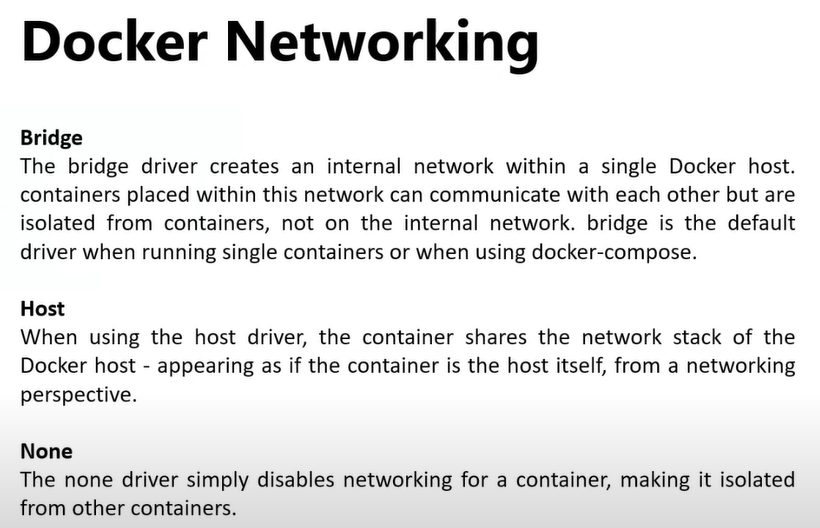
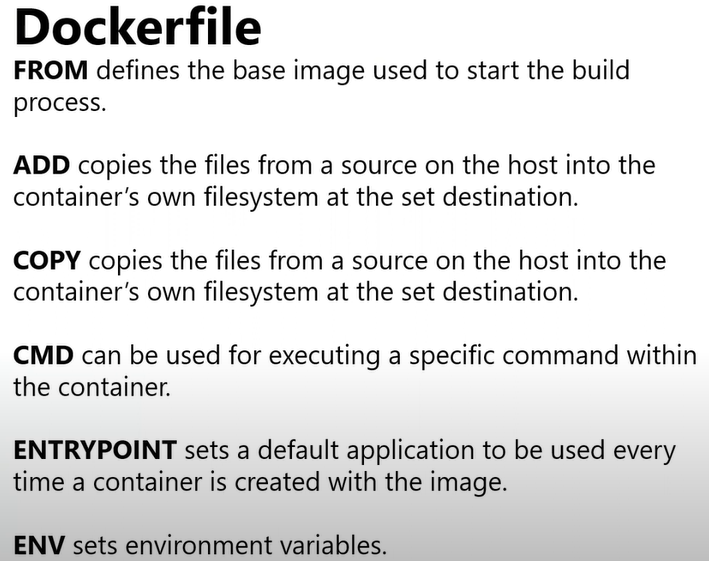
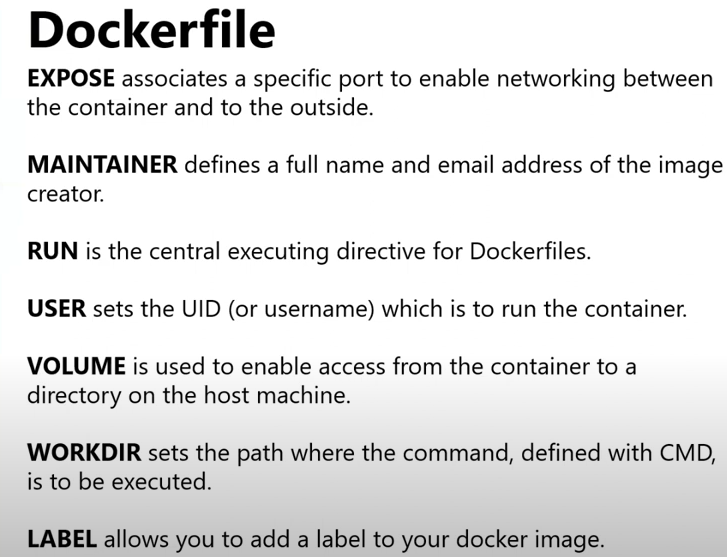
**DOCKER  
Topics:**

1. Docker basic commands
2. Docker Networking
3. Dockerfile
4. Docker volume
5. Docker compose
6. Docker container troubleshooting
7. Docker restart policy
8. Docker swarm & Features

* **Docker** is a software platform for building applications based on containers of small and lightweight execution environments that make shared use of the operating system kernel but otherwise run in isolation from one another.
* We can create n no of containers by using the single image
* Containers : simply known as tiny virtual machine with (app/webserver) with minimal installation
* Docker images : which shows the already downloaded images
* Docker history imagename: to see the layers and commands used by the image
* Docker pull imagename : which pulls the container ISO image from the docker hub
* Docker ps : which displays only running containers
* Docker ps -a : which displays the running & stopped containers as well
* Docker stop containerid : inorder to stop the container
* Docker start containerid : inorder to start the container
* Docker rmi imagename:tag ; inorder to remove the image along with tag as specified
* Docker –version : to get the current docker version.
* Docker login : to login to docker (private repo or to docker hub)
* Docker logout: to logout to docker (private repo or to docker hub)
* Docker search imagename : to search the images with in the docker hub or from repo’s
* **Docker inspect image imagename** : in order to inspect the image we can use parameters in place of image.
* Docker create imagename : to create a container using image
* Docker create –name d01 imagename : to create a container by using image and custom name as d01
* **Docker run -d –name d01 imagename** : it creates and run (without login) detach mode (d)
* Docker kill containername: when its in hung state forceful shutdown the container
* **Docker exec -it containername /bin/bash** : it logins to the container and opens the root bash terminal
* Docker rm containername : to remove the container (if running u can use –force)
* Docker rm $(docker ps -a -q) : its the command to remove all containers on it
* Docker rmi $(docker images -q) : its a command to remove all images on it
* Docker run –name some\_name image : it creates and enter into the container some times it might go into loop where no escape sequence also wont work so that time : we need to open other console and need to kill or stop the container so thet it will get exited in the previous console.
* -d uses detach mode so that we wont enter into the container
* The Docker daemon performs dynamic subnetting and IP address allocation for containers.



* Docker network ls : to see the docker networking
* Docker network ls –no-trunc : it displays the full network id’s
* Docker network ls --filter driver=bridge : it displays all the bridged network id’s list
* Docker network create newnetworkname : in order to create a network
* Docker network rm my-network : To remove the network named 'my-network'
* Docker network rm 3695c422697f my-network : to remove multiple networks at a time
* Docker run -d –name d01 -p 8080:80 imagename : which creates and run the container using d01 name from the given image and it runs on port 8080.
* Docker run -d –name d02 –net host imagename : it uses the host network to create and to run the container  
  It can be accessible through the direct ip of the docker host.
* Docker run -d –name d02 –net none imagename : it cannot communicate internally and externally  
  Bride network : its used to differentiate the applications based on the port numbers provided and accessible using the docker host ip

**Dockerfile**  
  


* In ADD the source can be URL as well where it doesnt support in the copy command.
* In ADD cmd it will extract the (zip/tar) and places at the destination as this function won't wrk in copy cmd.
* Docker build -t username/somename . : which takes the docker file from the current directory and builds the docker image with it.
* Docker run -dt –name somename -p 8081:80 imagename : which is used to create a container with the existing image set by docker file
* Docker push imagename : Make sure you must be logged in and you can push the image to dockerhub
* Docker tag oldname newname:tagname ( to assign tag to the existing image)
* Entrypoint should always run foreground then container will start if it runs background container stops

**DOCKERFILE**

**From ubuntu:18.04**

**Label maintainer=”RT”**

**Run apt-get update && apt-get -y install nginx**

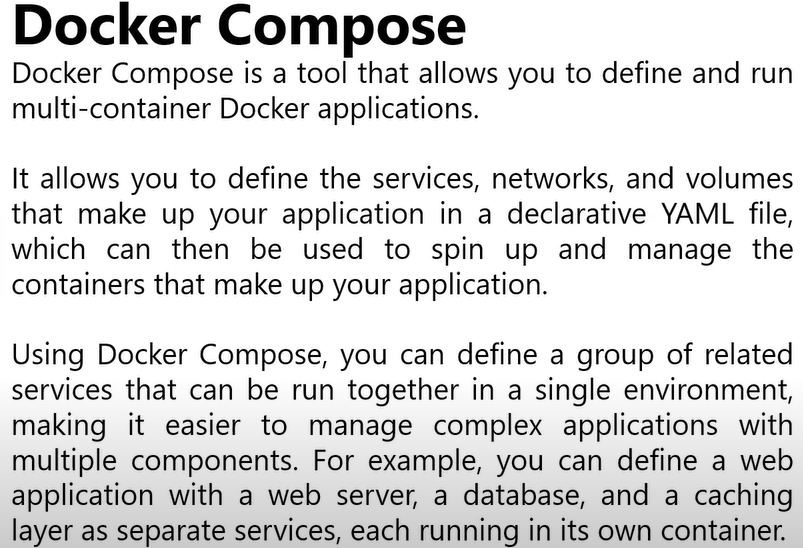
**Expose 80**

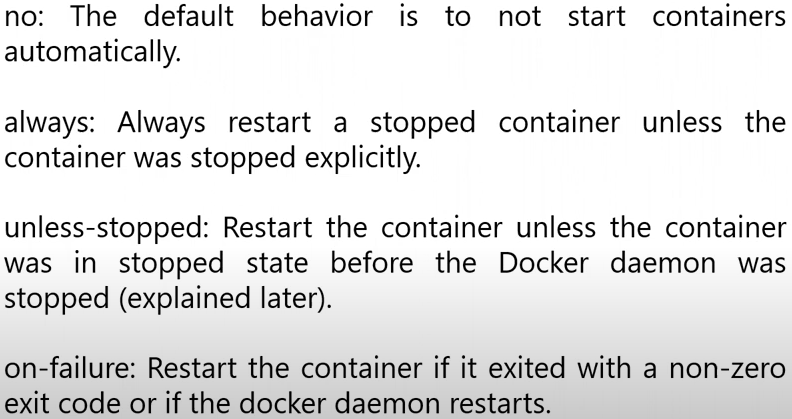
**Entrypoint [“/user/sbin/name”]**

**Cmd [“-D”,”FOREGROUND”]**

**Copy index.html /var/www/html/index.html**

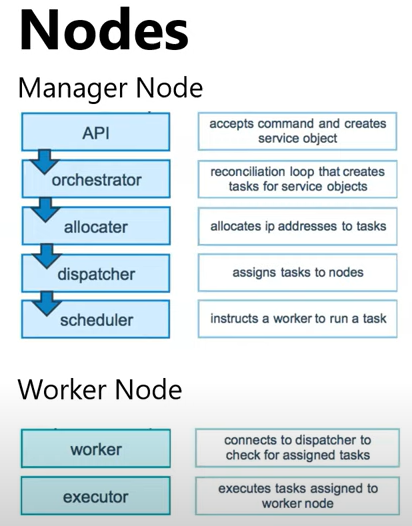
**Volume /var/www/html**  
  
 **Docker Volume**

* Docker volume : it allows the writable layer to the container which we may loose after restarting the container in order to not to loose the data we use persistent space called **docker volumes**
* Docker volume ls : to see the volumes with in the docker
* Docker volume inspect volumeid : it inspects the detailed info about the docker volume and where it is located
* Cd **var/lib/docker/volumes/ :** its the default path for the docker volumes
* Docker volume rm volumeid : to delete the specific volume individually
* Docker volume prune : its function is to delete all the left over unused volumes with it.
* Docker volume create somename : its used to create the docker volume
* Docker run -it –name=d02 –mount source=somename,destination=/data centos : its a command used to run the container and attached the already created volume from the cetos image.
* Main use of docker volumes is for the **persistence data.**
* Docker run -it –volumes-from existingvolumename –name newname centos /bin/bash : to create a container using the existing volume commands
* **Docker run -it –name newname -v testvol:/data centos** : its much easier way to create a container using volumename as testvol
* Docker run -it –name newname -v “$(pwd)”:/data1 centos : Here it creates a container by using the present working directory as a volume  
   **Docker Compose**   
  
* Mostly docker compose wont be installed on the machine we need to install externally  
  Sudo apt install docker-compose ( to install docker compose on the ubuntu machine)
* Docker-compose.yaml (we have to write in the yaml format and the file name should be the same only )  
  If the file name is different then in that case we have to use -f and followed by the file name
* **Docker-compose-up** (it runs the docker-compose.yaml file in the current directory and make the container up and running )
* Control c to stop all the containers   
    
   **Docker container Troubleshooting**
* Docker logs containername : it gives the all logs of that container
* Docker exec-it containernmae cmd : it executs the commands with the container and provide o/p
* Docker rename oldcontainername newcontainername  
  Most of the cases logs files stored in container are **/var/logs**Problem : 502 error code : Bad gateway  
  Step1:check the running container status.  
  Step2: Need to investigate each and every logs of the containers

**Docker Restart policy**

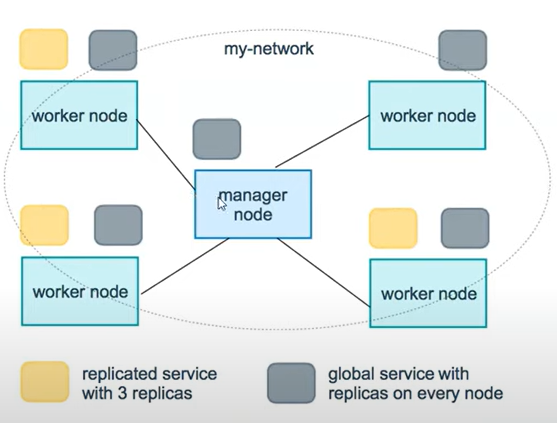
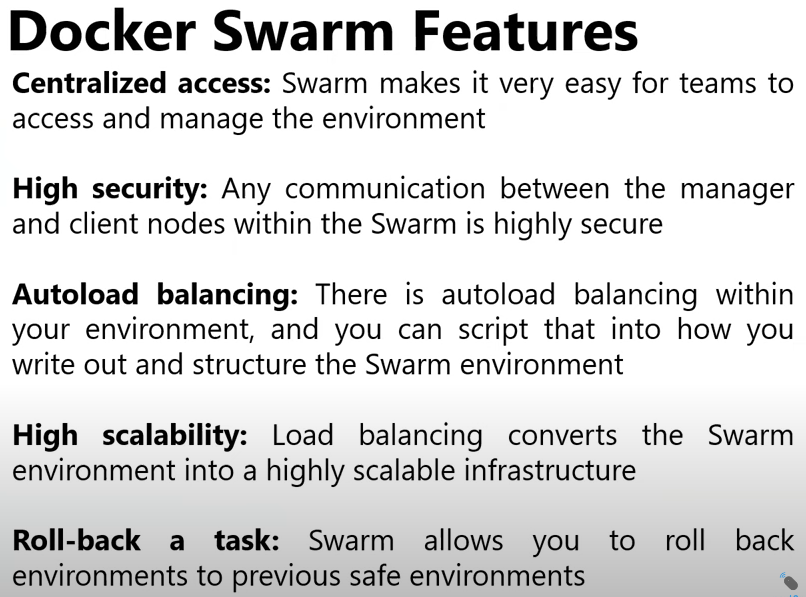
* It's a self healing mechanism in both docker and on kubernetes container
* By default restart policy is set to **NO**
* **Docker run –name always-policy –restart always alpine sleep 10**
* Docker run –name unless-stopped-policy –restart unless-stopped alpine sleep 10
* Docker run –name unless-stopped-policy –restart on -failure alpine sleep 10
* Always: this won't work for manual stop command.

**Docker Swarm**

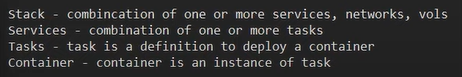
* Docker swarm is similar to that of the kubernetis   
  Nodes : (Manager Node & Worker Node)
* Docker runs on individual nodes with swarm we can achieve clustering
* Container were called **services or Tasks**  
  
* Stacks: In Docker Swarm, a stack is a way to define and manage a group of services that are deployed together as a single unit. It allows you to define an application's services, networks, and volumes in a declarative YAML file called a Compose file.
* A stack is created and deployed using the docker stack deploy command, which takes a Compose file as input and deploys the defined services onto the Docker Swarm cluster.
* When deploying a stack, Docker Swarm orchestrates the creation and management of the services, ensuring that they are distributed across the cluster's nodes according to the defined constraints and availability requirements. It handles scaling, load balancing, service discovery, and other orchestration tasks for the services within the stack.

**Services and Tasks**

* Docker containers are launched using services and they are 2 types of service deployments replicated and global (kubernetes replica set and damen sets)
* Global : if we deploy a task it will be divided with in all available worker nodes (it will creates all task)
* Replicated : Here we need to specify the count of the replica so that those many replica’s will be created

* Docker swarm init –advertise-addr ipaddress : which initilize the docker swarm using the following cmd
* Docker node ls : its to verify the worker nodes and the manager nodes
* Docker service ls : its to verify the services with in the docker.
* Docker stack ls : it shows the list of all stacks present on it
* Docker stack rm stackname: it removes the stack from it



* Docker stats : it used to show the statistics of the all running containers
* Docker stack rm stackname : it just removes the particular staks
* Docker swarm leave : it just leaves the worker node
* Docker node rm worker01 : inorder to remove the worker node
* Docker swarm leave –force : its used to leave the manager node from the swarm cluster.
* The network type of docker swarm is the **overlay**
* Note: in docker swarm it doesnt build images we need to use the existing images  
    
   Swarmpit
* Graphical ui for managing the stack called swarmpit we can install using the below cmd.
* After installing it will be expose on some port 888
* We can use it using ip:888

docker run -it --rm \

--name swarmpit-installer \

--volume /var/run/docker.sock:/var/run/docker.sock \

swarmpit/install:1.9