Docker certified Associate

Docker Installation on Ubuntu using Repositary

✓ Supported operating system

- Ubuntu Hirsute 21.04
- Ubuntu Groovy 20.10
- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)

✓ Uninstall old versions

✓ **apt-get remove docker docker-engine docker.io containerd runc**

Docker Installation on Ubuntu using Repositary

✓ Supported storage driver

- aufs
- Overlay2 by default these is available
- Btrfs

✓ Installation method

- ✓ Repository
- ✓ Install from the package
- ✓ Install using script

Docker Installation on Ubuntu using Repositary

- ✓ Installation method-Repository

```
root@docker:~# apt-get install \
> apt-transport-https \
> ca-certificates \
> curl \
> gnupg \
> lsb-release
```

- ✓ Add official GPG Key

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Docker Installation on Ubuntu using Repositary

✓ Command for the stable repository

```
echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

✓ Update and install the packages

```
sudo apt-get update
 sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Uninstall the docker engine

✓ Uninstall the docker engine

```
apt-get purge docker-ce docker-ce-cli containerd.io
```

To delete all images, containers, and volumes:

```
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd
```

Configure docker to start on boot

- ✓ Start the service using docker

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

systemd vs daemon.json

- ✓ Systemd unit file change to accept remote connection

Configuring Docker to accept remote connections can be done with the docker.service systemd unit file for Linux distributions using systemd, such as recent versions of RedHat, CentOS, Ubuntu and SLES, or with the daemon.json file which is recommended for Linux distributions that do not use systemd.

Configuring remote access with systemd unit file

Add the /lib/system/system/docker.service file

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://127.0.0.1:2375  <== unencrypted connection
```

systemd vs daemon.json

- ✓ Docker service restart to take config changes

```
systemctl daemon-reload
systemctl restart docker.service
```

```
netstat  –nltp
```

systemd vs daemon.json

- ✓ Daemon.json change to accept remote connection

```
{
  "hosts": ["unix:///var/run/docker.sock", "tcp://127.0.0.1:2375"]
}
```

Docker Container Command Line

- ✓ docker container run --name mynginx -dt nginx ⟸ Image that you want to pull

Container name that you want to apply Container run in detached mode with tty enable

- ✓ docker container stop [ID | name]
- ✓ docker container ls -a <<= to list all the container -l for the latest -s size -q Id
- ✓ docker container rm [ID | name] to remove the container
- ✓ docker container restart [ID | name] to restart the container
- ✓ docker container rename oldcontainername newcontainername

Docker Container Command Line

- ✓ docker container stats mynginx <<= find the cpu memory and disk io usage
- ✓ docker container exec -it mynginx /bin/bash <<= login interactively into container
- ✓ docker container pause [container name] <== pause the container process
- ✓ docker container unpause [container name] <== resume the process
- ✓ docker container top [container name] <== show process of container using pid
- ✓ docker container inspect [container name] <== find the container information
- ✓ docker container logs [container name] <== display container logs

Docker Container Command Line

- ✓ docker container port mynginx <<= list the port mapping
- ✓ docker container cp mytext.txt mynginx:/etc/ <<= copy file from host to container
- ✓ docker container cp snap/ mynginx:/etc/ <<= directory can be copied
- ✓ docker container create <== to create the continaer with custom parameter
- ✓ docker container export mynginx -o mynginx.tar <<= export the container filesystem
- ✓ docker container diff mynginx <== inspect the file and directory changes in container
- ✓ docker container commit<== create the docker image from container changes
- ✓ docker container prune<== remove the container that are stopped

Docker Container Command Line

- ✓ docker events <<= grab the live log of for the command perform on docker server
- ✓ docker import mynginx.tar <<= import the filesystem as image /use file that created by export command
- ✓ docker image tag 079391614c6f newnginx <<= import the filesystem as image
- ✓ docker container create <== to create the continaer with custom parameter
- ✓ Docker info <<= provide the information about the docker environment

Create custom docker image -ubuntu

- ✓ Create the directory customimage [you can choose any name for the directory
- ✓ Create the DockerFile inside the directory

```
FROM ubuntu
RUN apt-get update && apt-get install nginx -y
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- ✓ Docker image build -t vijay/nginx:v1 . To create the custom image
- ✓ Docker container run -dt --name testing vijay/nginx:v1
- ✓ Docker container exec -it testing /bin/bash <<= access the container interactively

Create custom docker image -centos

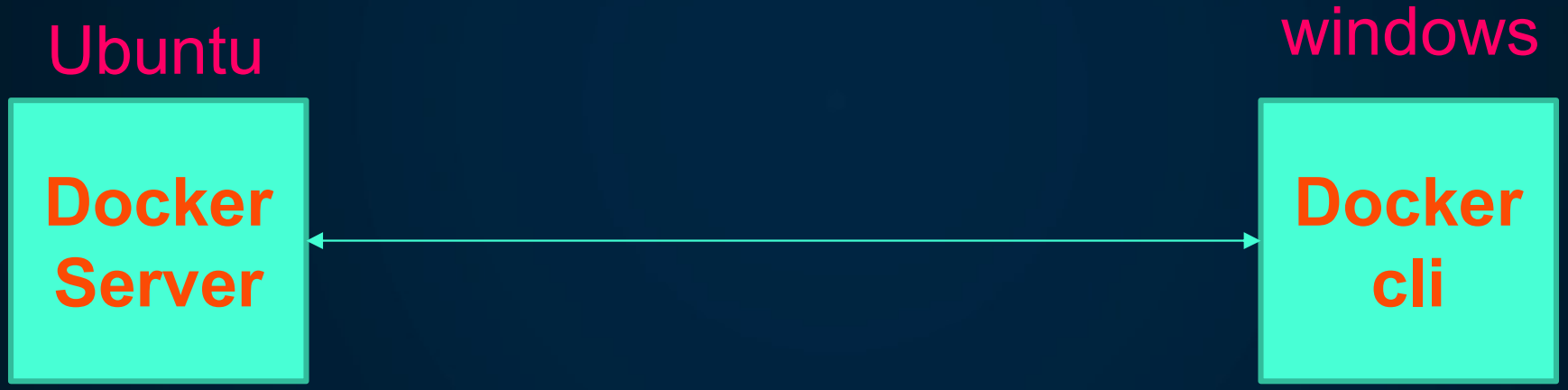
- ✓ Create the directory centoshttpd [you can choose any name for the directory
- ✓ Create the DockerFile inside the directory

```
FROM centos:7
RUN yum update -y && yum install httpd -y
COPY ./index.html  /var/www/html/index.html  < == index html file should be present in directory
EXPOSE 80
CMD ["httpd", "-D" ,"FOREGROUND"]
```

- ✓ Docker image build -t vijay/apche:v1 . To create the custom image
- ✓ Docker container run -dt --name myserver vijay/apache:v1
- ✓ Docker inspect myserver [grab Ip of container | and perform curl using host machine

Remotely execute command on docker server

Connection are unencrypted – port 2375



Steps to configure the unencrypted connection

Configuration step on Ubuntu server

- ✓ Vi /usr/lib/systemd/system/docker.service
- ✓ Add ExecStart=/usr/bin/dockerd -H fd:// -H tcp://192.168.0.246:2375 --containerd=/run/containerd/containerd.sock
- ✓ Systemctl restart daemon-reload
- ✓ Systemctl restart docker .service

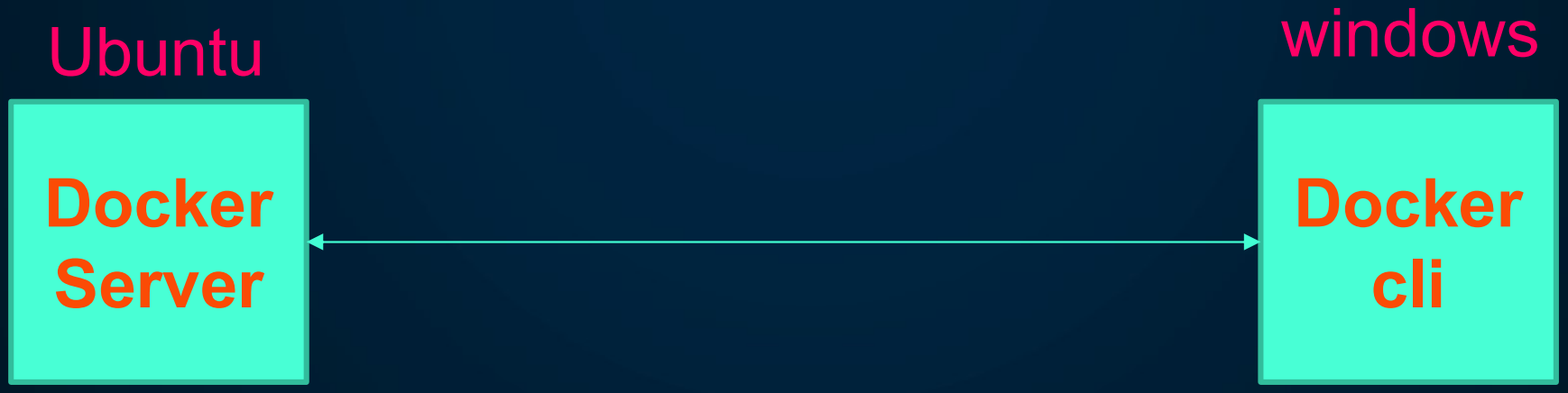
Configuration step on windows machine

- ✓ Install the docker for windows desktop
- ✓ Install WSL and its kernel plugin
- ✓ Run the CMD as administrator and set the variable as below
  set docker_host=tcp://192.168.0.246:2375
  then execute the command it will display the content of remote inventory

Remotely execute command on docker server

Connection are encrypted – port 2376

Use TLS (HTTPS) to protect the Docker daemon socket



Use TLS (HTTPS) to protect the Docker daemon socket

Create a CA, server and client keys with OpenSSL

- ✓ **Docker daemon's host machine**, generate CA private and public keys
- ✓ openssl genrsa -aes256 -out ca-key.pem 4096
- ✓ openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem
- ✓ openssl genrsa -out server-key.pem 4096
- ✓ openssl req -subj "/CN=docker.example.dom" -sha256 -new -key server-key.pem -out server.csr
- ✓ echo subjectAltName = DNS:docker.example.dom,IP:192.168.0.246,IP:127.0.0.1 >> extfile.cnf
- ✓ echo extendedKeyUsage = serverAuth >> extfile.cnf
- ✓ openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey ca-key.pem \ -CAcreateserial -out server-cert.pem -extfile extfile.cnf

For client authentication, create a client key and certificate signing request:

perform this step on the Docker daemon's host machine as well.

- ✓ openssl genrsa -out key.pem 4096
- ✓ openssl req -subj '/CN=client' -new -key key.pem -out client.csr
- ✓ echo extendedKeyUsage = clientAuth > extfile-client.cnf
- ✓ openssl x509 -req -days 365 -sha256 -in client.csr -CA ca.pem -CAkey ca-key.pem \
  -CAcreateserial -out cert.pem -extfile extfile-client.cnf

- ✓ chmod -v 0400 ca-key.pem key.pem server-key.pem <<= remove the write permission
- ✓ chmod -v 0444 ca.pem server-cert.pem cert.pem <<== remove the write permission

```
{
    "tls": true,
    "tlsverify": true,
    "tlscacert": "/etc/docker/ssl/ca.pem",
    "tlscert": "/etc/docker/ssl/server-cert.pem",
    "tlskey": "/etc/docker/ssl/server-key.pem"
}
```

Configuration in daemon.json in docker server

For client authentication, create a client key and certificate signing request:

perform this client end – scenario is based in windows 10

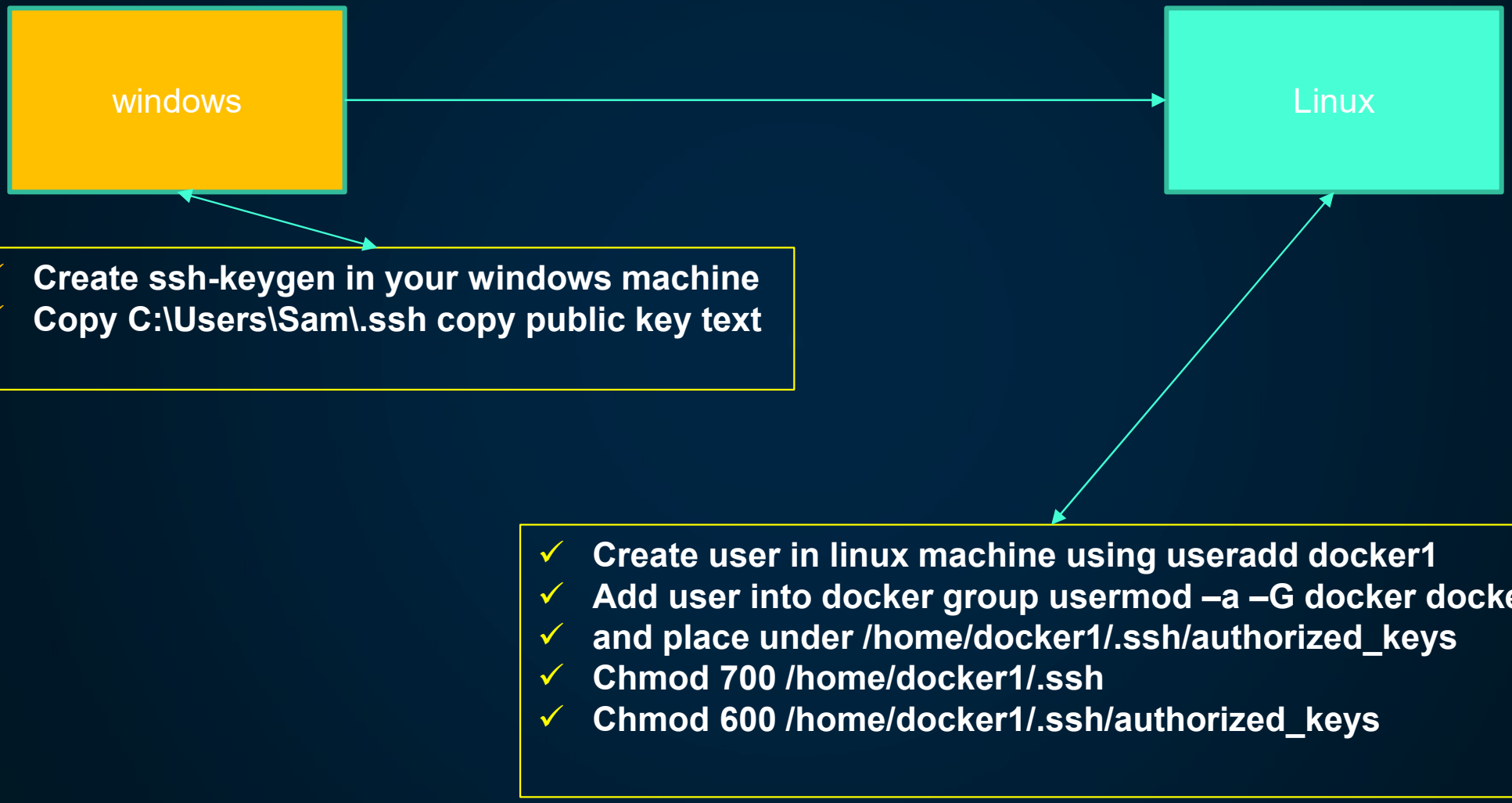
- ✓ Copy ca.pem,cert.pem,key.pem to C:\Users\Sam\.docker
- ✓ Set the environment variable in windows command prompt
- ✓ set DOCKER_HOST=tcp://192.168.0.246:2376
- ✓ set DOCKER_TLS_VERIFY=1
- ✓ Perform docker image ls command it will display the remote server images

```
C:\WINDOWS\system32>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\WINDOWS\system32>docker image ls
REPOSITORY      TAG         IMAGE ID       CREATED         SIZE
vijay/apache    v1          a28f235e8bb0   5 hours ago     474MB
vijay/nginx     v1          34b51c1dfb19   5 hours ago     162MB
newnginx        latest      079391614c6f   2 days ago      132MB
ubuntu          latest      1318b700e415   2 weeks ago     72.8MB
hello-world     latest      d1165f221234   5 months ago    13.3kB
centos          7           8652b9f0cb4c   9 months ago    204MB
```

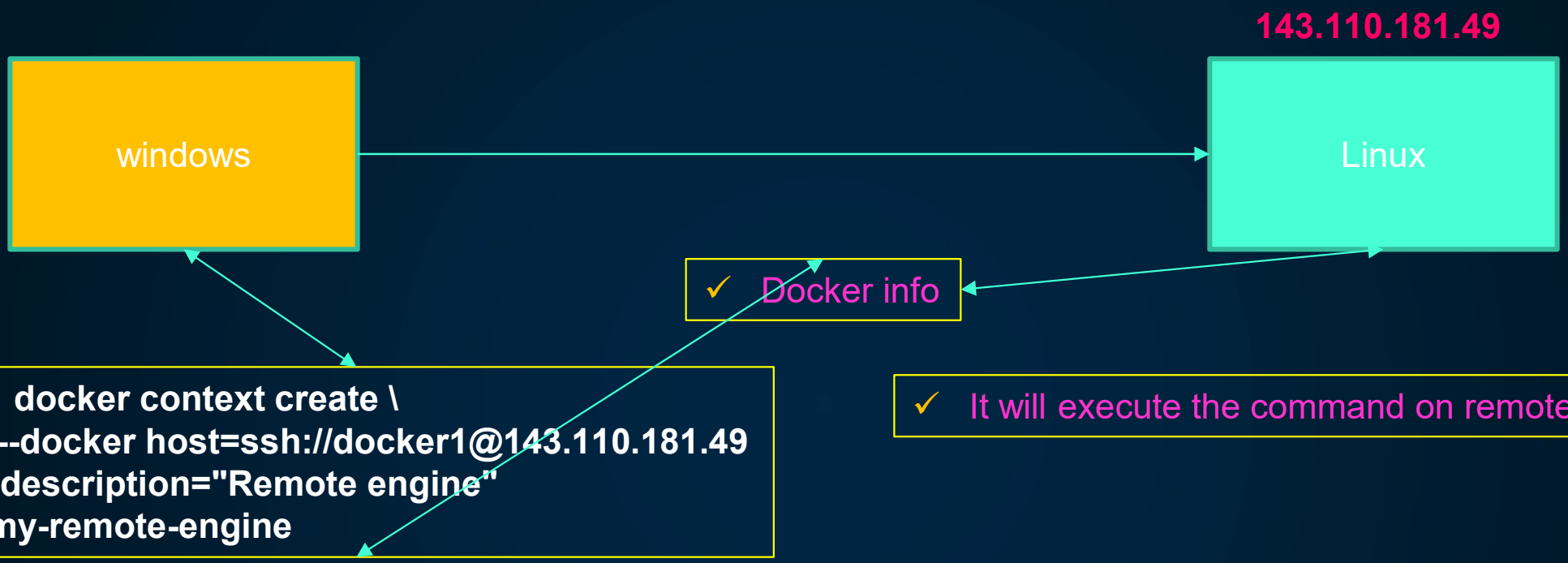
Daemon and client authentication mode

	Daemon mode	Client mode
Authenticate clients	tlsverify, tlscacert, tlscert, tlskey	
Do not authenticate clients	tls, tlscert, tlskey	
Authentication public CA pool		tls
Authentication server based on given CA		tlsverify, tlscacert
Authenticate with client certificate		tls, tlscert, tlskey
Authenticate with client certificate + authenticate server based on given CA		tlsverify, tlscacert, tlscert, tlskey

Use SSH to protect the Docker daemon socket



Use SSH to protect the Docker daemon socket



✓ export DOCKER_HOST=ssh://docker1@**143.110.181.49** **<<= if you are using linux client**

✓ Set DOCKER_HOST=ssh://docker1@**143.110.181.49 <<== if you are using the windows client**

Configure persistent storage using volume

- ✓ docker volume create myvol -d local
- ✓ Docker volume ls <<= to list the volume
- ✓ Create the container using persistent volume

- ✓ docker container run -v myvol:/vijay -itd --name mynginx2 nginx
- ✓ docker container exec -it mynginx2 /bin/bash <<= access the container verify the /vijay folder inside the container and create the file inside that directory

- ✓ Now remove the container forcefully not remove the persistent volume data and the data is still available under /var/lib/docker/myvol/_data folder

The same value can be mounted to another container and the data will be available again

- ✓ docker container run -v myvol:/data -itd --name nginx1 nginx

Configure the networking in the docker

- ✓ docker network ls <<= to list the available network inside the directory

```
root@docker:/# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
fd6bb11ff1a0   bridge    bridge    local
51b73499e336   host      host      local
44848359707f   none      null      local
```

- ✓ Three type of network bridge ,none,host

- ✓ docker network create custom-network --driver bridge --subnet 192.168.4.10/24

```
root@docker:/# docker network ls
NETWORK ID     NAME             DRIVER    SCOPE
fd6bb11ff1a0   bridge           bridge    local
ba6e2e138573   custom-network   bridge    local
51b73499e336   host             host      local
44848359707f   none             null      local
```

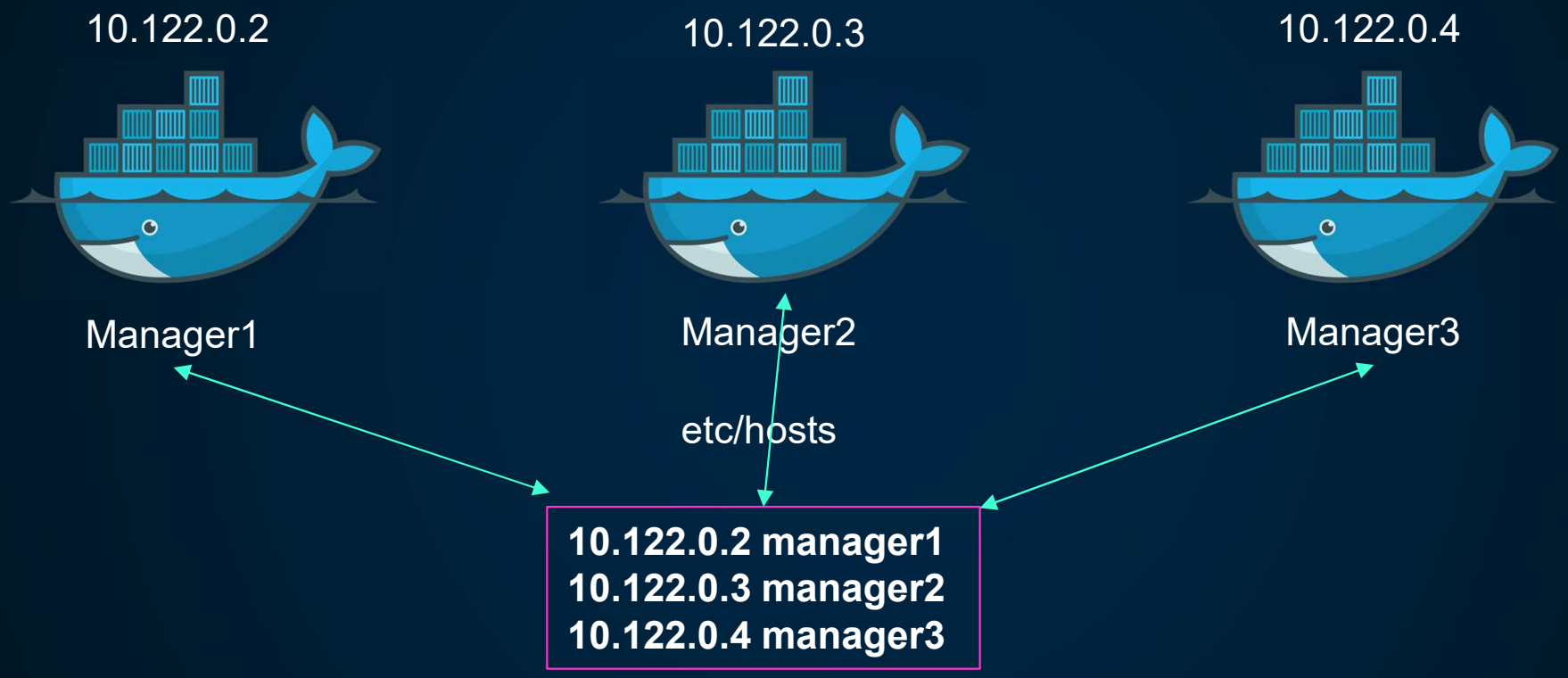
Container can be connect to different bridge

- ✓ docker network connect custom-network myredis <<= container connect to custom-network bridge that we made in previous task

```
root@docker:/# docker container inspect myredis | grep "IPAddress"
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.3",
                    "IPAddress": "172.17.0.3",
                    "IPAddress": "192.168.4.2",
```

- ✓ It will display the ip address for the both the bridge as we connect the myredis container to both the bridge network
- ✓ docker network disconnect custom-network myredis <= to disconnect the from the custom-network

Docker Swarm

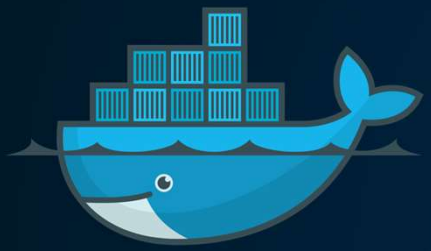


Install the docker on system using step >> https://docs.docker.com/engine/install/centos/

Systemctl enable docker.service & systemctl start docker.service

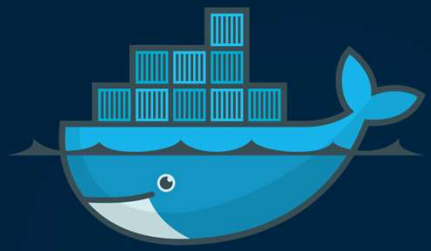
Docker Swarm

10.122.0.2



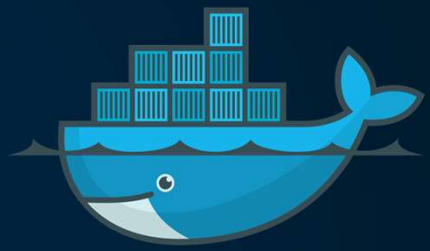
Manager1

10.122.0.3



Manager2

10.122.0.4



Manager3

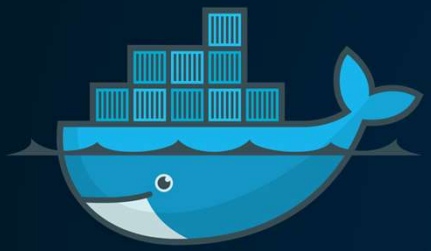
docker swarm init

```
[root@Manager1 ~]# docker swarm init --advertise-addr 10.122.0.2
Swarm initialized: current node (ytx22vrk0cvoemyie4i39h382) is now a manager.

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

    docker swarm join --token SWMTKN-1-2tpgxuy5acumjoef9ka3skjixp4rqy0v5ew6sia5b4teif4hxy-1nvmbenjjm81hik907009z1qk 10.122.0.2:2377

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

Use --advertise-addr if you have the multiple ethernet adapter active on system and you need to choose among them

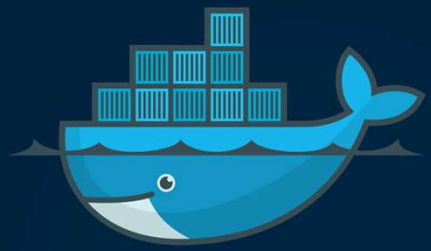
Docker Swarm

10.122.0.2



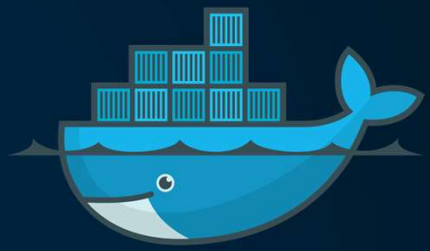
Manager1

10.122.0.3



Manager2

10.122.0.4

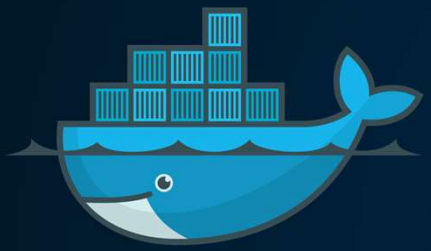


Manager3

```
[root@Manager1 ~]# docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2tpgxuy5acumjoef9ka3skjixp4rqy0v5ew6sia5b4teif4hxy-5i4xg9gnwsr2chul92ymlqcu0 10.122.0.2:2377
```

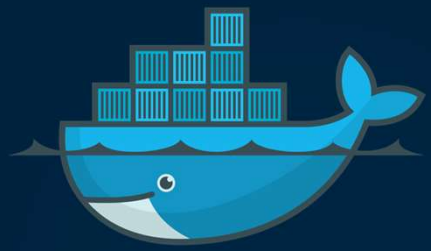
Docker Swarm

10.122.0.2



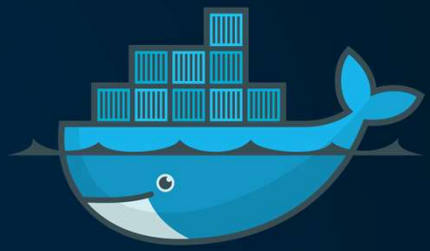
Manager1

10.122.0.3



Manager2

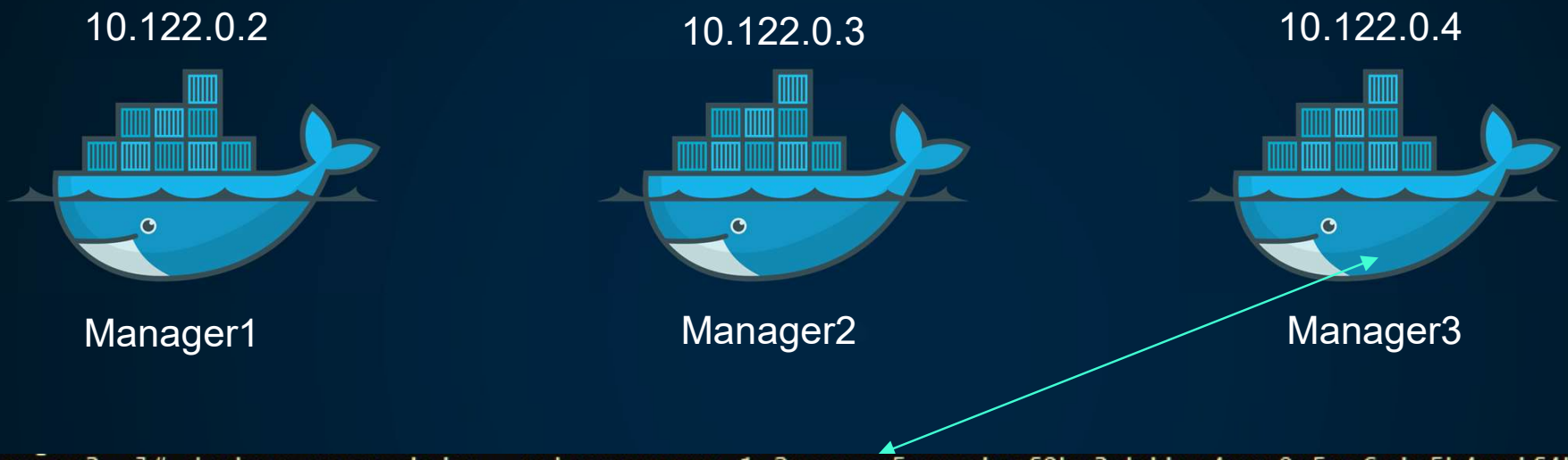
10.122.0.4



Manager3

```
[root@manager2 ~]# docker swarm join --token SWMTKN-1-2tpgxuy5acumjoef9ka3skjixp4rqy0v5ew6sia5b4teif4hxy-5i4xg9gnwsr2chu
l92ymlqcu0 10.122.0.2:2377
This node joined a swarm as a manager.
```

Docker Swarm



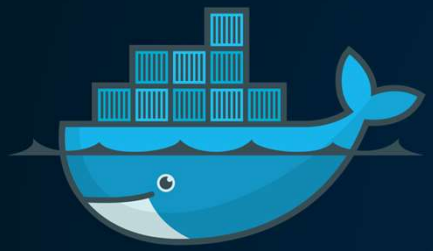

```
[root@manager3 ~]# docker swarm join --token SWMTKN-1-2tpgxuy5acumjoef9ka3skjixp4rqy0v5ew6sia5b4teif4hxy-5i4xg9gnwsr2ch
192ymlqcu0 10.122.0.2:2377
This node joined a swarm as a manager.
```

```
[root@manager3 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Leader           20.10.8
yklcsdfhi6y8k5zo061rwxksu     manager2   Ready     Active         Reachable        20.10.8
scnke0zv1lqi41gp2lmca5mwk *   manager3   Ready     Active         Reachable        20.10.8
[root@manager3 ~]#
```

all manager node listed in swarm * will display currently working node

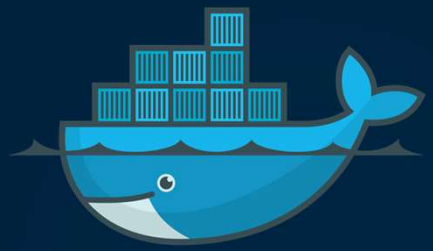
Docker Swarm -cluster

10.122.0.2



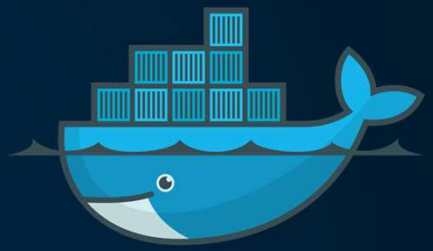
Manager1

10.122.0.3



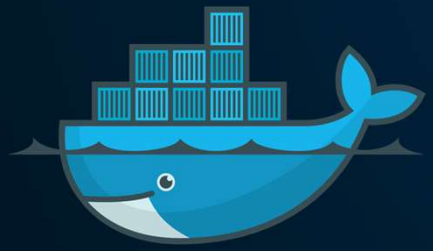
Manager2

10.122.0.4



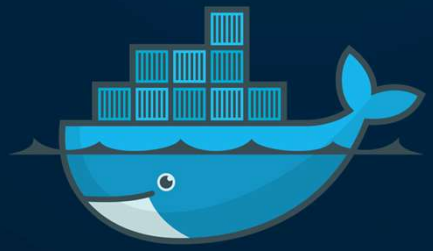
Manager3

10.122.0.5



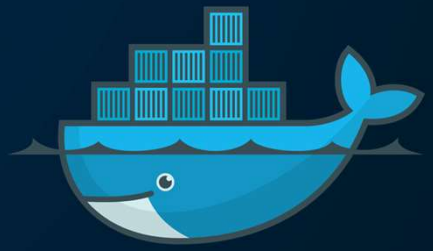
worker1

10.122.0.6



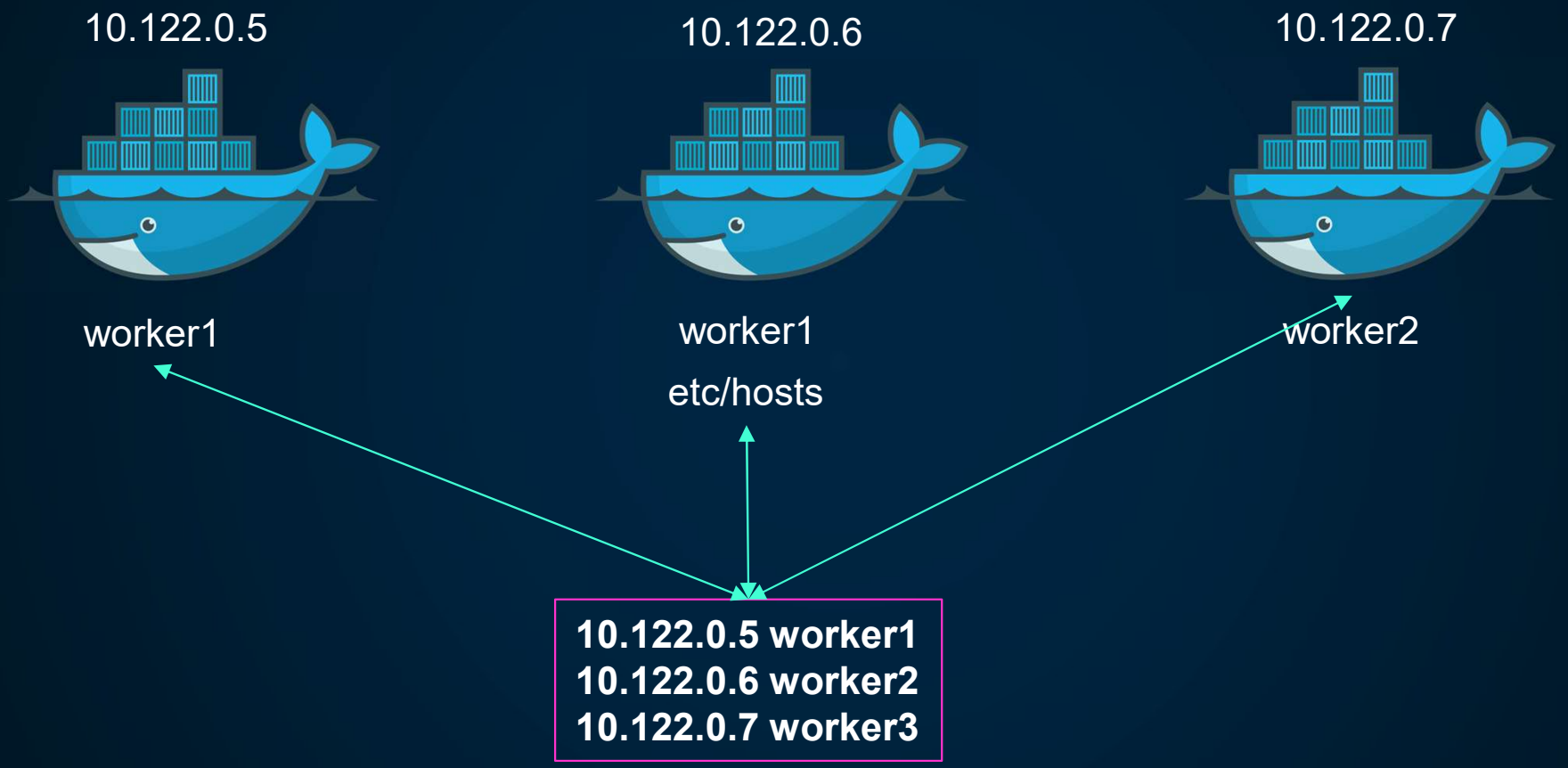
worker1

10.122.0.7

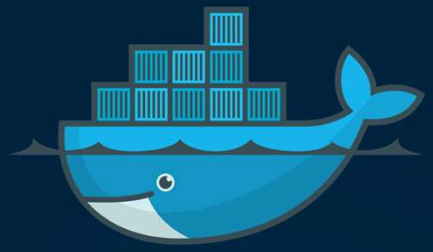


worker2

Docker Swarm worker node



Docker Swarm -cluster



Manager1
10.122.0.2

```
[root@Manager1 ~]# docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2tpgxuy5acumjoef9ka3skjixp4rqy0v5ew6sia5b4teif4h
xy-1nvmbenjjm81hik907009z1qk 10.122.0.2:2377
```

10.122.0.5

worker1

10.122.0.6

10.122.0.7

worker3

Docker Swarm status

```
root@Manager1 ~]# docker node ls
D                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
tx22vrk0cvoemyie4i39h382 *   Manager1   Ready     Active         Leader           20.10.8
klcsdfhi6y8k5zo061rwxksu     manager2   Ready     Active         Reachable        20.10.8
cnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
f8scklfxpij0fsnwoembsck4     worker1    Ready     Active                          20.10.8
u1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active                          20.10.8
uvyf12fc0blg2807r3wrakag     worker3    Ready     Active                          20.10.8
```

Leader node is actively take the decision while other two manager node manager2 and manager3 take Responsibilities if the manager1 experience any issue

Stopping manager1 node to check quoroum

Perform systemctl stop docker on the manager1

```
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
ytx22vrk0cvoemyie4i39h382     Manager1   Down      Active         Unreachable      20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
sf8scklfxpij0fsnwoembsck4     worker1    Ready     Active                          20.10.8
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active                          20.10.8
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active                          20.10.8
```

It will show the manager1 node is down and unreachable so it automatically make the manager2 to leader

Promote worker node to leader

Worker node can be promoted and demoted to work as manager from swarm command line

```
[root@manager2 ~]# docker node promote worker2
Node worker2 promoted to a manager in the swarm.
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VER
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Reachable        20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
sf8scklfxpij0fsnwoembsck4     worker1    Ready     Active                          20.10.8
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active         Reachable        20.10.8
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active                          20.10.8
[root@manager2 ~]#
[root@manager2 ~]# docker node demote worker2
Manager worker2 demoted in the swarm.
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VER
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Reachable        20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
sf8scklfxpij0fsnwoembsck4     worker1    Ready     Active                          20.10.8
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active                          20.10.8
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active                          20.10.8
```

Docker node update

--availability drain | pause | active

docker node update --availability drain worker1 << use active to reverse the scenario

```
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE V
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Reachable        20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
sf8scklfxpij0fsnwoembsck4     worker1    Ready     Drain                           20.10.8
```

--label-add <<= add label to the nodes

docker node update --label-add env worker1

docker node inspect worker1 | grep env <<= verify the lable attached to node

docker node update --label-rm env worker1

--role << convert manager to worker and worker to manager

docker node update --role manager worker2

Leaving the swarm node

--availability drain | pause | active

docker node update --availability drain worker1 << use active to reverse the scenario

```
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE V
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Reachable        20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
sf8scklfxpij0fsnwoembsck4     worker1    Ready     Drain                           20.10.8
```

Execute the docker swarm leave on worker on

```
[root@worker1 ~]# docker swarm leave
Node left the swarm.
[root@worker1 ~]#
```

Docker node ls

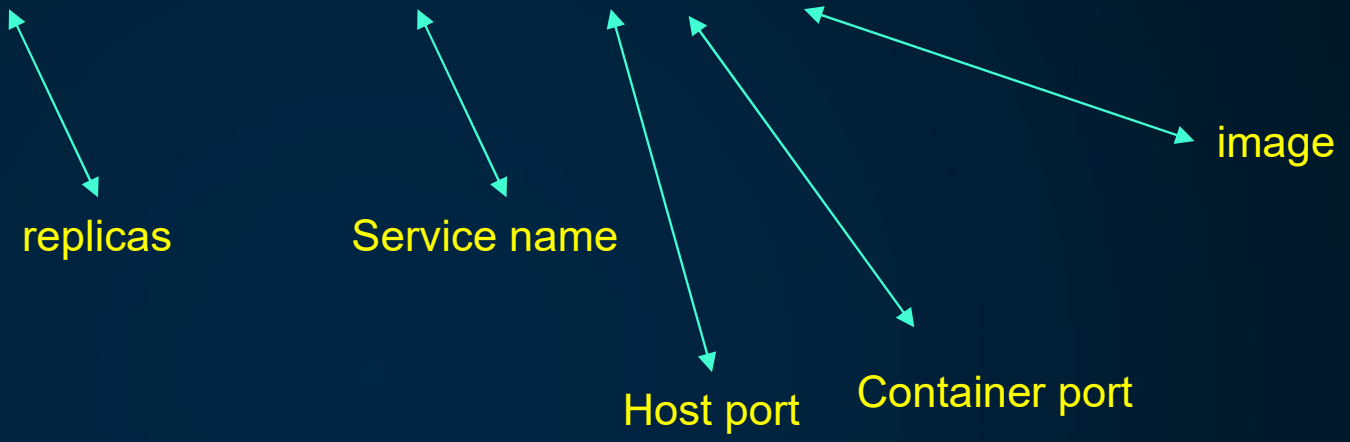
```
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VE
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Reachable        20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
sf8scklfxpij0fsnwoembsck4     worker1    Down      Drain                           20.10.8
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active                          20.10.8
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active                          20.10.8
```

docker node rm worker1

```
[root@manager2 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VE
ytx22vrk0cvoemyie4i39h382     Manager1   Ready     Active         Reachable        20.10.8
yklcsdfhi6y8k5zo061rwxksu *   manager2   Ready     Active         Leader           20.10.8
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable        20.10.8
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active                          20.10.8
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active                          20.10.8
```

Docker service create and replicas

docker service create --replicas=3 --name=myweb -p 80:80 nginx




```
[root@manager2 ~]# docker service ps myweb
ID             NAME      IMAGE          NODE      DESIRED STATE
l1nvbls6ets2   myweb.1   nginx:latest   worker3   Running
gy14abgrepjr   myweb.2   nginx:latest   worker2   Running
o5c8tndo6nbl   myweb.3   nginx:latest   manager3  Running
```

docker service update --replicas=1 myweb

Docker service rolling updates

docker service create --replicas=3 --name=myweb -p 80:80 nginx:1.16

```
[root@manager2 ~]# docker service ps myweb
ID             NAME      IMAGE          NODE       DESIRED STATE
9sup4zna7g46   myweb.1   nginx:1.16     manager2   Running
mgfvbz8kwugf   myweb.2   nginx:1.16     Manager1   Running
yxddwvd4ay7s   myweb.3   nginx:1.16     worker3    Running
```

docker image pull nginx:1.17

docker service update --replicas=3 --image=nginx:1.17 myweb

```
[root@manager2 ~]# docker service ps myweb
ID             NAME          IMAGE          NODE       DESIRED STATE
iv2jyzzoa7oa   myweb.1       nginx:1.17     manager3   Running
9sup4zna7g46    \_ myweb.1   nginx:1.16     manager2   Shutdown
n4cw9onajjb2   myweb.2       nginx:1.17     worker2    Running
mgfvbz8kwugf    \_ myweb.2   nginx:1.16     Manager1   Shutdown
rxvea0p09jcq   myweb.3       nginx:1.17     worker3    Running
yxddwvd4ay7s    \_ myweb.3   nginx:1.16     worker3    Shutdown
```

docker service update --rollback myweb

Labels and constrains

docker node update **--label-add type=cpu-optimized** worker1

Assign label to worker node server as cpu-optimized node

docker service create --constraint=node.labels.type==cpu-optimized --name mytest nginx

Service automatically mapped with the worker1

```
[root@manager2 ~]# docker service ps mytest
ID             NAME       IMAGE          NODE      DESIRED STATE
tfrespma9qvq   mytest.1   nginx:latest   worker1   Running
```

Hence placement of the service can be configured according to labels that you give to worker node

Docker node

docker node ls <<= display all the docker node including manager and worker

```
[root@Manager1 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
ytx22vrk0cvoemyie4i39h382 *   Manager1   Ready     Active         Leader
yklcsdfhi6y8k5zo061rwxksu     manager2   Ready     Active         Reachable
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable
pqtlsqk7102573mn2cmynjt1l     worker1    Ready     Active
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active
```

docker node promote worker1 <<= to promote the worker node to manager node

```
[root@Manager1 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
ytx22vrk0cvoemyie4i39h382 *   Manager1   Ready     Active         Leader
yklcsdfhi6y8k5zo061rwxksu     manager2   Ready     Active         Reachable
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable
pqtlsqk7102573mn2cmynjt1l     worker1    Ready     Active         Reachable
```

Docker node

docker node demote worker1 <<= demote the worker node from manager to worker

```
[root@Manager1 ~]# docker node demote worker1
Manager worker1 demoted in the swarm.
[root@Manager1 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
ytx22vrk0cvoemyie4i39h382 *   Manager1   Ready     Active         Leader
yklcsdfhi6y8k5zo061rwxksu     manager2   Ready     Active         Reachable
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable
oqtlsqk7102573mn2cmynjt1l     worker1    Ready     Active
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active
```

docker node inspect worker1 <<= to get the details information about the node

docker node ps worker1 | Manager 1<<= to find out the resources under worker or manager node

Docker node remove procedure

docker node update --availability drain worker1 <<= to drain the particular node

```
[root@Manager1 ~]# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
ytx22vrk0cvoemyie4i39h382 *   Manager1   Ready     Active         Leader
yklcsdfhi6y8k5zo061rwxksu     manager2   Ready     Active         Reachable
scnke0zv1lqi41gp2lmca5mwk     manager3   Ready     Active         Reachable
bqtlsqk7102573mn2cmynjt1l     worker1    Ready     Drain
au1ricg6w9qdma5i2ma9fnbnz     worker2    Ready     Active
7uvyf12fc0blg2807r3wrakag     worker3    Ready     Active
```

docker swarm leave from the worker node--availability drain worker1 <<= to drain the particular node

```
[root@worker1 ~]# docker swarm leave
Node left the swarm.
[root@worker1 ~]#
```

Docker rm | remove worker1 << = remove the worker node –f can be used for forceful action

```
[root@Manager1 ~]# docker node rm worker1
worker1
```

Docker swarm command

docker swarm join-token worker <== to create the token to join the worker node

Execute these on worker node

docker swarm join --token SWMTKN-1-2tpgxuy5acumjoef9ka3skjixp4rqy0v5ew6sia5b4teif4hxy-1nvmbenjjm81hik907009z1qk 10.122.0.2:2377

docker swarm update –autolock <== manager node locking mechanism

```
[root@Manager1 ~]# docker swarm update --autolock
Swarm updated.
To unlock a swarm manager after it restarts, run the `docker swarm unlock`
command and provide the following key:

    SWMKEY-1-s0cKESj2wxmbGmDQ2bt5AwbIX4F5H/ES2ooLrrffBL4
```

Store the key in the safe place

Docker swarm command

If the manager2 docker service restart then node will not participate again in swarm
Except we provide the unlock key to the manager node

docker swarm unlock to re-join the manager node

```
[root@manager2 ~]# docker swarm unlock
Please enter unlock key:
[root@manager2 ~]#
```

Docker system info

docker system info <<= display the information about the docker environment

Default logging driver is json-file ,
default runtime is runc
Default root dir /var/lib/docker

docker system df <<= display reclaimable size occupied by images

```
[root@Manager1 ~]# docker system df
TYPE                TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images              2         0         264.5MB   264.5MB (100%)
Containers          0         0         0B        0B
Local Volumes       0         0         0B        0B
Build Cache         0         0         0B        0B
```

Docker system info

docker system events <<= display the information about the docker environment

docker system events --since 2021-08-17

docker system events --filter

docker system events –until

docker system events –format

docker system prune –a remove the all dangling images

docker system prune –volumes <== remove the all unused volume

Docker backup and disaster recovery

Back up the swarm

- ✓ Docker manager nodes store the swarm state and manager logs in the /var/lib/docker/swarm/ directory. This data includes the keys used to encrypt the Raft logs. Without these keys, you cannot restore the swarm.
- ✓ If the swarm has auto-lock enabled, you need the unlock key to restore the swarm from backup. Retrieve the unlock key if necessary and store it in a safe location
- ✓ Stop Docker on the manager before backing up the data, so that no data is being changed during the backup.
- ✓ Back up the entire /var/lib/docker/swarm directory.
- ✓ Swarm backup consists Raft keys,membership,services,networks overlay,config,secrets swarm unlock key must be saved at different place

Docker backup and disaster recovery

Restore the swarm

- ✓ Restore the /var/lib/docker/swarm directory with the contents of the backup.
- ✓ Start Docker on the new node. Unlock the swarm if necessary.
- ✓ Re-initialize the swarm using the following command
- ✓ docker swarm init --force-new-cluster
- ✓ You must use the same IP as the node from which you made the backup.

Docker backup and disaster recovery

Docker UCP backup

- ✓ Backup should be restore on the same version of cluster.
- ✓ More than one backup at a time is not supported
- ✓ Ucp does not include swarm workloads
- ✓ docker swarm init --force-new-cluster

Docker backup and disaster recovery

Docker UCP backup contents

Data	Baked Up
Configuration	yes
Access control	yes
Certificate and keys	yes
Metrics data	yes
Organization	Yes
Volume	Yes
Overlay network	No
Config,secrets	No
Service	No

Data	Baked Up
ucp-metrics-data:	no
ucp-node-certs	no
Routing mesh settings	no
Interlock L7 ingress configuration	no
Kubernetes declarative objects Pods, deployments, replicasets, configurations	Yes

DTR=MSR DTR [Digital trusted Repository]

Feature

- ✓ Image and job management
- ✓ Availibility
- ✓ Efficiencey
- ✓ Built-in access-control
- ✓ Security scanning
- ✓ Image signing

Image store platfom

- ✓ Amazon s3
- ✓ Nfs
- ✓ Cleversafe
- ✓ Googlecloud storage
- ✓ Openstack
- ✓ Microsoft azure

https://docs.mirantis.com/msr/2.9/install/install-online.html

DTR=MSR DTR [Digital trusted Repository]

- ✓ DTR installed on worker node and for the high availabilities one more instance can be distributed across the node

Data	Baked Up
Configuration	yes
Repository metadata	yes
Access control to repos and images	yes
Notary data	yes
Scan results	Yes
Certification and keys	Yes
Image content	No
User,orgs.team	No
Vulnerability	No

Volume

- ✓ Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine.

- ✓ Volumes can be more safely shared among multiple containers.
- ✓ Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.

Differences between -v and --mount behavior

- ✓ As opposed to bind mounts, all options for volumes are available for both --mount and -v flags.

- ✓ When using volumes with services, only --mount is supported.

Domain 2: Image Creation, Management, and Registry

Domain 2: Image Creation, Management, and Registry

- ✓ The docker build command builds an image from a Dockerfile and a context
- ✓ Do not use root directory for build context
- ✓ To increase the build's performance, exclude files and directories by adding a .dockerignore file to the context directory

Rule	Behavior
`# comment`	Ignored.
`*/temp*`	Exclude files and directories whose names start with `temp` in any immediate subdirectory of the root. For example, the plain file `/somedir/temporary.txt` is excluded, as is the directory `/somedir/temp`.
`*/*/temp*`	Exclude files and directories starting with `temp` from any subdirectory that is two levels below the root. For example, `/somedir/subdir/temporary.txt` is excluded.
`temp?`	Exclude files and directories in the root directory whose names are a one-character extension of `temp`. For example, `/tempa` and `/tempb` are excluded.

Domain 2: Image Creation, Management, and Registry

- ✓ Use –f to specify the docker file location and user –t to assign the tag for docker file
- ✓ The Docker daemon runs the instructions in the Dockerfile
- ✓ . A Dockerfile must begin with a FROM instruction.

Docker Enterprise Edition installation

Visit the Url and register using business email https://www.mirantis.com/download/mirantis-cloud-native-platform/mirantis-kubernetes-engine/

Download the launchpad for client if you are trying to install the from windows to linux machine hosted on cloud platform


```
Chmod +x launchpad
./launchpad register  << fill  the necessary details on prompt
./launchpad.exe init  << initialize the configuration file
```

Docker Enterprise Edition installation

```
apiVersion: launchpad.mirantis.com/mke/v1.3
kind: mke
metadata:
  name: my-mke
spec:
  mke:
    version: 3.4.5
    adminUsername: admin
    adminPassword: admin123
    installFlags:
      - --default-node-orchestrator=kubernetes
      - --pod-cidr 20.0.0.0/16
  hosts:
  - role: manager
    ssh:
      address: 10.122.0.2
      keyPath: ~/.ssh/id_rsa
      user: root
    privateInterface: eth1
  - role: worker
    ssh:
      address: 10.122.0.3
      keyPath: ~/.ssh/id_rsa
      user: root
    privateInterface: eth1
```

./launchpad init > launchpad.yaml

It will only generate the file but you need to make changes according to your environment

./Launchpad apply

Script will fail to join the worker node if your machine are not able to communicate with other vm hence you can manually setup the swarm cluster

Docker swarm leave –force << run in manger node
Docker swarm init --advertise-addr 10.122.0.2
Docker swarm join-token worker

Join the worker node using generated token

Then re- run ./launchpad apply

Docker Secrets –only available in swarm

Password
SSH Private Key
SSL Certificate
important data such as the name of a database or internal server
Generic strings or binary content (up to 500 kb in size)

What it can store

Secrets are encrypted during transits and at the rest in docker swarm

A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

Secrets can be rotate by using update service command

Location of secret in windows container C:\ProgramData\Docker\secrets
Location of decrypted Secret in Linux containers : /run/secrets/<secret_name>

Docker Content Trust

- ✓ Content trust gives you the ability to verify both the integrity and the publisher of all the data received from a registry over any channel

- ✓ signatures allow client-side or runtime verification of the integrity and publisher of specific image tags.
- ✓ Trust for an image tag is managed through the use of signing keys
  A key set consists of the following classes of keys:

- ✓ an offline key that is the root of DCT for an image tag
- ✓ repository or tagging keys that sign tags
- ✓ server-managed keys such as the timestamp key, which provides freshness security guarantees for your repository
- ✓ Docker CLI we can sign and push a container image with the $ docker trust command
- ✓ To sign a Docker Image you will need a delegation key pair. These keys can be generated locally using $ docker trust key generate or generated by a certificate authority

Domain 6: Storage and Volumes

- ✓ Identify the correct graph drivers to uses with various operating systems
- ✓ Describe and demonstrate how to configure devicemapper
- ✓ Compare and contrast object and block storage and when they should be used
- ✓ Describe how an application is composed of layers and where these layers reside on the filesystem
- ✓ Describe how volumes are used with Docker for persistent storage
- ✓ Identify the steps you would take to clean up unused images on a filesystem, also on DTR. (image prune, system prune and from DTR)
- ✓ Demonstrate how storage can be used across cluster nodes, ex.
- ✓ Describe how to provision persistent storage to a Kubernetes pod using persistentVolumes
- ✓ Describe the relationship between container storage interface drivers, storageClass, persistentVolumeClaim and volume objects in Kubernetes

Domain 6: Storage and Volumes

✓ Identify the correct graph drivers to uses with various operating systems

Feature	overlay2	fuse-overlayfs	Btrfs/ZFS	aufs	Vfs	Devicemapper
Use Case/ feature	All platform	Rootless docker	snapshots	Support older os and Docker versiion	Testing purpose	Direct-lvm for production purpose
File Storage	Yes			Yes		
Block Storage			Yes			
File-system backing	Xfs ,ext4	Any	Btrfs/zfs	Xfs ,ext4	Any	Direct-lvm
Flaws			lot of memory.			
Stability	Yes			Yes		yes

Domain 6: Storage and Volumes

OverlayFS storage driver

- ✓ OverlayFS is a modern *union filesystem*
- ✓ The overlay driver only works with two layers.
- ✓ The overlay2 driver natively supports up to 128 lower OverlayFS layers
- ✓ each image layer is implemented as its own directory under /var/lib/docker/overlay

Device Mapper storage

- ✓ Device Mapper is a kernel-based framework
- ✓ device mapper requires the lvm2 and device-mapper-persistent-data packages to be install
- ✓ Loop-lvm is testing mode while direct-lvm is production mode
- ✓ Use dm.directlvm_device in dameon.json file to configure the driver

Domain 6: Storage and Volumes

Btrfs storage driver

- ✓ Btrfs is a next generation copy-on-write filesystem
- ✓ btrfs requires a dedicated block storage device such as a physical disk
- ✓ Btrfs provide the snapshot feature

ZFS storage driver

- ✓ ZFS is a next generation filesystem that supports many advanced storage technologies such as volume management, snapshots, checksumming, compression and deduplication, replication and more.
- ✓ ZFS is not supported on Docker EE or CS-Engine, or any other Linux platforms
- ✓ Changing the storage driver makes any containers you have already created inaccessible on the local system

Port Information

- ✓ Port 7946 TCP/UDP for container network discovery
- ✓ Port 4789 UDP for the container ingress network
- ✓ TCP port 2377 for cluster management communications

Differences between user-defined bridges and the default bridge

user-defined bridges

- ✓ User-defined bridges provide automatic DNS resolution between containers
- ✓ User-defined bridges provide better isolation.
- ✓ Containers can be attached and detached from user-defined networks on the fly.
- ✓ Each user-defined network creates a configurable bridge.

default bridge

- ✓ Linked containers on the default bridge network share environment variables.
- ✓ To configure the default bridge network, you specify options in daemon.json