Docker Compose :

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.

Prerequisites:

Docker Compose relies on Docker Engine for any meaningful work, so make sure you have Docker Engine installed either locally or remote, depending on your setup.

On desktop systems like Docker Desktop for Mac and Windows, Docker Compose is included as part of those desktop installs.

On Linux systems, first install the Docker Engine for your OS as described on the Get Docker page, then come back here for instructions on installing Compose on Linux systems.

Run this command to download the current stable release of Docker Compose:

```
#curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Apply executable permissions to the binary:

```
#chmod +x /usr/local/bin/docker-compose
```

Test the installation:

```
#docker-compose --version

#mkdir dockercomposefile
#touch dockercomposefile/docker-compose.yml
#vim dockercomposefile/docker-compose.yml
services :


  web :
    image : nginx
    ports :
    - 8080:80
  database :
    image : redis
:wq!
```

NOTE : We need to check the validity of compose file using the below command

```
#cd dockercomposefile
#docker-compose config
```

NOTE : Use the below link to understand the compatiable compose file versions for docker run time :

https://docs.docker.com/compose/compose-file/

```
#docker-compose up -d
```

up --> to run the yml file and create contianers & applications
-d --> Detatch Mode

```
#docker container ls

#docker-compose down

#docker-compose up -d
```

Scaling up Services :

```
#docker-compose up -d --scale database=5
#docker container ls
```

Scaling down Services :

```
#docker-compose up -d --scale database=1
#docker container ls

#docker-compose down
#docker container ls
```

Docker Volumes :

Docker volumes are a widely used and useful tool for ensuring data persistence while working in containers. Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container.

The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.

A container's writable layer is tightly coupled to the host machine where the container is running. The data cannot be easily moveable somewhere else.

Writing into a container's writable layer requires a storage driver to manage the filesystem.

```
#docker volume ls ( Lists all the available volumes )
#docker volume create myvol1 ( Creating new volume )
#docker volume ls
#docker volume inspect myvol1 ( Displaying info of a specific volume )
#docker volume rm myvol1 ( Deleting volume )
#docker volume prune ( Deleting all unused volumes )
```

Docker Swarm :

Each node of a Docker Swarm is a Docker daemon, and all Docker daemons interact using the Docker API. Each container within the Swarm can be deployed and accessed by nodes of the same cluster.

Features of Docker Swarm :
Some of the most essential features of Docker Swarm are:

Decentralized access: Swarm makes it very easy for teams to access and manage the environment
High security: Any communication between the manager and client nodes within the Swarm is highly secure
Autoload balancing: There is autoload balancing within your environment, and you can script that into how you write out and structure the Swarm environment
High scalability: Load balancing converts the Swarm environment into a highly scalable infrastructure
Roll-back a task: Swarm allows you to roll back environments to previous safe environments

Swarm Mode Key Concepts :
Service and Tasks :

- Docker containers are launched using services.
- Services can be deployed in two different ways - global and replicated.
- Global services are responsible for monitoring containers that want to run on a Swarm node. In contrast, replicated services specify the number of identical tasks that a developer requires on the host machine.
- Services enable developers to scale their applications.
- Before deploying a service in Swarm, the developer should implement at least a single node.
- Services can be used and accessed by any node of the same cluster.
- A service is a description of a task, whereas a task performs the work.
- Docker helps a developer in creating services, which can start tasks. However, when a task is assigned to a node, the same task cannot be attributed to another node.

Node :

- A Swarm node is an instance of the Docker engine.
- It is possible to run multiple nodes on a single server. But in production deployments, nodes are distributed across various devices.

How Does Docker Swarm Work?
In Swarm, containers are launched using services. A service is a group of containers of the same image that enables the scaling of applications. Before you can deploy a service in Docker Swarm, you must have at least one node deployed.

There are two types of nodes in Docker Swarm:

Manager node : Maintains cluster management tasks
Worker node : Receives and executes tasks from the manager node

Pre-Requisites :
1)Docker 1.13 or Higher
2)DOcker Machine need to be installed ondocker host

Steps to follow :
1)Ensure docker is installed on Based OS
2)Ensure Docker Machine utility is installed
3)Ensure Virtual box hypervisor is installed
4)Create one machine as worker node and others as worker nodes

Installing Docker machine on DOcker host :
#base=https://github.com/docker/machine/releases/download/v0.16.0 && curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine && sudo mv /tmp/docker-machine /usr/local/bin/docker-machine && chmod +x /usr/local/bin/docker-machine
#docker-machine -v **(Displays the docker machine)**

Installing Virtual Box on Base OS :

#yum install wget vim –y **(Installing wget & Vim softwares)**
#wget https://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo -P /etc/yum.repos.d/ **(Configuring Virtual Box Repository)**

#yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.**rpm (Configuring EPEL Repository)**

#yum update

#yum install binutils kernel-devel kernel-headers libgomp make patch gcc glibc-headers glibc-devel dkms -y

#Yum install VirtualBox-6.1 **(Installing Virtual Box)**
#which virtualbox
#reboot

#/sbin/vboxconfig **(Stopping & STarting the Virtual Box Services)**
#systemctl status vboxconfig **(Verify Virtual box Service is Running)**

Creating Docker Machines :

```
#docker-machine create --driver virtualbox  dm1
```
#docker-machine ls **(Lists all the docker machines created)**
#docker-machine ip <machinename> **(Displays the IP of docker machine)**

```
                                dm1
#docker-machine create --driver virtualbox  dw1
#docker-machine create --driver virtualbox  dw2
#docker-machine create --driver virtualbox  dm3
#docker-machine ls
```

Connecting to Docker different terminals :

```
#docker-machine ssh dm1
#docker-machine ssh dw1
#docker-machine ssh dw2
#docker-machine ssh dw3
```