**Generate the SSH Key**

Mac/Linux

<https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2>

Windows

Putty tool Links:

Generating a Key with **PUTTYGen**

<https://www.digitalocean.com/docs/droplets/how-to/add-ssh-keys/create-with-putty/>

Configuring **PUTTY** to access DO:

<https://www.digitalocean.com/docs/droplets/how-to/connect-with-ssh/putty/>

**Create VM’s**

Steps for creating the VM’s Manually / Automatically

**Connect to the VM**

#ssh root@ipaddress

ssh -i <path of privete key> root@ipaddress

**Install Docker [suitable for Kubernetes]**

# nano docker.sh

*# Install Docker CE*

*## Set up the repository:*

*### Install packages to allow apt to use a repository over HTTPS*

apt-get update && apt-get install -y **\**

apt-transport-https ca-certificates curl software-properties-common gnupg2

*### Add Docker’s official GPG key -* ***GNU Privacy Guard***

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

*### Add Docker apt repository.*

add-apt-repository **\**

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

**$(**lsb\_release -cs**)** \

stable"

*## Install Docker CE.*

apt-get update && apt-get install -y **\**

containerd.io=1.2.10-3 **\**

docker-ce=5:19.03.4~3-0~ubuntu-**$(**lsb\_release -cs**)** **\**

docker-ce-cli=5:19.03.4~3-0~ubuntu-**$(**lsb\_release -cs**)**

*# Setup daemon.*

cat > /etc/docker/daemon.json <<EOF

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

EOF

mkdir -p /etc/systemd/system/docker.service.d

*# Restart docker.*

systemctl daemon-reload

systemctl restart docker

Make file executable:

chmod +x docker.sh

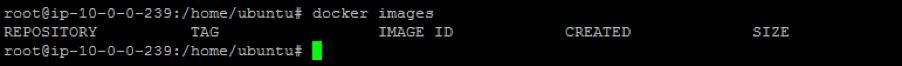
./docker.sh

**List containers**

docker ps -a

**List images**

docker images



**Obtain details information about your host**

docker info

**Image Operations:**

**Observe the directory Before creation of container**

ls -l /var/lib/docker/image/overlay2/imagedb/content/sha256

**Deploy a Ubuntu server container**

**Step 1 : docker pull ubuntu:18.04**

**Step 2 : List Docker images**

**docker images**

**docker image ls**

**Step 3 : Delete a image**

**docker rmi <image id or name:tag>**

**Step 4 : Docker image history**

**docker image history <image id>**

**Step 5 : Image inspect**

**docker image inspect <image id>**

**Step 6 : Remove unused images**

[**docker image prune**](https://docs.docker.com/engine/reference/commandline/image_prune/)

**Step 7 : Remove image**

[**docker image rm**](https://docs.docker.com/engine/reference/commandline/image_rm/) **<image id>**

**docker rmi <image id>**

**docker rmi -f <image id>**

**Container Operations:**

**Observe the directory Before creation of container**

ls -l /var/lib/docker/containers/

**Deploy a Ubuntu server container**

**Step 1 :docker run -it ubuntu:18.04 /bin/bash**

**#uname -a**

**docker run --name some-nginx -d -p 8080:80 nginx**

**Step 2.A : list the running container**

**docker ps**

**Step 2.b : Login to the container**

**docker exec -it <containerid> /bin/bash**

**Step 2.C : list process running in container**

**root@04707a043ac9:/# ps -ef**

**Step 3 : exit from the container**

**root@04707a043ac9:/# exit**

**Step 4 : Observe the directory After creation of container on docker host login**

**ls -l /var/lib/docker/containers/**

**Step 5 : Listing the Docker images**

**docker images**

**Step 6 : List the running containers**

**docker ps**

**Step 7 : Fetch info about container**

**docker inspect 04707a043ac9**

**Above info comes from**

**/var/lib/docker/containers/0f4ea7ba2e814169377263918a9cb027f72f0295c45dc96f8661d0630501bc27/config.v2.json**

**Step 8 : Login to your container**

**docker exec -i -t 04707a043ac9 /bin/bash**

**If you successful login you should be in your container bash login as below**

**root@04707a043ac9:/#**

**Step 9 : Validate your IP address of container**

**root@04707a043ac9:/# ip a**

**Exiting the container**

**root@04707a043ac9:/# exit**

**Another way to shutdown your container ( login as root on your host)**

**docker inspect 3b6bf78299b0 | grep Pid**

**kill 4683**

**You will notice container is stopped**

**docker ps**

**Container Operations Part II:**

**Step 1 : List the running containers**

**docker ps**

**Step 2 : List all the containers**

**docker ps -a**

**Step 3 : Look at the CPU ,Memory and storage, network performance of container**

**docker stats 04707a043ac9**

**Step 4 : Restarting the container**

**docker restart 04707a043ac9**

**Step 5 : Pause the container**

**docker pause 04707a043ac9**

**Step 6 : unPause the container**

**docker unpause 04707a043ac9**

**Step 7 : Rename your container Name**

**docker rename 04707a043ac9 web-server01**

**Syntax : docker rename <continerid> <desired name>**

**Step 8 : Stop your container**

**docker stop 04707a043ac9**

**Step 9 : Start your container**

**docker start 04707a043ac9**

**Step 10 : Delete the container**

**docker rm 04707a043ac9**

**Note : You cannot delete a container which is in start state**

**docker stop 04707a043ac9**

**Step 11 : Delete the container**

**docker rm 04707a043ac9**

**Step 12 : Validate that the container is deleted successfully**

**docker ps -a**

**And also validate container doesn’t exist in /var/lib/docker/containers**

**Force delete of a container when the container is running**

**docker rm -f 04707a043ac9**

**Delete all containers which are not running:**

**docker rm $(docker ps -a -q)**

**Step 13 : Namespaces**

Switch between the process namespaces by using nsenter, get the process id and run the below commands with specific name spaces.

**Step 13 a: Verify PID, Namespaces, CGroups**

**docker run -d --hostname hname --name test ubuntu:18.04 sleep 5000**

**ps aux | grep sleep**

**docker inspect test | grep -i pid**

**apt-get update**

**apt-get install net-tools**

**ifconfig**

Note: UTS = Unix Time Sharing, where the this manages the multi-user or multi-tasking isolation.

**nsenter --target <pid> --uts**

**hostname**

**nsenter --target <pid> --net**

**ifconfig**

**exit**

Note: Exit from nsenter to get back normal.

**Step 13 b: Verify the host name:**

**Docker exec -it test sh**

**/ hostname**

**/ ifconfig**

**exit  
hostname**

**ifconfig**

here the result of these commands will be different as the namespace associated with each process is different.

**Step 14 : CGroups**

**cat /proc/<pid>/cgroup**

**cat /sys/fs/cgroup/memory/system.slice/<docker-id>/memory.limit\_in\_bytes**

**docker run -d --memory 400m --hostname hname1 --name test1 ubuntu:18.04 sleep 5000**

**Volumes:**

**Step 1 : create a docker volume**

## **Create and manage volumes**

**Unlike a bind mount, you can create and manage volumes outside the scope of any container.**

**Create a volume:**

**$ docker volume create data-volume**

**List volumes:**

**$ docker volume ls**

**Inspect a volume:**

**$ docker volume inspect data-volume**

**Remove a volume:**

**$ docker volume rm data-volume**

**Step 2 : create a container with volume**

**docker run -d -it --name web-host -v myvol2:/data ubuntu:18.04**

**////Notes**

**Myvol2 : is created on the host if it does not exist**

**/data : is created in container if it does not exist**

**docker volume ls**

**docker volume inspect myvol2**

**Step 3 : login to container and touch a file**

**$ docker exec -it web-host /bin/bash**

**Step 4 : create new new file in the /data directory and exit container**

**touch data/mydata.txt**

**ls data/**

**exit**

**Step 5 : validate if file is there on your existing host file system**

**ls -l /var/lib/docker/volumes/myvol2/\_data**

**Step 6 : remove the web-host container and create a new container**

**docker rm -f web-host**

**docker run -d -it --name new-host -v myvol2:/data ubuntu:16.04**

**Step 7 : login to container and touch a file**

**docker exec -it new-host ls -l /data**

**Docker Hub**

**Create an account at hub.docker.com**

**Create a new container**

**docker run -i -t --name container1 ubuntu:14.04 /bin/bash**

**root@9d9d6e9dc6dc:/# apt-get update**

**root@9d9d6e9dc6dc:/# apt install apache2 git default-jdk maven**

**root@9d9d6e9dc6dc:/# exit**

**dockecontainer1 myuser/ubuntu:1.0**

**docker images**

**You should see your image in the list.**

**docker run -i -t myuser/ubuntu:1.0 /bin/bash**

**try installing curl again : notice the difference**

**root@87656d52abac:/# apt-get install git**

**root@87656d52abac:/# which git**

**docker push myuser/ubuntu:1.0**

**docker login**

**docker push myuser/ubuntu:1.0**

**docker rmi -f <imageid>**

**docker images**

**docker run -it /ubuntu**

**Simple example:**

1. **docker run --name container1 busybox echo 'Hello modified Image’**

**Syntax:** Save the image with your user name as pre the registration done in docker hub.

*docker commit <container id/name> myuser/mybusybox:1.0*

1. **docker commit container1 myuser/mybusybox:1.0**
2. **docker run myuser/mybusybox:1.0**

**Push image to docker hub:**

1. **docker push myuser/mybusybox:1.0**

**Login:** Use the docker hub credentials

1. **docker login**
2. **docker push myuser/mybusybox:1.0**

**Delete local image:**

1. **docker rmi -f <imageid>**
2. **docker images**
3. **docker run myuser/mybusybox:1.0**

**Docker Network:**

**docker network ls**

**docker run -d --name web01 --rm yeasy/simple-web**

**docker container ls**

**docker network inspect bridge**

**You should see container in the list with a local IP like 172.17.0.2/16 with name web01**

**ping 172.17.0.2**

**docker container stop web01**

**docker container ls**

**docker run -d --name web01 -p8080:80 --rm yeasy/simple-web**

**Ifconfig -a**

**curl http://<<public IP>>:8080**

**You can open the same link in browser**

**Now let’s try HOST network:**

**docker run -d --name web02 --network=host --rm yeasy/simple-web**

**docker container ls**

**http://<<public IP>> in browser**

**Now let’s try NONE network:**

**docker run -d --rm --name mybusybox --network none busybox sleep 5000**

**docker container ls**

**docker exec -it mybusybox sh**

**#ifconfig**

**You will see no network interfaces other loopback**

**docker container ls**

**docker exec -it web01 sh**

**#ifconfig**

**You will see network interfaces**

**docker container ls**

**docker exec -it web02 sh**

**#ifconfig**

**You will see network interfaces**

[**https://docs.docker.com/network/network-tutorial-standalone/**](https://docs.docker.com/network/network-tutorial-standalone/)

**Docker Cheat-sheet:**

[**https://design.jboss.org/redhatdeveloper/marketing/docker\_cheatsheet/cheatsheet/images/docker\_cheatsheet\_r3v2.pdf**](https://design.jboss.org/redhatdeveloper/marketing/docker_cheatsheet/cheatsheet/images/docker_cheatsheet_r3v2.pdf)

**Docker UI Portal:**

**$ docker volume create portainer\_data**

**$ docker run -d -p 8000:8000 -p 8085:9000 -v /var/run/docker.sock:/var/run/docker.sock -v portainer\_data:/data portainer/portainer**

**Docker File:**

**Example 1**

**$ nano dockerfile**

**# copy this code into dockerfile**

**FROM alpine:3.4**

**RUN apk update**

**RUN apk add vim**

**RUN apk add curl**

**#run the command to generate an image**

**$ docker build -t myimage:tag .**

**$ docker run -it myimage:tag /bin/sh**

**#verify curl is installed in the newly created image.**

**$ curl**

**Example 2:**

**#install git in ubuntu remote system**

**$ sudo apt install git-all**

**$ git --version**

**# Download project from git**

**$ git clone https://github.com/dockersamples/node-bulletin-board  
$ cd node-bulletin-board/bulletin-board-app**

**# verify the docker file for image creation steps**

**$ nano dockerfile**

# Use the official image as a parent image.

**FROM node:current-slim**

# Set the working directory in image.

**WORKDIR /usr/src/app**

# Copy the file from your host to your current location of image filesystem.

**COPY package.json .**

# Run the command inside your image filesystem.

**RUN npm install**

# Inform Docker that the container is listening on the specified port at runtime.

**EXPOSE 8080**

# Run the specified command within the container.

**CMD [ "npm", "start" ]**

# Copy the rest of your app's source code from your host to your image filesystem.

**COPY . .**

**$ docker build -t myBulletinBoard:1.0 .**

**$ docker images**

**# verify the steps taken to create the image**

**$ docker history <image id>**

**$ docker run --publish 8000:8080 --detach --name bb bulletinboard:1.0**

**# view the webpage as output**

**$ curl localhost:8000**

**Docker Registry**