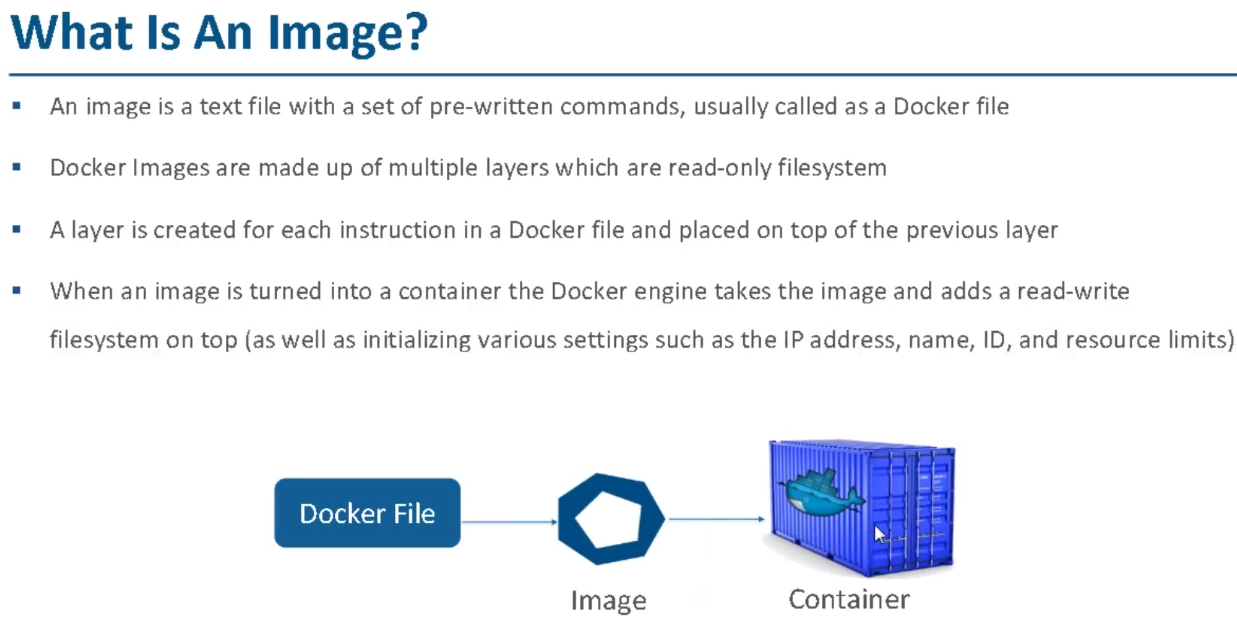
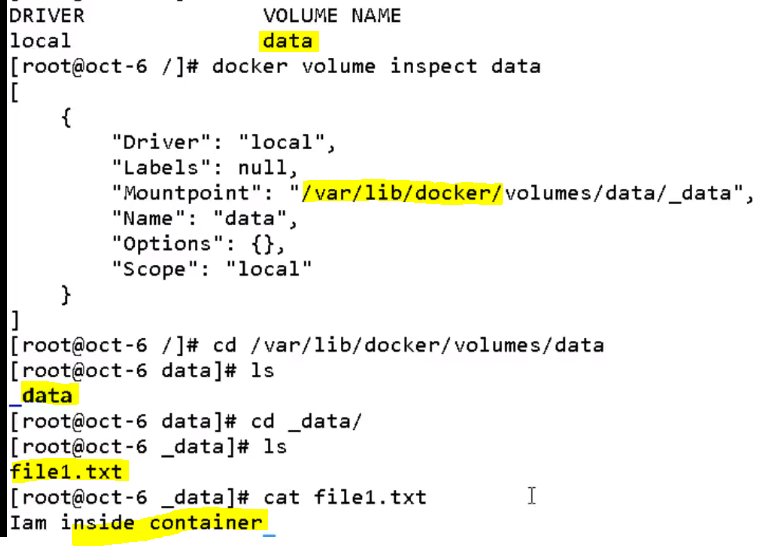
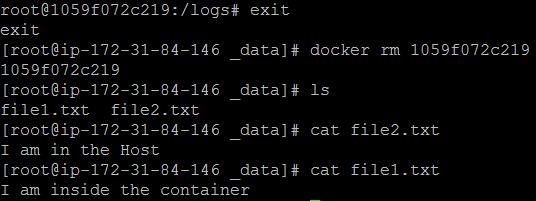
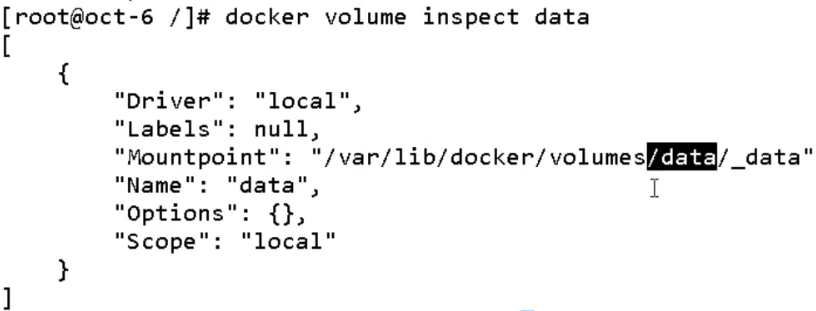
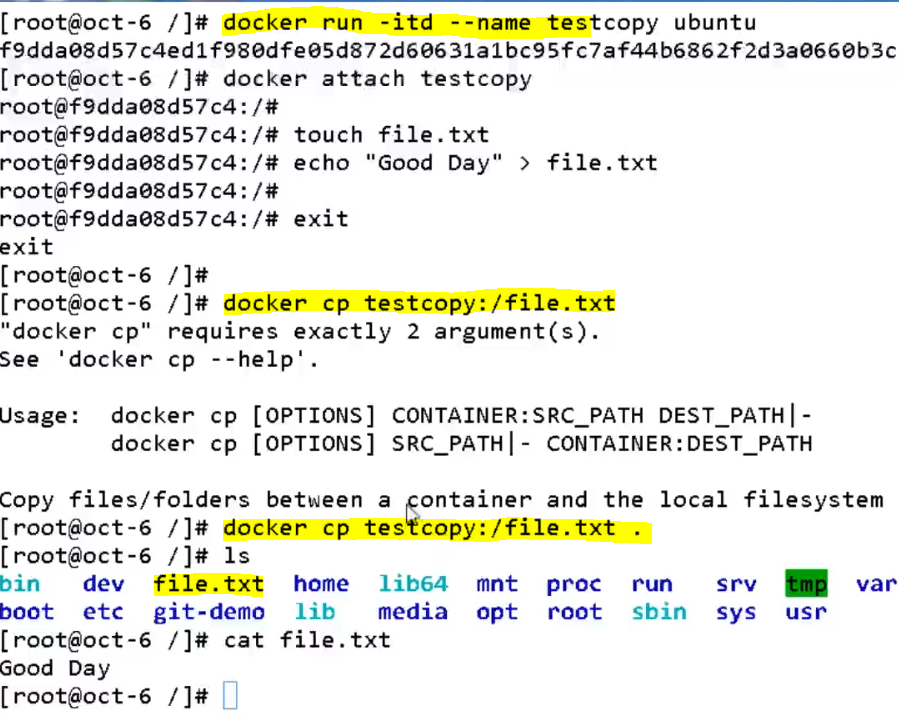
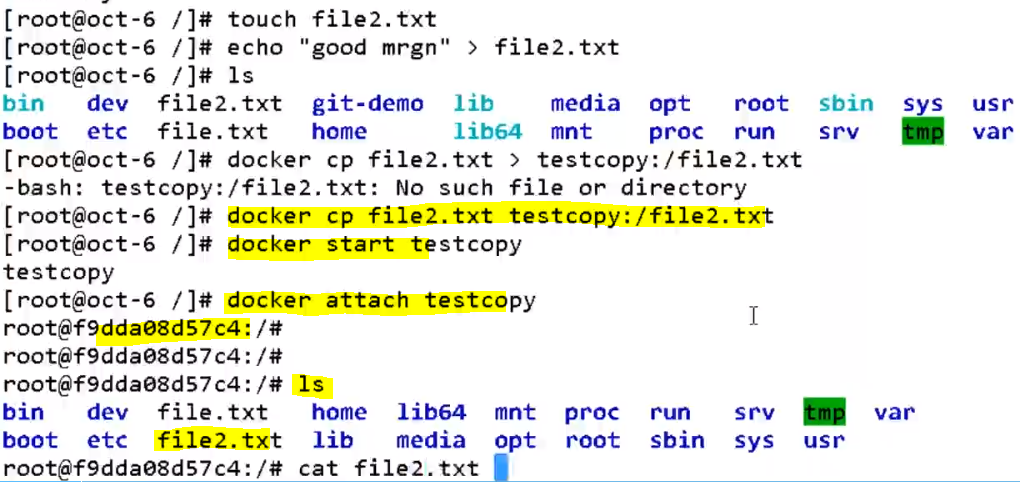
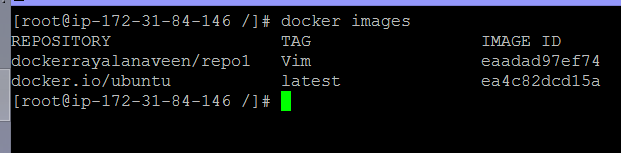
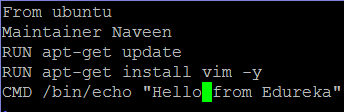
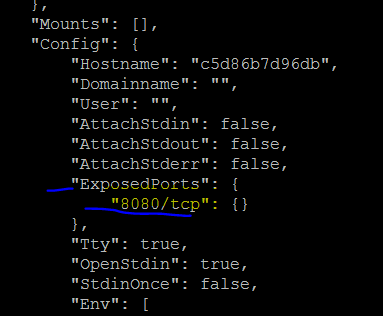
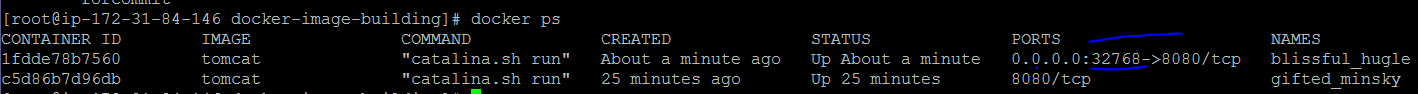
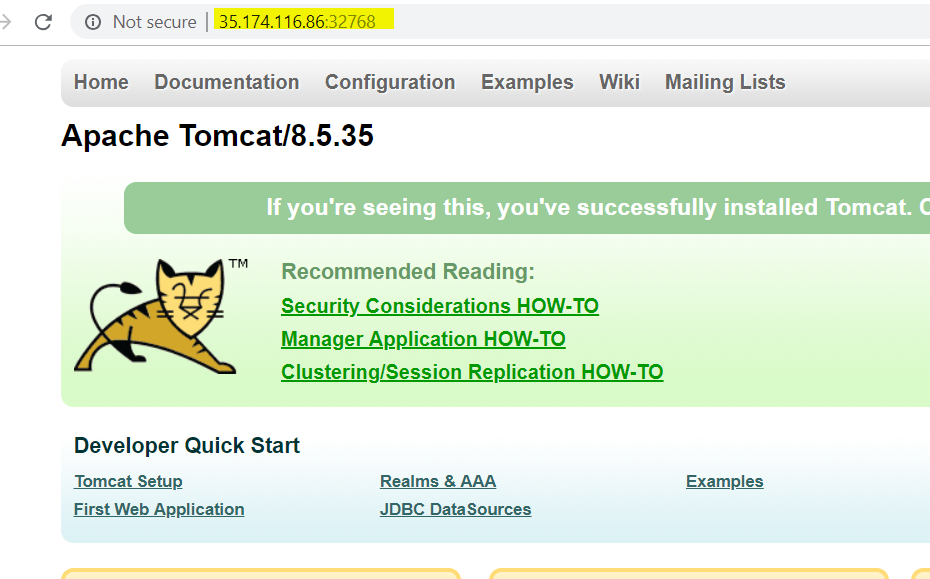
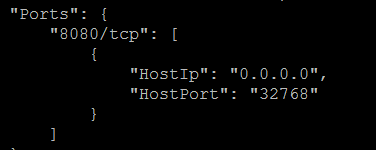
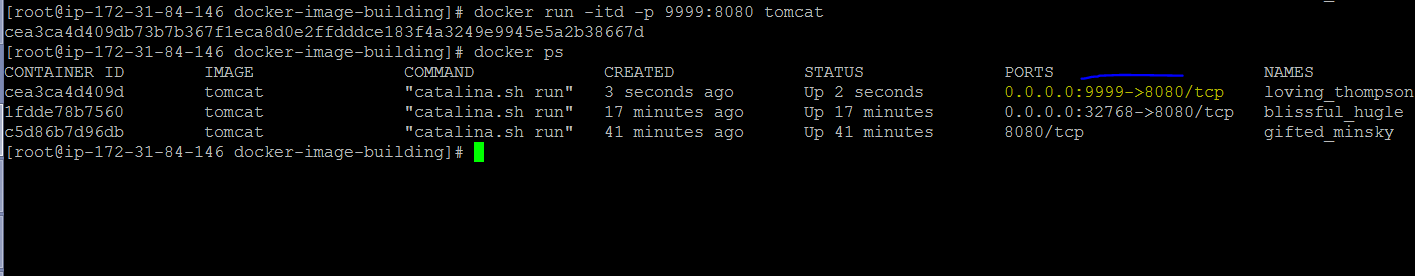
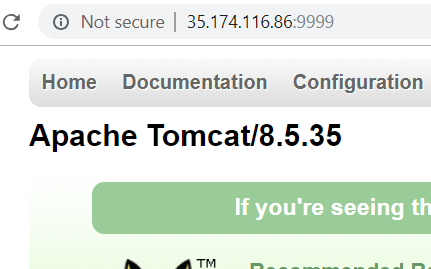
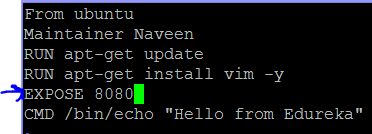
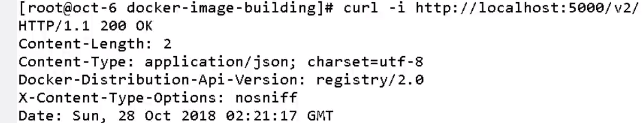
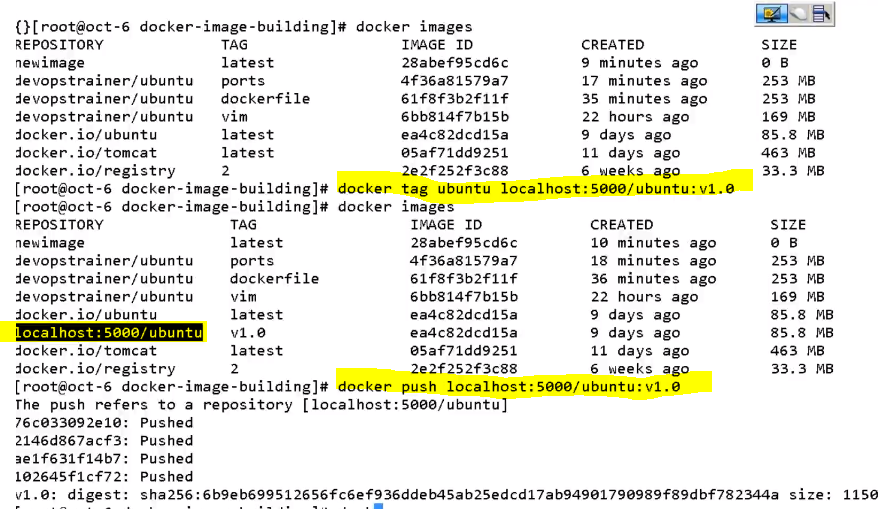
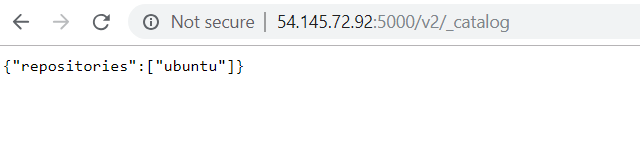
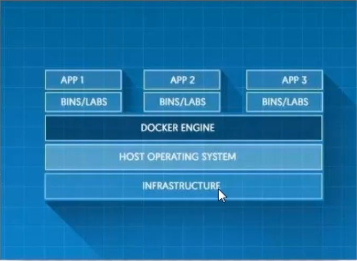
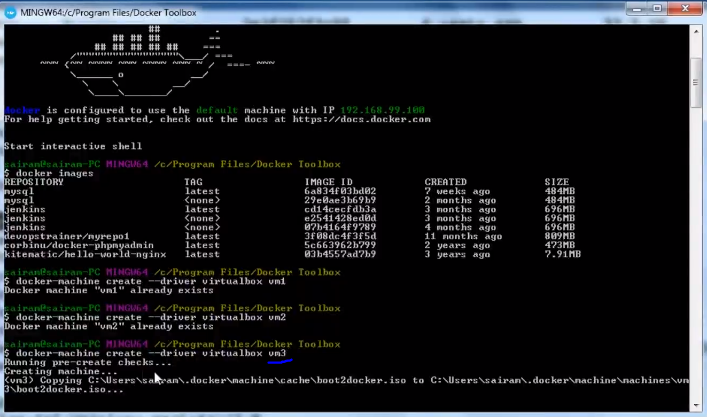
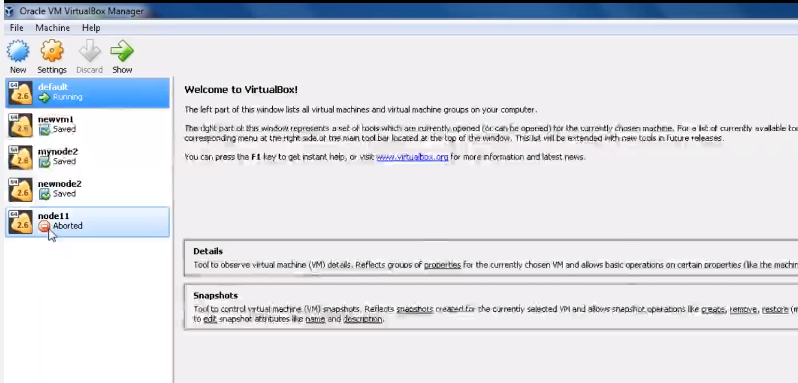
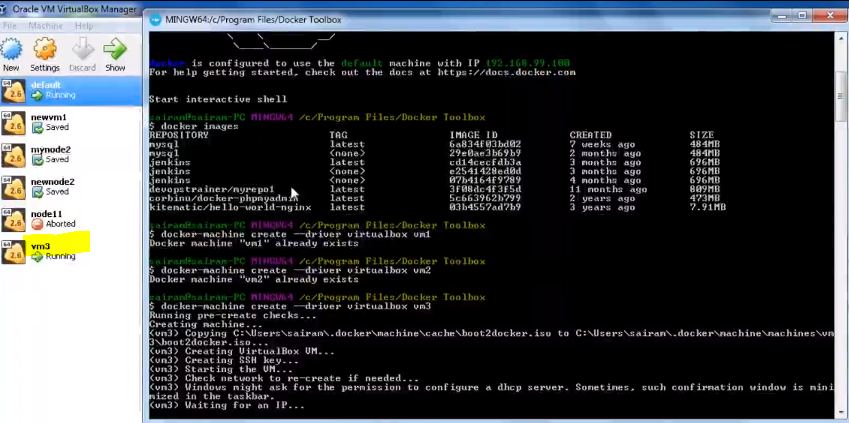
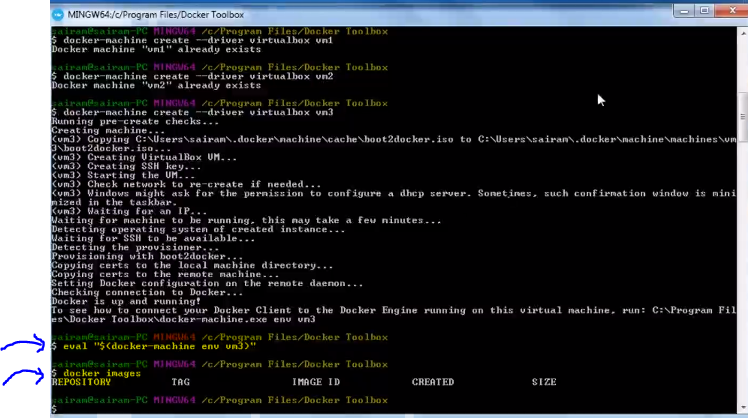
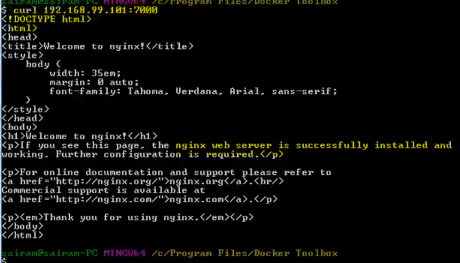
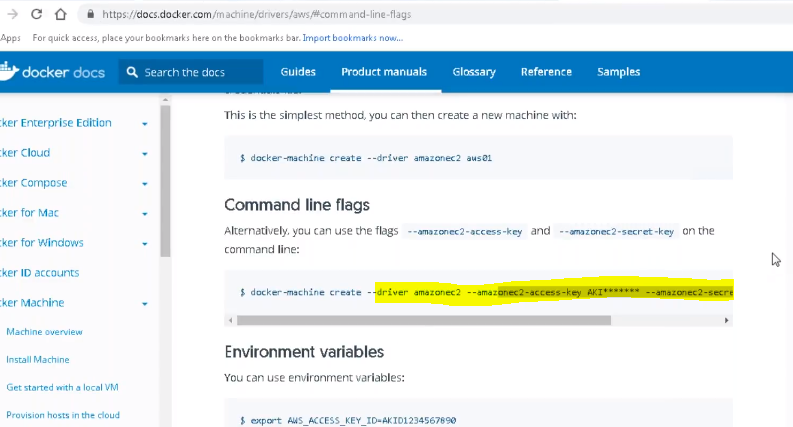
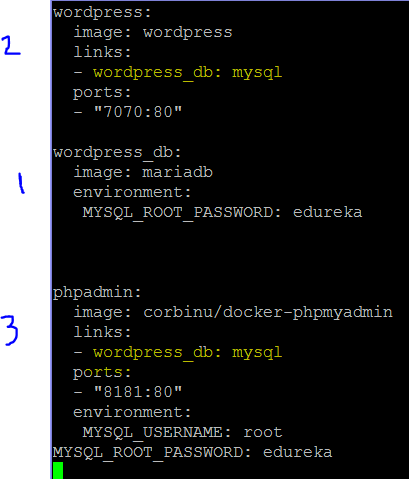
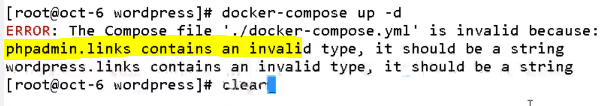
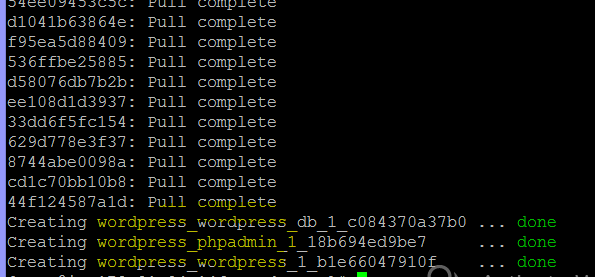
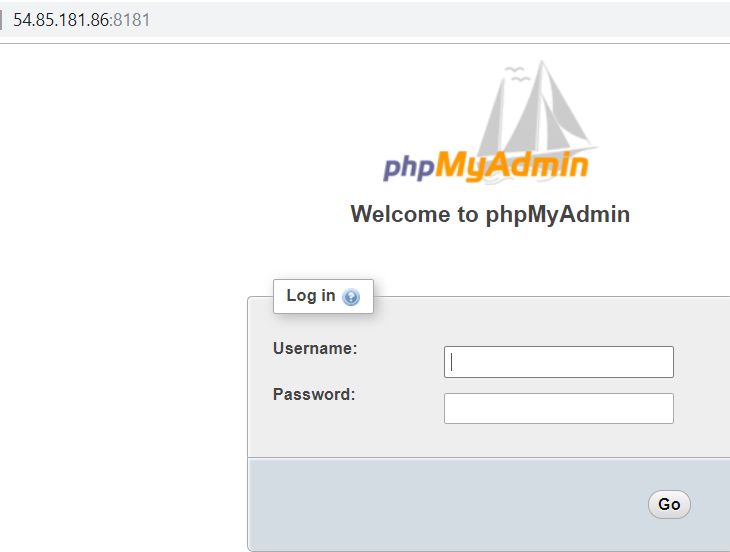
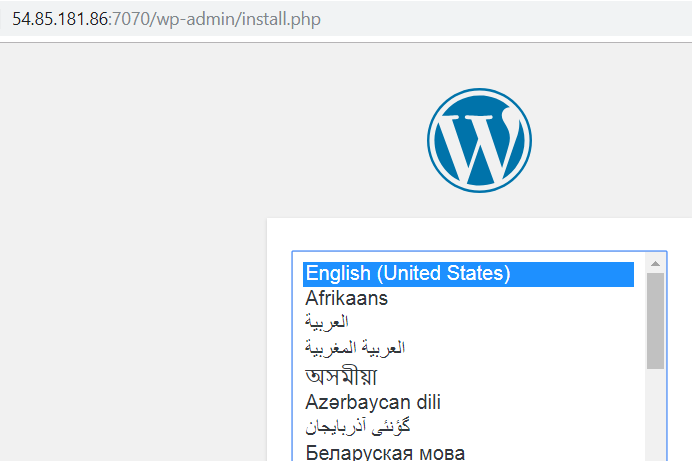
**Dockers Ideal Installation**

* **Docker File, Docker Image and Docker Container**: Docker file is built to get an image. Docker Image is a pile of file systems. Docker hub is public registry where you can store the images. Docker image is run to get the containers. If Docker Image is in public registry or local registry, pull it and run to create containers. Many containers can be created from one Docker Images.
* yum install docker
* systemctl start docker
* docker images (None will be there)
* docker run hello-world [It is pulling from public registry. As it is not in local, we see below statement unable to find image]
* docker ps
* docker ps –a (container will be seen but exited)
* docker images
* **Note**: Go to hub.docker.com and check for hello-world
* docker run --name mycontainer1 hello-world (to create a container from hello-world image. This time from local registry)
* docker ps –a (newly created and exited container will be seen)
* docker search ubuntu (To search for images, if any specific name as Ubuntu)
* **Note**: Registry has Images. Repositories has something similar to image. Check in docker.io for reference.
* Let us download some images
* docker pull dockerrayalanaveen/repo1
* docker images
* docker tag docker.io/dockerrayalanaveen/repo1 docker.io/dockerrayalanaveen/repo1:v3.0 (to tag and create the image from other one.)
* docker login -> Give dockerrayalanaveen, DevOps123
* docker push docker.io/dockerrayalanaveen/repo1:v3.0
* **Note**: Check in the docker.io under your name -> v3.0 will be seen.
* Docker ps –a [It is always exited. **How to keep them active**. Let us see]
* Docker run -itd ubuntu [Interactive terminal detached mode] [Pulled the image from repository by name ubuntu and creation of container without exiting]
* Docker images [image by name docker.io/ubuntu is created in image]
* Docker ps –a [The active container up will be seen]
* Docker stop containername -> to stop the active one.
* docker run -it ubuntu [to enter in to container as root. Now you can see if u can download apt]
* **Note**: Try yum install git –yum is not recognized. Try apt install git, it will work as this is ubuntu container.
* Exit [you will come out of container but container will be exited too].
* docker run -it ubuntu 🡪 Ctrl+p Ctrl+q {if you want to come out but keep the container alive]
* docker start containername [Exited docker container will start].
* **Note**: docker start – starts the stopped container. Docker run creates new container from image.
* Docker ps –a [check the instance, it should be up]
* Docker attach containername -> enters inside as root. Exit to come out.
* Note: If you have stopped the instance and you are back, give systemctl start docker
* Docker rm containername 🡪 deletes the container name.
* **Note**: You cannot delete a running container. So stop it and delete it. If you still want to, use –f. docker rm containername –f. If you want to delete image, docker rmi imagename.
* Docker run ubuntu or docker run Jenkins (pulls images and creates containers)
* **If you want to create a container, and you are inside container and want to share files with the host and vice-versa, there is a concept called volumes. Let us see**
* Docker volume ls 🡪 shows if there are any volumes created.
* **docker run -it --name test1 -v data:/logs ubuntu** [ run is for creating container. ‘it’ is for going in to container, name is the notation to create container name, test1 is the name of the container, v is for volume, data is folder created in the host (not container), logs is created in container and ubuntu is the image name from which the container is created.
* Now you are in the container. So
* ls [You will see logs directory. Now cd logs and ls –nothing will be there. Now create a file there]
* touch file.txt
* echo “I am in the container” > file1.txt
* exit [Come out of container. Now the file created in container should reflect in the host.]
* docker volume ls [you will see data folder created]
* 
* Now create a file in the host and give content
* Touch file2.txt
* Echo “I am in the host” > file2.txt
* Cat file2.txt
* Now enter in to the container and see if you can see the file2.txt. Before that how to enter in to container as it is already exited when you have come out. So first start and then attach
* Docker start containername {before this, do docker ps –a to see container name. Also ensure IMP: you have to stay in the same path. Cd /var/lib/docker/volumes/data/\_data]
* Docker attach containername
* Now you will be in the container. Cd logs -> ls -> you will see file1, file2
* Cat file2.txt -> I am in the Host -> it means you can see data interchangeably from host/container.
* Now if you exit from container, will the data be available in Host – lets see
* 
* **Where is the data in host? Below. Is this mountpoint sufficient if there are 100s of containers?** 
* If yes, lets see how.
* **docker run -it --name test2 -v data:/log1 ubuntu** [you will be logged in to container. Cd log1 and then ls. You will be seeing fil1.txt and file2.txt. It means you can access the data on host from any container created]
* Now if we create a container without volume, and then we want to attach volume to container. Let us see how 
* In the above example, it is important to use “cp” to copy the file from container to root of host.
* If you want to create file and then login in to container and access, below is to be done: 
* **Now let us see if we can create docker image from docker container. Ideally docker images are created from Docker files**.
* Remove all the containers and images and keep it clean. Docker rm cname and docker rmi image.
* docker run -it --name forcommit ubuntu [create container and enter it from image ubuntu]
* IN the container, apt-get update and apt-get install vim
* Vim [vim details will be displayed] – At this stage, container has installation like Vim. If any new container is created with vim, what needs to be done – let us see
* Control+P and Control+Q and come out with container being active.
* Docker ps –a
* **docker commit forcommit dockerrayalanaveen/repo1:Vim** [commit is the keyword to create image from container by name forcommit. Image is put in dockerrayalanaveen/repo1 with name Vim]
* **docker images** [now see the new image create] 
* docker run -it newimagecreated [Give the new image name and ‘it’ helps to enter in to container]
* vim [now do vim to see if it is already there]
* **Now let us see how to create docker image from docker file**
* Mkdir docker-image-building
* Cd docker-image-building
* Vi dockerfile
* 
* docker-image-building] # **docker build -t dockerrayalanaveen/repo1:dockerfile .** [**Note**: Command to get a docker image from docker file. Every command in the docker file will create a file system layer. Each and every statement will be executed. The MOST IMP thing to NOTE, in the script statement dockerfile is not the filename. It is the tag name for newly built image with name dockerrayalanaveen/repo1. How come we did not give the dockerfile (which has script) is not in the script? It is by default as we used **build** ] **O/P**: Successfully built 6ab584579936
* docker images
* 
* The above message came as we gave /bin/echo, so it is showing message from bin. To enter in to container, we need to override that, how? As below
* Docker run –it dockerrayalanaveen/repo1:dockerfile /bin/bash
* Now you will be in the container.
* Vim [type vim and see the vim being displayed] & exit
* **Now let us see the next docker file –docker image scenario**
* Docker images
* Docker history dockerrayalanaveen/repo:dockerfile -> it displays the history of image created. 
* Initially the image built was 85 MB but in that above updates were there and it became 59.5 MB
* Docker run –itd tomcat (container to be built from tomcat image. ‘itd’ for keeping the container up)
* Docker images [tomcat image should be displayed]
* Docker ps (will show active container as up)
* Docker inspect containername [Now you will see the ipaddress of the container]
* 
* 
* **Note**: The IP here is of container and is on port 8080 and not of Host. This is very important. If you want to expose container ID on to host, then we have to do **port forwarding**. Meaning: the container:8080 will be mapped or port forwarded to host:xxxx (random IP which we can give), then we can see tomcat services. If you do curl IPofContainer:8080, you will in results –tomcat is installed successfully. If you curl HostIP:8080, it is not giving anything. So port forwarding is important to see on GUI. To do port forwarding, we should check if ports are available. Above, expose is for 8080 and is for null. How to do port forwarding?
* docker run -itd -P tomcat [to create the container, P is for port and tomcat is image]
* docker ps
* 
* Now you see the port number generated as 32768
* Check the Host IP:32768 in browser, tomcat page will be displayed -> 
* Now do docker inspect containername and check if port is available or related to 32768. 
* If you do – curl internalIPofHOST:32768 in CLI, you will see the message as well.
* Now this 32768 is the random port given by system as we used –P in our script. To give port of our choice, we can give –p (small p) and give the port number. Let us see how it goes
* Docker run –itd –p 9999:8080 tomcat [If it is capital P, It is **free port** allocation. If it is small p, it is **binding port**]
* 
* So now, with same host we have 2 tomcats appearing. How? With 2 containers, we can say host:portfromIPviaContainer1 and host:portfromIPViacontainer2.
* Vi dockerfile -> and open the existing script of dockerfile 
* docker build -t dockerrayalanaveen/repo1:ports .
* docker ps –a [All are exited]
* Select the container created and exited – number.
* Docker inspect containernametheexitedone
* Check that the exposed port is 8080
* **Now let us see how we can share containers** **using Tar Ball**
* Docker ps –a
* Docker export containername > update.tar [To share the container, use export and give the name as update.tar – this is basically tar ball.]
* ls [You will see update.tar in the list]
* **Now let us see how we can create image from the container stored in tar ball**
* Docker import - newimage < update.tar
* Docker images (you will see the image created with name as newimage)
* **Now let us see how we can convert image in to a tar ball**
* docker images
* docker save –o update1.tar imagename
* ls [You will see updat1.tar]
* docker rmi newimagename [ we are removing as the new one will be creating with same name]
* docker load < update1.tar
* docker images (New Image is created)

|  |
| --- |
| **docker container to tar ball | tar ball to image**  docker export containername > update.tar | Docker import - newimage < update.tar    **docker image to tar ball | tar ball to image**  docker save –o update1.tar imagename | docker load < update1.tar    **Below is the recap again**  container -> **export** ->tarball -> **import** -> image  image -> **save** ->tarball -> **load** -> image |

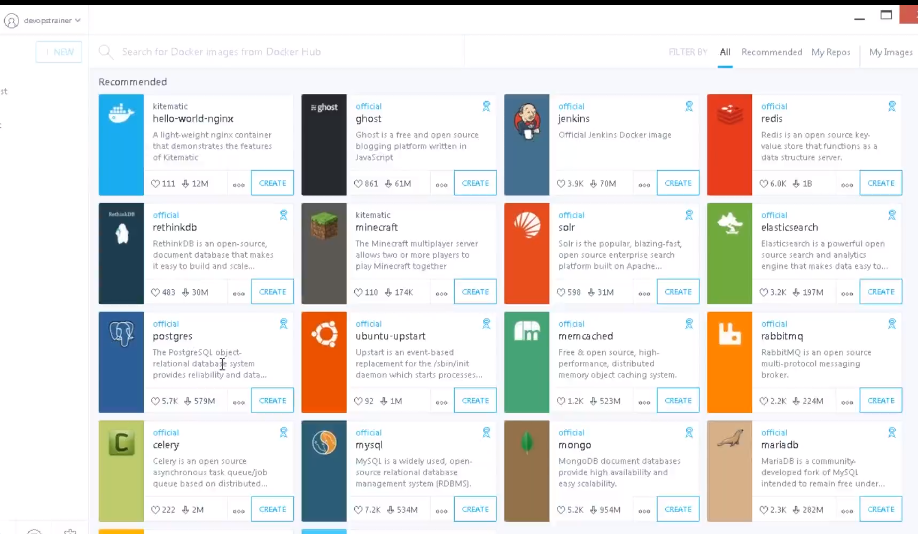
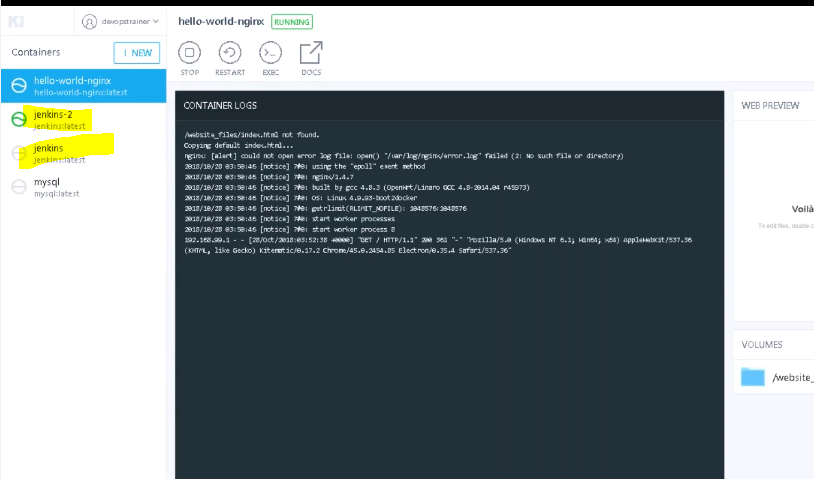
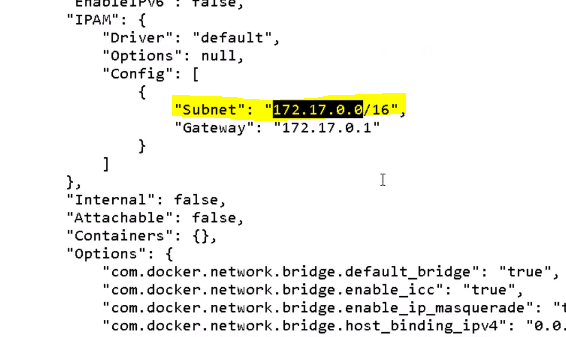
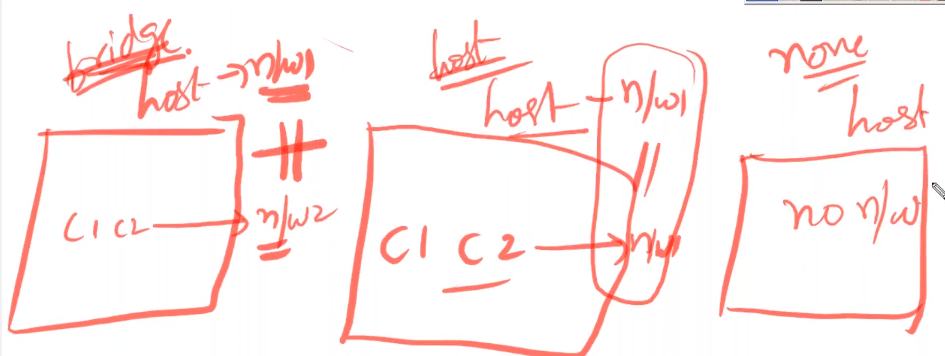
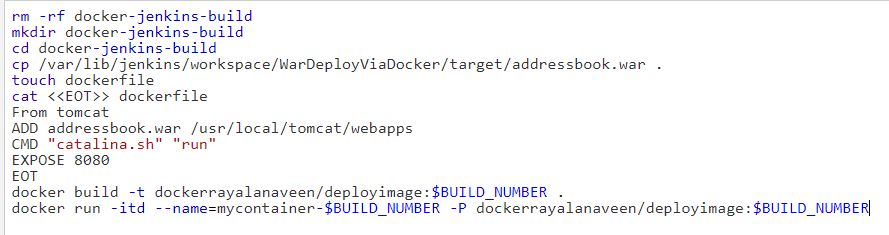
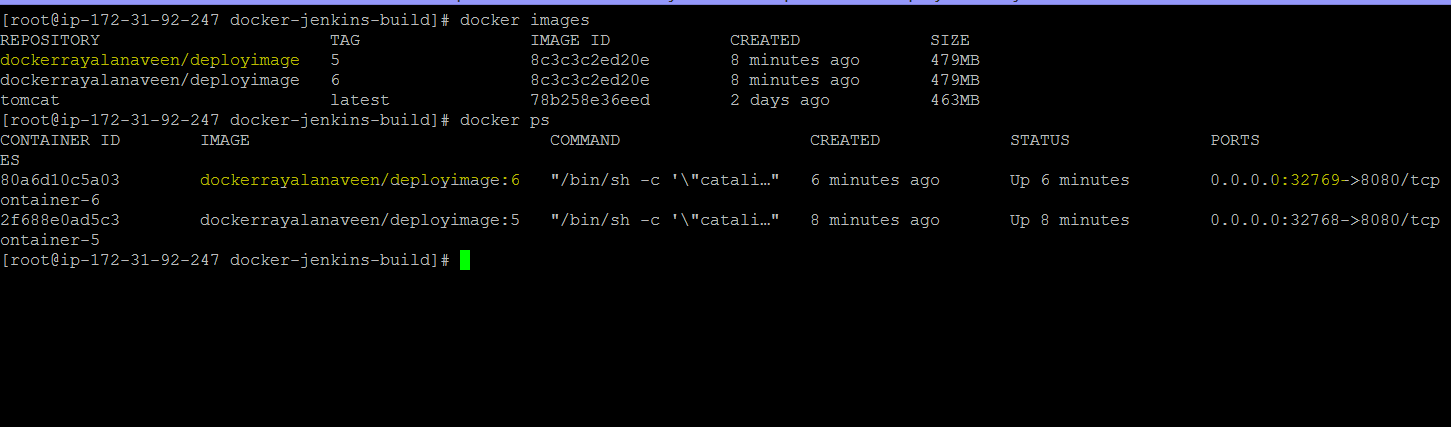
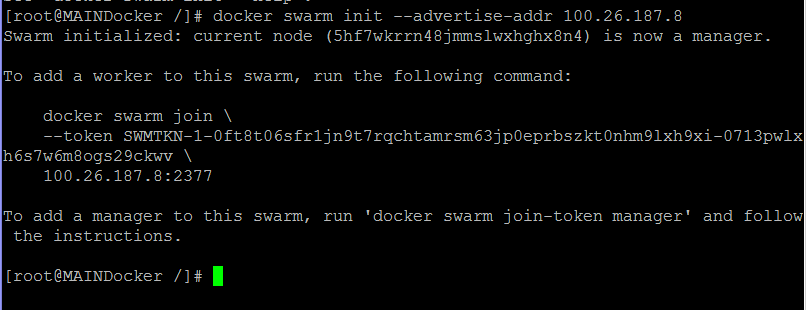
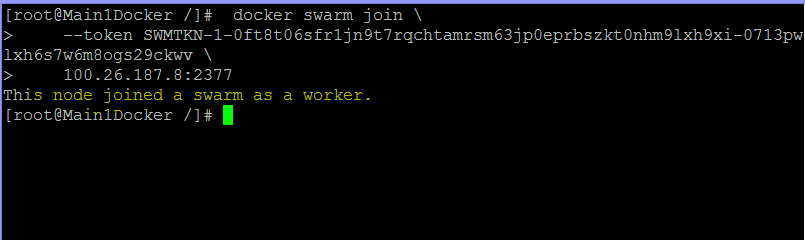
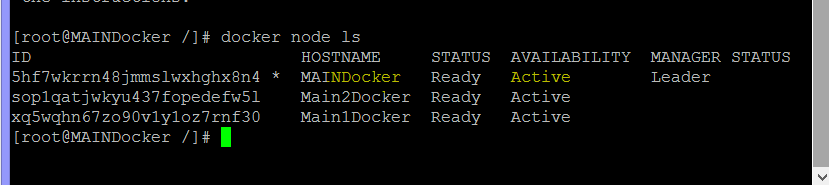
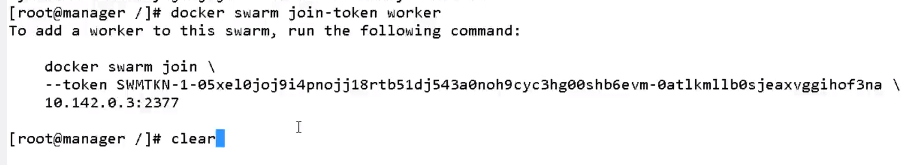
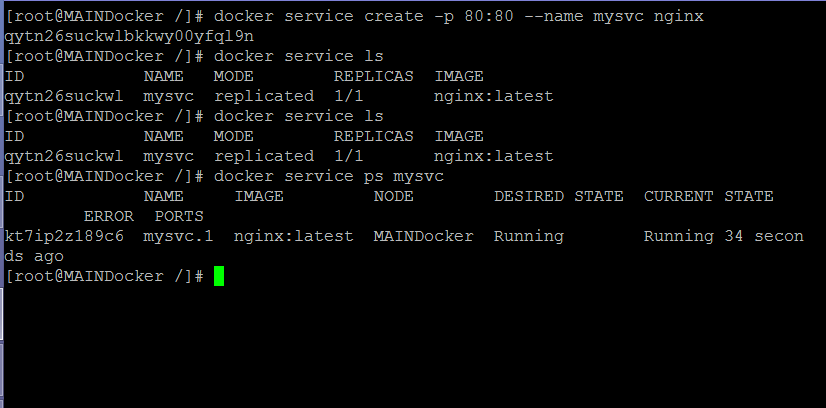
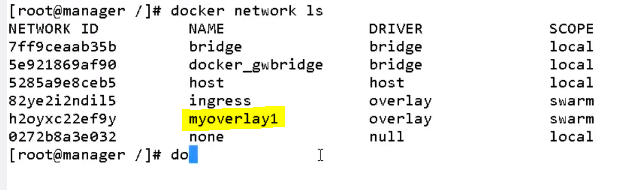
* **In hub.docker.com, orgs will have their own registries. We cannot create our own one but let us see how to work with our private registries**. **First pull the registry from public and work accordingly**.
* Docker pull registry:2
* Docker images (you will see the image being created)
* docker run -itd -p 5000:5000 registry: 2 [we are not giving docker.io/registry:2 as if we don’t give, it takes docker.io as default. We are giving 5000 as that is what mentioned in hub.docker.com site. Check by typing registry and see tags and scroll down to see port numbers]
* docker ps
* This one is the default path itseems and so remember: 
* Docker images
* 
* Above is creating image from image but changing the name from docker.io to localhost. Tagging the name of ubuntu in the images. Here ubuntu is image. and 5000/ubuntu -> here ubuntu is name. The push is to repository but not docker hub. The below script statement shows that repository **ubuntu (name)** is created.
* 
* Below – give publicIp:5000/v2/\_catalog to see the repository created.The below is the confirmation of registries under the path v2/\_catalog



* **Now let us look in to Docker Ecosystem**
* **Docker Host** is where docker daemon, docker service, VMs are running.
* **Docker Machine** is a tool which can spin up any number of Docker Hosts**.**
* **Docker Engine** is docker daemon which runs docker commands.
* **Kitematic** is desktop GUI for docker**.**
* **Docker tool box** is available only for windows and mac. (For older versions)
* **Docker** natively runs on **linux**. It does not run natively on windows. It is from **windows 10**. So for previous versions, a pre-configured shell is given for docker command-line environment**.**
* 
* Install Docker ToolBox – for windows from doc.docker.com
* Click Docker Quick Start in Windows. You will get a prompt.
* Docker images
* Let us see the code of docker machine which can spin up many hosts where docker is running. You can directly use docker –machine. Installation is not required as it is in-built in toolbox.
* Docker-machine create –driver virtualbox vmname 🡪 give anyvmname. 
* Now open (launch) the oracle virtual box . The name vm3 is yet to kicked off. (Virtual Machine is not up as of now)
* Now see vm3 is running (VM is in the background)
* Now you need to get in to VM3, there are two methods. First one is to double click vm3 in the VMbox. Or you put up a code in docker tool box. In the below code, vm3 will be up. How would you know – by docker images, see no images are there. It is new machine vm3.
* Now continue as normal coding structure.
* Docker run –itd –P 7000:80 nginx [80 is the default port for nginx and so 80]
* If error comes, login first in to docker hub [by docker login 🡪 give username, password
* Next also error came. Why-> capital P is given which is free port but you gave 8000. So give small p as it is binding port.
* Docker run –itd –p 7000:80 nginx
* **Remember** – All these are happening in VM3 Host instigated by Docker machine.
* If you see the IP address of VM3 in port 7000, nginx should be there. How would you know IP of vm3 -> docker-machine ip vm3 will give output. [vm3 is like linux host where we take IP and give port]
* See if nginx is successfully installed. 
* **Read the Docker Machine or Engineer related notes in** <https://docs.docker.com/machine/overview/>
* 
* Above is the tool to spin up VMs. These VMs spun by Docker machine is using BOOT2Docker OS. This is minimal linux. You can change OS if you want.
*  Docker compose is about running multiple containers. This is related to micro services architecture.
* Close all windows. Let us work back on linux machine.
* Docker compose comes with python pip manager.
* Yum install python-pip
* pip install docker-compose
* pip install --upgrade pip
* Now we want 3 containers where one will have WordPress, mariaDB and PHP
* Mkdir wordpress
* Cd wordpress
* Ls
* Vi dockercompose.yml
* 
* In the above file, first wordpress\_db will be created. Then wordpress and then phpadmin. What is PHPAdmin? phpMyAdmin is a free web application that provides a convenient GUI for working with the MySQL database management system. It is the most popular MySQLadministration tool that is used by millions of users worldwide and has won numerous awards and honors.
* Save
* docker-compose up –d [You do not have to give name as it is standard file in the same directory]. However some issue came. 
* Below is the change. There should not be space between “:” and “php” 
* Docker-compose up –d [pulls the images and builds the containers and completes the task] 
* 
* Three containers and three images were created and downloaded accordingly. All containers created are up. Now check with hostIp and port 7070 and 8181. You should see phpmyadmin and wordpress.
* 
* 
* Enter the details and get in. For PHPmyadmin, give the credentials. [root/edureka]. This is how all work in sync. This is how microservices work too.
* Docker-compose down [will bring all the containers down]
* **Note the docker-compose concept of building and creating containers and bringing them down. Recap**

Docker-compose up –d

Docker-compose down

* Docker images -> will be there
* Docker ps -a [all containers will exited and removed]
* Docker Kitematic -> This is getting Docker GUI. Click the kitematic icon. Give the docker hub credentials and login. 
* Click create of anything you want. Jenkins will be downloaded. Click the settings. 
* **Introduction to Docker Networking Concepts**
* 
* Default networks are bridge, host and none which come as part of docker installation.
* Docker network inspect bridge 
* Default network is in **Bridge network**. Container and Host are in separate IPs. If you want to access, do port forwarding. This is taken care by docker Ethernet. What is **Host Network**? It means all the containers and host are having same IPs. In this n/w, no confidentiality and everything is accessible. If confidentiality needed, use bridge network. **None Network** means no network needed. 
* Docker Swarm and Kuberenetes are container orchestration tools.
* **However let us see how to deploy in to tomcat using Docker in Jenkins**.
* /var/lib/jenkins/workspace/PackageforWarFile/target – verify if there is war file
* Write the script in build of Jenkins: 
* Run the script. If there is permission issue, give the below
* \*\*\*\*\*\*\*\*\*\*\*Possible Solution\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* **sudo usermod –a –G docker Jenkins**
* **systemctl start docker**
* \*\*\*\*\*\*\*\*\*\*\*End of Solution\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Run and it should be success. Check in CLI.
* Cd /var/lib/jenkins/workspace/dockerdeploy/docker-jenkins-build [dockerdeploy is the job, docker-jenkins-build is the directory]
* Ls
* You will see addressbook.war and dockerfile
* Cat dockerfile to see the script
* Now do -> docker images and docker ps
* 
* To deploy the war file, do the below: [HostIP+port given 32769] 
* **Docker Swarm – Now let us see what Docker Swarm Does**:
* It is about managing containers in cluster of docker hosts. It helps to manage high availability, load balancing, scaling, rolling updates etc. If a node or host is down, all the containers inside it will be up in other host using this feature Docker Swarm.
* Create 3 instances and ensure docker is installed in each of it. Docker swarm is not needed in any of the instances. It comes by default. Names -> Main, Main1 and Main2.
* IN Main1 🡪 docker swarm init –advertise-addr 100.26.187.8 (This is IP of main) TO make this manager. 
* Paste the below in both hosts main1 and main2 to make them workers. 
* IN the Main -> docker node ls
* 
* If you missed the token to join as worker – in the main, give **docker swarm join –token worker**
* 
* Let us run containers in the main one. Remember, we run containers in the form of service wrt Docker Swarm.
* Now take IPs of main, main1 and main2 and put 80, nginx is shown.
* This is called cluster wide port.
* Now let us drain one worker (Main1 or main2) – and see if the container is accessible from other soures. It should.
* 
* The above is to create container in the cluster (Main, Main1 and Main2). The image is of nginx. The container name is mysvc and the port forwarding is to 80. Pick any Host IP (main, Main1 or main2) and 80 port, it should show nginx. This is the beauty of Docker Swarm service.
* Check if the container is available – yes it is. To check where it is, docker service ps mysvc -> you know in the scenario, nginx is running in Main.
* Docker node update –availability drain worker1 [it means worker1 is exited and you can check the same in other host IPs]
* Docker service ps mysvc -> shows in which it is working.
* Now scaling up is required: How?
* Docker service scale mysvc=10 🡪 mysvc scaled to 10.
* Docker service ls -> 
* Docker ps on main, worker1, worker2 -> shows how many are working in each hosts
* If you want to up any specific host-> docker node update --availability active worker1 (if want to remove instead of active worker use drain worker.)
* Docker node ls -> to check if all the nodes are active or inactive (main, worker1 and worker2)
* Now let us drain manager. For it, do -> docker node update –availability drain main
* Docker service ps mysvc -> displays 10 containers with worker1 and worker2 division as manager is drained. This is **Auto-Balancing, Scaling up and Scaling Down**.
* There are three types of Networks -> bridge, host and none.
* Ideally when you say docker network ls – above 3 should come. But now you will have 5
* 
* Ingress -over lay sits above the host networks. (Assuming we have 3 hosts). **Ingress** will help containers in different hosts to communicate with each other. Docker\_gwbridge will help to connect the host network and ingress overlay network.
* Docker ps in all hosts and see which containers are displayed in what. Now do -> docker inspect container-name-of-that-host 🡪 
* If we have 1000 containers in swarm mode across manager, worker1 and worker2. All are attached to ingress. It is important to create our own network. I want 100 to interact with other 100. I do not want 800 to come in to this. Like 100 in overlay1 and 800 in overlay2. Something like that.
* Docker service rm mysvc [removing previous service]. We want to create our own service.
* Docker service ls -> nothing will be there
* **Docker network create –d overlay myoverlay1** -> create a new network where subnet network interface will be taken care by docker.
* **Docker network ls** -> 
* **Docker service create –name newsvc –network myoverlay1 –p 8000:80 nginx ->** service will be created.
* Docker service ls -> service will be seen.
* Docker service ps newsvc -> it shows where it is working main or worker1 or worker2
* Go to worker-1, docker ps -> take the container name
* Docker inspect containername -> in worker1 and see the output 
* This is Docker Swarm.
* **What is rolling update**? We need to upgrade the version. End users will downtime. So no user down time and roll updates using Docker Swarm. More efficient in Kubernetes. Docker and Kubernetes are called **container orchestration**.
* We cannot have one docker container in two networks? **No**, we cannot.
* There is NO auto-scaling but scaling up and scaling down is there using Docker Swarm.
* **Docker** is a **containerization** tool. **Docker Swarm** is a **container orchestration tool**.