



**INSTITUTE FOR ADVANCED
COMPUTING AND SOFTWARE
DEVELOPMENT AKURDI,
PUNE**

DOCUMENTATION ON

**“Web Application Deployment with Load Balancing, Automation,
and Monitoring using Jenkins, Docker Swarm, and Grafana”**

PG-DITISS March-2023

SUBMITTED BY:

GROUP NO: 01

MAHESH ANERAU (233406)

NILESH PATIL (233430)

**MR. KARTIK AWARI
PROJECT GUIDE**

**MR. ROHIT PURANIK
CENTRE CO-ORDINATOR**

ABSTRACT

In today's dynamic and competitive digital landscape, deploying web applications efficiently, ensuring high availability through load balancing, automating deployment pipelines, and implementing real-time monitoring are paramount. This project report offers a comprehensive account of how Jenkins, Docker Swarm, Grafana, and Nginx were synergistically employed to achieve these objectives.

Jenkins served as the central automation hub, orchestrating the entire deployment pipeline. Docker Swarm was adopted to facilitate container orchestration and, alongside Nginx, ensured load balancing across multiple nodes, underpinning high availability and optimized performance.

Grafana played a pivotal role in real-time monitoring, offering deep insights into key performance metrics, application health, and resource utilization. This comprehensive approach streamlined operations, empowered teams to preemptively address issues, and minimized downtime.

This report provides an exhaustive dive into the intricacies of Docker Swarm cluster setup, Jenkins automation configuration, Nginx load balancing setup, and Grafana monitoring implementation. It outlines the challenges faced during these processes and showcases innovative solutions.

Moreover, the report distills the invaluable lessons acquired during the project, emphasizing the potential for continued enhancements in deployment, automation, and monitoring practices. This project stands as a testament to the transformative impact of a harmonized technology stack in delivering scalable, resilient, and high-performance web applications in today's competitive digital landscape

TABLE OF CONTENTS

Topics	Page No.
ABSTRACT	
LIST OF ABBREVIATIONS	
LIST OF FIGURES	
LIST OF TABLES	
1. INTRODUCTION	1
1.1 Problem Statement	2
2. LITERATURE SURVEY	3
3. METHODOLOGY	
3.1 System Architecture	4
4. REQUIREMENT SPECIFICATION	5
4.1 Hardware Requirement	
4.2 Software Requirement	
5. WORKING	6
6. IMPLEMENTATION	9
7. APPLICATIONS	24
8. ADVANTAGES & DISADVANTAGES	25
9. CONCLUSION	26
10. REFERENCES	27

LIST OF ABBREVIATIONS

Sr. No.	Abbreviation	Full Form
1.	AWS	Amazon Web Services
2.	EC2	Elastic Compute Cloud
3.	SSH	Secure Shell
4.	SCP	Secure Copy
5.	CI	Continuous Integration
6.	CD	Continuous Deployment
7.	GIT	Global Information Tracker
8.	HTTPS	Secure Hyper Text Transfer Protocol
9.	SMTP	Simple Mail Transfer Protocol
10.	NLB	Network Load Balancing

LIST OF FIGURES

Figure No.	Figure Name	Page No
Figure 1.	Architecture Diagram	4
Figure 2.	Working Diagram	6

LIST OF TABLES

Table No.	Table Name	Page No.
Table 1.	Author and Publisher	3

1. INTRODUCTION

In the contemporary digital landscape, web applications stand as the cornerstone of modern business operations and customer engagement. The efficiency and effectiveness of these applications' deployment, scaling, and performance monitoring are integral to their success. This project report serves as an exploration of the seamless integration of cutting-edge technologies and industry best practices to address these critical aspects.

Background:

Historically, the deployment of web applications posed significant challenges, often resulting in complex and error-prone processes. The advent of containerization technologies, exemplified by Docker, has revolutionized this paradigm. Containers offer a standardized, portable way to package and run applications, ensuring consistency across diverse environments. Automation, another pivotal trend, has reshaped software development and deployment. Jenkins, a widely adopted automation server, plays a pivotal role in automating the build, test, and deployment pipeline, streamlining the software release cycle, and improving reliability.

Moreover, achieving high availability and optimal performance for web applications necessitates the use of load balancing mechanisms. Nginx, a versatile and high-performance web server, excels in this role by evenly distributing incoming traffic across multiple application instances, enhancing both performance and resilience. In concert with these efforts, real-time monitoring and performance analysis have become indispensable for maintaining application health and user satisfaction. Grafana, an open-source platform, empowers organizations to visualize and analyze key performance metrics, proactively identify issues, and ensure efficient resource utilization.

1.1 PROBLEM STATEMENT

In the rapidly evolving landscape of web application deployment, organizations face a multifaceted challenge in ensuring seamless, scalable, and highly performant application delivery. Traditional deployment methods are fraught with complexities, manual interventions, and limited scalability, resulting in operational inefficiencies and increased downtime. Moreover, the lack of real-time monitoring and automation compounds these challenges, making it difficult to swiftly identify and address issues, ultimately affecting user experience and business continuity.

Manual and Error-Prone Deployment: Traditional deployment processes often involve manual steps, leading to human errors, inconsistencies, and delays in releasing updates and new features.

Limited Scalability: Ensuring high availability and scaling to meet increasing user loads can be challenging, especially when dealing with monolithic applications or relying on static infrastructure.

Inadequate Load Balancing: Efficiently distributing incoming traffic to application instances to optimize performance and fault tolerance is a complex task, and traditional approaches may not be sufficient.

Lack of Real-time Monitoring: Gaining real-time insights into application performance, resource utilization, and potential issues is essential for maintaining application health and ensuring a positive user experience.

Operational Overhead: Managing deployment configurations, updates, and rollbacks across multiple nodes or instances can be resource-intensive, leading to increased operational costs and complexity.

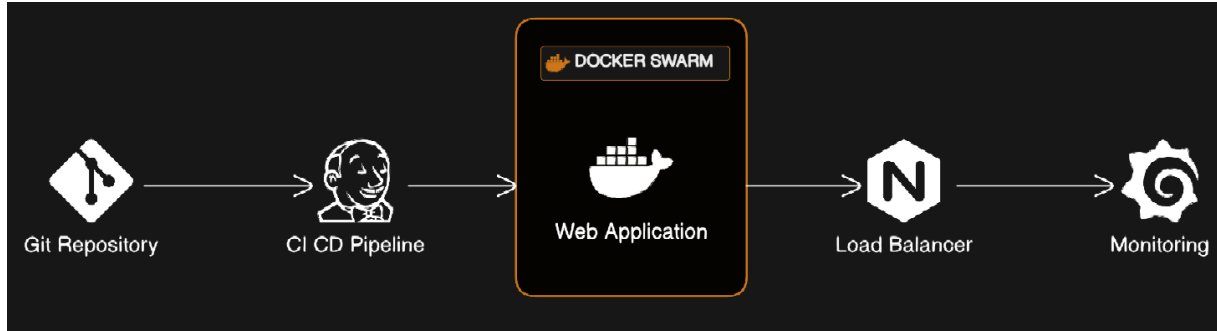
2. LITERATURE SURVEY

The process of DevOps ensures a faster production pipeline. Various strategies and methods can be followed for the continuous delivery of the product. But the time to market cannot be the only reason for all the applications. Some of them have to be stable and should be supported while adding new features and defining the different features to them. The deployment procedure or strategy is not going to be fixed as the design changes. It is very important to have a standard deployment pipeline that supports the versioning of the application with various features and is also supportive of the upgrade. There are multiple problems associated with the deployment that impacts the utility, robustness, correctness and security of the application. The factors that influence the process of deployment are to be controlled for the sake of optimized performance

Topic	Title	Author	Published by
Containerization and Docker	Docker: Up & Running	Sean P. Kane and Karl Matthias	O'Reilly Media
Container Orchestration with Docker Swarm	Docker Swarm Documentation	-	Docker
Continuous Integration and Automation with Jenkins	Jenkins: The Definitive Guide	John Ferguson Smart	O'Reilly Media
Load Balancing with Nginx	Nginx Official Documentation	-	Nginx
Real-time Monitoring with Grafana	Grafana Official Documentation	-	Grafana Labs

3. METHODOLOGY

3.1 SYSTEM ARCHITECTURE



The system architecture is as follows:

1. Git is used to manage the source code for the web page.
2. Jenkins is used to automate the CI/CD pipeline.
3. Docker Swarm is used to deploy the web application to a cluster of containers.
4. Nginx is used to load balance traffic between the containers.
5. Grafana is used to monitor the web application.

The following steps are involved in the deployment process:

- The web page code is pushed to a Git repository.
- Jenkins is configured to build and deploy the web application to Docker Swarm.
- A Docker Swarm cluster is created.
- The web application is deployed to the Docker Swarm cluster.
- Nginx is configured to load balance traffic between the containers.
- Grafana is configured to monitor the web application.

4. REQUIREMENT SPECIFICATION

4.1 HARDWARE REQUIREMENTS

Based on the hardware requirement need to configure every instance of infrastructure with requirement like;

1. Instance type: t2.micro
2. RAM: minimum 4GB
3. CPU: 2.5 GHZ

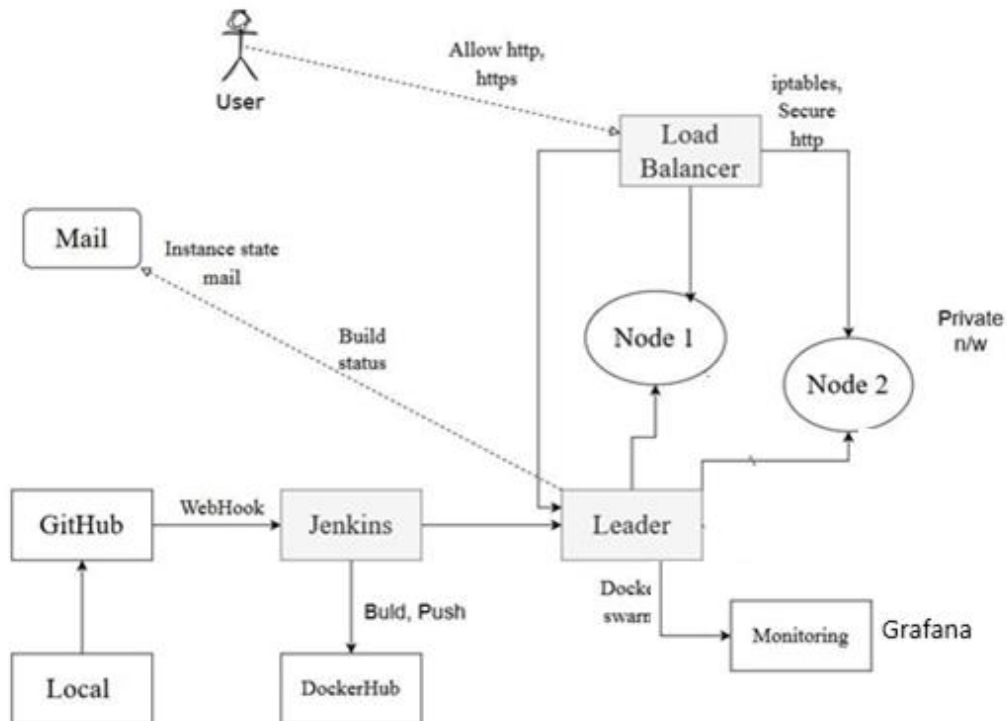
4.2 SOFTWARE REQUIREMENTS

1. Operating system: Debian
2. Platform: Amazon Web Service
3. CI Platform : Jenkins
4. Monitoring tool: Prometheus (Grafana)

In our project, we use Debian and other tools used are:

1. Git
2. Docker
3. Grafana

5. WORKING



The increasing adoption of Continuous Integration and Continuous Deployment (CI/CD) pipelines has transformed software development by enabling rapid and automated code integration, testing, and deployment.

AWS:

Amazon Web Services (AWS) is a comprehensive and widely used cloud computing platform provided by Amazon. It offers a vast array of cloud services that enable businesses and individuals to access and utilize computing resources over the internet on a pay-as-you-go basis. AWS provides a scalable and flexible environment to build, deploy, and manage applications and services without the need to invest in and manage physical hardware.

GIT:

Git is a widely used distributed version control system (VCS) that enables developers to collaborate on software development projects efficiently and track changes to source code over time. It was created by Linus Torvalds in 2005 to manage the development of the Linux kernel, and it has since become an essential tool for software development teams.

Jenkins:

Jenkins is an open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD) processes in software development. It was created to automate the building, testing, and deployment of software projects, helping teams streamline development workflows and ensure code quality.

Docker :

Docker is an open-source platform that enables developers to create, deploy, and manage applications and their dependencies in lightweight, isolated containers. Containers package software and its dependencies into a consistent environment that can run reliably across different environments, from development to production.

Docker Hub :

Docker Hub is a cloud-based registry service provided by Docker that serves as a central repository for Docker container images. It offers a platform for developers, teams, and organizations to store, share, and distribute container images, making it easier to collaborate on software development and streamline deployment workflows.

Docker Swarm:

It's a native clustering and orchestration solution for Docker, designed to manage a group of Docker containers and services as a single unit. It simplifies the deployment, scaling, and management of containerized applications by providing features for loadbalancing, high availability, and service discovery.

Nginx:

It is a popular open-source web server, reverse proxy server, and load balancer. It is renowned for its performance, scalability, and versatility, making it a widely used tool in modern web architecture. Nginx is designed to efficiently handle high levels of concurrent connections and serve static content swiftly, making it particularly suitable for serving web applications and websites.

HTTPS:

Docker Hub is a cloud-based registry service provided by Docker that serves as a central repository for Docker container images. It offers a platform for developers, teams, and organizations to store, share, and distribute container images, making it easier to collaborate on software development and streamline deployment workflows.

Grafana:

It is an open-source platform for monitoring and observability. It is designed to help users visualize and understand the performance of their systems and applications by creating interactive and customizable dashboards. Grafana is widely used in DevOps and IT operations to monitor infrastructure, applications, and various data sources in real-time.

Prometheus:

It is an open-source monitoring and alerting toolkit designed for reliability and scalability in modern, dynamic environments. It is primarily used to monitor the performance and health of computer systems, applications, and services in real-time. Prometheus is especially well-suited for monitoring containerized applications and microservices architectures, where traditional monitoring tools may fall short.

Node Exporter:

Node Exporter is a popular open-source component in the Prometheus monitoring ecosystem. It plays a crucial role in collecting system-level metrics from a target machine, such as a server or a virtual machine, and exposing these metrics in a format that Prometheus can scrape and store for monitoring and alerting purposes.

6. IMPLEMENTATION

Basic requirements-

Deploy local machine (Debian) + Launch all instance ensure all are in same subnet in our case it is us-east-1.

1st instance : Jenkins

Default vpc / Subnet us -east 1a /Enable public ip

2nd instance : Docker swarm leader

Default vpc /Subnet us -east 1a /Disbale public ip

3rd instance : Docker swarm node 1

Default vpc /Subnet us-east 1a/Disable public ip

4th instance : Docker swarm node 2

Default vpc/Subnet us-east 1a/Disable public ip

5th instance : Sql server

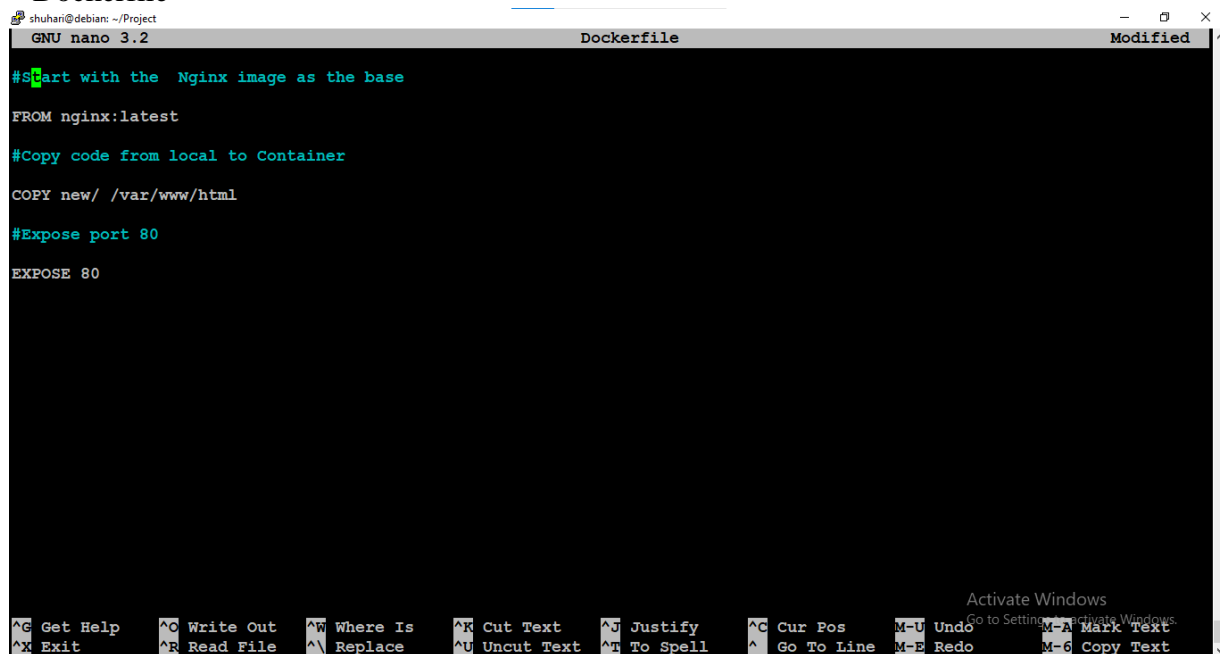
Default vpc/ Subnet us -east 1/Disable public ip

6th instance : load balancer

Default vpc /Subnet us-east 1/Enable public ip

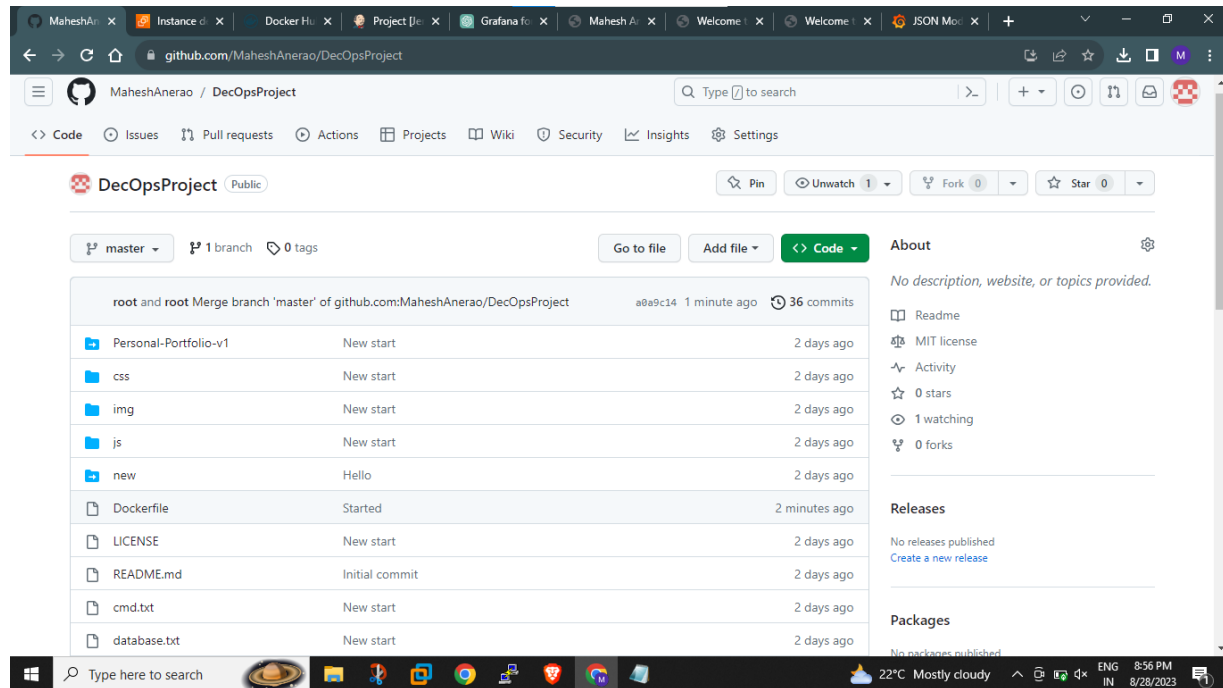
Implementation

Dockerfile



```
shuhari@debian: ~/Project
GNU nano 3.2 Dockerfile Modified
#Start with the Nginx image as the base
FROM nginx:latest
#Copy code from local to Container
COPY new/ /var/www/html
#Expose port 80
EXPOSE 80
```

Push all source code to GitHub private repository-



Generate GitHub token to create credentials in Jenkins-

Add global credentials on Jenkins-

- Add Webhook trigger
- Go to Jenkins -> credentials -> add credentials

Add Docker Hub credentials-

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > MareshAnerao/*****

Update credentials

Update
Delete
Move

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Username ?
MareshAnerao

☐ Treat username as secret ?

Password ?
Concealed [Change Password](#)

ID ?
Token

Description ?

[Save](#)

Activate Windows
Go to Settings to activate Windows.

Script-

← → ↻ ⚠ Not secure | 18.212.178.30:8080/job/project/configure

Dashboard > project > Configuration

Configure

General
Advanced Project Options
Pipeline

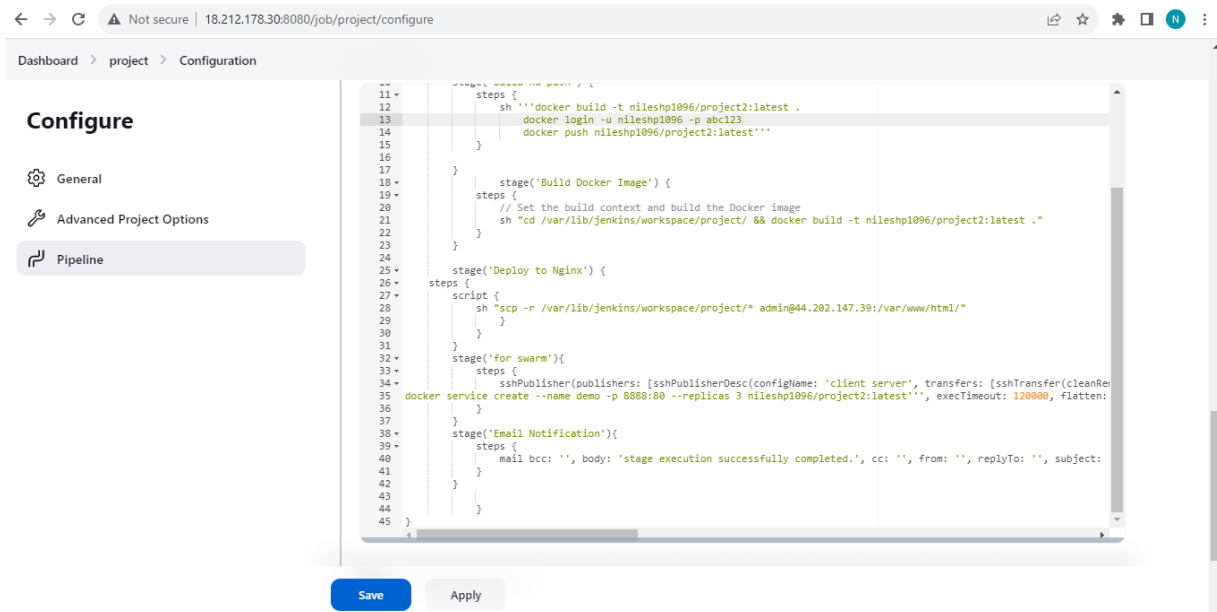
Pipeline

Definition
Pipeline script

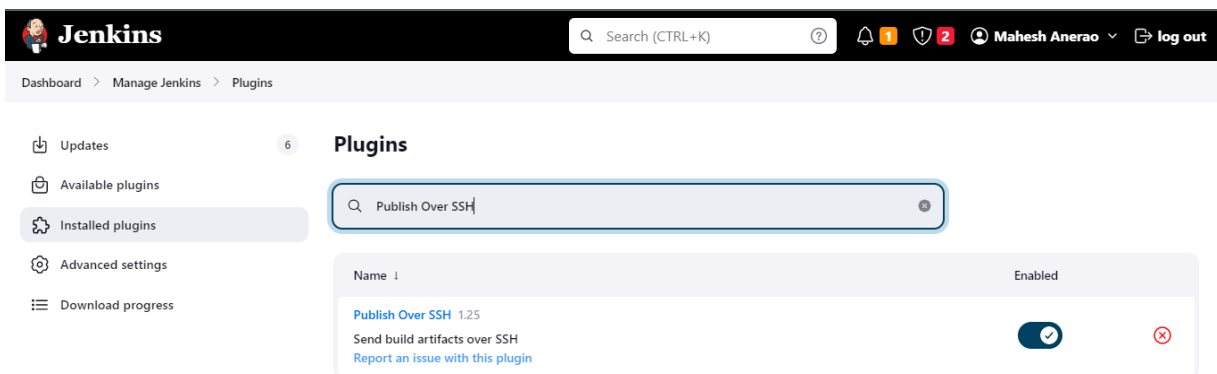
Script ?

```
1 pipeline {  
2   agent any  
3   stages {  
4     stage('Checkout') {  
5       steps {  
6         // Check out the code from GitHub  
7         git credentialsId: 'Token', uri: 'https://github.com/nileshp1096/project1.git'  
8       }  
9     }  
10    stage('build nd push') {  
11      steps {  
12        sh '''docker build -t nileshp1096/project2:latest .  
13          docker login -u nileshp1096 -p Nilesh@123  
14          docker push nileshp1096/project2:latest'''  
15      }  
16    }  
17  }  
18  stage('Build Docker Image') {  
19    steps {  
20      // Set the build context and build the Docker image  
21      sh "cd /var/lib/jenkins/workspace/project/ && docker build -t nileshp1096/project2:latest ."  
22    }  
23  }  
24 }
```

[Save](#) [Apply](#)



Publish over ssh-



Dashboard > Manage Jenkins > System >

Publish over SSH

Jenkins SSH Key ?

Passphrase ?

Concealed Change Password

Path to key ?

Key ?

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktZjEAAAAAG5vbmUAAAAEbm9uZQAAAAAAAAABAAAABlAAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAuK5RxtHwhd0U12WJj+kvR6K0HsGkOX6g3kSEffaSE3FqEendMDf57
BbXMwi21Epv3/X8FBvLduUUVK08Anl4UScaf/+0PZjhWfCwt/WeDbRvp0lcHbXWc+tcZU
LF8E/2++ARSG7Apk6/9Vif0zEjI04kgsTookNW8MwWBeefqKoiMk12AqjAshff6Zd/peXq
eCKOxgbUm/PTzbLGB9+BP3m3KdH5PI0FG5VST2KH07iYSDf+DiD0e/8mh2heV5pagNS4y8
0TRKxLT21hoFpOG6/MJPLxGAWAGA5s1rTo1uRsFvm8peKOG4nYEpfk4CNaiPy7SW43ZM3
H9NQYoRkxLbVMFIPCLivf+uPK4dUaAR2iK4InEdvvOHp1ofp8NmU11VFjk7jg7wLbF2FB
ND+Z3G3L2A7Z+3rUXNN7haDfavr+ANRwrhns56S/KtaEUePeAbwCmjJr905WxLcfmxQnj
8AbLYJJK0wMvVvKPLAKs56Kh6wV39upn1UWLKtU7AAAFkEGSuuhBkrrAAAAB3NzaC1yc2
-----
```

Save Apply

Activate Windows
Go to Settings to activate Windows.

Manage jenkins -> system -> publish over ssh section-

Dashboard > Manage Jenkins > System >

SSH Servers

SSH Server

Name ?

master

Hostname ?

54.85.8.118

Username ?

admin

Remote Directory ?

☐ Avoid sending files that have not changed ?

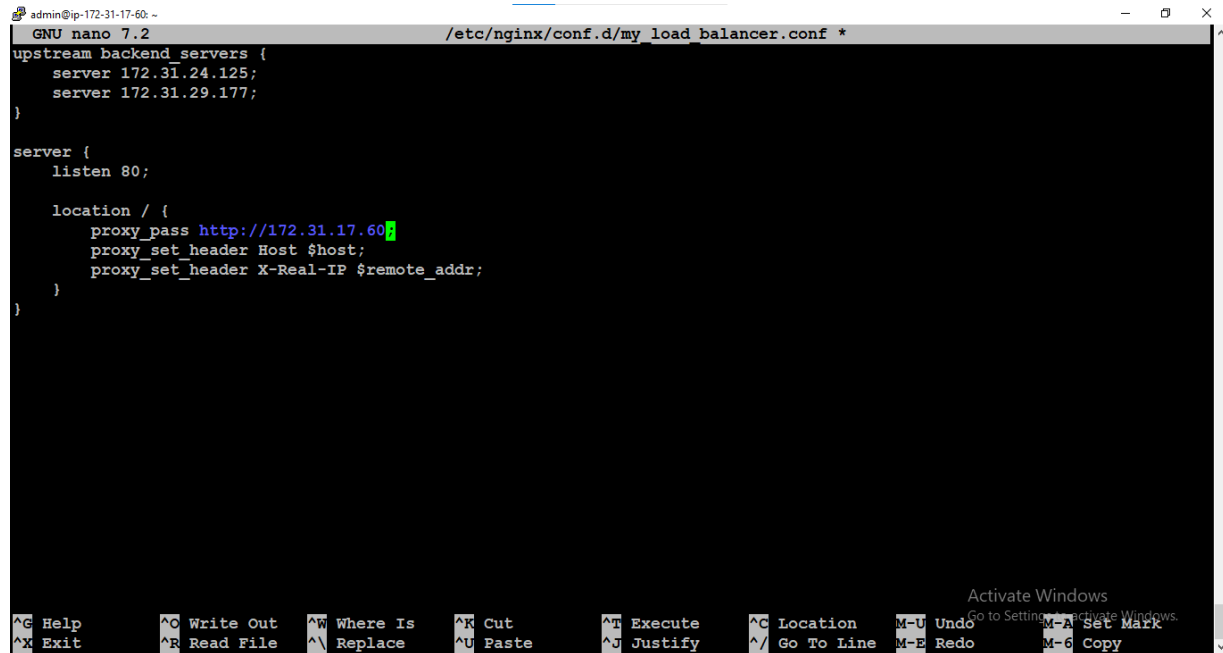
Advanced ▾

Save Apply

Activate Windows
Go to Settings to activate Windows.

Creation of load balancer-

- sudo apt-get install nginx
- sudo nano /etc/nginx/conf.d/my_load_balancer.conf
- sudo nginx -t
- sudo systemctl restart nginx



The screenshot shows a terminal window with the nano 7.2 editor open. The file being edited is /etc/nginx/conf.d/my_load_balancer.conf. The configuration content is as follows:

```
GNU nano 7.2 /etc/nginx/conf.d/my_load_balancer.conf *
upstream backend_servers {
    server 172.31.24.125;
    server 172.31.29.177;
}

server {
    listen 80;

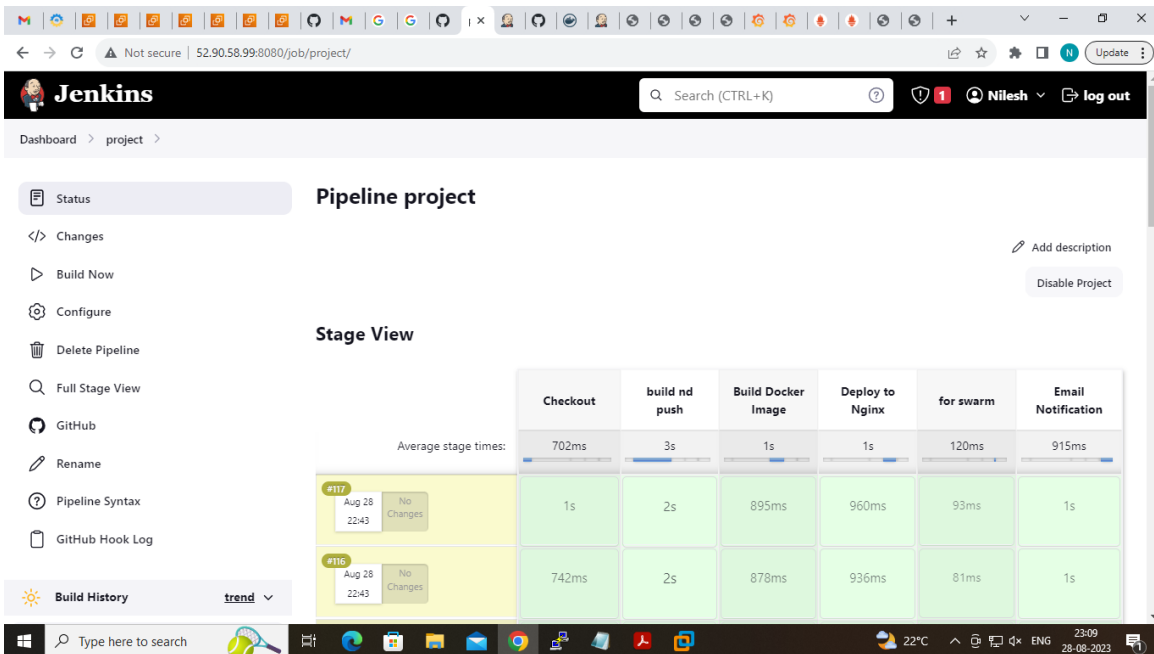
    location / {
        proxy_pass http://172.31.17.60;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

The terminal window also displays a list of nano editor shortcuts at the bottom:

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute	^C Location	M-U Undo	M-A Set Mark
^X Exit	^R Read File	^_ Replace	^U Paste	^J Justify	^_ Go To Line	M-E Redo	M-6 Copy

There is also a Windows activation watermark in the bottom right corner of the terminal window.

Build Successful-

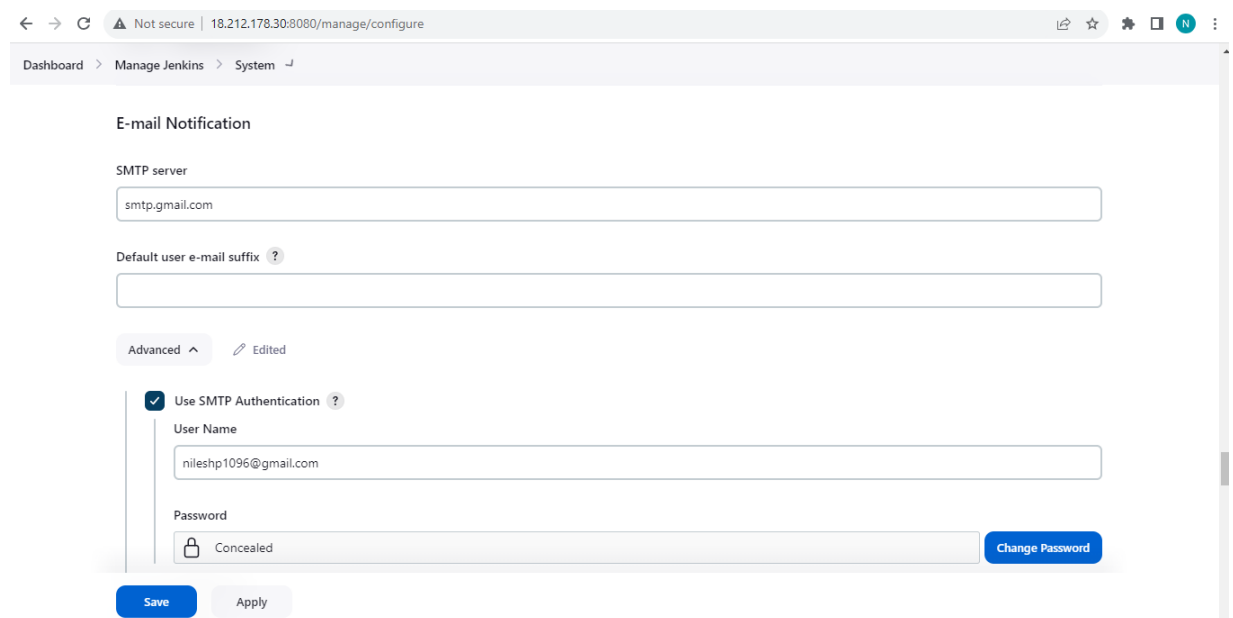


Email alert configuration-

Enable 2 step verification on your google account -> manage google account -> security -> password -> custom -> generate app based password

Jenkins : install plugin : email alert

Manage jenkins -> system -> extended email notification



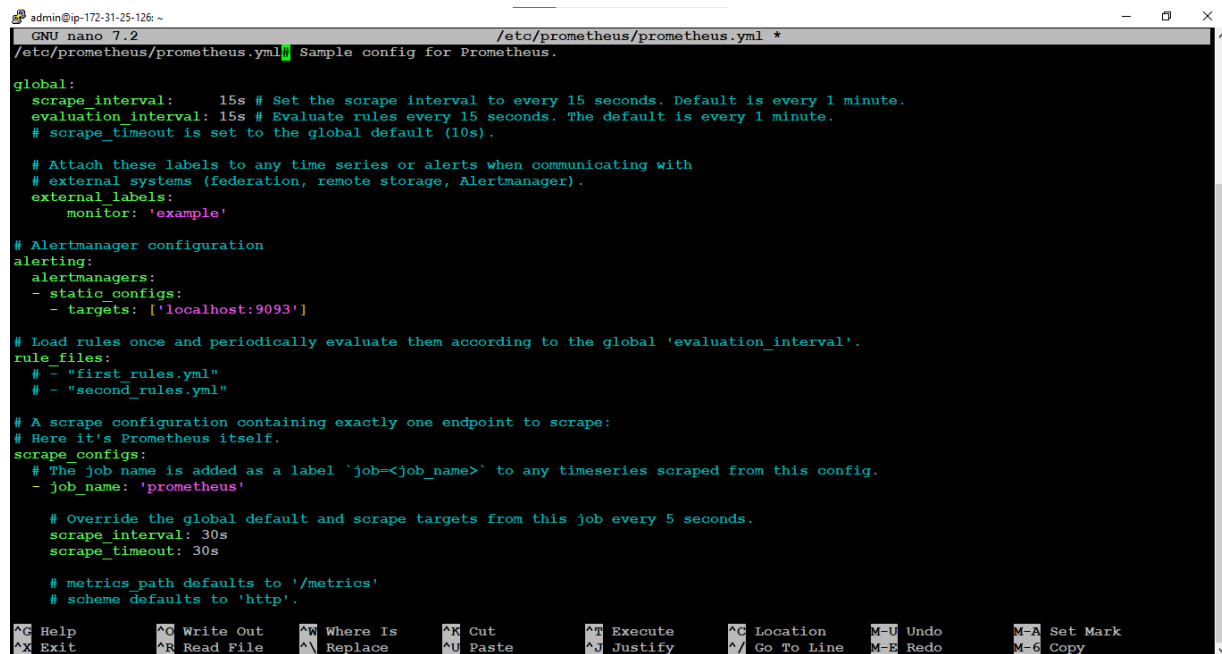
The screenshot shows the Jenkins 'E-mail Notification' configuration page. The browser address bar indicates the URL is '18.212.178.30:8080/manage/configure'. The breadcrumb trail is 'Dashboard > Manage Jenkins > System'. The page title is 'E-mail Notification'. Under 'SMTP server', the value 'smtp.gmail.com' is entered. The 'Default user e-mail suffix' field is empty. Below these fields are tabs for 'Advanced' (selected) and 'Edited'. In the 'Advanced' section, the 'Use SMTP Authentication' checkbox is checked. Under this, the 'User Name' field contains 'nileshp1096@gmail.com'. The 'Password' field is labeled 'Concealed' and has a 'Change Password' button next to it. At the bottom of the form are 'Save' and 'Apply' buttons.

Prometheus Installation:

- sudo apt-get install Prometheus
- sudo systemctl start Prometheus
- sudo systemctl enable Prometheus

Prometheus configuration:

- /etc/prometheus/prometheus.yml



```
admin@ip-172-31-25-126: ~  
GNU nano 7.2 /etc/prometheus/prometheus.yml *  
/etc/prometheus/prometheus.yml Sample config for Prometheus.  
  
global:  
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.  
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.  
  # scrape_timeout is set to the global default (10s).  
  
  # Attach these labels to any time series or alerts when communicating with  
  # external systems (federation, remote storage, Alertmanager).  
  external_labels:  
    monitor: 'example'  
  
# Alertmanager configuration  
alerting:  
  alertmanagers:  
    - static_configs:  
      - targets: ['localhost:9093']  
  
# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.  
rule_files:  
  # - "first_rules.yml"  
  # - "second_rules.yml"  
  
# A scrape configuration containing exactly one endpoint to scrape:  
# Here it's Prometheus itself.  
scrape_configs:  
  # The job name is added as a label 'job=<job_name>' to any timeseries scraped from this config.  
  - job_name: 'prometheus'  
  
    # Override the global default and scrape targets from this job every 5 seconds.  
    scrape_interval: 30s  
    scrape_timeout: 30s  
  
    # metrics_path defaults to '/metrics'  
    # scheme defaults to 'http'.  
  
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-C Copy
```

```
admin@ip-172-31-25-126: ~
GNU nano 7.2 /etc/prometheus/prometheus.yml *
# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets: ['localhost:9093']

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 30s
    scrape_timeout: 30s

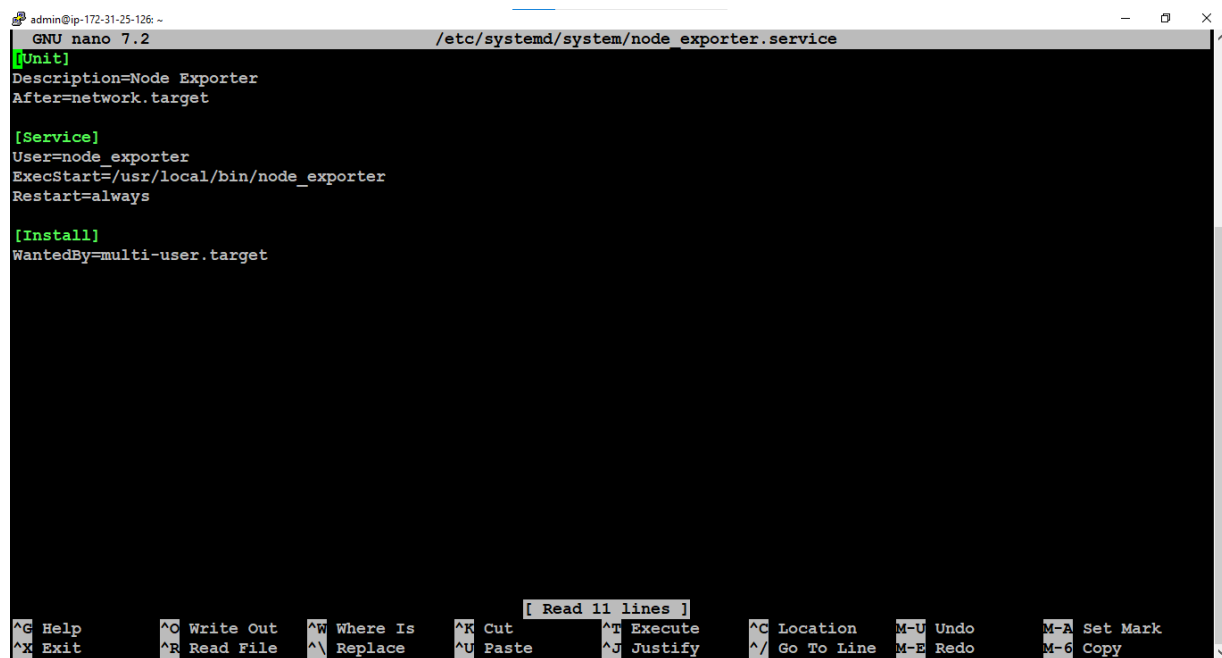
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['localhost:9090']
      - targets: ['54.85.8.118:9100']

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-6 Copy
```

Node Exporter Installation:

- wget
[https://github.com/prometheus/node_exporter/releases/download/v\\$VERSION/node_exporter-\\$VERSION.linux-amd64.tar.gz](https://github.com/prometheus/node_exporter/releases/download/v$VERSION/node_exporter-$VERSION.linux-amd64.tar.gz)
- tar -xzvf node_exporter-\$VERSION.linux-amd64.tar.gz
- sudo mv node_exporter-\$VERSION.linux-amd64/node_exporter /usr/local/bin/
- sudo nano /etc/systemd/system/node_exporter.service
- sudo systemctl daemon-reload
- sudo systemctl enable node-exporter



```
admin@ip-172-31-25-126: ~
GNU nano 7.2 /etc/systemd/system/node_exporter.service
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=node_exporter
ExecStart=/usr/local/bin/node_exporter
Restart=always

[Install]
WantedBy=multi-user.target

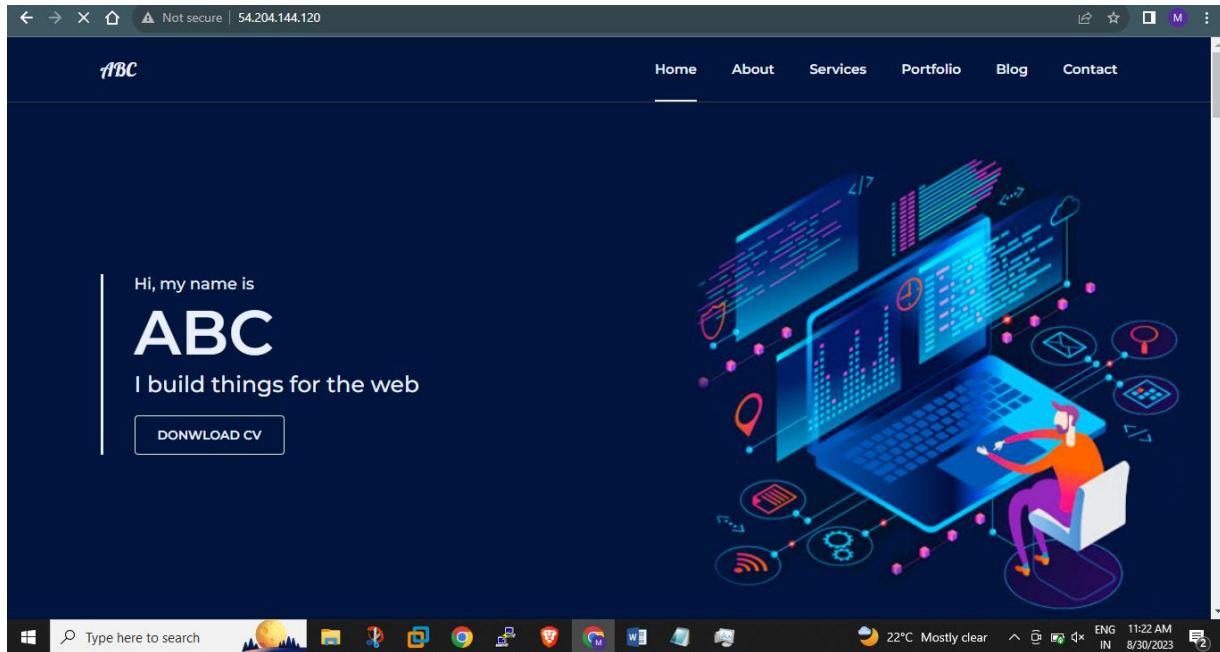
[ Read 11 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location  M-U Undo    M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify  ^_ Go To Line M-E Redo    M-6 Copy
```

Grafana Installation:

- sudo apt-get install -y software-properties-common
- sudo wget -q -O - <https://packages.grafana.com/gpg.key> | sudo apt-key add -
- sudo apt-get update
- sudo apt-get install grafana
- sudo systemctl start grafana-server
- sudo systemctl enable grafana-server

Output-

Web Page Deployment:



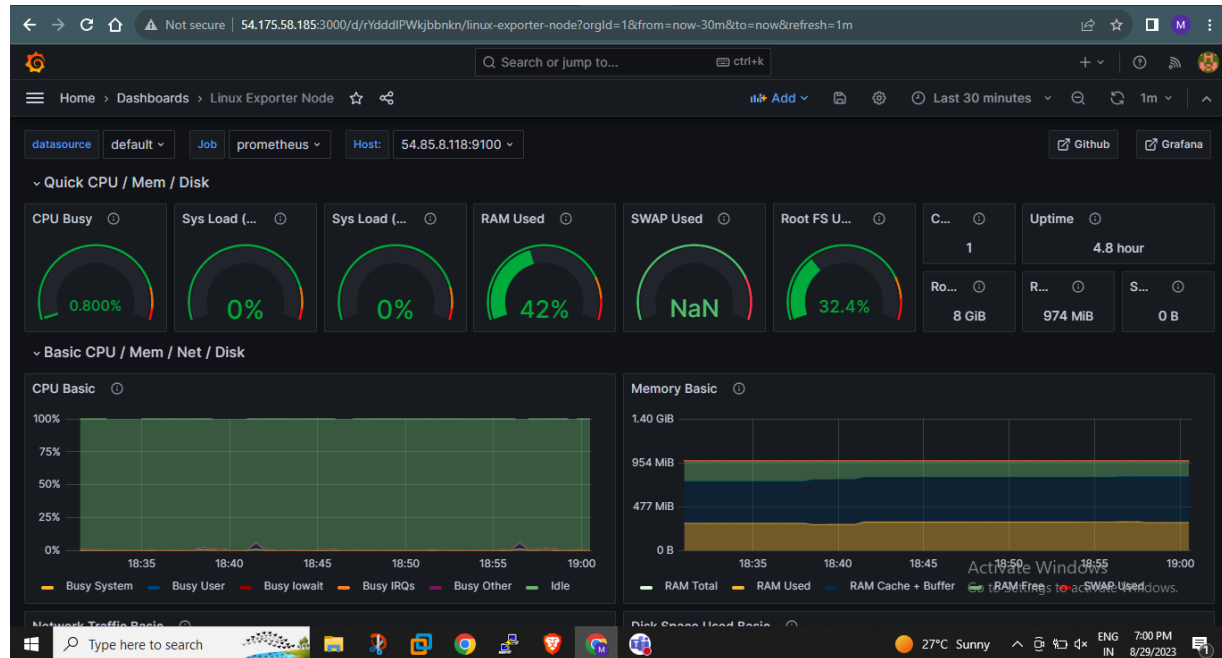
Prometheus Server:

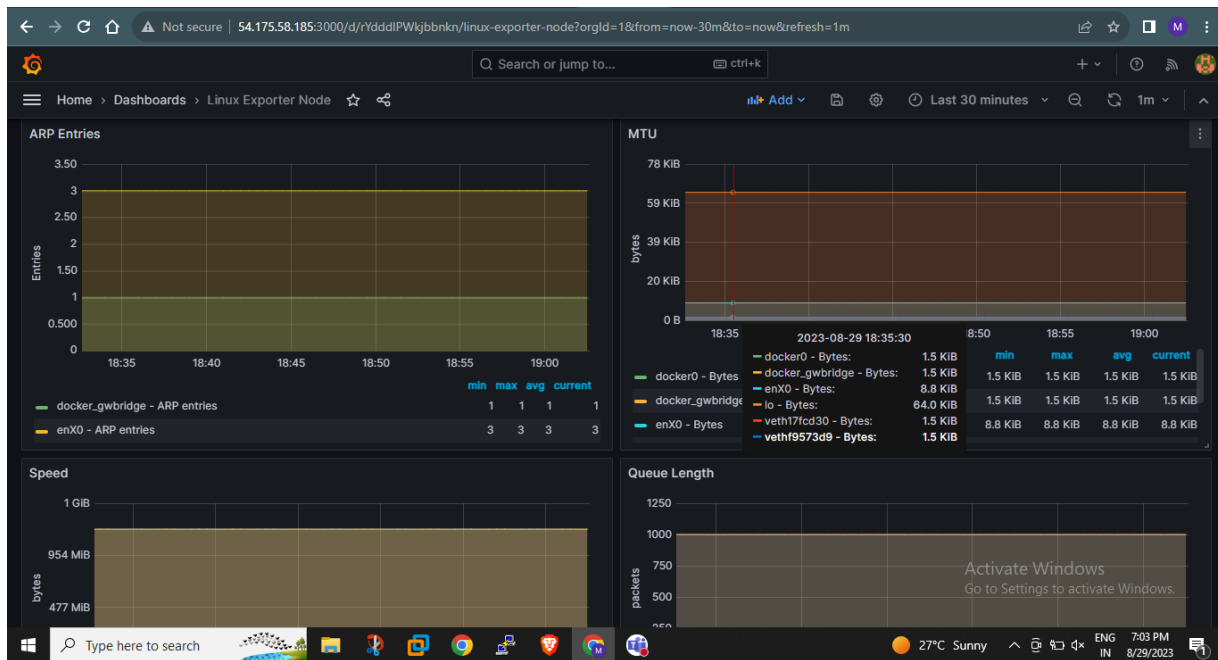
The screenshot shows the Prometheus web interface. The top navigation bar includes "Prometheus", "Alerts", "Graph", "Status", and "Help". The main heading is "Targets". Below it, there are tabs for "All", "Unhealthy", and "Collapse All". A summary line indicates "prometheus (2/2 up)" with a "show less" link. A table lists the targets:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://54.85.8.118:9100/metrics	UP	instance="54.85.8.118:9100" job="prometheus"	15.835s ago	65.34ms	
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	10.106s ago	5.688ms	

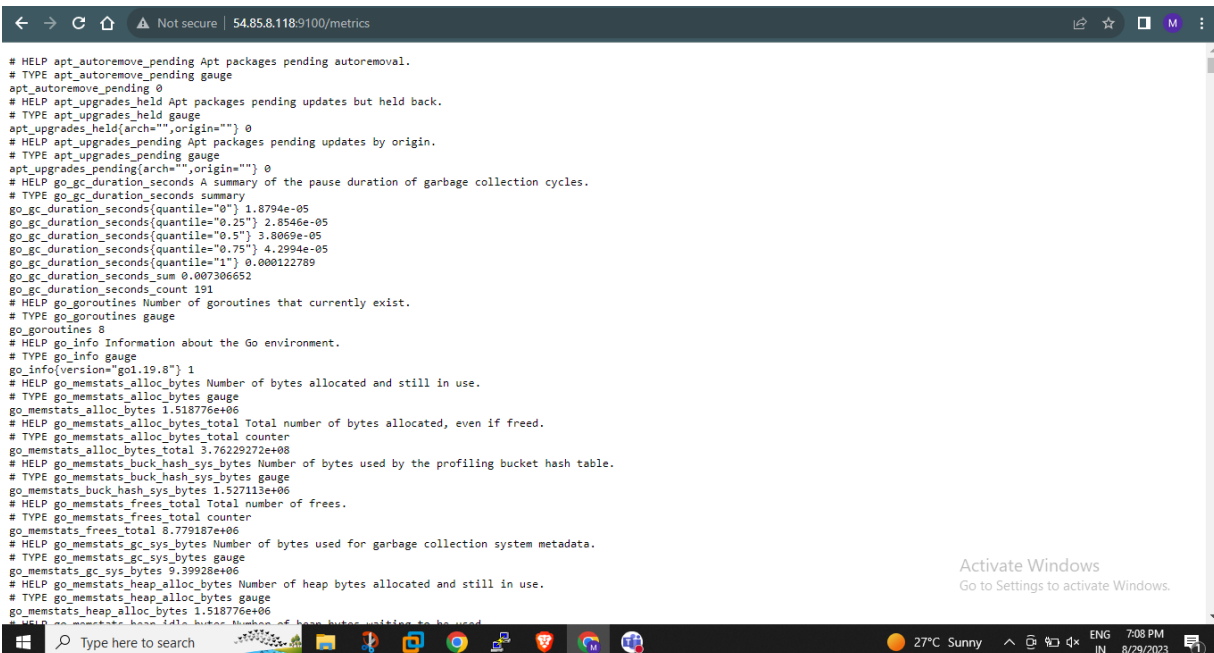
At the bottom right, there is a message: "Activate Windows. Go to Settings to activate Windows." The Windows taskbar at the bottom shows the search bar, application icons, and the system tray with the date and time: 7:07 PM, 8/29/2023.

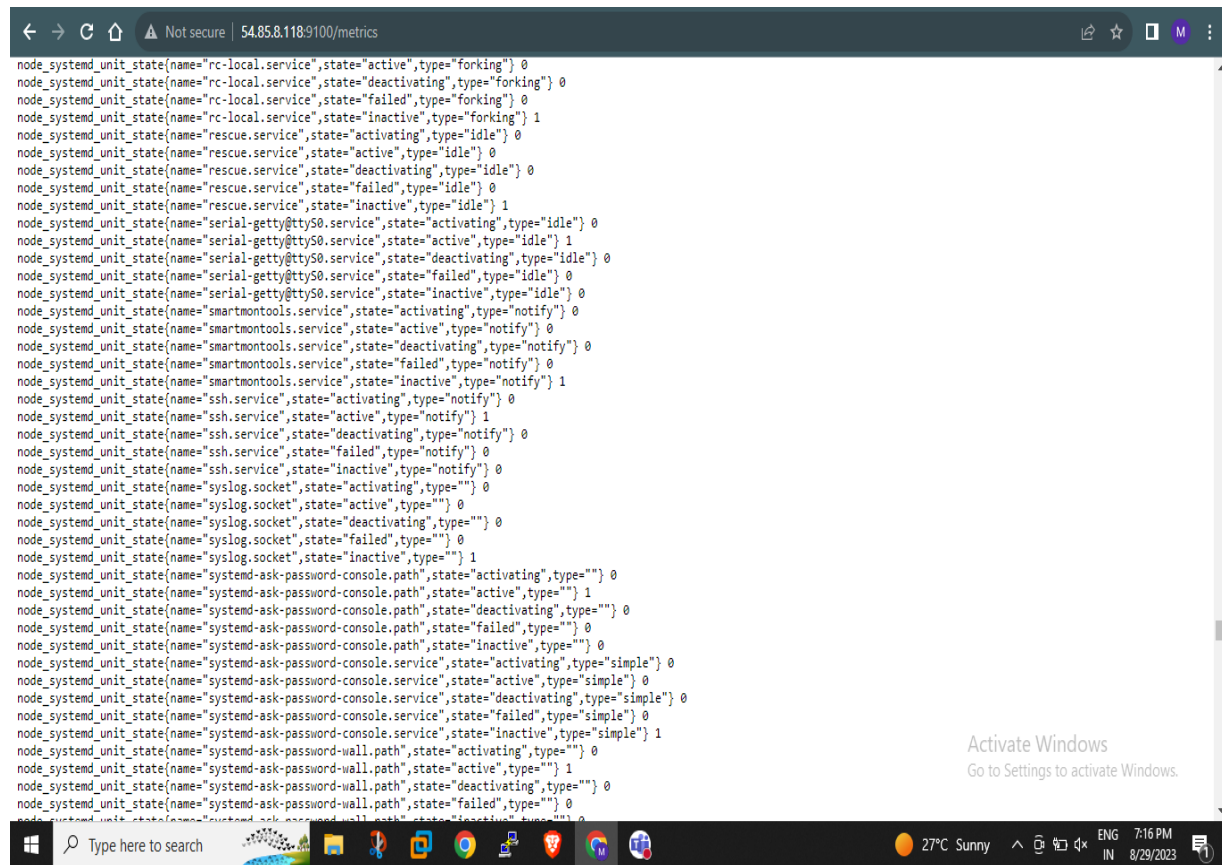
Grafana :





Metric:





```
node_systemd_unit_state(name="rc-local.service",state="active",type="forking") 0
node_systemd_unit_state(name="rc-local.service",state="deactivating",type="forking") 0
node_systemd_unit_state(name="rc-local.service",state="failed",type="forking") 0
node_systemd_unit_state(name="rc-local.service",state="inactive",type="forking") 1
node_systemd_unit_state(name="rescue.service",state="activating",type="idle") 0
node_systemd_unit_state(name="rescue.service",state="active",type="idle") 0
node_systemd_unit_state(name="rescue.service",state="deactivating",type="idle") 0
node_systemd_unit_state(name="rescue.service",state="failed",type="idle") 0
node_systemd_unit_state(name="rescue.service",state="inactive",type="idle") 1
node_systemd_unit_state(name="serial-getty@tty50.service",state="activating",type="idle") 0
node_systemd_unit_state(name="serial-getty@tty50.service",state="active",type="idle") 1
node_systemd_unit_state(name="serial-getty@tty50.service",state="deactivating",type="idle") 0
node_systemd_unit_state(name="serial-getty@tty50.service",state="failed",type="idle") 0
node_systemd_unit_state(name="serial-getty@tty50.service",state="inactive",type="idle") 0
node_systemd_unit_state(name="smartmontools.service",state="activating",type="notify") 0
node_systemd_unit_state(name="smartmontools.service",state="active",type="notify") 0
node_systemd_unit_state(name="smartmontools.service",state="deactivating",type="notify") 0
node_systemd_unit_state(name="smartmontools.service",state="failed",type="notify") 0
node_systemd_unit_state(name="smartmontools.service",state="inactive",type="notify") 1
node_systemd_unit_state(name="ssh.service",state="activating",type="notify") 0
node_systemd_unit_state(name="ssh.service",state="active",type="notify") 1
node_systemd_unit_state(name="ssh.service",state="deactivating",type="notify") 0
node_systemd_unit_state(name="ssh.service",state="failed",type="notify") 0
node_systemd_unit_state(name="ssh.service",state="inactive",type="notify") 0
node_systemd_unit_state(name="syslog.socket",state="activating",type="") 0
node_systemd_unit_state(name="syslog.socket",state="active",type="") 0
node_systemd_unit_state(name="syslog.socket",state="deactivating",type="") 0
node_systemd_unit_state(name="syslog.socket",state="failed",type="") 0
node_systemd_unit_state(name="syslog.socket",state="inactive",type="") 1
node_systemd_unit_state(name="systemd-ask-password-console.path",state="activating",type="") 0
node_systemd_unit_state(name="systemd-ask-password-console.path",state="active",type="") 1
node_systemd_unit_state(name="systemd-ask-password-console.path",state="deactivating",type="") 0
node_systemd_unit_state(name="systemd-ask-password-console.path",state="failed",type="") 0
node_systemd_unit_state(name="systemd-ask-password-console.path",state="inactive",type="") 0
node_systemd_unit_state(name="systemd-ask-password-console.service",state="activating",type="simple") 0
node_systemd_unit_state(name="systemd-ask-password-console.service",state="active",type="simple") 0
node_systemd_unit_state(name="systemd-ask-password-console.service",state="deactivating",type="simple") 0
node_systemd_unit_state(name="systemd-ask-password-console.service",state="failed",type="simple") 0
node_systemd_unit_state(name="systemd-ask-password-console.service",state="inactive",type="simple") 1
node_systemd_unit_state(name="systemd-ask-password-wall.path",state="activating",type="") 0
node_systemd_unit_state(name="systemd-ask-password-wall.path",state="active",type="") 1
node_systemd_unit_state(name="systemd-ask-password-wall.path",state="deactivating",type="") 0
node_systemd_unit_state(name="systemd-ask-password-wall.path",state="failed",type="") 0
node_systemd_unit_state(name="systemd-ask-password-wall.path",state="inactive",type="") 0
```

7. APPLICATIONS

Deploying a web application to a cluster of servers. Jenkins can be used to automate the deployment of a web application to a cluster of servers. This can be done by creating a Jenkins job that builds the application, creates Docker images for the application, and deploys the images to the Docker Swarm cluster.

Load balancing traffic across multiple servers. Docker Swarm can be used to load balance traffic across multiple servers. This can be done by creating a service in Docker Swarm that specifies the number of replicas of the application to run.

Automating the deployment of new features or bug fixes. Jenkins can be used to automate the deployment of new features or bug fixes to a web application. This can be done by creating a Jenkins job that triggers the deployment whenever a new commit is made to the application's source code.

Monitoring the health of the web application. Grafana can be used to monitor the health of the web application. This can be done by creating dashboards in Grafana that display metrics such as CPU usage, memory usage, and network traffic.

Here are some other benefits of using this stack:

It is scalable. The stack can be easily scaled up or down to meet the needs of the application.

It is reliable. The stack is designed to be highly available and resilient to failures.

It is secure. The stack can be configured to use security best practices.

Overall, the combination of Jenkins, Docker Swarm, and Grafana is a powerful tool that can be used to deploy, manage, and monitor web applications.

8. ADVANTAGES & DISADVANTAGES

Advantages:

Scalability: The project uses Docker Swarm, which is a container orchestration platform that can be used to scale applications up or down easily. This makes it a good choice for applications that need to be able to handle a large amount of traffic.

Reliability: The project uses Jenkins, which is a continuous integration and continuous delivery (CI/CD) tool that can be used to automate the deployment process. This helps to ensure that applications are deployed in a consistent and reliable manner.

Security: The project uses Grafana, which is a monitoring tool that can be used to monitor the health of applications. This helps to ensure that applications are running smoothly and that any problems are detected early.

Cost-effectiveness: The project uses open-source tools, which can help to reduce the cost of deployment.

Disadvantages:

Complexity: The project uses a number of different tools, which can make it complex to set up and manage.

Skill requirements: The project requires a certain level of technical skills to set up and manage.

Learning curve: There is a learning curve associated with the project, which can make it difficult for beginners to get started.

Dependency on third-party tools: The project relies on third-party tools, such as Docker Swarm and Grafana. This means that the project is subject to the availability and stability of these tools.

Overall, the project "Web Application Deployment with Load Balancing, Automation, and Monitoring using Jenkins, Docker Swarm, and Grafana" has a number of advantages, such as scalability, reliability, security, and cost-effectiveness. However, it also has some disadvantages, such as complexity, skill requirements, and a learning curve

9. CONCLUSION

The project "Web Application Deployment with Load Balancing, Automation, and Monitoring using Jenkins, Docker Swarm, and Grafana" is a comprehensive solution for deploying web applications. It uses a combination of open-source tools that are scalable, reliable, secure, and cost-effective. However, the project is also complex and requires a certain level of technical skills to set up and manage.

The specific advantages and disadvantages of the project will vary depending on the specific needs of the organization. If the organization is looking for a scalable, reliable, and secure way to deploy web applications, then this project may be a good option. However, if the organization is not comfortable with the complexity or learning curve of the project, then it may be a better option to choose a different approach.

Overall, the project is a valuable resource for organizations that are looking to deploy web applications in a scalable, reliable, and secure manner. However, it is important to carefully consider the specific needs of the organization before deciding whether or not to use the project.

REFERENCES

- [1] Leite, Leonardo, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. (2019) "A survey of DevOps concepts and challenges." *ACM Computing Surveys (CSUR)* 52 (6): 1-35.
- [2] Altunel, Haluk. (2017) "Agile project management in product life cycle." *International Journal of Information Technology Project Management (IJITPM)* 8 (2):50-63.
- [3] Kumudavalli, M. V., and G. Venkatesh. (2021) "Cloud Computing based Monolithic to Containerization using Elastic Container Service for Phylogenetic Analysis." In *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, 1540-1543. IEEE.
- [4] Hu, Tengfei, and Yannian Wang. (2021) "A kubernetes autoscaler based on pod replicas prediction." In *2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, 238-241. IEEE.
- [5] de Aguiar Monteiro, Luciano. (2021) "A proposal to systematize introducing DevOps into the software development process." In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 269-271. IEEE.
- [6] Arouk, Osama, and Navid Nikaein. (2020) "5G cloud-native: Network management & automation." In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, 1-2. IEEE.
- [7] Moravcik, Marek, and Martin Kontsek.(2020) "Overview of Docker container orchestration tools." In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 475-480. IEEE.