

**Mini Project Report**  
**On**  
**“Machine Learning for CyberSecurity”**

Submitted in partial fulfillment for the award of the degree

of

**Third Year**

**In**

**INFORMATION TECHNOLOGY**

Submitted by

- 1. Nehal Janardhan Ingole (19141228)**
- 2. Gaurav Bhausheb Pagare (19141245)**

Under the Guidance of

**Prof. R. S. Mawale**



---

**Government College of Engineering,  
Karad, Maharashtra, India**

**( An Autonomous Institute of Government of Maharashtra)**

**Academic Year 2021 - 2022**

---

**Government College of Engineering,  
Karad, Maharashtra, India**

( An Autonomous Institute of Government of Maharashtra)

**Department of Information Technology**

**CERTIFICATE**

This is to certify that the project entitled “Machine Learning for CyberSecurity” has been carried out by the team:

1. Nehal Janardhan Ingole (19141228)
2. Gaurav Bhausheb Pagare (19141245)

of T. Y. B. Tech IT class under the guidance of Prof. R. S. Mawale during the  
The academic year 2021 -22 (Sem- VI).

**Prof. R. S. Mawale**

**Dr. S. J. Wagh Head**

**Project Guide**

**Head  
Information Technology Department**

**External Examiner**

## **ACKNOWLEDGEMENT**

Apart from individual efforts, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental throughout the project work.

It is our privilege to express our gratitude towards our project guide, Prof. R. S. Mawale for her valuable guidance, encouragement, inspiration and whole-hearted cooperation throughout the project work. We thank him for being a motivation through all our highs and importantly, our lows.

We deeply express our sincere thanks to our Head of Department Dr. S. J. Wagh for encouraging and allowing us to present the project on the topic “Machine Learning for CyberSecurity” and providing us with the necessary facilities to enable us to fulfill our project requirements as best as possible. We take this opportunity to thank all faculty members and staff of the Department of Information Technology, who have directly or indirectly helped our project.

We pay our respects to honorable Principal Dr. A. T. Pise for their encouragement. Our thanks and appreciations also go to our family and friends, who have been a source of encouragement and inspiration throughout the duration of the project

## **ABSTRACT**

This age is the age of Automation, the machines are doing our work, a lot of calculations and managing the big task from general household to big corporates. As the world is developing with the technologies. These all stuff are good but it also comes with great risk of security and one of the most common things which we used on a daily basis is the websites. On the web, the most common types of attacks are DOS-based attacks. This kind attacks is very common and anyone with good knowledge can do these kinds of attacks. These kinds of attack can heavily impact our resource consumption and might take our website down resulting in loss of business

In order to prevent these things, we have developed a project in which we will secure webserver from DoS attacks using Machine Learning. In this project we are using K-mean clustering algorithm to detect the suspicious IP's which are trying to perform such attacks. We will automate the complete flow via the jenkins

Once the suspicious IP's are detected and found we can either hand over them to our security teams, take some action on them or block them from using our website

# LIST OF FIGURES

---

<b>Figure Number</b>	<b>Figure Caption</b>	<b>Page Number</b>
3.1	Elbow method theory 1	12
3.2	Elbow method theory 2	14
3.3	Elbow method theory 3	14
4.1	Arcticture of Dos attack prevention system	15
4.2	Using Jenkins to Establish Same setup	15

# **Table of Content**

---

Abstract	4
List of Figures	5

---

<b>Sr. No.</b>	<b>Table of Contents</b>	<b>Page No.</b>
Chapter 1	Introduction	8
1.1	Background	8
1.2	Importance of the project	8
1.3	Motivation	9
1.4	Scope	9
1.5	Expected Outcome	9
Chapter 2	Literature Survey	10
2.1	K-Means Clustering Approach to Analyze NSL-KDD Intrusion Detection Dataset	10
Chapter 3	Releated Theory and Problem Definatn	11
3.1	Problem Definatn	11
3.2	Releated Theory	11
Chapter 4	Design Methodology	15
4.1	Proposed system Architecture	15
4.2	Internal Logic of the system	16
4.3	Technical Specifications	17
Chapter 5	Implementaion	19

5.1	Understanding the tools and technologies used	19
5.2	Setting up the required environment	22
5.3	Implementation	36
Chapter 6	Result and Declaration	52
6.1	Result	52
6.2	Discussion	52
Chapter 7	Conclusion and Future Scope	53
7.1	Conclusion	53
7.2	Future Scope	53
Ref	Refrence	54

# **Chapter 1**

## **UnderStanding the Project**

### **1.1 Background**

A Denial-of-Service (DoS) attack is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash. In both instances, the DoS attack deprives legitimate users (i.e. employees, members, or account holders) of the service or resource they expected. Victims of DoS attacks often target web servers of high-profile organizations such as banking, commerce, and media companies, or government and trade organizations. Though DoS attacks do not typically result in the theft or loss of significant information or other assets, they can cost the victim a great deal of time and money to handle.

### **1.2 Importance of the project :**

With the implementation of these kinds of projects in the industry environment these DOS attack can be prevented or stoped. Currently the project been designed is capable of detecting the DOS attack but using similar approach models can be created for different kinds of attacks too (tipically for pattern based attacks). The project reduces the task of the security team in the organization as we have automated most of the stuff using jenkins.

### **1.3 Motivation :**

Security have been a major issue in todays digital world organizations does there business via the websites/ webapp. But here we observe there are many kinds of attack / threats on them. Most common is the DOS- attack. This not only consumes resources and energy of the organization but also lead in loss of business. For Security and monitoring team via the manual ways and methodology its very challenging to observe and protect our websites 24 \* 7 from threats and hackers. So here automation comes into picture we can use automation tools and its integration with machine learning to solve these cyber security requirements

### **1.4 Scope :**

- => The approach used for creating machine learning model for the DOS attack using similar approach model can be developed for other pattern based attacks too.
- => Internally for creation of the model we have used the K-mean clustering algorithm (Discussed further in detail). And in this algorithms we need to decide the hyper parameter i.e number of cluster to be formed this is decided by analysis over the data (elbow method). Now we can provide this report of analysis frequently to our data scientist team so that if any changes need in code in future it can be done. This will reduce task of data scientist as well as there will be continuous improvement in the project

### **1.5 Expected Outcome :**

On successful implementation of the project we will be detecting the Suspicious IP as well as blocking those from accessing the website by triggering the firewalld rules.

## **Chapter 2**

### **Literature Survey**

#### **2.1 K-Means Clustering Approach to Analyze NSL-KDD Intrusion Detection Dataset ([LINK](#))**

In this paper the author have present the complete analysis of NSL-KDD Intrusion Detection Dataset using clustering approach. Simple K-Means clustering algorithm is used to identify the different clusters and group them in four major attack categories. They have also provide a complete analysis of different kind of attacks present in training and testing dataset. Testing dataset was used to validate the results. Performance of the algorithm is investigated during different execution of the program on the input data set. The execution time for the algorithm is also analyzed.

# **Chapter 3**

## **Releated Theory and Problem Definatnion**

### **3.1 Problem Definition:**

Our Final Goal is to an automated system which will be very useful for a server in following ways:

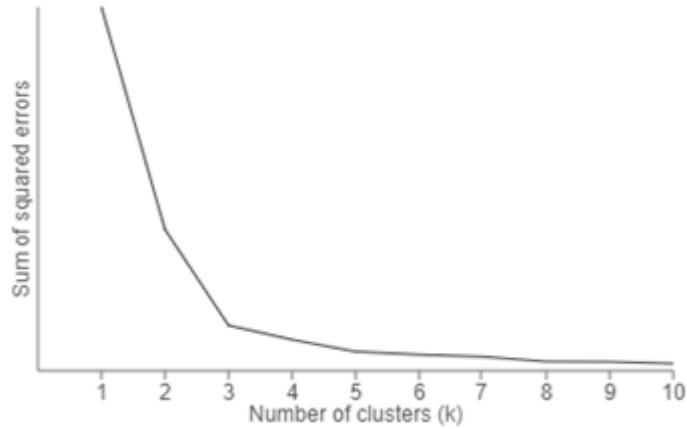
- => This system will keep log of the information about the clients hit or request to the server.
- => This log data of clients will be used for finding the unusual pattern of a client request for example if a client is sending request repeatedly. (DOS -attack)
- => If any kind of unusual pattern we got then we can use Jenkins to perform certain task for example it will trigger the firewall to block that IP which is causing this trouble/problem.

### **3.2 Related Theory:**

Here for detecting the suspicious IP we are using the K-mean clustering algorithim. In the K-means clustering algorithm data points are assigned into K groups, where K represents the number of clusters based on the distance from each group's centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression.

Now in this K-means clustering algorithm one question arises how to decide the value of K One of the methods is called “Elbow” method can be

used to decide an optimal number of clusters. Here you would run K-mean clustering on a range of K values and plot the “percentage of variance explained” on the Y-axis and “K” on X-axis.



### 3.1 Elbow method Theory 1

In the picture above we can see that as we add more clusters after 3 it doesn't give much better modeling on the data. The first cluster adds much information, but at some point, the marginal gain will start dropping.

The sum of squared distance is also called as wcss value and in python we can find this using the inertia\_ keyword. For our requirement each time we can get variety of results but we know there can be at least 2 kind of output suspicious or non suspicious so minimum value of K is 2.

### **3.2.1 Code for finding value of K:**

---

```
inertias = []

K = range(2,len(training_data)+1)

for k in K:

    kmeanModel = KMeans(n_clusters=k)

    kmeanModel.fit(data_scaled)

    inertias.append(kmeanModel.inertia_)

plt.plot(K, inertias, 'o-')

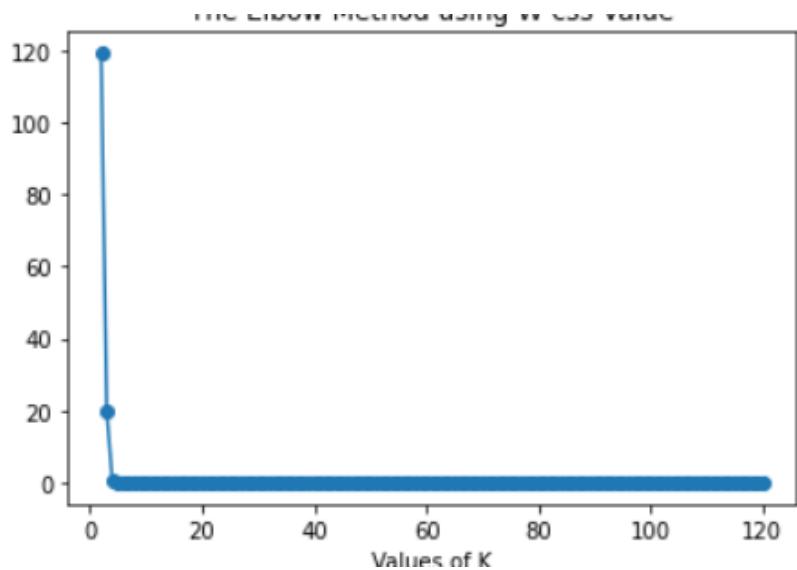
plt.xlabel('Values of K')

plt.ylabel('W-css')

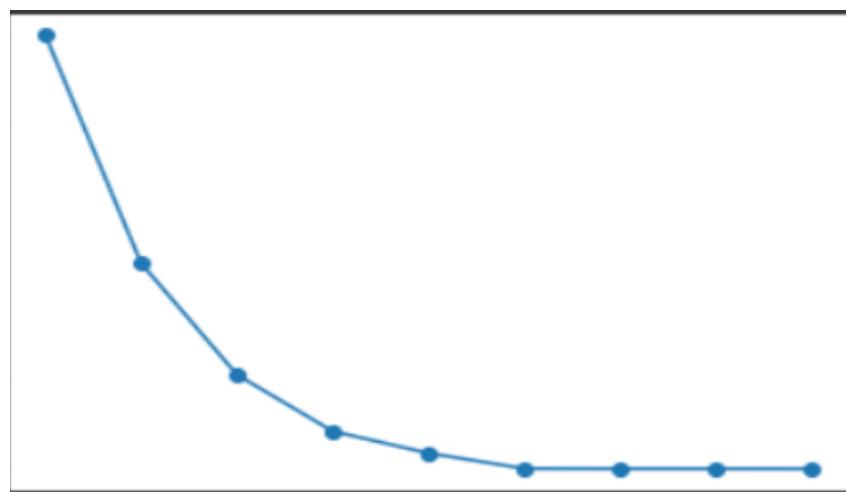
plt.title('The Elbow Method using W-css Value')

plt.show()
```

---



### 3.2 Elbow method Theory 2

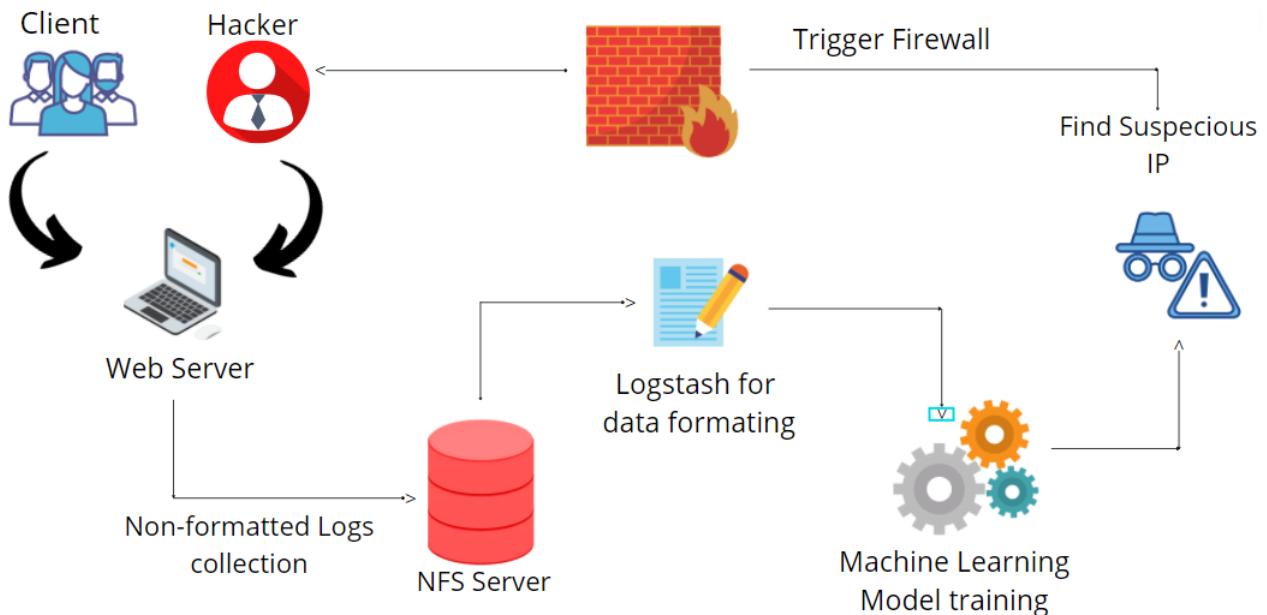


### 3.3 Elbow method Theory 3

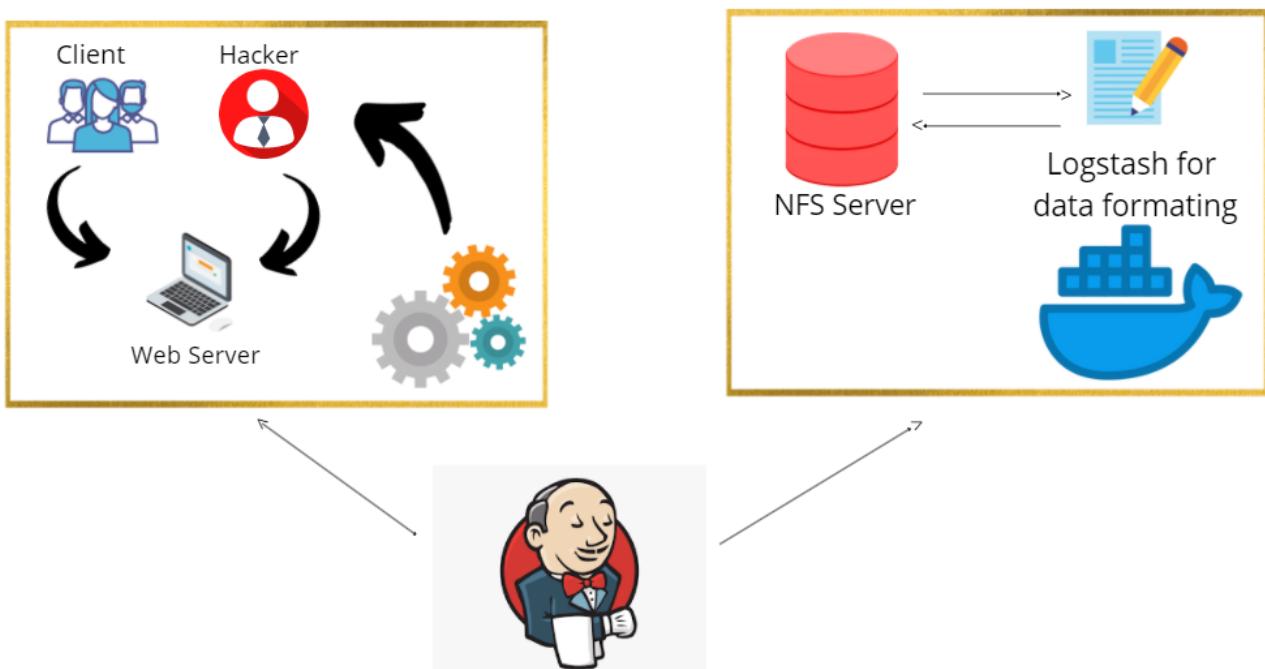
# Chapter 4

## Design Methodology

### 4.1 Proposed system Architecture:



4.1 Arcticture of Dos attack prevention system



4.2 Using Jenkins to Establish Same setup

## **4.2 Internal Logic of the System:**

- => At the most we will be hosting our demo website using the apache httpd web server
- => In order to replicate the real world scenario we will hit our webserver as genuine client and with one of the virtual machine we will try to perform DOS attack over the web server
- => Now apache web server generates the logs which we are storing in the NFS server so that those log can be accessed by other teams in the organization
- => The logs collected and present in the NFS can't be directly given to the machine learning model so for this purpose we will use logstash for this transformation operation
- => Now we will give the formatted logs (csv file) to the machine learning code. Now the model will be created with this code
- => further over the data depending on the number of hits done by the client in a particular time span i.e approx >3000 requests within an hour. If such cases are identified they will be considered to be suspicious
- => List of this suspicious IP will be used and firewalld rule will be created to block those IP from using the website.
- => Ok now we know the final architecture which we need to achieve. Now we can do the same this with automation using jenkins
- => In the jenkins we have one master node setup and 2 nodes i.e one is web server and other is the one with docker and nfs running
- => In the jenkins we will be configuring the multiple jobs i.e one job to get the raw data from webserver to the nfs server. Further once that job is

done other job will be automatically get triggered i.e to launch a logstash docker container.

=> Further we will be training our model via another job getting triggered after logstash job is done. The output generated will be the the list of suspicious IP. this will be stored in BlockedIP.txt file (discussed further)

=> Now using the suspicious IP we will be triggering the firewall rule

### **4.3 Technical Specifications:**

Hardware Requirements: (Note all the nodes are virtual machines)

=> Jenkins-Master Node

---

RAM	8GB and Above
HDD	50 GB
Image	RHEL-8
Processor	Intel® Core™ i5

=> Web-Server (Ec2- instance)

Image (AMI)	RHEL-8
Instance-type	T2.micro (1 CPU, 1 GB ram)

=> Docker-NFS-server (Ec2- instance)

Image (AMI)	RHEL-8
Instance-type	T2.large (2 CPU, 8 GB ram)

Software Requirements (dependencies for the softwares and some common s/w eg. vim, git, yum-utils, etc if not available is installed further):

**1. Jenkins - Master**

- a. Java-OpenJDK
- b. Jenkins package
- c. Python3.6

**2. Web-Server**

- a. Httpd package
- b. Java-OpenJDK
- c. Python3.6
  - i. Os-sys module
  - ii. Time module

**3. Docker-NFS-server**

- a. NFS-server package
- b. Docker-ce package
- c. Java-OpenJDK
- d. Python3.6
  - i. Skit learn module
  - ii. Numpy

**4. Windows Machine**

- a. Putty
- b. HTML/CSS
- c. Browser (Google Chrome)

# **Chapter 5**

## **Implementation**

### **5.1 Understanding the tools and technologies used:**

Python :

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

HTML , CSS :

HTML (Hypertext Markup Language) is responsible for the structure of a web page. For example, you can define headings, paragraphs, texts, and images in HTML. CSS (Cascading Style Sheets) is responsible for the style and layout of a web page. You can define styles, such as colors, fonts, margins, and you can even create simple animations in CSS. Both

languages, HTML and CSS, are independent and should be saved in separate files.

Putty:

Putty is a free implementation of SSH (and telnet) for PCs running Microsoft Windows (it also includes an xterm terminal emulator). You will find PuTTY useful if you want to access an account on a Unix or other multi-user system from a PC. Here we have used it to connect to our jenkins master as well as the instances running in the AWS cloud

Amazon Web Server

AWS (Amazon Web Services) is a comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings. AWS services can offer organization tools such as compute power, database storage and content delivery services.

Jenkins

Jenkins is used to build and test your product continuously, so developers can continuously integrate changes into the build. Jenkins is

the most popular open source CI/CD tool on the market today and is used in support of DevOps, alongside other cloud native tools.

## Docker

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid multicloud environments.

## Httpd

httpd is the Apache HyperText Transfer Protocol (HTTP) server program. It is designed to be run as a standalone daemon process. When used like this it will create a pool of child processes or threads to handle requests.

## NFS Services:

NFS services uses Network File Sharing (NFS) is a protocol that allows you to share directories and files with other Linux clients over a network. Shared directories are typically created on a file server, running

the NFS server component. Users add files to them, which are then shared with other users who have access to the folder. An NFS file share is mounted on a client machine, making it available just like folders the user created locally. NFS is particularly useful when disk space is limited and you need to exchange public data between client computers

## 5.2 Setting up the required Environment:

Step 1: Launch 2 ec2-instances on AWS instance with RHEL-8 AMI. Also one instance with t2.micro machine-type and other with t2.large machine-type . Here we are going to use instance with t2.micro machine-type for web-services and Instance for t2.large machine-type for Docker and NFS server. We have also tagged them with same names too.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
Web-Server	i-08c28c42e34f7deb1	t2.micro	ap-south-1a	running	2/2 checks passed	None	ec2-13-232-174-175.ap...	13.232.174.175
Docker-n-nfs	i-0a049de0b8f1d91a4	t2.large	ap-south-1a	running	2/2 checks passed	None	ec2-13-232-222-148.ap...	13.232.222.148

Step 2: Login to the instances with the ec2-user as username and authentication key. Now for easy access we will first allow the root login via password in both the instances for this we need to do changes in /etc/ssh/sshd\_config file. Here we need to mention PasswordAuthincation yes by default its no.

```
[root@ip-172-31-33-111:~]# Using username "ec2-user".
[root@ip-172-31-33-111:~]# Authenticating with public key "imported-openssh-key"
[ec2-user@ip-172-31-33-111 ~]$ sudo su - root
[root@ip-172-31-33-111 ~]# vim /etc/ssh/sshd_config
```

```
[root@ip-172-31-33-111:~]#
# Authentication:
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile .ssh/authorized_keys

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
>PasswordAuthentication no
>PasswordAuthentication yes
```

←

```
# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes
ChallengeResponseAuthentication no
```

Step 3: Now we also need to set new root password for this we can use the passwd command as shown below. Further Restart the sshd service

```
[root@ip-172-31-33-111 ~]# passwd root  
Changing password for user root.  
New password:  
BAD PASSWORD: The password fails the dictionary check - it is based on a dictionary word  
Retype new password:  
passwd: all authentication tokens updated successfully.  
[root@ip-172-31-33-111 ~]#
```

```
root@ip-172-31-33-111:~  
[ec2-user@ip-172-31-33-111 ~]$ sudo su - root  
Last login: Tue Oct 26 18:21:10 UTC 2021 on pts/1  
[root@ip-172-31-33-111 ~]# vim /etc/ssh/sshd_config  
[root@ip-172-31-33-111 ~]# systemctl restart sshd  
[root@ip-172-31-33-111 ~]#
```

Note: We need to perform step 2 and 3 in both of the ec2-instances

Step 4: Now as these instances are going to be jenkins worker nodes so we need java installed in both the instance.

```

root@ip-172-31-33-111:~#
[root@ip-172-31-33-111 ~]# yum install java -y
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Last metadata expiration check: 0:10:43 ago on Tue 26 Oct 2021 05:59:43 PM UTC.
Dependencies resolved.
=====
Package           Architecture Version      Repository   Size
=====
Installing:
java-1.8.0-openjdk          x86_64    1:1.8.0.312.b07-1.el8_4      rhel-8-appstream-rhui-rpms 337 k
Installing dependencies:
alsa-lib                    x86_64    1.2.4-5.el8                rhel-8-appstream-rhui-rpms 471 k
atk                         x86_64    2.28.1-1.el8               rhel-8-appstream-rhui-rpms 272 k
cairo                       x86_64    1.15.12-3.el8              rhel-8-appstream-rhui-rpms 721 k
copy-jdk-configs            noarch   3.7-4.el8                 rhel-8-appstream-rhui-rpms 27 k
fontconfig                  x86_64    2.13.1-3.el8               rhel-8-baseos-rhui-rpms 275 k
fribidi                     x86_64    1.0.4-8.el8                rhel-8-appstream-rhui-rpms 89 k
gdk-pixbuf2                 x86_64    2.36.12-5.el8              rhel-8-baseos-rhui-rpms 467 k
gdk-pixbuf2-modules         x86_64    2.36.12-5.el8              rhel-8-appstream-rhui-rpms 109 k
giflib                      x86_64    5.1.4-3.el8                rhel-8-appstream-rhui-rpms 51 k
graphite2                   x86_64    1.3.10-10.el8              rhel-8-appstream-rhui-rpms 122 k
gtk-update-icon-cache       x86_64    3.22.30-6.el8              rhel-8-appstream-rhui-rpms 32 k
harfbuzz                     x86_64    1.7.5-3.el8                rhel-8-appstream-rhui-rpms 294 k
hicolor-icon-theme          noarch   0.17-2.el8                 rhel-8-appstream-rhui-rpms 48 k
jasper-libs                  x86_64    2.0.14-4.el8               rhel-8-appstream-rhui-rpms 167 k
java-1.8.0-openjdk-headless x86_64    1:1.8.0.312.b07-1.el8_4      rhel-8-appstream-rhui-rpms 34 M
javapackages-filesystem     noarch   5.3.0-1.module+el8+2447+6f56d9a6 rhel-8-appstream-rhui-rpms 30 k
jbigkit-libs                 x86_64    2.1-14.el8                 rhel-8-appstream-rhui-rpms 55 k
libX11                      x86_64    1.6.8-4.el8                rhel-8-appstream-rhui-rpms 611 k
libX11-common                noarch   1.6.8-4.el8                rhel-8-appstream-rhui-rpms 158 k
libXau                      x86_64    1.0.9-3.el8                rhel-8-appstream-rhui-rpms 37 k
libXcomposite                x86_64    0.4.4-14.el8               rhel-8-appstream-rhui-rpms 29 k
libXcursor                   x86_64    1.1.15-3.el8               rhel-8-appstream-rhui-rpms 36 k

[root@ip-172-31-33-111 ~]# java -version
openjdk version "1.8.0_312"
OpenJDK Runtime Environment (build 1.8.0_312-b07)
OpenJDK 64-Bit Server VM (build 25.312-b07, mixed mode)
[root@ip-172-31-33-111 ~]#

```

Step 5: Now lets move toward configuration of the web server. For this we need to install the httpd software. Here I have installed more packages also like git and vim

```

root@ip-172-31-33-111:~#
[root@ip-172-31-33-111 ~]# yum install httpd vim git -y
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Last metadata expiration check: 0:02:17 ago on Tue 26 Oct 2021 05:59:43 PM UTC.
Dependencies resolved.
=====
Package           Architecture Version      Repository   Size
=====
Installing:
git              x86_64       2.27.0-1.el8    rhel-8-appstream-rhui-rpms 164 k
httpd            x86_64       2.4.37-39.module+el8.4.0+12865+a7065a39.1 rhel-8-appstream-rhui-rpms 1.4 M
vim-enhanced     x86_64       2:8.0.1763-15.el8  rhel-8-appstream-rhui-rpms 1.4 M
Installing dependencies:
apr              x86_64       1.6.3-11.el8   rhel-8-appstream-rhui-rpms 125 k
apr-util          x86_64       1.6.1-6.el8   rhel-8-appstream-rhui-rpms 105 k
emacs-filesystem noarch      1:26.1-5.el8   rhel-8-baseos-rhui-rpms 69 k
git-core          x86_64       2.27.0-1.el8   rhel-8-appstream-rhui-rpms 5.7 M
git-core-doc      noarch      2.27.0-1.el8   rhel-8-appstream-rhui-rpms 2.5 M
gpm-libs          x86_64       1.20.7-17.el8  rhel-8-appstream-rhui-rpms 39 k
httpd-filesystem noarch      2.4.37-39.module+el8.4.0+12865+a7065a39.1 rhel-8-appstream-rhui-rpms 39 k
httpd-tools        x86_64       2.4.37-39.module+el8.4.0+12865+a7065a39.1 rhel-8-appstream-rhui-rpms 106 k
mailcap            noarch      2.1.48-3.el8   rhel-8-baseos-rhui-rpms 39 k
mod http2          x86_64       1.15.7-3.module+el8.4.0+8625+d397f3da rhel-8-appstream-rhui-rpms 154 k
perl-Error         noarch      1:0.17025-2.el8  rhel-8-appstream-rhui-rpms 46 k
perl-Git            noarch      2.27.0-1.el8   rhel-8-appstream-rhui-rpms 78 k
perl-TermReadKey  x86_64       2.37-7.el8    rhel-8-appstream-rhui-rpms 40 k
redhat-logos-httpd noarch      84.4-1.el8    rhel-8-baseos-rhui-rpms 29 k
vim-common         x86_64       2:8.0.1763-15.el8 rhel-8-appstream-rhui-rpms 6.3 M
vim-filesystem     noarch      2:8.0.1763-15.el8 rhel-8-appstream-rhui-rpms 48 k
Installing weak dependencies:
apr-util-bdb       x86_64       1.6.1-6.el8   rhel-8-appstream-rhui-rpms 25 k
apr-util-openssl  x86_64       1.6.1-6.el8   rhel-8-appstream-rhui-rpms 27 k

```

Step 6: Now we want our demo website to be configured for this we used our git repo which we created

(<https://github.com/Ingole712521/Industrial-Training>). Here we have cloned this repo in /var/www/html i.e document root of httpd

```

root@ip-172-31-33-111:~#
[root@ip-172-31-33-111 ~]# git clone https://github.com/Ingole712521/Industrial-Training-.git /var/www/html/
Cloning into '/var/www/html'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 11 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (11/11), 51.81 KiB | 10.36 MiB/s, done.
[root@ip-172-31-33-111 ~]#

```

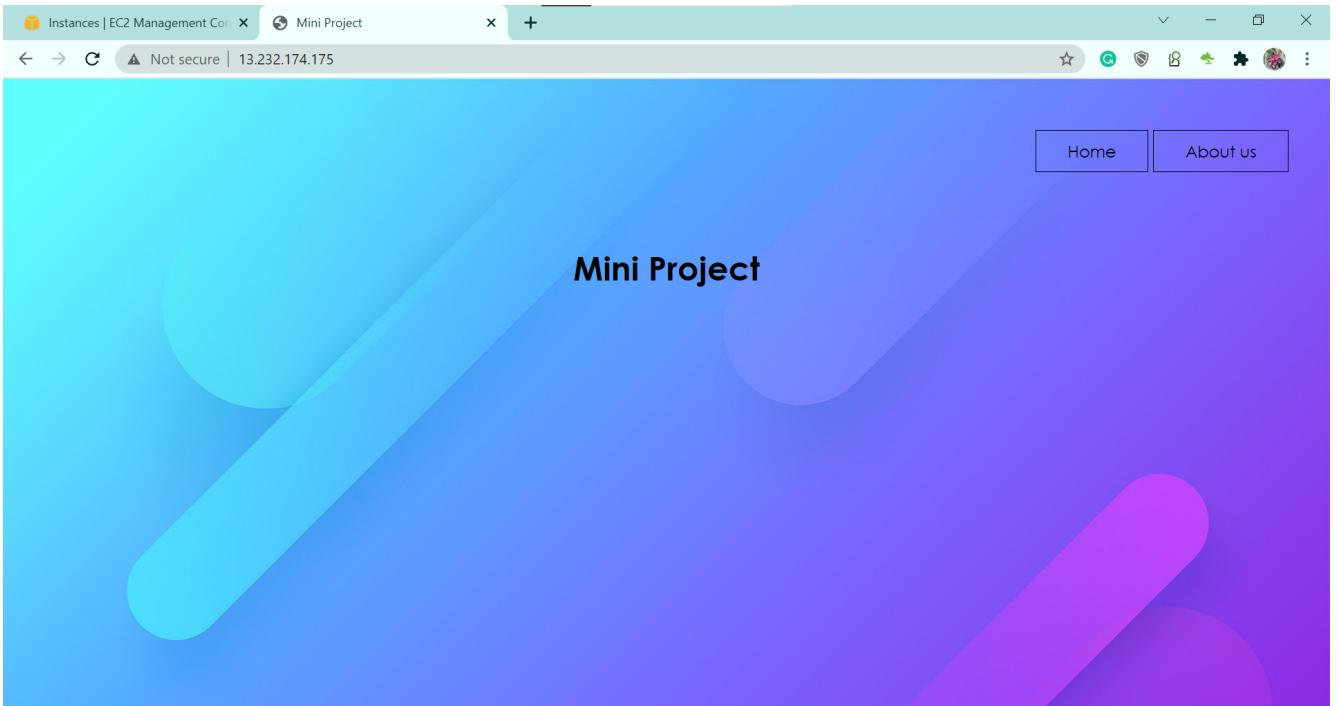
Step 7: Further we have started and enabled the services for httpd. Also in httpd we have the logs been collected at /etc/httpd/logs. Root user have the permission to view them.

```

root@ip-172-31-33-111:~# systemctl start httpd
[root@ip-172-31-33-111 ~]# systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[root@ip-172-31-33-111 ~]#

```

## Webpage:



## Logs:

```

root@ip-172-31-33-111:/etc/httpd/logs
[root@ip-172-31-33-111 ~]# cd /etc/httpd/logs/
[root@ip-172-31-33-111 logs]# ls
access_log error_log
[root@ip-172-31-33-111 logs]# cat access_log
152.57.170.125 - - [26/Oct/2021:18:09:15 +0000] "GET / HTTP/1.1" 200 514 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36"
152.57.170.125 - - [26/Oct/2021:18:09:15 +0000] "GET /style.css HTTP/1.1" 200 612 "http://13.235.103.159/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36"
152.57.170.125 - - [26/Oct/2021:18:09:15 +0000] "GET /iamge.jpg HTTP/1.1" 200 43505 "http://13.235.103.159/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36"
152.57.170.125 - - [26/Oct/2021:18:09:16 +0000] "GET /favicon.ico HTTP/1.1" 404 196 "http://13.235.103.159/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36"
[root@ip-172-31-33-111 logs]# cat access_log | wc -l
4
[root@ip-172-31-33-111 logs]#

```

Step 8: Here we have launch the web-instance in the aws and aws uses its security group as the firewalld. So in the os we need to separately install and start the firewall

```
root@ip-172-31-33-111:~# [root@ip-172-31-33-111 ~]# yum install firewalld
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Red Hat Update Infrastructure 3 Client Configuration Server 8 HA
Red Hat Enterprise Linux 8 for x86_64 - AppStream from RHUI (RPMs)
Red Hat Enterprise Linux 8 for x86_64 - AppStream from RHUI (RPMs)
Red Hat Enterprise Linux 8 for x86_64 - BaseOS from RHUI (RPMs)
Red Hat Enterprise Linux 8 for x86_64 - BaseOS from RHUI (RPMs)
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) from RHUI
Dependencies resolved.

=====
Package           Architecture     Version      Repository   Size
=====
Installing:
firewalld        noarch         0.8.2-7.el8_4    rhel-8-baseos-rhui-rpms 489 k
Installing dependencies:
firewalld-filesystem  noarch         0.8.2-7.el8_4    rhel-8-baseos-rhui-rpms 77 k
ipset             x86_64          7.1-1.el8     rhel-8-baseos-rhui-rpms 45 k
ipset-libs        x86_64          7.1-1.el8     rhel-8-baseos-rhui-rpms 71 k
iptables          x86_64          1.8.4-17.el8   rhel-8-baseos-rhui-rpms 586 k
iptables-ebtables x86_64          1.8.4-17.el8   rhel-8-baseos-rhui-rpms 71 k
libnetfilter_conntrack x86_64          1.0.6-5.el8   rhel-8-baseos-rhui-rpms 65 k
libnfnfnetlink    x86_64          1.0.1-13.el8   rhel-8-baseos-rhui-rpms 33 k
libnftnl          x86_64          1.1.5-4.el8   rhel-8-baseos-rhui-rpms 83 k
nftables          x86_64          1:0.9.3-18.el8  rhel-8-baseos-rhui-rpms 313 k
python3-firewall  noarch         0.8.2-7.el8_4    rhel-8-baseos-rhui-rpms 394 k
python3-nftables  x86_64          1:0.9.3-18.el8  rhel-8-baseos-rhui-rpms 27 k
python3-slip       noarch         0.6.4-11.el8   rhel-8-baseos-rhui-rpms 39 k
python3-slip-dbus  noarch         0.6.4-11.el8   rhel-8-baseos-rhui-rpms 39 k

Transaction Summary
=====
Install 14 Packages

=====
[root@ip-172-31-33-111 mini_project]# yum install iptables-services
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Last metadata expiration check: 0:24:37 ago on Wed 27 Oct 2021 11:07:46 AM UTC.
Dependencies resolved.

=====
Package           Architecture     Version      Repository   Size
=====
Installing:
iptables-services x86_64          1.8.4-17.el8   rhel-8-baseos-rhui-rpms 62 k

Transaction Summary
=====
Install 1 Package

Total download size: 62 k
Installed size: 20 k
Is this ok [y/N]: y
Downloading Packages:
iptables-services-1.8.4-17.el8.x86_64.rpm
                                                               658 kB/s | 62 kB   00:00
Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing : 1/1
  Installing : iptables-services-1.8.4-17.el8.x86_64 1/1
  Running scriptlet: iptables-services-1.8.4-17.el8.x86_64 1/1
  Verifying  : iptables-services-1.8.4-17.el8.x86_64 1/1
  Installed products updated.

Installed:
  iptables-services-1.8.4-17.el8.x86_64

Complete!
[root@ip-172-31-33-111 mini_project]#
```

Further we also need to allow hits over port 80 for this we can use the command :

```
# firewall-cmd --zone=public --add-port=80/tcp  
--permanent  
  
# firewall-cmd --reload
```

Now lets move toward configuring the Docker and nfs servers in other ec2-instance of t2.large machine-type

Step 9: For having the nfs setup we need to install nfs-utils package. Now we want to have one folder /share1 where my nfs client can put there data and we can also access it directly simply like accessing a normal directory. For this we need to first create a folder say /share1 now we need to mention in the nfs configuration file (/etc/exports) which folder is the shared one. Format for mentioning this is

```
# <folder name> <client_ip> (permissions)
```

Further we also need to start the nfs services. We can view our shared folders using the command

```
# exposefs -v
```

```

root@ip-172-31-37-229:~# [root@ip-172-31-37-229 ~]# yum install nfs-utils -y
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Last metadata expiration check: 0:12:49 ago on Tue 26 Oct 2021 06:18:59 PM UTC.
Package nfs-utils-1:2.3.3-41.el8.x86_64 is already installed.
Dependencies resolved.

=====
| Package           | Architecture | Version      | Repository   | Size |
|=====             |=====         |=====        |=====         |===== |
| Upgrading:       |              |              |              |       |
| nfs-utils        | x86_64      | 1:2.3.3-41.el8_4.2 | rhel-8-baseos-rhui-rpms | 498 k |
| Transaction Summary |
| Upgrade 1 Package |
| Total download size: 498 k
| Downloading Packages:
| nfs-utils-2.3.3-41.el8_4.2.x86_64.rpm | 7.0 MB/s | 498 kB | 00:00 |
| Total | 5.8 MB/s | 498 kB | 00:00 |
| Running transaction check |
| Transaction check succeeded. |
| Running transaction test |
| Transaction test succeeded. |
| Running transaction |
|  Preparing : |
| Running scriptlet: nfs-utils-1:2.3.3-41.el8_4.2.x86_64 | 1/1 |
| Running scriptlet: nfs-utils-1:2.3.3-41.el8_4.2.x86_64 | 1/2 |
| Upgrading : nfs-utils-1:2.3.3-41.el8_4.2.x86_64 | 1/2 |
| Running scriptlet: nfs-utils-1:2.3.3-41.el8_4.2.x86_64 | 1/2 |
| Running scriptlet: nfs-utils-1:2.3.3-41.el8.x86_64 | 2/2 |
| Cleanup : nfs-utils-1:2.3.3-41.el8.x86_64 | 2/2 |

```

```

root@ip-172-31-37-229:~# [root@ip-172-31-37-229 ~]# mkdir /share1
[root@ip-172-31-37-229 ~]# vim /etc/exports
[root@ip-172-31-37-229 ~]# cat /etc/exports
/share1 13.235.103.159(rw,no_root_squash)
[root@ip-172-31-37-229 ~]# systemctl start nfs-server
[root@ip-172-31-37-229 ~]# exportfs -v
/share1 13.235.103.159(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no root squash,no all squash)
[root@ip-172-31-37-229 ~]#

```

```

[root@ip-172-31-37-229 ~]# systemctl start nfs-server
[root@ip-172-31-37-229 ~]# systemctl enable nfs-server
[root@ip-172-31-37-229 ~]# systemctl status nfs-server
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor preset: disabled)
   Drop-In: /run/systemd/generator/nfs-server.service.d
             └─order-with-mounts.conf
   Active: active (exited) since Tue 2021-10-26 18:35:23 UTC; 9min ago
     Main PID: 17654 (code=exited, status=0/SUCCESS)
       Tasks: 0 (limit: 23524)
      Memory: 0B
        CGroup: /system.slice/nfs-server.service

Oct 26 18:35:23 ip-172-31-37-229.ap-south-1.compute.internal systemd[1]: Starting NFS server and services...
Oct 26 18:35:23 ip-172-31-37-229.ap-south-1.compute.internal systemd[1]: Started NFS server and services.
[root@ip-172-31-37-229 ~]#

```

**Step 10:** Now lets setup the docker in the system. For this we need to first get the repository for the docker added in the system (for this we need yum-utils package too). And then we can install the docker-ce package

```

root@ip-172-31-37-229:~#
[root@ip-172-31-37-229 ~]# yum install yum-utils -y
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Last metadata expiration check: 0:17:39 ago on Tue 26 Oct 2021 06:18:59 PM UTC.
Package yum-utils-4.0.18-4.el8.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-37-229 ~]# yum-config-manager \
>   --add-repo \
>     https://download.docker.com/linux/rhel/docker-ce.repo
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Adding repo from: https://download.docker.com/linux/rhel/docker-ce.repo
[root@ip-172-31-37-229 ~]#

```

```

root@ip-172-31-37-229:~#
[root@ip-172-31-37-229 ~]# dnf install docker-ce
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

Docker CE Stable - x86_64
Dependencies resolved.

=====
Package           Arch      Version            Repository      Size
=====
Installing:
docker-ce         x86_64    3:20.10.10-3.el8      docker-ce-stable 22 M
Installing dependencies:
container-selinux  noarch    2:2.167.0-1.module+el8.4.0+12646+b6fd1bdf  rhel-8-appstream-rhui-rpms 52 k
containerd.io     x86_64    1.4.11-3.1.el8       docker-ce-stable 28 M
docker-ce-cli     x86_64    1:20.10.10-3.el8      docker-ce-stable 29 M
docker-ce-rootless-extras x86_64    20.10.10-3.el8      docker-ce-stable 4.6 M
docker-scan-plugin x86_64    0.9.0-3.el8        docker-ce-stable 3.7 M
fuse-common       x86_64    3.2.1-12.el8       rhel-8-baseos-rhui-rpms 21 k
fuse-overlayfs   x86_64    1.6-1.module+el8.4.0+11822+6cc1e7d7  rhel-8-appstream-rhui-rpms 73 k
fuse3             x86_64    3.2.1-12.el8       rhel-8-baseos-rhui-rpms 50 k
fuse3-libs        x86_64    3.2.1-12.el8       rhel-8-baseos-rhui-rpms 94 k
iptables          x86_64    1.8.4-17.el8       rhel-8-baseos-rhui-rpms 586 k
libcgroup          x86_64    0.41-19.el8       rhel-8-baseos-rhui-rpms 70 k
libnetfilter_conntrack x86_64    1.0.6-5.el8       rhel-8-baseos-rhui-rpms 65 k
libnfnetwork      x86_64    1.0.1-13.el8      rhel-8-baseos-rhui-rpms 33 k
libnftnl           x86_64    1.1.5-4.el8       rhel-8-baseos-rhui-rpms 83 k
libslirp            x86_64    4.3.1-1.module+el8.4.0+11822+6cc1e7d7  rhel-8-appstream-rhui-rpms 69 k
policycoreutils-python-utils x86_64    2.9-14.el8        rhel-8-baseos-rhui-rpms 252 k
slirp4netns        x86_64    1.1.8-1.module+el8.4.0+11822+6cc1e7d7  rhel-8-appstream-rhui-rpms 51 k
Enabling module streams:
container-tools      rhel8

Transaction Summary
=====
```

Exposing the docker port (2376) so that we can access it from outside world too.

```

root@ip-172-31-37-229:~#
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket containerd.service

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0:2376
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd 229.
# Both the old, and new location are accepted by systemd 229 and up, so using the old location
# to make them work for either version of systemd.
StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.
# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make
# this option work for either version of systemd.
StartLimitInterval=60s

# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

# Comment TasksMax if your systemd version does not support it.
-- INSERT --

```

## Starting the docker services

```

root@ip-172-31-37-229:~#
[root@ip-172-31-37-229 ~]# vim /usr/lib/systemd/system/docker.service
[root@ip-172-31-37-229 ~]# systemctl start docker
[root@ip-172-31-37-229 ~]# systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-172-31-37-229 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2021-10-26 18:43:45 UTC; 13s ago
     Docs: https://docs.docker.com
 Main PID: 20128 (dockerd)
    Tasks: 8
   Memory: 33.6M
      CGroup: /system.slice/docker.service
              └─20128 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0:2376

Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.213907231Z" level=warning msg=">>>
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.213954688Z" level=warning msg=">>>
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.214190241Z" level=info msg="Load
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.430759226Z" level=info msg="Def>
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.513010725Z" level=info msg="Load
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.542410870Z" level=info msg="Doc>
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.542685676Z" level=info msg="Dae>
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.597130616Z" level=info msg="API>
Oct 26 18:43:45 ip-172-31-37-229.ap-south-1.compute.internal dockerd[20128]: time="2021-10-26T18:43:45.607080116Z" level=info msg="API>
[root@ip-172-31-37-229 ~]# 

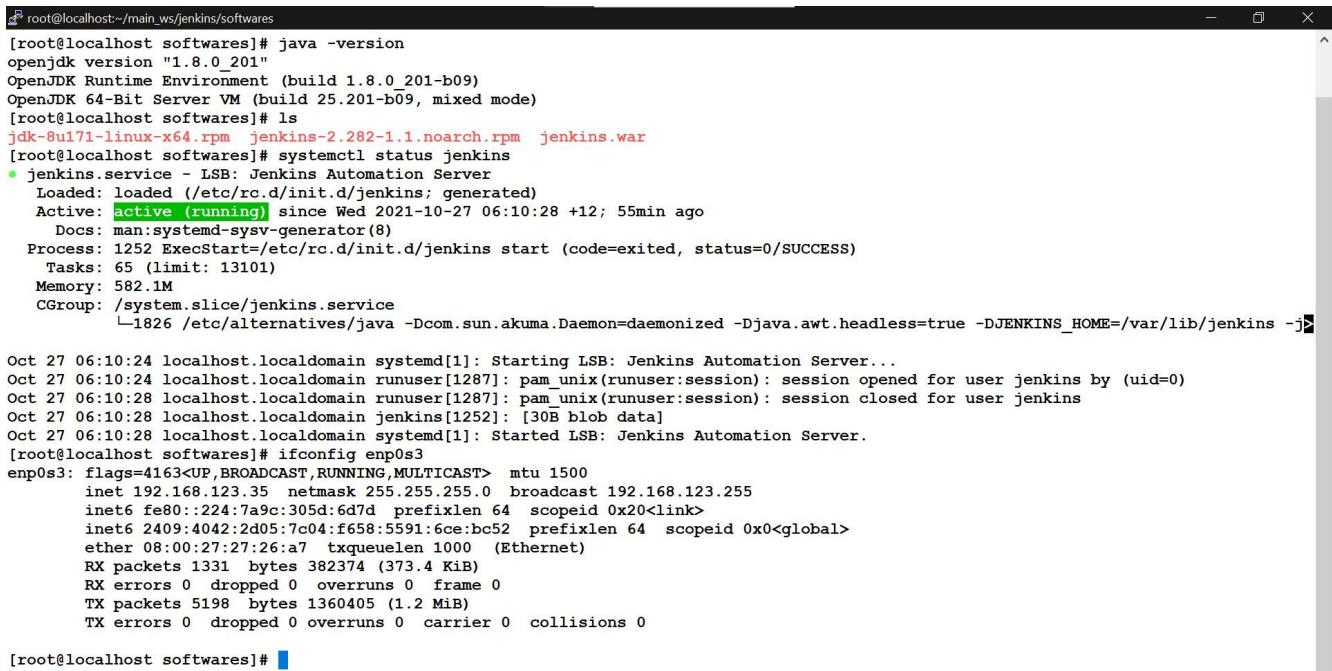
```

Step 11: Now lets configure the jenkins here in our case it is running over the locally. Here we need java to run jenkins further for the installation we

also need the jenkins rpm which we can get over the internet for the installation we can then simply use the command

```
# rpm -ivh <package_name>
```

Further we need to start and enable the services of jenkins



```
root@localhost:~/main_ws/jenkins/softwares [root@localhost softwares]# java -version
openjdk version "1.8.0_201"
OpenJDK Runtime Environment (build 1.8.0_201-b09)
OpenJDK 64-Bit Server VM (build 25.201-b09, mixed mode)
[root@localhost softwares]# ls
jdk-8u171-linux-x64.rpm jenkins-2.282-1.noarch.rpm jenkins.war
[root@localhost softwares]# systemctl status jenkins
● jenkins.service - LSB: Jenkins Automation Server
  Loaded: loaded (/etc/rc.d/init.d/jenkins; generated)
  Active: active (running) since Wed 2021-10-27 06:10:28 +12; 55min ago
    Docs: man:systemd-sysv-generator(8)
  Process: 1252 ExecStart=/etc/rc.d/init.d/jenkins start (code=exited, status=0/SUCCESS)
   Tasks: 65 (limit: 13101)
  Memory: 582.1M
  CGroup: /system.slice/jenkins.service
          └─1826 /etc/alternatives/java -Dcom.sun.akuma.Daemon=daemonized -Djava.awt.headless=true -DJENKINS_HOME=/var/lib/jenkins -j

Oct 27 06:10:24 localhost.localdomain systemd[1]: Starting LSB: Jenkins Automation Server...
Oct 27 06:10:24 localhost.localdomain runuser[1287]: pam_unix(runuser:session): session opened for user jenkins by (uid=0)
Oct 27 06:10:28 localhost.localdomain runuser[1287]: pam_unix(runuser:session): session closed for user jenkins
Oct 27 06:10:28 localhost.localdomain jenkins[1252]: [30B blob data]
Oct 27 06:10:28 localhost.localdomain systemd[1]: Started LSB: Jenkins Automation Server.
[root@localhost softwares]# ifconfig enp0s3
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.123.35 netmask 255.255.255.0 broadcast 192.168.123.255
          inet6 fe80::224:7a9c:305d:6d7d prefixlen 64 scopeid 0x20<link>
        inet6 2409:4042:2d05:7c04:f658:5591:6ce:bc52 prefixlen 64 scopeid 0x0<global>
          ether 08:00:27:27:a7 txqueuelen 1000 (Ethernet)
          RX packets 1331 bytes 382374 (373.4 KiB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 5198 bytes 1360405 (1.2 MiB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@localhost softwares]#
```

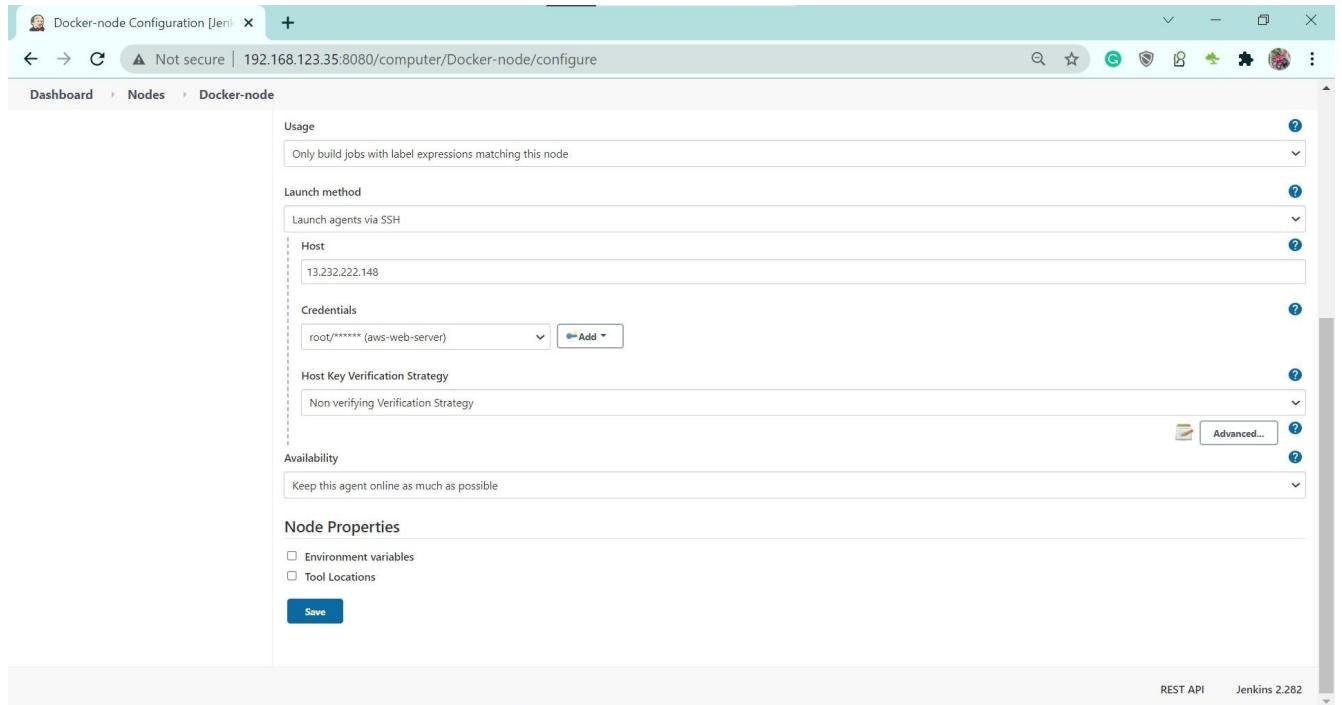
Jenkins by default runs on port number 8080.

The screenshot shows the Jenkins interface for the 'mini\_project' view. On the left, a sidebar contains links for 'New Item', 'People', 'Build History', 'Edit View', 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'New View'. Below this is a 'Build Queue' section stating 'No builds in the queue.' The main content area has tabs 'All' and 'mini\_project' (which is selected), with a '+' button. A message indicates 'This view has no jobs associated with it. You can either add some existing jobs to this view or create a new job in this view.' There is also a 'add description' link.

Steps 12: Now lets configure the nodes (worker nodes for the jenkins)

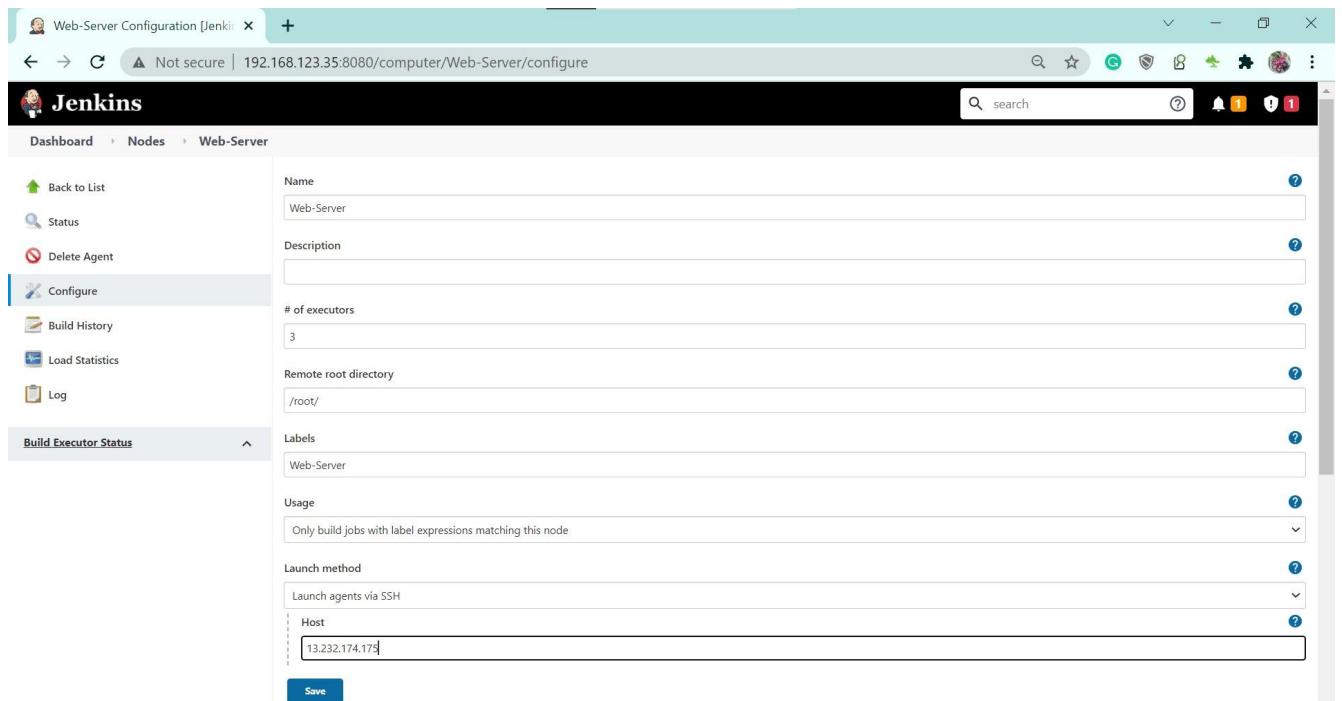
Docker-nfs node:

The screenshot shows the Jenkins 'Docker-node Configuration' page. The left sidebar includes 'Back to List', 'Status', 'Delete Agent', and 'Configure' (which is selected). Other options like 'Build History', 'Load Statistics', and 'Log' are also listed. The main form fields are: 'Name' (Docker-node), 'Description' (empty), '# of executors' (2), 'Remote root directory' (/root/), 'Labels' (Docker-node), 'Usage' (Only build jobs with label expressions matching this node), 'Launch method' (Launch agents via SSH), and 'Host' (IP address 13.232.222.148). A 'Save' button is at the bottom.



The screenshot shows the configuration page for a Jenkins node named 'Docker-node'. The page includes sections for Usage, Launch method, Credentials, Availability, Node Properties, and a Save button. The 'Launch method' section is set to 'Launch agents via SSH' with a host IP of '13.232.222.148'. The 'Credentials' section lists 'root/\*\*\* (aws-web-server)' with an 'Add' button. The 'Availability' section has the option 'Keep this agent online as much as possible' checked. The 'Node Properties' section contains checkboxes for 'Environment variables' and 'Tool Locations'. A 'Save' button is located at the bottom.

## Web-Server Node:



The screenshot shows the configuration page for a Jenkins node named 'Web-Server'. The page includes sections for Name, Description, # of executors, Remote root directory, Labels, Usage, Launch method, and a Save button. The 'Name' field is set to 'Web-Server'. The 'Description' field is empty. The '# of executors' field is set to '3'. The 'Remote root directory' field is set to '/root/'. The 'Labels' field is set to 'Web-Server'. The 'Usage' field is empty. The 'Launch method' section is set to 'Launch agents via SSH' with a host IP of '13.232.174.175'. A 'Build Executor Status' section is visible on the left.

The screenshot shows the Jenkins configuration page for a node named 'Web-Server'. The configuration includes:

- Usage:** Only build jobs with label expressions matching this node.
- Launch method:** Launch agents via SSH.
- Host:** 13.232.174.175
- Credentials:** root/\*\*\*\*\*\*\*\* (aws-web-server)
- Host Key Verification Strategy:** Non verifying Verification Strategy.
- Availability:** Keep this agent online as much as possible.
- Node Properties:** Environment variables and Tool Locations.
- Save** button.

## Status of the nodes:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Docker-node	Linux (amd64)	1.4 sec ahead	6.34 GB	0 B	6.34 GB	226ms
	master	Linux (amd64)	In sync	36.45 GB	2.03 GB	36.45 GB	0ms
	Web-Server	Linux (amd64)	1.4 sec ahead	8.13 GB	0 B	8.13 GB	2238ms
	Data obtained	0.56 sec	0.53 sec	0.46 sec	0.44 sec	0.44 sec	0.42 sec

Build Queue: No builds in the queue.

Build Executor Status:

- master: 1 Idle, 2 Idle
- Docker-node: 0 Idle

## 5.3 Implementaion:

Step 1: At the most we will be configuring the jenkins jobs i.e Move logs to nfs folder. (The Job is restricted to be running on web-server node only)  
Now via this job we are moving the log file from /etc/http/logs to

/share1/current\_logs. Further we are also reloading the httpd services. Once this is done we will copy the logs in /share1/current\_logs to /share1/all\_logs with appropriate time-stamp. So that in future if we need to see past logs we can refer these logs

**General**

Description:

[Plain text] Preview

- Commit agent's Docker container
- Define a Docker template
- Discard old builds
- GitHub project
- Delivery Pipeline configuration
- This project is parameterized
- Throttle builds
- Disable this project
- Execute concurrent builds if necessary
- Restrict where this project can be run

Label Expression:

Web-Server

Label Web-Server matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

**Source Code Management**

Save Apply Advanced...

**General**

**Build**

Execute shell

Command:

```
mv -f /etc/httpd/logs/* /share1/current_logs/
apachectl graceful
sleep 30
cp /share1/current_logs/access_log /share1/all_logs/access_log-$(date +%F)-$(date +%T)
cp /share1/current_logs/error_log /share1/all_logs/error_log-$(date +%F)-$(date +%T)
```

See the list of available environment variables Advanced...

Add build step ▾

**Post-build Actions**

Add post-build action ▾

Save Apply

Step 2: Now we will be configuring our second job i.e launch logstash via docker (This job is restricted to be ran on the Docker-nfs-node only). The job will be triggered only if previous job (Move logs to nfs folder) was build successful. Via this job we will first check if already formatted old logs are present if yes it will remove those then proceed. Further the job will launch an logstash container. Now logstash takes a min or two to generate its output i.e (web-server-logs.csv) file. For this we will wait till its there once the file is generated we don't need the container so we will remove the logstash container.

(Note: make shure that the volume mounted /mini\_project/logstash have read write and execute powers for all or docker user if need we can give those via

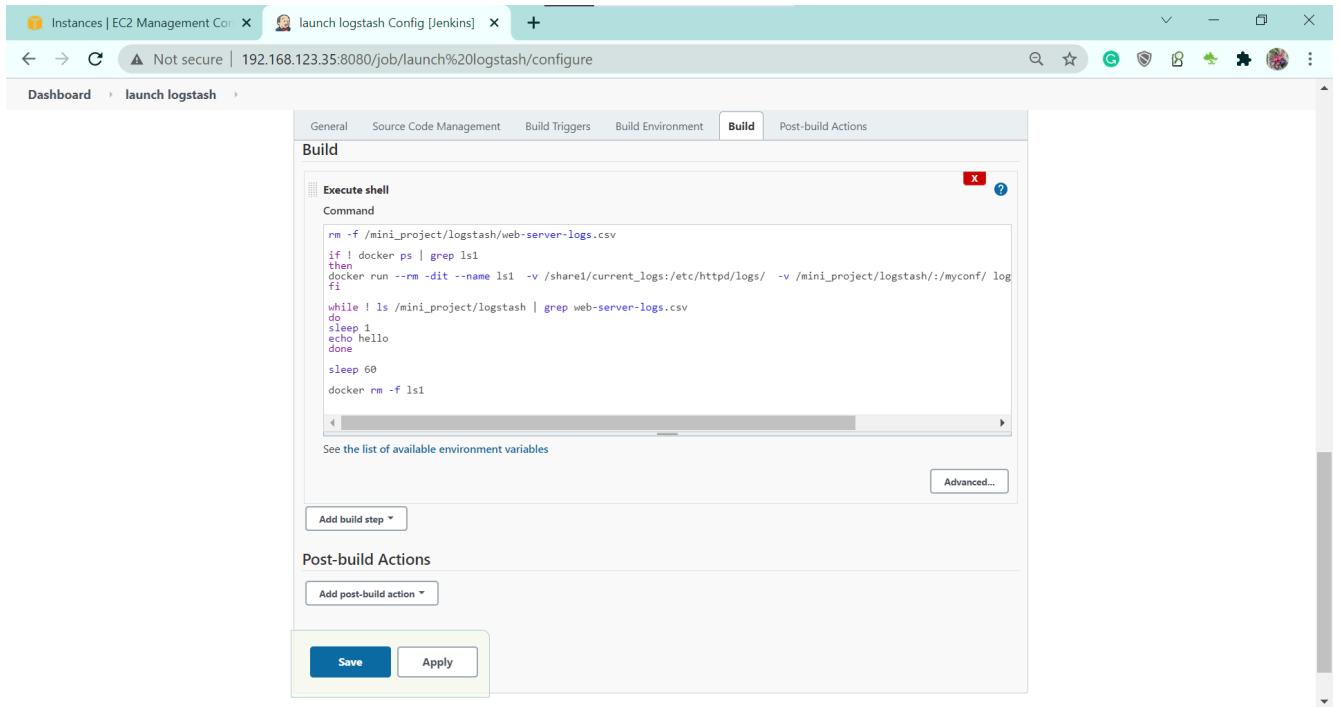
```
# chmod ugo+rwx /mini_project/logstash
```

)

The screenshot shows the Jenkins 'General' configuration page for a job named 'launch logstash'. The 'General' tab is selected. Key settings include:

- Restrict where this project can be run:** The 'Label Expression' field contains 'Docker-node'. A note below states: 'Label Docker-node matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.' An 'Advanced...' button is available.
- Source Code Management:** The 'None' option is selected.
- Build Triggers:**
  - Build after other projects are built:** This checkbox is checked. The 'Projects to watch' dropdown contains 'Move logs to nfs folder'.
  - Trigger options:** The 'Trigger only if build is stable' radio button is selected.
  - Build periodically:** This checkbox is unchecked.

At the bottom of the page are 'Save' and 'Apply' buttons.



Here in the logstash we are having our custom requirements i.e we want logstash to take input from a particular file we want logstash to give output in particular destination. Here we need to use custom configuration file. (hence we have used -f option to pass that custom conf file)

```

root@ip-172-31-37-229:~/mini_project/logstash

input {
  file {
    id      => "web-server-logs-input"
    path   => ["/etc/httpd/logs/access_*"]
    start_position => beginning
  }
}

filter {
  grok {
    match => ["message" , "%{COMBINEDAPACHELOG}"]
  }
  mutate {
    convert => {
      "response" => "integer"
    }
  }
  date {
    match      => [ "timestamp" , "dd/MMM/YYYY:HH:mm:ss Z" ]
    locale     => en
    remove_field => "timestamp"
  }
}

output {
  csv {
    id      => "web-server-logs-output"
    fields => ["clientip","response","bytes","@timestamp"]
    path   => "/myconf/web-server-logs.csv"
  }
}
~
~
"my.conf" 33L, 573C

```

In the logstash for almost any kind of requirement we use the plugins here we had the requirement to filer apache logs. We used grok plugin for this grok internally uses the regular expression to find and filter required data from raw data. Grok have many other keyword too eg. if we need to find IP in the file we can use %IP:client.

Further in similar way to retrieve the status code and timestamp we used the mutate and data plugin. Also via the input and output we have mention the input destination and the output destination with required format

Launching logstash via docker:

```
# docker run --rm -dit --name ls1
-v /share1/current_logs:/etc/httpd/logs/
-v /mini_project/logstash:/myconf/
logstash:7.7.1 -f /myconf/my.conf
```

With the use of -f option we are telling location of custom conf file. Also the output file generated will be in /myconf/ but we have mounted it to /mini\_project/logstash so at the end we can see that in this folder too.

Step 3: Now here we will be configuring our next job i.e Find BlockedIP. Via this job we will be getting the suspicious IP. (This job is restricted to be ran in Docker-nfs-node only). This job will be triggered once the previous job i.e launch logstash will be build successfully. In this job once the blockedIP.txt file is created it is moved to /share1/BlockedIP folder so that my web server can use this to block those IP

The screenshot shows the Jenkins job configuration page for 'Find BlockedIP'. The job is named 'Find BlockedIP Config [Jenkins]'. The 'General' tab is selected, showing the following configuration:

- Execute concurrent builds if necessary
- Restrict where this project can be run  
Label Expression: Docker-node
- Label Docker-node matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

The 'Source Code Management' section is set to 'None'.

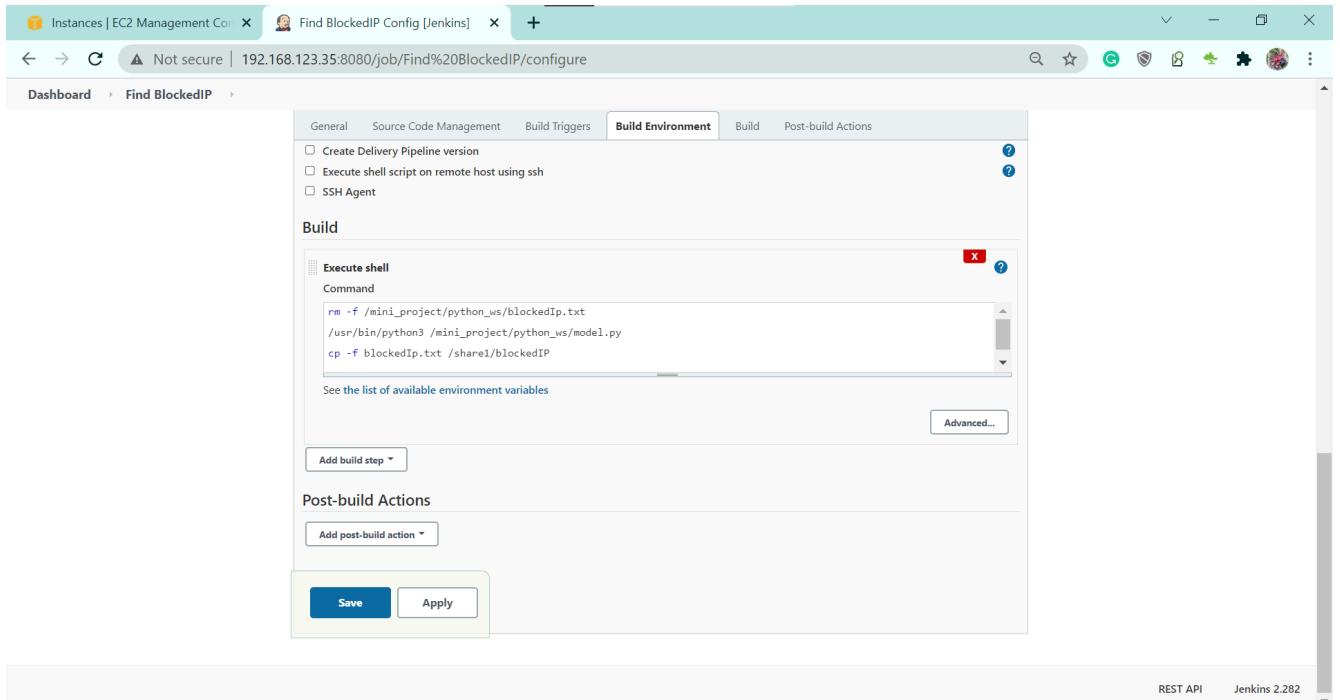
The 'Build Triggers' section includes:

- Build after other projects are built  
Projects to watch: launch logstash.
- Trigger only if build is stable
- Trigger even if the build is unstable
- Trigger even if the build fails

Other trigger options shown but unchecked:

- Build periodically
- GitHub hook trigger for GITScm polling
- Poll SCM

At the bottom of the configuration page are two buttons: 'Save' and 'Apply'.



## Model.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

ds =
pd.read_csv('/mini_project/logstash/web-server-logs.csv',header=None)
ds.columns =
['IP','Status_code','Bytes_trf','TimeStamp']
ds = ds[['IP','Status_code','TimeStamp']]
ds.info()

ds = ds[['IP','Status_code']]
ds = ds.groupby(['IP',
'Status_code']).Status_code.agg('count').to_frame('Count').reset_index()
ds = ds.sort_values(['Count'],ascending=False)
ds.head(20)
```

```

training_data = ds.drop(['IP'],axis=1)
training_data.head(20)

sc = StandardScaler()
data_scaled = sc.fit_transform(training_data)

model=KMeans(n_clusters=2)
pred=model.fit_predict(data_scaled)

final_dataset = ds
final_dataset["Cluster"] = pred
final_dataset.head()

a = []
for index,row in ds.iterrows():
    if row['Count'] > 500:
        a.append(row['Cluster'])
blockedCluster = max(set(a), key = a.count)

blockedIP = []
for index,row in ds.iterrows():
    if row['Cluster'] == blockedCluster:
        blockedIP.append(row['IP'])

for b_ip in blockedIP:
    print("Blocked IP: {}".format(b_ip))

blockedIP=set(blockedIP)

with open('blockedIp.txt', 'w') as filehandle:
    for listitem in blockedIP:
        filehandle.write('%s\n' % listitem)

```

Now in this code we have use the k-mean clustering algorithm after multiple test we have found that depending on the number of hits its good to have at 2 clusters as static for now.

Once the model is trained we can use it and using it we have created the blockedIP.txt file compromising of the BlockedIP

Step 4: Now we come to our last job (this job is restricted to be ran in web-server node only). Via this job we are only asking the internal firewall running in the webserver to block the suspicious IP.

The screenshot shows the Jenkins configuration interface for a job named 'block IP's Config'. The 'General' tab is selected. Under 'Label Expression', the value 'Web-Server' is specified. In the 'Source Code Management' section, 'Git' is chosen. Under 'Build Triggers', 'Build after other projects are built' is selected, with 'Find BlockedIP' as the project to watch. The 'Build' tab is visible at the bottom.

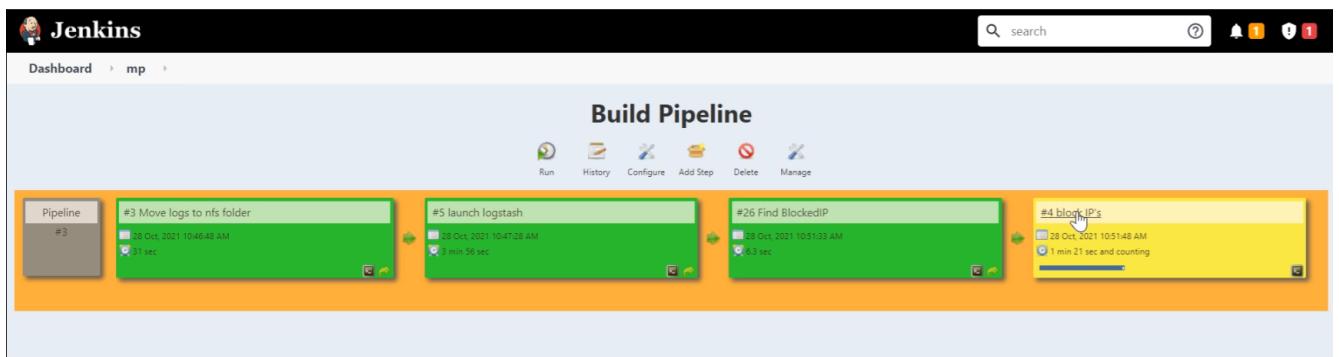
The screenshot shows the Jenkins configuration interface for the same job, focusing on the 'Build Environment' tab. Under 'Build', there is a single step: 'Execute shell' with the command '/usr/bin/python3 /mini\_project/ipblock.py'. The 'Post-build Actions' section is empty. The 'Build' tab is visible at the bottom.

## Ipblock.py

```
from os import system
with open("/share1/blockedIP/blockedIp.txt","r") as file1:
    for i in file1:
        system('iptables -A INPUT -s {} -j
DROP'.format(i.rstrip('\n')))
    system('service iptables save')
```

Here after this packets receiving from these suspicious IP will be dropped and not hit the server.

Step 5: Now lets create the view in jenkins for the build pipeline



This is created because we have set in the proper way for each job which one is there upstream job and which one is there down stream job

Step 6: Now from my windows machine with the help of a bash script we will be hitting the web server with multiple fake/ dummy request and try to increase the load over the server. Before or during this process many genuin clients might have also hit the webserver

```
#!/bin/bash
while true
do
```

```
curl http://13.232.174.175/gaurav.html  
curl http://13.232.174.175/gaurav2001.html  
curl http://13.232.174.175/jack.html  
curl http://13.232.174.175/nehal.html  
curl http://13.232.174.175/asdasfadfsf.html  
curl http://13.232.174.175/asdadw.html
```

done

IP address of attacker:

The screenshot shows a web browser window with two tabs: "Instances | EC2 Management Con" and "What Is My IP Address - See Your". The main content is from "WhatIsMyIPAddress.com". It displays the IP address 2409:4042:2d05:7c04:5d51:37ae:fdc0:2996 (IPv6) and 152.57.186.218 (IPv4). Below this, it shows ISP: Jio, City: Nagpur, Maharashtra. To the right, there is a map of Maharashtra with a pin pointing to a location. A tooltip on the map shows the IP address 2409:4042:2d05:7c04:5d51:37ae:fdc0:2996. At the bottom, there is an advertisement for Trailhead.

Now Lets trigger the pipeline and see the console outputs of the job

## Move logs to nfs folder:

The screenshot shows a Jenkins console output page for a build named "Move logs to nfs folder #3". The build was triggered by a user and is running as SYSTEM. The log output shows the command to move access and error logs from /etc/httpd/logs to /share1/current\_logs, followed by a sleep command and a timestamp. It then copies the logs to /share1/all\_logs and triggers a new build of "launch logstash". The build finished successfully.

```
Started by user unknown or anonymous
Running as SYSTEM
Building remotely on Web-Server in workspace /root/workspace/Move logs to nfs folder
[Move logs to nfs folder] $ /bin/sh -xe /tmp/jenkins3280525418256019915.sh
+ mv -f /etc/httpd/logs/access_log /etc/httpd/logs/error_log /share1/current_logs/
+ apachectl graceful
+ sleep 30
++ date +%F
++ date +%T
+ cp /share1/current_logs/access_log /share1/all_logs/access_log-2021-10-27-22:47:20
++ date +%F
++ date +%T
+ cp /share1/current_logs/error_log /share1/all_logs/error_log-2021-10-27-22:47:20
Triggering a new build of launch logstash
Finished: SUCCESS
```

## launch logstash:

The screenshot shows a Jenkins console output page for a build named "launch logstash #5". The build was triggered by an upstream project and is running as SYSTEM. The log output shows the configuration of Logstash to read from /mini\_project/logstash/web-server-logs.csv and write to /etc/httpd/logs/. It then runs a Docker container with Logstash and prints "hello" to the logs. The build finished successfully.

```
Started by upstream project "Move logs to nfs folder" build number 3
originally caused by:
Started by user unknown or anonymous
Running as SYSTEM
Building remotely on Docker-node in workspace /root/workspace/launch logstash
[launch logstash] $ /bin/sh -xe /tmp/jenkins5476337551760017444.sh
+ rm -f /mini_project/logstash/web-server-logs.csv
+ grep ls1
+ docker ps
+ docker run --rm -dit --name ls1 -v /share1/current_logs:/etc/httpd/logs/ -v /mini_project/logstash:/myconf/ logstash:7.7.1 -f /myconf/my.conf
5606985fb86d1fa2727749b32908eaa0da3450c8485ec0570291bb324cf693b9
+ ls /mini_project/logstash
+ grep web-server-logs.csv
+ sleep 1
+ echo hello
hello
+ ls /mini_project/logstash
+ grep web-server-logs.csv
+ sleep 1
+ echo hello
hello
+ grep web-server-logs.csv
+ ls /mini_project/logstash
+ sleep 1
+ echo hello
hello
```

```

+ echo hello
hello
+ grep web-server-logs.csv
+ ls /mini_project/logstash
+ sleep 1
+ echo hello
hello
+ grep web-server-logs.csv
+ ls /mini_project/logstash
+ sleep 1
+ echo hello
hello
+ grep web-server-logs.csv
+ ls /mini_project/logstash
+ sleep 1
+ echo hello
hello
+ ls /mini_project/logstash
+ grep web-server-logs.csv
+ sleep 1
+ echo hello
hello
+ grep web-server-logs.csv
+ ls /mini_project/logstash
+ sleep 1
+ echo hello
hello
+ grep web-server-logs.csv
+ ls /mini_project/logstash
+ sleep 1
+ docker rm -f ls1
ls1
Triggering a new build of Find BlockedIP
Finished: SUCCESS

```

## Find BlockedIP:

**Console Output**

```

Started by upstream project "launch logstash" build number 5
originally caused by:
Started by upstream project "Move logs to nfs folder" build number 3
originally caused by:
    Started by user unknown or anonymous
Running as SYSTEM
Building remotely on Docker-node in workspace /root/workspace/Find BlockedIP
[Find BlockedIP] $ /bin/sh -c /tmp/jenkins5430206451406772359.sh
+ rm -f /mini_project/python_ws/blockedIp.txt
+ /usr/bin/python3 /mini_project/python_ws/model.py
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          -----          --    
 0   IP          1115 non-null   object 
 1   Status_code 1115 non-null   int64  
 2   TimeStamp   1115 non-null   object 
 dtypes: int64(1), object(2)
memory usage: 26.3+ KB
Blocked IP: 152.57.186.218
+ cp -f blockedIp.txt /share1/blockedIP
Triggering a new build of block IP's
Finished: SUCCESS

```

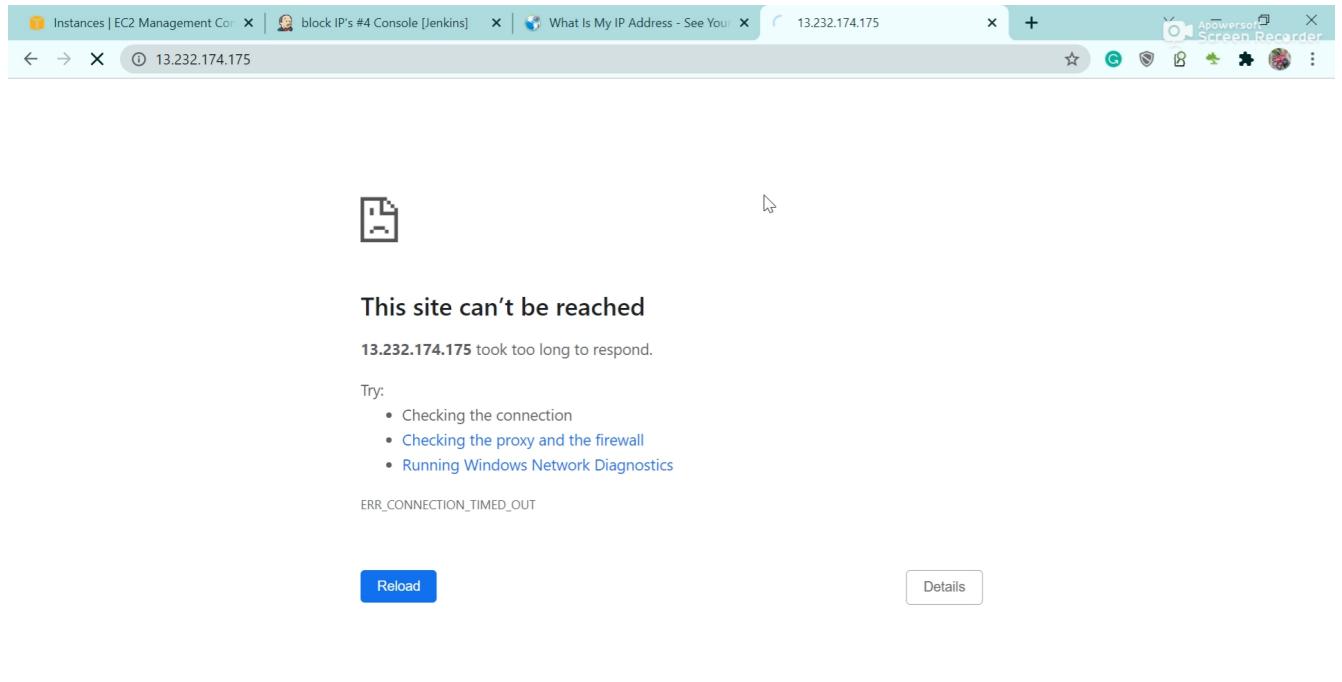
From here we can see it have detected the windows machine IP as suspicious IP.

block IP's:

The screenshot shows a Jenkins job named "block IP's #4 Console [Jenkins]" running on port 8080. The console output displays the following log entries:

```
Started by upstream project "Find BlockedIP" build number 26
originally caused by:
Started by upstream project "launch logstash" build number 5
originally caused by:
Started by upstream project "Move logs to nfs folder" build number 3
originally caused by:
Started by user unknown or anonymous
Running as SYSTEM
Building remotely on Web-Server in workspace /root/workspace/block IP's
[block IP's] $ /bin/sh -xe /tmp/jenkins3223754099714481798.sh
+ /usr/bin/python3 /mini_project/ipblock.py
iptables: Saving firewall rules to /etc/sysconfig/iptables: [  OK  ]
```

Now as the firewall rules are saved so lets check the connectivity via the windows machine

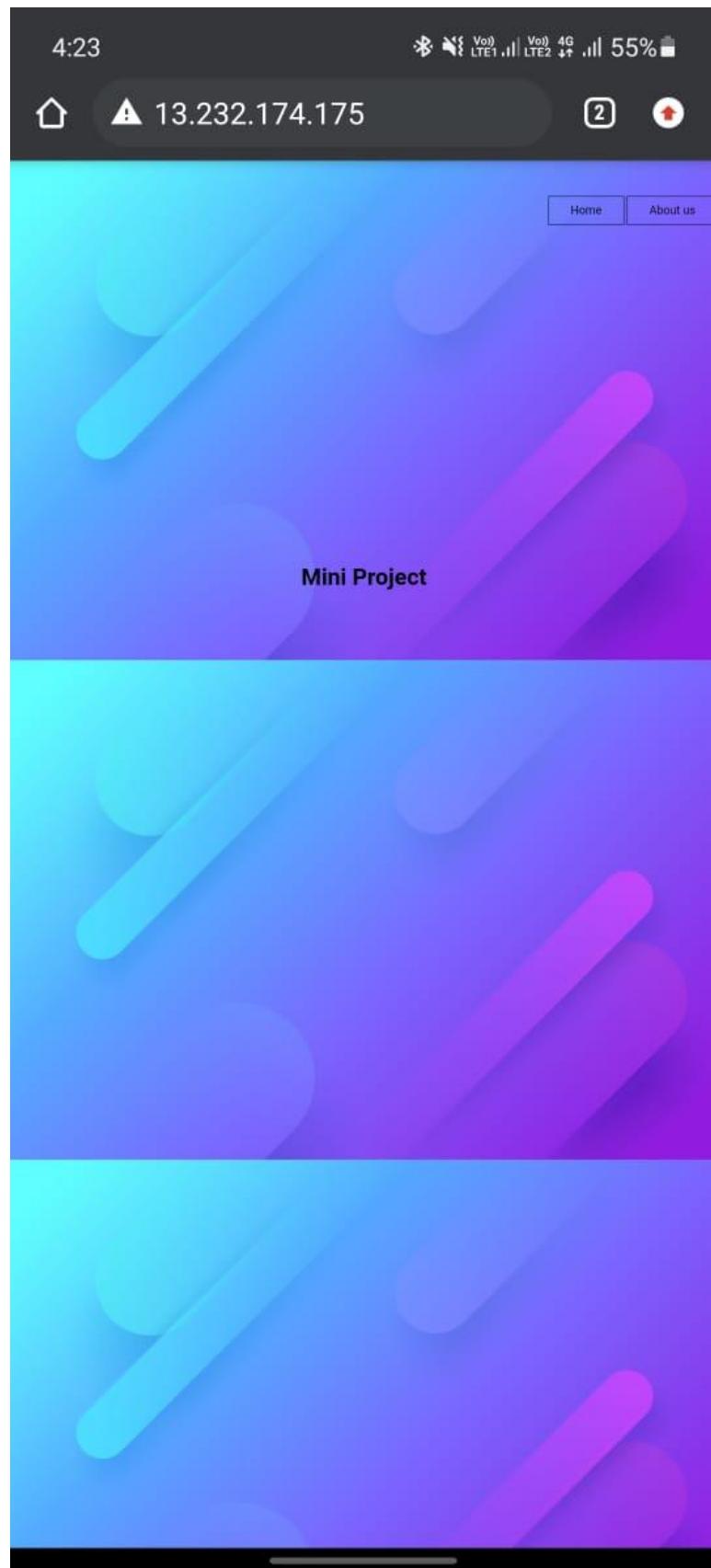


Now windows can't connect

We can also check the rules via going inside the web-server

A screenshot of a terminal window with a black background and white text. The command entered is "iptables -L INPUT -v -n". The output shows a single rule: "Chain INPUT (policy ACCEPT 31408 packets, 4518K bytes) pkts bytes target prot opt in out source destination 77 17432 DROP all -- \* \* 152.57.186.218 0.0.0.0/0". The prompt "[root@ip-172-31-33-111 ~]#" is visible at the bottom.

Now at this same time we have also tried to connect to the webserver by different device and its works absolutely fine



# **Chapter 6**

## **Result and Declaration**

### **6.1 Result:**

1. Our Environment was configured Successfully (Jenkins-master node, webserver, docker-nfs node)
2. The Machine learning model working internally gave the correct output i.e suspicious IP
3. The Whole pipeline worked successfully and at the end the suspicious IP was not able to connect to the web server while other genuin client can connect

### **6.2 Discussion:**

1. Inspite the project aim was achieved but the project have many scopes of future improvement resulting in reduction of the manual efforts of the datascientists and the security team
2. Manual work is reduced as the pipeline have the facility to be triggered by unique url so our monitoring tools can do this job as an when the load increase the first job is triggered and the series start

# **Chapter 7**

## **Conclusion and Future Scope**

### **7.1 Conclusion**

In this project we have successfully implemented the k-mean clustering algorithm for the suspicious IP detection. We did the data processing using the logstash for which we setup the docker launch logstash over it created our own custom configuration file. We used the jenkins to automate the stuff on a single click the all the jobs can be run in such a way the pipeline were designed. Finally the implementation also gave a good response

### **7.2 Future Scope**

1. The approach used for creating machine learning model for the DOS attack using similar approach model can be developed for other pattern based attacks too.
2. Internally for creation of the model we have use the K-mean clustering algorithm (Discussed further in detail). And in this algorithms we need to decide the hyper parameter i.e number of cluster to be formed this is decided by analysis over the data (elbow method). Now we can provide this report of analysis frequently to our datascientest team so that if any changes need in code in future it can be done. This will reduce task of datascientest as well as there will be continuous improvement in the project

## **References**

1. [K-Means Clustering Approach to Analyze NSL-KDD Intrusion Detection Dataset](#)
2. [https://hub.docker.com/\\_/logstash](https://hub.docker.com/_/logstash)
3. <https://www.elastic.co/guide/en/logstash/current/input-plugins.html>
4. <https://www.elastic.co/guide/en/logstash/current/filter-plugins.html>
5. <https://www.elastic.co/guide/en/logstash/current/output-plugins.html>
6. <https://us-cert.cisa.gov/ncas/tips/STo4-015>