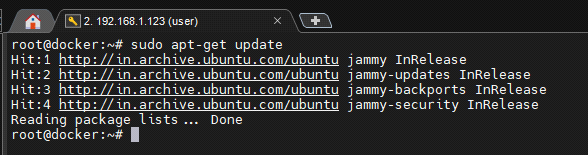
**Docker**

**DOCKER INSTALLATION STEPS:**

1. uninstall docker if it is installed before. using the below cmd.

# sudo apt-get remove docker docker-engine docker.io containerd runc

2. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

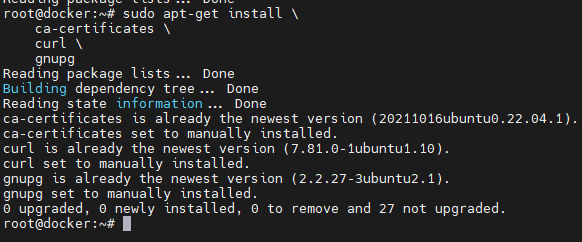
# sudo apt-get update

# sudo apt-get install \

ca-certificates \

curl \

gnupg



3. Add Docker’s official GPG key:

# sudo mkdir -m 0755 -p /etc/apt/keyrings

# curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

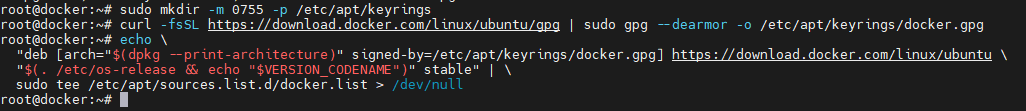
4. Use the following command to set up the repository:

# echo \

"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] <https://download.docker.com/linux/ubuntu> \

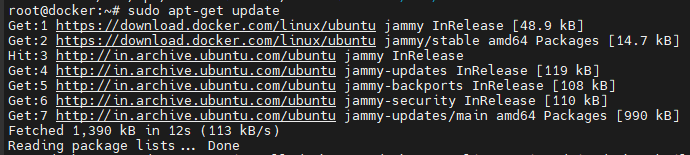
"$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null



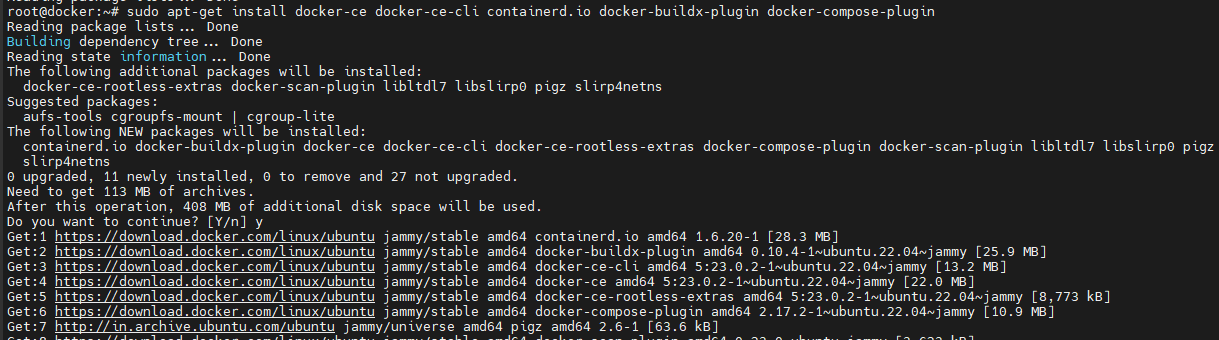
5. Update the apt package index:

# sudo apt-get update



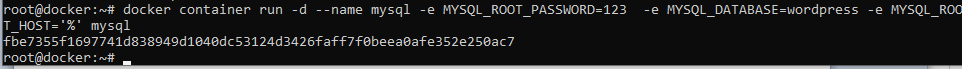
6. Install Docker Engine, containerd, and Docker Compose.

# sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin



MYSQL &WORDPRESS DEPLOYMENT IN DOCKER:

# docker container run -d --name mysql -e MYSQL\_ROOT\_PASSWORD=123 -e MYSQL\_DATABASE=wordpress -e MYSQL\_ROOT\_HOST='%' mysql

we can check IP of container using 2 methods.

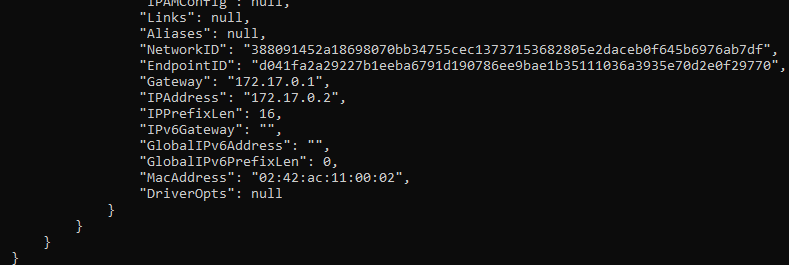
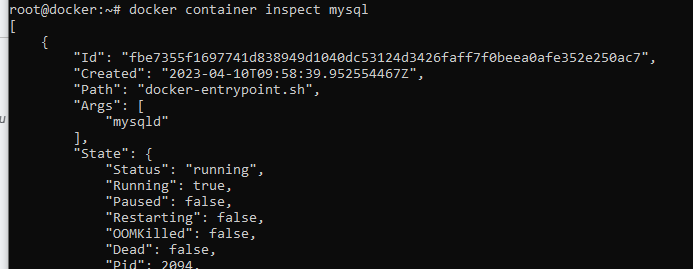
* we can go inside of the container and check ip of it

# docker container exec -it mysql bash

# ip a

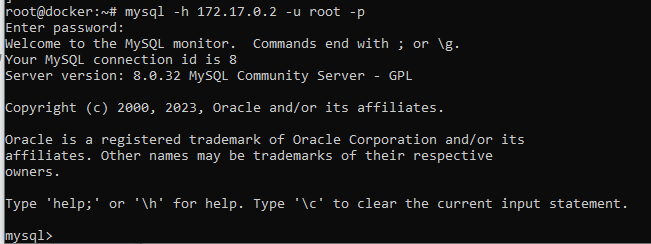
* we can directly indspect the container.

# docker container inspect mysql

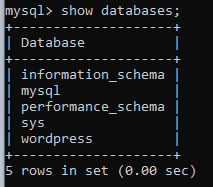
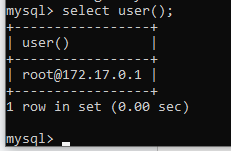


To connect to the mySql container use the below command. replace the<ip:172.17.0.2 > with ur mysql container IP

# mysql -h 172.17.0.2 -u root -p



Check is the data Base and users exit or not.

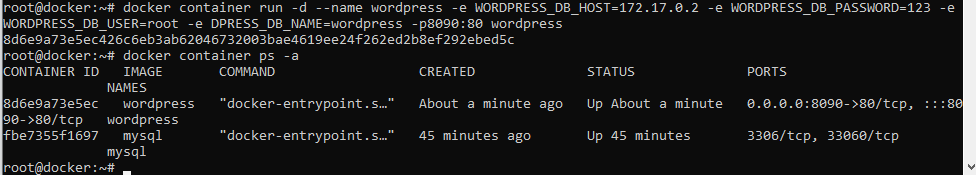
 

Deploy a wordpress container and do port mapping also.

# docker container run -d --name wordpress -e WORDPRESS\_DB\_HOST=172.17.0.2 -e WORDPRESS\_DB\_PASSWORD=123 -e WORDPRESS\_DB\_USER=root -e WORDPRESS\_DB\_NAME=wordpress -p8090:80 wordpress

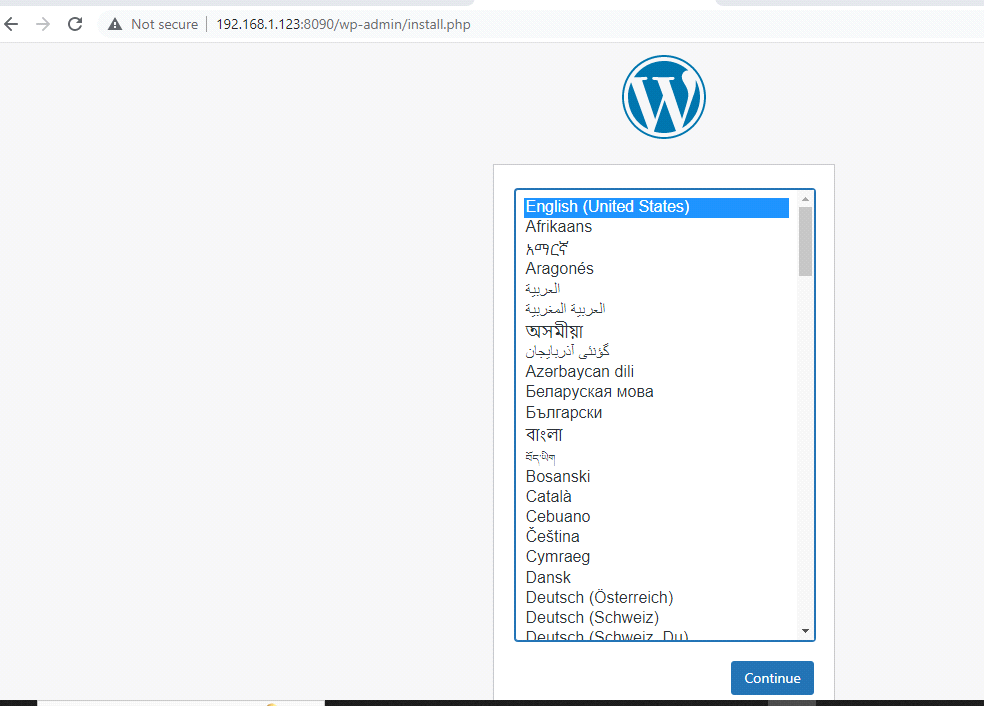
check the status of both the containers. to check status cmd

# docker container ps



copy the IP of VM and browse it. <192.168.1.123:8090>

change the IP and port number with your IP and port. then you will get the page like below.



**Configure Static <IP address> for Docker host:**

**NAT:**

Outgoing connections will be allowed and Incoming connections are blocked. For allowing incoming also we need to use Port Forwarding.

**Bridge Interface:**

Instead of getting IP address from virtual box we will get IP from the router itself.

The way how our Base machine getting IP from router simply virtual machine also gets IP.

**DHCP:**

If we connect our machine with Wi-Fi (router) we will get IP automatically. we cannot guarantee we will get the same IP. This process is called DHCP. Like that only if we shut down and start the virtual machine in Virtual Box it will also changes.

**How to make IP address of VM Static:**

1. To communicate with our VM from Third party tools (command prompt)

ssh username@<IP of VM>

2. Open config File. In that file already some entries exit in that entries DHCP make it as false actually it is true.

# vim /etc/netplan/00-installer-config.yaml

network:

ethernets:

enp0s3:

dhcp4: false

addresses: [192.168.1.40/24] # here based on your base machine give some IP and its bits ex:255.255.255.0(means 24 bits)

routes:

- to: default

via: 192.168.1.1 #give here your gateway IP

name servers:

addresses:

- "8.8.8.8"

- "8.8.4.4"

version:2

3. To apply changes in config file we need to provide the below cmd.

# netplan apply

!!! "When we do port mapping it is important."

**Deploy Jenkins CICD application on Docker:**

-> Jenkins listens on Port 8080 & Slave listens on 50000

-> Jenkins root dir: /var/lib/jenkins on servers & /var/jenkins\_Home for containers.

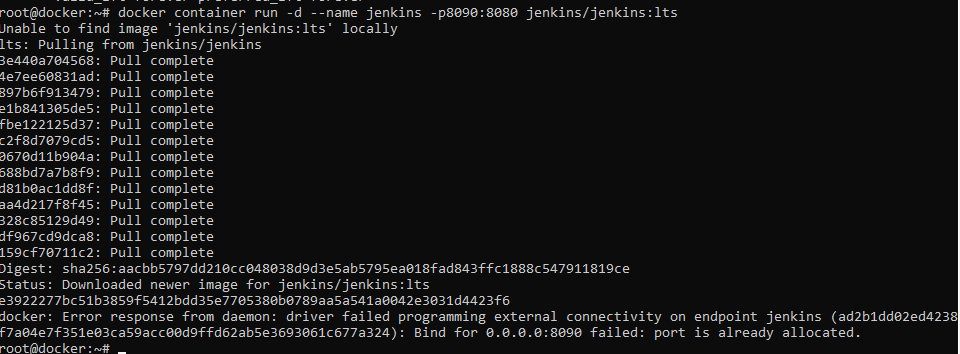
-> We can build CICD jobs using DSL(domain specific language) or groovy script or create jobs or pipeline.

-> In Production level we need to go for jenkins/jenkins CICD LTS version image. Because there is no need to update Jenkins.

-> Only for dev/Testing we use weekly version.

-> before installing Jenkins & port mapping make your IP as static using the before steps.

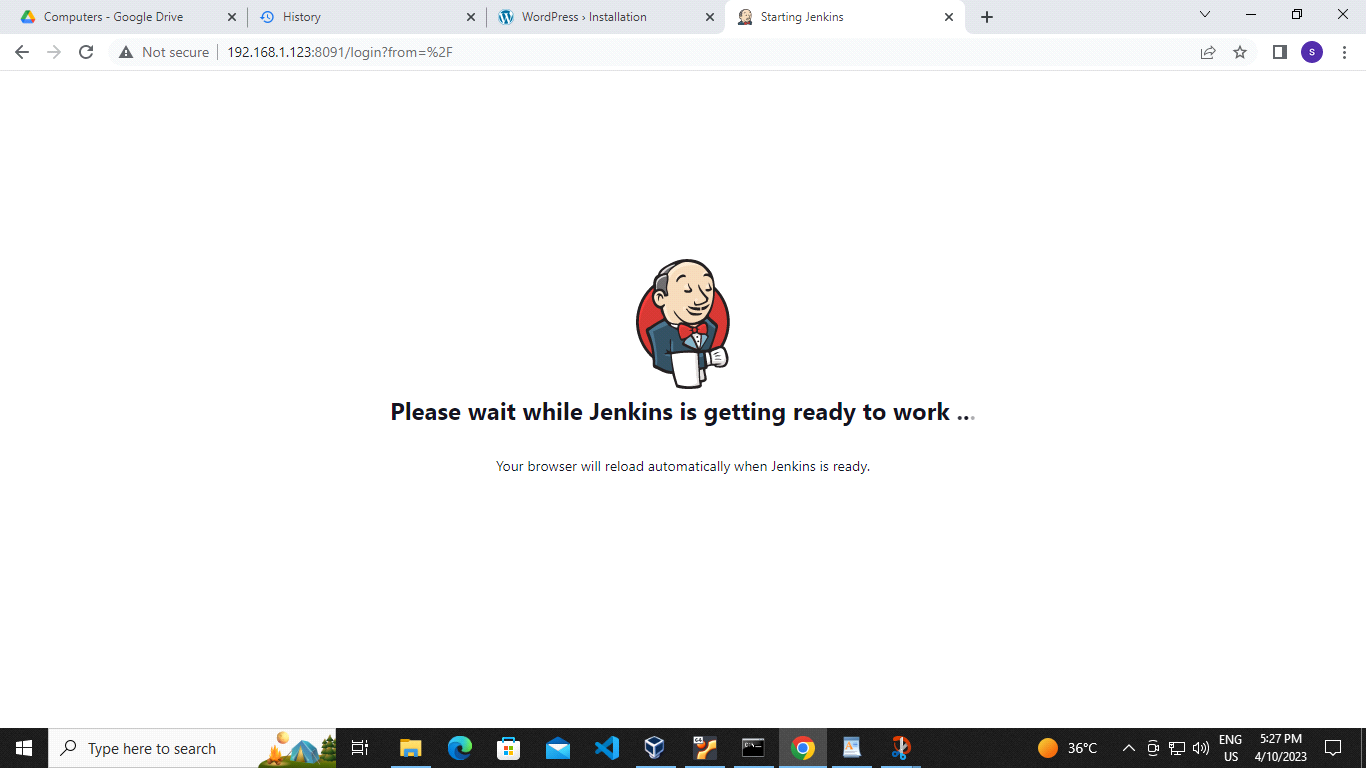
# docker container run -d --name jenkins -p8090:8081jenkins/jenkins:lts



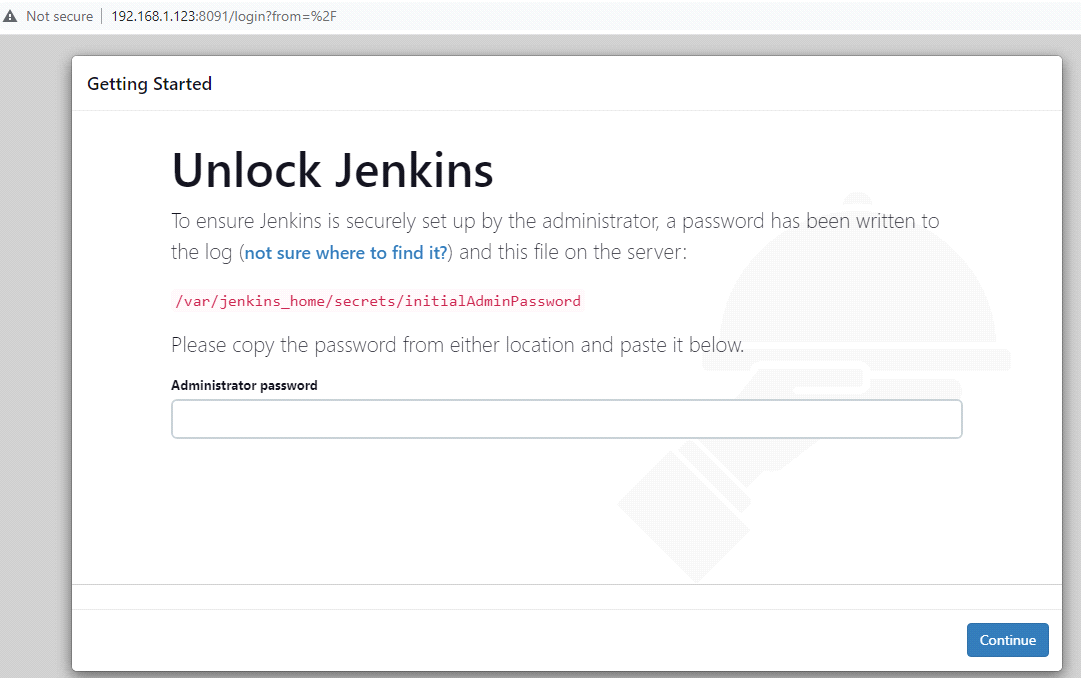
check the status of the container using below cmd

# docker container ps

Now copy IP of the container & paste it on browser(<192.168.1.123:8091) then we get jenkins page like below.



In the below image it is mentioning like to unlock the jenkins use path /var/jenkins\_home/secrets/initialAdminPassword in this path we will get key to unlock jenkins.

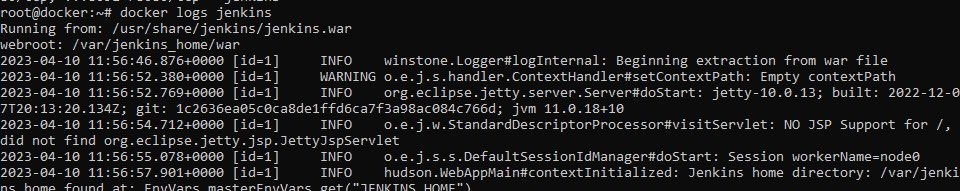


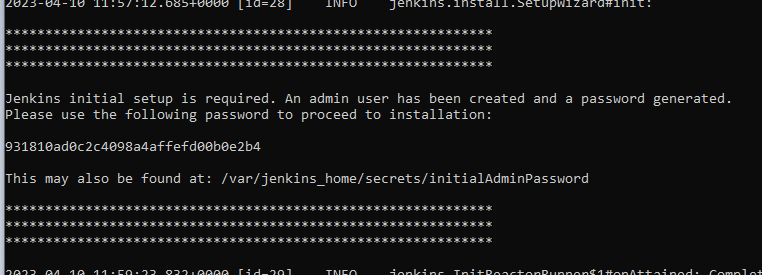
To get the passwd we have 2 methods:

1. go inside of the container and copy the passwd by using above path &

2. we can check the logs

# docker logs jenkins

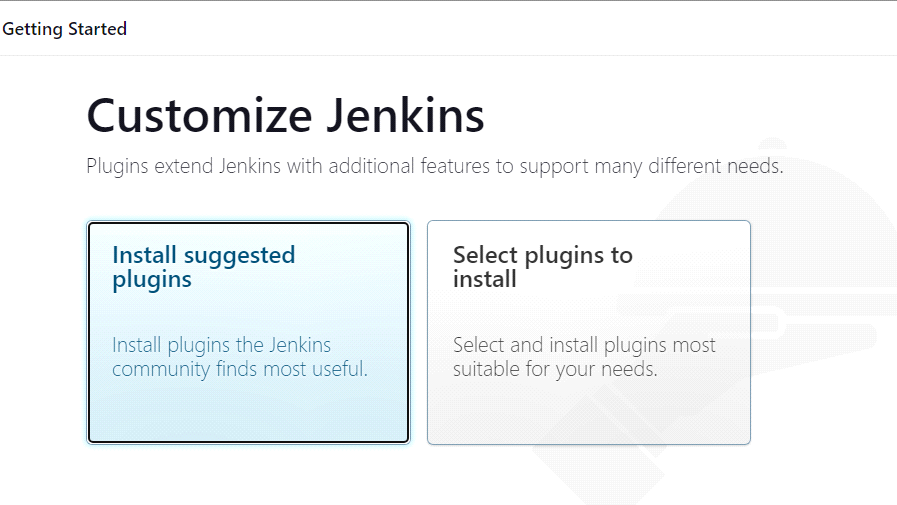




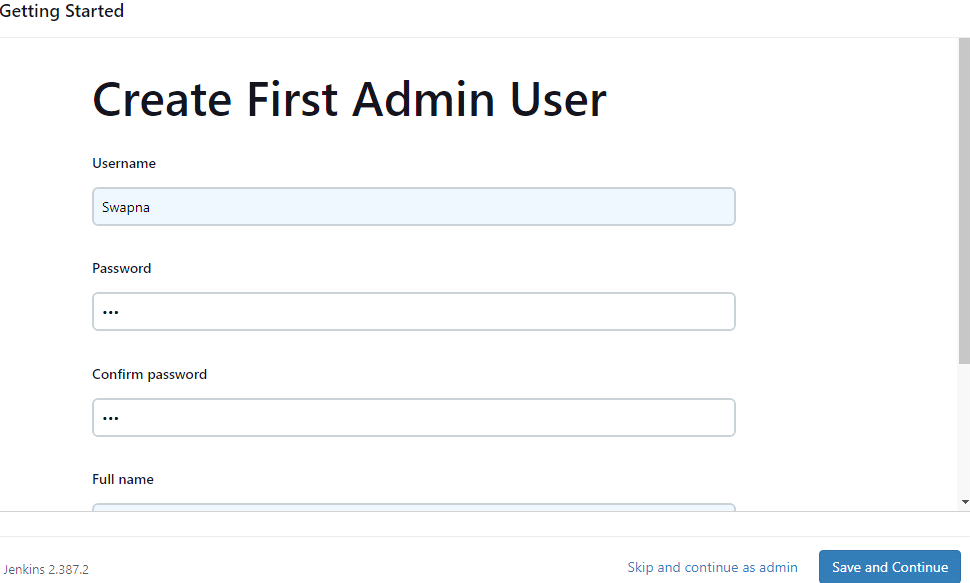
# docker logs -f jenkins( if we execute this cmd it will shows the current state of our jenkins installation after providing passwd).

Copy and paste the passwd on the browser. then it will look like this and then click continue.

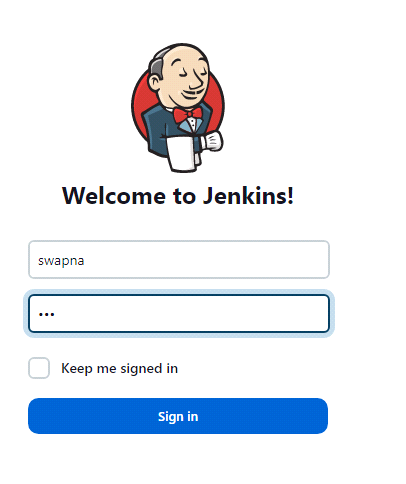
In the below picture select install suggested pluggins. then it will download some pluggins.



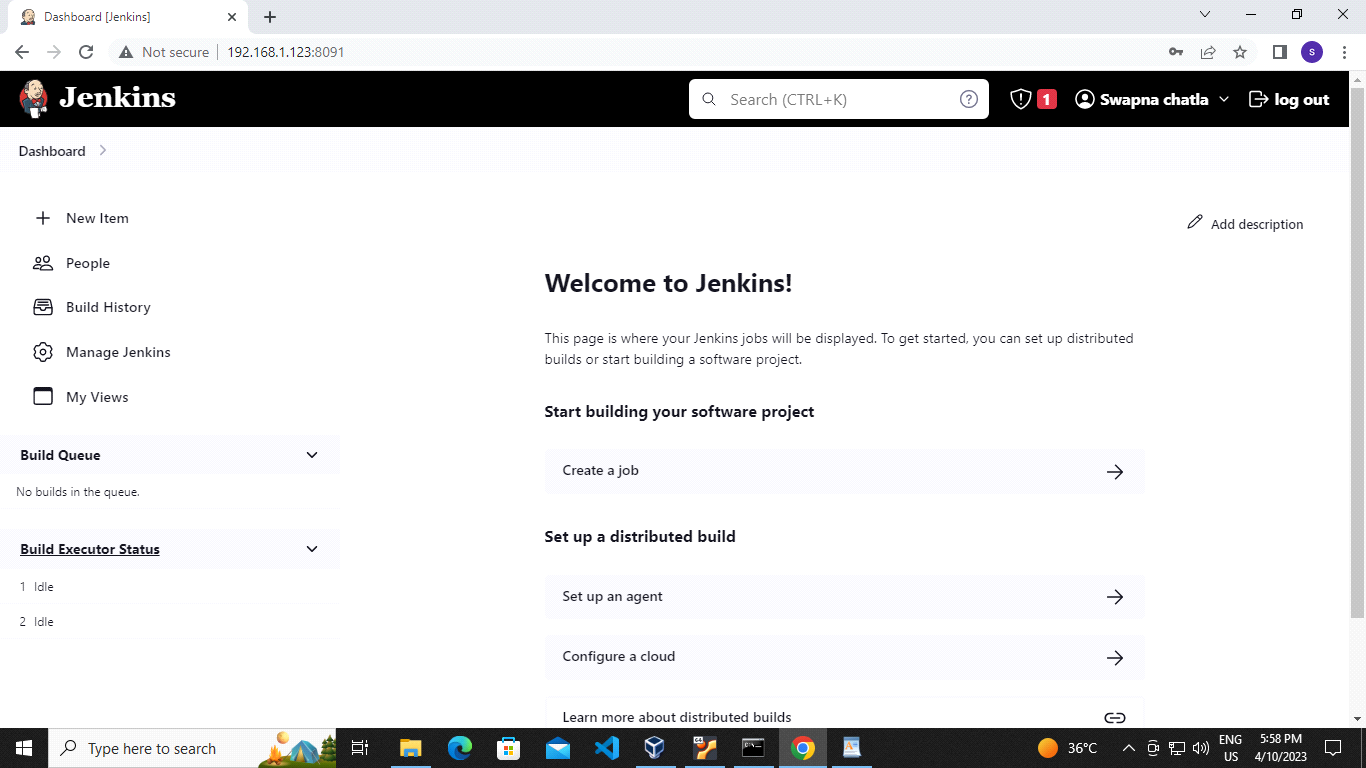
For installing pluggins it will take some time Then we will get authentication picture like below. In that prove username name and password based on your wish.



after providing details click save and continue the u will get the jenkins dashboard page like below.logged in by entering the username & passwd which u provide in above step.



now finally we are in the page of jenkins dashboard page. It will look like below



**DOCKER NETWORK TYPES:**

Predefined network: The networks which are created by docker while installing is called pre-defined newtork.

Custom network: network which are created by us is called custom netwotk

Bridge:

It is a default network driver for containers. Docker network supports default & user-defined bridge network. In real time we recommended to create user defined bridge network. It is a software bridge created by docker. Containers in the same bridge can communicate with each other, where as other bridge network is blocked. Docker bridge network creates some rules (ip tables) on the host machine. By manipulating IP tables & docker network connect cmd 2 cointaioners can talk to each other. docker container in the same network can talk to each other.

**Docker default bridge network:**

These are installed when we download docker packages. If we try to delete the network is will not happen.

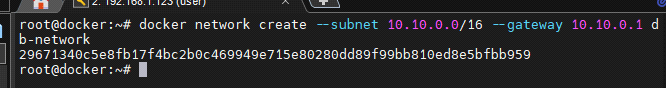
# docker network remove <networkName>

# docker network inspect <networkName> # to see the info about particular network

Docker user-defined bridge network:

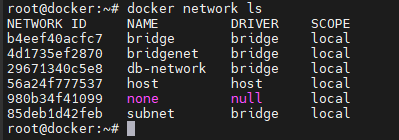
-> Create a user-defined bridge network with the following cmd. replace db-network name with the name u want for creating network.

# docker network create --subnet 10.10.0.0/16 --gateway 10.10.0.1 db-network



-> list the networks by using the below cmd.

# docker network ls



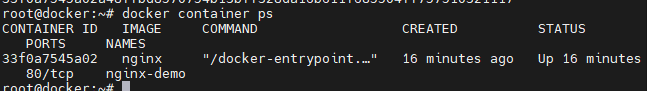
-> create a container with the network which we created above using below cmd.

# docker container run -d --name nginx-demo --network db-network nginx



-> check the status of the container using the cmd below

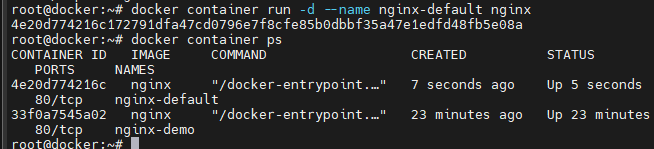
# docker container ps



# docker container inspect nginx-demo

-> create another container with the default network

# docker container run -d --name nginx-default nginx

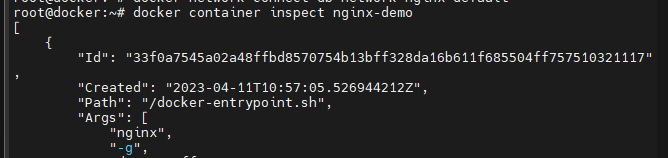


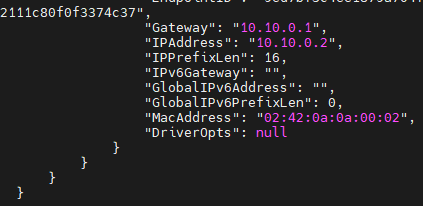
-> To make connection between both the networks provide the below cmd.

# docker network connect db-network nginx-default

-> provide the below command to get the ip of the container

# docker container inspect nginx-demo





use now go inside of the container with the below cmd. Install the package which is needed to run ping cmd

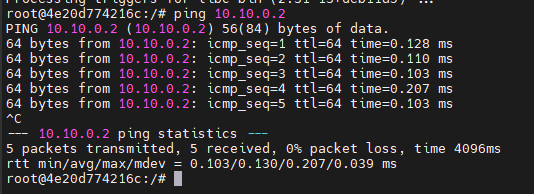
# docker container exec -it nginx-default bash

/# apt update

/# apt install iproute2 iputils-ping

/# ip a

/# ping 10.10.0.2 # paste the ip of the container belongs to user-defined network



**User defined Bridge network managed by Docker Daemon:**

let us create a subnet with the following cmd

# docker network create w-network

# docker network ls

we didn't mention subnet in the above cmd. let's check subnet is attached or not. If attached which type is attached.

# docker network inspect w-network

IP is attached and is classB type. If we don't pass any subnet or gateway by default docker daemon will take control & it will pass the default series as per the program what they written in the background.

**Points to remember while using Bridge Network:**

There are 2 type of bridge networks. 1. Predefined network 2. Custom network

1. Predefined network:

The networks which are created by docker while installing it.

2. Custom network:

the networks which are created by user. It is also called as user-defined network.

--> Default bridge network is created by docker Engine packages itself.

--> pre-defined bridge network cannot be removed(docker0->default).

--> User-defined bridge network can be create with or without passing subnet values.

--> It is not recommended to use default bridge network for application to avoid risks.

--> Containers can be connected & disconnected from user-defined & default bridge network. The container which is not attached to any network in that the cmd will not run.

--> User-defined network helps to assign different network settings like DNS, MTU, IP tables rules etc.,

--> By default environment variables can be shared to containers on default bridge using --link, which is not possible in user-defined network. we cannot remove a network which is attached to a container. even forcefully also we can't remove that container.

--> In user-defined bridge network, we can ping containers with the name of the container as well as <IP> address also. But in default bridge network we cannot ping the containers with the name.

--> when we creating a bridge network if we didn't provide subnet by default docker daemon will take control & it will allot the subnet ip as per the program what they written in the background.

**Host Network:**

In host networks containers & Host machines share same networks means containers have the same IP of host. Here there is no need of port mapping. But container port & application ports should not be the same.

--> Application running on the Host & Containers share common network stack. make sure that all applications in containers & Host doesn't listen on the same port.

--> Only network namespaces will be shared with host, but rest will be isolated.

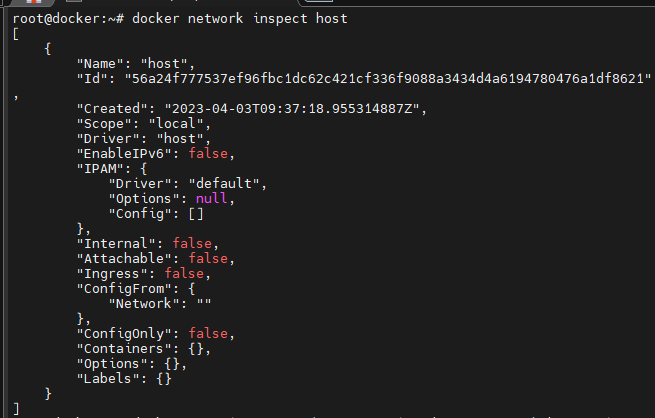
--> It is not recommended in real time. whenever the port number is not static at that situations use Host Network.

--> cannot create user-defined host network on docker host machine. host network is predefined network and cannot be removed.

**Host Network Creation:**

--> In docker, host network is not alloted with any ip. To check that use the below command

# docker network inspect host



--> create a container on host network

# docker container run -d --name nginx-demo --network host nginx

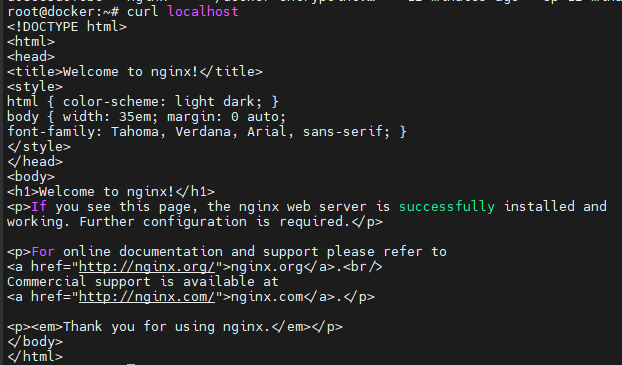
--> check the status of the container

# docker container ps



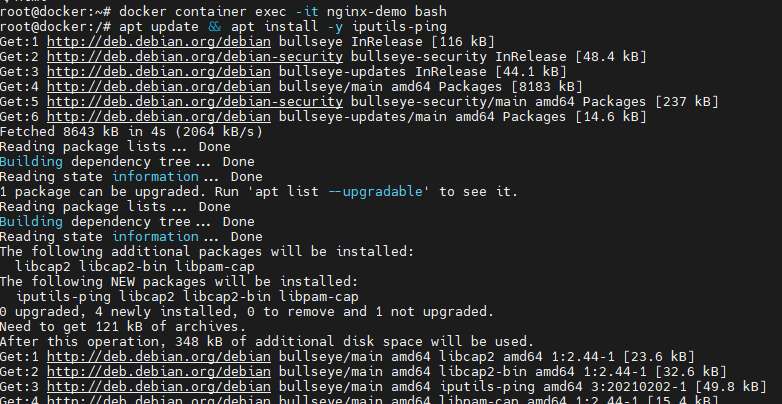
--> check that the content what is in the application of the container

# curl localhost

Go inside of the container and install iputils-ping iproute2

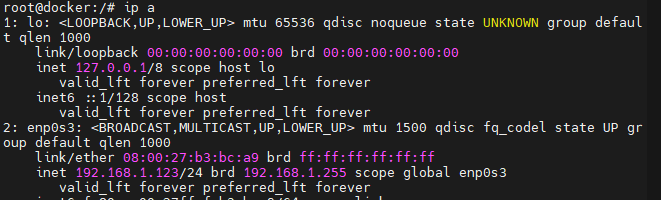
# docker container exec -it nginx-demo bash

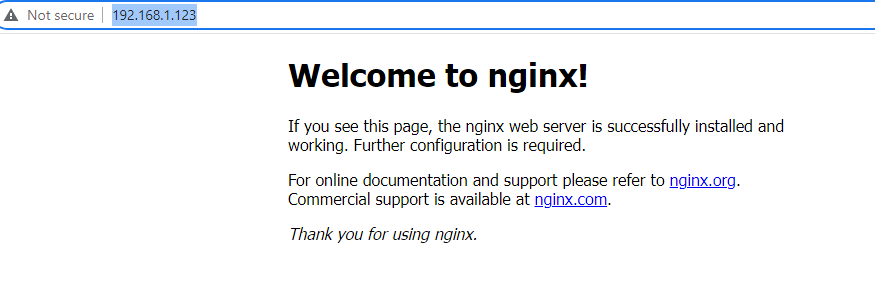
/# apt update && apt install -y iputils-ping



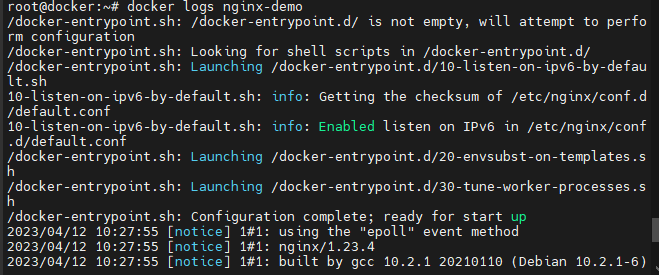
--> now provide the cmd to check the ip of a container

/# ip a

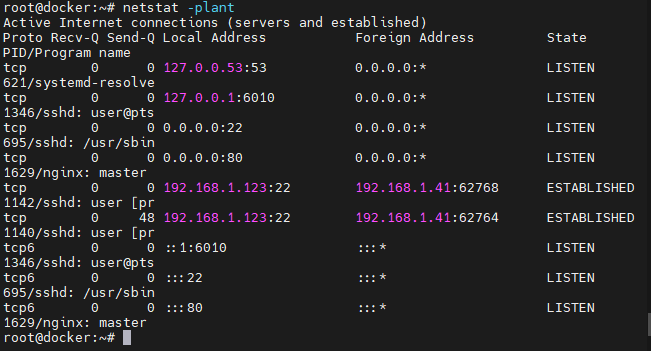


--> copy the the ip and paste it on browser <http://192.168.1.123/> then we get the following page.

check the docker logs to know the information what is going on in the backend

# docker logs nginx-demo --> To kow the network statistics use the below cmd.

# netstat -plant



**Points to remember while using Host Network:**

--> By default Host network will be available on Docker Host machine.

--> Docker host network provide direct access to application running in container without using port mapping(No NAT).

--> It has to be used when other apects of container to be isolated, butr not network stack. We can't run 2 or more applications on host or container with the same port no.

--> Containers which need to handle large range of port, where port are dynamically opened. We can't attach container to bridge network, while it was attached to host network.

--> Docker host network is only available on linux hosts. we can't create multiple host networks. Predefined host network will not be allowed to remove.

**None network:**

--> Only I want to test my application but I don't want to connect any network stack in my docker then we will go for None network.

--> disconnect from none network & attach to any bridge network.

--> Creating a container on the none network instructs Docker not to provision any connected virtual Ethernet adapters for the new container. It will have its own network namespace and so it will be isolated, but without adapters connected across the namespace boundary, it will not be able to use the network to communicate outside the container. Containers configured this way will still have their own loopback interface, and so multiprocess containers can still use connections to localhost for interprocess communication.

You can verify this by inspecting the network configuration yourself. Run the following command to list the available interfaces inside a container on the none network:

**DOCKER STORAGES**

There are two types of Docker Storages

1. Storage Driver 2. Data Storages

DATA STORAGES:

There are three types of storages.

1. Bind mount

2. Volume mount

3. tmpfs mount

VOLUME MOUNT:

--> volume mount can be controlled using Docker API.

--> Volume mount shares the host file system under docker root directory (/var/lib/docker/->volume) without create volume as pre-requisite we can launch container.

--> Single volume can be shared to multiple containers. we can attach volume as read-only to containers to ensure data is not controlled from application inside for write operation.

--> Volumes can be used while creating container using --v(volume) or --mount options.

--> When we want to use volume mount for docker swarm tool, It is recommended to use --mount instead of -v option.

--> We can create static volume and dynamic volume mount.

--> Without removing container we cannot delete volume.

# docker info

use the below command to check the created volumes

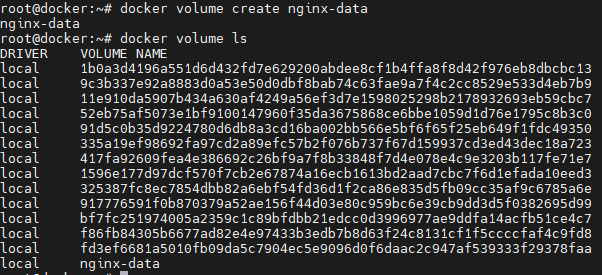
**Steps to create docker volume:**

Use the below command to see the created volumes list

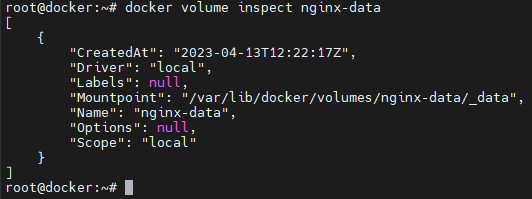
# docker volume ls

use the below command to create volume

# docker volume create nginx-data



# docker volume inspect nginx-data



# ls -l /var/lib/docker/volumes/nginx-data/\_data

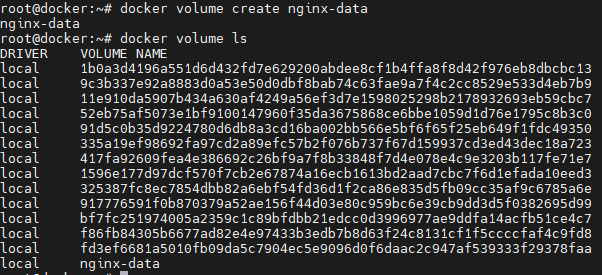


**Steps to attach 2 containers to single Volume:**

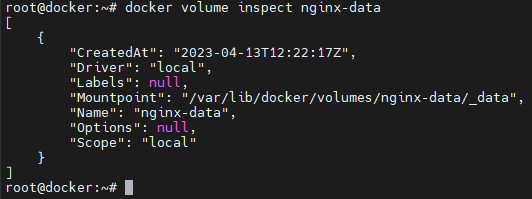
# docker volume ls

use the below command to create volume

# docker volume create nginx-data



# docker volume inspect nginx-data

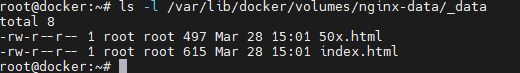


# docker container run -d --name nginx-demo -v nginx-data:/usr/share/nginx/html -p8090:80 nginx



If the content in the volume is manipulated then it will be reflects on container

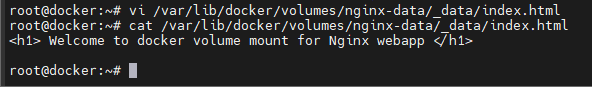
# ls -l /var/lib/docker/volumes/nginx-data/\_data



# vi /var/lib/docker/volumes/nginx-data/\_data/index.html

d shift+g # by using this we can delete the whole content in above file. Then click insert button or i & Type the content like below

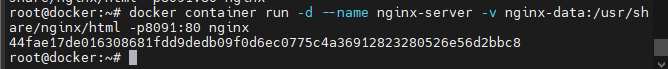
<h1> Welcome to docker volume mount for Nginx webapp </h1>

copy the IP of the container and browse it <http://192.168.1.123:8090/>



now attach the same volume to another container for that first create another container

# docker container run -d --name nginx-server -v nginx-data:/usr/share/nginx/html -p8091:80 nginx

Now try to modify the volume again

# vi /var/lib/docker/volumes/nginx-data/\_data/index.html

<h1> hi.. I modified the content in the volume again</h1>

now browse both the IP addresses <http://192.168.1.123:8090/>



<http://192.168.1.123:8091/>



--> Now stop the container & then check the volume path is the data remained there or not. It will be there.

--> I want to mount this particular volume as read only permission which means I don't want to give a flexibility to my container to manipulate the data.

# docker container exec -it nginx-server bash

/# cat /usr/share/nginx/html/index.html

/# echo "<h1> welcome to docker volume mount for Nginx webapp v2</h1>" > /usr/share/nginx/html/index.html

stop the container

Once a volume is attached to container we cannot detach it. we need to create another one.

**Steps to create docker volume with read-only permissions:**

# docker container run -d --name nginx-server -v nginx-data:/usr/share/nginx/html:ro -p8090:80 nginx

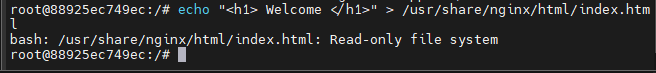


# docker container exec -it nginx-server bash

/# cat /usr/share/nginx/html/index.html



/# echo "<h1> Welcome </h1>" > /usr/share/nginx/html/index.html

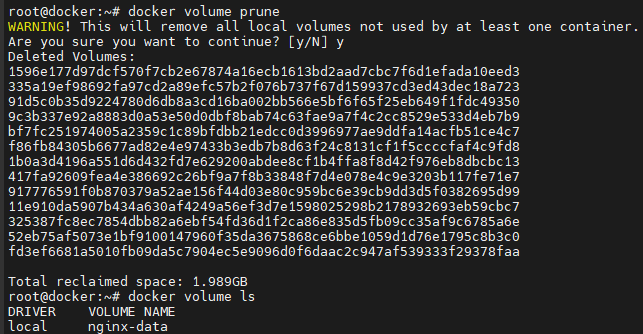


**Without creating volume, initiate container creation usnig -v option:**

A volume can be created while creating container also.

To remove all unused docker volumes we use the below command.

# docker volume prune



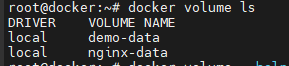
now create a container and mention the volume name also. the below command creates ata a time container and volume also. check above picture there is no volume with the name which is mentioned in the below command.

# docker container run -d --name nginx-demo -v demo-data:/usr/share/nginx/html nginx



By using above command container and volume are created. Now list the volumes

#docker volume ls



Remove the volume using below command

# docker volume rm demo-data



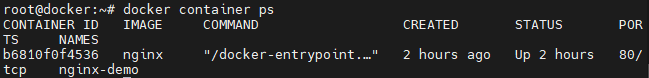
After applying the above command it is showing error because the volume is attached to a container.

# docker volume inspect demo-data

# docker container inspect nginx-demo

--> If we stop the container can we remove it

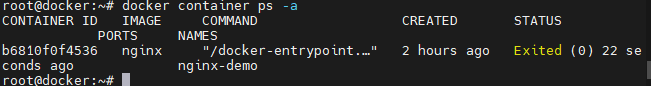
# docker container ps



# docker container stop nginx-demo



# docker container ps -a



# docker volume rm demo-data



volume cannot be removed even if the container is in running or on stop state. After removing the container only volume gets deleted.

**Creating volume mount using --mount:**

In volume -v it separates using ':' in --mount it separates using ','

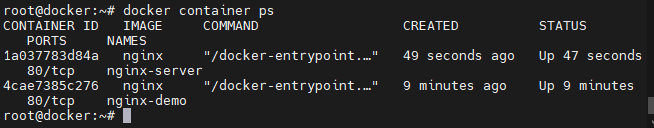
# docker container run -d --name nginx-demo --mount type=volume,src=nginx-data,dst=/usr/share/nginx/html nginx



# docker container run -d --name nginx-server --mount type=volume,src=nginx-data,dst=/usr/share/nginx/html,readonly nginx

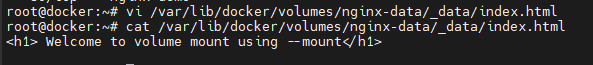


# docker container ps



# vi /var/lib/docker/volumes/nginx-data/\_data/index.html

<h1> Welcome to volume mount using --mount</h1>



To know the ip of container use docker inspect command

# docker container inspect nginx-server

# docker container inspect nginx-demo

# curl 172.17.0.2



# curl 172.17.0.3



**Jenkins deployment Using Volume mount:**

In this task first Deploy a container using jenkins CICD image with volume mount and then do portforwarding between the docker container application and base machine.

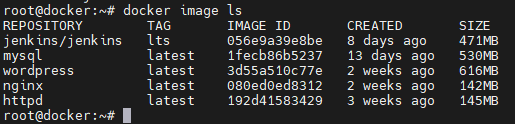
# docker volume ls



# docker container run -d --name jenkins-server -p8090:8080 -v jenkins-data:/var/jenkins\_home jenkins/jenkins:lts



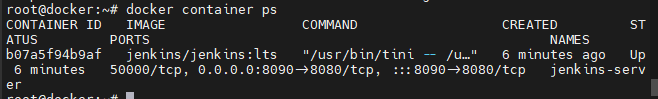
# docker image ls



# docker volume ls



# docker container ps



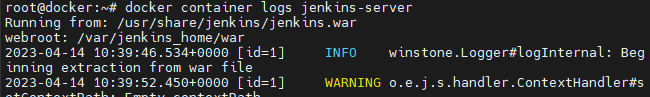
copy paste the ip of the base machine and port number it will be look like this <http://192.168.1.123:8090> browse it then jenkins application page will be visible.

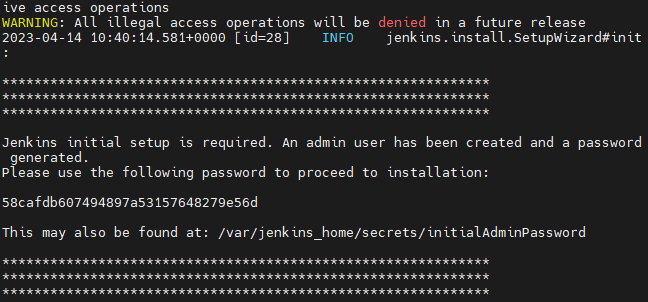
# cat /var/lib/docker/volumes/jenkins-data/\_data/secrets/initialAdminPassword



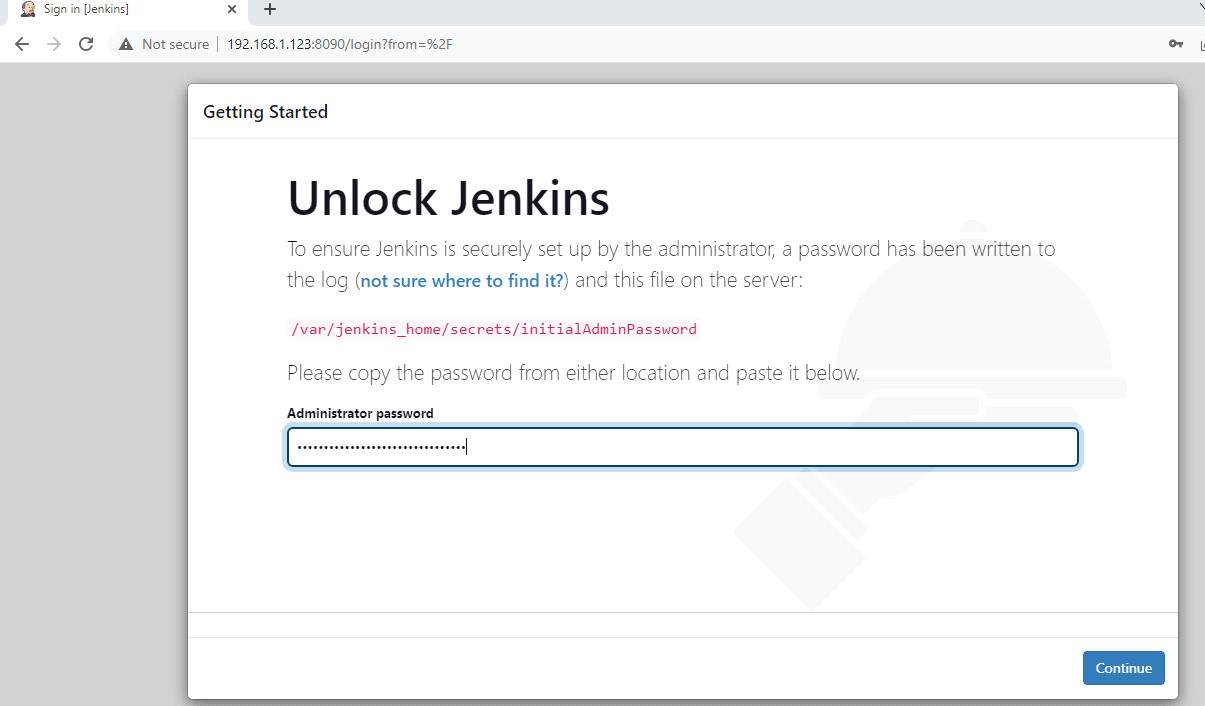
(or)

# docker container logs





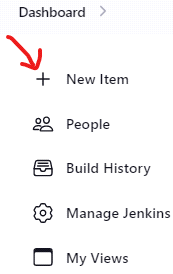
copy the password using anyone of the above methods and paste it on the jenkins application page.



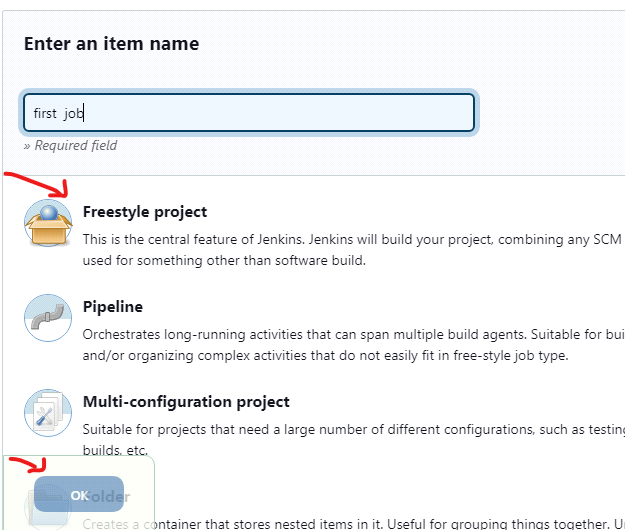
click on continue then it will takes to "Customize Jenkins Page - click on Install suggested pluggins option". then it will download pluggins to download pluggins it takes some time then click on continue it will takes u to the "Create First Admin User Page their provide UserName, Password, ConfirmPassword, Full Name, email address - click on save & continue - continue - provide username & Password then it will takes u dashboard page."

After providing user credentials jenkins will delete its InitialAdminPassword File.

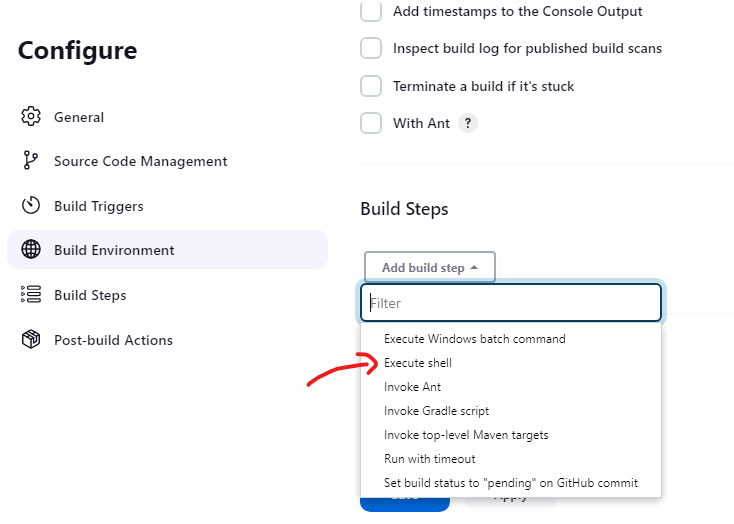
In dashboard of jenkins click on new item



click on Free style job - ok then it will takes you configure page



-Build steps- execute shell

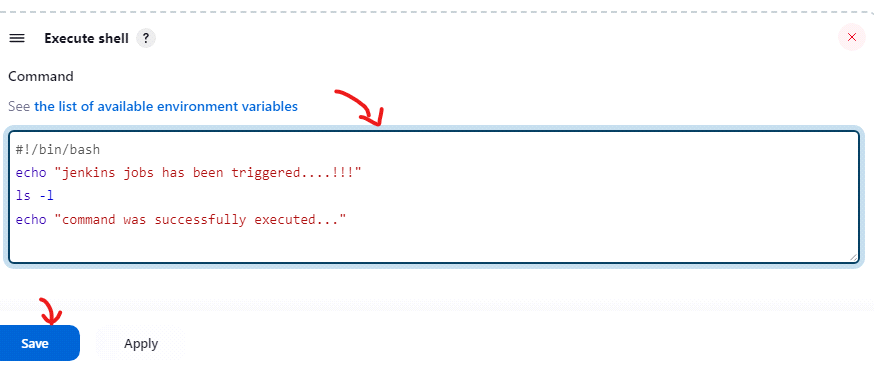


#!/bin/bash

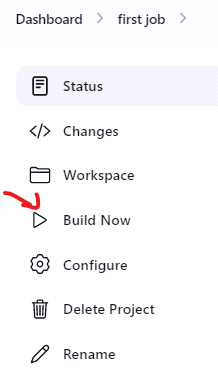
echo "jenkins jobs has been triggered....!!!"

ls -l

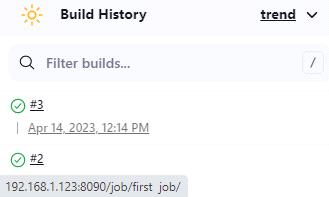
echo "command was successfully executed..."



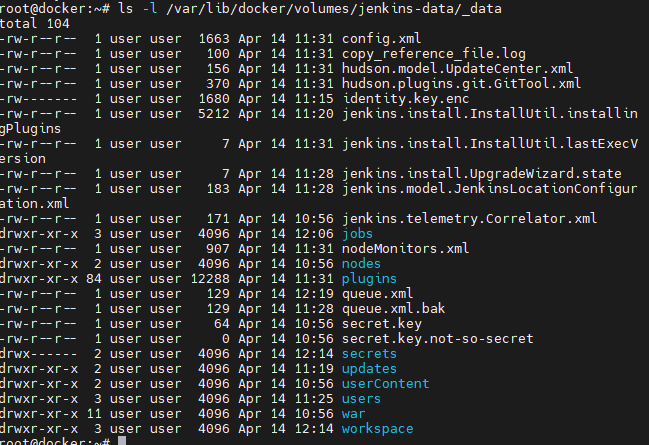
click on save then it will takes u the below page their click on Build now



If the build is suucess it will shown like below



list the data in the container # ls -l /var/lib/docker/volumes/jenkins-data/\_data



Now remove the container. for removing the container we need to stop it first

# docker container stop jenkins-server

# docker container rm jenkins-server



# docker container create -d --name jenkins-cicd -p8090:8080 -v jenkins-data:/var/jenkins-home jenkins/jenkins:lts



after creating the container browse it <http://192.168.1.123:8090/> replace the and port number with your IP and Port number then use the below command copy & paste the password in the browser

# docker container logs jenkins-cicd

# ls -l /var/lib/docker/volumes/jenkins-data/\_data