**Docker Installation Steps**

**First, update your existing list of packages:**sudo apt update

**Next, install a few prerequisite packages which let apt use packages over HTTPS:**  
sudo apt install apt-transport-https ca-certificates curl software-properties-common

**Then add the GPG key for the official Docker repository to your system:**  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

**Add the Docker repository to APT sources:**  
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

**Next, update the package database with the Docker packages from the newly added repo:**  
sudo apt update

**Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:**  
apt-cache policy docker-ce

**Finally, install Docker:**  
sudo apt install docker-ce

**Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it’s running:**  
sudo systemctl status docker

**Some Commands to try:**  
docker run –help  
docker run hello-world  
sudo docker run hello-world  
docker search ubuntu  
sudo docker search ubuntu

**To view the options available to a specific command, type:**docker docker-subcommand --help

**To view system-wide information about Docker, use:**docker info

**Execute the following command to download the official ubuntu image to your computer:**  
docker pull ubuntu  
docker images 🡪 to check the available images  
docker rmi -f <image\_name>  
  
**As an example, let’s run a container using the latest image of Ubuntu. The combination of the -i and -t switches gives you interactive shell access into the container:**  
docker run -it ubuntu  
docker run -it -d ubuntu > to make sure container is live after run.   
sudo docker exec -it <cont\_id> bash

**Some more commands:**

sudo docker rm -f <cont\_id> 🡪 **to forcefully remove the container.**docker run -it -p 82:80 -d ubuntu 🡪 **to map port 80 of container to port 82 of OS.  
  
Committing and pushing an Image:**

docker commit <cont\_id> < name of image> **🡪 to save changes of container in docker file, this will create a new image with name provided.**docker login **🡪 to login to registry for pushing the image**docker push <imagename>[:tag] 🡪**To push image to docker registry**  
 **Scenario : Lets create a ubuntu container>install apache on it> save the changes.**

sudo docker rm -f $(sudo docker ps -a -q) 🡪 **to remove all container at once.**

sudo usermod -aG docker $USER 🡪**to run docker commands without sudo.**

**Docker Hub :**

docker login  
docker commit <cont\_id> <userid/name of image> 🡪 **to save changes of container in docker file, this will create a new image with name provided.**  
docker push <username/image>

**Building Docker file:**

sudo vi Dockerfile 🡪 **name should be as shown, no changes else docker will not be able to find it.**

FROM ubuntu  
RUN apt-get update  
RUN apt-get -y install apache2  
ADD . /var/www/html  
ENTRYPOINT apachectl -D FOREGROUND  
  
docker build . -t new\_dockerfile   
echo $name 🡪 **check envt variable**.

**Docker Volume related Commands:**

docker volume create <name of volume> 🡪**to create a docker volume**docker run -it --mount source=<name of volume>,target=<path to directory> -d imagename 🡪**To attach a volume to docker container  
  
BIND MOUNT Related commands:**

docker run -d \  
-it \  
 --name devtest \  
 --mount type=bind,source="$(pwd)"/target,target=/app \  
 nginx:latest

pwd indicates that we are attaching home directory to docker container, we can attach any directory with full path mentioned. Eg. /etc/ansible

**Scenario to Understand difference between Volume and Bind Mount**

1. Create a volume and attach it to ubuntu container.
2. Enter the container and traverse to the directory you mounted and create a file.
3. Come out of container
4. Try finding it in your host machine, you won’t see it as volume is a logical storage entity that we attach to container.
5. Now create another container by attaching your home directory to it as bind mount.
6. Create a file in home directory.
7. Go inside container, traverse to the attached path , there you can see the file created from outside the container.
8. Now create a new file in same path from within the container.
9. Come out of it, you can see that file in your home directory.
10. So the Zeist of this activity is to show that changes made from outside are also visible from inside the container and vice versa. Because it’s a physical path mounted to a container.

**Scenario to Link the Containers**

1. Create a ubuntu container.  
   docker run -it --name container1 -d ubuntu
2. Create another ubuntu container with below command:  
   docker run -it --name container2 --link container2 -d ubuntu
3. Now go inside second container and after installing ping do :  
   ping container1

The connection is established.

**For Running docker from Jenkins:**

**Run below as root:**  
1. RUN echo "jenkins ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

2. echo "jenkins ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

3. echo "alias docker='sudo docker '" >> /home/jenkins/.bashrc

**Run below as user jenkins:**

1. sudo groupadd docker

2. sudo usermod -aG docker $USER

**Creating Swarm**

**For Leader Node**:  
docker swarm init --advertise-addr=address of master  
  
**For Worker Node (Worker should have docker installed in it)**  
Run the command that will be shown after initialising the cluster from master node.

**Deploy an Application in Docker Swarm Cluster:**docker service create --name apache --replicas 3 -p 80:80 hshar/webapp

You can use image and service name of your choice.

**Now test the connectivity to application in browser by:**  
http://<ip addr of master/leader>:80

Service creates endpoint to access the application inside container from outer world.  
So when we talk about swarm cluster we talk in terms of Service.