Docker

In this notes we will be cover below mentioned topics

1- Docker Basic
- What is Docker ?
- Difference Between Virtualization & Containerization
- Docker Engine
- Docker architecture
- Docker Registry

2- Docker Installation
- Docker Server Installation - Centos 7
- Docker editions

3- Basic Usage of Docker
- Docker - Image Search, Pull, Cretae Container, Docker logs etc..

4- Build And Configure Docker Image with dockerfile
- Create Dockerfile
- Bulid Image

5- Upload Docker Image to Docker Hub - Pushing Images/Pulling Images
- Docker Hub
- Pushing Images/Pulling Images

6- Create a Swarm Cluster And Deploy Service Into The Cluster
- Manager / Node
- Adding Nodes to the Cluster
- Managing The Cluster

7- Deploy a service to the swarm

8- Manage Docker container CPU/Memory resources

9- Docker Data Volume
- Mounting a Data volume
- Share data between containers
- Sharing Data between the Host and the Docker Container

10- Manage Docker Networks
- Networking Basic
- Bridge Networking
- Create a new network, and then deploy a container on our new network
- Overlay Networking

11- Docker Backup & Restore
- Backup the image in the docker registry hub
- Backup images to compressed formats such as ‘tar’ files
- Restore Container

Docker Basic

What is Docker ?

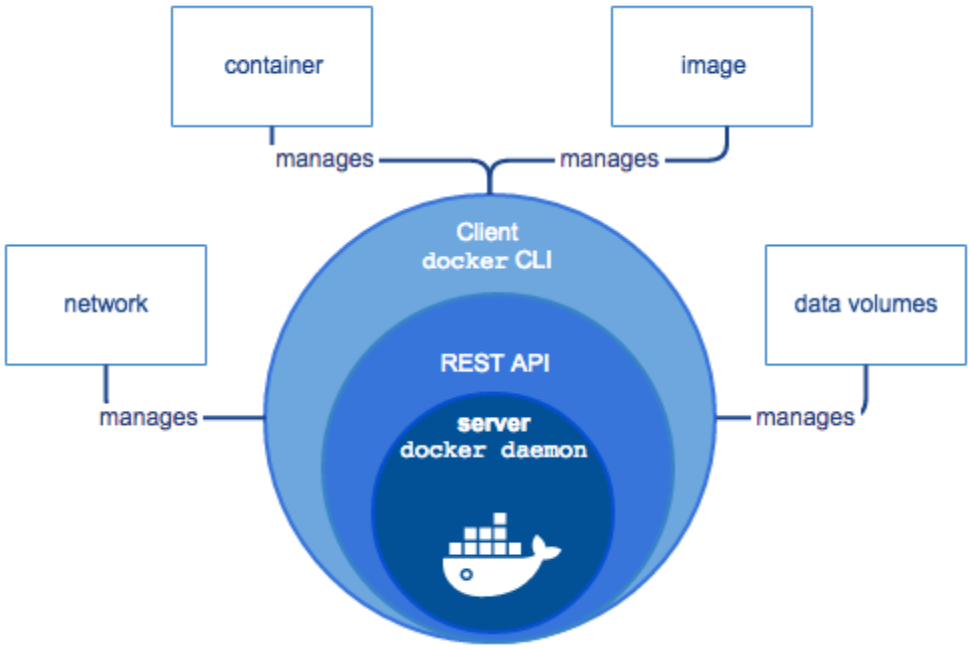
- Docker is open source projects that enable you to provide seem less in environment Production.
- Docker also provides you, ability to run multiple isolated OS on single host.
- Docker enable you to utilize maximum resources from you Hardware.

Difference Between Virtualization & Containerization

Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and hence more secure	Process-level isolation and hence less secure

Docker Engine

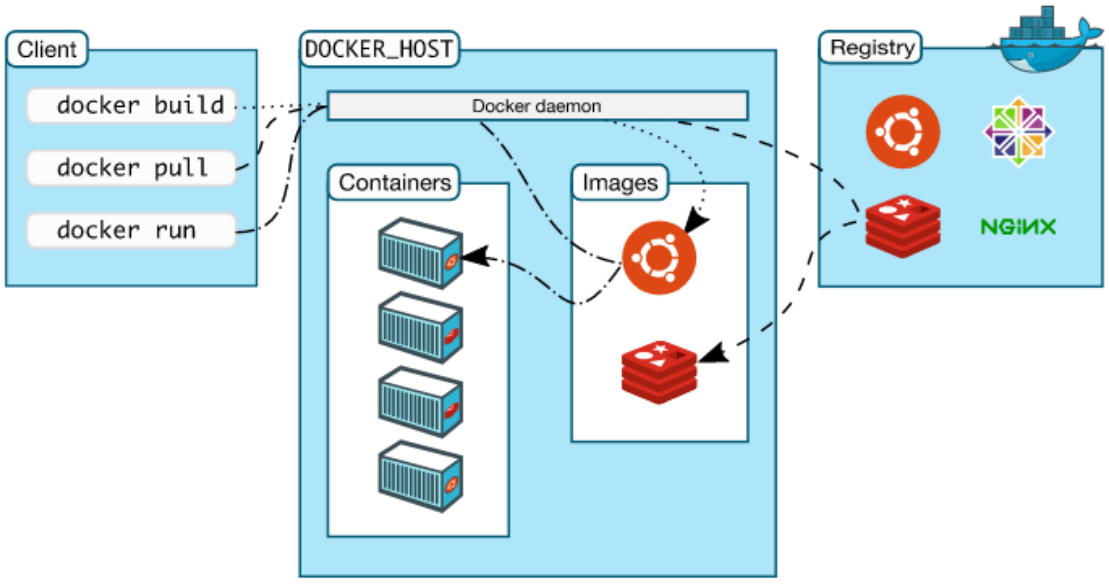
- A server which is a type of long-running program called a daemon process.
- The REST API is used to specify interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface client.



Docker architecture

Its architecture consists mainly three parts.

- **Client**
  Docker provides Command Line Interface (CLI) tools to client to interact with Docker daemon. Client can build, run and stop application. Client can also interact to Docker_Host remotely.
- **Docker_Host**
  It contains Containers, Images, and Docker daemon. It provides complete environment to execute and run your application.
- **Registry**
  It is global repository of images. You can access and use these images to run your application in Docker environment.



Docker Daemon
Daemon run on host machine. Daemons create and manage Docker object: Images, Containers, Networds, Volume, Data, etc. The user does not directly interact with the Daemon, but insted through the docker client.

Docker Client
Primary user interface to Docker. It accepts command from the user and communicate back and forth with a Docker daemon.

Docker Images
Images are used to create Docker containers. Docker provide a simple way to build new images or existing Images. Docker Images are build component of Docker.

Docker Containers
Containers are created from Docker Images. They hold everything that is needed for an application to run. Each container is an isolated and secure application plateform. Docker containers are the run component of Docker.

Docker Registry

- Docker registry is a storage component for docker image
- We can stor the Image in either Public & Private repositories
- Docker Hub is Docker's vwery own cloud repositry

Use of The Docker Registry

- Coltrole where your image are being stored
- Integrate image storage with your in-house developement workflow

Docker Installation

Prerequisites:

- It only works on a 64-bit Linux installation.
- It requires Linux kernel version 3.10 or higher.

Note- I'll be working from a Centos-7.2 server, and I'll be logged in as root.

Step-1 Check the kernel version and the OS architecture.

[root@docker-server ~]# uname -a

Linux docker-server 3.10.0-327.el7.x86_64 #1 SMP Thu Nov 19 22:10:57 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux

You can see that I`m using the kernel version is 3.10.0 with a 64Bit Kernel (x86_64).

[root@docker-server ~]# cat /etc/centos-release
CentOS Linux release 7.2.1511 (Core)

The command shows that the Centos version is 7.2.

Step-2 It is recommended to update Ubuntu before you install new software.

[root@docker-server ~]# yum update

Step-3 Now Install Docker

Note-

Docker is available in two editions:

- **Community Edition (CE)**
  Docker Community Edition (CE) is ideal for developers and small teams looking to get started with Docker and experimenting with container-based apps. that's available for free of cost. Docker CE has two update channels, stable and edge:

  Stable gives you reliable updates every quarter
  Edge gives you new features every month

- **Enterprise Edition (EE).**
  Docker Enterprise Edition (EE) is designed for enterprise development and IT teams who build, ship, and run business critical applications in production at scale. that's not available for free of cost. For more information about Docker EE, including purchasing options.

Click - https://www.docker.com/pricing

Docker package is included in the default CentOS repository. So to install docker , simply run below yum command :

[root@docker-server ~]# yum install docker -y

If you want to install Docker Community Edition (CE), Use the below mention step

1- Install the Docker CE dependencies..

[root@docker-server ~]# yum install yum-lvm2 utils device-mapper-persistent-data -y

2- Installing Docker CE (Install Docker CE Repositery and install docker)

[root@docker-server ~]# yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo

[root@docker-server ~]# yum install docker-ce -y

3- Verify The Docker Verion

[root@docker-server ~]# docker --version
Docker version 17.12.0-ce, build c97c6d6

Step-4 Start the Docker services

[root@docker-server ~]# systemctl start docker
[root@docker-server ~]# systemctl enable docker

Step-5 Check the status of the Docker

[root@docker-server ~]# 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 Mon 2018-01-15 18:03:49 IST; 1min 36s ago
Docs: https://docs.docker.com
Main PID: 28494 (dockerd)
CGroup: /system.slice/docker.service
├─28494 /usr/bin/dockerd
└─28499 docker-containerd --config /var/run/docker/containerd/containerd.toml
Jan 15 18:03:49 docker-server systemd[1]: Started Docker Application Container Engine.
Jan 15 18:03:49 docker-server dockerd[28494]: time="2018-01-15T18:03:49.912071806+05:30"
level=info msg="API listen on /var/run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.

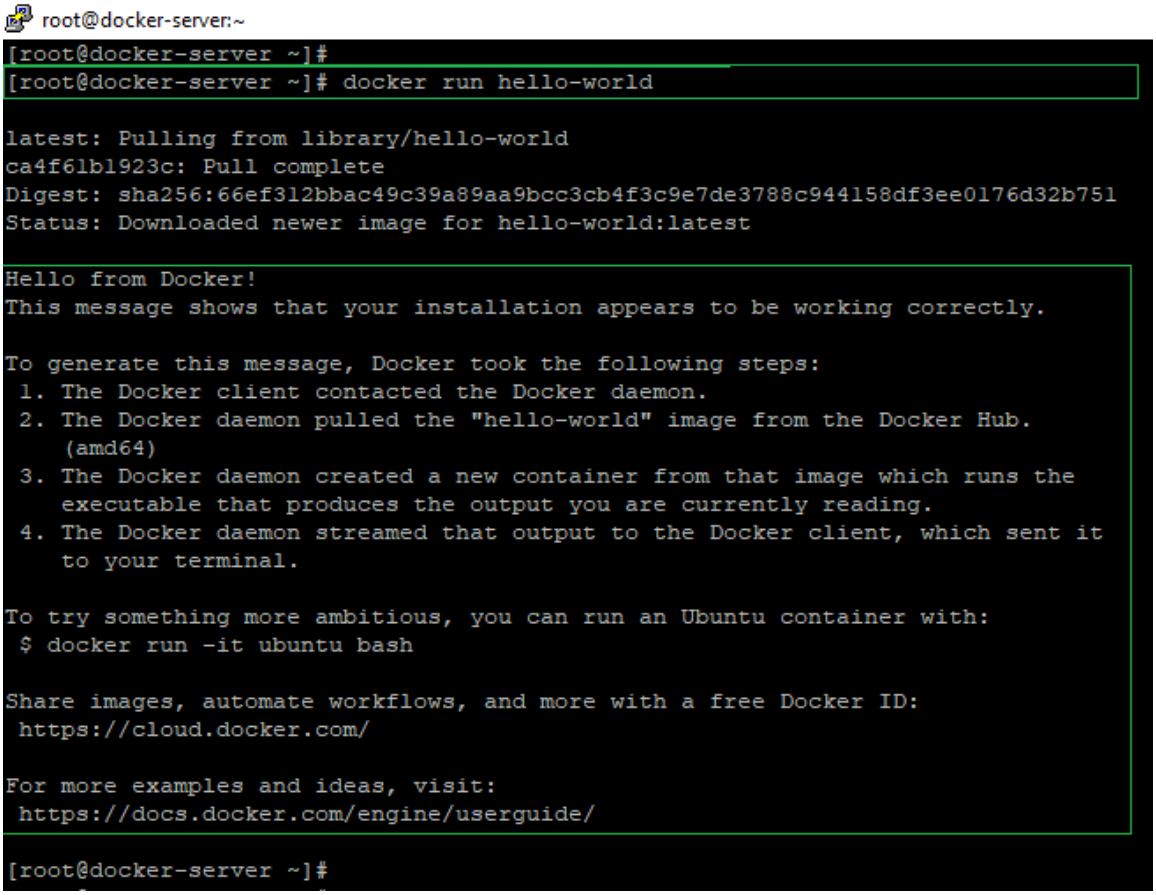
Step-5 Test Docker

[root@docker-server ~]# docker run hello-world

Note- The above command docker run hello-world has three parts.

- **docker-** It is docker engine and used to run docker program. It tells to the operating system that you are running docker program.
- **run-** This subcommand is used to create and run a docker container.
- **hello-world-** It is a name of an image. You need to specify the name of an image which is to load into the container.

When successfully run above command then, this will return the welcome message:



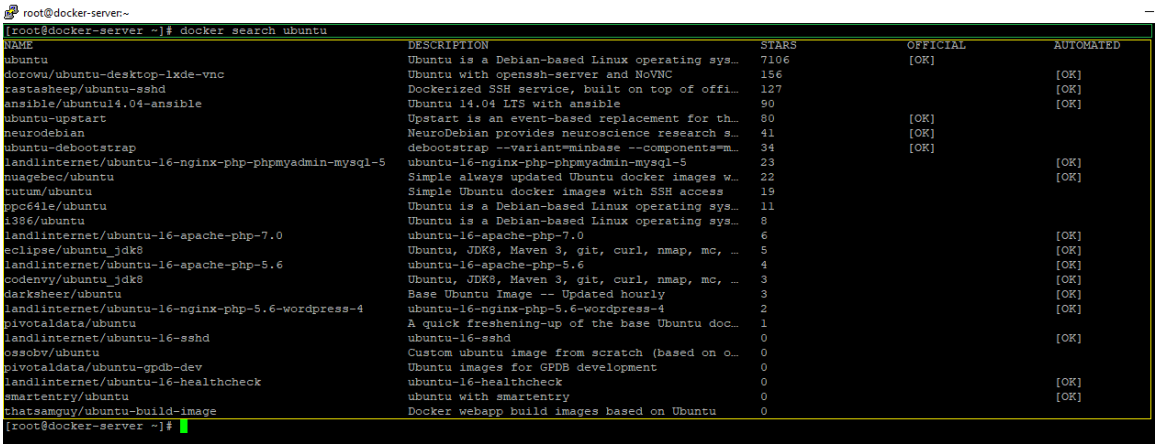
Now docker is installed in your system. You can start making a container by downloading a Docker Image from the Docker Registry.

Basic Usage of Docker

In This Section, I will show how to download a docker image, build a container and how to access the container.

1- To create a new container, choosing a base image with the OS- ubuntu or centos or another. Search for a base image with the docker search command.

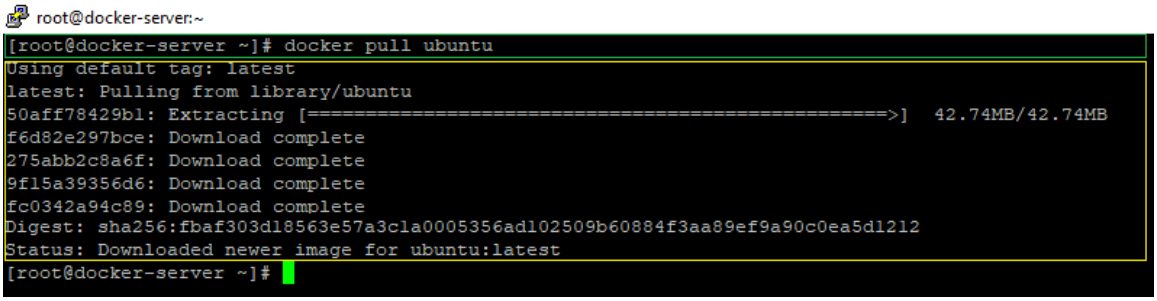
[root@docker-server ~]# docker search ubuntu



This command will show you all ubuntu images.

2- Now Download the base image to our server

[root@docker-server ~]# docker pull ubuntu



This command downloads an image to your server from docker registry/DockerHub.

3- Check the Downloaded images.

[root@docker-server ~]# docker images

root@docker-server:~

```
[root@docker-server ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              00fd29ccc6f1        4 weeks ago         111MB
hello-world         latest              f2a91732366c        7 weeks ago         1.85kB
[root@docker-server ~]#
```

4- Remove Docker Images

[root@docker-server ~]# docker rmi <REPOSITRY Name /IMAGE ID >

5- Launch New Container with Image.

[root@docker-server ~]# docker run -i -t ubuntu:16.04 /bin/bash

```
[root@docker-server ~]# docker run -i -t --name=Server-Linux01  ubuntu:16.04 /bin/bash
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu
Digest: sha256:fbaf303d18563e57a3c1a0005356ad102509b60884f3aa89ef9a90c0ea5d1212
Status: Downloaded newer image for ubuntu:16.04

root@3cc56c7445eb:/#
root@3cc56c7445eb:/# exit
exit
[root@docker-server ~]#
```

The above command is divided as follows:

-i is used to start an interactive session.
-t allocates a tty and attaches stdin and stdout.
ubuntu:16.04 is the image that we used to create the container.
bash (or /bin/bash) is the command that we are running inside the Ubuntu container.

Note- The container will stop when you leave it with the command exit. If you like to have a container that is running in the background, you just need to add the **-d** option in the command
or
To exit from docker container type CTRL + P + Q. This will leave container running in background an provide you host system console.

[root@docker-server ~]# docker run -i -t --name=Server-Linux02 -d ubuntu:16.04 /bin/bash

```
[root@docker-server ~]# docker run -i -t --name=Server-Linux02 -d  ubuntu:16.04 /bin/bash
5c54131e98833bea385ad1974d4292724cc6bdf676ccbf0f848bc65d9d8abefc
[root@docker-server ~]#
```

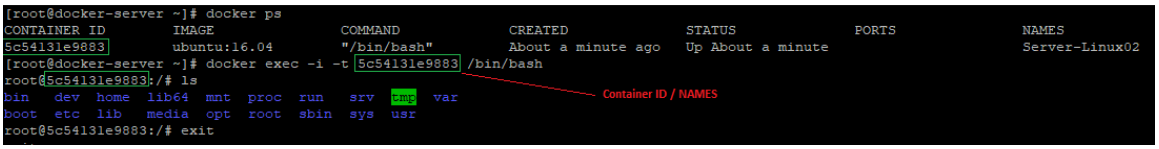
6- Now you can see the container running in the background by using command

[root@docker-server ~]# docker ps

```
[root@docker-server ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
5c54131e9883        ubuntu:16.04        "/bin/bash"         About a minute ago   Up About a minute                       Server-Linux02
[root@docker-server ~]#
```

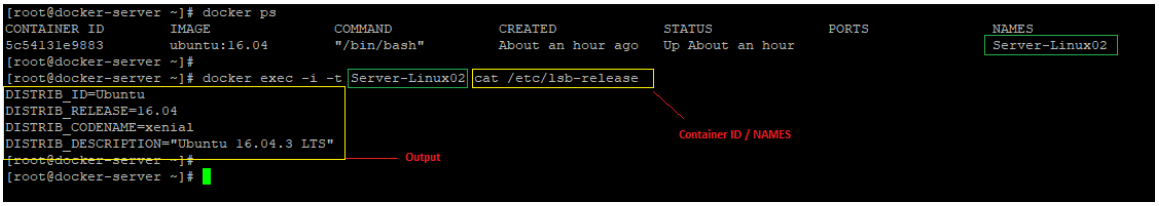
7- Access the shell of container that runs in the background mode

[root@docker-server ~]# docker exec -i -t 5c54131e9883 /bin/bash



Or Run Command Directly Without Access bash shell

[root@docker-server ~]# docker exec -i -t 5c54131e9883 cat /etc/lsb-release



Other e.q.

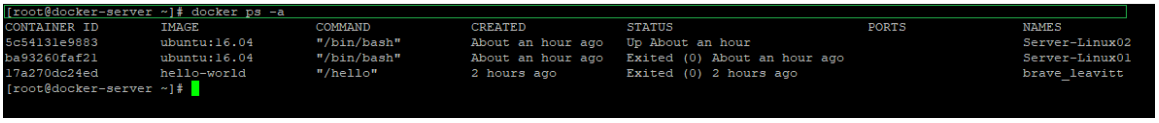
Update system-

[root@docker-server ~]# docker exec e18de3b27825 apt-get update

Install Apache Package-
[root@docker-server ~]# docker exec e18de3b27825 apt-get install apache2 -y

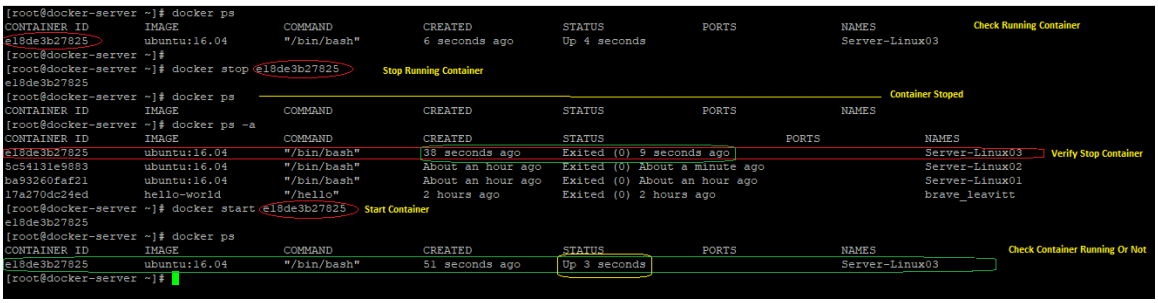
8- To list all containers (including stopped container) use following command.

[root@docker-server ~]# docker ps -a



9- Start/Stop Container

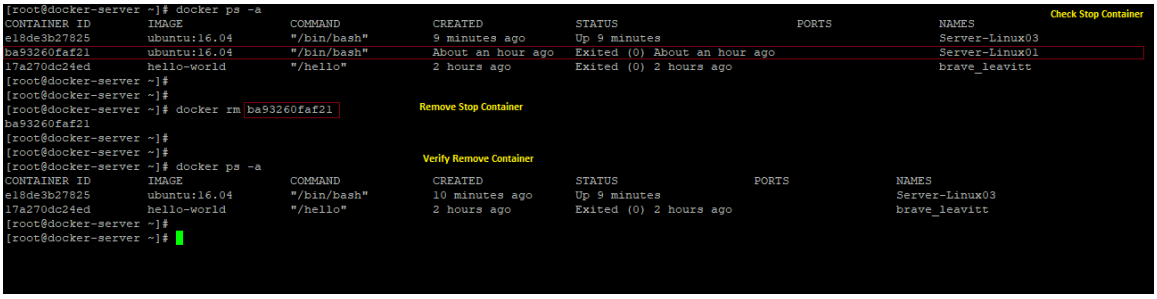
docker stop <CONTAINER ID>
docker start <CONTAINER ID>



10- Remove The Container

If you like to remove the container, stop it first and then remove it with the command.

[root@docker-server ~]# docker rm <CONTAINER ID>

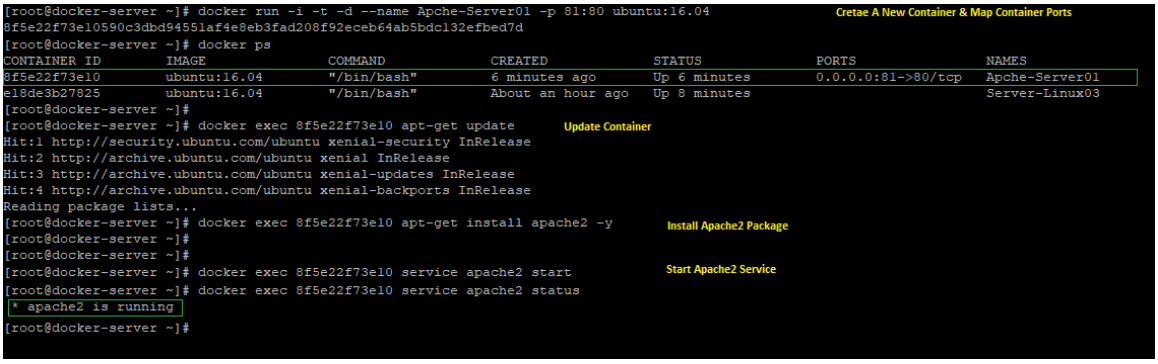


11- Run Apache inside Docker container and access apache Server

[root@docker-server ~]# docker run -i -t -d --name Apche-Server01 -p 81:80 ubuntu:16.04

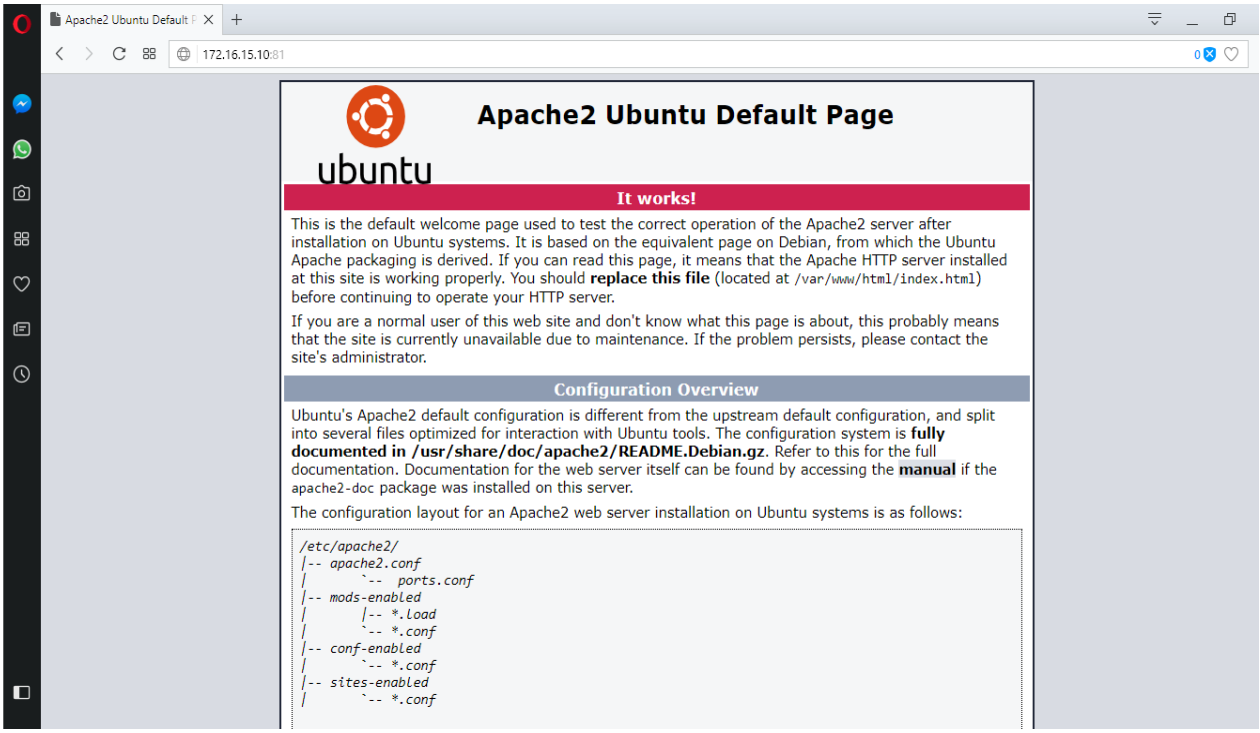
(**-p** option exposes the host port to container port.)

[root@docker-server ~]# docker ps
[root@docker-server ~]# docker exec 8f5e22f73e10 apt-get update
[root@docker-server ~]# docker exec 8f5e22f73e10 apt-get install apache2 -y
[root@docker-server ~]# docker exec 8f5e22f73e10 service apache2 start
[root@docker-server ~]# docker exec 8f5e22f73e10 service apache2 status



In order to visit the page served by the Apache2 container, open a browser from a remote location in your LAN and type the IP address of your machine using the HTTP protocol.

http://docker-server-ip:81



12- View Logs for a Docker Container

[root@docker-server ~]# docker logs <Containe ID>

13- Rename Docker Container

[root@docker-server ~]# docker rename <Old_Name> <New_Name>

Build And Configure Docker Image with dockerfile

- A Dockerfile is a text configuration file written in a popular, human-readable Markup Language called YAML.
- It is a step-by-step script of all the commands you need to run to assemble a Docker Image.
- The docker build command processes this file generating a Docker Image in your Local Image Cache, which you can then start-up using the docker run command, or push to a permanent Image Repository.

Below are some dockerfile commands you must know:

1- Create Dockerfile

The following Dockerfile sets up an SSHd service in a container that you can use to connect to and inspect other container's volumes, or to get quick access to a test container.

[root@docker-server ~]# vim dockerfile

```
# The line below states we will base our new image on the Latest Official Ubuntu
FROM ubuntu:16.04

# Identify the maintainer of an image
MAINTAINER Ashutosh Maurya <ashutoshsmaurya@gmail.com>

# Update the image to the latest packages and install SSH Package
RUN apt-get update && apt-get install -y openssh-server

# Create a Dir
RUN mkdir /var/run/sshd

# Set Password
RUN echo 'root:Ashu@324' | chpasswd

# Enable Root Login
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i
/etc/pam.d/sshd

#Define an variable.
ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

# Expose port 80
```

EXPOSE 22

Start SSH Service
RUN service ssh start

#Last is the actual command to start up SSHD within our Container
CMD ["/usr/sbin/sshd", "-D"]

:wq (Save & Quit)

```
# The line below states we will base our new image on the Latest Official Ubuntu
FROM ubuntu:16.04

# Identify the maintainer of an image
MAINTAINER Ashutosh Maurya <ashutoshsmaurya@gmail.com>

# Update the image to the latest packages and install SSH Package
RUN apt-get update && apt-get install -y openssh-server

# Create a Dir
RUN mkdir /var/run/sshd

# Set Password
RUN echo 'root:Ashu@324' | chpasswd

# Enable Root Login
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd

#Define an variable.
ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

# Expose port 80
EXPOSE 22

# Start SSH Service
RUN service ssh start

#Last is the actual command to start up SSHD within our Container
CMD ["/usr/sbin/sshd", "-D"]
```

Now build the image using:

[root@docker-server ~]# docker build -t ssh-server .

```
[root@docker-server ~]# docker build -t ssh-server .
Sending build context to Docker daemon  559.8MB
Step 1/12 : FROM ubuntu:16.04
 ---> 00fd29ccc6f1
Step 2/12 : MAINTAINER Ashutosh Maurya <ashutoshsmaurya@gmail.com>
 ---> Running in b93e0c03aebf
Removing intermediate container b93e0c03aebf
 ---> 98b475c47a39
Step 3/12 : RUN apt-get update && apt-get install -y openssh-server
 ---> Running in f28cb8d7fd9a
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [56.8 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:6 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [539 kB]
Get:8 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:9 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial-updates/universe Sources [234 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [903 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [13.1 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [735 kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 Packages [18.5 kB]
Get:17 http://archive.ubuntu.com/ubuntu xenial-backports/main amd64 Packages [5162 B]
Get:18 http://archive.ubuntu.com/ubuntu xenial-backports/universe amd64 Packages [7146 B]
Get:19 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.7 kB]
```

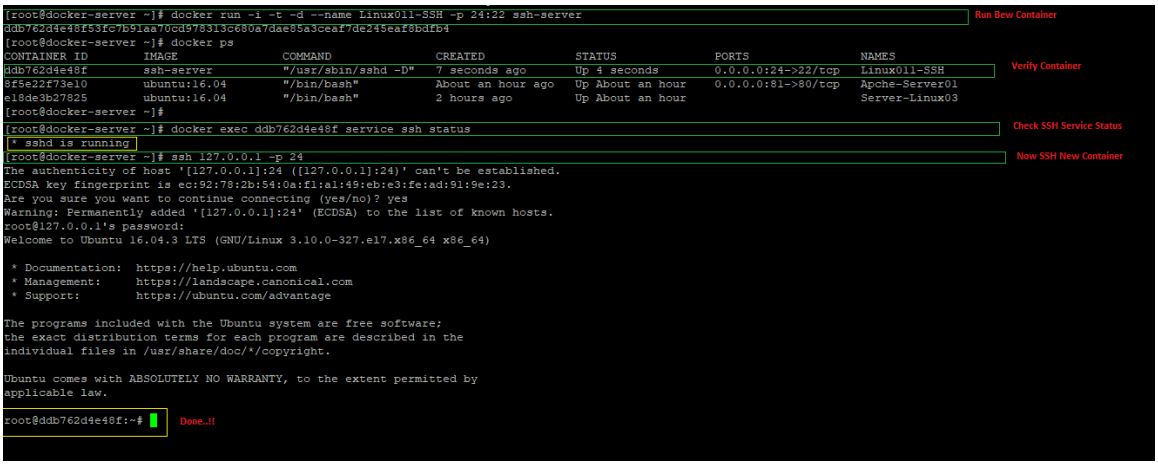
2- When the command completed successfully, we can check the new image 'ssh-server' with the docker command below

[root@docker-server ~]# docker images

```
[root@docker-server ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ssh-server          latest              bf040e61afaf        About a minute ago  205MB
ubuntu              16.04               00fd29ccc6f1        4 weeks ago         111MB
ubuntu              latest              00fd29ccc6f1        4 weeks ago         111MB
hello-world         latest              f2a91732366c        7 weeks ago         1.85kB
[root@docker-server ~]#
```

3- Now run the new container with command below and check ssh

[root@docker-server ~]# docker run -i -t -d --name Linux011-SSH -p 24:22 ssh-server
[root@docker-server ~]# docker ps
[root@docker-server ~]# docker exec ddb762d4e48f service ssh status
[root@docker-server ~]# ssh 127.0.0.1 -p 24 ***(- Enter Password Wich Define in Dockerfile)***



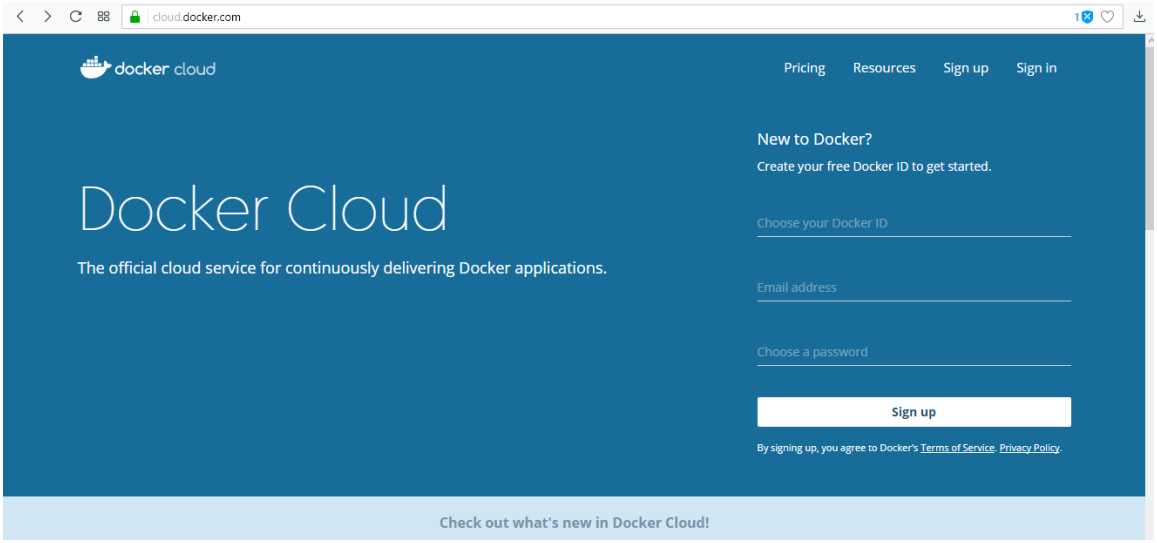
Upload Docker Image to Docker Hub - Pushing Images/Pulling Images

Above we've successfully built an image and created a container with it, let's move on to learn about Docker hub and see how to use it to share images.

Docker hub is a public registry maintained by Docker (the company). It has over 15,000 images that can be downloaded and used to build containers. Docker hub also provides authentication, workflow tools like web looks and builds triggers, and privacy tools like private repositories for storing images you don't want to share publicly.

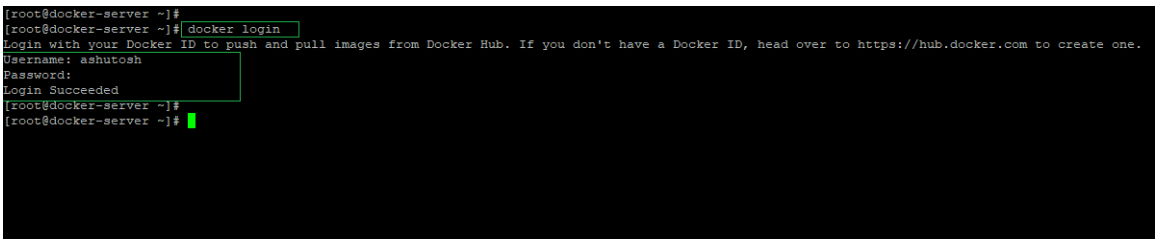
Pulling Images:-

1- Create an Account - https://hub.docker.com



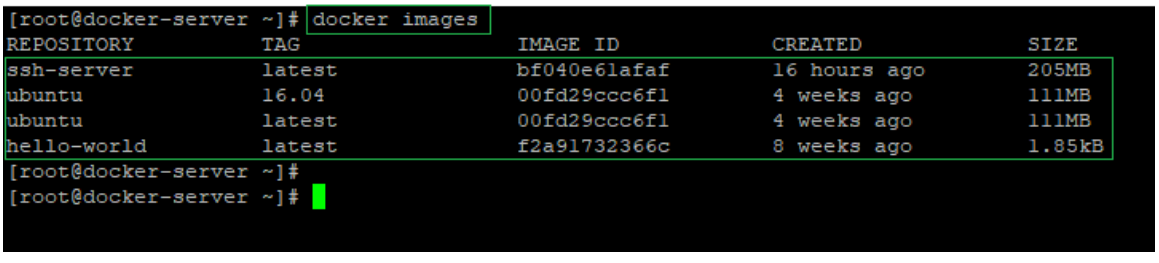
2- Log into the Docker public registry on your local machine.

[root@docker-server ~]# docker login



3- Check the image ID using

[root@docker-server ~]# docker images



4- Tag the image: It is more like naming the version of the image. It's optional but it is recommended as it helps in maintaining the version (same like ubuntu:16.04 and ubuntu:17.04)

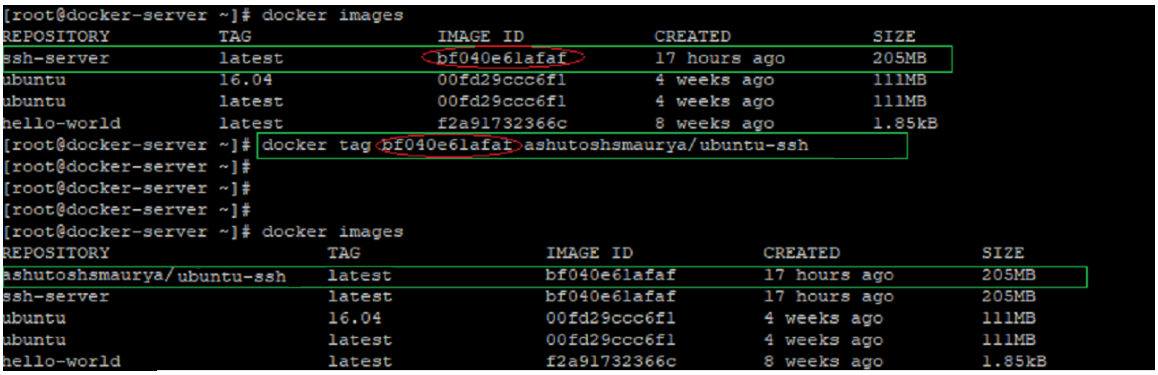
[root@docker-server ~]# docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ssh-server	latest	bf040e61afaf	17 hours ago	205MB
ubuntu	16.04	00fd29ccc6f1	4 weeks ago	111MB

ubuntu	latest	00fd29ccc6f1	4 weeks ago	111MB
hello-world	latest	f2a91732366c	8 weeks ago	1.85kB

[root@docker-server ~]# docker tag bf040e61afaf ashutoshsmaurya/ubuntu-ssh

The parameters of a docker tag command include both names and tags. Does this mean it is possible to assign a new name? It does indeed:

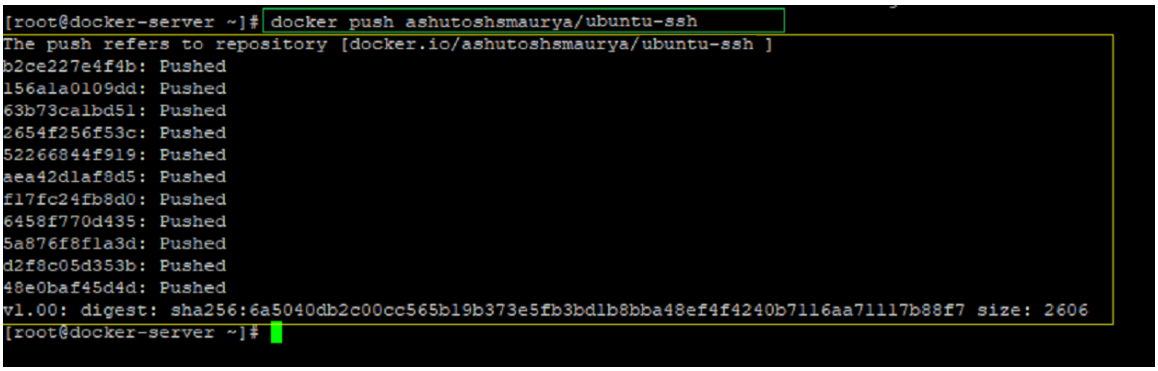


Note- Tags could also be explicitly specified as command-line parameters at build-time:

docker build -t demo/ubuntu-ssh .

5- Upload your tagged image to the repository

[root@docker-server ~]# docker push ashutoshsmaurya/ubuntu-ssh



Note- A few things to keep in mind:

[root@docker-server ~]# docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ssh-server	latest	bf040e61afaf	17 hours ago	205MB
ubuntu	16.04	00fd29ccc6f1	4 weeks ago	111MB
ubuntu	latest	00fd29ccc6f1	4 weeks ago	111MB

hello-world	latest	f2a91732366c	8 weeks ago	1.85kB

- Docker CLI does not recognize that the original image 'ubuntu-ssh' is supposed to end up in the remote registry. After rebuilds, all images should be tagged with an appropriate prefix before being pushed.

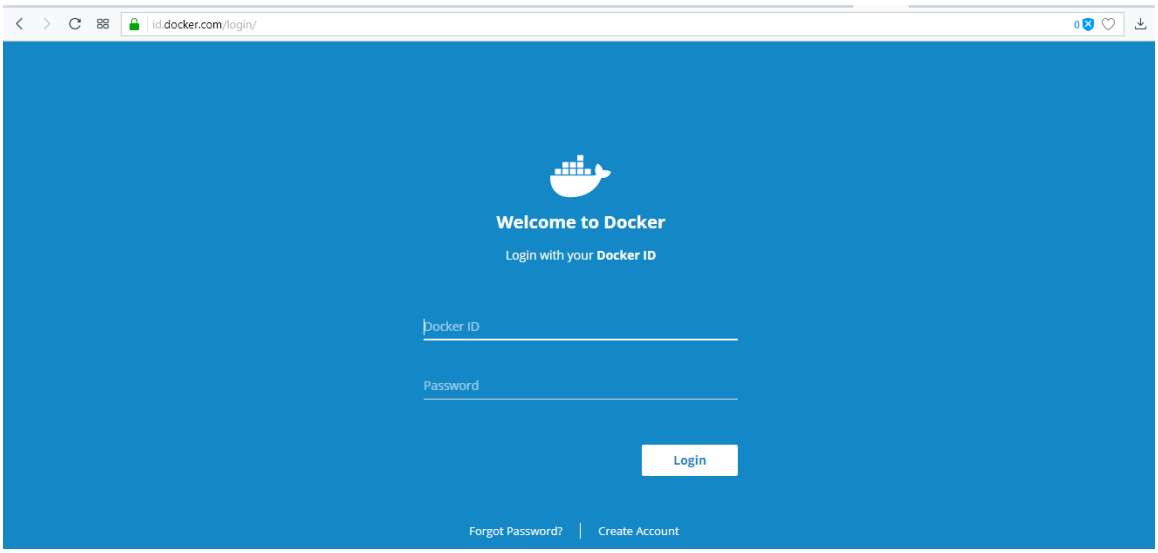
- A push does not happen automatically on rebuilds; docker push should always be executed explicitly.

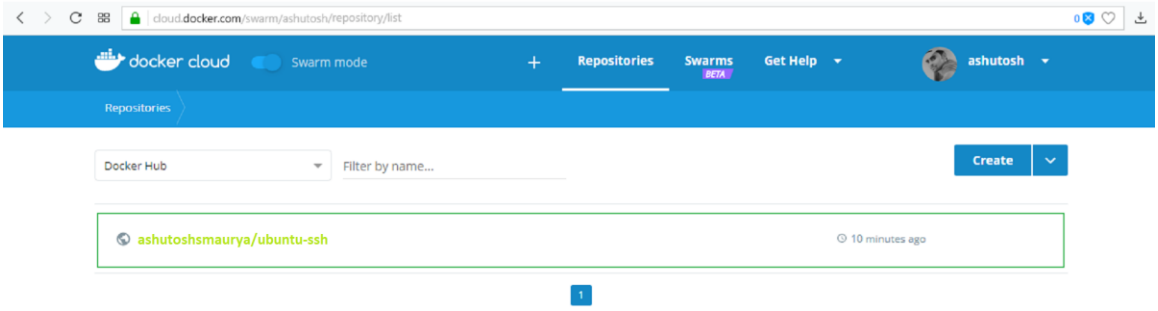
- If the push command argument has no tag component (e.g. docker push username/ssh-server), all images and tags associated with the name "username/ubuntu-ssh" will be pushed.

- If the push command argument specifies a particular tag (e.g. docker push username/ubuntu-ssh), only the specified image and tag will be pushed.

- Removing an image from the remote repository is not trivial.

6- Log into Docker Hub, you will see the new image there, with its pull command. (https://hub.docker.com)





Pulling Image:-

You can how transfer your pushed image to another host that's running a Docker server by logging in to Docker and running a container from the shared image "ashutoshsmaurya/ubuntu-ssh":

1- Login to your other docker server - My Other Docker Server IP - 172.16.10.60

[root@docker-server ~]# ssh ashu@172.16.10.60 --> SSH Other Docker Server
ashu@my-docker:~$ sudo su - --> Switch to Root User

root@my-docker:~# docker images --> Check Images

```
[root@docker-server ~]#
[root@docker-server ~]# ssh ashu@172.16.10.60
ashu@172.16.10.60's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/                    SSH Other Docker Server

  System information as of Sat Jan 13 13:22:28 EST 2018

  System load:  0.07              Processes:           79
  Usage of /:   2.6% of 56.96GB   Users logged in:     1
  Memory usage: 6%                IP address for eth0:    172.16.10.60
  Swap usage:   0%                IP address for docker0: 172.17.42.1

  Graph this data and manage this system at:
    https://landscape.canonical.com/

184 packages can be updated.
117 updates are security updates.

Last login: Sat Jan 13 13:22:31 2018 from 172.16.15.10
ashu@my-docker:~$
ashu@my-docker:~$ sudo su -                                  Login to root user
[sudo] password for ashu:
root@my-docker:~#                                            Check Docker IMAGE
root@my-docker:~#
root@my-docker:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
root@my-docker:~#
```

2- Login to your docker username and password

root@my-docker:~# docker login

```
root@my-docker:~#
root@my-docker:~# docker login
Username: ashutoshsmaurya
Password:

Login Succeeded
root@my-docker:~#
```

3- Now Pull The Share IMAGE - "ashutoshsmaurya/ubuntu-ssh"

root@my-docker:~# docker pull ashutoshsmaurya/ubuntu-ssh

```
root@my-docker:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
root@my-docker:~# docker pull ashutoshsmaurya/ubuntu-ssh
latest: Pulling from ashutoshsmaurya/ubuntu-ssh

d4bcfe3fb921: Pull complete
c4cc29c71ec1: Pull complete
43ea2a37afe7: Pull complete
6d26059fef3f: Pull complete
7d016c45d574: Pull complete
71d4d62c116f: Pull complete
70b4363675a7: Pull complete
17b77e4b459d: Pull complete
25eeb4f46913: Pull complete
4838c791a07a: Pull complete
dfc47eb2ad3c: Pull complete
fc5ca4f92b41: Pull complete
16bf380b2252: Pull complete
91a77228a772: Pull complete
361a84f69d2f: Pull complete
956ca47c3045: Pull complete
aadc881df77f: Pull complete
Digest: sha256:52365836ea5bee93209af566b302b82719d167f75fc247ffc64da0656fd5bd9d
Status: Downloaded newer image for ashutoshsmaurya/ubuntu-ssh:latest
```

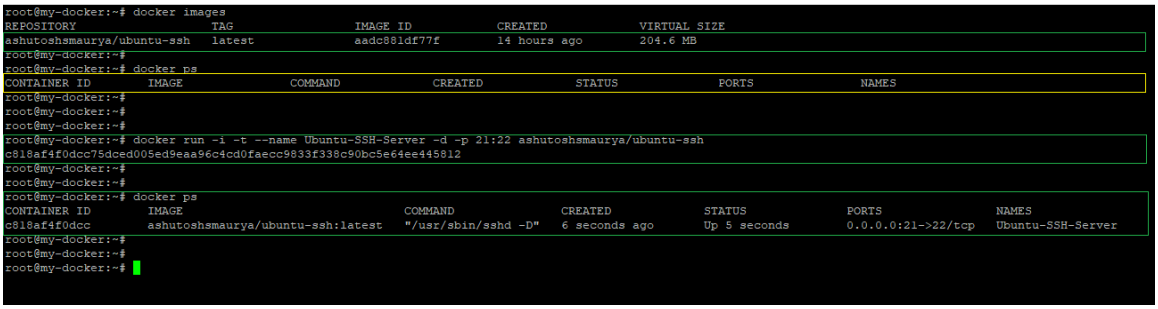
4- Verify The IMAGE Pull Or Not and Run this Image "ashutoshsmaurya/ubuntu-ssh"

root@my-docker:~# docker images

```
root@my-docker:~# docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ashutoshsmaurya/ubuntu-ssh  latest              aadc881df77f        14 hours ago        204.6 MB
root@my-docker:~#
root@my-docker:~#
```

root@my-docker:~# docker run -i -t --name Ubuntu-SSH-Server -d -p 21:22 ashutoshsmaurya/ubuntu-ssh

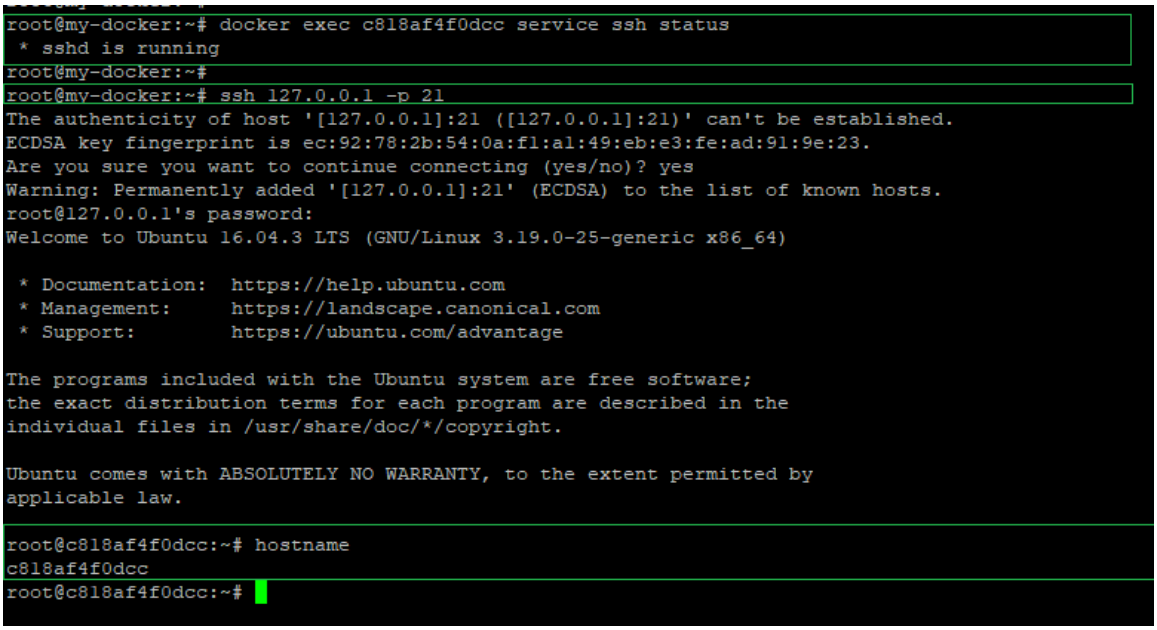
root@my-docker:~# docker ps



5- Check SSH

root@my-docker:~# docker ps
root@my-docker:~# docker exec c818af4f0dcc service ssh status

root@my-docker:~# ssh 127.0.0.1 -p 21



Create a Swarm Cluster And Deploy Service Into The Cluster

Docker Swarm is a clustering and scheduling tool for Docker containers. With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.

When Docker released its latest version, Docker Engine v1.12, it included quite a few changes to the capabilities provided by Docker Swarm. In today's article, we'll be exploring how to deploy a service using Docker's Swarm Mode.

Here are some common terms associated with Docker Swarm:

- Node: A node is an instance of a Swarm. Nodes can be distributed on-premises or in public clouds.
- Swarm: a cluster of nodes (or Docker Engines). In Swarm mode, you orchestrate services, instead of running container commands.
- Manager Nodes: These nodes receive service definitions from the user, and dispatch work to worker nodes. Manager nodes can also perform the duties of worker nodes.
- Worker Nodes: These nodes collect and run tasks from manager nodes.
- Service: A service specifies the container image and the number of replicas. Here is an example of a service command which will be scheduled on 2 available nodes:

  # docker service create --replicas 2 --name mynginx nginx
- Task: A task is an atomic unit of a Service scheduled on a worker node. In the example above, two tasks would be scheduled by a master node on two worker nodes (assuming they are not scheduled on the Master itself). The two tasks will run independently of each other.

Note- Swarm 'network-agnostic' (overlay networks to be configured separately) - Read - Manage Docker Networks

Prerequisites:

- Minimum two nodes with Docker installed (swarm manager and cluster nodes)
- All the nodes should be able to talk to each other using public or private IP addresses.
- Install Docker Engine on each server (Manager, Node1 & Node2)

In this setup, I have total 2 nodes. 1 Swarm manager node and two other nodes to join the cluster with the following private IP addresses.

	IP Add	**Hostname**

Manager	-	172.16.15.10	docker-server
Node1	-	172.16.10.60	node1
Node2	-	172.16.11.12	node2

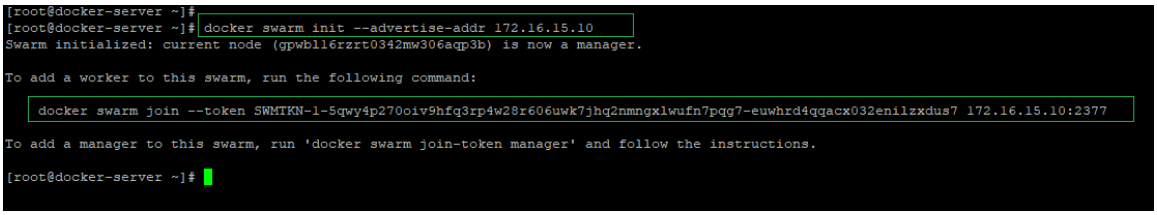
Manager Node

1. Run the following command with the manager nodes IP for initializing the swarm cluster.

[root@docker-server ~]# docker swarm init --advertise-addr **172.16.15.10**

*** 172.16.15.10 - Manager Server IP Address**

You will get the following output once swarm is initialized.



You can see, the output has the steps to join other nodes to this swarm manager node.

2. To know the swarm cluster info

[root@docker-server ~]# docker info

```
[root@docker-server ~]# docker info
Containers: 5
 Running: 3
 Paused: 0
 Stopped: 2
Images: 14
Server Version: 17.12.0-ce
Storage Driver: devicemapper
Swarm: active
 NodeID: gpwbll6rzrt0342mw306aqp3b
 Is Manager: true
 ClusterID: ksqly3w9pa0cydd2dnl79lzwd
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 172.16.15.10
 Manager Addresses:
  172.16.15.10:2377
Runtimes: runc
```

3. To know the information about all the nodes in the cluster

[root@docker-server ~]# docker node ls

```
[root@docker-server ~]# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
gpwbll6rzrt0342mw306aqp3b *   docker-server       Ready               Active              Leader
[root@docker-server ~]#
[root@docker-server ~]#
```

Now swarm manager ready , we can add our other nodes to the manager to form a multi node cluster.

4- You can get the swarm token with run this command

[root@docker-server ~]# docker swarm join-token worker

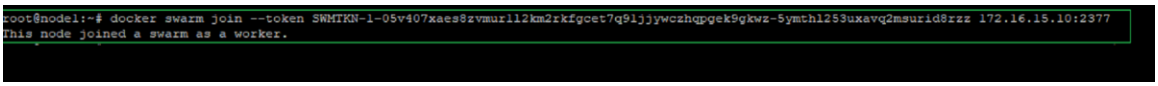
```
[root@docker-server ~]#
[root@docker-server ~]# docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5qwy4p270oiv9hfq3rp4w28r606uwk7jhq2nmngxlwufn7pqg7-euwhrd4qqacx032enilzxdus7 172.16.15.10:2377

[root@docker-server ~]#
```

Now Execute the swarm join command from the manager output on all the nodes..

Adding Nodes to the Cluster

Node1-

root@node1:~# docker swarm join --token SWMTKN-1-05v407xaes8zvmur1l2km2rkfgcet7q91jjywczhqpgek9gkwz-5ymthl253uxavq2msurid8rzz 172.16.15.10:2377



After the command has been executed successfully, you'll see this response:

This node joined a swarm as a worker.

If Any Error Something Like-
Error response from daemon: error while validating Root CA Certificate: x509: certificate has expired or is not yet valid.

Solution:
1- Check the Date And Time - Manager & Node Server
2- On Manager

-> cd /etc/pki/tls/certs/
-> cp ca-bundle.crt{,.orig}
-> wget http://curl.haxx.se/ca/cacert.pem -O ca-bundle.crt --no-check-certificate
-> service docker stop ; sleep 3 ; killall docker ; service docker restart
-> docker swarm init --advertise-addr <Manager_Server_IP>

Now Execute the swarm join command from the manager output on all the nodes..

Log out of node-1, and then repeat this process with node-2 to add it to your cluster.

Node-2

root@node2:~# docker swarm join --token SWMTKN-1-05v407xaes8zvmur1l2km2rkfgcet7q91jjywczhqpgek9gkwz-5ymthl253uxavq2msurid8rzz 172.16.15.10:2377

```
root@node2:~#
root@node2:~# docker swarm join --token SWMTKN-1-05v407xaes8zvmurl12km2rkfgcet7q91jjywczhqpgek9gkwz-5ymthl253uxavq2msurid8rzz 172.16.15.10:2377
This node joined a swarm as a worker.
root@node2:~#
```

Now added two worker nodes to the cluster.

5- Managing The Cluster

Once you joined all the extra nodes, you can list the swarm node information by executing the following command on the manager node.

[root@docker-server ~]# docker node ls

You will get the output will the swarm cluster info as shown below.

```
[root@docker-server ~]# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
tv6q43iazwrgf33zo2hitg9yj *   docker-server       Ready               Active              Leader
zlivg6lrsz8lz9iyw8nevmebl     node1               Ready               Active
ytty6wtz3c5z7833dlddt3anp     node2               Ready               Active
[root@docker-server ~]#
[root@docker-server ~]#
```

This output shows that we're dealing with a 3-node Docker Swarm and its nodes — a manager and two workers.

Note-
The AVAILABILITY column shows whether or not the scheduler can assign tasks to the node:

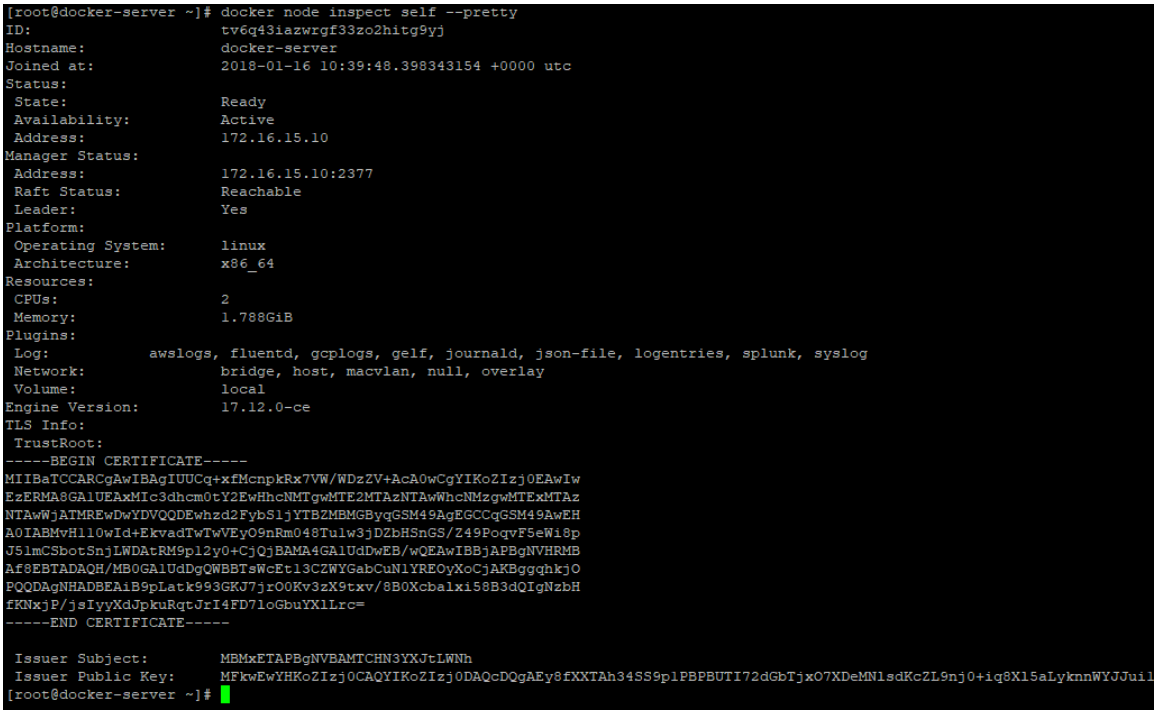
- **Active** means that the scheduler can assign tasks to the node.
- **Pause** means the scheduler doesn't assign new tasks to the node, but existing tasks remain running.
- **Drain** means the scheduler doesn't assign new tasks to the node. The scheduler shuts down any existing tasks and schedules them on an available node.

The MANAGER STATUS column shows node participation in the Raft consensus:

- No value indicates a worker node that does not participate in swarm management.
- **Leader** means the node is the primary manager node that makes all swarm management and orchestration decisions for the swarm.
- **Reachable** means the node is a manager node participating in the Raft consensus quorum. If the leader node becomes unavailable, the node is eligible for election as the new leader.
- **Unavailable** means the node is a manager that is not able to communicate with other managers. If a manager node becomes unavailable, you should either join a new manager node to the swarm or promote a worker node to be a manager.

6- View the details for an individual node. The output defaults to JSON format, but you can pass the --pretty flag to print the results in human-readable format.

[root@docker-server ~]# docker node inspect self --pretty



-> Single Node

[root@docker-server ~]# docker node ls
[root@docker-server ~]# docker node inspect zlivg6lrsz81z9iyw8nevmebl --pretty

```
[root@docker-server ~]# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
tv6q43iazwrgf33zo2hitg9yj *   docker-server       Ready               Active              Leader
zlivg6lrsz8lz9iyw8nevmebl     node1               Ready               Active
ytty6wtz3c5z7833dlddt3anp     node2               Ready               Active
[root@docker-server ~]# docker node inspect zlivg6lrsz8lz9iyw8nevmebl --pretty
ID:                     zlivg6lrsz8lz9iyw8nevmebl
Hostname:               node1
Joined at:              2018-01-16 10:40:07.686838161 +0000 utc
Status:
 State:                 Ready
 Availability:          Active
 Address:               172.16.10.60
Platform:
 Operating System:      linux
 Architecture:          x86_64
Resources:
 CPUs:                  1
 Memory:                1.954GiB
Plugins:
 Log:           awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk, syslog
 Network:               bridge, host, macvlan, null, overlay
 Volume:                local
Engine Version:         18.01.0-ce
TLS Info:
 TrustRoot:
-----BEGIN CERTIFICATE-----
MIIBaTCCARCgAwIBAgIUUCq+xfMcnpkRx7VW/WDzZV+AcA0wCgYIKoZIzj0EAwIw
EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTgwMTE2MTAzNTAwWhcNMzgwMTExMTAz
NTAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABMvHll0wId+EkvadIwTwVEyO9nRm048Tulw3jDZbHSnGS/Z49PoqvF5eWi8p
J5lmCSbotSnjLWDAtRM9pl2y0+CjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBTsWcEt13CZWYGabCuNlYREOyXoCjAKBggqhkjO
PQQDAgNHADBEAiB9pLatk993GKJ7jrO0Kv3zX9txv/8B0XcbalxI58B3dQIgNzbH
fKNxjP/jsIyyXdJpkuRqtJrI4FD7loGbuYXlLrc=
-----END CERTIFICATE-----

 Issuer Subject:        MBMxETAPBgNVBAMTCHM3YXJtLWNh
```

7- View the other management commands that you can run on the manager node, type:

root@node2:~# docker node --help

```
root@node2:~# docker node --help

Usage:  docker node COMMAND

Manage Swarm nodes

Options:


Commands:
  demote      Demote one or more nodes from manager in the swarm
  inspect     Display detailed information on one or more nodes
  ls          List nodes in the swarm
  promote     Promote one or more nodes to manager in the swarm
  ps          List tasks running on one or more nodes, defaults to current node
  rm          Remove one or more nodes from the swarm
  update      Update a node

Run 'docker node COMMAND --help' for more information on a command.
root@node2:~#
```

Deploy a service to the swarm

Now that we have our swarm up and running, Now deploy a service to the swarm..

1- Open a terminal and ssh into the machine where you run your manager node.

[root@docker-server ~]#

2- Deploy a web server service using the official Nginx container image:

[root@docker-server ~]# docker service create --replicas 2 --name webserver -p 80:80 nginx

- **docker service create** command creates the service.
- **--name** flag names the service webserver.
- **--replicas** flag specifies the desired state of 2 running instance.
- **-p 80:80** In this command, we're mapping port 80 in the Nginx container to port 80 on the cluster so that we can access the default Nginx page from anywhere.

```
[root@docker-server ~]# docker service create --replicas 2 --name webserver -p 80:80 nginx
ujizlwwgz2454lp7meoa4dlue
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
[root@docker-server ~]#
```

3- Display the details about a service - webserver

[root@docker-server ~]# docker service inspect --pretty webserver

- **Inspect** Service Display detailed information on one or more services

```
[root@docker-server ~]# docker service inspect --pretty webserver

ID:             ujizlwwgz2454lp7meoa4dlue
Name:           webserver
Service Mode:   Replicated
 Replicas:      2
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         nginx:latest@sha256:285b49d42c703fdf257d1e2422765c4ba9d3e37768d6ea83d7fe2043dad6e63d
Resources:
Endpoint Mode:  vip
Ports:
 PublishedPort = 80
  Protocol = tcp
  TargetPort = 80
  PublishMode = ingress
```

4- See the list of running services

[root@docker-server ~]# docker service ls

```
[root@docker-server ~]# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
ujizlwwgz245        webserver           replicated          2/2                 nginx:latest        *:80->80/tcp


[root@docker-server ~]#
[root@docker-server ~]#
```

5- Check which nodes the 'webserver' services is running

[root@docker-server ~]# docker service ps webserver

```
[root@docker-server ~]# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
ujizlwwgz245        webserver           replicated          2/2                 nginx:latest        *:80->80/tcp
[root@docker-server ~]#
[root@docker-server ~]#
[root@docker-server ~]# docker service ps webserver
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
22nvastnwol9        webserver.1         nginx:latest        node2               Running             Running 9 minutes ago
asd65fgbueij        webserver.2         nginx:latest        node1               Running             Running 9 minutes ago
[root@docker-server ~]#
```

6- Scale the webserver service that we started earlier to Six instances.

Swarm is the ability to **scale** a service, that is, spin up additional instances of a service.

[root@docker-server ~]# docker service scale webserver=6

```
[root@docker-server ~]# docker service scale webserver=6
1/6: running   [==================================================>]
2/6: running   [==================================================>]
3/6: running   [==================================================>]
4/6: running   [==================================================>]
5/6: running   [==================================================>]
6/6: running   [==================================================>]
verify: Service converged
[root@docker-server ~]#
```

7- Run docker service ps <SERVICE-ID> to see the updated task list

[root@docker-server ~]# docker service ps webserver

```
[root@docker-server ~]# docker service ps webserver
ID             NAME          IMAGE          NODE           DESIRED STATE   CURRENT STATE            ERROR   PORTS
22nvastnwol9   webserver.1   nginx:latest   node2          Running         Running about an hour ago
asd65fgbueij   webserver.2   nginx:latest   node1          Running         Running about an hour ago
z1jfxz2oypdl   webserver.3   nginx:latest   docker-server  Running         Running 3 minutes ago
sa7rcb4i6ofz   webserver.4   nginx:latest   docker-server  Running         Running 3 minutes ago
ssflia5541bw   webserver.5   nginx:latest   node2          Running         Running 5 minutes ago
bo399e2osq1c   webserver.6   nginx:latest   node1          Running         Running 5 minutes ago
[root@docker-server ~]#
```

You can see that swarm has created 4 new tasks to scale to a total of 6 running instances of nginx latest. The tasks are distributed between the four nodes of the swarm. Two is running on manager (docker-server).

8- See the containers running on the node where you're connected. (Running on manager-docker-server)

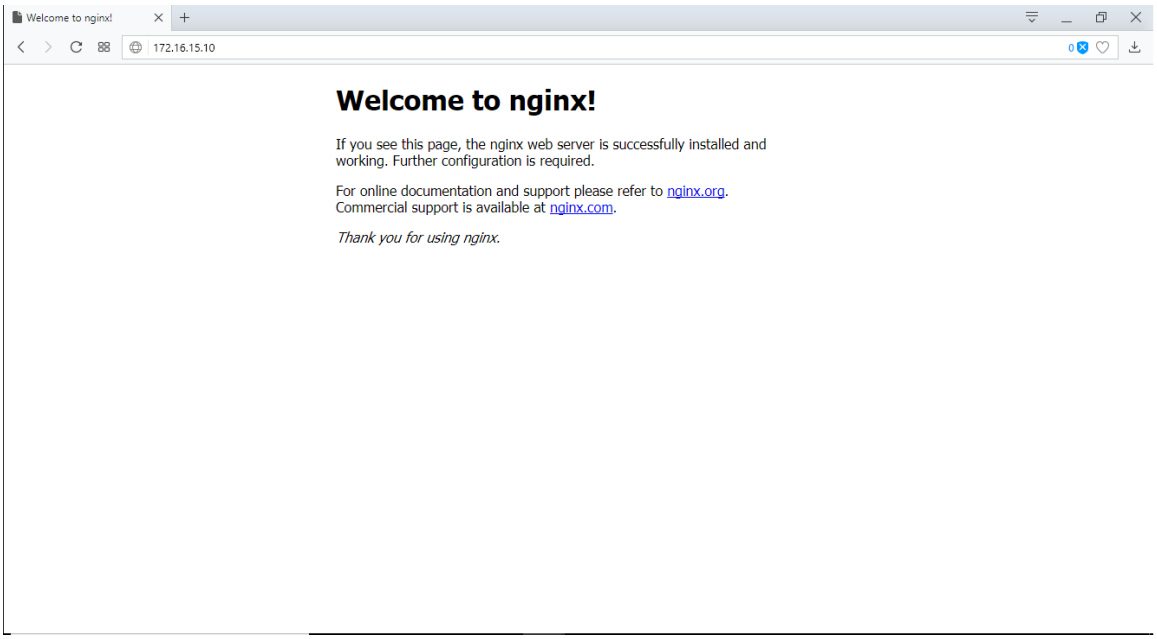
[root@docker-server ~]# docker ps

```
[root@docker-server ~]#
[root@docker-server ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
2182d8dd2ee5   nginx:latest   "nginx -g 'daemon of…"   13 minutes ago   Up 12 minutes   80/tcp    webserver.4.sa7rcb4i6ofzi0s8wg6769b6w
05ec61a2ad22   nginx:latest   "nginx -g 'daemon of…"   13 minutes ago   Up 12 minutes   80/tcp    webserver.3.z1jfxz2oypdlb08ozl9cwk22h
[root@docker-server ~]#
```

9- Accessing the Service

You can access the service by hitting any of the manager or worker nodes. It does not matter if the particular node does not have a container scheduled on it. That is the whole idea of the swarm.

Try out a curl to any of the Docker Machine IPs (manager or Node1/2/) or hit the URL (**Error! Hyperlink reference not valid.**) in the browser. You should be able to get the standard NGINX Home page.



10- To view the available all Docker Swarm commands

[root@docker-server ~]# docker swarm --help

```
[root@docker-server ~]# docker swarm --help

Usage:  docker swarm COMMAND

Manage Swarm

Options:


Commands:
  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm

Run 'docker swarm COMMAND --help' for more information on a command.
[root@docker-server ~]#
[root@docker-server ~]#
```

Cleaning Up

Let's clean up the service we created, the containers we started, and finally disable Swarm mode.

1- Remove The service

docker service rm <Serivce_Name>

2- Kill or Stop The Running Container

docker kill yourcontainerid1 yourcontainerid2
docker stop yourcontainerid1 yourcontainerid2

3- Remove node from the Swarm

docker swarm leave --force
docker swarm leave --force

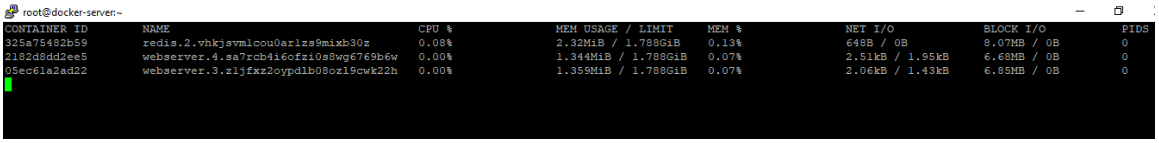
Manage docker container CPU/Memory resources

By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler will allow. Docker provides ways to control how much memory, CPU, or block IO a container can use, setting runtime configuration flags of the docker run command.

This guide covers CPU and memory resources limits that you can place on your Docker containers..

1- Display a live stream of container(s) resource usage statistics

[root@docker-server ~]# docker stats



The stats command above gives data about CPU usage, memory usage, network usage and I/O usage.

2- memory

Things are much simpler when it comes to memory. Memory can be limited with a short command (-m flag), and the limits are applied to both memory and swap.

[root@docker-server ~]# docker run -ti -m 300M --memory-swap 300M -d --name Ubuntu-Linux ubuntu:16.04

[root@docker-server ~]# docker ps

[root@docker-server ~]# docker stats

```
[root@docker-server ~]# docker run -ti -m 300M --memory-swap 300M -d --name Ubuntu-Linux ubuntu:16.04
6f3c1c4aeb174ef56fdea79a131216lf92937cb9a2719bc38145d9ccd6e8cad9
[root@docker-server ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
6f3c1c4aeb17        ubuntu:16.04        "/bin/bash"              32 seconds ago      Up 29 seconds                           Ubuntu-Linux
c16ef0b345a8        nginx:latest        "nginx -g 'daemon of…"   13 minutes ago      Up 13 minutes       80/tcp              webserver.2.ql31tlai65cqe2awd8a02nozf
28cb60950036        redis:3.0.6         "/entrypoint.sh redi…"   13 minutes ago      Up 13 minutes       6379/tcp            redis.1.5fw2qc4crjinh1zh5o9sx1a09
65c8f93aadd7        redis:3.0.6         "/entrypoint.sh redi…"   13 minutes ago      Up 13 minutes       6379/tcp            redis.3.v7p0vuycomw53uma8exyis91r
648fde19a6a8        nginx:latest        "nginx -g 'daemon of…"   13 minutes ago      Up 13 minutes       80/tcp              webserver.5.n800lm4kkj5v5twe5k18qvxek
8b359187330e        nginx:latest        "nginx -g 'daemon of…"   13 minutes ago      Up 13 minutes       80/tcp              webserver.1.oi5iwl8ascql7corfej56nh73
8338fd60de92        nginx:latest        "nginx -g 'daemon of…"   13 minutes ago      Up 13 minutes       80/tcp              webserver.6.kt38119mljnbcb6exoq0ylukf
8afce6bfbbed        ubuntu              "/bin/bash"              About an hour ago   Up About an hour                        Ubuntu_Server
825a75482b59        redis:3.0.6         "/entrypoint.sh redi…"   2 hours ago         Up 2 hours          6379/tcp            redis.2.vhkjsvmlcou0arlzs9mixb30z
2182d8dd2ee5        nginx:latest        "nginx -g 'daemon of…"   5 hours ago         Up 5 hours          80/tcp              webserver.4.sa7rcb4i6ofzi0s8wg6769b6w
05ec61a2ad22        nginx:latest        "nginx -g 'daemon of…"   5 hours ago         Up 5 hours          80/tcp              webserver.3.zljfxz2oypdlb08ozl9cwk22h
[root@docker-server ~]# docker stats
CONTAINER ID        NAME                                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
6f3c1c4aeb17        Ubuntu-Linux                        0.00%               1.816MiB / 300MiB     0.61%               648B / 0B           3.68MB / 0B         0
c16ef0b345a8        webserver.2.ql31tlai65cqe2awd8a02nozf   0.00%           1.332MiB / 1.788GiB   0.07%               648B / 0B           6.67MB / 0B         0
28cb60950036        redis.1.5fw2qc4crjinh1zh5o9sx1a09   0.08%               956KiB / 1.788GiB     0.05%               648B / 0B           7.01MB / 0B         0
```

This example command will limit the container memory and swap space usage to 300MB each.

Currently controlling the amount of allocated memory and swap separately is not possible in Docker. By default, when a container is launched there are no set memory limits, which can lead to issues where a single container can hog up all the memory and make the system unstable.

3- CPU

Each container is assigned a “share” of the CPU. By default, this is set to 1024. By itself 1024 CPU share does not mean anything. When only a single container is running, it will use all the available CPU resources. However, if you launch another container and they both have 1024 CPU share defined, then each container will claim at least 50% of CPU resources.

CPU share is set using the -c or --cpu-shares flag when launching the container.

For example:

[root@docker-server ~]# docker run -ti -c 1024 ubuntu:16.04 /bin/bash

Another option to setting CPU limits is CPU Completely Fair Scheduler (CFS). In this case we are setting CPU Period (100ms by default) and CPU Quota (number of cpu ticks allocated to container).

For example:

[root@docker-server ~]# docker run -ti --cpu-period=50000 --cpu-quota=10000 ubuntu:16.04 /bin/bash

4- Disk

Disk space and read/write speeds can be limited in Docker. By default, read/write speed are unlimited; however if required, they can be limited as need be using cgroups.

Each container is allocated 10GB of space by default. This value can be too much or too little depending on the application or micro-service. The amount of allocated disk space can be altered when first launching the container.

More Info-
https://docs.docker.com/engine/admin/resource_constraints/#limit-a-containers-access-to-memory

Docker Data Volume

Data Volume

A "data volume" is a marked directory inside of a container that exists to hold persistent or commonly shared data. Assigning these volumes is done when creating a new container.

Some points to keep in mind about Data Volumes.

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created. By default, it is not deleted when the container is stopped. It is not even garbage collected when there is no container referencing the volume.
- The data volumes are independently updated. Data volumes can be shared across containers too. They could be mounted in read-only mode too.

A- Mounting a Data volume

1- First mounting a data volume in one of our containers.

root@node2:~# docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-ssh-backup	latest	8bf6f0392a04	18 hours ago	205MB
ubuntu	latest	2a4cca5ac898	2 days ago	111MB

root@node2:~# docker run -it -d -v /Data --name container1 -p 27:22 ubuntu-ssh-backup

This will launch a container (named container1) , and you will be at the prompt in the container.

```
root@node2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu-ssh-backup   latest              8bf6f0392a04        18 hours ago        205MB
ubuntu              latest              2a4cca5ac898        2 days ago          111MB
root@node2:~#
root@node2:~#
root@node2:~# docker run -it -d -v /Data --name container1 -p 27:22 ubuntu-ssh-backup
bf4f2294e98d2cb99ea296b1e86588d6e46c7d63131594ld7db3a671f58acab6
root@node2:~#
root@node2:~#
root@node2:~#
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                   NAMES
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   4 seconds ago       Up 3 seconds        0.0.0.0:27->22/tcp      container1
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   40 minutes ago      Up 40 minutes       0.0.0.0:26->22/tcp      Ubuntu-Server-SSH
root@node2:~#
```

2- Access The Container - container1

root@node2:~# docker exec -it bf4f2294e98d /bin/bash
root@bf4f2294e98d:/# ls

Data boot etc lib media opt root sbin sys usr
bin dev home lib64 mnt proc run srv tmp var

root@bf4f2294e98d:/#

```
root@node2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu-ssh-backup   latest              8bf6f0392a04        18 hours ago        205MB
ubuntu              latest              2a4cca5ac898        2 days ago          111MB
root@node2:~# docker exec -it bf4f2294e98d /bin/bash
root@bf4f2294e98d:/#
root@bf4f2294e98d:/#
root@bf4f2294e98d:/# ls
Data  boot  etc   lib    media  opt   root  sbin  sys  usr
bin   dev   home  lib64  mnt    proc  run   srv   tmp  var
root@bf4f2294e98d:/#
```

Notice that a volume named Data is visible now.

3- Create a file named Test.txt in Data volume and Exit Container

root@bf4f2294e98d:/# cd Data/
root@bf4f2294e98d:/Data# pwd
/Data

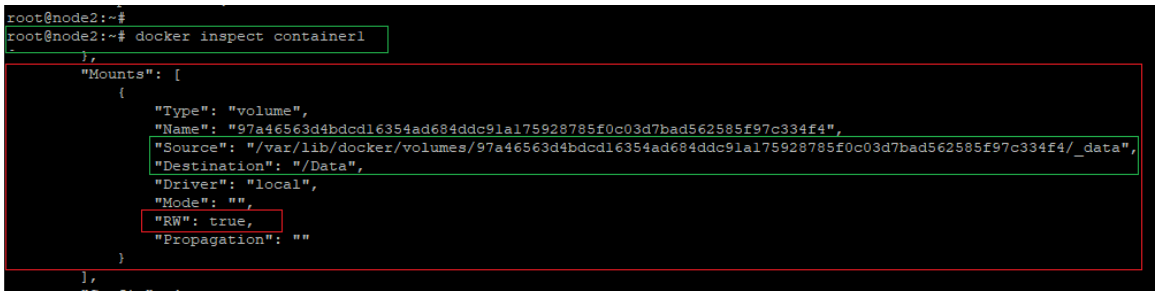
root@bf4f2294e98d:/Data# touch Test.txt
root@bf4f2294e98d:/Data# ls
Test.txt

```
root@bf4f2294e98d:/# cd Data/
root@bf4f2294e98d:/Data# pwd
/Data
root@bf4f2294e98d:/Data#
root@bf4f2294e98d:/Data# touch Test.txt
root@bf4f2294e98d:/Data#
root@bf4f2294e98d:/Data#
root@bf4f2294e98d:/Data# ls
Test.txt
root@bf4f2294e98d:/Data# exit
root@node2:~#
```

4- Now inspect the container..

root@node2:~# docker inspect container1

you should look for Mounts attribute in the output. A sample output from my machine is shown below:



When you mounted a volume **(/Data),** it has created a folder **/var/lib/docker/volume**…. for you, which is where it puts all the files, etc that you would have created in that volume. Note that we had created a **Test.txt** over there (we will come to that in a while).

* Also notice that the RW mode is set to true i.e. Read and Write.

5- Now Restart The Container - (container1), And see if our volume is still available and that file1.txt exists...

Stop container1-
root@node2:~# docker stop container1

Check container1 Stop Or Not

root@node2:~# docker ps

d18d4614e43e ubuntu-ssh-backup "/usr/sbin/sshd -D" About an hour ago Up About an hour

0.0.0.0:26->22/tcp Ubuntu-Server-SSH

Now restart the container1

root@node2:~# docker restart container1

Again check container1 running or not

root@node2:~# docker ps

bf4f2294e98d ubuntu-ssh-backup "/usr/sbin/sshd -D" 39 minutes ago Up 4 seconds
0.0.0.0:27->22/tcp container1
d18d4614e43e ubuntu-ssh-backup "/usr/sbin/sshd -D" About an hour ago Up About an hour
0.0.0.0:26->22/tcp Ubuntu-Server-SSH

root@node2:~# docker exec container1 ls /Data/ -->And our file is still present.
Test.txt

```
root@node2:~#
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      38 minutes ago      Up 38 minutes       0.0.0.0:27->22/tcp       container1
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      About an hour ago   Up About an hour    0.0.0.0:26->22/tcp       Ubuntu-Server-SSH
root@node2:~#
root@node2:~#
root@node2:~# docker stop container1
container1
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      About an hour ago   Up About an hour    0.0.0.0:26->22/tcp       Ubuntu-Server-SSH
root@node2:~#
root@node2:~#
root@node2:~# docker restart container1
container1
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      39 minutes ago      Up 4 seconds        0.0.0.0:27->22/tcp       container1
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      About an hour ago   Up About an hour    0.0.0.0:26->22/tcp       Ubuntu-Server-SSH
root@node2:~#
root@node2:~# docker exec container1 ls /Data/
Test.txt
root@node2:~#
```

Note-

1- If u want to remove the container1. the data volume is still present on the host. This is a ghost volume and could remain there on your machine consuming space.

2- If u want to remove container1 with Data Volume , then use a **-v** option while removing the container1.

root@node2:~# docker rm --help

```
root@node2:~# docker rm --help

Usage:  docker rm [OPTIONS] CONTAINER [CONTAINER...]

Remove one or more containers

Options:
  -f, --force     Force the removal of a running container (uses SIGKILL)
  -l, --link      Remove the specified link
  -v, --volumes   Remove the volumes associated with the container
```

B- Share data between containers

Above We Are Create a container (**container1**) and mount a volume inside - **/Data** dir and create a **Test.txt** file in **/Data** volume ...

1- If we execute a command on the running container1 i.e. see the contents of our /Data volume, you can see that the one files (Test.txt)are present.

root@node2:~# docker exec container1 ls /Data
Test.txt

2- Now launch another container (container2) but it will mount the data volume from container1..

root@node2:~# docker run -it -d --volumes-from container1 --name container2 -p 28:22 ubuntu-ssh-backup

root@node2:~# docker ps

```
root@node2:~# docker run -it -d --volumes-from container1 --name container2 -p 28:22  ubuntu-ssh-backup
30d4ea23dd7d1514dc321129b374331a744cb3c5fb75cf05d5d0d8fc9f29c17e
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS              PORTS                NAMES
30d4ea23dd7d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   17 seconds ago      Up 16 seconds       0.0.0.0:28->22/tcp   container2
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   2 hours ago         Up About an hour    0.0.0.0:27->22/tcp   container1
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   3 hours ago         Up 3 hours          0.0.0.0:26->22/tcp   Ubuntu-Server-SSH
root@node2:~# exec
```

Note- The **--volumes-from** flag is then used to mount the **/Data** volume inside of other containers.

3- Access the Conatainer - conatiner2 , we can see that the Data folder is present and if we do a ls inside of that, we can see our two files: Test.txt

root@node2:~# docker ps

```
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS              PORTS                NAMES
30d4ea23dd7d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   39 minutes ago      Up 39 minutes       0.0.0.0:28->22/tcp   container2
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   2 hours ago         Up 2 hours          0.0.0.0:27->22/tcp   container1
```

root@node2:~# docker exec -it container2 /bin/bash

root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/# ls

```
Data   boot   etc    lib    media   opt    root   sbin   sys   usr
bin    dev    home   lib64  mnt     proc   run    srv    tmp   var
```

root@30d4ea23dd7d:/# ls Data/
Test.txt

root@30d4ea23dd7d:/# exit

```
root@node2:~#
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS              PORTS                   NAMES
30d4ea23dd7d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   39 minutes ago      Up 39 minutes       0.0.0.0:28->22/tcp      container2
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   2 hours ago         Up 2 hours          0.0.0.0:27->22/tcp      container1
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"   3 hours ago         Up 3 hours          0.0.0.0:26->22/tcp      Ubuntu-Server-SSH
root@node2:~#
root@node2:~# docker exec -it container2 /bin/bash
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/# ls
Data  boot  etc   lib    media  opt   root  sbin  sys  usr
bin   dev   home  lib64  mnt    proc  run   srv   tmp  var
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/# ls Data/
Test.txt
root@30d4ea23dd7d:/#
root@30d4ea23dd7d:/# exit
root@30d4ea23dd7d:/#
```

Note-
You can launch multiple containers too , all using the same data volume from container1.
Foe Exa

root@node2:~# docker run -it -d --volumes-from container1 --name container3 ubuntu
root@node2:~# docker run -it -d --volumes-from container1 --name container4 centos

C- Sharing Data between the Host and the Docker Container

The other common use for Docker containers is as a means of sharing files between the host machine and the Docker container.

1- Let's create a folder to share the container..

root@node2:~# mkdir /Share_Data

2- Run The Container

root@node2:~# docker run -d -v /Share_Data/:/mnt -p 29:22 --name conatiner3 -i ubuntu-ssh-backup

-v /Share_Data:/mnt — We set up a volume that links the **/mnt** directory from inside the container to the **/Share_Data** directory on the host machine.

If you make any changes to the '/Share_Data folder', you'll be able to see them from inside the Docker container '/mnt' folder in real-time as well.

Manage Docker Networks

A- Networking Basic

1- The docker network command is the main command for configuring and managing container networks.

root@node2:~# docker network

```
[root@docker-server ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
b51703b9d4cc        bridge              bridge              local
01bf4bc556e0        docker_gwbridge     bridge              local
8b5679aeb414        host                host                local
cmc57d48mrap        ingress             overlay             swarm
b8ee58fac2b0        none                null                local
[root@docker-server ~]# docker network

Usage:  docker network COMMAND

Manage networks

Options:


Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
```

2- List networks

When you install Docker, it creates three networks automatically. You can list these networks..

root@node2:~# docker network ls

```
root@node2:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
432ff82cc384        bridge              bridge              local
56f1d80fd100        host                host                local
292060745d10        none                null                local
root@node2:~#
```

*Docker connects to the bridge network by default; this allows deployed containers to be seen on your network.

You can see that each network gets a unique ID and NAME. Each network is also associated with a single driver. Notice that the “bridge” network and the “host” network have the same name as their respective drivers.

3- Inspect a network

Check network configuration details. These details include; name, ID, driver, IPAM driver, subnet info, connected containers, and more.

root@node2:~# docker network inspect bridge

```
root@node2:~# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "432ff82cc38448cb23bf1a11baf447cf524f5c35d8d66c1ad17595b84ba59362",
        "Created": "2018-01-16T05:57:41.202878065-05:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.42.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
```

Note- The syntax of the docker network inspect command is docker network inspect <network>, where <network> can be either network name or network ID. In the example above we are showing the configuration details for the network called “bridge”. Do not confuse this with the “bridge” driver.

4- List network driver plugins

[root@node2 ~]# docker info

```
[root@node2 ~]# docker info

Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: active
 NodeID: tv6q43iazwrgf33zo2hitg9yj
 Is Manager: true
 ClusterID: btq87ryi5fr7k6tu0jwazln8b
 Managers: 1
 Nodes: 3
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
```

B- Bridge Networking

1- Every clean installation of Docker comes with a pre-built network called bridge.

[root@node2 ~]# docker network ls

```
root@node2:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
432ff82cc384        bridge              bridge              local
56f1d80fd100        host                host                local
292060745d10        none                null                local
root@node2:~#
```

- The output above shows that the bridge network is associated with the bridge driver.
- In this example the network and the driver have the same name - but they are not the same thing!
- And also shows that the bridge network is scoped locally. This means that the network only exists on this Docker host. This is true of all networks using the bridge driver - the bridge driver provides single-host networking.

2- Now check list the bridges on your Docker host

[root@node2 ~]# brctl show

```
[root@node2 ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242f58d4c6f       no
```

The output above shows a single Linux bridge called docker0. This is the bridge that was automatically created for the bridge network. You can see that it has no interfaces currently connected to it.

You can also use the ip a command to view details of the docker0 bridge.

[root@node2 ~]# ip a

```
[root@node2 ~]# ip a
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:f5:8d:4c:6f brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:f5ff:fe8d:4c6f/64 scope link
       valid_lft forever preferred_lft forever
```

3- Connect a container

The bridge network is the default network for new containers. This means that unless you specify a different network, all new containers will be connected to the bridge network.

Now cretae a new container..

root@node1:~# docker run -itd --name webserver nginx

root@node1:~# docker ps -->**verify container**

```
root@node1:~#
root@node1:~# docker run -itd --name webserver nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
e7bb522d92ff: Already exists
6edc05228666: Already exists
cd866a17e81f: Already exists
Digest: sha256:285b49d42c703fdf257d1e2422765c4ba9d3e37768d6ea83d7fe2043dad6e63d
Status: Downloaded newer image for nginx:latest
853a0fac769367de5b103a58793b9ef68aefa1035bc21e083123e395b5db2c6d
root@node1:~#
root@node1:~# docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                  NAMES
853a0fac7693        nginx                          "nginx -g 'daemon of…"   6 seconds ago       Up 5 seconds        80/tcp                 webserver
ee89f0573e96        ashutoshsmaurya/ubuntu-ssh     "/usr/sbin/sshd -D"      About an hour ago   Up About an hour    0.0.0.0:23->22/tcp     Ubuntu-SSH
root@node1:~#
```

As no network was specified on the docker run command, the container will be added to the bridge network.

root@node1:~# brctl show

```
root@node1:~#
root@node1:~# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.56847afe9799       no              vethfefb898
```

Notice how the docker0 bridge now has an interface connected. This interface connects the docker0 bridge to the new container just created.

You can inspect the bridge network again.. to see the new container attached to it.

root@node1:~# docker network inspect bridge

```
root@node1:~# docker network inspect bridge

 "Containers": {
     "853a0fac769367de5b103a58793b9ef68aefa1035bc21e083123e395b5db2c6d": {
         "Name": "webserver",
         "EndpointID": "cdfe895a66d7a425c14a39f840d775b90c1e1c46a24ac40b7762ec4ae77aa1d1",
         "MacAddress": "02:42:ac:11:00:02",
         "IPv4Address": "172.17.0.2/16",
         "IPv6Address": ""
```

4- Test network connectivity

The output to the previous docker network inspect command shows the IP address of the new container. or can check the container ip address-

root@node1:~# docker exec webserver hostname -i

172.17.0.2

```
root@node1:~#
root@node1:~# docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                  NAMES
853a0fac7693        nginx                          "nginx -g 'daemon of…"   11 minutes ago      Up 11 minutes       80/tcp                 webserver
ee89f0573e96        ashutoshsmaurya/ubuntu-ssh     "/usr/sbin/sshd -D"      2 hours ago         Up 2 hours          0.0.0.0:23->22/tcp     Ubuntu-SSH
root@node1:~# docker exec webserver hostname -i
172.17.0.2
root@node1:~#
```

a- Ping the IP address of the container..

root@node1:~# ping 172.17.0.2

```
root@node1:~# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.422 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.103 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.087 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.091 ms
```

b- Verify the container can connect to the outside world too...

First check the conatiner ID

root@node1:~# docker ps

Second run a shell inside that ubuntu container and install ping program..

root@node1:~# docker exec -it 853a0fac7693 /bin/bash
root@853a0fac7693:/#
root@853a0fac7693:/#
root@853a0fac7693:/# apt-get update && apt-get install -y iputils-ping
root@853a0fac7693:/#
root@853a0fac7693:/# ping google.com

```
root@node1:~# docker ps
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS              PORTS                  NAMES
853a0fac7693        nginx                       "nginx -g 'daemon of…"   20 minutes ago      Up 20 minutes       80/tcp                 webserver
ee89f0573e96        ashutoshsmaurya/ubuntu-ssh  "/usr/sbin/sshd -D"      2 hours ago         Up 2 hours          0.0.0.0:23->22/tcp     Ubuntu-SSH
root@node1:~#
root@node1:~# docker exec -it 853a0fac7693 /bin/bash
root@853a0fac7693:/#
root@853a0fac7693:/#
root@853a0fac7693:/# apt-get update && apt-get install -y iputils-ping
Get:1 http://nginx.org/packages/mainline/debian stretch InRelease [2863 B]
Get:2 http://security.debian.org stretch/updates InRelease [63.0 kB]
Get:3 http://nginx.org/packages/mainline/debian stretch/nginx amd64 Packages [21.7 kB]
Get:5 http://security.debian.org stretch/updates/main amd64 Packages [337 kB]
Ign:4 http://cdn-fastly.deb.debian.org/debian stretch InRelease
Get:6 http://cdn-fastly.deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:7 http://cdn-fastly.deb.debian.org/debian stretch Release [118 kB]
Get:8 http://cdn-fastly.deb.debian.org/debian stretch-updates/main amd64 Packages [6499 B]
Get:9 http://cdn-fastly.deb.debian.org/debian stretch Release.gpg [2434 B]
Get:10 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 Packages [9531 kB]
root@853a0fac7693:/# ping google.com
PING google.com (172.217.20.110) 56(84) bytes of data.
64 bytes from ams17s01-in-f14.1e100.net (172.217.20.110): icmp_seq=1 ttl=43 time=395 ms
64 bytes from ams17s01-in-f14.1e100.net (172.217.20.110): icmp_seq=2 ttl=43 time=395 ms
64 bytes from ams17s01-in-f14.1e100.net (172.217.20.110): icmp_seq=3 ttl=43 time=395 ms
64 bytes from ams17s01-in-f14.1e100.net (172.217.20.110): icmp_seq=4 ttl=43 time=395 ms
64 bytes from ams17s01-in-f14.1e100.net (172.217.20.110): icmp_seq=5 ttl=43 time=395 ms
root@853a0fac7693:/#
```

This shows that the new container can ping the internet and therefore has a valid and working network configuration.

5- Configure NAT for external connectivity

Cretae a new NGINX container and map port 8080 on the Docker host to port 80 inside of the container. This means that traffic that hits the Docker host on port 8080 will be passed on to port 80 inside the container...

root@node1:~# docker run -itd --name webserver -p 8080:80 nginx

root@node1:~# docker ps

```
root@node1:~# docker run -itd --name webserver -p 8080:80 nginx
73b75d7dcd49adb5d20edf40691c3c3e0c0898f16264306 7c3019246866 7655a
root@node1:~#
root@node1:~#
root@node1:~# docker ps
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS              PORTS                  NAMES
73b75d7dcd49        nginx                       "nginx -g 'daemon of…"   7 seconds ago       Up 6 seconds        0.0.0.0:8080->80/tcp   webserver
ee89f0573e96        ashutoshsmaurya/ubuntu-ssh  "/usr/sbin/sshd -D"      2 hours ago         Up 2 hours          0.0.0.0:23->22/tcp     Ubuntu-SSH
root@node1:~#
root@node1:~#
```

Container is running as well as the port mapping - 0.0.0.0:8080->80/tcp maps port 8080 on all host interfaces to port 80 inside the **webserver** container.

Now that the container is running and mapped to a port on a host interface you can test connectivity to the NGINX web server.

root@node1:~# curl 127.0.0.1:8080

```
root@node1:~# curl 127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@node1:~#
```

C- Create a new network, and then deploy a container on our new network.

1- Create a Network

Create a network with a subnet of 10.0.1.0/24, a gateway of 10.0.1.1, and the name new_subnet.

root@node2:~# docker network create --driver=bridge --subnet=10.0.1.0/24 --gateway=10.0.1.1 new_subnet

```
root@node2:~# docker network create --driver=bridge --subnet=10.0.1.0/24 --gateway=10.0.1.1 new_subnet
2f9b8bff53d376c172fce71d9ac4f9d7cb7a9b3c9af1eb6ce3de23d2f0dd3650
root@node2:~#
root@node2:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
432ff82cc384        bridge              bridge              local
d085b1b25db4        docker_gwbridge     bridge              local
56f1d80fd100        host                host                local
cmc57d48mrap        ingress             overlay             swarm
2f9b8bff53d3        new_subnet          bridge              local
292060745d10        none                null                local
```

2- Let's attach a container to our newly created network - new-subnet

root@node2:~# docker run -it -d --name Test-PC --network=new_subnet ubuntu-ssh-backup

```
root@node2:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
432ff82cc384        bridge              bridge              local
d085b1b25db4        docker_gwbridge     bridge              local
56f1d80fd100        host                host                local
cmc57d48mrap        ingress             overlay             swarm
2f9b8bff53d3        new_subnet          bridge              local
292060745d10        none                null                local
root@node2:~# docker run -it -d --name Test-PC --network=new_subnet ubuntu-ssh-backup
f8b2c3d038c33319f7514f7c81e803ablfdc65a1ff3036922ce9b038087bdb6d
root@node2:~#
root@node2:~#
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
f8b2c3d038c3        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      5 seconds ago       Up 4 seconds        22/tcp                  Test-PC
5c4d7bc7bf89        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      2 hours ago         Up 2 hours          0.0.0.0:29->22/tcp      conatiner3
30d4ea23dd7d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      3 hours ago         Up 3 hours          0.0.0.0:28->22/tcp      container2
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      5 hours ago         Up 4 hours          0.0.0.0:27->22/tcp      container1
```

3- Check IP Address - New Container - Test-PC

root@node2:~# docker exec Test-PC hostname -i

10.0.1.2

```
root@node2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
f8b2c3d038c3        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      11 minutes ago      Up 11 minutes       22/tcp                  Test-PC
5c4d7bc7bf89        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      2 hours ago         Up 2 hours          0.0.0.0:29->22/tcp      conatiner3
30d4ea23dd7d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      3 hours ago         Up 3 hours          0.0.0.0:28->22/tcp      container2
bf4f2294e98d        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      5 hours ago         Up 4 hours          0.0.0.0:27->22/tcp      container1
d18d4614e43e        ubuntu-ssh-backup   "/usr/sbin/sshd -D"      5 hours ago         Up 5 hours          0.0.0.0:26->22/tcp      Ubuntu-Server-SSH
root@node2:~#
root@node2:~#
root@node2:~# docker exec Test-PC hostname -i
10.0.1.2
root@node2:~#
```

Now Container Sussceesfuuly Attached with New Network - new-subnet.

D- Overlay Networking

1- In this step you'll initialize a new Swarm, join a single worker node, and verify the operations worked.

[root@docker-server ~]# docker swarm init --advertise-addr 172.16.15.10

```
[root@docker-server ~]# docker swarm init --advertise-addr 172.16.15.10
Swarm initialized: current node (yef6zfsb9dkt5055hw6dnq280) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-26dalfgtf0hw639v1qq1zi0sfx3zqrr56macwlbty9m99pygjm-178a3noqw8o4ga3zygw6c2f01 172.16.15.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[root@docker-server ~]#
```

2- Copy the entire docker swarm join ... command that is displayed as part of the output from your terminal output. Then, paste the copied command into the second terminal.

root@node1:~# docker swarm join --token SWMTKN-1-26da1fgtf0hw639v1qq1zi0sfx3zqrr56macwlbty9m99pygjm-178a3noqw8o4ga3zygw6c2f01 172.16.15.10:2377

```
root@node1:~# date
Fri Jan 19 01:53:16 EST 2018
root@node1:~# docker swarm join --token SWMTKN-1-26dalfgtf0hw639v1qq1zi0sfx3zqrr56macwlbty9m99pygjm-178a3noqw8o4ga3zygw6c2f01 172.16.15.10:2377
This node joined a swarm as a worker.
root@node1:~#
```

3- Run a docker node ls to verify that both nodes are part of the Swarm.

[root@docker-server ~]# docker node ls

```
[root@docker-server ~]#
[root@docker-server ~]# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
r239ktsup4rxlvu420zkwbmd2 *   docker-server       Ready               Active              Leader
sdk4tgn4wllafehswb9ielul2     node1               Ready               Active
[root@docker-server ~]#
[root@docker-server ~]#
```

4- Create an overlay network

Now that you have a Swarm initialized it's time to create an overlay network.

a- Create a new overlay network called "overnet"..

[root@docker-server ~]# docker network create -d overlay overnet

```
[root@docker-server ~]#
[root@docker-server ~]# docker network create -d overlay overnet
p0ki8ndyurilbqfobwdvfqu7x
[root@docker-server ~]#
```

b- verify the network was created successfully.

[root@docker-server ~]# docker network ls

```
[root@docker-server ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
088918d3b837        bridge              bridge              local
01bf4bc556e0        docker_gwbridge     bridge              local
8b5679aeb414        host                host                local
wzolg4cplb3i        ingress             overlay             swarm
b8ee58fac2b0        none                null                local
p0ki8ndyuril        overnet             overlay             swarm
[root@docker-server ~]#
```

The new "overnet" network is shown on the last line of the output above. Notice how it is associated with the overlay driver and is scoped to the entire Swarm.

Note- The other new networks (ingress and docker_gwbridge) were created automatically when the Swarm cluster was created.

c- Run 'docker network ls' command Second - Node1 Terminal

root@node1:~# docker network ls

```
root@node1:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9d2a430f8498        bridge              bridge              local
3f1a5804380f        docker_gwbridge     bridge              local
56f1d80fd100        host                host                local
wzolg4cplb3i        ingress             overlay             swarm
292060745d10        none                null                local
root@node1:~#
```

Notice that the “overnet” network does not appear in the list. This is because Docker only extends overlay networks to hosts when they are needed..

d-Check more information about the “overnet” network.

[root@docker-server ~]# docker network inspect overnet

```
[root@docker-server ~]# docker network inspect overnet
[
    {
        "Name": "overnet",
        "Id": "p0ki8ndyurilbqfobwdvfqu7x",
        "Created": "0001-01-01T00:00:00Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": null,
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4097"
        },
        "Labels": null
    }
]
[root@docker-server ~]#
```

5- Create a service

a- Create a new service called myservice on the overnet network with two tasks/replicas.

[root@docker-server ~]# docker service create --name myservice --network overnet --replicas 2 -p 8088:80 nginx

```
[root@docker-server ~]# docker service create --name myservice --network overnet --replicas 2 -p 8088:80 nginx
oxzp5hppmppcjbyuxxpzpe2ol
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
[root@docker-server ~]#
```

b- Verify that the service is created or not

[root@docker-server ~]# docker service ls

```
[root@docker-server ~]# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
oxzp5hppmppc        myservice           replicated          2/2                 nginx:latest        *:8088->80/tcp
[root@docker-server ~]#
[root@docker-server ~]#
```

c- Verify that a single task (replica) is running on each of the two nodes in the Swarm..

[root@docker-server ~]# docker service ps myservice

```
[root@docker-server ~]# docker service ps myservice
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE               ERROR               PORTS
kee9c4fsuelr        myservice.1         nginx:latest        docker-server       Running             Running about a minute ago
iewlamzr2gju        myservice.2         nginx:latest        node1               Running             Running 2 minutes ago
[root@docker-server ~]#
```

Note- each task/replica is running on a different node.

d- Now that the second node is running a task on the "overnet" network it will be able to see the "overnet" network. Lets run docker network ls from the second terminal to verify this.

[root@docker-server ~]# docker network ls

```
[root@docker-server ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
088918d3b837        bridge              bridge              local
01bf4bc556e0        docker_gwbridge     bridge              local
8b5679aeb414        host                host                local
wzolg4cplb3i        ingress             overlay             swarm
b8ee58fac2b0        none                null                local
p0ki8ndyuril        overnet             overlay             swarm
```

We can also run docker network inspect overnet on the second terminal to get more detailed information about the "overnet" network and obtain the IP address of the task running on the second terminal.

[root@docker-server ~]# docker network inspect overnet

```
[root@docker-server ~]# docker network inspect overnet
[
    {
        "Name": "overnet",
        "Id": "p0ki8ndyurilbqfobwdvfqu7x",
        "Created": "2018-01-19T12:39:20.451057186+05:30",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "a250665f84bea474b16d764581c1a00d160d8f29512054d7207de05ef18602dc": {
                "Name": "myservice.1.kee9c4fsuelrhxyvbli19tg4r",
                "EndpointID": "87aadaac399436a06b51839eaf582d8b82ff991f173899c86f4640bb89e4353f",
                "MacAddress": "02:42:0a:00:00:08",
                "IPv4Address": "10.0.0.8/24",
                "IPv6Address": ""
            }
```

Above command shows containers/tasks running on the local node. This means that 10.0.0.8 is the IPv4 address of the container running on the second node.

e- Test the network

First Check the ID of the services task..

[root@docker-server ~]# docker ps

```
[root@docker-server ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
a250665f84be        nginx:latest        "nginx -g 'daemon of…"   16 minutes ago      Up 16 minutes       80/tcp              myservice.1.kee9c4fsuelrhxyvbli19tg4r
[root@docker-server ~]#
[root@docker-server ~]#
```

Log on to the service task.

[root@docker-server ~]# docker exec -it a250665f84be /bin/bash

f- Install the ping command and ping the service task running on the second node where it had a IP address of 10.0.0.3 from the docker network inspect overnet command.

```
root@a250665f84be:/#
root@a250665f84be:/#
root@a250665f84be:/# apt-get update && apt-get install -y iputils-ping
Get:1 http://security.debian.org stretch/updates InRelease [63.0 kB]
Get:4 http://security.debian.org stretch/updates/main amd64 Packages [337 kB]
Ign:2 http://cdn-fastly.deb.debian.org/debian stretch InRelease
Get:3 http://cdn-fastly.deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:5 http://cdn-fastly.deb.debian.org/debian stretch Release [118 kB]
Get:6 http://cdn-fastly.deb.debian.org/debian stretch-updates/main amd64 Packages [6499 B]
Get:7 http://cdn-fastly.deb.debian.org/debian stretch Release.gpg [2434 B]
Get:8 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 Packages [9531 kB]
Get:9 http://nginx.org/packages/mainline/debian stretch InRelease [2863 B]
Get:10 http://nginx.org/packages/mainline/debian stretch/nginx amd64 Packages [21.7 kB]
Fetched 10.2 MB in 32s (310 kB/s)
Reading package lists... Done
Reading package lists... Done
```

Check ping status

```
root@a250665f84be:/# ping google.com
PING google.com (74.125.196.138) 56(84) bytes of data.
64 bytes from yk-in-f138.1e100.net (74.125.196.138): icmp_seq=1 ttl=40 time=296 ms
64 bytes from yk-in-f138.1e100.net (74.125.196.138): icmp_seq=2 ttl=40 time=296 ms
64 bytes from yk-in-f138.1e100.net (74.125.196.138): icmp_seq=3 ttl=40 time=296 ms
64 bytes from yk-in-f138.1e100.net (74.125.196.138): icmp_seq=4 ttl=40 time=296 ms
```

The output above shows that both tasks from the myservice service are on the same overlay network spanning both nodes and that they can use this network to communicate.

Docker Backup & Restore

This config will describe a procedure of how to back up a Docker container as well as it will also show how to recover a Docker container from backup.

1- Before backing up a container, you need to identify its container ID. To know the container ID of a Docker instance, you can list the containers in that system.

root@node1:~# docker ps

```
root@node1:~# docker ps
CONTAINER ID        IMAGE                          COMMAND               CREATED             STATUS              PORTS                   NAMES
ee89f0573e96        ashutoshsmaurya/ubuntu-ssh     "/usr/sbin/sshd -D"   10 seconds ago      Up 9 seconds        0.0.0.0:23->22/tcp      Ubuntu-SSH
root@node1:~#
root@node1:~#
root@node1:~#
```

2- After that, we'll choose the containers we wanna backup and then we'll go for creating the snapshot of the container. We can use docker commit command in order to create the snapshot.

docker commit -p <container-ID> <backup-name>

root@node1:~# docker commit -p ee89f0573e96 ubuntu-ssh-backup

```
root@node1:~#
root@node1:~# docker commit -p ee89f0573e96 ubuntu-ssh-backup
sha256:8bf6f0392a04d18be1503f7a6bda23d0b74de530ac143bbbfd545715ab456d5f
root@node1:~#
root@node1:~#
```

This snippet shows the docker backup of container 'ee89f0573e96 ' (WordPress container), being taken in the name 'ubuntu-ssh-backup'.

3- This will generated a snapshot of the container as the docker image. We can see the docker image by running the command docker images as shown below.

root@node1:~# docker images

```
root@node1:~# docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
ubuntu-ssh-backup           latest              8bf6f0392a04        19 seconds ago      205MB
ashutoshsmaurya/ubuntu-ssh  latest              bf040e61afaf        42 hours ago        205MB
root@node1:~#
root@node1:~#
```

4- **It is not advisable to store the backups in the same Docker host machine as the container, as a hardware crash in it can tamper with the backups too.**

So we have two options, one is that we can login into the docker registry hub and push the image and the next is that we can backup the docker image as tarballs for further use.

First Option-

Backup the image in the docker registry hub-

root@node1:~# docker login
root@node1:~# docker tag 8bf6f0392a04 ashutoshsmaurya/container-backup
root@node1:~# docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ashutoshsmaurya/container-backup	latest	8bf6f0392a04	10 minutes ago	205MB
ubuntu-ssh-backup	latest	8bf6f0392a04	10 minutes ago	205MB

root@node1:~# docker push ashutoshsmaurya/container-backup

```
root@node1:~#
root@node1:~# docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
ubuntu-ssh-backup           latest              8bf6f0392a04        4 minutes ago       205MB
ashutoshsmaurya/ubuntu-ssh  latest              bf040e61afaf        42 hours ago        205MB
root@node1:~# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: ashutoshsmaurya
Password:
Login Succeeded
root@node1:~#
root@node1:~# docker tag 8bf6f0392a04 ashutoshsmaurya/container-backup
root@node1:~#
root@node1:~# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
ashutoshsmaurya/container-backup    latest              8bf6f0392a04        10 minutes ago      205MB
ubuntu-ssh-backup                   latest              8bf6f0392a04        10 minutes ago      205MB
ashutoshsmaurya/ubuntu-ssh          latest              bf040e61afaf        42 hours ago        205MB
root@node1:~# docker push ashutoshsmaurya/container-backup
The push refers to repository [docker.io/ashutoshsmaurya/container-backup]
a7688c93b221: Pushed
b2ce227e4f4b: Mounted from ashutoshsmaurya/ubuntu-ssh
156a1a0109dd: Mounted from ashutoshsmaurya/ubuntu-ssh
63b73ca1bd51: Mounted from ashutoshsmaurya/ubuntu-ssh
2654f256f53c: Mounted from ashutoshsmaurya/ubuntu-ssh
52266844f919: Mounted from ashutoshsmaurya/ubuntu-ssh
aea42d1af8d5: Mounted from ashutoshsmaurya/ubuntu-ssh
f17fc24fb8d0: Mounted from ashutoshsmaurya/ubuntu-ssh
6458f770d435: Mounted from ashutoshsmaurya/ubuntu-ssh
5a876f8f1a3d: Mounted from ashutoshsmaurya/ubuntu-ssh
d2f8c05d353b: Mounted from ashutoshsmaurya/ubuntu-ssh
48e0baf45d4d: Mounted from ashutoshsmaurya/ubuntu-ssh
latest: digest: sha256:7b5ff94074dd0e747219acedcd3528ba1bbcfab4ddc3eac8de70a7adcda5971a size: 2813
```

Second Option-

Backup images to compressed formats such as ‘tar’ files and copy them over to an external server.

root@node1:~# docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ashutoshsmaurya/container-backup	latest	8bf6f0392a04	10 minutes ago	205MB
ubuntu-ssh-backup	latest	8bf6f0392a04	10 minutes ago	205MB

root@node1:~# docker save -o container-backup.tar ubuntu-ssh-backup

Now verify backup file genrated or not

root@node1:~# ls

```
root@node1:~# ls
container-backup.tar  dockerfile

root@node1:~#
```

Restore Container

1- If we have pushed those docker images in the registry hub, then we can simply pull that docker image and run it out of the box

root@node2:~# docker pull ashutoshsmaurya/container-backup

```
root@node2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
root@node2:~#
root@node2:~#
root@node2:~# docker pull ashutoshsmaurya/container-backup
Using default tag: latest
latest: Pulling from ashutoshsmaurya/container-backup
50aff78429b1: Already exists
f6d82e297bce: Already exists
275abb2c8a6f: Already exists
9f15a39356d6: Already exists
fc0342a94c89: Already exists
8b0068e18736: Pull complete
a3dd9b063af9: Pull complete
24c2f6889376: Pull complete
f0976d371e47: Pull complete
9cdb22636986: Pull complete
21190de243be: Pull complete
398a88724f34: Pull complete
Digest: sha256:7b5ff94074dd0e747219acedcd3528balbbcfab4ddc3eac8de70a7adcda5971a
Status: Downloaded newer image for ashutoshsmaurya/container-backup:latest
root@node2:~#
root@node2:~#
root@node2:~#
root@node2:~# docker images
REPOSITORY                        TAG                 IMAGE ID            CREATED             SIZE
ashutoshsmaurya/container-backup  latest              8bf6f0392a04        About an hour ago   205MB
root@node2:~#
```

2- But if we have backed up those docker images locally as tarball file, then we can easy load that docker image using docker load command

root@node2:~# docker load -i container-backup.tar

```
root@node2:~#
root@node2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
root@node2:~#
root@node2:~#
root@node2:~#
root@node2:~# docker load -i container-backup.tar
aea42d1af8d5: Loading layer [==================================================>]  98.48MB/98.48MB
52266844f919: Loading layer [==================================================>]  2.048kB/2.048kB
2654f256f53c: Loading layer [==================================================>]  3.072kB/3.072kB
63b73ca1bd51: Loading layer [==================================================>]   5.12kB/5.12kB
156a1a0109dd: Loading layer [==================================================>]   5.12kB/5.12kB
b2ce227e4f4b: Loading layer [==================================================>]  3.072kB/3.072kB
a7688c93b221: Loading layer [==================================================>]  43.01kB/43.01kB
Loaded image: ubuntu-ssh-backup:latest
root@node2:~#
```

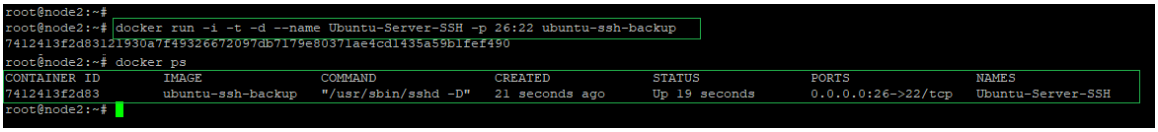
3- Docker images have been loaded successfully Or Not

root@node2:~# docker images

```
root@node2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu-ssh-backup   latest              8bf6f0392a04        About an hour ago   205MB
root@node2:~#
root@node2:~#
```

4- Once the backup images are listed in the Docker host, you can restore the container by using 'docker run' command and specifying the backup image.

root@node2:~# docker run -i -t -d --name Ubuntu-Server-SSH -p 26:22 ubuntu-ssh-backup



Done..!!
Successfully completed all topics..!!

Reference- https://docs.docker.com **-Ashutosh**