**DOCKER 🡪**

* The main purpose of docker is to package and containerized application & to ship them and to run them anywhere anytime as many times you want.
* Dockerhub is "Public Docker Registry/Repository"
* Docker Image is package or template just like a VM template which is used to create one or more containers.
* Docker Container are running instances of images.
* /var/lib/docker/ is the location of docker containers, images, networks etc.

**NOTE 🡪** Kernel should be same to run any container on docker engine. We can't run any linux docker container on windows machine and vice versa. Because, Kernel is not same. so, Windows containers can run on Windows only, and Linux containers can run on Linux only.

To make it possible, we need to setup VM then we can proceed with linux container on windows.

**COMMANDS 🡪**

1. docker ps -a 🡪 shows all state of containers
2. docker ps 🡪 shows only running containers
3. docker pull <imagename> 🡪 Only pull image from Docker Hub
4. docker run <imagename:tag> 🡪 First pull image if it does not exist then make container from that image. i.e., "docker run" calls first "docker pull" incase image does not exist locally.
5. docker stop/start/rm <NAMES/CONTAINER ID> 🡪 To stop Container
6. docker images 🡪 List Images
7. docker rmi <imagename:tag> 🡪 To delete image
8. docker exec <CONTAINER ID> <OS Command>
9. docker inspect <image/container etc>
10. docker run -p 8080:80 <Dockerimage> 🡪 -p: publish, cmd use for port mapping, to set connectivity between docker container and external user.

**NOTE 🡪** Why some container gets immediately stopped after running "docker run <imagename>”. Example, after running "docker run ubuntu" command, docker container immediately get stop.  
**Answer** 🡪 Container is basically used for deploying any application/task/process not OS. Container only lives if the process inside it is alive. If the service in container is stopped, then the container exists. This is the reason, ubuntu container always immediately get stopped after running "docker run ubuntu" command because there is no process inside it.

Resolution Command 🡪 docker run -td <imagename> [-d: detach mode, to run container in background]

**DOCKER IMAGES 🡪**

All below task will include in docker image.

1. OS Ubuntu
2. Update apt repo
3. Install dependencies using apt
4. Install Python dependencies using pip
5. Copy source code to /opt folder
6. Run the web server using “flask” command

*FROM ubuntu*

*ENV HOME /root*

*ENV DEBIAN\_FRONTEND noninteractive*

*RUN apt-get update && apt-get -y install python python-pip apt-utils curl*

*RUN pip install flask flask-mysql*

*COPY . /opt/source-code*

*ENTRYPOINT FLASK\_APP=/opt/source-code/app.py flask run --host=0.0.0.0 --port=5000*

**DOCKERFILE INSTRUCTIONS:**EXPOSE🡪 Expose n/w protocol (TCP by default) to container. e.g., EXPOSE 80/udp  
ENV 🡪 Sets Env Variable  
ENTRYPOINT 🡪 Meant to provide the executable  
CMD 🡪 To pass the default arguments to the executable.

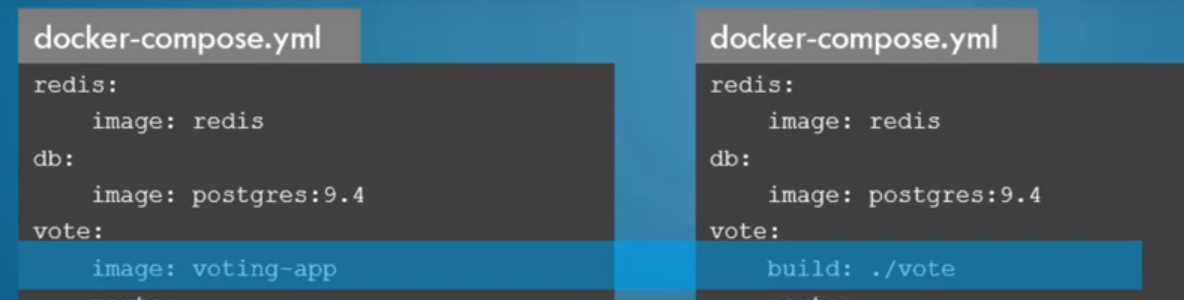
Docker Build Command Scenarios:

* When "Dockerfile/dockerfile” (which docker assumes its name by default)  
  docker build /root/docker/ -t sagar01/my\_custom\_app
* When "sam" is dockerfile name 🡪 Go to sam's directory 🡪 run docker build command  
  docker build /root/docker/ -f sam -t sagar01/my\_custom\_app  
  OR  
  docker build . -f sam -t sagar01/my\_custom\_app

**DOCKER COMPOSE 🡪**

Used to start all docker container (from different images) which is required for application stack. So, write docker yaml file and put all configurations in yaml and run "**docker-compose up -d**" to start the application stack in one command.

If any image does not exist in docker hub in case of own application, then we should define as below in docker-compose.yaml.



./vote 🡪 Define the location of application code.

Docker compose has 3 versions. Version 3 is most current and recommended.

**DOCKER REGISTRY 🡪**

Syntax : Registry/User/Image

Image : docker.io/nginx/nginx

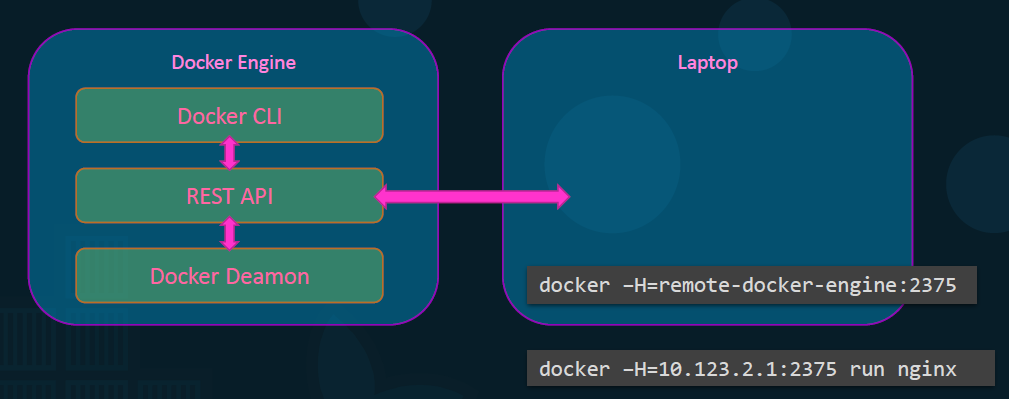
docker login private-registry.io 🡪 command use to push and pull images from/to docker hub by using your Docker login ID credentials.

docker run private-registry.io/apps/internal-app 🡪 run container from logged in registry

**DOCKER ENGINE 🡪**

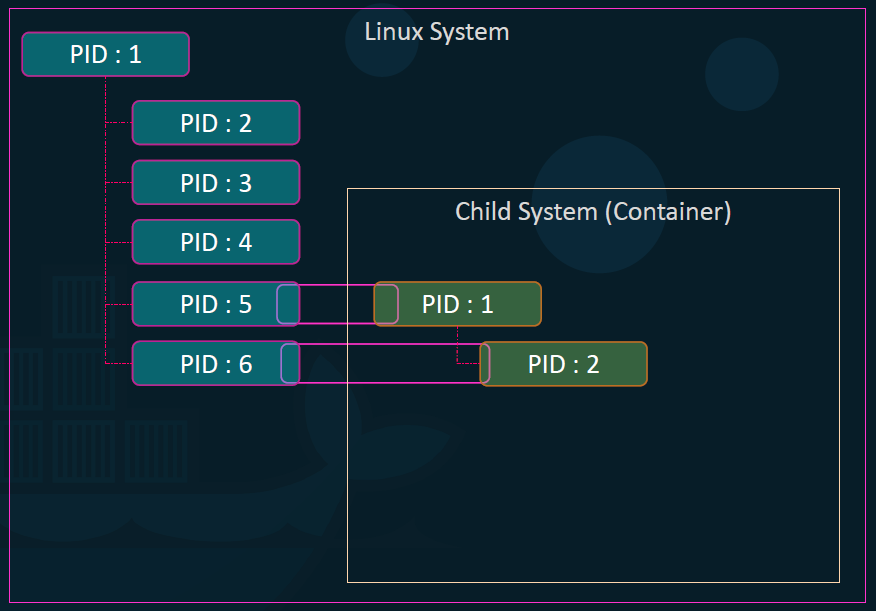
When we install docker on any OS then it basically installs Docker Engine, Which consist.

1. Docker CLI - Command line interface to execute docker commands thus it uses REST API to talk with docker daemon
2. REST API - Talk to the Docker daemon and provide instructions
3. Docker Daemon - It manage background process like docker images, containers, networks, volumes etc.



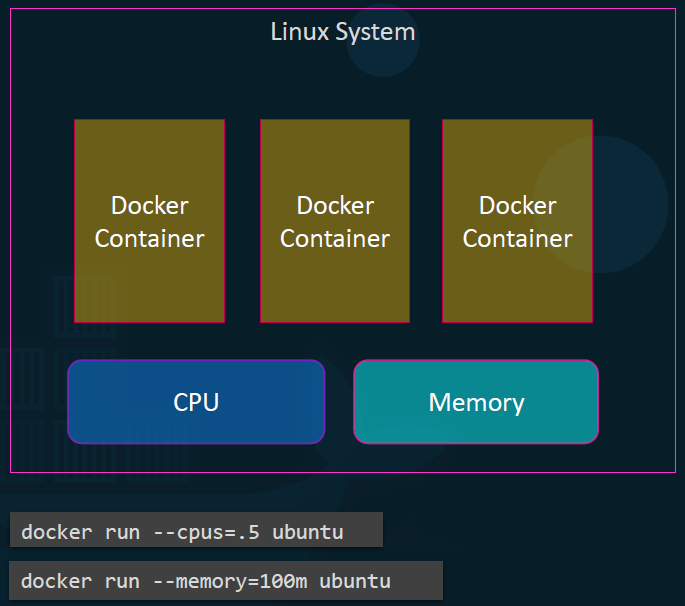
**NOTE 🡪** We can use Docker CLI from any host X, if DOCKER ENGINE (API and DAEMON) running on host Y. It can be done just execute below command

* docker -H=10.123.1.2:2375 run nginx

**NAMESPACE – PID 🡪**

PID can’t be same among docker container and Docker HOST. PID5 (DOCKER HOST PID) map with PID1 (NGINX Container PID) with different PID number.

**CGROUPS 🡪**  
By default, Docker HOST and Docker Container use same resources, so there is no limit for resource utilization by Docker Container.  
So , in order to set resource limit docker use CGROUPS (Control Groups) which allocate resource limit to each docker container.



**DOCKER VOLUMES 🡪**

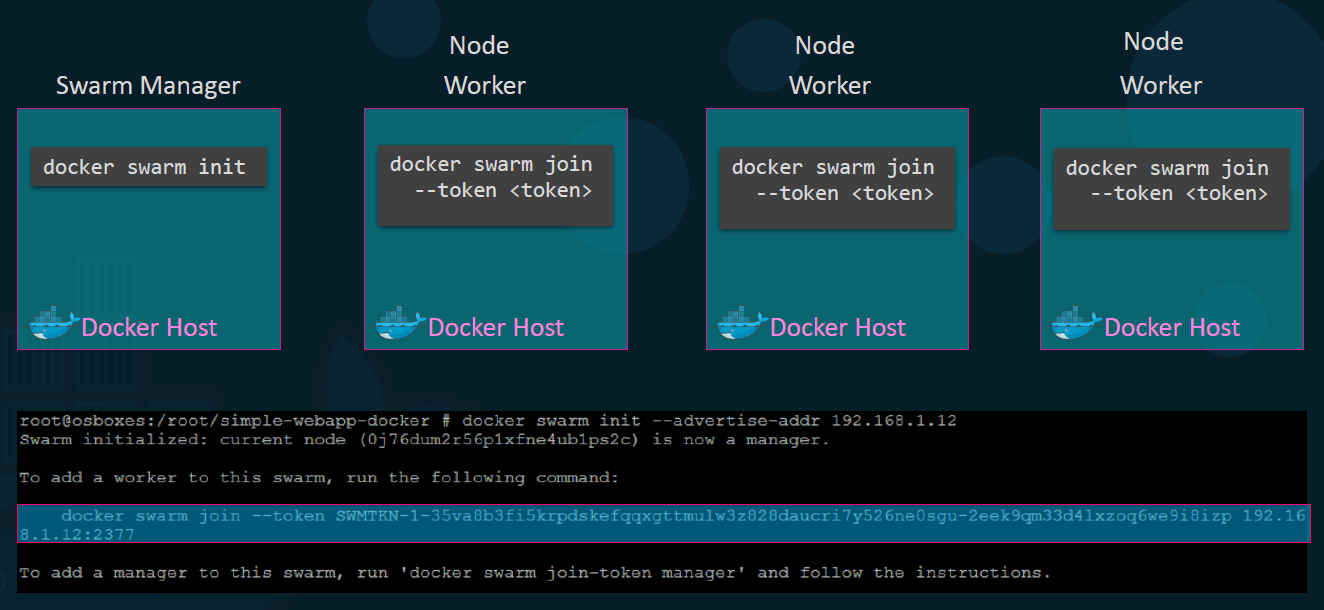
Docker has two options to store container files in Docker host

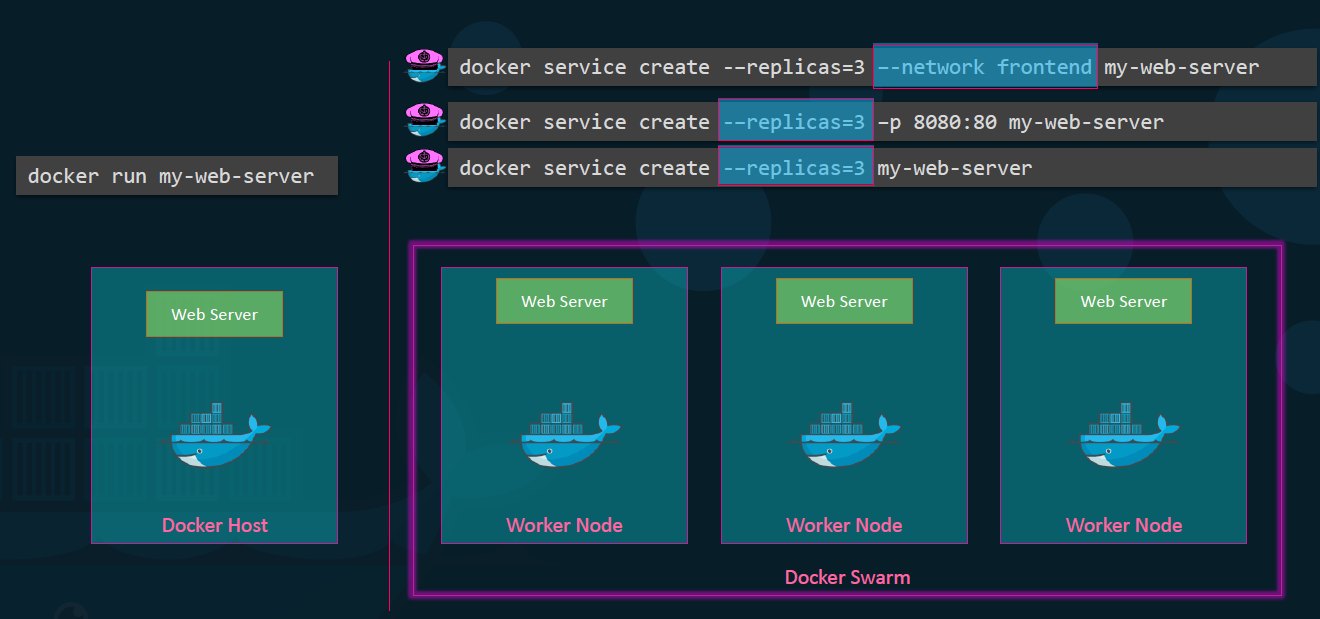
1. **Volumes** : By default, docker store container’s data in **/var/lib/docker/volumes/** on docker host.   
   docker run -v datadir:/var/lib/mysql mysql   
   (Here, datadir directory automatically build in /var/lib/docker/volumes/)
2. **Bind Mounts** : Use to mount data other than default path.   
   docker run -v /opt/data:/var/lib/mysql mysql

**CONTAINER ORCHESTRATION 🡪**

Docker Swarm and Kubernetes

**Docker Swarm** 🡪 Here one is manager (master node) and other are worker (slave node) runs on separate docker host with running docker service on each docker host.  
And, make sure Docker worker nodes should join master node to orchestrate any application.





Best practice is to use more than 1 swarm manager in cluster to avoid single type of failure. Suppose, If we are using 3 swarm manager then only 1 swarm manager will be a leader who will take decision after taking mutual decision among 3 swarm manager.

Why mutual decision concept is here , if Leader (swarm manager) takes decision on its own without any mutual decision to add other worker nodes & by some kind of issue that leader get fail to add node worker.

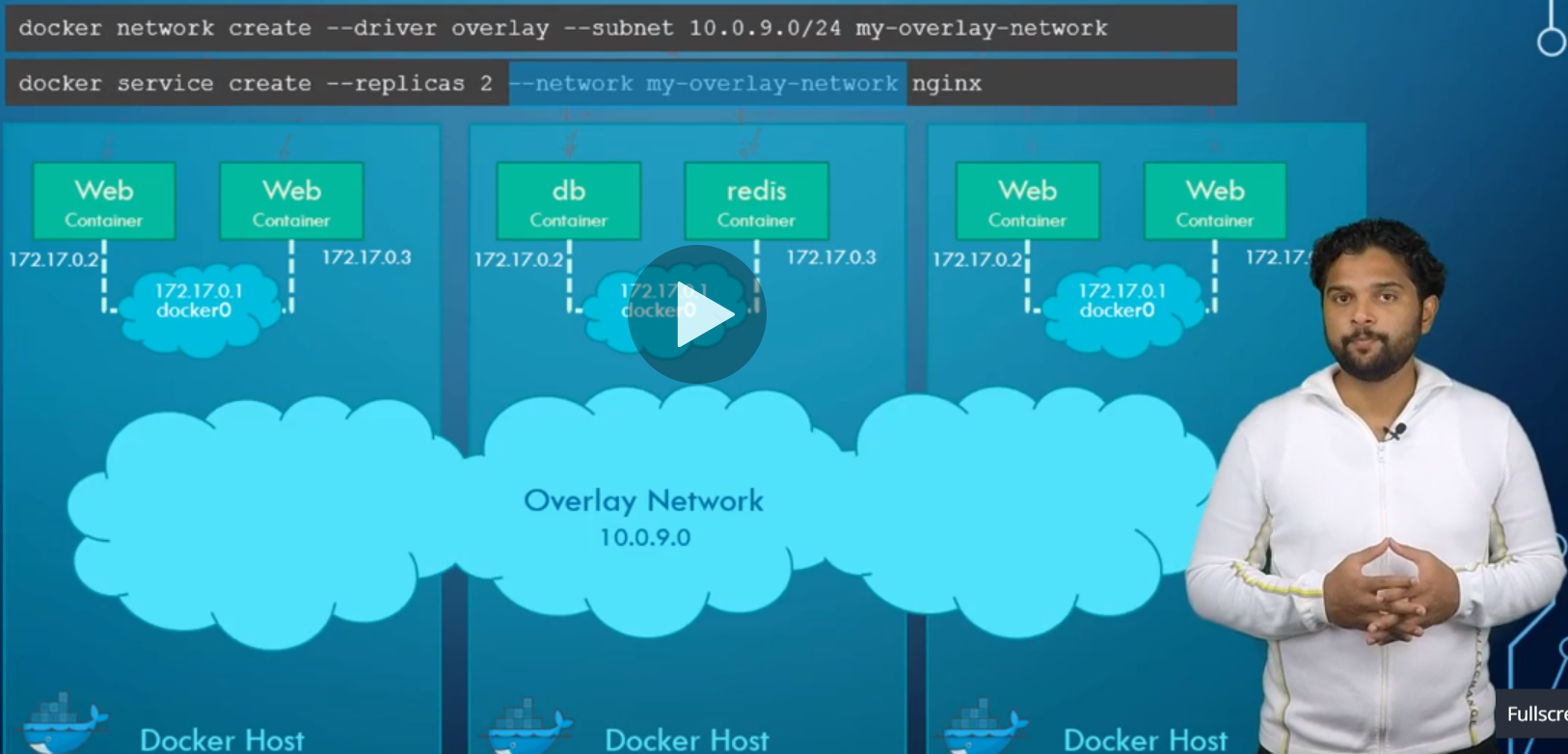
Then Cluster get inconsistent because rest two manager are not aware about failure of leader.

RAFT Algorithm is used to make Leader if swarm manager is more than 1.

Docker recommends 7 managers. And there is no limit of manager.

**DOCKER NETWORKING 🡪**

Docker has by default 3 networks

1. Bridge Network – Default network (created by docker) for containers. (172.17.X.X is the by default series for bridge network)
2. None 🡪 Used for other network except Bridge. **(docker run Ubuntu –network=none)**
3. Host 🡪 Used for other network except Bridge. **(docker run Ubuntu –network=host)**
4. Overlay Network 🡪 Used to basically make a successful network among different docker host.  
     
   
5. Ingress Network 🡪 It is a by default network when any swarm cluster deploy. In below image, An External user cannot publish port to access both containers (172.17.0.2 and 172.17.0.3) because all containers are deployed in docker swarm. So Docker swarm automatically create Ingress network along with swarm load-balancing (internal load balancer) to route request on both the containers.  
     
   To use the ingress network in the swarm, following ports should be open -  
   **Port 7946 TCP/UDP for container network discovery.  
   Port 4789 UDP for the container ingress network.**  
      
   