**\*\*We can run as many containers as we need with the same image, provided containers names are different.**

**\*\*Service discovery will be disabled in bridge network,(if u create own custom network then it will get enabled so containers can ping by IP/name)**

**Q. What is Docker?**

Docker is a very popular containerization platform which packages your application along with its dependencies in the form of containers, so that your application can work seamlessly in any environments be it Dev,test,or production.

* 

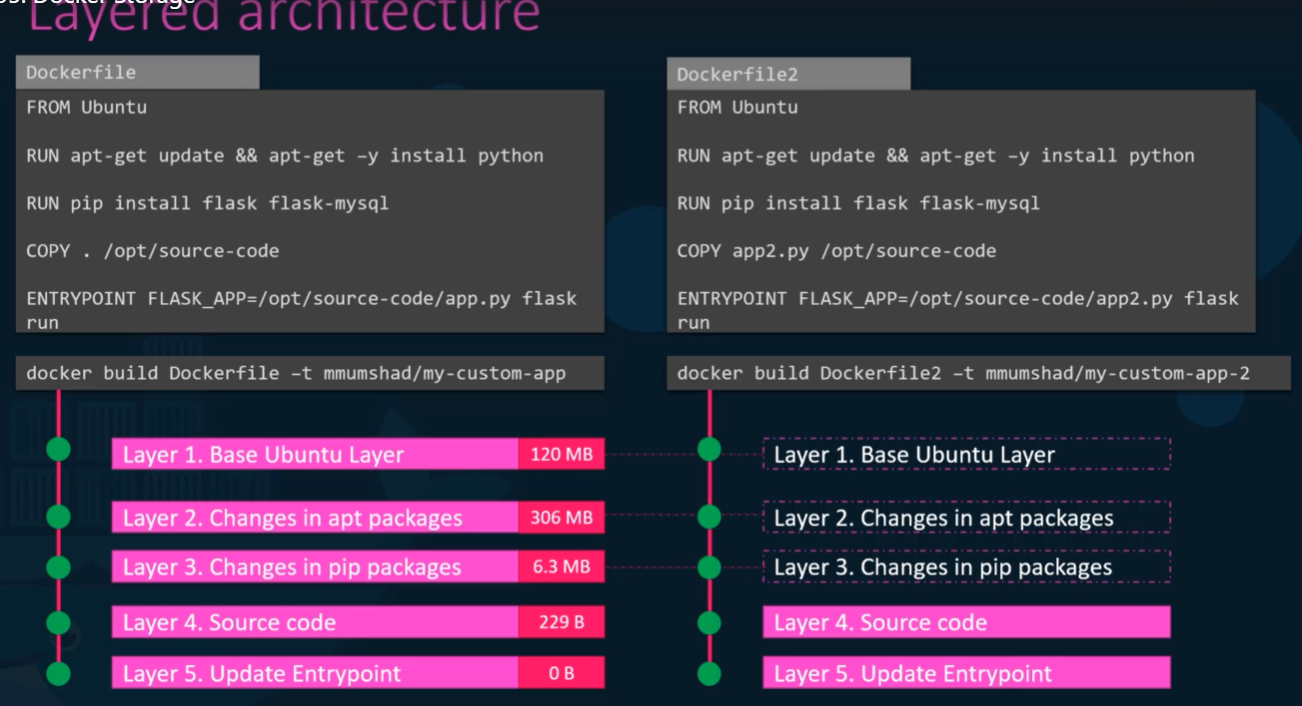
With Docker both Developers and operation teams works together and Developers tells us the info about the application ,so operations team can write the Dockerfile accordingly, so that bridges the gap.

Containers will have their own Networks ,Processes ,mounts they'll be indepedent to each other.

**How is Docker different from standard virtualization?**

Docker is operating system level virtualization. Unlike hypervisor virtualization, where virtual machines run on physical hardware via an inter-mediation layer ("the hypervisor"),

**Can you please explain the layered architecture of image(Union FS) , when the image is build?**

 Whenever we build an image with the help of dockerfile for the first time an image layers(Read-only) will be created for each instructions in the dockerfile,as shown above. Thereafter if we build same/diff image and say some of the instructions in the dockerfile are same ,then docker will not build those instructions again as they’re same instead it’ll re-use the same layer from the cache..This way docker builds images fast and efficiently and saves disk space.

Whenever we build the image for the first time using “Docker build” command then an image layer will be created ,which is READ only by default”

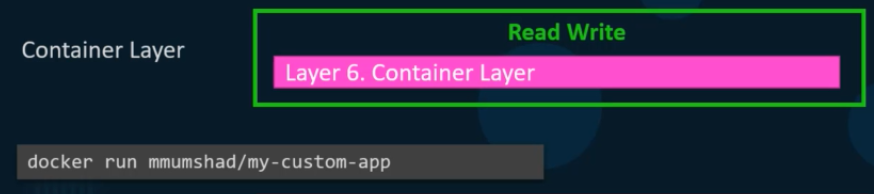


You can modify that by using a new build.

Whenever we create a container out of this image a new writable layer will be created on top of the image layers.



This writable layer is used to store the data created by the container such as log files or any files modified by the user ,life of this layer though as long as the container is alive.



**Disadvantage**: One of the drawback with the docker is whenever one layer changes all the subsequent layers will be executed again(despite of no changes),while re-building.To avoid this we need to do (Dockerfile optimization)

**How to attach a volume to a container to preserve the data while the container is running?**

Firstly will create a volume using the create volume cmd as shown.

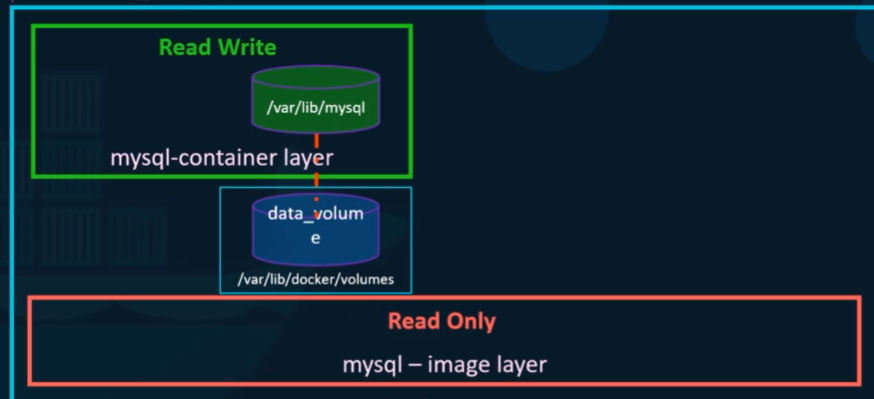
It’ll create a volume (data\_volume) under var/lib/docker/volumes



After creating the volume ,we can mount that by using –v



When you do this a new writable layer will be created and it’ll be attached to the specified volume,so that the data can be preserved ,though the container dies for the unknown reason.



Note :: It is not mandatory to create the volume prior by using **docker volume create**,whenever we write **docker run –v** it’ll create the volume automatically and attach it to the container.

**And this is called volume mounting**.

**Can you please explain docker networking ?(Network drivers)**

Whenver we install docker ,it automatically creates 3 networks

1)Bridge(Default)—

2)None -----

3)Host-----

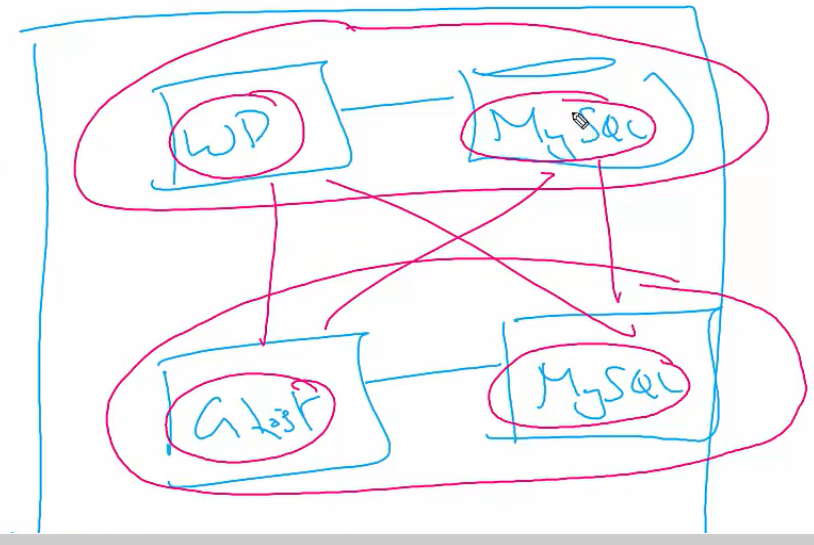
**Bridge**: The bridge network is a private default network created by docker on the host.When multiple containers are running on the same host and using the bridge network ,all the containers can communicate each other,all containers get an internal IP address and these containers can access each other, using this internal IP. If required we can access these containers from the outside world by port forwarding.

Types: 1)Default bridge network 2)Custom bridge network

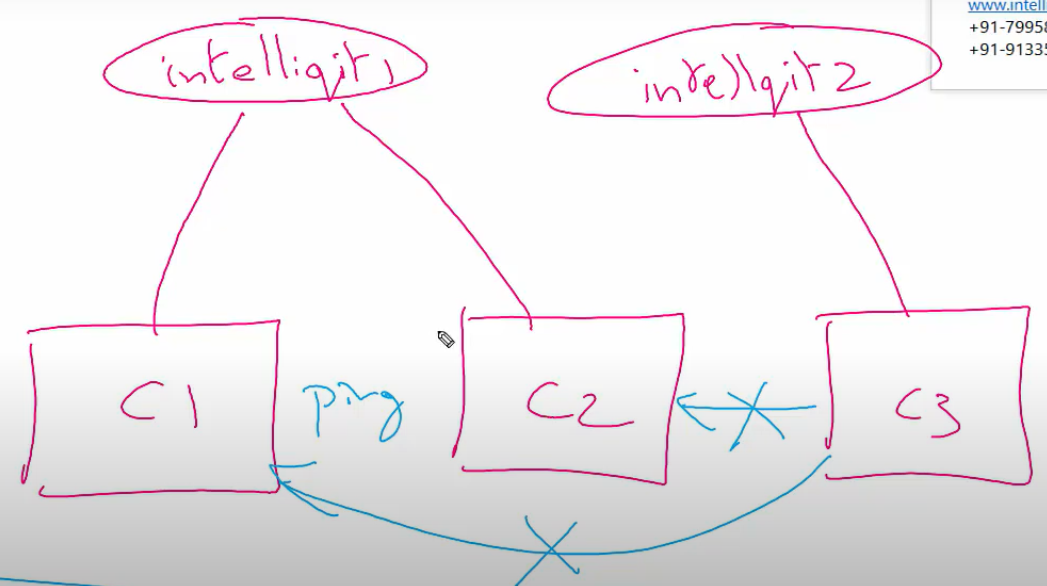


Ex: Docker run ubuntu

Disadvantage:



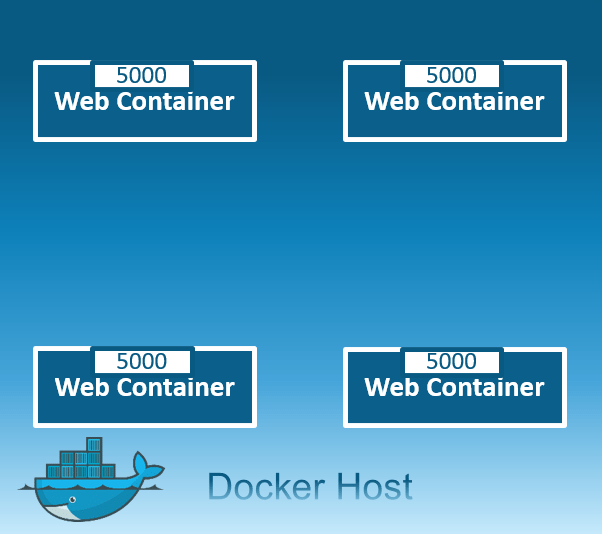
**Note** :: One container can be attached to multiple networks.(using **docker network attach**)



**Host**: This driver removes the network isolation between the docker host and the docker containers to use the host’s networking directly.Containers share the same network namespace ,Tcp/ip stack as your host is running.(Its not a docker best practice to run a container on this network)

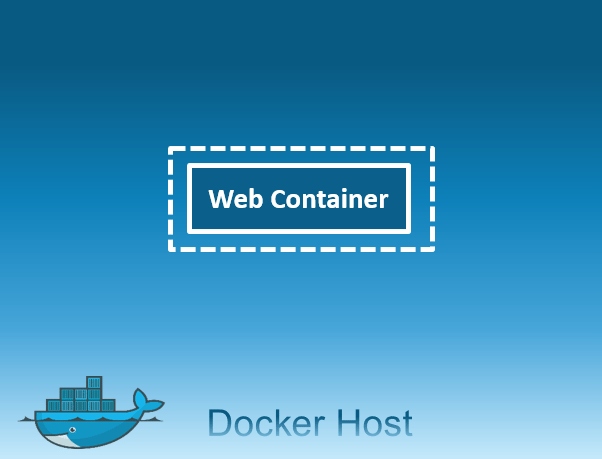
So with this, you will not be able to run multiple web containers on the same host, on the same port as the port is now common to all containers in the host network.

Ex : Docker run ubuntu – network=host



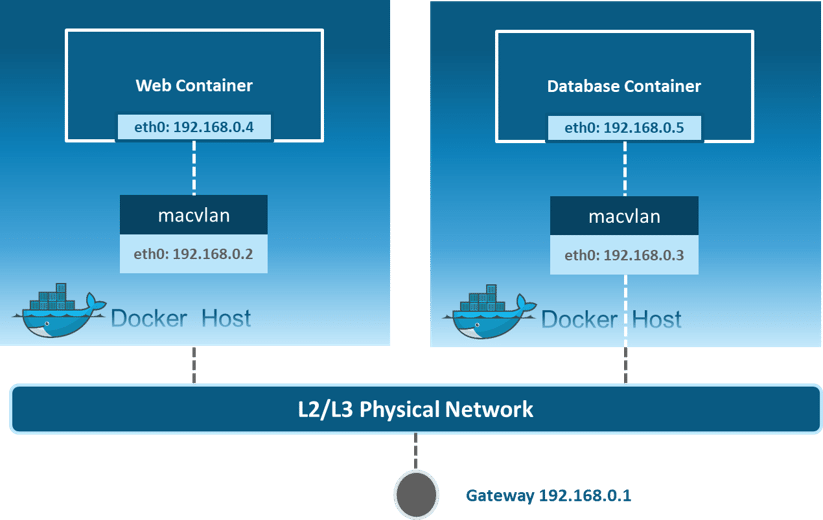
**None**: In this kind of network, containers are not attached to any network and do not have any access to the external network or other containers. So, this network is used when you want to completely disable the networking stack on a container and, only create a loopback device.

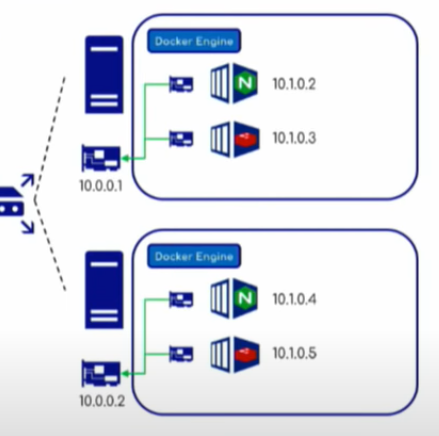
Ex: Docker run ubuntu -- network=none





**Macvlan:** Allows you to assign a MAC address to a container, making it appear as a physical device on your network. Then, the Docker daemon routes traffic to containers by their MAC addresses. Macvlan driver is the best choice when you are expected to be directly connected to the physical network, rather than routed through the Docker host’s network stack.





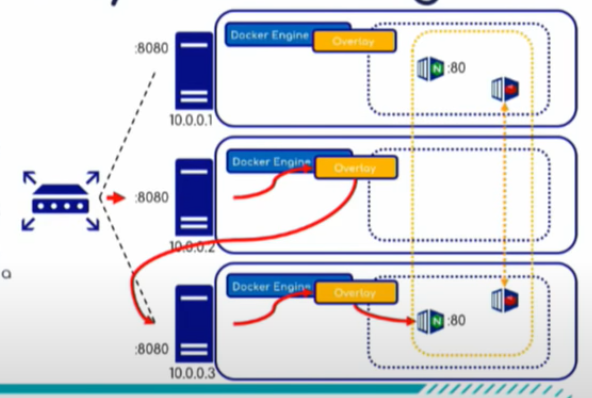
**Overlay**: Creates an internal private network that spans across all the nodes participating in the swarm cluster.

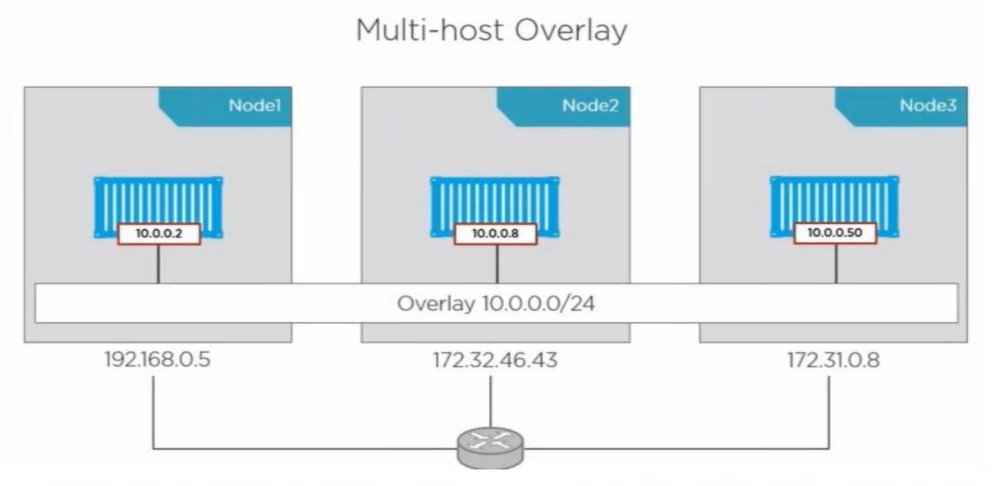


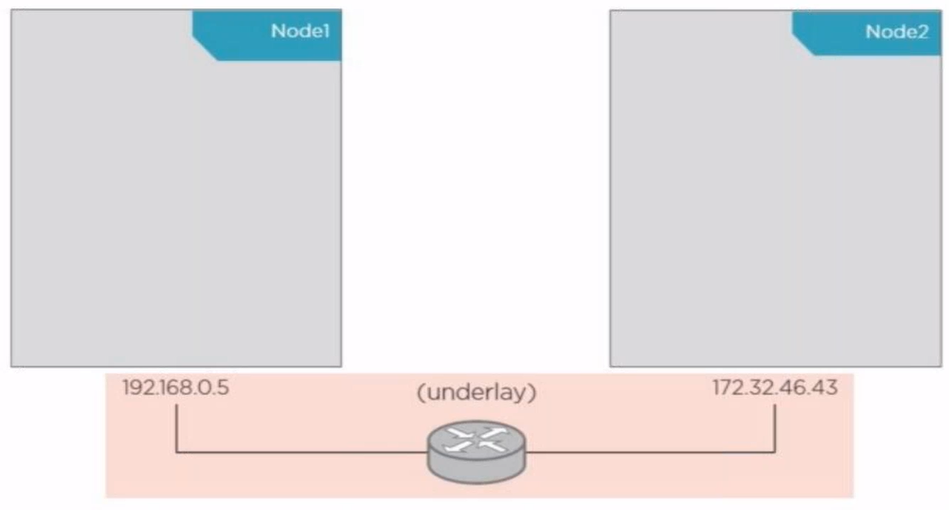
\*Containers can communicate with each other ,regardless of they are in diff hosts.

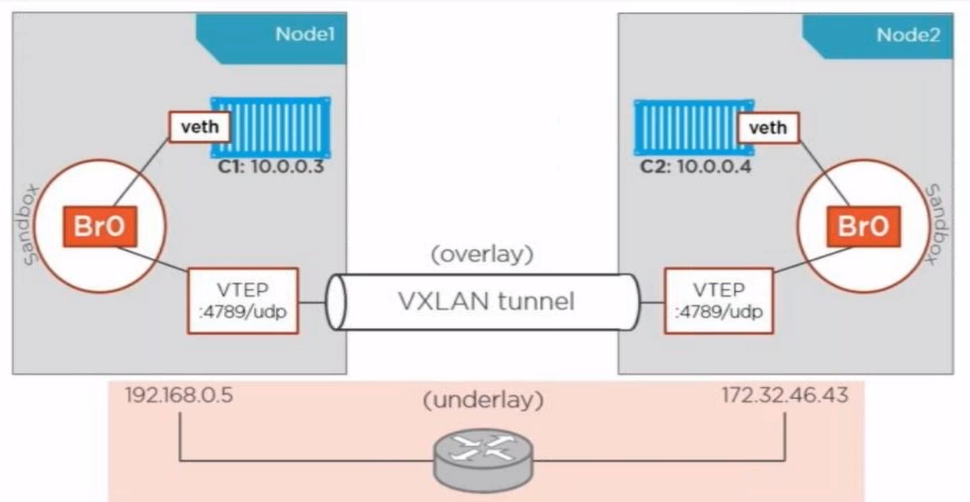
\*A VXLAN tunnel will be created and that establishes a communication.

\*Publishing a port applies to all nodes in the swarm cluster.Regardless of nodes connected.







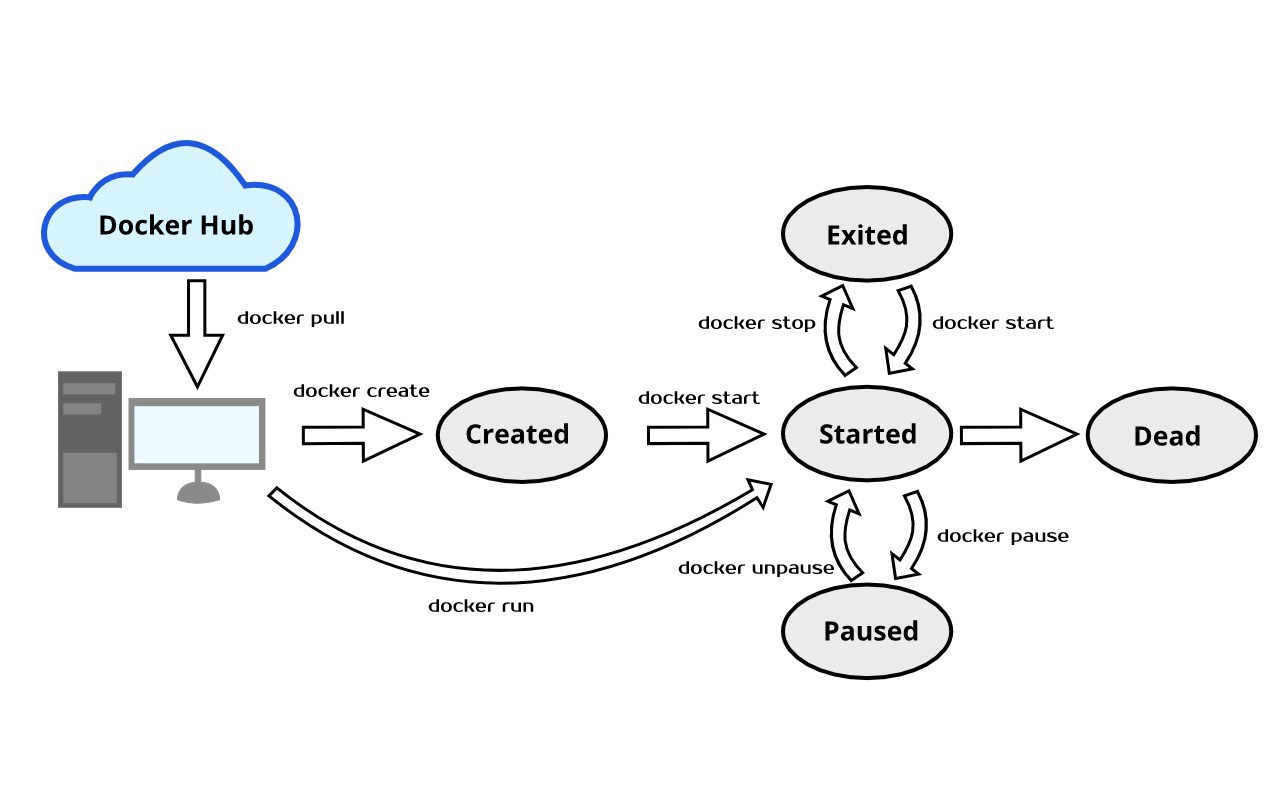


**Q. What is the lifecycle of Docker Container?**  
The life cycle of the Docker container is as below:

When we pull an image from the docker hub,docker daemon will save the image in the cache, and if we run a container using a “docker run image name” then it’ll go to started state,actually this command comprises of two “docker create & docker start”.

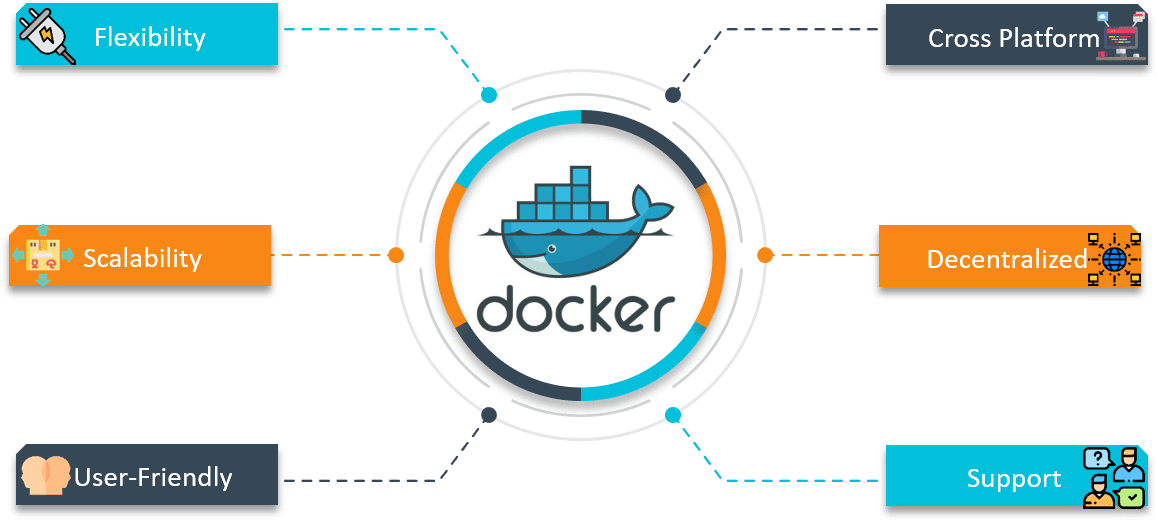
If we stop the running container using “docker stop “ it’ll go to exited state ,and when we start the stopped(Exited) container again by using “docker start” then it’ll go to Started state. If we pause the container by docker pause it’; go to “paused state” to un pause it we need to use “docker un-pause”

* Created
* Started
* Exited
* Paused
* Dead



<https://medium.com/future-vision/docker-lifecycle-tutorial-and-quickstart-guide-c5fd5b987e0d>

**Goals of Docker Networking?**



**Flexibility** – Docker provides flexibility by enabling any number of applications on various platforms to communicate with each other.

**Cross-Platform** – Docker can be easily used in cross-platform which works across various servers with the help of Docker Swarm Clusters.

**Scalability** – Docker is a fully distributed network, which enables applications to grow and scale individually while ensuring performance.

**Decentralized** –  Docker uses a decentralized network, which enables the capability to have the applications spread and highly available. In the event that a container or a host is suddenly missing from your pool of resource, you can either bring up an additional resource or pass over to services that are still available.

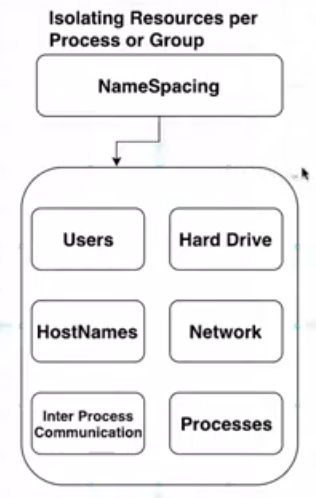
**User – Friendly** – Docker makes it easy to automate the deployment of services, making them easy to use in day-to-day life.

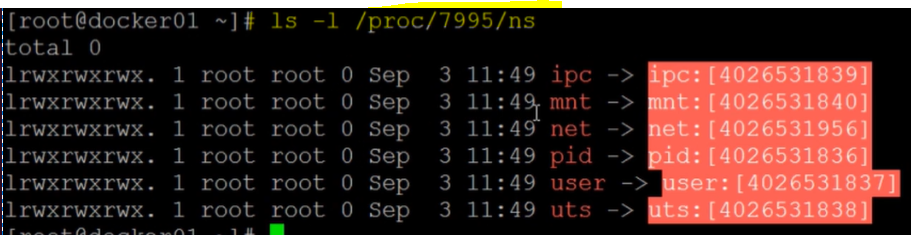
**Support** – Docker offers out-of-the-box supports. So, the ability to use Docker Enterprise Edition and get all of the functionality very easy and straightforward, makes Docker platform to be very easy to be used.

**What are the different kinds of namespaces available in a Container?**

Docker uses a technology called namespaces to provide the isolated workspace called the container.When you run a container, Docker creates a set of namespaces for that container.

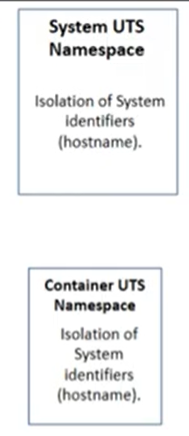
These namespaces provide a layer of isolation.



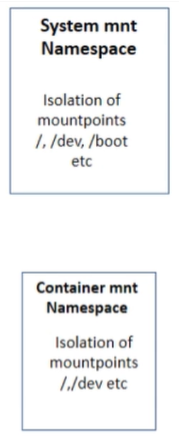


Docker Engine uses namespaces such as the following on Linux:

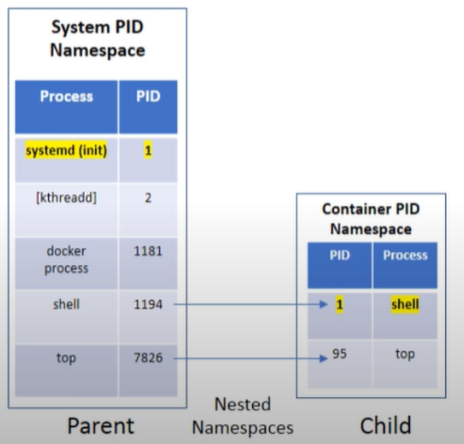
1. **UTS Namespace (uts)**: UTS stands for Unix Timesharing System. In UTS namespace every container gets its own hostname and domain name.



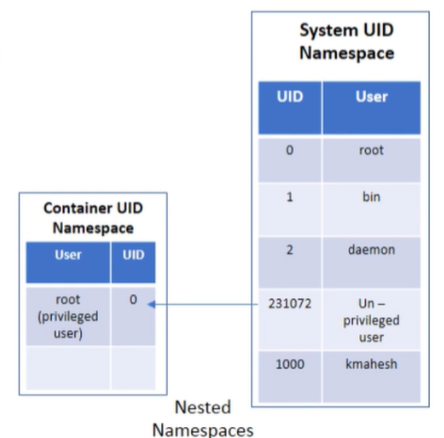
1. **Mount Namespace(mnt)** : This namespace provides its own file system within a container. With this namespace we get root like / in the file system on which rest of the file structure is based.



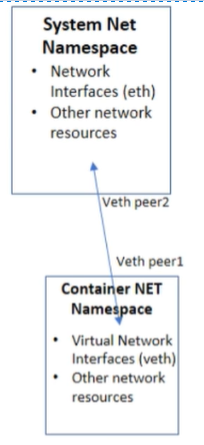
1. **PID Namespace** : This namespace is used to uniquely identify a processes.(Container process Id’s are the child of host process Id’s)



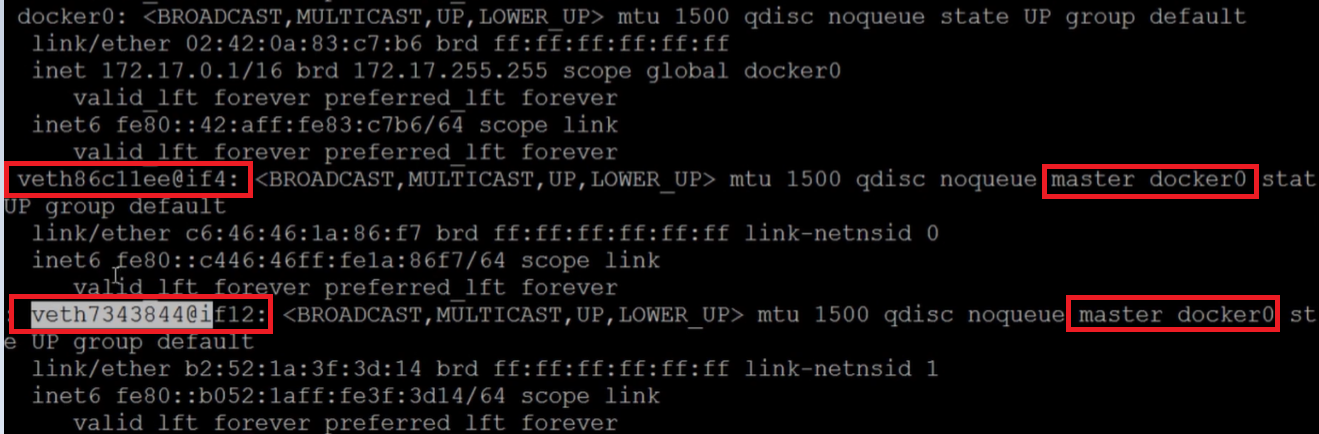
1. **IPC Namespace** : IPC stands for Inter Process Communication. This namespace covers shared memory, semaphores, named pipes etc resources that are shared by processes. The items in this namespace do not cross the container boundary.
2. **User Namespace (UID)**: This namespace contains the users and groups that are defined within a container.



1. **Network Namespace** : With this namespace, container gets its own network resources like- ports, devices etc. With this namespace, Docker creates an independent network stack within each container.







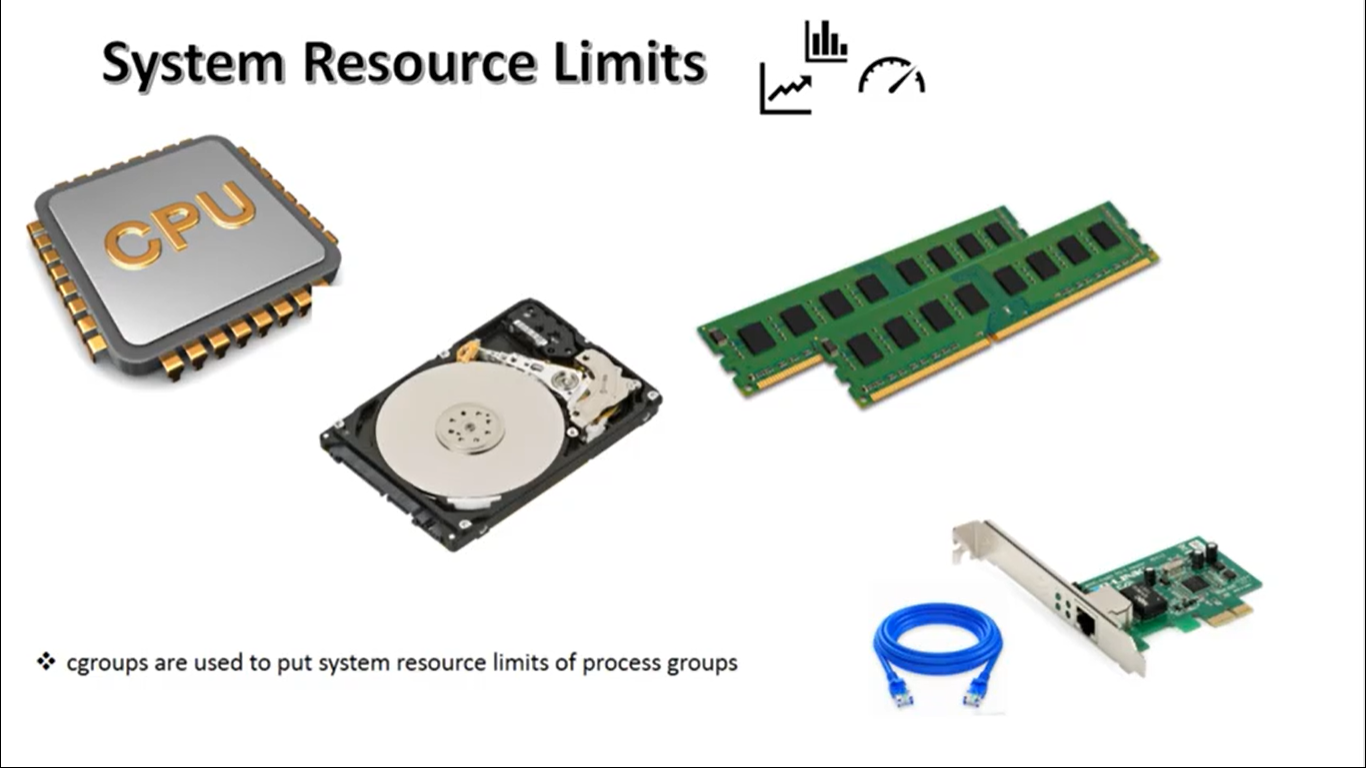
**90. How will you monitor Docker in production?**

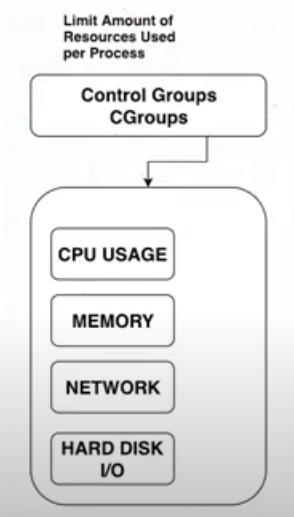
Docker provides tools like docker stats and docker events to monitor Docker in production. We can get reports on important statistics with these commands.

**Docker stats** : When we call docker stats with a container id, we get the CPU, memory usage etc of a container. It is similar to top command in Linux. (resource utilization)

**Docker events** : Docker events are a command to see the stream of activities that are going on in Docker daemon. Some of the common Docker events are: attach, commit, die, detach, rename, destroy etc.  
We can also use various options to limit or filter the events that we are interested in.

**Control groups**: It allows us to limit resources such as cpu,memory,network bandwidth etc for containers.





. A cgroup limits resources to a specific app.

**Capabilities:**

For better security, Docker provides an option to run a container process under non-root user, using a USER directive inside a Dockerfile

[https://blog.trendmicro.com/trendlabs-security-intelligence/why-running-a-privileged-container-in-docker-is-a-bad idea/#:~:text=By%20David%20Fiser%20and%20Alfredo,not%20accessible%20in%20ordinary%20containers.](https://blog.trendmicro.com/trendlabs-security-intelligence/why-running-a-privileged-container-in-docker-is-a-bad%20idea/#:~:text=By%20David%20Fiser%20and%20Alfredo,not%20accessible%20in%20ordinary%20containers.)

++Ctrl pq—to comeout of container

++set 🡪 to see environment var

**++LXC containers**: LXC is an operating-system-level virtualization method for running multiple isolated Linux systems on a control host using a single Linux kernel.

++Same OS kernel :: We can run containers with any os if the kernel is same.(Linux)

++docker.io/library/myimage—**Default**

## Clean up (--rm)

By default a container’s file system persists even after the container exits. This makes debugging a lot easier (since you can inspect the final state) and you retain all your data by default. But if you are running short-term **foreground** processes, these container file systems can really pile up. **If instead you’d like Docker to automatically clean up the container and remove the file system when the container exits, you can add the --rm flag:**

--rm=false: Automatically remove the container when it exits

**++ps –eaf**

**++ps –aux**

**docker cp script.sh youthful\_chaplygin:/tmp/script.sh**

script.sh –filename

youthful\_chaplygin—Container name

-v : Volume

-p :: publish

-e==Env variable

--name

-d :: detached mode

MYSQL\_ROOT\_PASSWORD=db\_pass123

**Docker compose**:: It is a tool for defining and running multi-container docker application.

\*\* On a single host

Version

Service

images

**Note** :: We can run diff flavours of OS containers ,provided the underlying OS kernel is of same(Linux)

Cat/etc/\*release\*

Uname -r