code. Trivy is a vulnerability scanning tool that detects security issues in container images, file systems, and Git repositories. OWASP Dependency Check is used to identify known vulnerabilities in third-party libraries and dependencies by comparing them with public vulnerability databases.

Infrastructure automation is another key area in cloud-native deployments. Terraform is an infrastructure-as-code (IaC) tool that allows users to define and provision cloud infrastructure using configuration files. It helps in automating the creation of virtual machines, Kubernetes clusters, networking components, and storage resources in cloud environments such as AWS, Azure, and Google Cloud.

Monitoring and logging are critical for maintaining the reliability and performance of deployed applications. Prometheus is an open-source monitoring system that collects metrics from applications and infrastructure components. Grafana is a visualization tool that provides interactive dashboards to analyze metrics and system performance. Together, Prometheus and Grafana help administrators monitor system health, detect anomalies, and optimize resource utilization.

Recent research studies emphasize the importance of integrating security into the DevOps lifecycle, known as DevSecOps. Integrating security tools into CI/CD pipelines helps detect vulnerabilities early in the development process, reducing the risk of security breaches and deployment failures. Kubernetes-based deployments have been proven to offer better scalability, fault tolerance, and resource management compared to traditional virtual machine-based deployments.

Several studies also highlight that microservices and container orchestration platforms significantly improve system availability and reduce downtime. Cloud-native architectures enable organizations to deploy applications faster, improve collaboration between development and operations teams, and enhance overall system security and reliability.

# 3. METHODOLOGY

## 3.1 SYSTEM ARCHITECTURE:

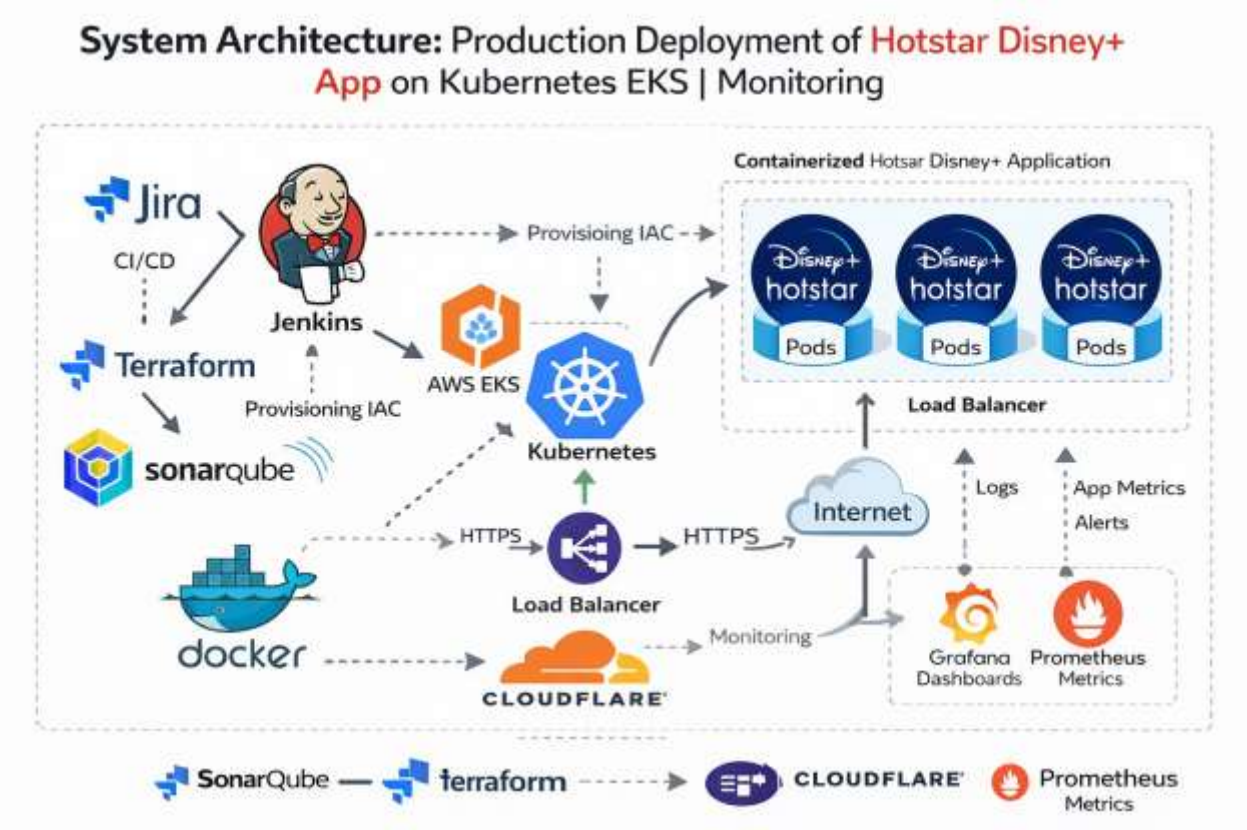



**Figure 1: Framework Diagram**

The methodology of this project focuses on designing, developing, deploying, and monitoring a cloud-native web application using DevSecOps practices and Kubernetes orchestration. The project follows a structured approach starting from project planning and system design, followed by source code management, automation, containerization, cloud deployment, and monitoring. Initially, the project requirements were analyzed and an overall system architecture was designed to integrate various DevOps and cloud tools. The application source code was managed using Git and GitHub to maintain version control and ensure collaboration and traceability of changes.

An automated CI/CD pipeline was implemented using Jenkins to streamline the build, test, and deployment processes. The pipeline automatically retrieves source code, performs static code analysis, scans dependencies and container images for vulnerabilities, builds Docker images, and deploys the application to the Kubernetes cluster. DevSecOps practices were integrated into the pipeline by incorporating security tools such as SonarQube, OWASP Dependency Check, and Trivy to identify security vulnerabilities at different stages of the software development lifecycle.

Docker was used to containerize the Disney+ Hotstar web application, ensuring portability and consistency across different environments. The containerized application was deployed on Amazon Elastic Kubernetes Service (EKS), where Kubernetes manifests were used to manage deployments, services, and ingress configurations. Kubernetes provides features such as auto-scaling, self-healing, and load balancing, which enhance the reliability and availability of the application.

Infrastructure provisioning was automated using Terraform, which enabled the creation and management of cloud resources such as virtual machines, networking components, and Kubernetes clusters through configuration files. Monitoring was implemented using Prometheus to collect system and application metrics, while Grafana was used to visualize these metrics through dashboards. Finally, the deployed system was tested to validate the functionality of the CI/CD pipeline, security mechanisms, Kubernetes deployment, and monitoring setup, ensuring that the project meets the defined objectives and performs efficiently in a production-like environment.

# 4. REQUIREMENT SPECIFICATION

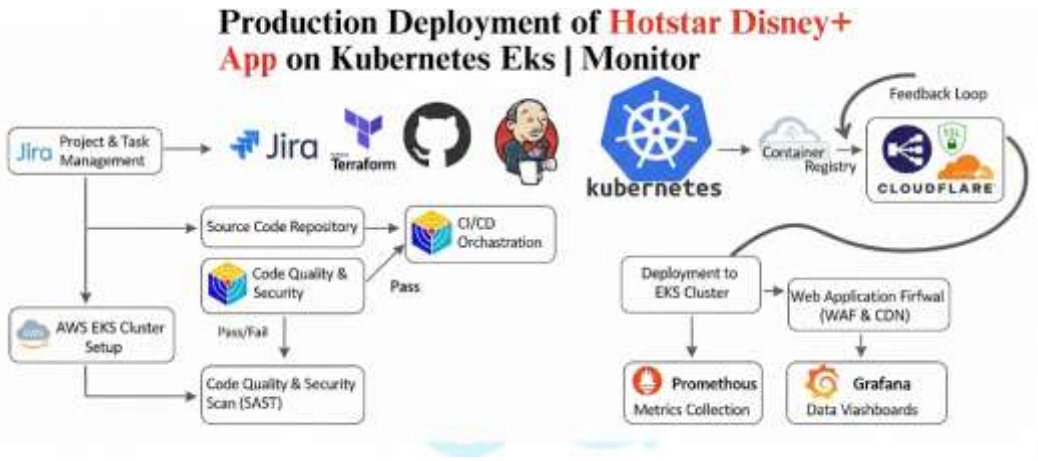
## 4.1 SOFTWARE REQUIREMENTS

a. **Operating System (OS):** Ubuntu 20.04/22.04 LTS, Windows 10/11, or Linux distributions
b. **Languages:** HTML, CSS, JavaScript, Bash scripting, YAML
c. **DevOps Tools:** Jenkins, Git, GitHub, Terraform
d. **Containerization & Orchestration:** Docker, Kubernetes, Amazon EKS
e. **Security & Code Analysis Tools:** SonarQube, OWASP Dependency Check, Trivy
f. **Monitoring Tools:** Prometheus, Grafana
g. **Cloud Platform:** Amazon Web Services (AWS)
h. **Other Tools:** Nginx, Cloudflare, SSH Client, Web Browser

## 4.2 HARDWARE REQUIREMENTS

a. **Processor:** Quad-core 2.5 GHz minimum / 8-core recommended
b. **RAM:** 8 GB minimum / 16 GB recommended for smooth Kubernetes and Docker operations
c. **Storage:** 50 GB SSD minimum (Docker images, logs, tools)
d. **Network:** Stable high-speed internet connection for cloud access and CI/CD operations
e. **Cloud Resources:** AWS EC2 instances, Kubernetes cluster nodes, and storage volumes

# 5.WORKING




## Phase 1: Requirement Analysis and System Design

In the first phase, the project requirements are analyzed to understand the tools, infrastructure, and deployment environment. The system architecture is designed by selecting DevOps tools such as GitHub, Jenkins, Docker, Kubernetes, Terraform, SonarQube, Trivy, Prometheus, and Grafana. This phase focuses on planning the CI/CD pipeline and cloud infrastructure layout.

## Phase 2: Source Code Development and Version Control

In this phase, the application source code is developed and stored in a GitHub repository. Git is used for version control to track changes and manage different versions of the project. This ensures collaboration and rollback capability in case of failures.

## Phase 3: CI/CD Pipeline Implementation

In this phase, Jenkins is configured to automate the build and deployment process. Jenkins fetches the source code from GitHub and executes pipeline stages such as build, test, and deployment. Automation reduces manual errors and speeds up the development lifecycle.

## Phase 4: Security Integration (DevSecOps Phase)

In this phase, security tools are integrated into the pipeline. SonarQube is used for static code analysis, OWASP Dependency Check scans third-party libraries, and Trivy scans Docker images for vulnerabilities. This phase ensures that security issues are detected early in the development lifecycle.

## Phase 5: Containerization Using Docker

In this phase, the application is containerized using Docker. A Dockerfile is created to package the application and its dependencies into a container image. The Docker image is pushed to a container registry for deployment.

## Phase 6: Kubernetes Deployment

In this phase, the containerized application is deployed on a Kubernetes cluster using Amazon EKS. Kubernetes manifests such as Deployment, Service, and Ingress are used to manage pods and expose the application to users. Kubernetes provides auto-scaling, load balancing, and self-healing capabilities.

## Phase 7: Infrastructure Automation Using Terraform

In this phase, Terraform is used to provision and manage cloud infrastructure automatically. Terraform scripts create virtual machines, networking components, and Kubernetes clusters. This phase ensures infrastructure consistency and repeatability.

## Phase 8: Monitoring and Visualization

In this phase, Prometheus is configured to collect metrics from Kubernetes and application pods. Grafana is used to visualize these metrics using dashboards. This phase helps in real-time monitoring and performance analysis of the deployed system.

## Phase 9: Testing and Validation

In the final phase, the system is tested to verify pipeline execution, security scanning, Kubernetes deployment, and monitoring dashboards. The application is accessed through a web browser to confirm successful deployment and system functionality.

# 6. IMPLEMENTATION

The application source code was stored in GitHub and managed using Git for version control. Jenkins was configured to create an automated CI/CD pipeline for building and deploying the application. Docker was used to containerize the application and push the image to a container registry. The containerized application was deployed on a Kubernetes cluster using Amazon EKS for scalability and reliability. Prometheus and Grafana were implemented to monitor system performance and visualize metrics in real time.

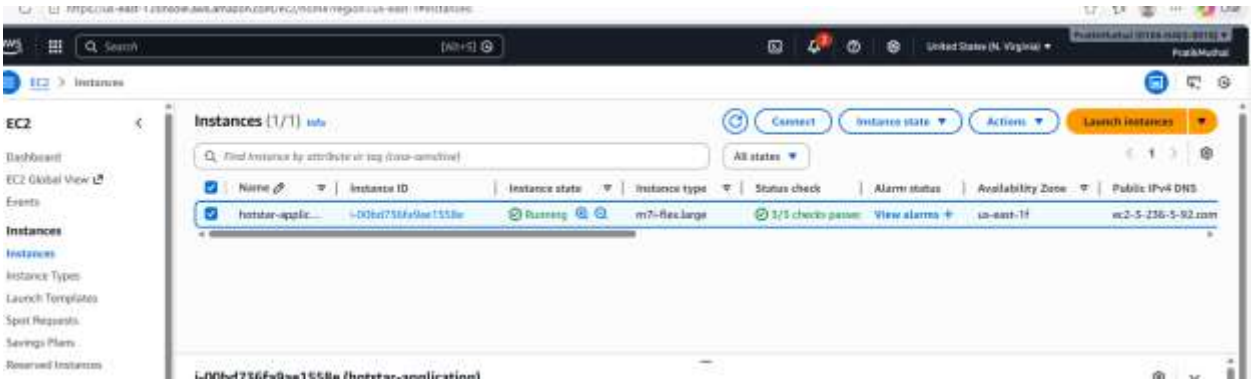
```
Disney-Hotstar-Kubernetes-Project/
│
├── src/                    # Application source code
│   ├── index.html
│   ├── style.css
│   ├── script.js
│
├── Docker/                 # Docker files
│   ├── Dockerfile
│   └── docker-compose.yml
│
├── Jenkins/                # CI/CD pipeline files
│   └── Jenkinsfile
│
├── Kubernetes/             # Kubernetes deployment files
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── ingress.yaml
```

```
│
├── Terraform/                # Infrastructure as Code scripts
│   ├── main.tf
│   ├── variables.tf
│   └── outputs.tf
│
├── Monitoring/               # Monitoring configurations
│   ├── prometheus.yaml
│   └── grafana-dashboard.json
│
├── Security/                 # DevSecOps tools configuration
│   ├── sonar-project.properties
│   └── dependency-check.sh
│
├── Screenshots/              # Project result screenshots
│
├── Report/                   # Project documentation
│   └── Final_Project_Report.docx
│
└── README.md                 # Project description
```

## 6.1 Environment:

### 1. Configure Infrastructure In AWS Cloud :

- **Launch an EC2 Instance Ubuntu (22.04) T3 X micro Instance.**

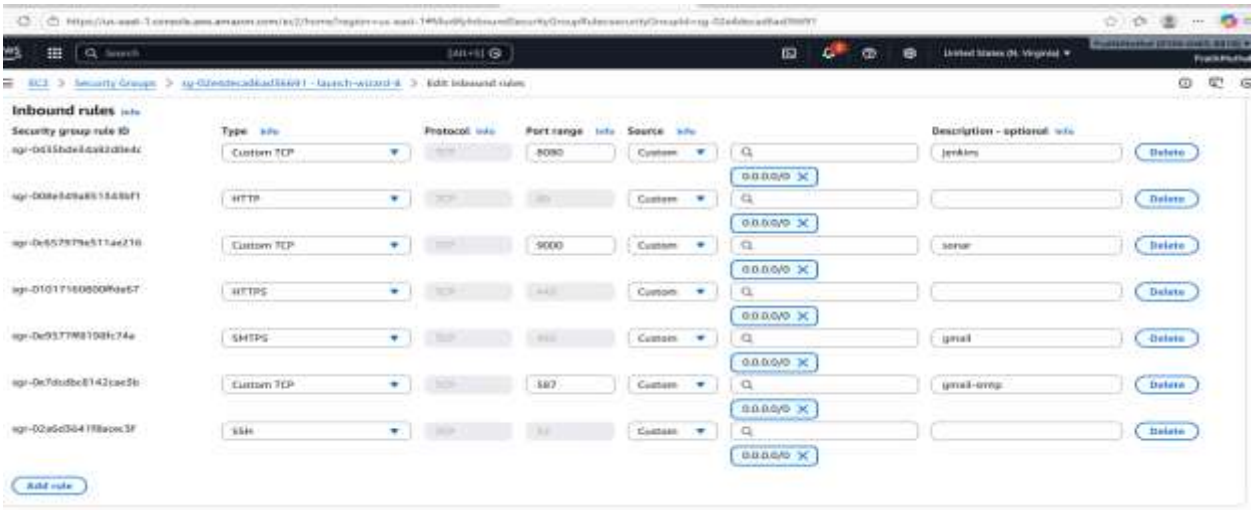



### 2.Configure Security Group :

- **Create or modify a security group to allow the following ports:**

| Port | Protocol | Description |
|---|---|---|
| 22 | TCP | SSH (for remote access) |
| 80 | TCP | HTTP (Web traffic) |
| 443 | TCP | HTTPS (Secure web traffic) |
| 8080 | TCP | Web applications (Tomcat, etc.) |
| 587 | TCP | SMTP (Email sending) |
| 465 | TCP | SMTP over SSL |
| 3000 | TCP | Web apps (Grafana, Node.js, etc.) |
| 9000 | TCP | SonarQube / Web applications |

Set the Source to Anywhere (0.0.0.0/0) unless you want to restrict access



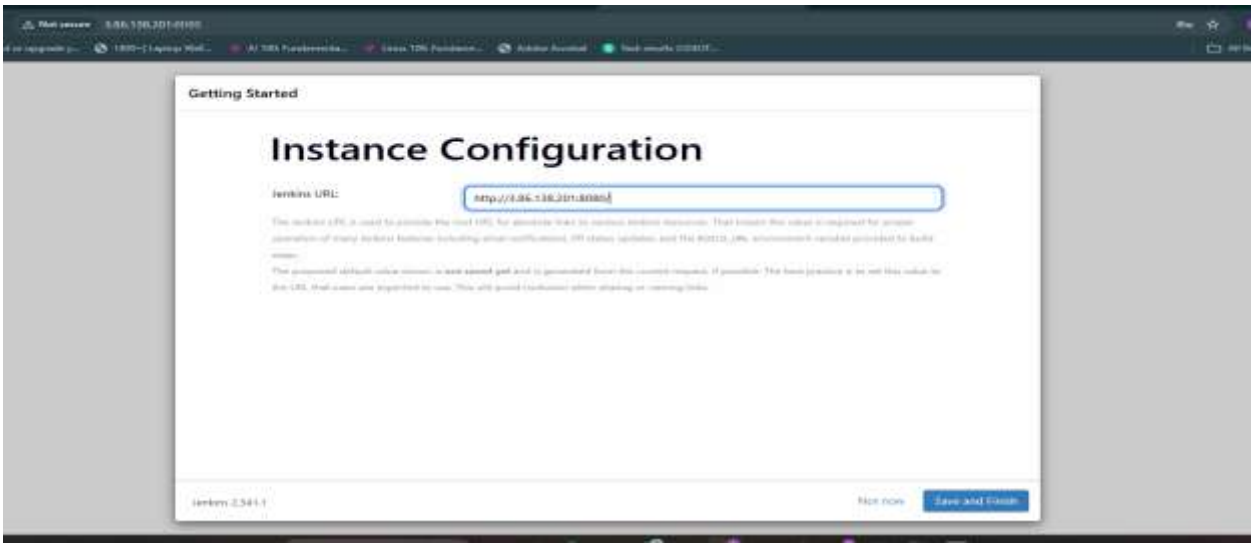

**3.Install Jenkins, Docker, awscli, terraform, kubectl, eksctl and Trivy, Clone the GITHUB Project repositories :**

- **Install the TOOLS in the VM machine via Script, add executable permission to shell script chmod +x *.sh .**

```
echo "Making all .sh files executable..."
chmod +x *.sh

echo "Permissions updated successfully!"
ubuntu@ip-10-0-1-103:~/hotstar-kubernetes/scripts$ chmod +x *.sh
ubuntu@ip-10-0-1-103:~/hotstar-kubernetes/scripts$ ll
total 44
drwxrwxr-x 2 ubuntu ubuntu 4096 Mar 19 20:21 ./
drwxrwxr-x 8 ubuntu ubuntu 4096 Mar 19 20:21 ../
-rwxrwxr-x 1 ubuntu ubuntu  396 Mar 19 20:21 awscli.sh*
-rwxrwxr-x 1 ubuntu ubuntu  820 Mar 19 20:21 docker.sh*
-rwxrwxr-x 1 ubuntu ubuntu  393 Mar 19 20:21 eksctl.sh*
-rwxrwxr-x 1 ubuntu ubuntu  939 Mar 19 20:21 grafana.sh*
-rwxrwxr-x 1 ubuntu ubuntu  570 Mar 19 20:21 jenkins.sh*
-rwxrwxr-x 1 ubuntu ubuntu  460 Mar 19 20:21 kubectl.sh*
-rwxrwxr-x 1 ubuntu ubuntu  294 Mar 19 20:21 permissionexecute.sh*
-rwxrwxr-x 1 ubuntu ubuntu  846 Mar 19 20:21 terraform.sh*
-rwxrwxr-x 1 ubuntu ubuntu  637 Mar 19 20:21 trivy.sh*
ubuntu@ip-10-0-1-103:~/hotstar-kubernetes/scripts$ sh
```

## 6.2 Jenkins:

- Unlock Jenkins using an administrative password and install the suggested plugins. Retrieve the initial admin password:




**Install Plugins like JDK, SonarQube Scanner, NodeJs, OWASP Dependency Check**

1. Eclipse Temurin Installer (Install without restart)

2. SonarQube Scanner (Install without restart)

3. NodeJs Plugin (Install Without restart) – 16.20.2

4. OWASP Dependency Check Plugins

5. Stage view

6. jdk

Docker plugin

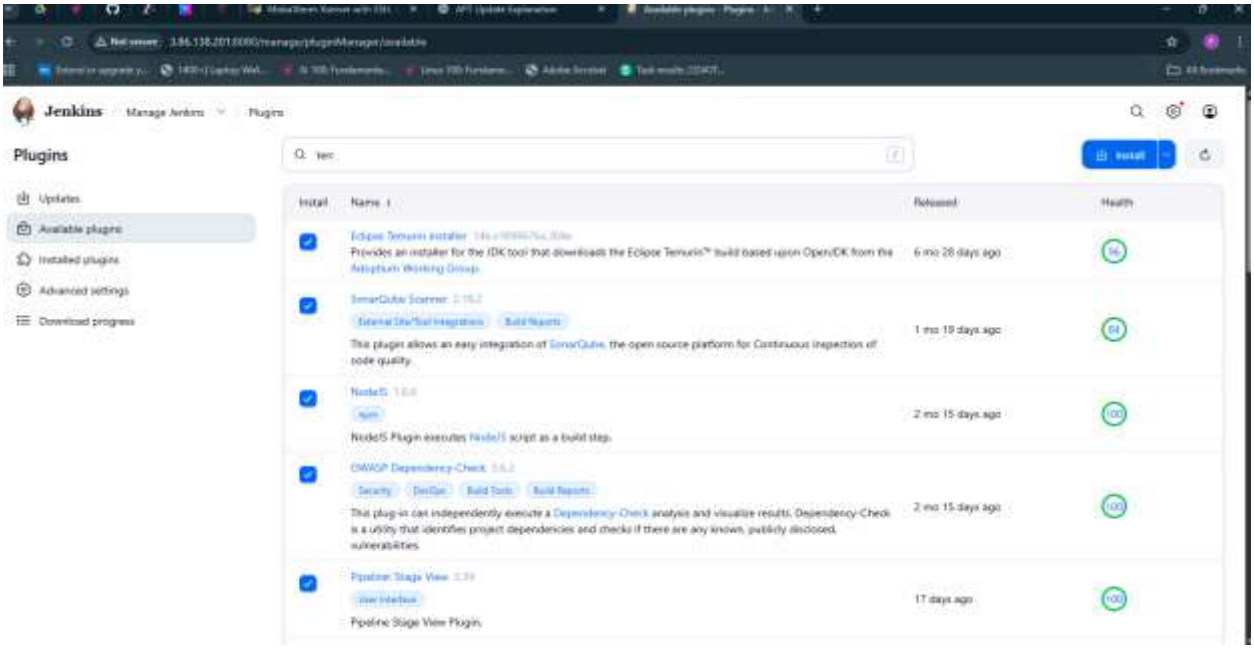
7. Docker

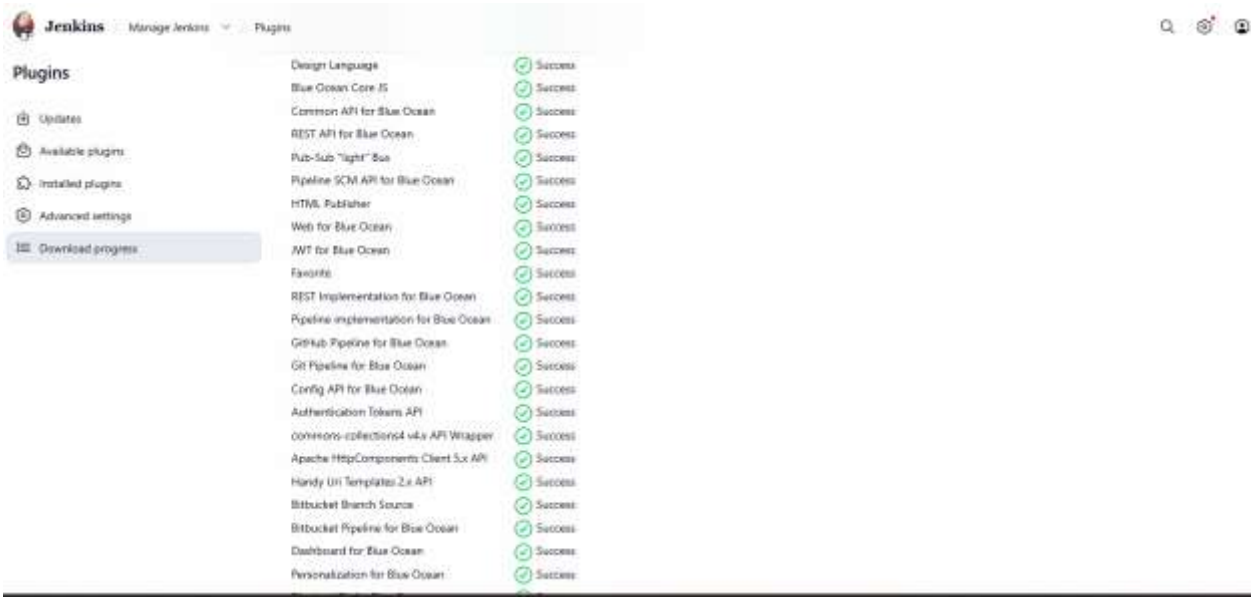
8. Docker Commons

9. Docker Pipeline

10. Docker API

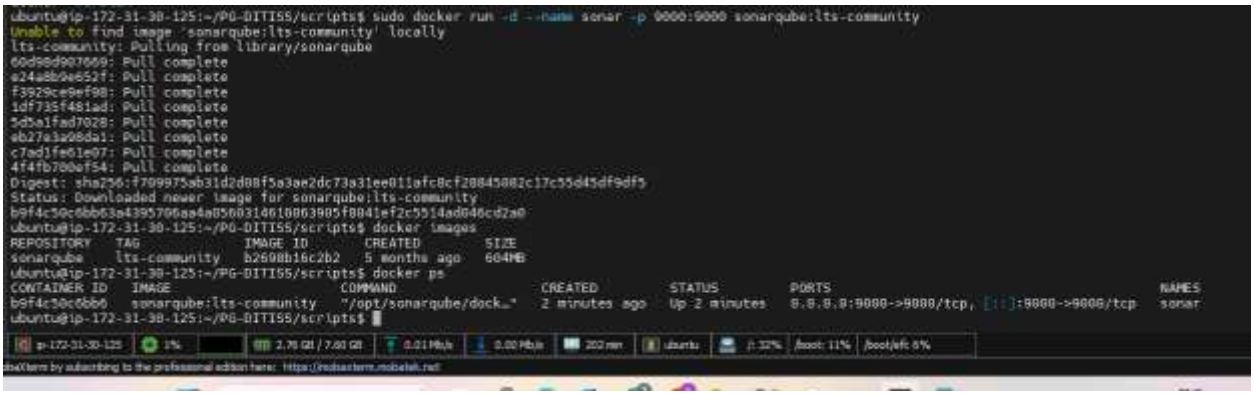
11. docker-build-step



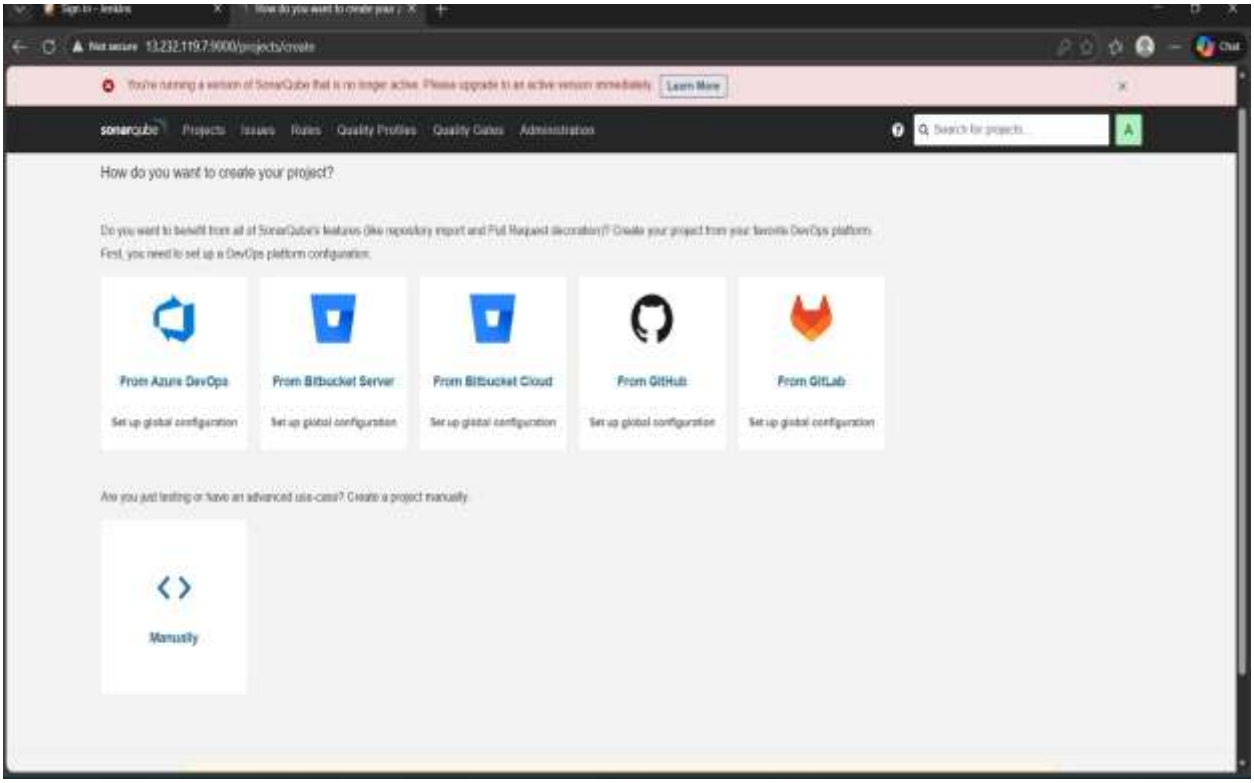




### 6.3 Setup SonarQube Server:

- we create a sonarqube container
- docker run -d --name sonar -p 9000:9000 sonarqube:lts-community-




**Sonarqube Dashboard:**

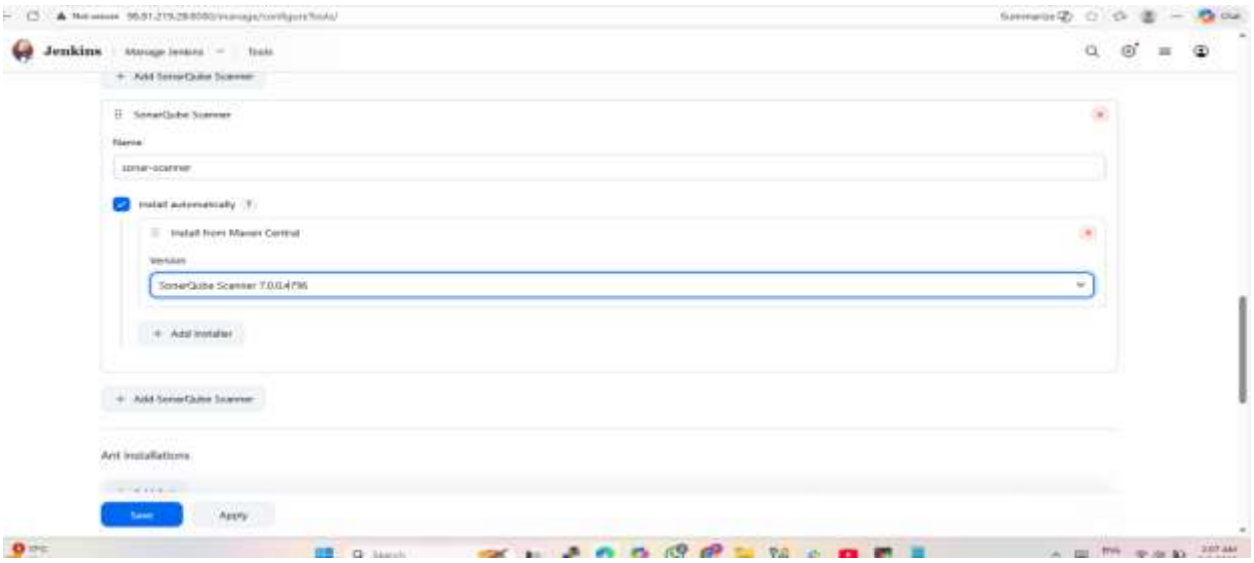



### 6.4 Add Credentials in Jenkins:

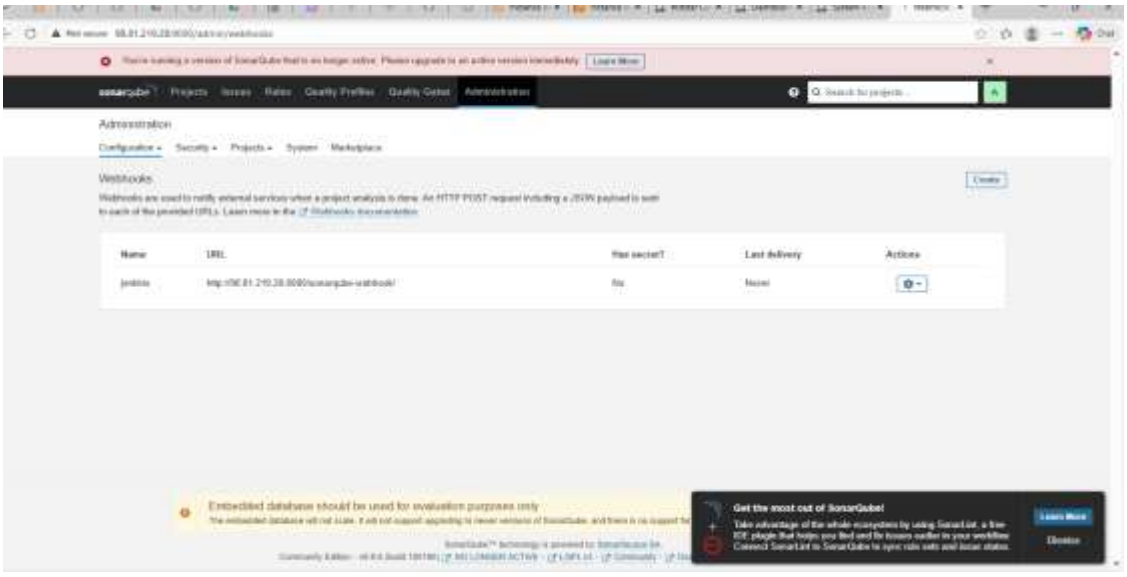
- The Configure System option is used in Jenkins to configure different server

  Global Tool Configuration is used to configure different tools that we install using Plugins

  We will install a sonar scanner in the tools.

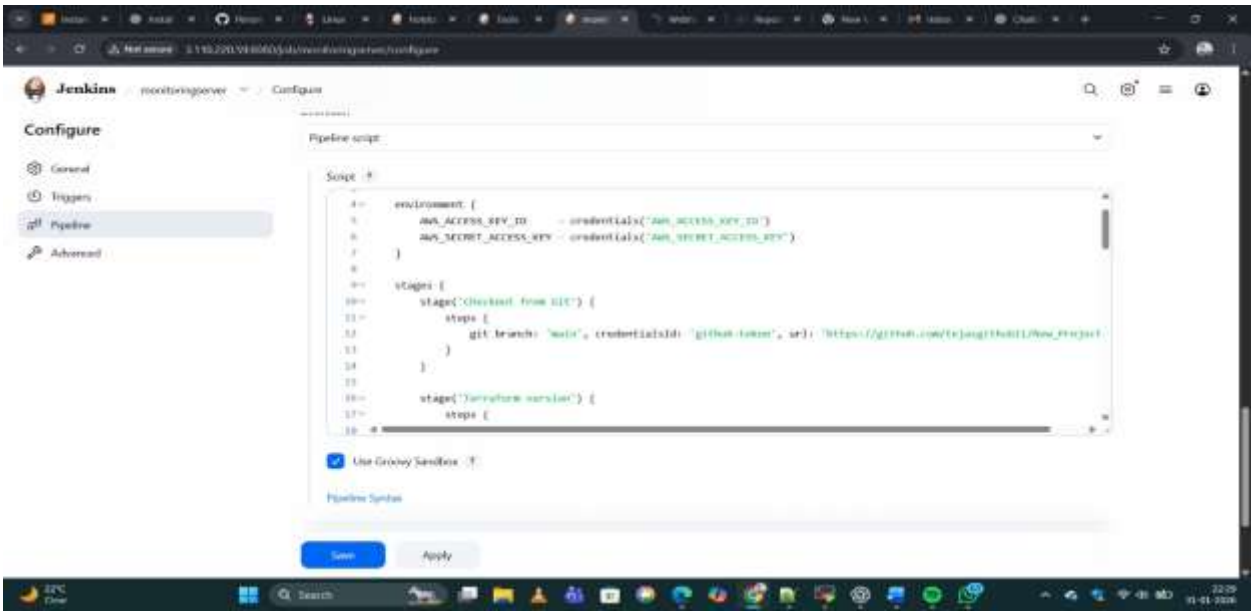



- In the Sonarqube Dashboard add a quality gate also:

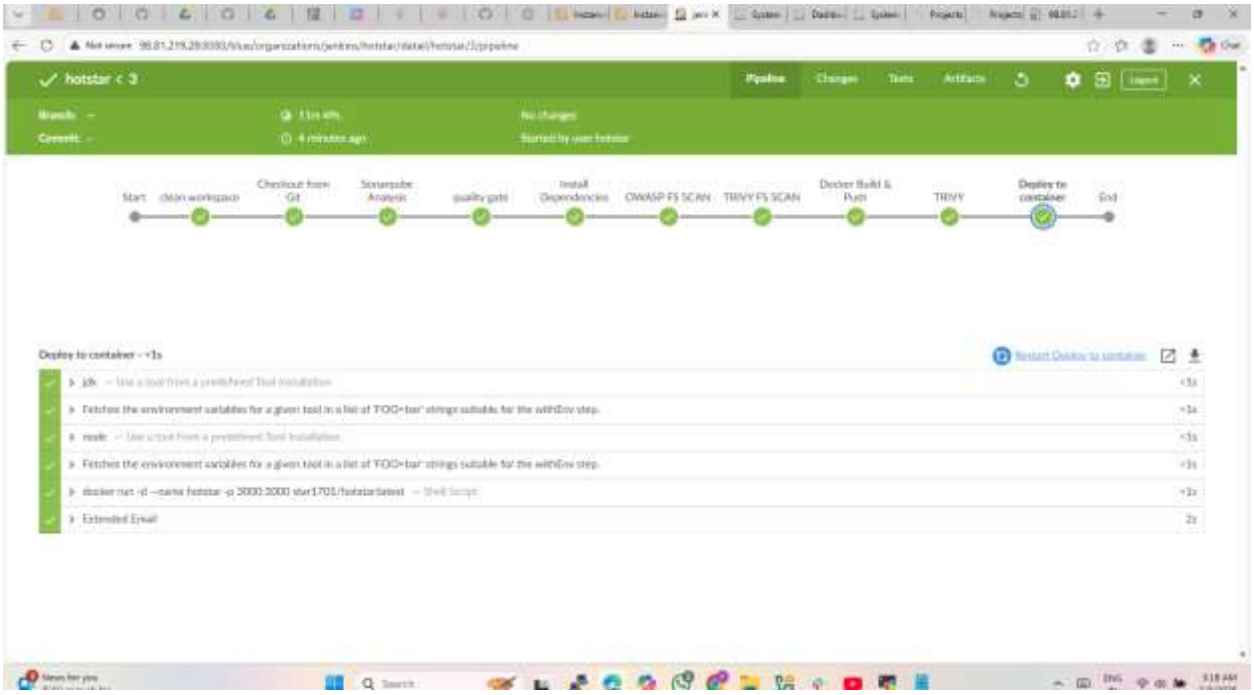



**Create Job for Hotstar:**

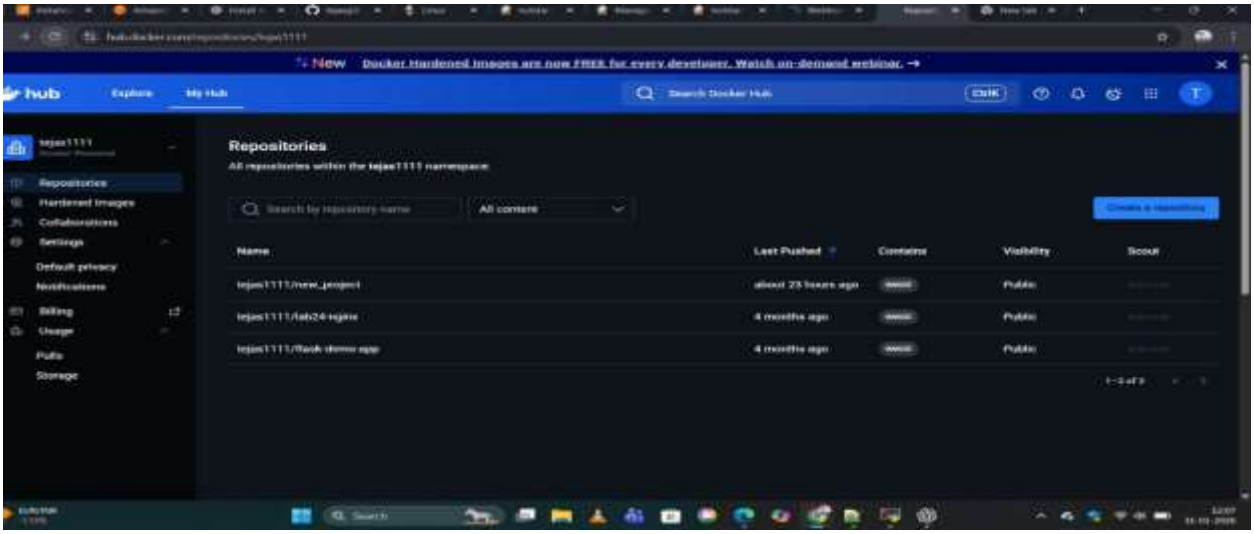
- Add a pipeline , to test the Github Clone stage of Private Registry



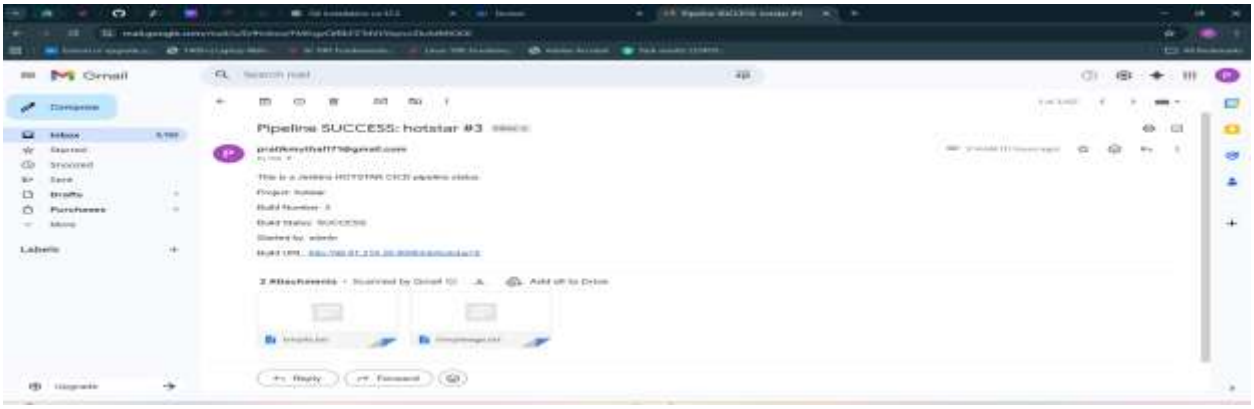

## 6.5 Build pipeline:

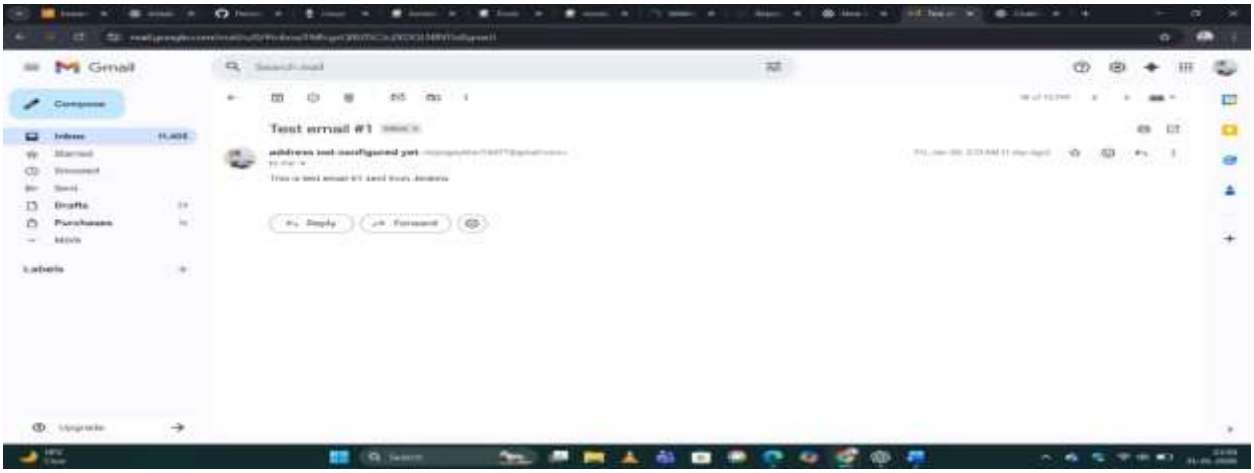


**Docker:**




**Email Notification:**



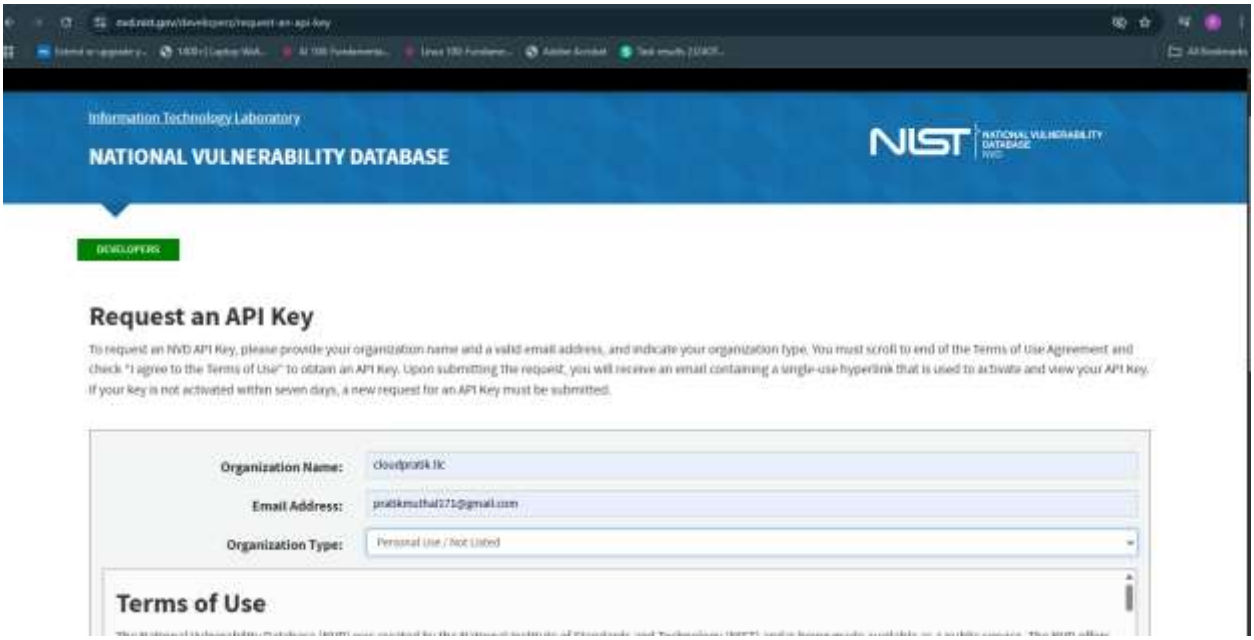



## Add credentials as Username and password in Jenkin:

- **Extended Email Notification :**

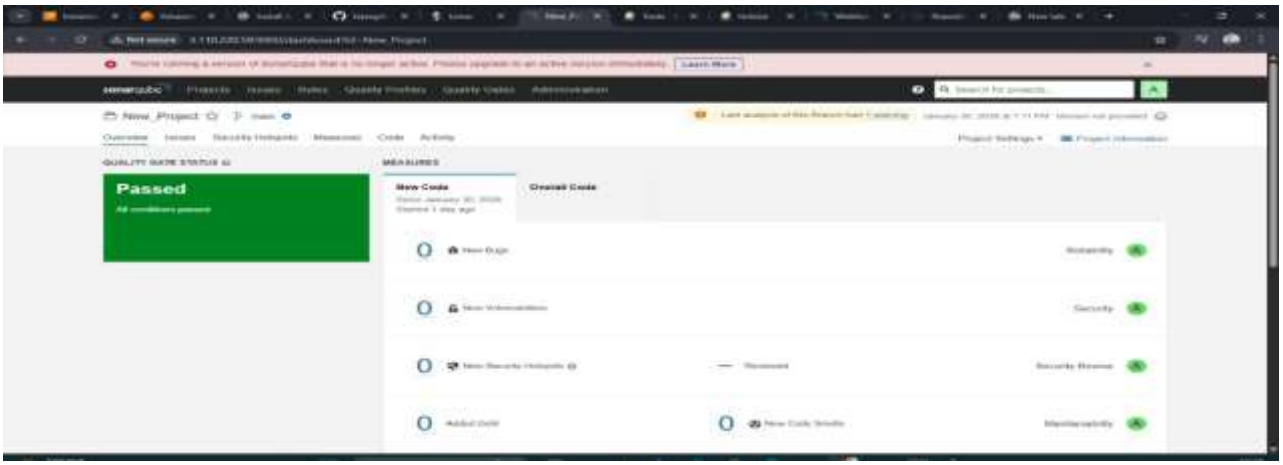

## 6.6 Register for NVD API for Dependency Check:

- The National Vulnerability Database (NVD) API provides access to security vulnerabilities (CVEs) and is often used with tools like OWASP Dependency-Check to identify security risks in software dependencies.



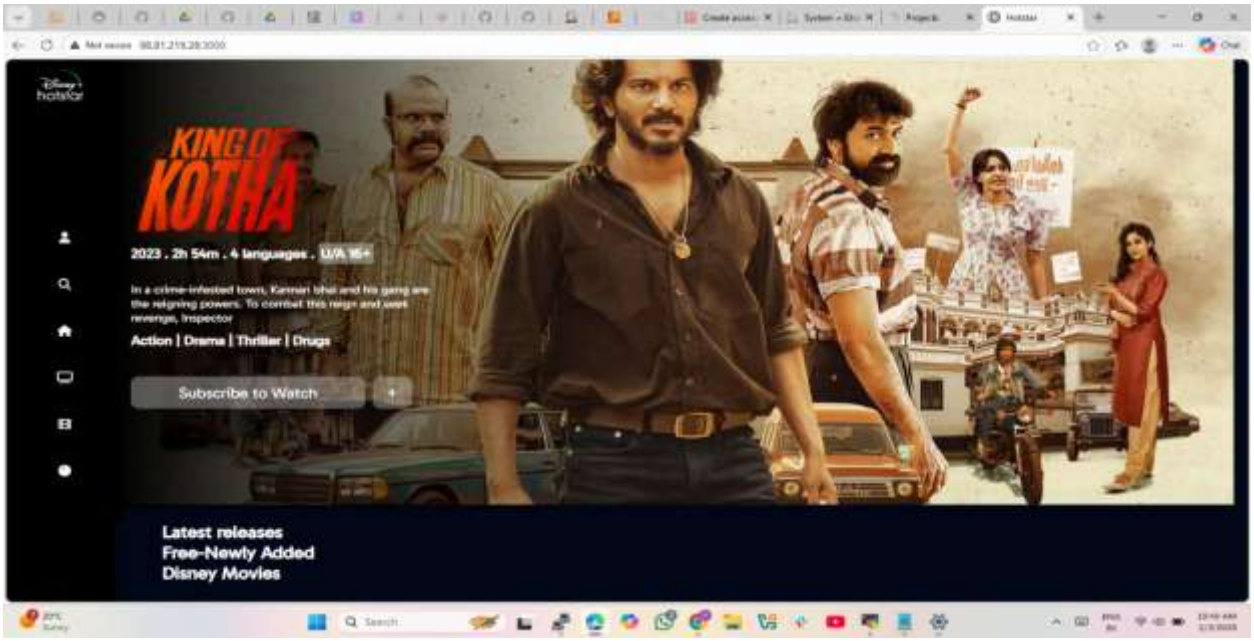

**You can see the report has been generated and the status shows as passed. You can see that there are 943 lines it scanned. To see a detailed report, you can go to issues:**

- You will see that in status, a graph will also be generated and Vulnerabilities.

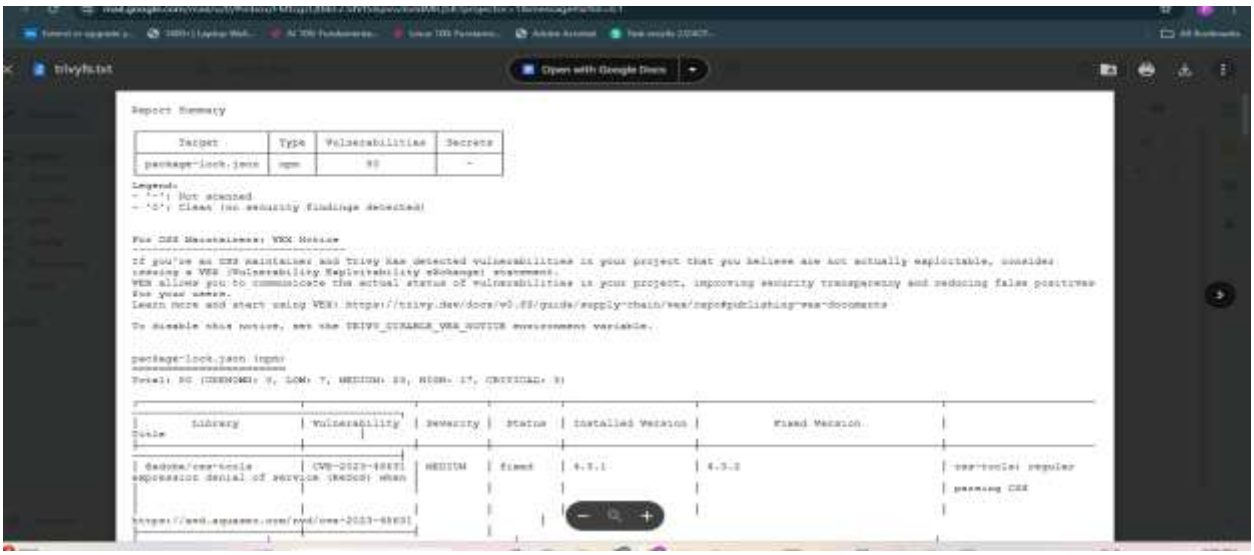


**<public-ip:3000>**

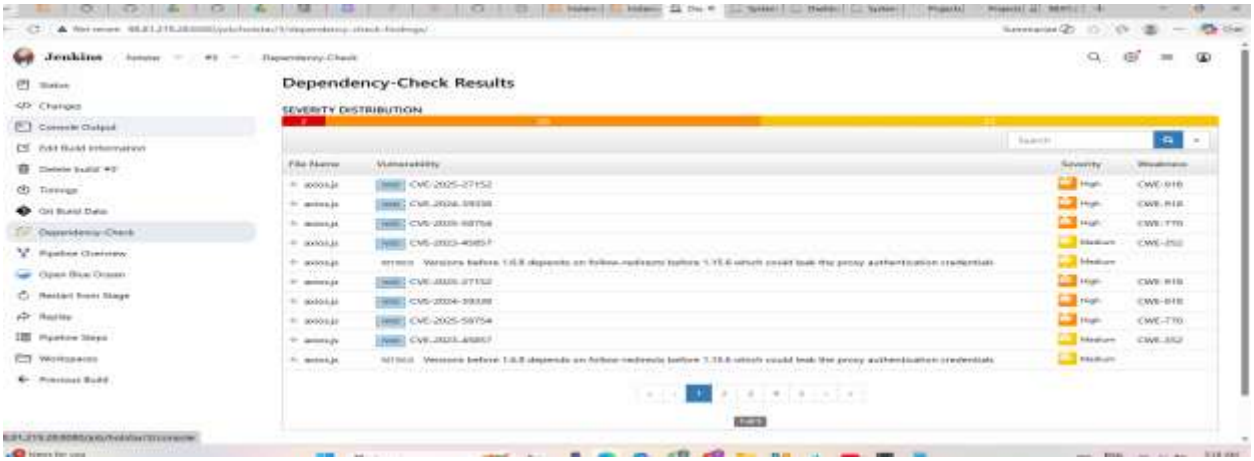
**Our Applciation is live with this output:**



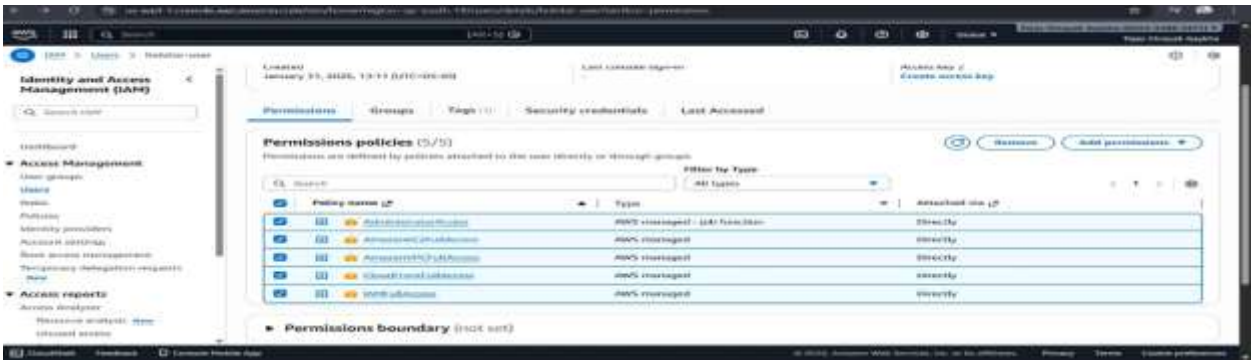

**Open Files trivy as attached to view vulnerabilities:**

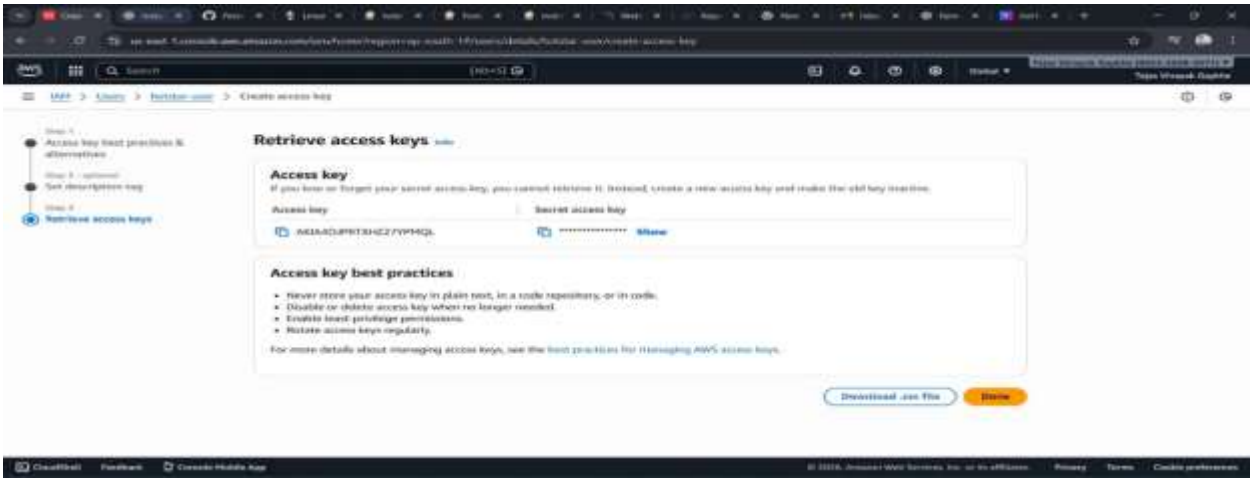


**Dependency check – Result:**




**EKS cluster Step on aws:**






**Aws configure in VM:**

**# Step - 3: Create EKS Cluster using eksctl #**

****Syntax:****

eksctl create cluster --name cluster-name \

--region region-name \

--node-type instance-type \

--nodes-min 2 \

--nodes-max 2 \

--zones <AZ-1>,<AZ-2>

**Option 1 for Asian Region ## Mumbai: <br/>**

eksctl create cluster --name cloudaseem-cluster4 --region ap-south-1 --node-type t2.medium --zones ap-south-1a,ap-south-1b

**Option 2 for US ## N. Virgina: <br/>**

eksctl create cluster --name cloudaseem-cluster4 --region us-east-1 --node-type t2.medium --zones us-east-1a,us-east-1b

```
5-30 01:02:42 [✔]  all EKS cluster resources for "hotstar-cluster"
ve created
5-30 01:02:42 [ℹ]  nodegroup "ng-97cfb2af" has 2 node(s)
5-30 01:02:42 [ℹ]  node "ip-192-168-25-204.ap-south-1.compute.intern
dy
5-30 01:02:42 [ℹ]  node "ip-192-168-63-236.ap-south-1.compute.intern
eady
5-30 01:02:43 [ℹ]  waiting for at least 2 node(s) to become ready in
b2af"
5-30 01:02:43 [ℹ]  nodegroup "ng-97cfb2af" has 2 node(s)
5-30 01:02:43 [ℹ]  node "ip-192-168-25-204.ap-south-1.compute.intern
eady
5-30 01:02:43 [ℹ]  node "ip-192-168-63-236.ap-south-1.compute.intern
rady
5-30 01:02:43 [ℹ]  created 1 managed nodegroup(s) in cluster "hotsta
ster"
5-30 01:02:44 [ℹ]  kubectl command should work with '/root/.kube/confi
ry "kubectl get nodes'
5-30 01:02:44 [✔]  EKS cluster hotstar-cluster in "ap-south-1" regi
ion is ready
```


**To update this manifest.yml file with your Docker images name and apply manifest.yml file**

**---**

apiVersion: apps/v1

kind: Deployment

metadata:

name: hotstar-deployment

spec:

replicas: 2

strategy:

type: RollingUpdate

selector:

matchLabels:

app: hotstar

template:

metadata:

labels:

app: hotstar

spec:

containers:

- name: hotstar-container

image: star1701/hotstar

ports:

- containerPort: 3000

---

apiVersion: v1

kind: Service

metadata:

name: hotstar-service

spec:

type: LoadBalancer

selector:

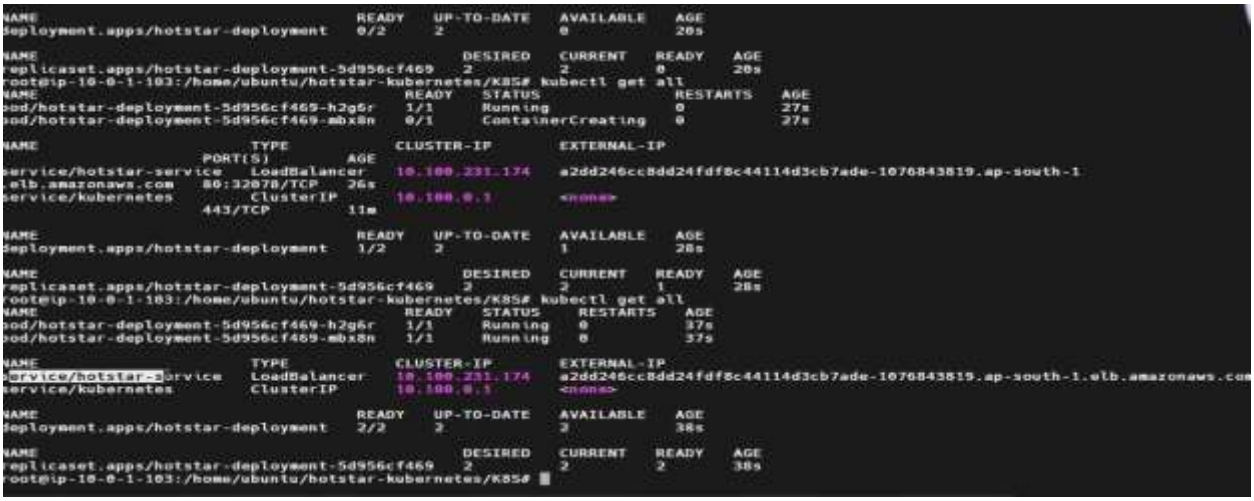
app: hotstar

ports:

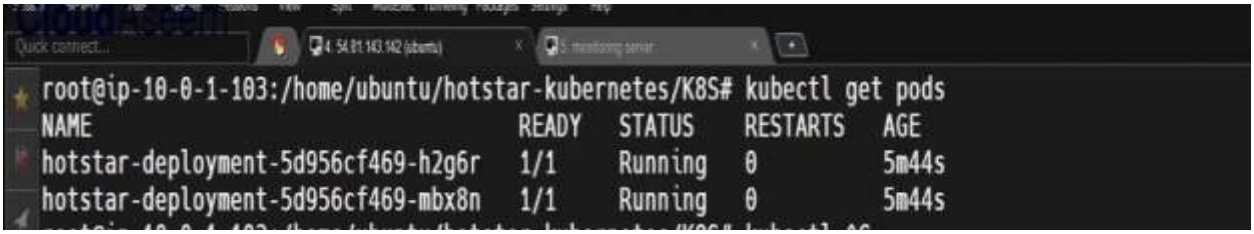
- port: 80

targetPort: 3000 and execute this command

**kubectl apply -f manifest.yml:**

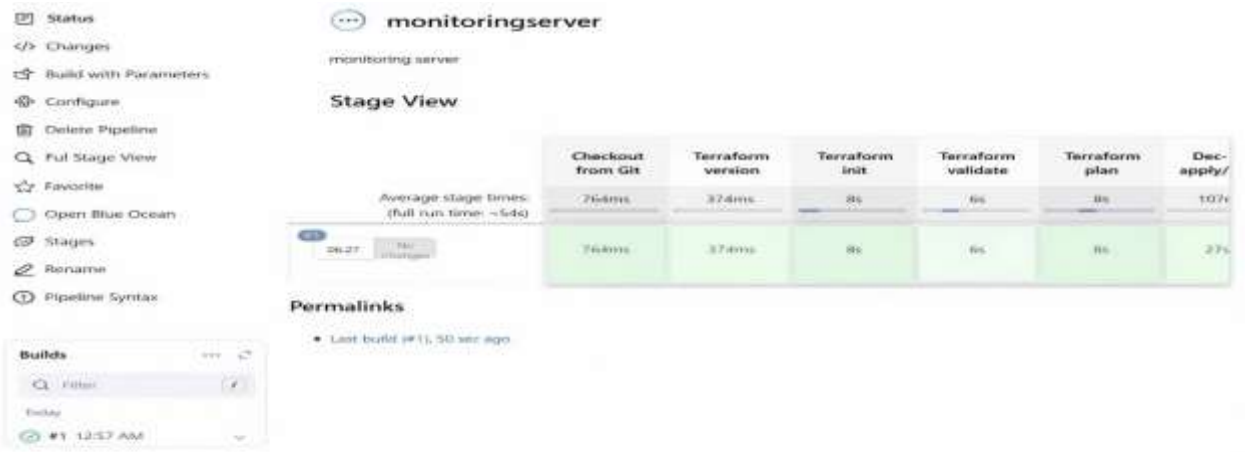



**TEST Kubernetes Auto Healing Function Kubernetes has a built-in self-healing mechanism that automatically replaces failed or deleted pods when they are managed by a Deployment, ReplicaSet, or StatefulSet.**

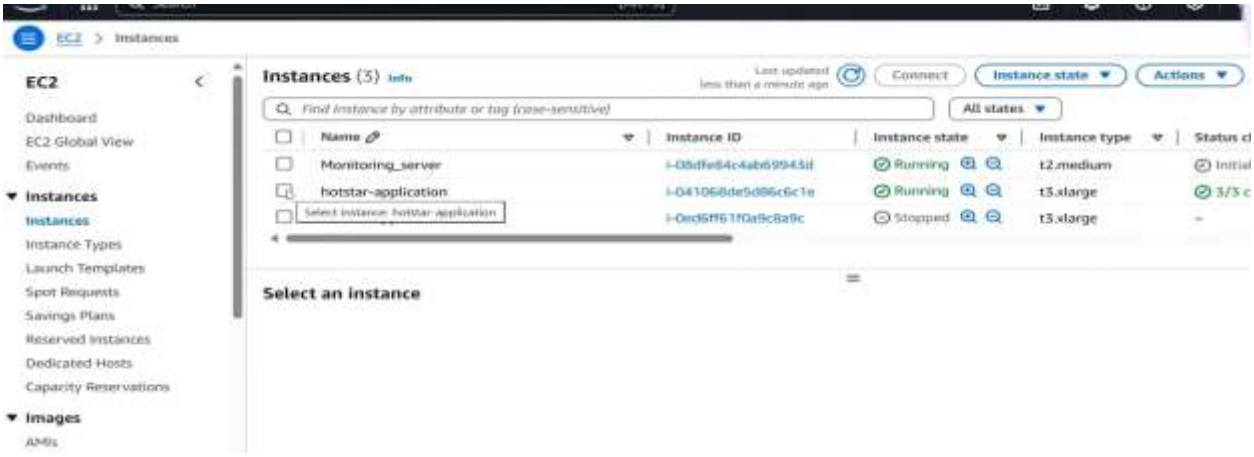



**Successfully Deployed a Hotstar on Kubernetes with Loadbalancer Enabled with AutoHealing.**

**Monitoring Server setup with Jenkins + Terraform:**




## Verify the Monitoring server:

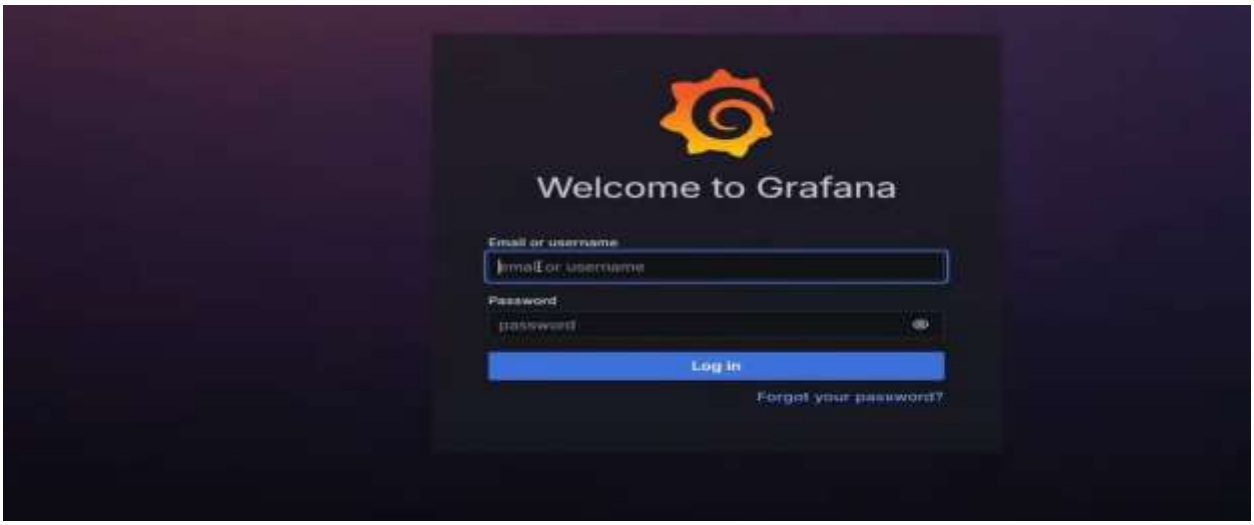


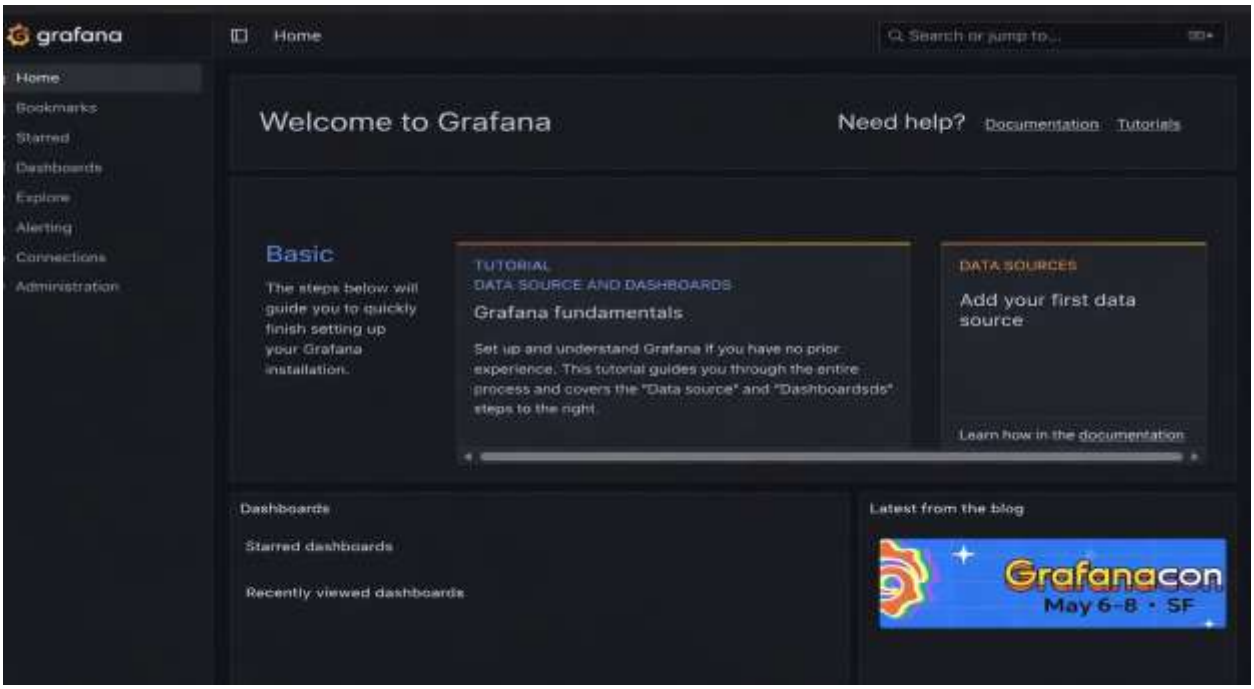

## Installing Grafana and Prometheus for Monitoring:

- Grafana and Prometheus are commonly used for monitoring Kubernetes clusters, EC2 instances, and other infrastructure components. Follow these steps to install them on an Ubuntu server.

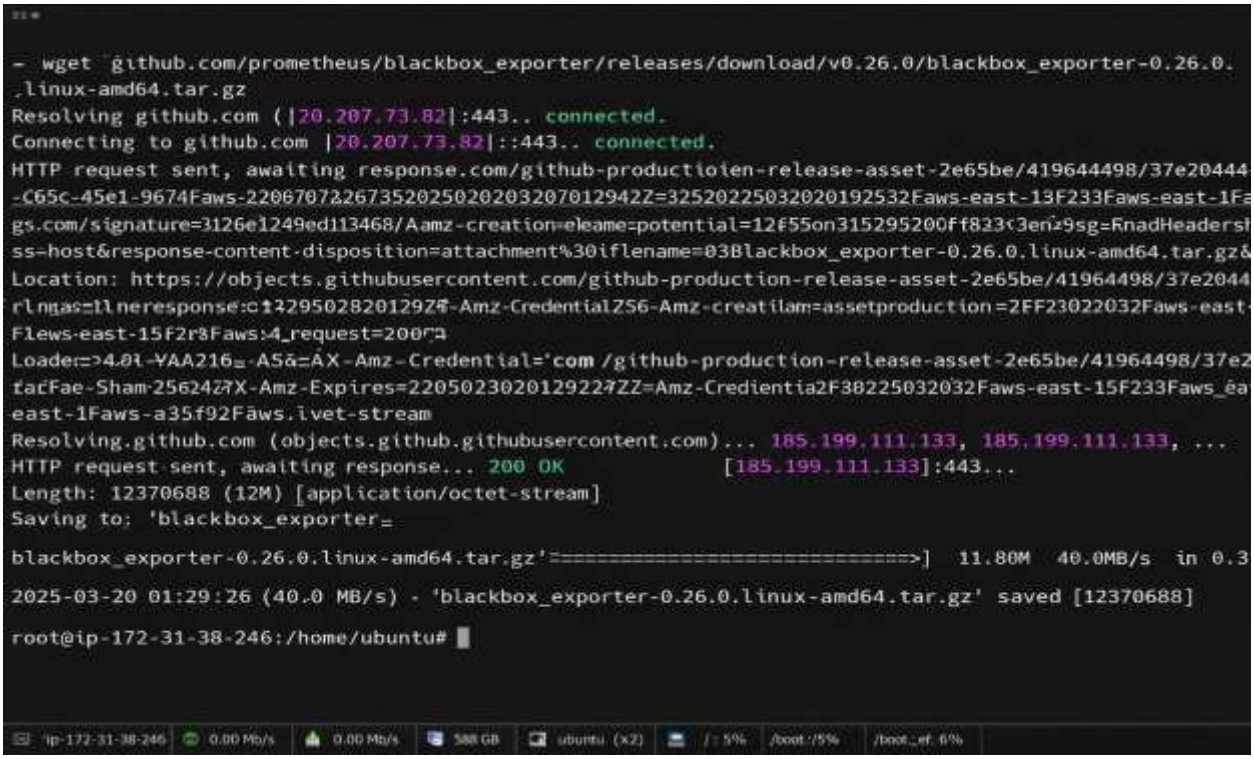
**After installation, you can access Grafana at:**

- # http://your-server-ip:3000 (default user: admin, password: admin)



**Install Blackbox exporter:**




**Step 1: Edit prometheus.yml :**

Open the **Prometheus configuration file**:

Add the following **scrape jobs** at the end of the file:

- job_name: 'blackbox'

metrics_path: /probe

params:

module: [http_2xx] # Look for a HTTP 200 response.

static_configs:

- targets:

- http://prometheus.io # Target to probe with HTTP.

- http://IP:3000 # Target to probe with HTTPS.

relabel_configs:

- source_labels: [__address__]

target_label: __param_target

- source_labels: [__param_target]

target_label: instance

- target_label: __address__

replacement: 13.232.214.2:9115 # The blackbox exporter's real hostname.

- job_name: node_exporter

static_configs:

- targets:

- 'IP:9100'

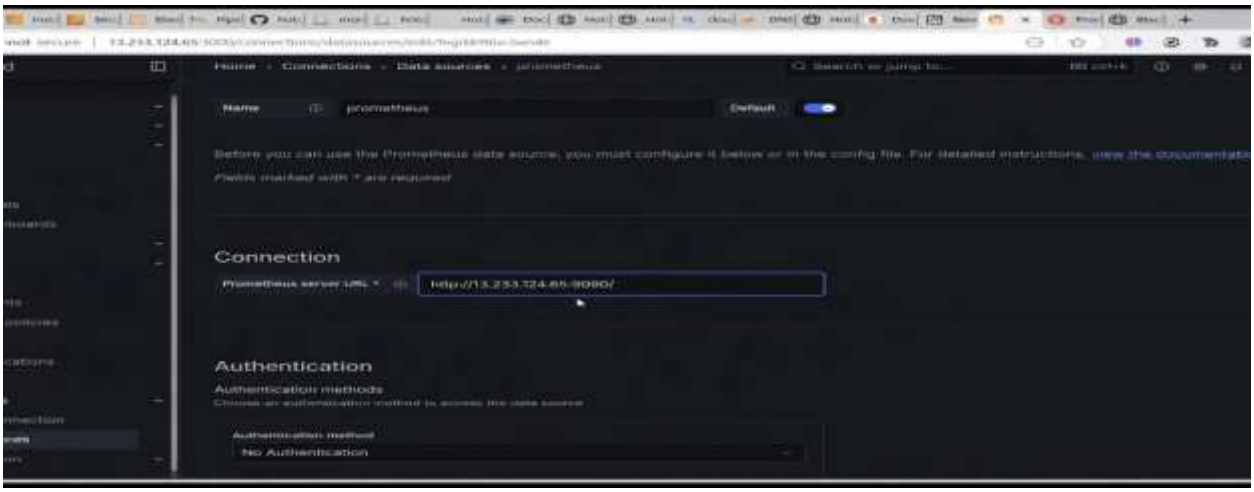
**Step 2: Restart Prometheus to Apply Changes:**

- **pgrep Prometheus and kill PID**

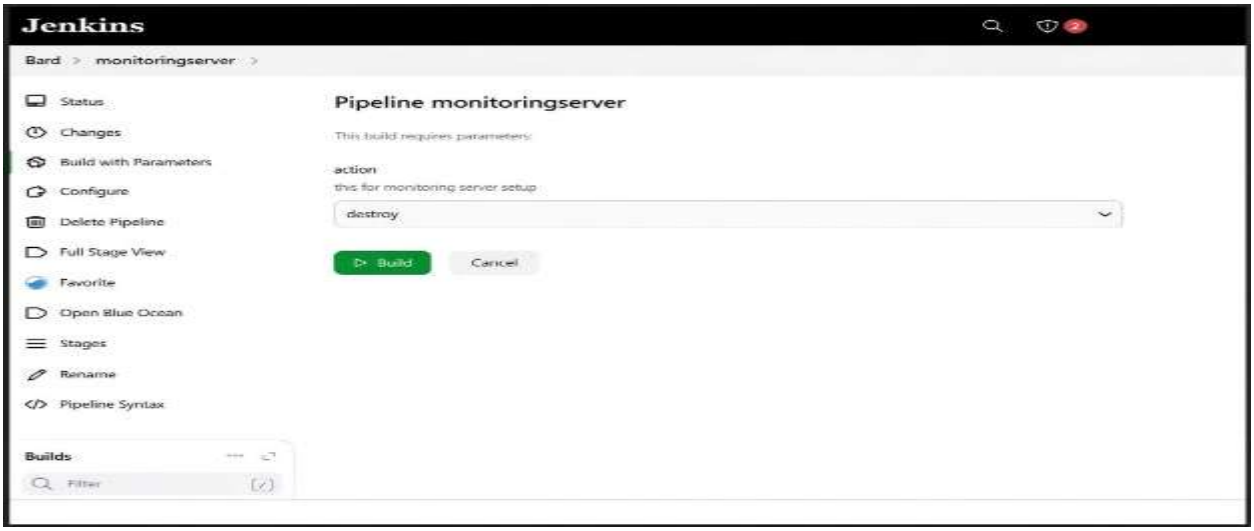
```
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# nano prometheus.yml
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# pgrep prometheus

root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# kill 6166
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# time=2025-03-20T01:35.401Z le
Received an OS signal, exiting gracefully..." signal=terminated
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1040 msg="opping scrape discovery manag
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:0189 msg="Stopping notify discovery manag
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:0205 msg="Stopping rule manager" componen
time=2025-03-20T01:35:15.401Z level=INFO source=manager.go:0106 msg="Rule manager stopped
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1036 msg="Stopping scrape manager...."
time=2025-03-20T01:35:15.407Z level=INFO source=main.go:1050 msg="Scrape discovery manager
time=2025-03-20T01:35:15.407Z level=INFO source=main.go:0702 msg="Notify discovery manager stopped
time=2025-03-20T01:35:15.407Z level=INFO source=notifier.go:0702 msg="Stopping scrape manager.."
time=2025-03-20T01:35:15.407Z level=INFO source=motifier.go:0345 msg="Scrape manager
time=2025-03-20T01:35:15.407Z level=INFO source=motifier.go:0345 msg="Notify discovery manager sto
time=2025-03-20T01:35:15.407Z level=INFO source=motifier.go:0702 msg="Stopping manager...
time=2025-03-20T01:35:15.407Z level=INFO source=notifier.go:0345 msg="Draining any remaining notifi
time=2025-03-20T01:35:15.407Z level=INFO source=notifier.go:0345 msg="Remaining notifications draine
time=2025-03-20T01:35:15.407Z level=INFO source=motifier.go:1345 msg="Notification manager stopped
time=2025-03-20T01:35:15.407Z level=INFO source=main.go:1375 msg="See you next time!"

[1]+  Done
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64#

root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64#
```

- **Restart ./Prometheus**

**Step 3: Connect Prometheus to Grafana:**




**Step - 4: 1. Delete Monitoring Server with Jenkins pipeline with action as destroy:**



# 7. ADVANTAGES & DISADVANTAGES

## Advantages

1. **Real-Time System Monitoring:**
   Prometheus collects real-time metrics, and Grafana displays them in easy-to-understand dashboards.
2. **Improved Performance Analysis:**
   CPU, memory, disk, and network usage can be monitored, helping to identify system bottlenecks.
3. **Better Security Control:**
   AWS Security Groups act as a firewall to allow or block traffic based on ports and IP addresses.
4. **Easy Visualization:**
   Grafana provides graphical dashboards, making system monitoring simple and user-friendly.
5. **Scalable and Cloud-Friendly:**
   These tools work well with cloud environments like AWS and Kubernetes, making the system scalable.
6. **Automation Support:**
   Prometheus can trigger alerts when thresholds are exceeded, helping in proactive system management.

## Disadvantages

1. **Complex Configuration:**
   Setting up Prometheus exporters, Grafana dashboards, and security group rules requires technical knowledge.
2. **Resource Consumption:**
   Monitoring tools consume CPU, RAM, and storage, especially in large systems.
3. **Limited Firewall Features:**
   AWS Security Groups provide basic firewall functionality compared to advanced firewalls like pfSense.
4. **Alert Management Complexity:**
   Configuring alerts in Prometheus and Grafana can be difficult for beginners.
5. **Cloud Dependency:**
   AWS Security Groups work only within AWS, so they are not useful for on-premise environments.

# 8. CONCLUSION

This project presents the implementation of a cloud-based application deployment and monitoring system using modern DevOps and cloud technologies. The application was containerized using Docker and deployed on a Kubernetes cluster to achieve scalability and high availability. A CI/CD pipeline was implemented using Jenkins to automate the build and deployment process, which reduces manual errors and improves deployment efficiency.

Prometheus and Grafana were successfully integrated to monitor system performance, resource utilization, and application health in real time. AWS Security Groups were configured as firewalls to control inbound and outbound traffic, ensuring that only authorized ports and services were accessible. This enhanced the security of the deployed infrastructure and protected it from unauthorized access.

The project demonstrates how automation, monitoring, and security can be combined to build a reliable and scalable cloud infrastructure. It provides practical exposure to DevOps practices, cloud computing, containerization, and monitoring tools, which are widely used in the IT industry. Overall, the project proves that integrating monitoring and security mechanisms significantly improves system performance, reliability, and security, making it suitable for real-world enterprise environments.

# 9. REFERENCES


1. Docker Documentation, "Docker Overview and Containerization", Available: https://docs.docker.com
2. Kubernetes Documentation, "Kubernetes Concepts and Architecture", Available: https://kubernetes.io/docs
3. Jenkins Documentation, "Jenkins User Handbook", Available: https://www.jenkins.io/doc
4. Prometheus Documentation, "Monitoring Systems and Time Series Database", Available: https://prometheus.io/docs
5. Grafana Documentation, "Grafana Dashboards and Visualization", Available: https://grafana.com/docs
6. AWS Documentation, "Amazon EC2 and Security Groups", Available: https://docs.aws.amazon.com
7. SonarQube Documentation, "Static Code Analysis and Quality Gates", Available: https://docs.sonarqube.org
8. Wazuh Documentation, "Security Monitoring and Intrusion Detection", Available: https://documentation.wazuh.com
9. Elastic Stack Documentation, "Elasticsearch, Logstash, and Kibana", Available: https://www.elastic.co/guide
10. Red Hat, "Introduction to DevOps and CI/CD Pipelines", Red Hat Official Blog, 2022.