**DOCKER CONTAINERIZATION TOOL DOCUMENTATION**

**Virtualization with Hypervisor**:

1) what is virtualization process? Why do we use it?

Virtualization is nothing but exposing or creating IT services using software and hardware resources on the physical server and make available to users and applications to multiple environments on the compute cloud or cloud environment.

**WHAT IS HYPERVISOR? WHY DO WE USE IT? WHAT ARE THE TYPES OF HYPERVISOR?**

1. Hypervisor is a software installed on top of the physical server and used to create VMs where we run workloads inside the VMs.

2.There are two types of hypervisors:

1. Hardware Hypervisor also known as Native or Bare-Metal Hypervisor

2. Software Hypervisor also known as Host OS Hypervisor.

3. Difference between two hypervisors:

1. Hardware hypervisor virtualize hardware resources and software hypervisor virtualize software like OS. The name itself defines the difference b/w two hypervisors.

2. Hardware hypervisor is installed directly on the physical server whereas software hypervisor is installed on top the host operating system on the physical server

3. Hardware hypervisor has direct access to the hardware resources on the physical server and host OS.whereas in software hypervisor it make use of the host OS with the separate guest os installed on VMs to communicate with the resources on the physical server.

4. There is a drawback with type2 hypervisor. If the host OS goes down with any issue the entire hypervisor, VM's goes down and the status of workloads will also be down.There is a risk of losing entire data on VM if OS crashes down.

**Limitations of Hypervisor:**

1. Monitoring or provisioning VM is a time consuming process compared to containers.

2. Allocation of resources is not dynamic.

3. When resources are allocated to the VMs, major portion of the resource utilization is consumed by OS and the least portion which is left consumed by an application which makes application workflow process slow.

4. we can run no of VMs on the single piece of psychical infrastructure but lesser number compared to containers.

5. On physical server we install one OS and run multiple applications whereas in VMs we install different or same guest OS on every VM to run workloads for different purpose.

**SETUP OR CREATE VM USING HYPERVISOR:**

**HOW DO WE CREATE A VM ON THE VIRTUAL VM BOX OR HYPERVISOR?**

**Note**: Here in the example I'm using type2 hypervisor and unbuntu OS with the version 22.04 LTS.Always prefer Ubuntu 18,20,22 LTS(Long Term Service) for production level services.Ubuntu with version 21 for testing purpose.

->> Architecture: X86\_64(amd64),arm64,armhf and s390x.

->> We have multiple types of type2 hypervisor like VM Ware workstation, Oracle Solaris Zones,Oracle VM Server for x86,VMware Fusion etc. Here in the example I have taken Oracle VirtualBox.

->> In Ubuntu OS we need to create an additional user and password to login VM whereas in other OS like redhat, federo etc., we need to create password for the root user. Not required of additional user to be created.

1. Goto the browser and search for Oracle VM Box with version 7.0.6.It is type2 hypervisor which is installed on windows host OS on the physical server.

2. Once you click on the download link exe.file executes on your windows local system.Double click on the exe file of VM Box. Follow the instructions and perform installation process.

3. Once installation process has been completed, open VM box and start creating VMs with help of the type2 hypervisor Oracle VirtualBox.

4. Now goto browser and search for an OS to be installed on VM as Guest OS.Select Platform according to your base system requirements and Download latest version. Here in the example I have taken Ubuntu 22.04 LTS.

5. Download ISO image file of Ubuntu 22.04 LTS on your windows local system.

6. Open VM Box, select machine then new to create VM on VM Box.Give machine name and select ISO file path where Ubuntu image is downloaded and click next.Configure your VM during installation process.

7. Once VM installed successfully reboot the system.First we need to login with user and password which we add during installation process then we switch to root user.

8. Once VM with Ubuntu OS is created.We install docker to create containers to run workloads inside the containers.

**Docker Overview:**

Before devops principle is introduced, software developer’s team used to develop the code according to the client’s requirement and perform build, test to the code. Once the code is successfully deployed on the system’s environment, they send it to the operations team. They perform deployment of an application on the production server. If the application deployment fails, they will send back to the developers team. The chain process goes continuously until the successful deployment of an application is done. This process delays the application deployment. To Bridge the gap between two teams, devops came into the picture. Docker is the right solution to perform build, test and deploy of an application faster. Here both the teams collaborate and work together on a single platform to faster the process of an application deployment.

**WHAT IS DOCKER? WHY DO WE USE IT? WHAT ARE THE BENEFITS OF DOCKER?**

**Containerization Tool:**

1. Docker is one of the containerization platform tools introduced in 2013 and written in go language which is an open-source freely available in the market.

2. Docker is a software installed on VMs to create containers to run workloads.

3. Docker is used to build, test, deploy an application faster on a single platform compared to other software development traditional approach.

4. We need not install any separate OS on docker as it makes use of kernel packages and system resources to run workloads inside the container.

5. There is no burden of additional software installation process like guest OS which makes containers provide light-weight environment throughout the lifecycle.

6. It makes container portable which helps us to migrate from one platform to another platform easily.

7. Container is a pre-installed box bundled with libraries, packages, dependencies, software code, app code etc. which are required to run OS and application on container and deploys application faster.

8. Docker is scalable, reliable and Portable due to its services.Docker provides security to your applications with an isolation layer on docker host machine.

**Docker Tools:**

Docker platform provides multiple tools like docker engine(CE and EE) supports on linux system, docker desktop(Windows and MacOS), docker compose, docker swarm etc. Docker compose is now used as docker-compose-plug-in.

* Docker Engine: Docker engine is a manual tool from docker used to build, spin, and share containers. Since docker is platform independent, we can migrate the container from one platform to another easily. We can run any application written in any language on any docker environment. Docker engine runs on different platforms like VM, Bare-Metal, Cloud, desktop on windows and linux OS.
* Docker Desktop: If we need all features of docker on a single particular platform we use docker desktop. If we want everything should be an automatic process to run Kubernetes containers or pods, security, bug fixing, networking, patch work etc., use docker desktop.
* Docker Compose Tool: This tool is used for orchestration process on docker engine. Where we write docker file with all the configurations and specifications related to image, container, volume, port forwarding, networking etc. The docker file is to be sent as an input to docker engine to execute the file in an organized manner. This process is known as orchestration.
* Docker Swarm: It is used to create a cluster. It is easy to install and configure the cluster on docker but functionalities and services are less compared to Kubernetes. No of containers running and provisioning on docker are less than the Kubernetes cluster. Kubernetes provides various dependencies and plug-ins which are not contributed by docker due to which Kubernetes become popular than docker.

**DOCKER CONTAINER TOOLS:**

* LXC/LXD is the tool used to create container by docker. This tool supports only on Linux system not supported for Windows and MacOS.
* CRI-O, RKT, containerd, Docker-ce (CE and EE), Docker enterprise, podman, runc.

**DOCKER COMPONENTS:**

* Docker daemon (it’s a server used for communication)
* Docker-Client-CLI (used to run commands)
* Docker registry (OCl file,ISO images)
* Docker Objects (anything we perform on docker is known as an object like image download, containers, volume, networking etc., anything would be considered as an object.)

**WHAT IS KERNEL? WHAT ARE THE KERNEL PACKAGES AND USES?**

1. Computers can only read binary language.They do not understand human readable language. The kernel in linux system acts as an interpreter between the user and application or system.

2. The job of kernel is to convert the binary language into human readable language and vice-versa.

3. Kernel packages are nothing but Namespaces and Cgroups(Control Groups).These are the existing features of linux OS since it is introduced.

4. The job of kernel packages is to communicate with system's hardware and resources like CPU, memory, storage.

5. It sits between applications and hardware resources to make connection of all your application software to hardware to perform tasks.

6. Namespaces are used to create an isolation between the applications and containers running on the same host OS. Cgroups are used to manage resources like CPU, memory, storage allocated to applications.

7. Docker uses these kernel packages of linux system to communicate with hardware resources on the physical server.

8. Docker does not install any kernel packages.It makes use of kernel packages of linux system to create isolation between containers and resource management.

**PRE REQUISITE STEPS TO INSTALL DOCKER:**

1. Make sure what kind of OS version we need for the linux system.

2. Architecture which is required.

3. We need to delete the previous version of docker which is already installed before you install the new one.

4. Delete the folders like docker.io,/var/lib/docker,docker-ce,docker compose,docker-cli.

5. Uninstall docker packages and docker engine.

**Docker Installation Process:**

**Installations Methods:**

There are three types of installation methods:

1. Docker installation from apt docker repository which is also known as package manager.

2. Manual installation process using "DEB" packages.

3. Conventional Script.

**STEP TO BE FOLLOWED BEFORE INSTALL DOCKER:**

1. First we need to create Keyrings directory under /etc/apt/keyrings/docker.gpg/

2. We need to download GPG key under the specific directory.

3. Create the docker repository under /etc/apt/sources.list.d/docker.list

4. Download docker packages under the specified directory. It is also known as package management.

**Note**: All these steps are also available in the docker official websites.

**DOCKER ENGINE STEUP:**

-> Goto official Docker website <https://docs.docker.com/engine/install/ubuntu/>

-> Select the OS platforms like ubuntu,Federos,Redhat etc. according to the choice.

-> Take the latest or recent previous version of OS according to your system requirement or client requirement.

-> We can check required OS platforms on docker official website.

-> Choose the method from the available methods on the website.

-> First install the pre-requisites packages, then update the index package.Provide read permission to the docker public key.

-> Update index package whenever you install register new entries or services on docker.

-> Then install docker engine and other required docker packages.

-> Docker engine is successfully installed now.Verify the installation of docker by using the command:$docker -v or docker --version.If docker is successfully installed will get the deatils of docker engine and version of docker.

-> Start creating the docker containers with help of docker images from dockerhub registry and play around easily.

**DOCKER CONTAINER:**

1. What is a container? why do we use it? what are the advantages?

-> Container is a pre-installed box, packaged with libraries,services,dependencies,app code etc. which are required to run OS and application.

-> Docker is a software which helps to build,test,deploy an application faster.

-> A docker container is nothing but a software installed within the container along with application code,services and other dependencies to perform the deployment of an application faster.

-> Developers can directly deploy an application on production environment using containers with minimal configurations.They no need to do it from scratch.

-> In Docker need not to intsall seperate OS system and kernel packages.It will make use of harware resources and kernel packages of linux system to deliver its multiple functionalities.

-> It makes docker get rid of additional burden of software installations like OS which makes containers provide light-weight environment through out the lifecycle.

-> Docker seperates software or an application from the infrastructure dependency which makes container protable.We can move it from one platform to another platform irrespective of insfrastructure being used beneath.

-> We can run any application written in any language on any docker environment as it is platform independent.

-> Containers are independent from each other where multiple applications are running on the same docker host machine and runs as an isolated system on the docker host machine.

**FEATURES OF DOCKER CONTAINER:**

-> There is dynamic allocation of resources.It is an automate process done by docker.

-> Docker containers are scalable which means it automatically grows and shrinks according to the application requirements.

-> It is moere securable due to the creation of isolation layer between the conatiners and applications running on the same docker host machine.

-> Docker containers are portable due its light-weight environment nature.Easily moved from one platform to another platform.

-> It is also reliable to perform ochestration instead of manual intervention to perform regular tasks.

**DOCKER IMAGES:**

**what is an image? why do we use it? what are the types of images?**

-> Once docker engine is setup and ready to go.The next step is to download an image.

-> An image is a configured OCI file with all the specifications required for OS and application deployment.

-> An image is also known as repository and the place we store images is known as dockerhub or docker registery.

-> We can register an account with dockerhub to store images.

-> we have two types of images or repositories.Public and private repositories.

-> Authentication and authorization is required to pull and push images from the private image repositories.

-> We can download an image from public repository without any authentication because it is an open source and freely available on the public cloud but we cannot push an image to the public image repository without authentication.

-> Images can be shared across multiple containers for configuring containers with same specifications.

-> We can also build our own images and customise an image with the specific requirements according to the client. Then push it to the public repository with an authentication.

**WHAT KIND OF AN IMAGE NEEDS TO BE PREFERRED WHILE CHOOOSING AN IMAGE FROM THE DOCKERHUB:**

-> Three things to be prioritized before choosing as image on the dockerhub registery.It is always recomended to choose an official image which is known as a certified imaged.

-> Second option needs to be verified publisher.It is an image verified and published by docker community.

-> Third priority should be sponsered OSS(open source software).This has to be least priority.

**INFORMATION ABOUT IMAGES LAYERS:**

-> An image is compressed with multiple layers which are converted as folders and stored in the overlay2 storage driver. These layers are visible while downloading an image.

-> These layers are nothing but required services,packages and denpendencies that we configure while building an image.

-> We can see the layers and size of an image using the command $docker image history followedny image name and tag.

-> <missing> layers are only availble with the system where it is created.These layers are not available on the everyone's system.

-> We cannot predict an actual size of an image due to the existing layers of an image which are already downloaded and shared with other images.

-> Images are read only layers that we cannot modify the data of an image.If we want to customize an image along with the existing data we can do it on containers.

-> Because conatiners are read-write layers where you can modify or add addtional configurations to the data but we cannot modify an image data.

-> Third party providers like JFROG,AWS ECR,GITLAB REPOSITORY,AZURE CONTAINER REGISTRY,GOOGLE CONTAINER REGISTRY many more as well to store images as repositories.

-> Use commands #Docker images(list of images),#Docker image history <imagenamne:tag>(deatils of an image),#Docker image build -t <imagename:tag>(to build an image),#Docker pull <imagename>(pull an image from dockerhub).

**WORK-FLOW OF DOCKER IMAGE**: When we run pull command, docker CLI tool communicates with docker daemon then it communicates with the local system if an image is available in the local host system. If it is not available in the local host machine, establishes connection to the dockerhub repository to pull an image from the docker and downloaded on the host machine. The path of an image is /var/lib/docker/images/overlay2.

**Note**: Here overlay2 is a storage drive used to by docker to store images. Each and every layer of an image is converted as folder or directory and stored under overlay2 stroage drive.

**DOCKER CONTAINER CREATION:**

**HOW TO CREATE A CONTAINER USING DOCKER IMAGE?**

* First we need to download an image to create a container out of it.
* Please choose recommended approaches before you select an image. Always good to prefer an offical image which is officially certified.
* Download an image from the docker-hub using command $Docker pull <imagename:tag>.
* Verify if image is downloaded using the command $Docker images.
* Create a container using the image with the command $Docker container -d --name <containername> <imagename:tag>
* Check list of containers using command $Docker container ps -a or $docker ps -a to check if container is created.
* To check IP address and hostname of a container use command #docker container inspect <containername>. Other way is to check use command #cat /etc/hosts.

**DIFFERENCE BETWEEN RUN AND CREATE COMMAND:**

The run command creates container and runs as a backend server in a detach mode whereas in create commmand, it will only create the container but the status of the container will be halt. We need to manually start the container using start command $docker container start.

Note: EXEC command used to login the container.

**RESTART POLICIES OF DOCKER CONTAINER:**

**Why do we use restart policies?**

Restart policies are used to set some particular rules to a container. These policies are to be followed by containers to make sure application to be in running status.

**For Example**: An administrator manages all the containers on the VM's. Whenever he shutdowns VM, all the containers on VM will be exited and the applications will be in down status. When he runs hypervisor like bare-metal, VMware or any cloud, all the containers on the VM have to be uphand running otherwise the applications running inside the VM's will be in down status. An administrator has to start all the containers manually using start command. This is the complex task and time consuming for an administrator. To simplify this particular task we use restart policies of container on docker.

1. There are four types of restart polices of a docker container.

1. --restart no(never)

2. --restart Always

3. --On-Failure

4. --Unless stop

**1. Restart no(never):** This is a default policy that enables to a container by docker without using --restart no option while creating a container.

--> It never performs restart of a container with this default policy by docker engine.

--> Irrespective of exited code >=0, container never gets restarted if you restart docker daemon. This is the default feature enable by docker. We have to start containers manually using start command.

--> We can check the exit status with two commands: $Docker container ps -a or $Docker ps. and $Docker container inspect <containername>.

**2. Restart Always:** If this policy is assigned to a container, container always gets restart whenever we restart docker daemon.

--> Containers exited with any code, it always performs restart of a container. Does not perform any restart if container is stopped manually. We need to start the container manually using start command.

--> Whenever we stop container manually this policy never tries to restart a container even if you restart docker daemon. Again we have to start the container using start command.

**3. On-Failure: --restart on-failure:max-retries:** This policy is used to restart the containers upon the failure of exiting container successfully with code 0.

--> The containers which are exited with the code>0 are considered as not successfully exited from the loop. This indicates that there are some failures of a container.

--> If the exit code of a container is >0, it indicates that containers are exited due to some failures. This policy is when it exactly performs restart of a container.

--> If the exit code of a container is 0 or we stop container manually it does not perform any restart by docker daemon.

-->This restart policy have got an option to perform max tries to restart a container.

**4. Unless-Stopped:** This policy never perform restart of a container unless it is stopped with code>0. This policy never tries to restart a container exited with code 0 or we stopped it manually. The restart policy is ignored when we stop it manually.

--> If the exited code is >0 only when it tries to restart a container when you restart the node. If the container is running before reboot, then the policy tries to restart the container but if you restart your docker daemon it doesn’t perform any restart of a container.

-->It is little similar to restart always policy.

**Points to be remembered for the restart policy:**

* The containers should be up hand running for atleast ten seconds to apply the restart policy.
* We can attach restart policies during or after container is created.
* Policies can be modified with an option —update.
* The restart policy is ignored by docker if the container is stopped manually. It does not apply until you restart docker daemon or start it manually using start command.

**HOW TO PERFORM SSH CONNECTION BETWEEN VM AND LOCAL SYSTEM:** One of the reasons we prefer to use SSH connection is When we create VM on the hypervisor, The interface of VM is not user-friendly to copy commands that's when we access VM from windows local system through SSH connection.

--> We can access VM from local system through NAT(Network Address Translation) by port forwarding.

--> VMs are already configured with SSH connection during installation process. We configure host Port and guest port on networks to connect from local syatem.

--> Host port is the port assigned to the windows local system and guest is assinged to VM to establish connection between local system and VM.

--> Any packet triggered on the local system port is redirected to guest port on VM to make the connection possible.

--> We can connect VM through SSH connection from the local system by using command $ SSH -p<hostport> <username of VM>@<IPaddress of VM>.

--> We can be able to login VM through windows local system from command prompt or other mediums like mobaxterm, putty etc.

**HOW TO ACCESS VM FROM LOCAL WINDOWS SYTEM WITHOUT ANY PORT FORWARDING:**

--> configure networks on Virtual Box from NAT to bridge adaptor. Then goto advance under which choose promiscuous mode and select an option called allow all.

--> It means that any network packet arrived can be read by adaptor on the host system. Which means that any network request from local system can be read by bridge network to access VM without any port forwarding.

--> When network is connected to bridge adapter then the router shared by local system is also shared by VM. which means that VM and local system are connected to the same router and shares same network.

--> When we install docker engine on VM, Docker host machine shares an Ip address from the same network series of local system because they are connected to the same router through bridge network.

--> The default docker 0 interface network is also enabled on the docker host machine when install docker engine which is 172.17.0.1/16.

--> When we create a container the default docker 0 interface network series enables IP address to the container.

-->Use command SSH <username of VM>@<IP address of VM> shared from the local system router to login VM using bridge adapter network.

**WHY DO WE NEED TO ASSIGN STATIC IP ADDRESS TO THE NODE:** when we connect to the bridge network the IP addresses are assigned dynamically to the nodes from the router. whenever we reboot the node we have new address assigned to the particular node from the router. Due to dynamic IP, nodes will have different identify over time. It will become more difficult for us to identify the specific nodes on the network. That’s when we assign static IP address to a container.

**ADVANTAGES:**

* The network Identify of the node does not change.
* Administrators can retain tight control over identities of node.
* Network resources can be mapped to the unchanging IP address of the specific node.

**NOTE:**

* Mistakes cannot be made during the configuration of static IP address.
* Administrators need to be more careful while assigning static IP. Administrators should not accidentally assign duplicate IP addresses to any node.
* Setting or changing IP address configuration is manual process and time consuming.
* Use command $ IP a to check IP address of the node. We can also use command $cat /etc/hosts to check IP address of the host machine.

**HOW TO ENABLE STATIC IP ADDRESS TO DOCKER HOST MACHINE:**

* Open file /etc/netplan/00-installer-config.yaml with editors like nano, vim etc.
* Configure network file with static IP address, subnet mask, default gateway, name servers and other values.
* Dynamic Host Configuration Protocol (DHCP) is responsible for assigning dynamic IP addresses to the host node. By default it is configure as true in the network config file.
* Change it from true to false to stop assigning dynamic IP’s to the node and configure the file with desired address, subnet mask, default gateway, name servers and other values.
* If you are confused or not known which IP address range to be chosen, open command prompt and type command ipconfig to know IP address, subnet mask and other values.
* To save the changes to the file press CTRL X + CTRL Y followed by enter.
* Once configuration is done, use command $ netapply to apply the changes and reboot the node.

**HOW TO ACCESS APPLICATIONS FROM THE BROWSER:** By default docker runs through non-network UNIX socket:/var/run/docker.sock.which means that docker cli and docker daemon needs to be installed on the same host server. Then only both docker cli and daemon are able to communicate with each other within the unix socket file.

--> They are not able to communicate If docker CLI and docker daemon are installed on different host servers. To establish connection between them which are running on different servers we use options like SSH or TLS(HTTPS).

--> By default, Containers gets an IP address from docker 0 interface networks series 172.17.0.0/16 which is the default program designed by docker.

--> Container is able to communicate within the host machine by using curl command along with an IP address assigned by default.

--> It is restricted to communicate to the browser or internet because the IP addresses shared by containers are private Ip address.

--> To access application running inside the container on docker host machine from the browser we have an option called port forwarding.

--> We can enable port forwarding while creating the container. By default containers listen to the port 80. Bind the host port to the container port to enable port forwarding while creating a container.

--> Whenever a packet sent to host port, it redirects the packet to the container port to access application inside the container.

--> To access application from the browser use docker host IP address along with host port. For Example: 192.168.1.54:8080.

--> Host machine should not use same port number for multiple containers. Otherwise it throws an error.

**Note:** Port forwarding is enabled only when creating the container. We cannot assign any port forwarding after container is created.

**DOCKER NETWORKS:**

**Supporting network types of docker:** we have three default enabled networks by docker during installation of docker engine. They are bridge, host and none networks. We also have third party networks like overlay, Ipvlan, Macvlan.

**Bridge Network:**

* This is the default bridge network enabled by docker while creating a container.
* The default docker 0 interface is assigned when we install docker engine. It’s a default program written by docker to ensure the default docker 0 interface is available to all the containers created under the default bridge network.
* The containers created under the bridge network shares IP addresses from the default network series i.e., 172.17.0.1/16.
* There are two types of bridge networks. Default and user-defined bridge networks.

**Default Bride Network:**

* This is a software bridge network enabled by docker.
* The containers created under this network shares IP addresses from the default network series.
* The containers created under the default bridge network can communicate with each other but cannot communicate with the containers running under different bridge network on the same host machine.
* Docker bridge network creates IPtables which restrict communication between the containers running under different bridge network on the same host machine.
* We have two ways to establish the communication between the two containers running under different bridge network on the same docker host machine.
* Access communication by manipulating IPtables.
* Using command $docker network connect <networkname> <containername>.
* To check docker0 interface on the host machine using command $IP a.
* We cannot assign any static IP address to a container running under default bridge network using option --IP. Even If we try with an option-IP it fails.
* Containers running under default bridge network can address each other with only IP addresses by default.
* If we want containers running under the same default bridge network to communicate with each other with names, we need to manipulate the hosts file on the docker host machine.
* By default, environment variables are shared among containers on default bridge network.

**User Defined Bridge Network:**

* We can create and customize our own bridge network. These networks are known as user-defined bridge networks.
* In user-defined bridge networks, we can create two types of bridge networks. Software and hardware bridge network drivers.
* User-defined networks are customized with subnet and gateway while creating a network.
* If we do not assign any parameters while creating a network, default network series of docker0 interface will be assigned to a network by docker daemon which is the default program designed by docker in the backend server. For example: 172.18.01/16.
* We can address containers running under the user-defined bridge network with IP address and names as well.
* User-defined bridge networks helps to assign different types of network settings like DNS, MTU, IPtables etc., according to the application requirement.
* We cannot share environment variables among containers running under the user-defined bridge network by default. We need to use an option --link to share environment variables among containers.
* Static IP address can be assigned to a container using option --IP in the command unlike default bridge network.

**POINTS TO BE REMEMBERED:**

* Bridge, host, none networks are created by docker engine package. These three networks are default networks enabled by docker engine.
* User-defined bridge network can be created with or without subnet and gateway values.
* It is not recommended to use default bridge network in real time basis to avoid risks.
* Containers can be connected and disconnected from both default and user- defined networks on the fly using command #docker container disconnect <networkname> <containername>.
* Pre-defined networks are created by docker and user-defined networks are created by users.
* When containers are attached to a user-defined network, we cannot delete the these bridge networks due to active endpoints.
* Endpoints can be verified by using command $docker network inspect <networkname > then refer endpoints column. To check the containers attached to the network refer containers column.
* Networks which are not in use can be removed by using command #docker network prune.

**HOST NETWORK:**

* It’s a pre-defined network by docker. Host network assigns IP address to containers from the docker host network from the router.
* Applications and Containers run under host network shares the network stack of docker host machine. There is no need of any port forwarding to access application from the browser.
* To access application from browser use IP address of the host machine followed by container port number.
* There is no default interface or IP address enabled on host network like default bridge network.
* Host network doesn’t have any particular IP address instead it instructs containers to use the IP address of docker host machine.
* Apart from network stack all other namespaces like pid, ipc, utc etc are isolated from the containers as usual.
* We have to ensure the applications and containers under host network running on the same docker host machine do not share same port number 80.
* Host machine supports only on linux system not supported for docker desktop for windows,MacOs or docker EE on windows server.
* In general, it’s not recommended to use host network, only if there are dynamic ports assigned to containers.
* It is not recommended to use host network in real time basis because we share IP address of a docker host machine to containers which means we are giving access of sensitive files of host machine to containers to manipulate host network data which effect all the applications running on docker host machine.
* The applications and containers which are already existing will also share the same IP address of the docker host machine by default.

**WHEN WE USE HOST NETWORK:**

* When we do not want to isolate network from containers and applications running on the same docker host machine we use host network.
* If we want to share the network stack of host machine to containers we use it.
* There is no static port or where we have wide range of ports on your containers we use host network. Here port forwarding does help to access application from the bowser due to wide port range.

**HOW TO TROUBLESHOOT IF THE CONTAINER EXITED WITH CODE (1)? HOW TO RESOLVE IT:**

If we use same port number to multiple containers, it throws an error with exit code (1). Then we have to check the logs of a container to verify the same by using the command #docker logs <containername>.

**Note:** Here it shows if bind port is already in use. If the port is already used by any application or container running on the same docker host machine, we can verify it here.

We have another command to check all the available ports on the docker host machine. #Netstat – plant.

**POINTS TO BE REMEMBERED:**

* Only one host network is allowed which is created by docker host machine. We cannot create multiple host networks but we can create an host instance of doing host work.
* Host network provides direct access to the applications running inside the container from browser without any port forwarding.
* It is preferred to use when we want to share only network stack from the docker host machine and isolate other packages or namespaces to a container.
* Containers where it handles large range of ports, where ports are dynamically assigned to containers, we use host network.
* We cannot attach a container to a bridge network when it is already assigned to a host network.
* Host network is only available on linux system.
* Pre-defined networks are not allowed to remove.
* We cannot disconnect containers from host network like default bridge network and none network.
* We need to execute --network option while creating a container to connect to host network later we cannot connect containers to host network once it is created like other networks.

**NONE NETWORK:**

* None network is also a default network created by docker engine. None will not communicate outside the docker host machine because by default it is not connected to any network.
* It will have only loopback interface which is available on docker host machine, local system and container to communicate internally within the docker host machine.
* We cannot access an application out the container with curl <IP address of container> command because it doesn’t have any network or IP address.
* We can access application only from inside the container. It is only for internal communication.
* If we want to access curl command with IP address we can disconnect container from non network and attach to bridge network.

**LIMITATIONS OF NONE NETWORK:**

* If we want to create a container should not be attached to any network and is used only for testing purpose to deploy application as a container, then we choose none network.
* We have to login to the particular container to access the application. We cannot access the application outside the container.
* Application does not have any IP address to connect from the outside of container.
* The image has configured with all the packages and services that are required to run an application inside the container without any network..
* We cannot install any packages inside the container because it is unable to reach any repository without network.
* None network containers are able to disconnect and connect to the bridge network.

**OVERLAY NETWORK:** Overlay network is used to communicate with containers running under two different host machines. The containers created under overlay network will be assinged a unique IP address irrespecting of where containers are running. There will be synchronization between the two nodes. We need a data store to implement the overlay network to store data as a high-availablity. Here we are using etcd as a high-availability key-value pair storage.

**ETCD:** Etcd is a free and open source distributed and secure key-value store used to store critical information of overlay network introduced by harshicorp. Etcd is written in Go language and is extremely fast in distrubuted systems without disturbing the performance overhead from clustered machines.

**FEATURES OF ETCD:**

1. One of the main features of etcd is strong consistency model, which ensures all nodes in the cluster see the same data consistently.
2. It is well suited for the applications which need strong consistency, that allows to co-ordinate the actions of multiples nodes
3. Etcd provides high-availability and automatic failover which means it continues operating consistently even if some cluster nodes fails.
4. Another feature is to support distributed transactions which means it allows multiple nodes to make changes to the data in store in a co-ordinated way, ensuring data consistency.

**Procedure:**

To immplement overlay network, first we need to deploy etcd cluster on the host machine. We can download etcd from package manager or source code. Below are the steps to follow:

**Download from package manager:**

$ sudo apt-get update

$ sudo apt-get install etcd

Step1: Pre-built binary installation.

We can start by updating package repositories and installing the required tools by using below commands:

.$ sudo apt-get update

$ sudo apt-get install wget curl -y

Step2: Navigate to the binaries release page on github. Ensure to choose 3.5 and higher version. Then download the binaries using CURL or WGET commands:

$ wget <https://github.com/etcd-io/etcd/releases/download/v3.5.6/etcd-v3.5.6-linux-amd64.tar.gz>

Step3: Once package is downloaded, extract the downloaded archieve with the below command:

$ tar xvf etcd-v3.5.6-linux-amd64.tar.gz

Step4: Rename the extracted directory. Then open directory and move all the binary files into /usr/local/bin floder as below:

$ mv etcd-v3.5.6-linux-amd64 etcd

$ cd etcd

$ sudo mv etcd etcdctl etcdutl /usr/local/bin/

Step5: verify the installed binary and check if it is working on your system by checking the installed version:

$ etcd --version

etcd Version: 3.5.6

Git SHA: cecbe35ce

Go Version: go1.16.15

Go OS/Arch: linux/amd64

We can also check installed etcdctl and etcdutl utilities versions as below:

$ etcdctl version

Resulting output:

etcdctl version: 3.5.6

API version: 3.5

$ etcdutl version

etcdutl version: 3.5.6

API version: 3.5

Note: We can check the detailed installation process here on official website. <https://linuxconfig.org/how-to-install-etcd-on-ubuntu>

ETCD SERVICE SETUP: According to the istallation method we need to stop,start or restart the etcd service on your system. If etcd database is installed with the package manager, run commands as below:

* First start the etcd service with the command $ sudo service etcd start.
* To stop the etcd service with the command $ sudo service etcd stop.
* To restart use command $ sudo service etcd restart.
* To check the service status use command $ sudo service etcd status.
* To check health status of cluster use command $ ./etc cluster-health.

Note: We have to run the docker service again with etcd parameters. We need to stop the older docker server and reinstall docker with new etcd configuration using command:

$ service stop docker or systemctl docker stop.

sudo /usr/dockerd -H tcp//0.0.0.0:2735 -H unix:/var/run/docker.sock --cluster-store=etcd://192.168.1.68:2379 --cluster-advertise=192.168.1.88:2379.

Etcd is successfully configured with the above steps. Once etcd is successfully configured, create overlay network. We need two docker host machines to work with Overlay network.

Now any one of the node clusters create overlay network and updated etcd, then the changes reflects on all the other node clusters that are sharing data from etcd cluster. To create overlay network use below command:

$ docker network create -d <type of driver> overlay <driver name> webapp-overlay.

The same overlay network is created on the other host machine because we are using etcd as a high-available key-value pair to store docker cluster node information that available across multiple nodes connected to etcd. We have not used any subnet and gateway while creating the network. It assigns the default network series of overlay network i.e.,10.0.0.0/24 like we have docker 0 interace in bridge network.

Note: If we check the details of overlay network using command docker network inspect <networkname>. We see "scope" as global which means this network is used across all the nodes. We can also see how many nodes which are using this particular etcd cluster can this network. Right now we are using overlay network.

**HOW THIS PARTICULAR DATA IS STORED IN ETCD**: Here we run few CLI commands and verify how data is stored in etcd.

$ ./etcdctl is /docker/nodes (These are the nodes sahred by etcd).

$ ./etcdctl is /docker/network/v1.0/network (Created networks are stored under etcd).

$ docker network ls.

Then launch a container using overlay network on both nodes. using a command $ docker container run -d --name nginx-demo --network webapp-overlay nginx.

**Note**: We cannot create another container with same name on the other node.

Then login container and run ping command $ ping -c3 10.0.0.3. Two networks are visible, one is from bridge and another is from overlay network. To check login inside the container and use command $ip a.

**Note**: We can address container running on other docker host machine with name like user-defined bridge network. Because here domain is the point to overlay IP address.

**POINTS TO BE REMEMBERED ON OVERLAY NETWORK:**

* Overlay netwrok is the default feature available on docker swarm tool.
* We have immplemented overlay network for docker engine machines using external key-value storage service.
* Etcd key-value storage is highly prefered compared to consul, Redis etc.
* Etcd uses VXLAN as backend component for overlay network. VXLAN plays an important role in overlay network.
* Once overlay network is implemented, these details are stored in etcd. The information stored in etcd cluster needs to be available for all the nodes.

**DOCKER STORAGE:**

What is storage? Why do we use docker storage?

Storage is used to store sensitive data that can be accessed later when required. Containers are ephemeral which means the data of a container is not persistent. If the container's data required to be accessed even after deleting the container, we can store the data in a storage. So that we can access the data later where not required to work from scratch which saves time.

There are three types of storages in docker. They are as below:

* Volume Mount.
* Bind Mount.
* Tmpfs Mount.

**1. VOLUME MOUNT:** Volume mount can be created or controlled by using docker API calls or CLI commands. The volume mount stored under the docker area /var/lib/docker/volumes. If volumes are not available in the system while creating the container, docker API call in the backend server creates the volume first then container is launched. We can share volumes across multiple containers.

**Note**: In docker swarm tool use --mount option to attach a volume to a container. Use -v option for docker engine.

**POINTS TO BE REMEMBERED**:

* Volume mount shares only docker area under the host file system which is docker root directory /var/lib/docker/volumes.
* We need not to mention the absolute path instead we can mention relative path to access the volume data.
* We can launch a container without creating volume as a pre-requisite step.
* We can attach volume as read-only file to containers to ensure the data is not modified by an application. Becuase containers are by defalt enabled with read-write options.
* Any data that is modified should be under the container but not on the volume mount.
* We can use -v or --volume or --mount options to attach volumes while creating a container.
* We can create both static and dynamic volumes.
* If we modify data on volume mount, the modifications will reflect on the nodes attached to that particular volume.
* We cannot delete a volume while it is used by a container. There will be active endpoints which do not volume to be deleted instead it throws an error.

**2. BIND MOUNT**: Bind mount helps to stote data of container application anywhere in the host file system. We can store data on any root directory or parent directory of the host file system. If bind mount is used, Containers will have access to the entire host file system on the docker host machine. Here we need use an absolute path of the docker host machine to mount the staorage to a container.

**POINTS TO BE REMEMBERED**:

* Bind amount allows access to sentitive files on docker host machine which has to be used only when required.
* In real time basis, we are not recommended to use bind mount due to the risk that sensitive information can be modified by containers.
* Bind mount expects storage file or directory to be available while creating a container whereas in volume mount, it creates storage file or directory if it is not already available in the system while creating the container.
* The empty file or directory like parition mount is created with -v option if storage file is not already available.
* Bind volume can be shared across multiple containers like voume mount.
* While using docker swam tool use --mount option to mount bind mount.

**3. TMPFS MOUNT:** It will store data on the RAM not the disk. The sensitive information of a container stored on RAM until it is in use. Once container is stopped or removed the tmpfs mount files also be deleted. This type of storage is not persistent, used only when the data is required to be available for certain period of time to fulfil the purpose.

**POINTS TE BE REMEMBERED**:

* This type of storage used for temparory pupose not throughout the lifecycle.
* If we stop or restart container the data stored as tmpfs mount will be lost.
* We use either --tmpfs or --mount to mount the storage but not -v option in tmpfs mount.

**HOW TO CREATE A VOLUME AND HOW TO MOUNT IT ON A CONTAINER:**

$ docker volume create <volume name> <name of the application>**.**

$ docker volume inspect <volume name> to verify the details of volume.

$ docker volume ls to vetify the available volumes.

$ ls -l /var/lib/docker/volumes root directory where volume is created.

Now we can mount this volume to a container as below:

$ docker container -d --name nginx-demo -v <volume name:relative path> -p<host port:container port> <imagename>

Note: /usr/share/nginx/html/index.html this is the location of application file on docker container.

$ ls -l /var/lib/docker/volumes/nginx-data/-data/ this is the location of volume mount on docker host machine.

Note: By default containers are enabled with read-write permissions.If we do not want containers to modify any data on volume mount, delete the existing container and create new one with read-only permission. We cannot modify permissions on running container. Same procedure can be followed with all the three storages.

**DOCKER FILE:** Docker file is a simple text written in key-value pair format. It does not need any programming language to write a docker file. Docker file is used for orchestration process in docker to manage data in an organized manner. It is a customized image build system from docker. A customized image is designed to satify the oragnizational requirements for an application deployment. we need an understanding of a docker file to build our own image.

**POINT TO BE REMEMBERED**:

* Docker file is known as orchestration tool in docker which has all the commands that are required to build and create an image.
* By deafult #docker build command looks for a dockerfile where it can find all the configurations of an image to build an image.
* We can use name conventions like dockerfile.prod,dockerfile.dev etc for multiple docker environments
* Dockerfile fulfills the gap between the application and clients requirements. There is not required to look for parameters that fulfills the requirements.
* We can manually build an image but we can automate the build process using continous integration tools like jenkins, dockerhub build, codecommit etc.

**SAMPLE DOCKER FILE:**

FROM: Ubuntu:22.04

MAINTANER: "Shravya Sri<gmail.com>"

RUN: apt update \

&& apt install nginx

RUN: apt clean

COPY: index.html /var/www/html

EXPOSE: 80

CMD: ["nginx",".g","daemon off;"]

save the file :wq!

Note: Key will always be written in capital letters.

**DOCKER DIRECTIVES**:

**FROM:** The first directive we use in dockerfile is FROM. We can also use ARG. To define a base image, it will connect with dockerhub and download the image. It works on tasks accroding to the dockerfile.

For Example: FROM [--platform=<platform>] <image>[@<digest>][AS<name>]

Note: Use smallest image as base image to save the storage of DHM(docker host machine) if you are using any third party module. We can delete an image once task is completed.

**MAINTAINER AND RUN DIRECTIVE (WITH OR WITHOUT BUILD CACHE):**

Maintainer is used to define author details who builds an image. It is an optional directive. Run is used to execute commands. We have two types of formats to execute commands.One is shell and the other is exec.

(linux /bin/.sh and windows: cmd /S/C)

(["executable", "param1", "param2"])

For Example: RUN apt-get update (shell) to use shell commands

RUN ["apt-get","update"]

RUN /bin/bash -c "apt-get update" (bash) to execute bash commands.

**LABEL AND ENV(ENVIRONMENT):**

Env is used to set environment variables for containers. It helps to pass values inside the container. We can override the values of ENV while creating the container. Env values are persistent and may cause unexpected behaviour. To check environment variables listed in the docker file use command $docker image inspect <imagename>.

For Example: ENV MYSQL\_ROOT\_PASSWORD=siri123.

ENV DEBAIN\_FRONTEND=non intercative.

ENV org.opentainers.image.authors=<gmail id>.

Note: Use environment variables while creating the container not in an image. Use env variables when required not to store any sensitive information.

Labels are used to add meta-data to an image. To add an additional informatiomation to an image. It supports key-value format. We can declare LABEL in single or multiple lines in dockerfile. Recommended to use label instead of maintainer.

For Example: LABEL version="22.04"

LABEL ENVIRONMENTS"production"

**ARG AND EXPOSE**:

ARG is used to set values at image level(build time). This is the only directive which is accepted as file step in docker file. ARG variables can be called using $ARG if u want to call any argument in the dockerfile.

HTTP-proxy, https-proxy, ftp-proxy,NO-proxy, All-proxy.

Expose is used to inform docker on which port the container is accessed. The values passed in expose does not effect the actual port. It supports tcp and ucp protocols. Default:TCP.

For Example: EXPOSE: 80.

**COPY AND ADD:**

Copy is used to copy files from build system to image layer. We can also set ownership as well. Copy -chwon onlyworks for linux containers not for windows containers. Copy with or without --link option but we use --link option only works with buildkit. It is used to access local files from the docker host machihe.

For Example: cp /etc/otp/file cp /etc/docker/file.

Add is similar to copy directive. We can also add ownership to the files like copy. ADD -chwon onlyworks for linux containers not for windows containers same as copy. But the additional usgae of ADD is to extract zip/tar files to image layer. Add directive is used to access remote URLs for its source argument. It is used to fetch remote files.

For Example: ADD URL.

**ENTRYPOINT AND CMD DIRECTIVE:**

Entrypoint directive is used to be executed after creating the container.To define the command use ENRTYPOINT and CMD for arguments. When we use both these directives together we can perform more functionalities. Any values that is passed while creating the container will be added as argument to ENTRYPOINT.

CMD directive is a default command used to run after creating the container. The default command in dockerfile can be over written using $ docker run/create. This directive works better with ENTRYPOINT as an argument. We need to define only one CMD, If defined multiple CMDs last CMD will be effected.

Note: If we want to change the entire command go for CMD otherwise choose ENTRYPOINT to append the data to the existing or default command. If we want to have some fixed values which should not be over written or change some particular portion of the command use the combination of both ENTRYPOINT AND CMD directives.

**SHELL AND WORKDIR:**

Shell is used to make sure the commands to be executed under particular shell. This directive is mostly used for windows containers to defind CMD or powershell. Multiple shell directives can be able to create and accepted.

Workdir directive is used to configure working directory. We can define it multiple times in a dockerfile. Container exec shows WORKDIR as working directory.

For Example: SHELL ["/bin/bash","-c"]

**HEALTHCHECK AND STOPSIGNAL DIRECTIVES:**

Whenever we build an image using dokerfile, an intermediate container will be created to execute particular task and exits from the loop. Before it exits converted into an image layer then exits the loop. For that a signal is required to be sent to the container to exit the loop. This particular directive is used to send the singal to the contiainer.

For Eexmple: Format SIG<NAME> or SIGKILL <PID>.

Note: We can over write the default signal using --stop-signal option.

Healthcheck is used to check the health status of the application running inside the container. Healthcheck is executed using CMD.

For Example: HEALTHCHECK [options] CMD <command>

HEALTHCHECK NONE

--interval=DURATION (default:30s)

--timeout=DURATION (default:30s)

--start-period=DURATION (default:0s)

**APPLICATION DEPLOYMENT ON JENKINS USING DOCKERFILE:** Jenkins images are deprecated in favour of image jenkins/jenkins:lts. It is provided and maintained by jenkins community. Choose this particular image to download jenkins on docker host machine. usin command $ docker pull jenkins/jenkins:lts.

Jenkins is an automation tool used to automate deployments. First write the dockerfile with all the configurations using all the required directives in the file. Goto jenkins dashboard and add credentials of docker to integrate docker with jenkinsfile. Jenkins can be able to access and read the dokerfile to execute tasks according to the dockefile once authentication is provided through credentials on jenkins.

It automates all the manual processes of creating image, container, attaching volumes, networking, security etc., we have other automation tools like ansible, codecommit etc.