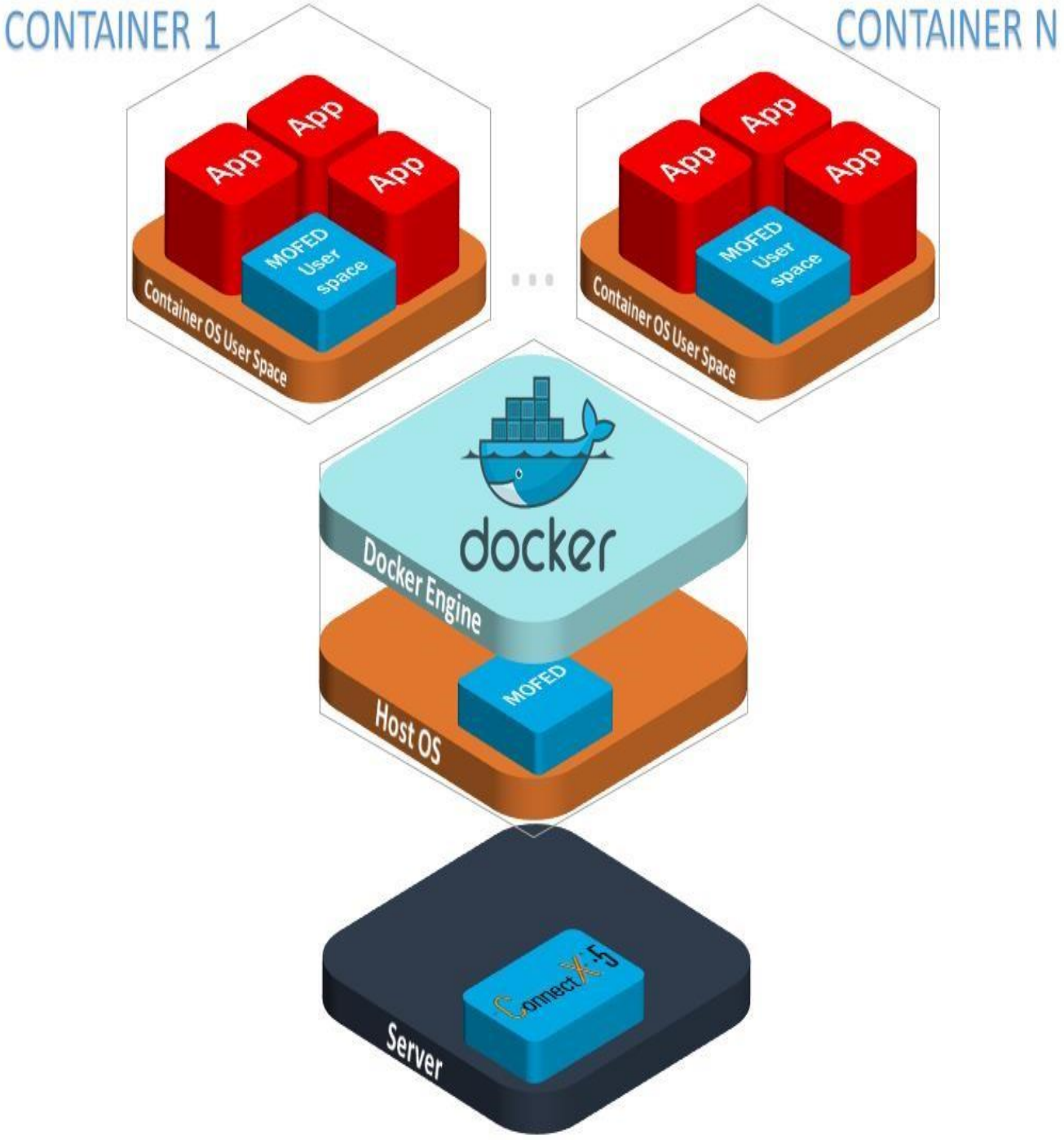
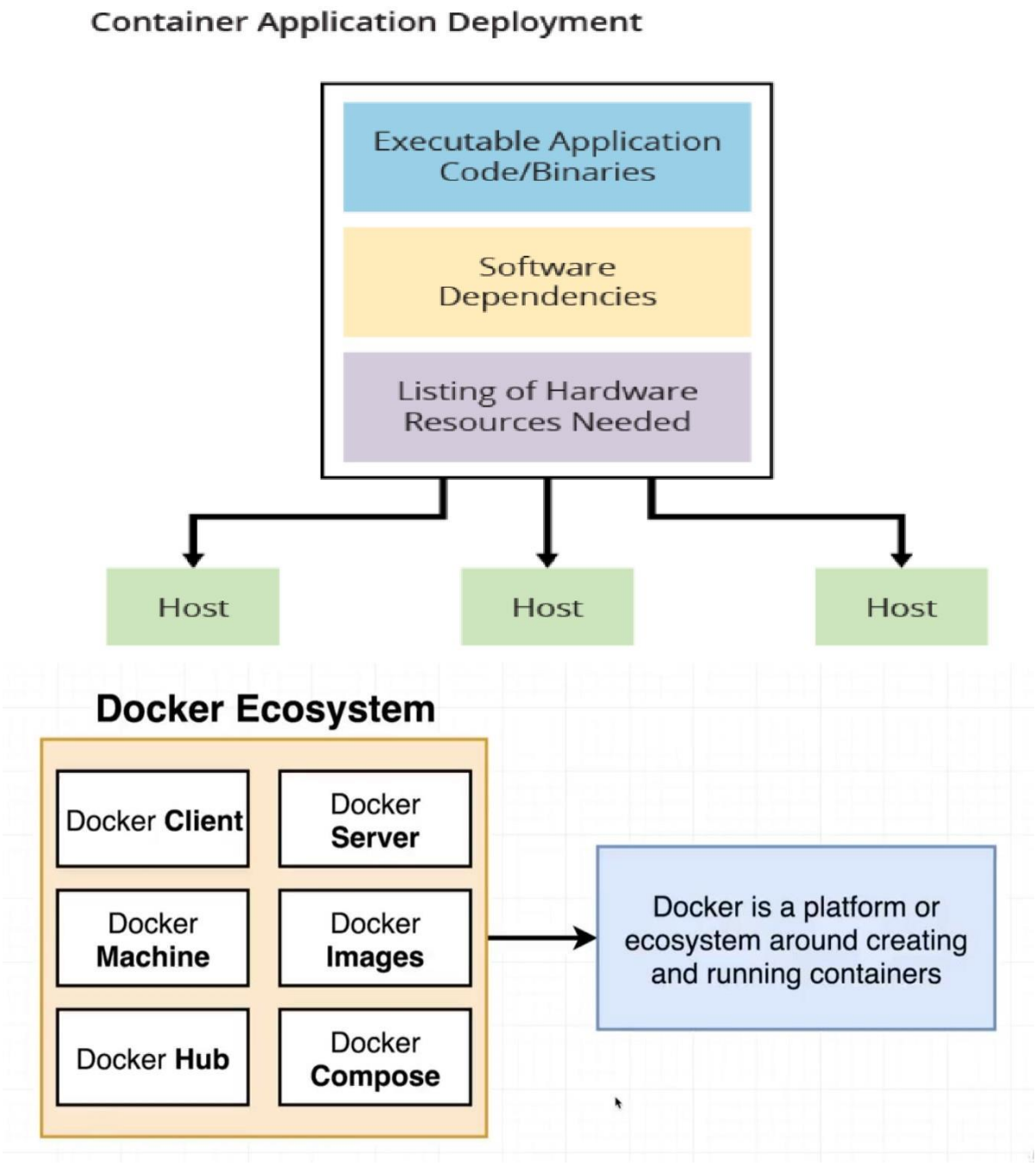

CONTAINER 1
CONTAINER N
App
App
App
MOFED User space
Container OS User Space
App
App
App
MOFED User space
Container OS User Space
docker
Docker Engine
MOFED
Host OS
Server

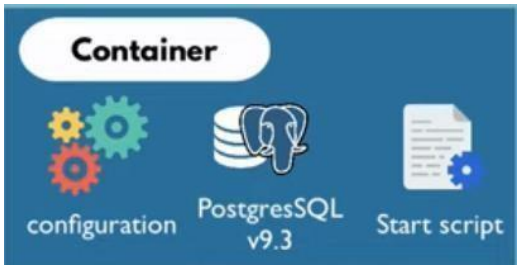
Q. Docker Vs VM (Virtual Machine)

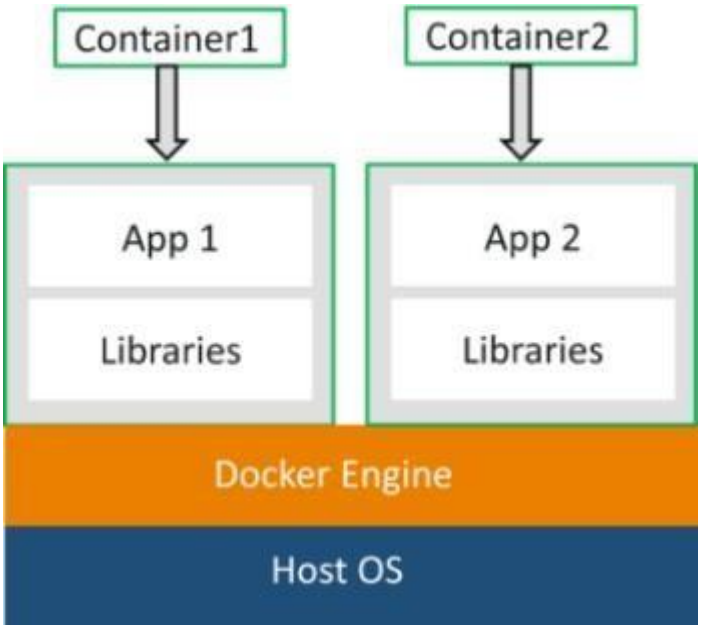
Virtual Machines Vs Docker Containers	
Virtual Machines	**Docker Containers**
Need more resources	Less resources are used
Process isolation is done at hardware level	Process Isolation is done at Operating System level
Separate Operating System for each VM	Operating System resources can be shared within Docker
VMs can be customized.	Custom container setup is easy
Takes time to create Virtual Machine	Creation of docker is very quick
Booting takes minutes	Booting is done within seconds

Q. What is Docker?



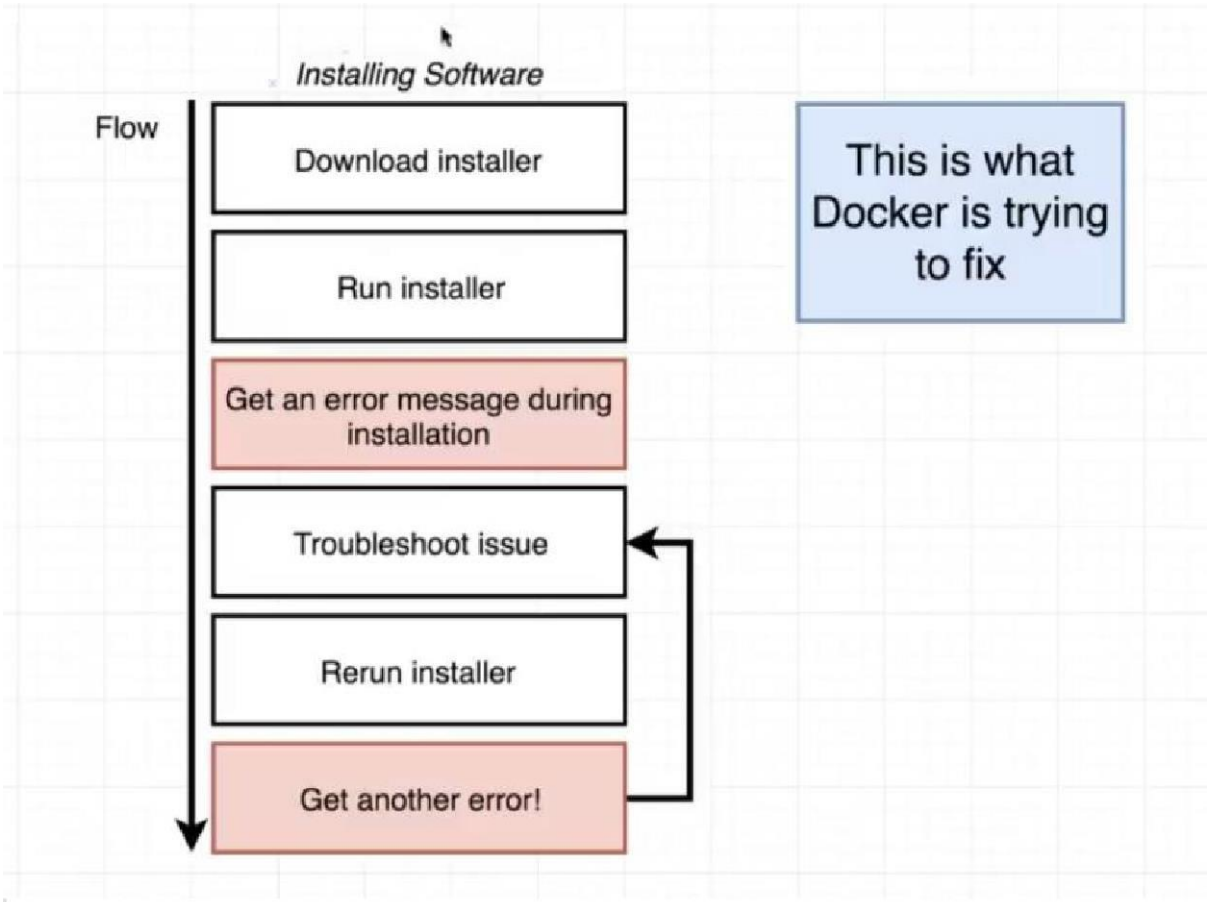
- Docker is a containerization platform tool for building an application in which container contains packaged binary files and its library files with its required dependencies, and further these containers are easily shipped to run on other machines.
- Each and every application runs on separate containers and has its own set of dependencies & libraries.
- This makes sure that each application is independent of other applications, giving developers surety that they can build applications that will not interfere with one another.
- Docker is a tool designed to make it easier to create, deploy and run applications by using containers.
- In simple words, Docker is a tool which is used to automate the deployment of applications in lightweight containers so that applications can work efficiently in different environments

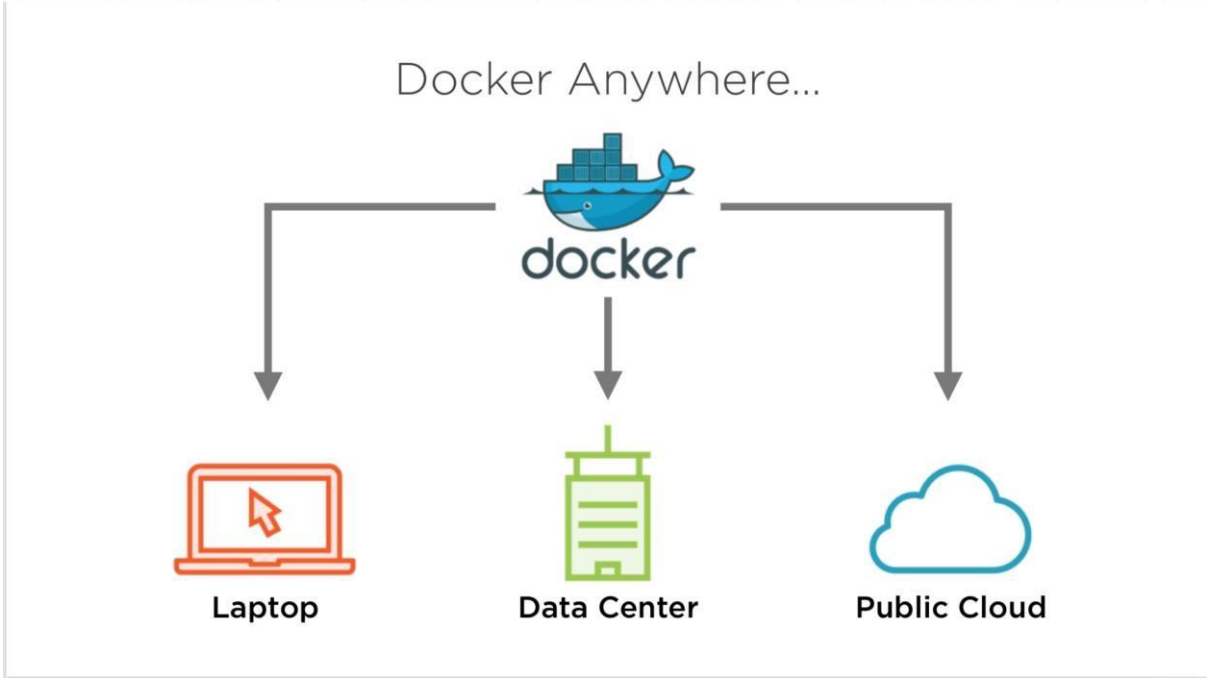




Note: Container is a software package that consists of all the dependencies required to run an application

What is the purpose of Docker?





- The purpose of Docker is to help developers and dev-ops team in becoming more productive and less error prone.
- Setup and deployment of new projects becomes much more easier and time efficient with the help of Docker.
- Consider a scenario where operating system windows is installed in your system and you have to deploy and test your application in different operating system let's say fedora, centos and ubuntu. How will you do that? This is where Docker comes to your rescue.
- Again consider a scenario where you have to test your application with different php versions let's say php 7.1, php 7.2 and php 7.3 and using different web server

combinations such as nginx and apache. How will you do that? Doesn't that seem complicated to you? This is where Docker comes to your rescue.

Let's use Docker with an example!

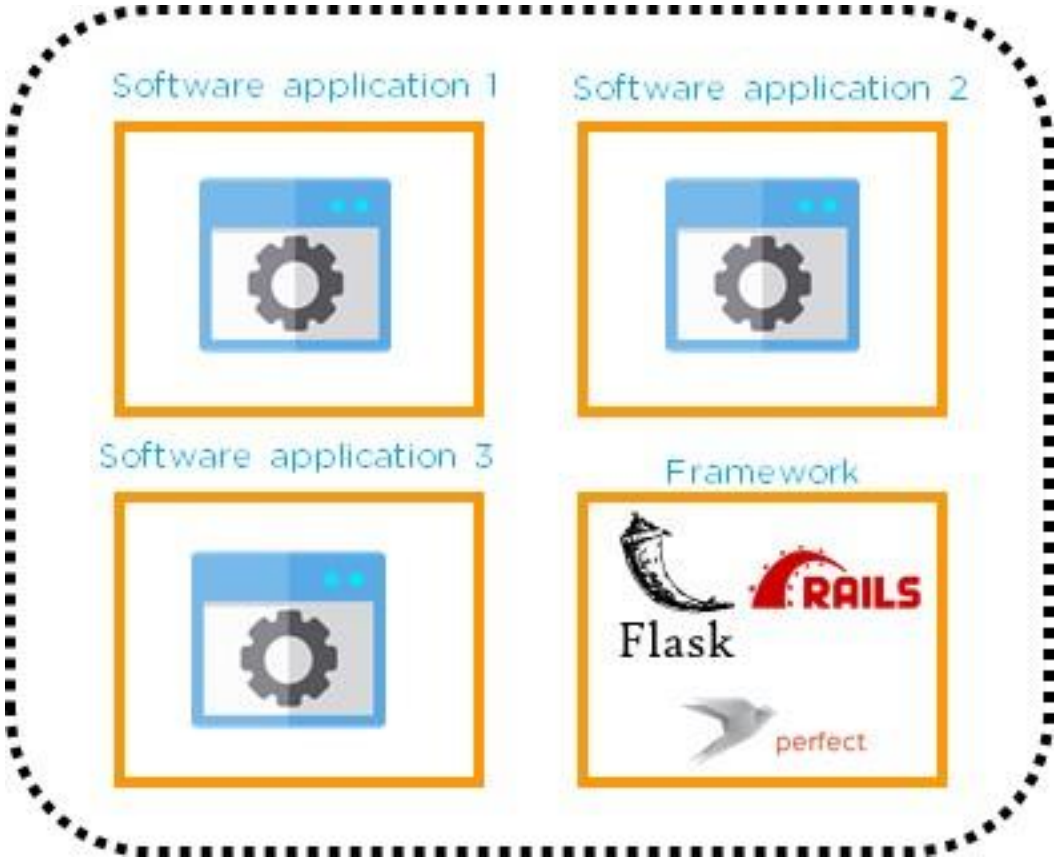
Image a situation where you plan to rent a house in Airbnb

But, in the house, there are 3 rooms and only one cupboard and kitchen

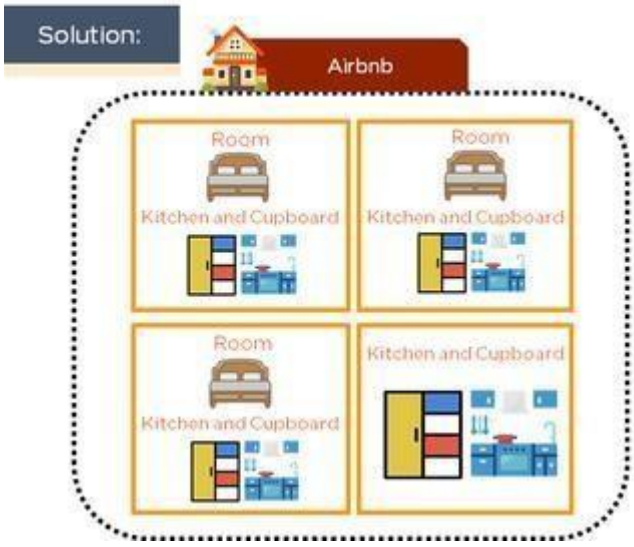
And none of the guests are ready to share the cupboard and the kitchen

Because every individual has different preferences when it comes to the cupboard and the kitchen usage

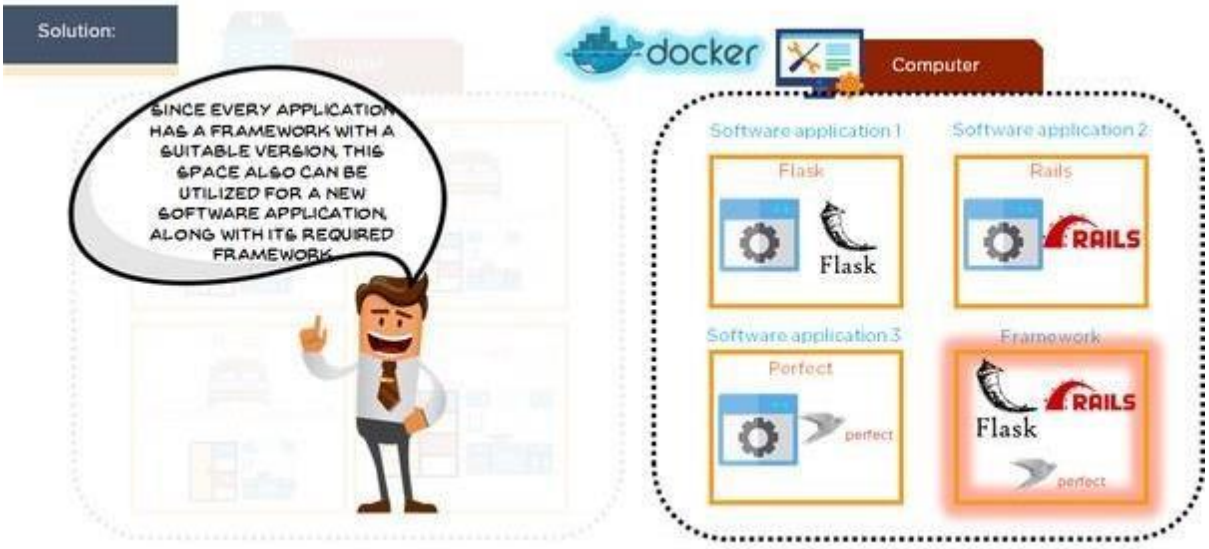
Let's use this example with computers, where all the three applications use different frameworks



Problem statement: But, what if a person wants to run all his applications with their suitable frameworks?
Solution: Docker will help you run applications with their suitable

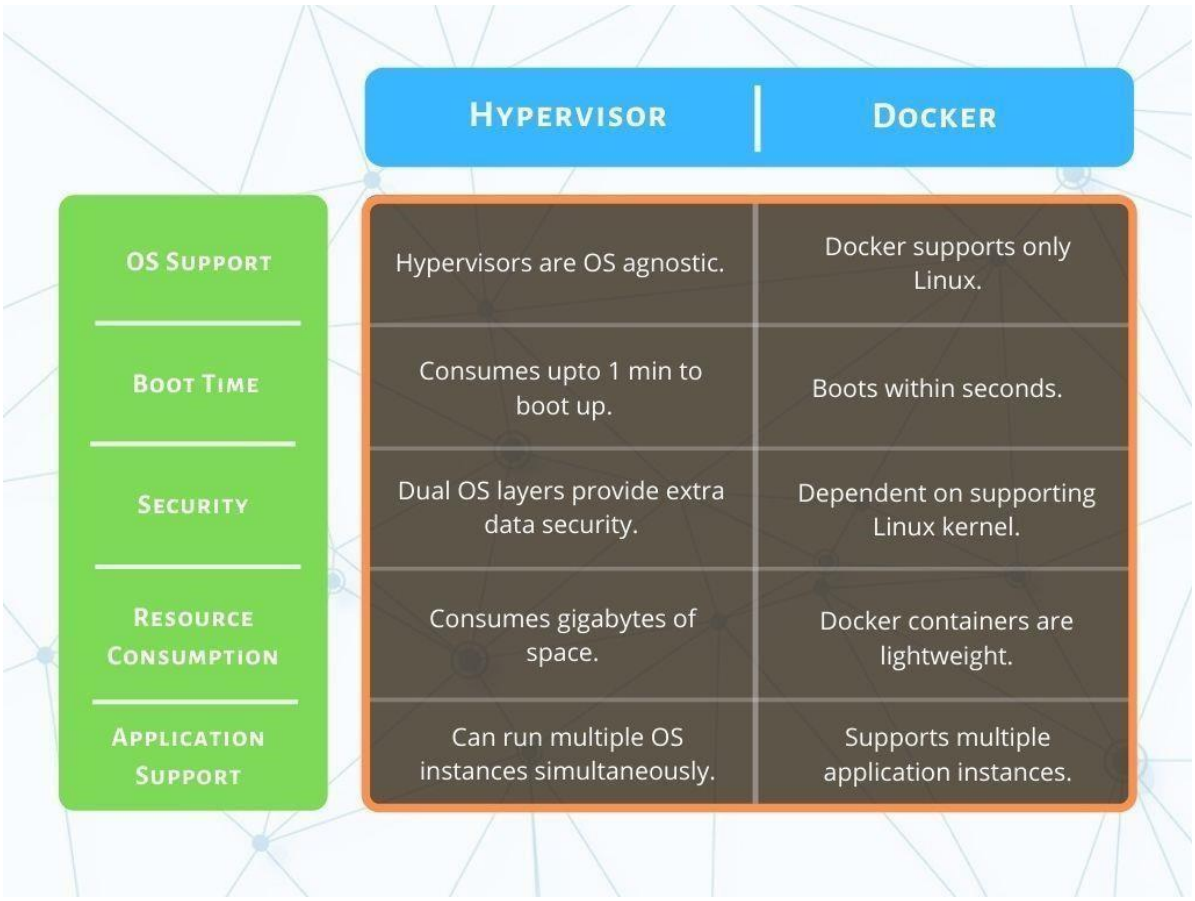


Likewise,

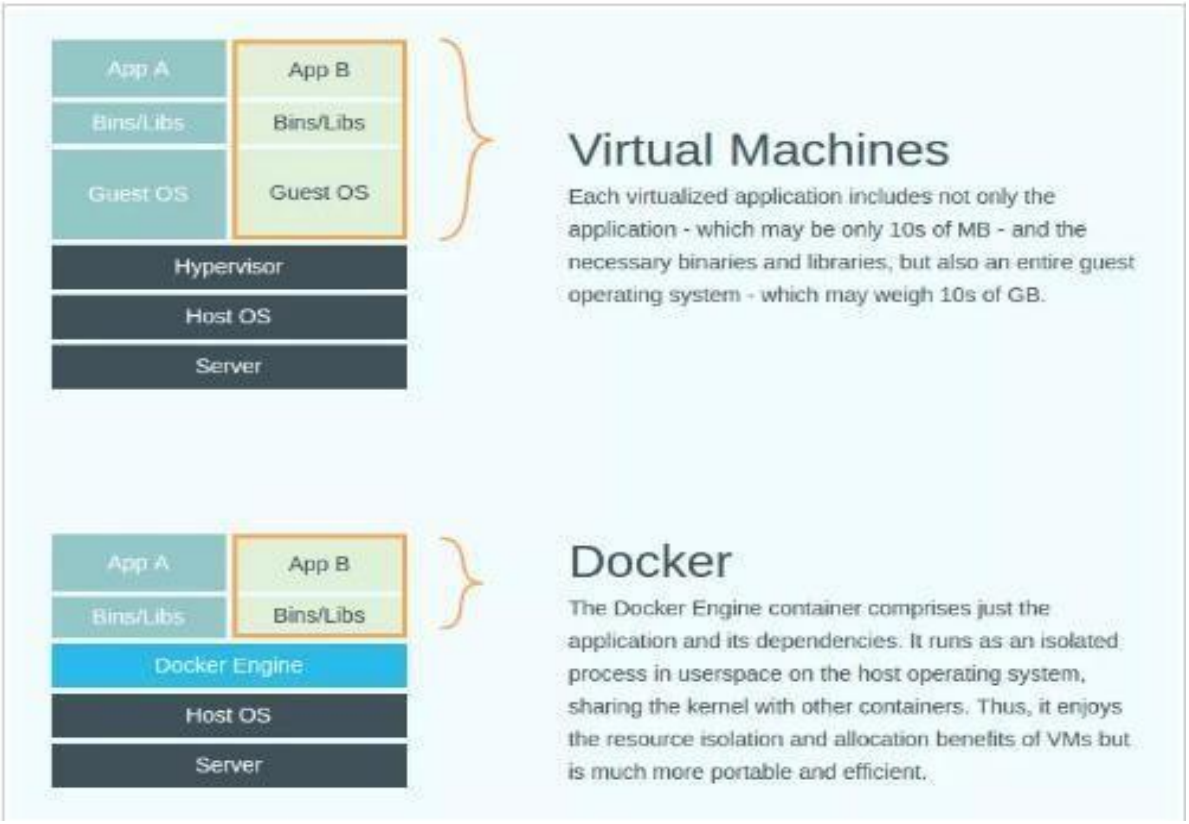


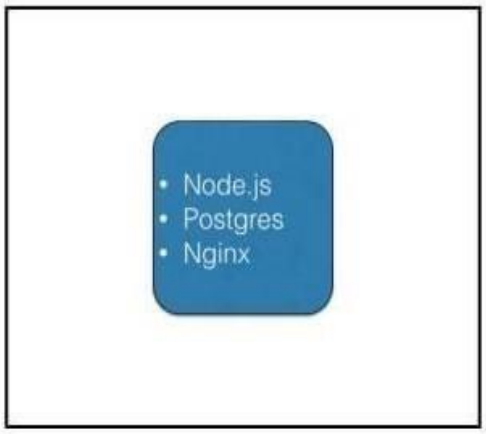
As a result, Docker makes more efficient use of system resources

Q. What is the advantage of Docker over hypervisors?



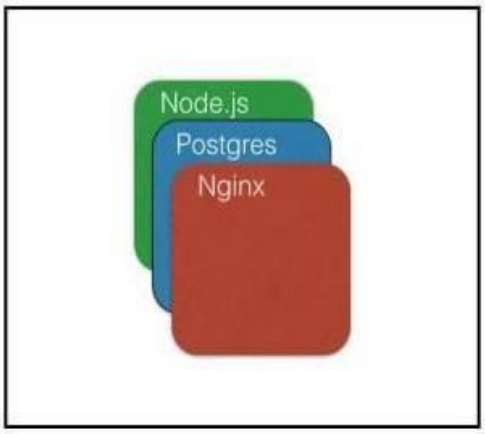
- Docker is light weight and more efficient in terms of resource uses because it uses the host underlying kernel rather than creating its own hypervisor.
- To run an application in a virtualized environment (e.g., vSphere), we first need to create a VM, install an OS inside and only then deploy the application.
- To run the same application in docker, all you need is to deploy that application in Docker.
- There is no need for additional OS layer. You just deploy the application with its dependent libraries, docker engine (kernel, etc.) provides the rest.





OS containers

- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones



App containers

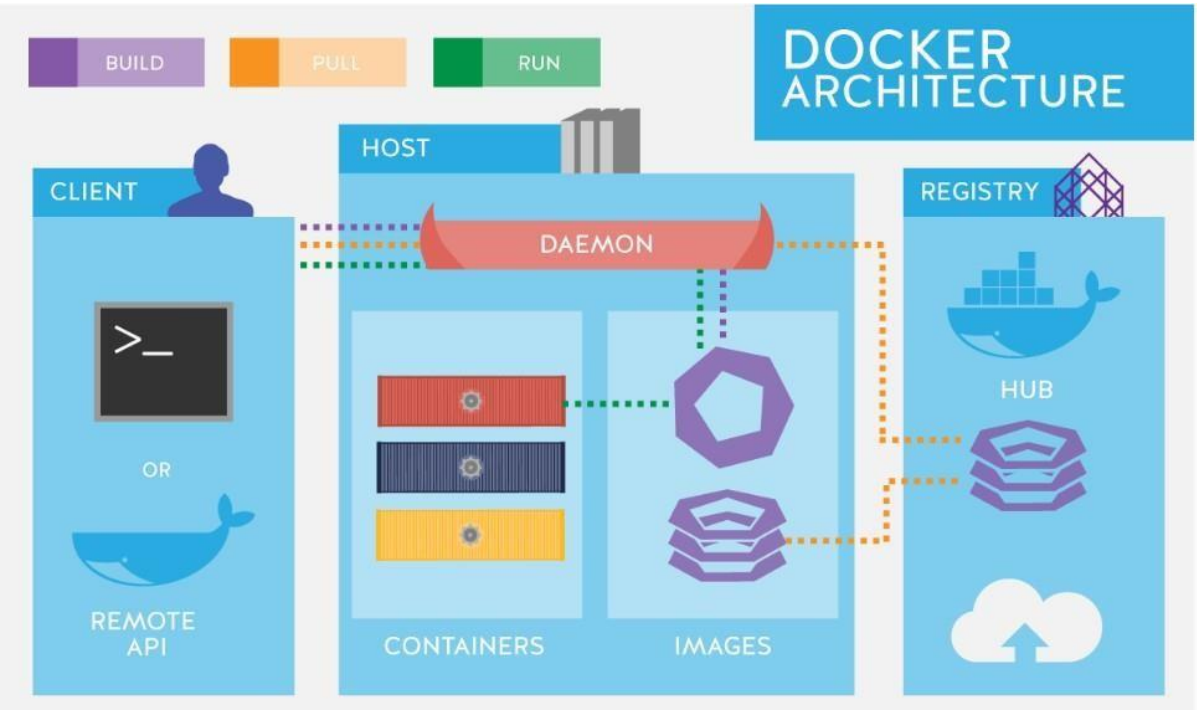
- Meant to run for a single service
- Layered filesystems
- Built on top of OS container technologies
- Examples - Docker, Rocket

Q. How is Docker different from other container technologies?

- Docker containers are very easy to deploy in any cloud platform.
- It can get more applications running on the same hardware when compared to other technologies, it makes it easy for developers to quickly create, ready-to-run containerized applications and it makes managing and deploying applications much easier.
- You can even share containers with your applications. With other containerization methods, these features are not possible.
- There is no limitation on running Docker as the underlying infrastructure can be your laptop or else your Organization's Public / Private cloud space
- Docker with its Container HUB forms the repository of all the containers that you are ever going to work, use and download.
- Sharing of applications is as well possible with the Containers that you create.
- Docker is one of the best-documented technologies available in the Containerization space.

Docker Architecture

- Docker uses a client-server architecture.
- The Docker client consists of Docker build, Docker pull, and Docker run.
- The client approaches the Docker daemon that further helps in building, running, and distributing Docker containers.
- Docker client and Docker daemon can be operated on the same system; otherwise, we can connect the Docker client to the remote Docker daemon. Both communicate with each other using the REST API, over UNIX sockets or a network.



The basic architecture in Docker consists of three parts:

- Docker Client
- Docker Host
- Docker Registry

Docker Client

- It is the primary way for many Docker users to interact with Docker.
- It uses command-line utility or other tools that use Docker API to communicate with the Docker daemon.
- A Docker client can communicate with more than one Docker daemon.

Docker Host

In Docker host, we have Docker daemon and Docker objects such as containers and images. First, let's understand the objects on the Docker host, then we will proceed toward the functioning of the Docker daemon.

- **Docker Objects:** ○ **What is a Docker image?** A Docker image is a type of recipe/template that can be used for creating Docker containers. It includes steps for creating the necessary software.
    - ○ **What is a Docker container?** A type of virtual machine created from the instructions found within the Docker image. It is a running instance

of a Docker image that consists of the entire package required to run an application.

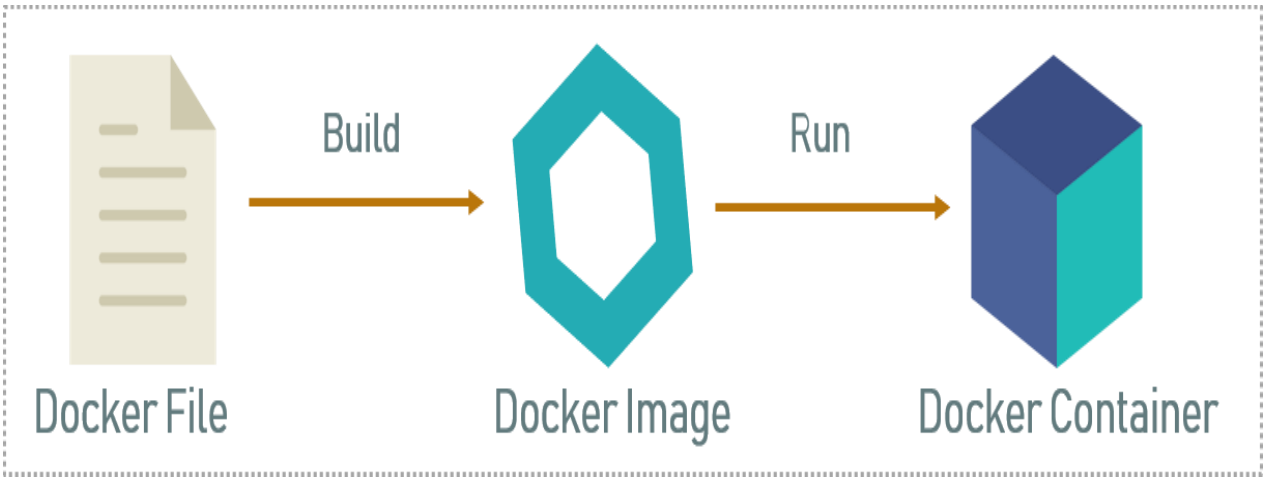
Docker Daemon

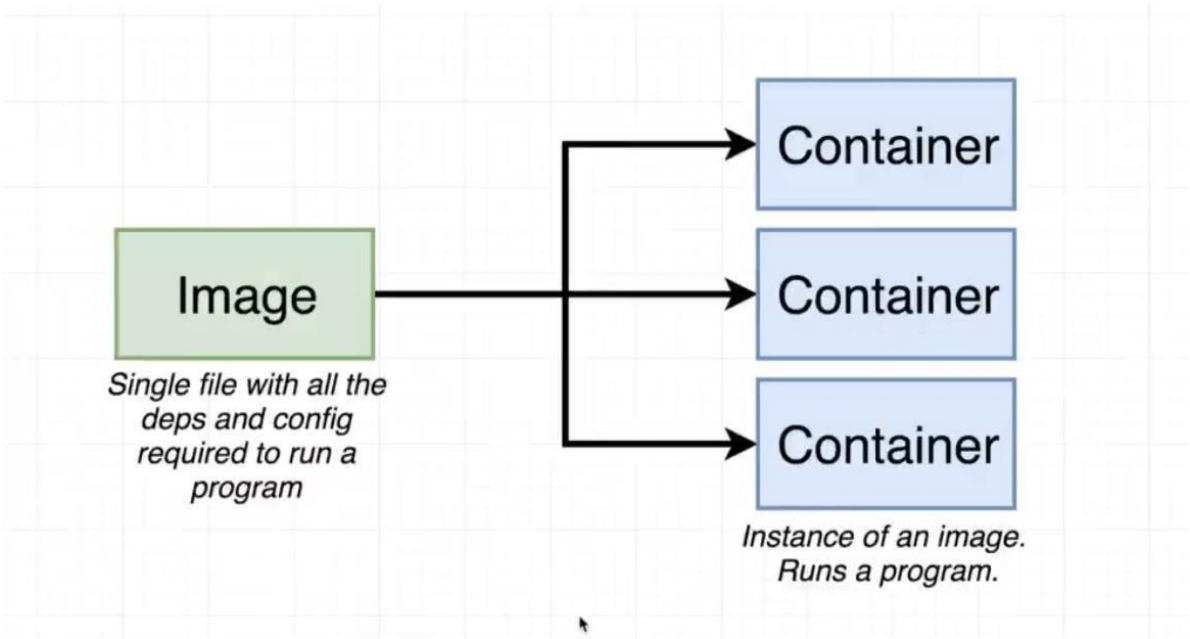
- Docker daemon helps in listening requests for the Docker API and in managing Docker objects such as images, containers, volumes, etc. Daemon issues to build an image based on a user's input and then saves it in the registry.
- In case we don't want to create an image, then we can simply pull an image from the Docker hub (which might be built by some other user). In case we want to create a running instance of our Docker image, then we need to issue a run command that would create a Docker container.
- A Docker daemon can communicate with other daemons to manage Docker services.

Docker Registry

- Docker registry is a repository for Docker images which is used for creating Docker containers.
- We can use a local/private registry or the Docker hub, which is the most popular social example of a Docker repository.

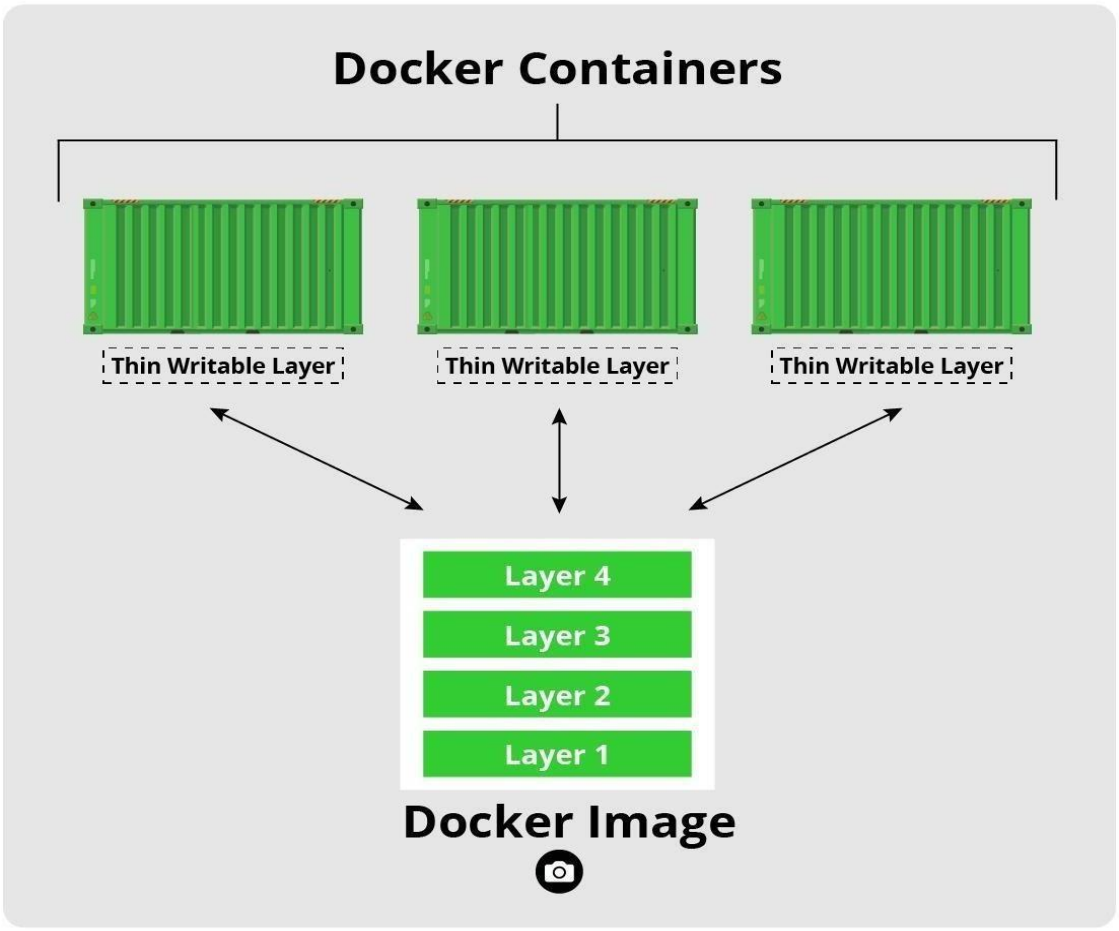
Q. What is Docker image?





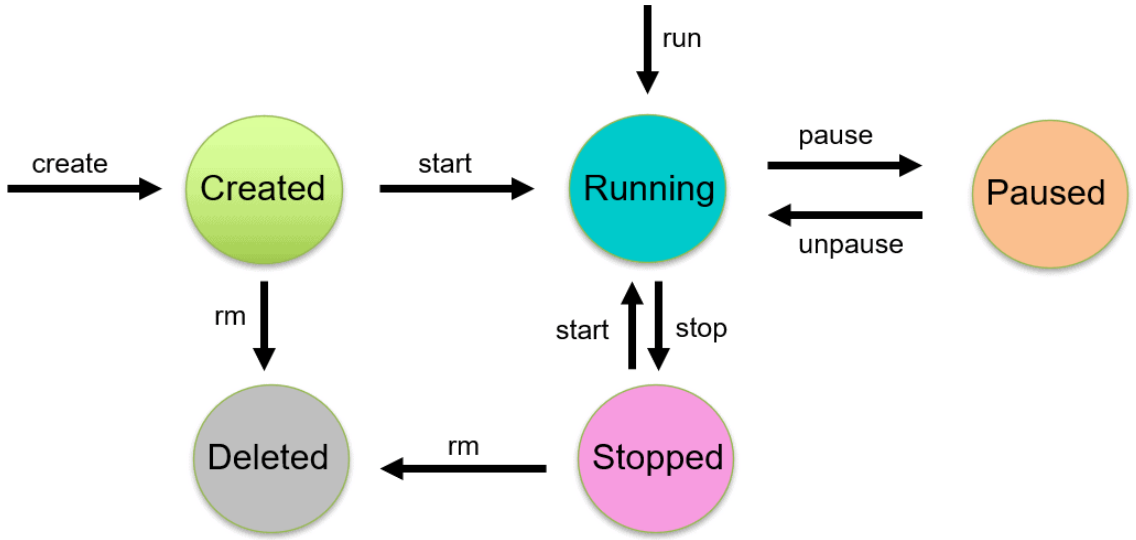
- **A Docker image is a snapshot, or template, from which new containers can be started.**
- Docker images are stopped containers (or classes if you're a developer).
- We can stop a container and create a new image from it.
- Docker images are considered build time constructs whereas containers are runtime constructs.
- It's a representation of a filesystem plus libraries for a given OS.
- **Images are created and stored in layered fashion.**
- All the images downloaded from the Docker hub present in the Dockerhost.
- If an image specified in the Docker run command is not present in the Docker host, by default, the Docker daemon will download the image from the Docker public registry (Docker hub).
- Container is a writable layer on top on an image.
- A new image can be created by executing a set of commands from Dockerfile (it's also possible but not recommended to take a snapshot from a running container).

Q. What is Docker container?



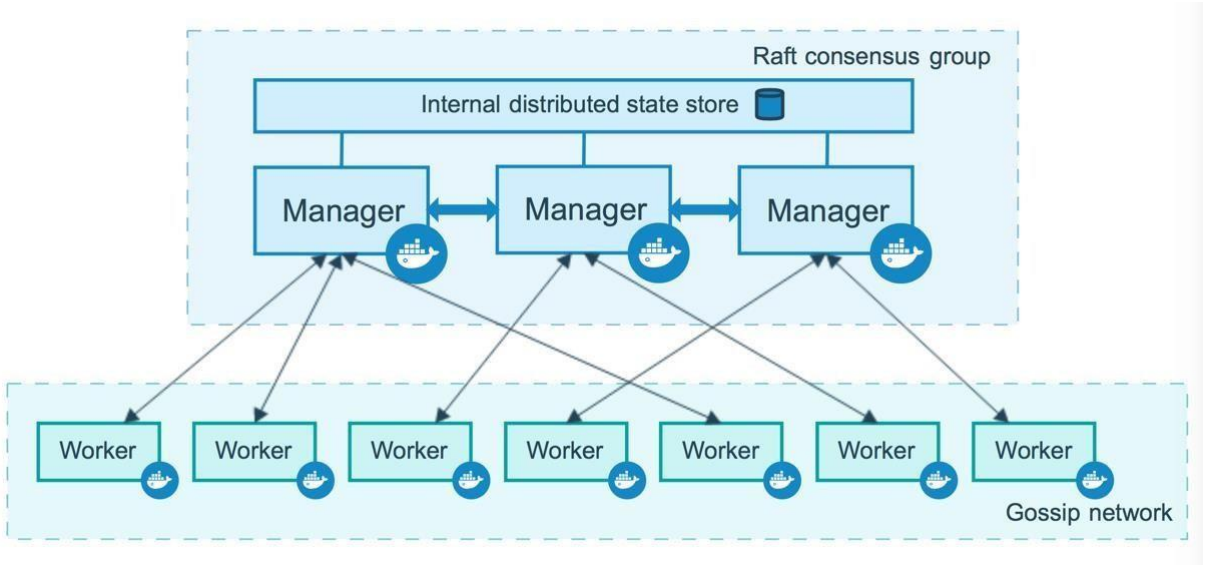
- Docker containers are basically runtime instances of Docker images.
- Docker is a tool designed to make it easier to create, deploy and run applications by using containers.
- Docker containers will contain all the binaries and libraries required for application or microservice. So, application is present in a container, or we have containerized application. Now, that same container can be used in the Test and Prod environment.
- Docker Containers are a lightweight solution to Virtual Machines, and it uses the host OS. The best part, you don't have to pre-allocate any RAM to the Docker Container, it will take it as and when required. So, with Docker Container I don't have to worry about wastage of resources.

Q. What is the lifecycle of Docker Container?



The life cycle of the Docker container is as below:

- Create a container.
- Run the Docker container.
- Pause the Container.
- Un-pause the Container.
- Start the Container.
- Stop the Container.
- Restart the Container.
- Kill the Container.
- Destroy the Container. **Q. What is Docker hub?**
- Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities.
- You can also upload your own Docker built images to Docker hub. **Q. What is Docker Swarm?**



- Docker Swarm as the way of managing (orchestrating) the Docker containers.
- nifty containers need to be orchestrated
- The problem is, when you have lots of containers running, they need to be managed: you want enough of them running to handle the load, but not so many they are bogging down

the machines in the cluster. And well, from time to time, containers are going to crash, and need to be restarted.

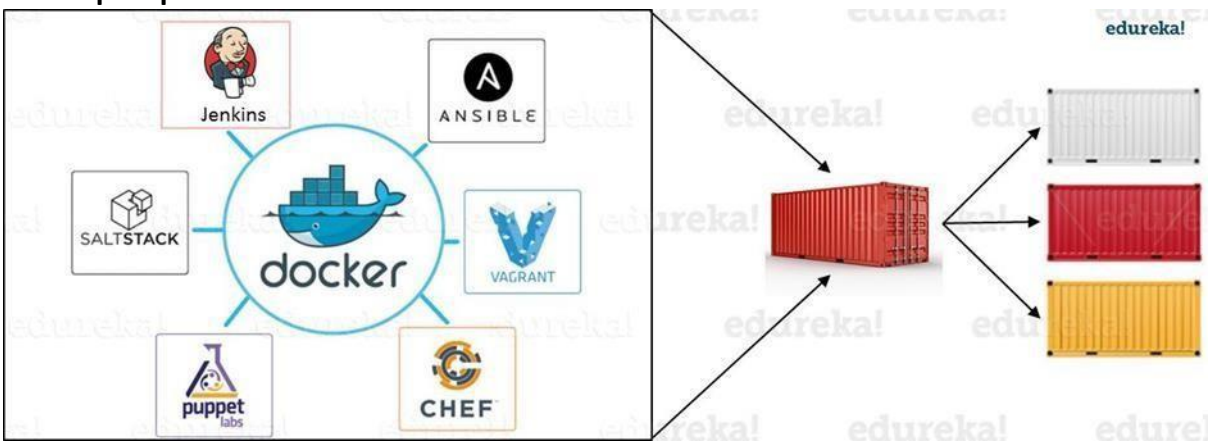
- Need to Define nodes, define services, Set how many replica (the "desired state") you want to run and where to run with these details in Docker Swarm mode it will manage the cluster
- There are two types of nodes: manager nodes (which you use to define services), and worker nodes (which are told what to do by manager nodes based on your service definitions).
- We have to tell docker Swarm how many desired state for the service you want Then you submit a service definition to a manager node. The service definition consists of one or more Tasks (which are the atomic unit of scheduling in Docker Swarm mode), and how many replicas of that service you want to run on the cluster. That was easy.

Q. What is Dockerfile used for?

- A Dockerfile is a text document that contains instructions on how to build your own images.
- Dockerfile is the build instructions to build the image.

Q. Can I use JSON instead of YAML for my compose file in Docker?

- YES, we can use JSON instead of the default YAML for your Docker compose file.
- To use JSON file with compose need to specify the filename to use as the following:
- docker-compose -f docker-compose.json up **Q. Tell us how you have used Docker in your past position?**



- integrated docker with Jenkins.
- integrated docker with Ansible
- integrated docker with terraform **Q. How to create Docker container?**
- Docker container can be created by Docker image `docker run -t -i <command name>`
- The command above will create the container.
- To check whether the Docker container is created and whether it is running or not, you could make use of the following command.
- This command will list out all the Docker containers along with its statuses on the host that the Docker container runs.

  `docker ps -a`

Q. How to stop and restart the Docker container?

- The following command can be used to stop a certain Docker container with

the container | docker stop | CONTAINER_ID |

- The following command can be used to restart a certain Docker container with the container id as docker restart CONTAINER_ID

How far do Docker containers scale? Are there any requirements for the same?

- Large web deployments like Google and Twitter and platform providers such as Heroku and dotCloud, all run on container technology.
- Containers can be scaled to hundreds of thousands or even millions of them running in parallel.
- Talking about requirements, containers require the memory and the OS at all the times and a way to use this memory efficiently when scaled. **Q. What platforms does Docker run on?**

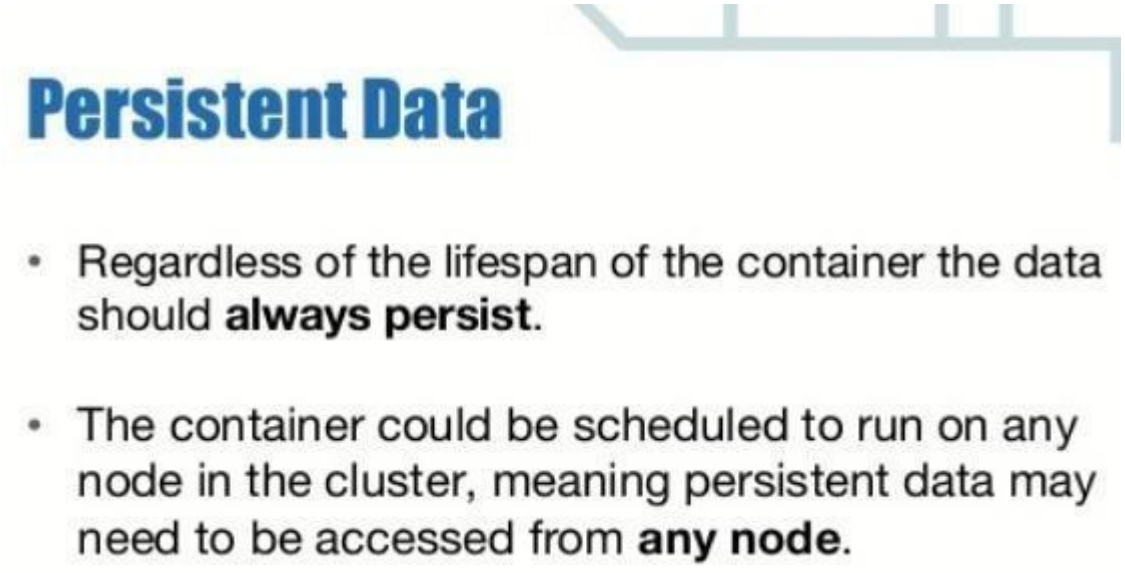
Docker is currently available on the following platforms and also on the following Vendors or Linux:

- Ubuntu 12.04, 13.04
- Fedora 19/20+
- RHEL 6.5+
- CentOS 6+
- Gentoo
- ArchLinux
- openSUSE 12.3+
- CRUX 3.0+

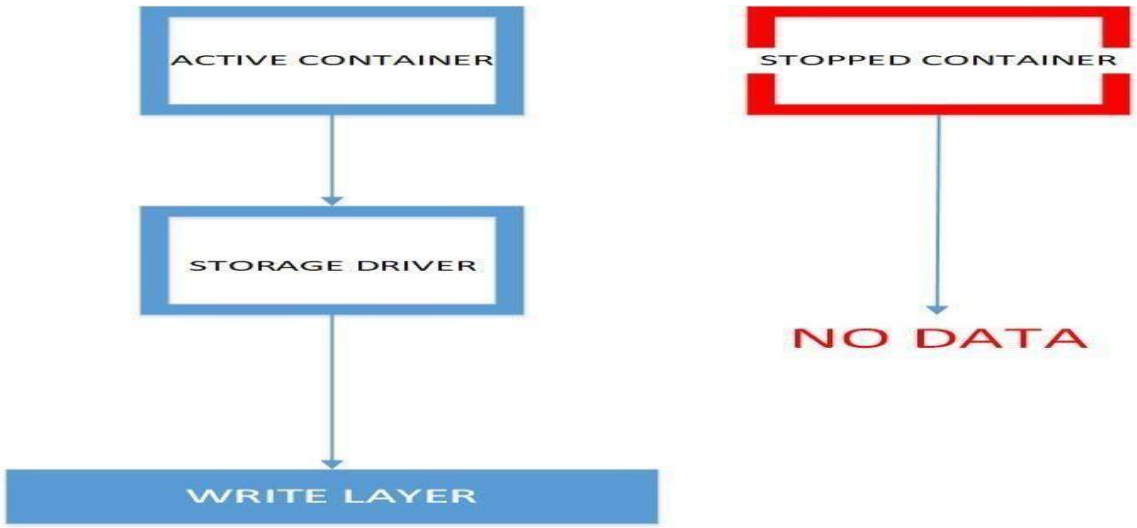
Docker is currently available and also is able to run on the following Cloud environment setups given as below: • Amazon EC2

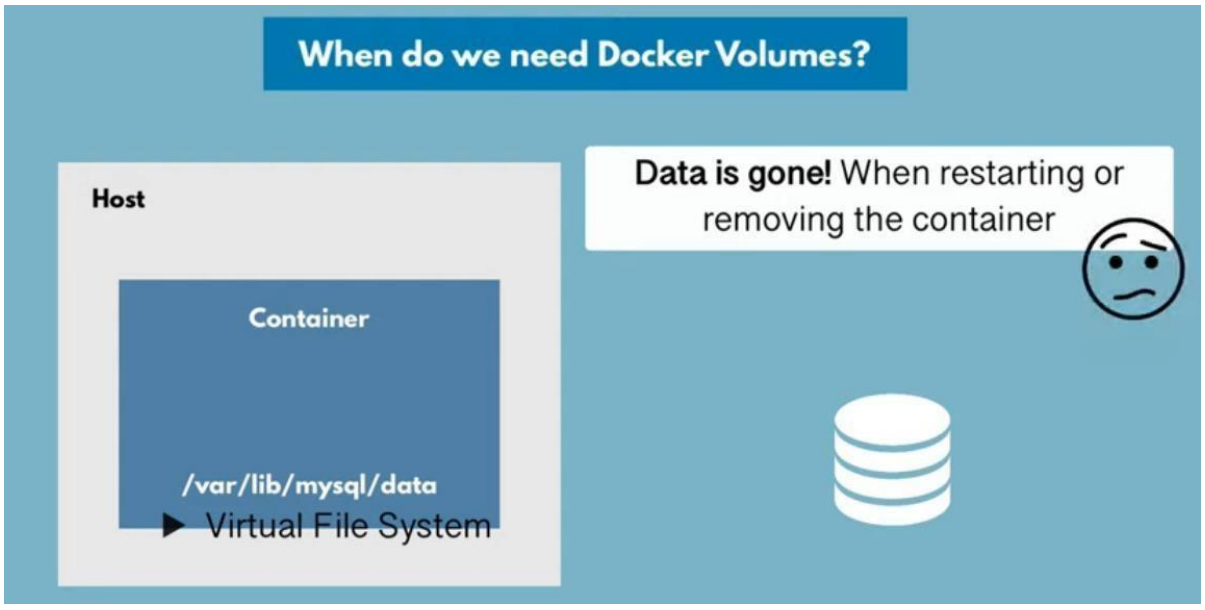
- Google Compute Engine
- Microsoft Azure
- Rackspace

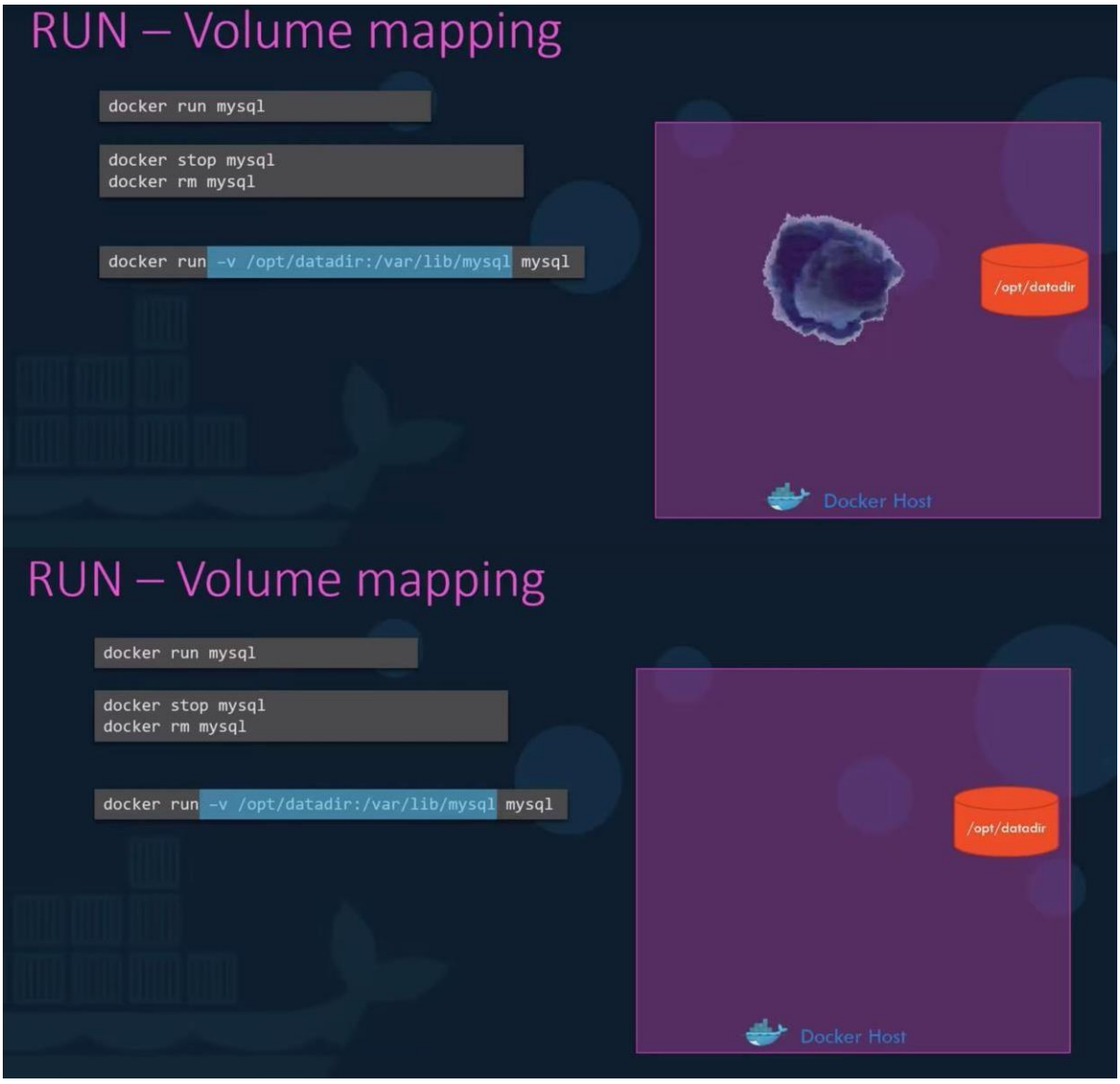
Docker is extending its support to Windows and Mac OSX environments and support on Windows has been on the growth in a very drastic manner. **Q. Do I lose my data when the Docker container exits?**



Volume mapping


ACTIVE CONTAINER
STORAGE DRIVER
WRITE LAYER
STOPPED CONTAINER
NO DATA


When do we need Docker Volumes?
Host
Container
/var/lib/mysql/data
Virtual File System
Data is gone! When restarting or removing the container



- There is no loss of data when any of your Docker containers exits as any of the data that your application writes to the disk in order to preserve it.
- This will be done until the container is explicitly deleted.
- The file system for the Docker container persists even after the Docker container is halted.

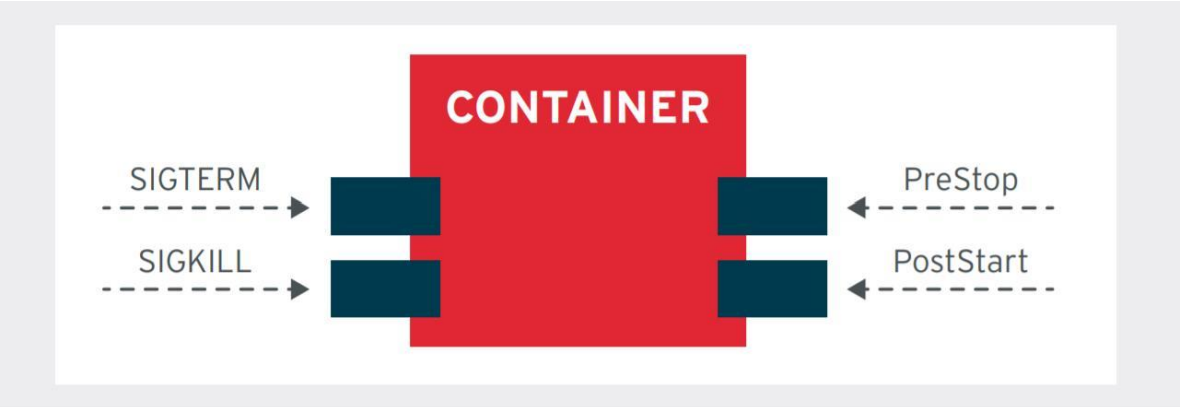
Q. What, in your opinion, is the most exciting potential use for Docker?

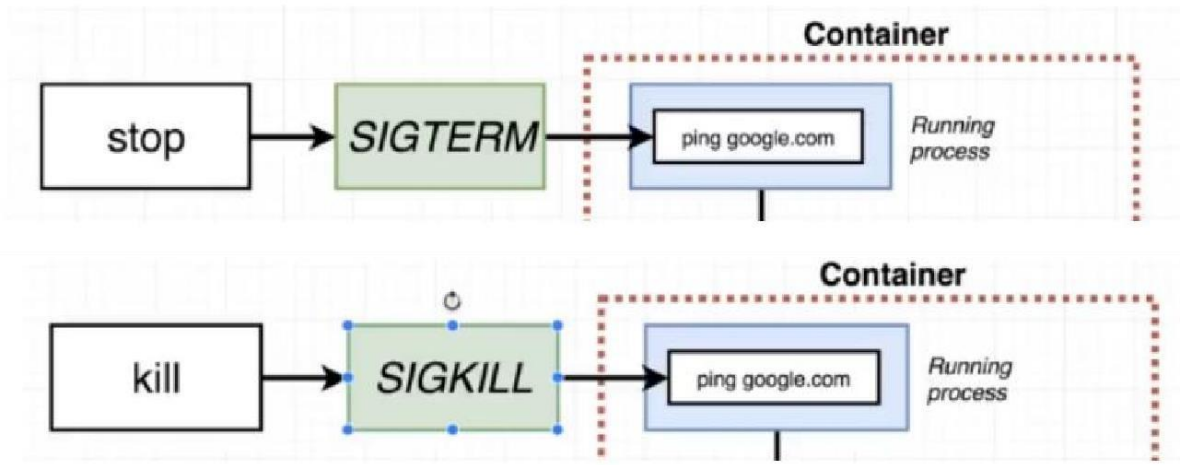
- The most exciting potential use of Docker that I can think of is its build pipeline.
- Most of the Docker professionals are seen using hyper-scaling with containers, and indeed get a lot of containers on the host that it actually runs on.
- These are also known to be blatantly fast. Most of the development – test build pipeline is completely automated using the Docker framework.

Q. Why is Docker the new craze in virtualization and cloud computing?

- Docker is the newest and the latest craze in the world of Virtualization and also Cloud computing because it is an ultra-lightweight containerization app that is brimming with potential to prove its mettle.

Q. Why do my services take 10 seconds to recreate or stop?





- When you run docker stop, you are instructing the Docker daemon to send a signal to the process running the container to stop.
- By default, it does this by sending a SIGTERM and then wait a short period so the process can exit gracefully. If the process does not terminate within a grace period (10s by default, customisable), it will send a SIGKILL.
- However, your application may be configured to listen to a different signal - SIGUSR1 and SIGUSR2, for example.
- In these instances, you can use the STOPSIGNAL Dockerfile instruction to override the default.

Q. How do I run multiple copies of a Compose file on the same host?

- Docker's compose makes use of the Project name to create unique identifiers for all of the project's containers and resources.
- In order to run multiple copies of the same project, you will need to set a custom project name using the –p command line option or you could use the COMPOSE_PROJECT_NAME environment variable for this purpose. **Q. What's the difference between up, run, and start?**
- On any given scenario, you would always want your -compose docker up. Using the command UP, you can start or restart all the services that are defined in a docker- compose.yml file.
- In the "attached" mode, which is also the default mode – we will be able to see all the log files from all the containers.
- In the "detached" mode, it exits after starting all the containers, which continue to run in the background showing nothing over in the foreground.

- Using docker-compose run command, we will be able to run the one-off or the ad-hoc tasks that are required to be run as per the Business needs and requirements.
- This requires the service name to be provided which you would want to run and based on that, it will only start those containers for the services that the running service depends on.
- Using the run command, you can run your tests or perform any of the administrative tasks as like removing / adding data to the data volume container.
- It is also very similar to the docker run –ti command, which opens up an interactive terminal to the containers an exit status that matches with the exit status of the process in the container.
- Using the docker-compose start command, you can only restart the containers that were previously created and were stopped. This command never creates any new Docker containers on its own.

Q. What's the benefit of "Dockerizing?"

Listed below are the few advantages of dockerizing your environment.

Continuous Integration: Any changes in the code will be automatically deployed immediately and would be available for testing anytime. Thus, Docker helps in Continuous Integration by significantly reducing the time.

Continuous Delivery: The transition time from development to production can be greatly reduced as one container can be used across multiple environments. This way applications can be delivered much faster and in a more reliable way than ever before.

Portability: Docker can be moved from one server to another with ease. Docker images come very handy while moving the container from one server to another without much efforts thereby saving a lot of time. The images can be either private or public.

Scalability: A Docker container is very lightweight in size like tens of megabytes when compared to gigabytes in the case of virtual machines. Thus, multiple docker containers can be launched on a single machine and are highly scalable as per the demand.

Micro Services Integration: Integrating microservices with the applications running on containers is easy. Each tier of a multi-tier application running on Docker behaves as an independent container and can be used to integrate microservices with the application.

Reduced Cost: Due to its various advantages such as continuous integration and continuous delivery, Docker significantly reduces the cost of running an application . **Q. How many containers can run per host?**

- Depending on the environment where Docker is going to host the containers, there can be as many containers as the environment supports.
- The application size, available resources (like CPU, memory) will decide on the number of containers that can run on an environment.
- Though containers create newer CPU on their own but they can definitely provide efficient ways of utilizing the resources.
- The containers themselves are super lightweight and only last as long as the process they are running.

Q. Is there a possibility to include specific code with COPY/ADD or a volume?

- COPY <src> <dest>
- The COPY instruction will copy new files from <src> and add them to the container's filesystem at path <dest>
- ADD <src> <dest>

- The ADD instruction will copy new files from <src> and add them to the container's filesystem at path <dest>.
- COPY copies a file/directory from your host to your image.
- ADD copies a file/directory from your host to your image, but can also fetch remote URLs, extract TAR files, etc...
- Use COPY for simply copying files and/or directories into the build context.
- Use ADD for downloading remote resources, extracting TAR files, etc..
- The main difference between these two is that Add command can also read the files from a URL.

Q. Will cloud automation overtake containerization any sooner?

- Docker containers are gaining the popularity in Continuous Integration / Continuous Development pipelines.
- Having said that there is equal responsibility on all the key stakeholders at each Organization to take up the challenge of weighing the risks and gains on adopting technologies that are budding up on a daily basis.
- Docker will be extremely effective in Organizations that appreciate the consequences of Containerization.

Q.How to use the port in Docker.

$ docker run -d redis:latest
74fa693703d98fd2d09bfad0ffd916af1709ef07c27016c1985c6965639a68c4
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

74fa693703d9 redis:latest "docker-entrypoint.s..." 28 seconds ago Up 28 seconds 6379/tcp frosty_elbakyan
curl -k localhost:6379

curl: (7) Failed to connect to 127.0.0.1 port 6379: Connection refused

- *Redis* is running, but cannot access it. The reason is that each container is sandboxed. If a service needs to be accessible by a process not running in a container, then the port needs to be exposed via the Host/ Docker port-forwarding .
- Once exposed, it is possible to access the process as if it were running on the host OS itself.

Define the port to be used for the remote connection by Docker port- forwarding . docker run -p [port_number]:6379 -d redis Define the host-name or IP

For example, to open and bind to a network port on the host you need to provide the parameter *-p <host-port>:<container-port>*.

```
$ docker run -d --name redisHostPort -p 6379:6379 redis:latest

4dd56fcecf62eb4b7ce4d651c5c5b60d91e428dfde00975106abef1c98276594 $
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
4dd56fcecf62 redis:latest "docker-entrypoint.s..." 9 seconds ago Up 9 seconds
0.0.0.0:6379->6379/tcp redisHostPort
```

By default, the port on the host is mapped to **0.0.0.0**, which means all IP addresses. You can specify a particular IP address when you define the port mapping

Port binding

- This method is used for outside the same network.
- To allow communication via the defined ports to containers outside of the same network, you need to publish the ports by using -p flag on docker run to publish and map one or more ports, or the -P flag to publish all exposed ports and map them to high-order ports.
- You can do port porting via one of the below ways:

1. Expose a port via Dockerfile by **–expose** and publish it with the **-P** flag. It will bind the exposed port to the Docker host on a random port.
2. Expose a port via Dockerfile by **–expose** and publish it with the **-p** 6379**:** 6379 flag, this will bind the expose port to Docker host on certain port 6379 with guest 6379.

Bind the port by docker container run command:

- The problem with running processes on a fixed port is that you can only run one instance.
- We would prefer to run multiple *Redis* instances and configure the application depending on which port Redis is running on.

$ docker run -d --name redisDynamic -p 6379:6379 redis:latest
74435b177c7d7b3ac3c2833c9efa6215735c23ce0f3432bad36d281c2fca2b14

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
74435b177c7d redis:latest "docker-entrypoint.s…" 5 seconds ago Up 4 seconds 0.0.0.0:32768->6379/tcp redisDynamic **For two instances**

docker run -d --name redisDynamic1 -p 6379:6379 redis:latest docker run -d --name redisDynamic2 -p 6380:6379 redis:latest

Access The Redis Containers:

We can access the Redis containers using the host computer's IP address and port number.
http://ip_address:6379 http://ip_address:6380

Q. Explain Port mapping in docker

- Publishing Docker ports via **-P or -p**
- There are two ways of publishing ports in Docker:
- Using the -P flag (**Random Port Mapping)**
- Using the -p flag (**Fixed Port mapping using UDP/TCP**)

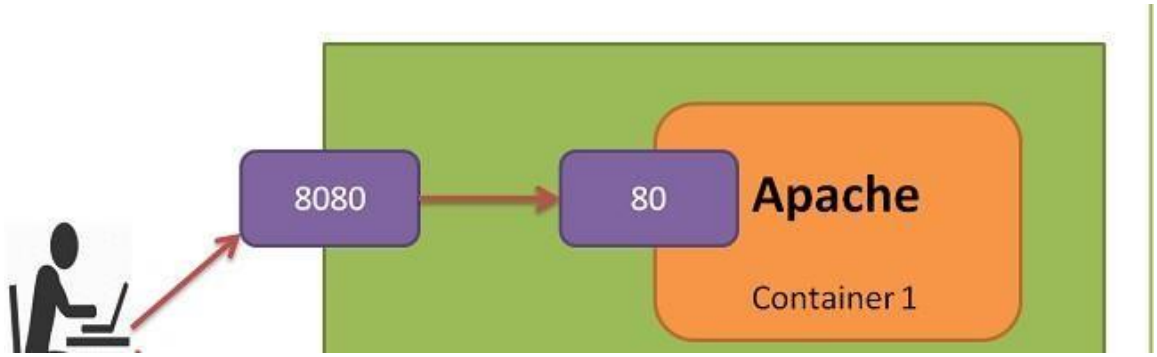
Fixed Port mapping using UDP/TCP

- when you expose a port from Dockerfile that means you are mapping a port defined in your image to your newly launched container

```
#docker run -d -p 8080:80 --name=mycontaniername myimagename
#docker run -d --name MyWebServer -p 8080:80 httpd
```

- when you want to change the protocol from default i.e tcp to udp , use:

```
#docker run -d -p 8080:80/udp --name=mycontinername myimagename
#docker run -d --name MyWebServer -p 8080:80/udp httpd
```



lets say when you want to expose your image port to any specific IP address from your docker host , use:

```
#docker run -d -p 192.168.0.100:8080:80 --name=mycontaniername myimagename docker run -d -p 192.168.0.100:8080:80 --name MyWebServer httpd
```

now when you want to map multiple ports exposed in your Dockerfile to high random available ports , use:

```
#docker run -d -P --name=mycontaniername3 myimagename
$ docker run -d --name MyWebServer -P httpd
```

to check port mapping , use:

```
#docker port myimagename
```

Random Port Mapping

First up, stop and remove the Container so that we can use the same Container Name i.e. MyWebServer.

```
$ docker stop MyWebServer
MyWebServer
```

```
$ docker rm MyWebServer
MyWebServer
```

- Now, let us start the httpd Container with an extra parameter i.e. -P. What this parameter does is that it "Publish all exposed ports to random ports". So in our case, the port 80 should get mapped to a random port, which is going to be the public port.

Execute the following command:

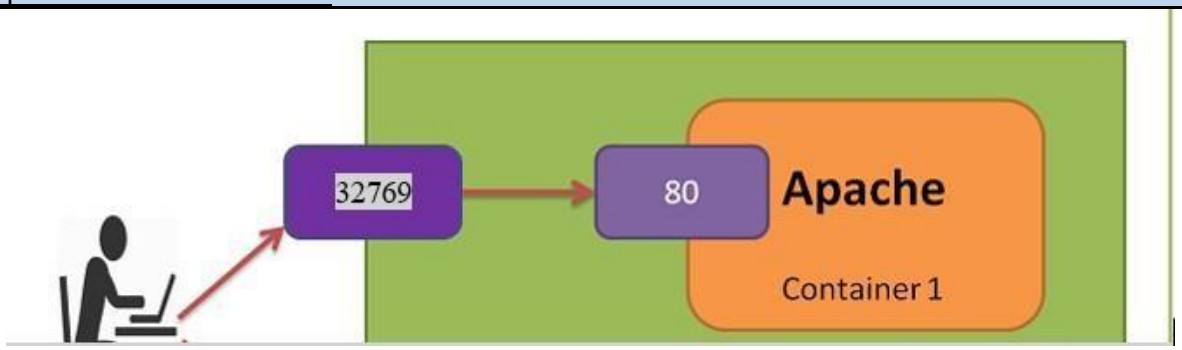
```
$ docker run -d --name MyWebServer -P httpd
60debd0d57bf292b0c3f006e4e52360feaa575e45ae3caea97637bb26b490b10
```

Next, let us use the port command again to see what has happened:

```
$ docker port MyWebServer
80/tcp -> 0.0.0.0:32769
```

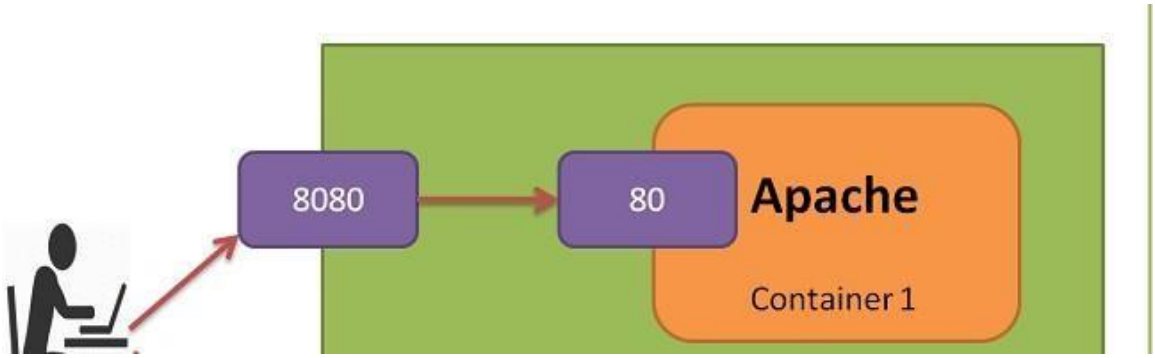
We can see that port 80 has got mapped to port 32769. So if we access our web site at

```
http://<HostIP>/<HostPort>
```



Specific Port Mapping

So what if we wanted to map it to a port number other than 32769. You can do that via the **-p** (**note the lowercase**) parameter.



This parameter format is as follows:

-p HostPort:ContainerPort

For e.g. -p 8080:80

The first parameter is the Host port and we are now making that 8080. The second port is what Apache httpd exposes i.e. 80.

Let us check out everything again:

```
$ docker stop MyWebServer
MyWebServer
```

```
$ docker rm MyWebServer MyWebServer
```

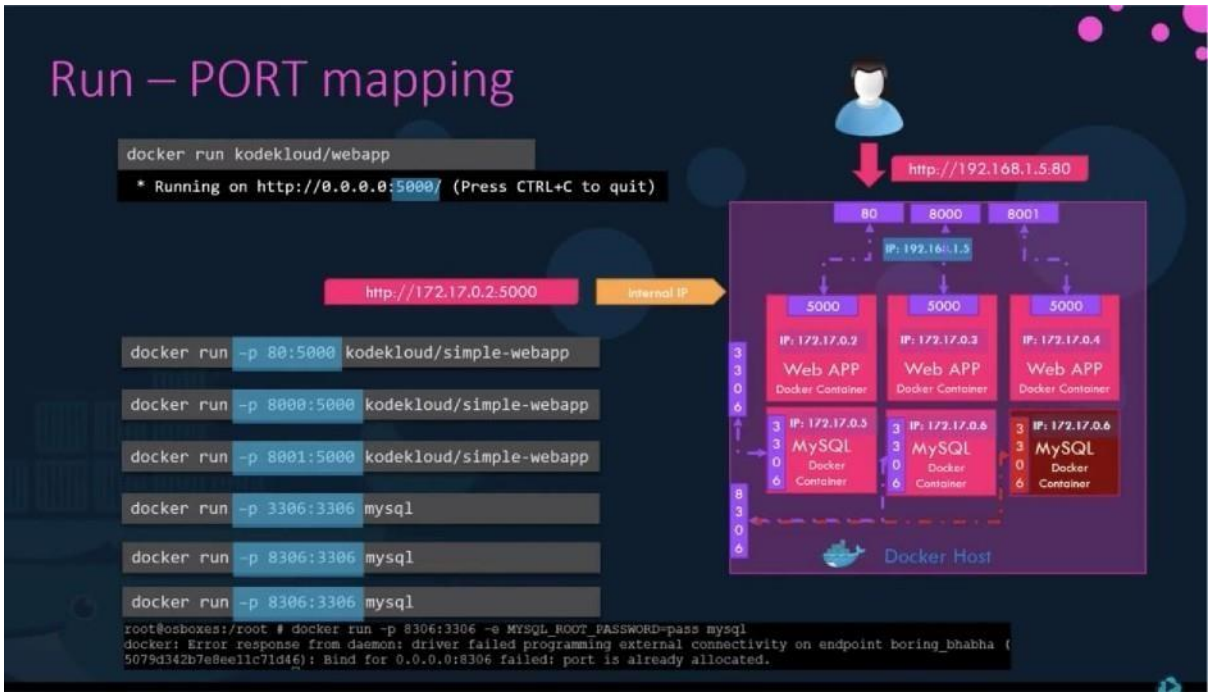
```
$ docker run -d --name MyWebServer -p 8080:80 httpd
02ce77550940cb1f37361b74af8913e46d6a507d06c2579b8a8b49e389b1e75f
```

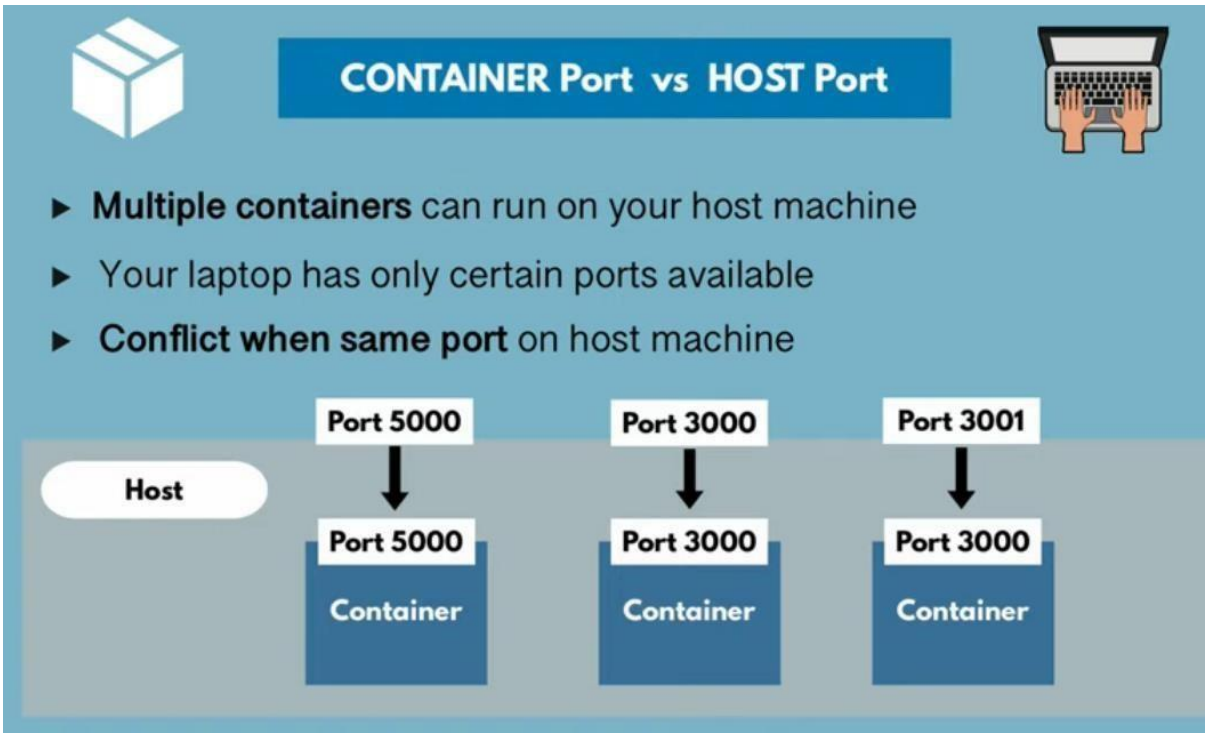
Now, let us give the port command again:

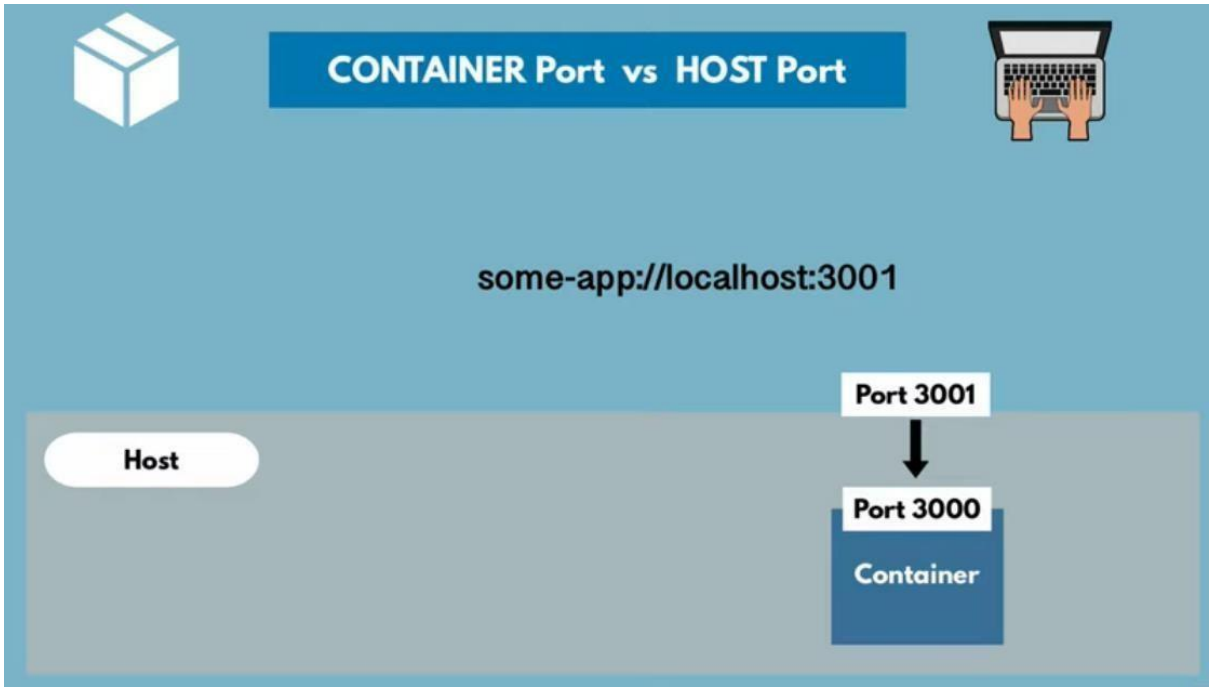
```
$ docker port MyWebServer
80/tcp -> 0.0.0.0:8080
```

Flag value	Description
p 8080 80	Bina container's TCP port 80 to host's port 8080
p 192.0.2.1:80B0:80	Bind container's TCP port 80 to host's port 8080 for connections to host IP 192.0.2.1. By default, Docker binds published container ports to the 0.0.0.0 IP address, which matches any IP address on the system

-p 8080: 80/udp	Bind container's UDP port 80 to host's port 8080
-p 8080:80/tcp -p 8080 :80/udp	Bind container's TCP port 80 to host's TCP port 8080, and bind container's UDP port 80 to host's UDP port 8080
-p 2346-2346:2346-2346/tcp	Specify **hostPort** and **containerPort** as a range of ports. Note that the number of container ports specified in the range should be equivalent to rhe number of host ports specified in the ranqe.
-p 2346-2346:2346/top	Specify **hostPort** range only. In such a case, **containerPort** must not be in a specific range. Container port will be published anywhere within the sFated **hostPort** range.







Binding Ports

Forward everything

If you append -P (or --publish-all=true) to docker run, Docker identifies every port the Dockerfile exposes (you can see which ones by looking at the EXPOSE lines). Docker also finds ports you expose with --expose 8080 (assuming you want to expose port 8080). Docker maps all of these ports to a host port within a given epehmeral port range. You can find the configuration for these ports (usually 32768 to 61000)

Forward selectively

You can also specify ports. When doing so, you don't need to use ports from the ephemeral port range. Suppose you want to expose the container's port 8080 (standard http port) on the host's port 80 (assuming that port is not in use). Append -p 80:8080 (or --publish=80:8080) to your docker run command. For example:

```
docker run -p 80:8080 nginx

## OR ## docker run --publish=80:8080 nginx
```

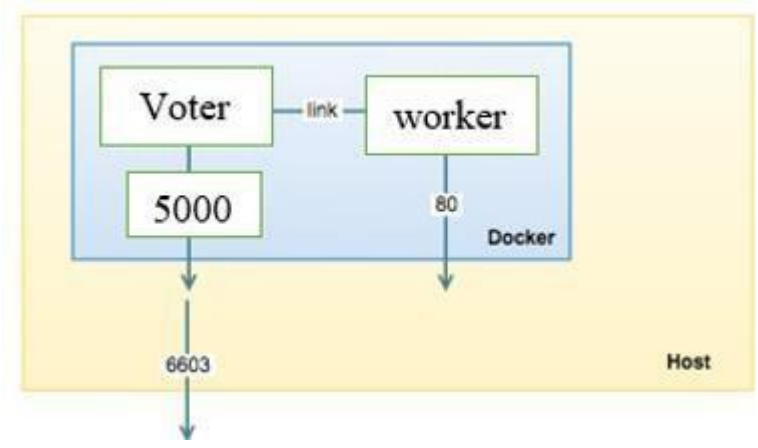
Custom IP *and* port forwarding

By default, Docker exposes container ports to the IP address 0.0.0.0 (this matches any IP on the system). If you prefer, you can tell Docker *which* IP to bind on. To bind on IP address 10.0.0.3, host port 80, and container port 8080:

```
docker run -p 10.0.0.3:80:8080 nginx
```

Q.What is container linking in docker ?

Container Linking allows multiple containers to link with each other. It is a better option than exposing ports. docker run -p 5000:80 –link redis:redis voter-app docker run –link redis:redis –link db:db worker-app



By linking containers, you provide a secure channel via which Docker containers can communicate to each other but Docker compose is the recommended way when no increases

Q. Is there a way to identify the status of a Docker container?

- We can identify the status of a Docker container by 'docker ps –a', which will in turn list down all the available docker containers with its corresponding statuses on the host **Q. What are the differences between the 'docker run' and the 'docker create'?**
- Docker run is basically for running commands in the container.
- **docker run -it <Container Name> /bin/bash**
- The above is for creating a bash terminal. And make us use bash commands in the container.
- Docker create is to create a container from a Docker Image.
- **docker create -d /var/lib:/var/lib --name docker-ubuntu ubuntu**
- The above is to create a docker a container of the name "docker-ubuntu" from the image "ubuntu"

Q. What are the various states that a Docker container can be in at any given point in time?

There are four states that a Docker container can be in, at any given point in time. Those states are as given as follows:

• Running
• Paused
• Restarting
• Exited

Q. Can you remove a paused container from Docker?

- No, it is not possible to remove a container from Docker that is just paused.
- It is a must that a container should be in the stopped state, before it can be removed from the Docker container.

Q. Is there a possibility that a container can restart all by itself in Docker?

- No, it is not possible.
- The default –restart flag is set to never restart on its own.

Q. What is the preferred way of removing containers - 'docker rm -f' or 'docker stop' then followed by a 'docker rm'?

- The best and the preferred way of removing containers from Docker is to use the '**docker stop**', as it will allow sending a **SIG_HUP** signal to its recipients giving them the time that is required to perform all the finalization and clean-up tasks.
- Once this activity is completed, we can then comfortably remove the container using the 'docker rm' command from Docker and thereby updating the docker registry as well.

Q. Difference between Docker Image and container?

- Docker container is the runtime instance of docker image.
- Docker Image does not have a state and its state never changes as it is just set of files whereas docker container has its execution state.

12. What are the commands that are available in the Dockerfile?

The followings are the commands that are available in the Dockerfile: Add
CMD
Entry point
ENV

EXPOSE
FROM
MAINTAINER
RUN
USER
VOLUME
WORKDIR
Now, let us look at another Dockerfile shown below:

```
FROM ubuntu
MAINTAINER vijay (vijay15.biradar@gmail.com) date
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT [“/usr/sbin/nginx”,”-g”,”daemon off;”] EXPOSE
80
```

Here, what we are building is an image that will run the nginx proxy server for us.
These instructions inform Docker that we want to create an image:
FROM a ubuntu base image with the tag of latest
MAINTAINER Author field of the generated images is vijay
CMD Defining a command to be run after the docker is up is [date]
RUN -running a package update and then installing nginx on newly created operating system.
The ENTRYPOINT is then running the nginx executable
EXPOSE command will open the mentioned port on the docker image to allow access to outside world to 80
EXPOSE port will be used by default. However, if we want to change the host port then we have to use -p parameter.
If you build the image and run the container as follows:
docker build -t webserver- myimage:v1 . docker images
| grep webserver- myimage
webserver- myimage v1 d79c7313bba5 6 minutes ago 16.1MB **docker run -d -p 80:80 --name webserver- myimage**
You will find that it will have nginx started on port 80. And if you visit the page via the host IP, you will see the following point: http://localhost:80

Q) Explain about configure networking in Docker?
Answer:
bridge: The default network driver
host: For stand-alone containers , remove network isolation between the container and the docker host
Overlay: Overlay networks connect multiple docker daemons
macvlan: for assigning MAC address for container none:
disable all neworking

Q What is the command to create a docker swarm?
Answer: docker swarm init –advertise-addr <manager IP>
What is docker Docker compose

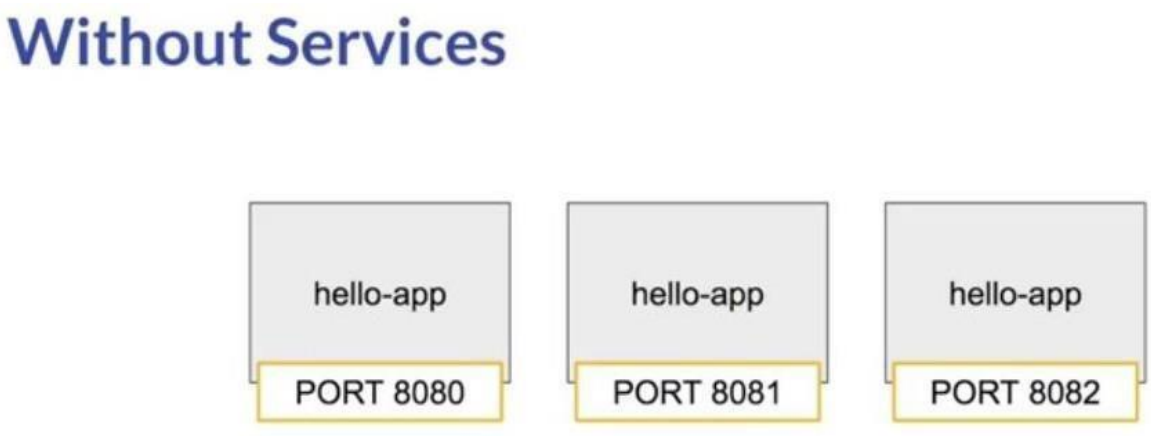
- tool for defining & running multi-container docker applications
- use yaml files to configure application services (docker-compose.yml)
- can start all services with a single command : **docker compose up**
- can stop all services with a single command : **docker compose down**
- **can scale up selected services when required**
- docker-compose is allows you to run multiple services as kind of micro service by defining them in a single configuration file
- docker compose is a docker tool for defining and running multi containers docker application.
- docker compose allows us to define all the services in a configuration file and with one command it will spin up all the containers that we need.
- it uses yaml files to configure application services (docker-compose.yml)
- it uses single command to start and stop all the services (docker-compose up & docker-compose down)
- it can scale up services whenever required.

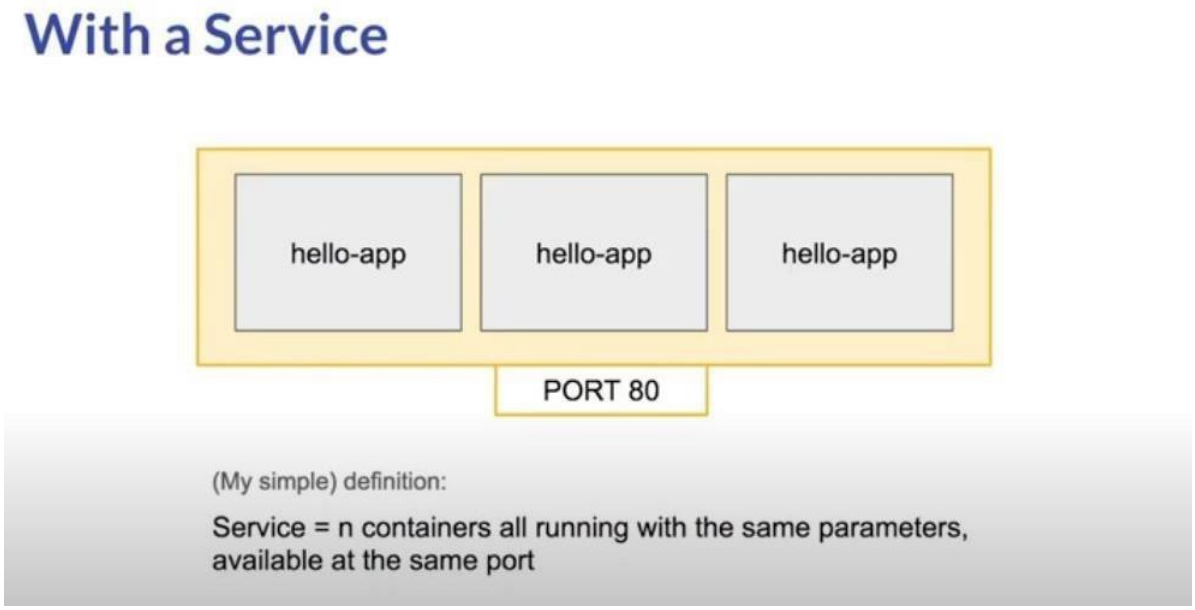
```
version: '3' services:
  web:    image:
    nginx
  db:
    image:   mysql
    ports: -
    "3306:3306"
    environment:
    - MYSQL_ROOT_PASSWORD=password
    - MYSQL_USER=user
    - MYSQL_PASSWORD=password
    - MYSQL_DATABASE=demodb

docker-compose   up
docker-compose up d
docker-compose ps
```

What is a service?

- A service is a group of containers of the same **image:tag**.
- Services make it simple to scale your application.
- Services are really just "containers in production."
- A service only runs one image, but it codifies the way that image runs—what ports it should use, how many replicas of the container should run so the service has the capacity it needs, and so on.
- Scaling a service changes the number of container instances running that piece of software, assigning more computing resources to the service in the process.
- When you create a service, you specify which container image to use and which commands to execute inside running containers. You also define options for the service including: the port where the swarm makes the service available outside the swarm an overlay network for the service to connect to other services in the swarm CPU and memory limits and reservations a rolling update policy the number of replicas of the image to run in the swarm
- docker service create --replicas 3 -p 80:80 --name hello-app nginx
- docker service scale hello-app=8





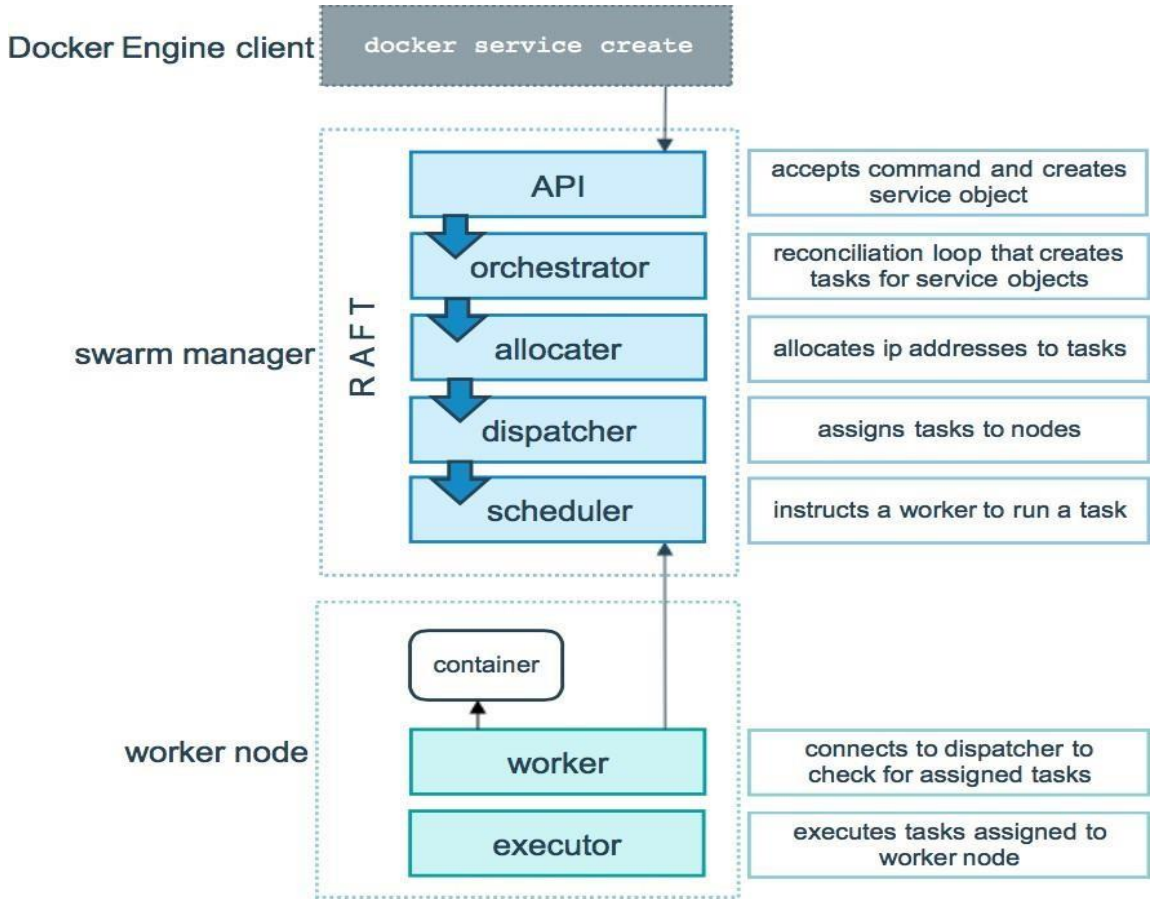
Docker services vs docker container

- The docker run command creates and starts a container on the local docker host.
- A docker "service" is one or more containers with the same configuration running under docker's swarm mode.
- It's similar to docker run in that you spin up a container. • The difference is that you now have orchestration
- Docker run will start a single container.
- With docker service you manage a group of containers (from the same image). You can scale them (start multiple containers) or update them.

Services deployment types in Docker

- There are two types of service deployments Replicated services and global services
- For a replicated service, you specify the number of identical tasks you want to run.
- A global service is a service that runs one task on every node.
- There is no pre-specified number of tasks. Each time you add a node to the swarm, the orchestrator creates a task and the scheduler assigns the task to the new node.

How swarm mode accepts service create requests and schedules tasks to worker nodes.



Q. Can we run multiple apps on one server with Docker?

Yes, theoretically we can run multiples apps on one Docker server. But in practice, it is better to run different components on separate containers.

With this we get cleaner environment and it can be used for multiple uses. **Q.**

What are the main features of Docker-compose?

Some of the main features of Docker-compose are as follows:

Multiple environments on same Host: We can use it to create multiple environments on the same host server.

Preserve Volume Data on Container Creation: Docker compose also preserves the volume data when we create a container.

Recreate the changed Containers: We can also use compose to recreate the changed containers.

Variables in Compose file: Docker compose also supports variables in compose file. In this way we can

Create variations of our containers.

Q. What is the most popular use of Docker?

The most popular use of Docker is in build pipeline.

With the use of Docker it is much easier to automate the development to deployment process in build pipeline.

We use Docker for the complete build flow from development work, test run and deployment to production environment.

Q. What is the role of open source development in the popularity of Docker?

Since Linux was an open source operating system, it opened new opportunities for developers who want to contribute to open source systems.

One of the very good outcomes of open source software is Docker.
It has very powerful features.
Docker has wide acceptance due to its usability as well as its open source approach of integrating with different systems.

Q. What is Docker Machine?

We can use Docker Machine to install Docker Engine on virtual hosts.
It also provides commands to manage virtual hosts.
Some of the popular Docker machine commands enable us to start, stop, inspect and restart a managed host.
Docker Machine provides a Command Line Interface (CLI), which is very useful in managing multiple hosts.

Q. Why do we use Docker Machine?

There are two main uses of Docker Machine:
Old Desktop : If we have an old desktop and we want to run Docker then we use Docker Machine to run Docker. It is like installing a virtual machine on an old hardware system to run Docker engine.
Remote Hosts : Docker Machine is also used to provision Docker hosts on remote systems. By using Docker Machine you can install Docker Engine on remote hosts and configure clients on them.

Q. How will you create a Container in Docker?

To create a Container in Docker we have to create a Docker Image. We can also use an existing Image from Docker Hub Registry.
We can run an Image to create the container.

Q. Do you think Docker is Application-centric or Machine-centric?

Docker is an Application-centric solution.
It is optimized for deployment of an application.
It does not replace a machine by creating a virtual machine. Rather, it focuses on providing ease of use features to run an application.

Q. Can we run more than one process in a Docker container?

Yes, a Docker Container can provide process management that can be used to run multiple processes.
There are process supervisors like runit, s6, daemontools etc that can be used to fork additional processes in a Docker container.

Q. What are the objects created by Docker Cloud in Amazon Web Services (AWS) EC2? Docker Cloud creates following objects in AWS EC2 instance:
VPC : Docker Cloud creates a Virtual Private Cloud with the tag name dc-vpc. It also creates Class Less
Inter-Domain Routing (CIDR) with the range of 10.78.0.0/16 .
Subnet : Docker Cloud creates a subnet in each Availability Zone (AZ). In Docker Cloud, each subnet is tagged with dc-subnet.
Internet Gateway : Docker Cloud also creates an internet gateway with name dc-gateway and attaches it to the VPC created earlier.
Routing Table : Docker Cloud also creates a routing table named dc-route-table in Virtual Private Cloud. In this Routing Table Docker Cloud associates the subnet with the Internet Gateway.

Q. How will you take backup of Docker container volumes in AWS S3?

We can use a utility named Dockup provided by Docker Cloud to take backup of Docker container volumes in S3.

Q. What are the three main steps of Docker Compose?

Three main steps of Docker Compose are as follows:

Environment : We first define the environment of our application with a Dockerfile. It can be used to recreate the environment at a later point of time.

Services : Then we define the services that make our app in docker-compose.yml. By using this file we can define how these services can be run together in an environment.

Run : The last step is to run the Docker Container. We use docker-compose up to start and run the application.

Q. What is Pluggable Storage Driver architecture in Docker based containers?

Docker storage driver is by default based on a Linux file system. But Docker storage driver also has provision to plug in any other storage driver that can be used for our environment.

In Pluggable Storage Driver architecture, we can use multiple kinds of file systems in our Docker Container.

In Docker info command we can see the Storage Driver that is set on a Docker daemon. We can even plug in shared storage systems with the Pluggable Storage Driver architecture.

Q. What are the main security concerns with Docker based containers?

Docker based containers have following security concerns:

Kernel Sharing: In a container-based system, multiple containers share same Kernel. If one container causes Kernel to go down, it will take down all the containers. In a virtual machine environment we do not have this issue.

Container Leakage: If a malicious user gains access to one container, it can try to access the other containers on the same host. If a container has security vulnerabilities it can allow the user to access other containers on same host machine.

Denial of Service: If one container occupies the resources of a Kernel then other containers will starve for resources. It can create a Denial of Service attack like situation.

Tampered Images: Sometimes a container image can be tampered. This can lead to further security concerns. An attacker can try to run a tampered image to exploit the vulnerabilities in host machines and other containers.

Secret Sharing: Generally one container can access other services. To access a service it requires a Key or Secret. A malicious user can gain access to this secret. Since multiple containers share the secret, it may lead to further security concerns.

Q. How can we check the status of a Container in Docker?

We can use `docker ps –a` command to get the list of all the containers in Docker. This command also returns the status of these containers.

Q. What are the main benefits of using Docker?

Docker is a very powerful tool. Some of the main benefits of using Docker are as follows: **Utilize Developer Skills** : With Docker we maximize the use of Developer skills. With Docker there is less need of build or release engineers. Same Developer can create software and wrap it in one single file.

Standard Application Image : Docker based system allows us to bundle the application software and Operating system files in a single Application Image that can be deployed independently.

Uniform deployment : With Docker we can create one package of our software and deploy it on different platforms seamlessly.

Q. How does Docker simplify Software Development process?

Prior to Docker, Developers would develop software and pass it to QA for testing and then it is sent to Build & Release team for deployment.

In Docker workflow, Developer builds an Image after developing and testing the software. This Image is shipped to Registry. From Registry it is available for deployment to any system. The development process is simpler since steps for QA and Deployment etc take place before the Image is built. So Developer gets the feedback early. **Q. What is the basic architecture behind Docker?**

- Docker is built on client server model.
- Docker server is used to run the images.
- We use Docker client to communicate with Docker server.
- Clients tell Docker server via commands what to do.
- Additionally there is a Registry that stores Docker Images.
- Docker Server can directly contact Registry to download images.

Q. What are the popular tasks that you can do with Docker Command line tool?

- Docker Command Line (DCL) tool is implemented in Go language.
- It can compile and run on most of the common operating systems.
- Some of the tasks that we can do with Docker Command Line tool are as follows:
- We can download images from Registry with DCL.
- We can start, stop or terminate a container on a Docker server by DCL.
- We can retrieve Docker Logs via DCL.
- We can build a Container Image with DCL.

Q. What type of applications- Stateless or Stateful are more suitable for Docker Container?

- Docker was designed for stateless applications and horizontal scalability, with containers deleted and replaced as needed
- We can create a container out of our application and take out the configurable state parameters from application.
- Now we can run same container in Production as well as QA environments with different parameters.
- This helps in reusing the same Image in different scenarios.
- stateless application is much easier to scale with Docker Containers than a stateful application.
- Databases are not suited for this approach, and Docker is evolving to support the needs of stateful enterprise apps.
- Docker supports for few database services and it doesn't supports all of the database services that you might expect out of your Docker environment. **Q. How can Docker run on different Linux distributions?**
- Docker directly works with Linux kernel level libraries.
- In every Linux distribution, the Kernel is same.
- Docker containers share same kernel as the host kernel.
- Since all the distributions share the same Kernel, the container can run on any of these distributions.

Q. Why do we use Docker on top of a virtual machine?

- Generally, we use Docker on top of a virtual machine to ensure isolation of the application.
- On a virtual machine we can get the advantage of security provided by hypervisor.
- We can implement different security levels on a virtual machine.

- Docker can make use of this to run the application at different security levels. **Q. How can Docker container share resources?**
- We can run multiple Docker containers on same host.
- These containers can share Kernel resources.
- Each container runs on its own Operating System and it has its own user-space and libraries. So, in a way Docker container does not share resources within its own namespace. But the resources that are not in isolated namespace are shared between containers. These are the Kernel resources of host machine that have just one copy.So in the back-end there is same set of resources that Docker Containers share. **Q. What is Docker Entrypoint?**
- We use Docker Entrypoint to set the starting point for a command in a Docker Image. We can use the entrypoint as a command for running an Image in the container.
  E.g. We can define following entrypoint in docker file and run it as following command: ENTRYPOINT [“mycmd”]
- % docker run mycmd
  ENTRYPOINT cannot be overriden at run time with normal commands such as docker run [args].
- ENTRYPOINT can be overriden with --entrypoint.
- The ENTRYPOINT specifies a command that will always be executed when the container starts.
- Otherwise, if you want to make an image for general purpose, you can leave ENTRYPOINT unspecified and use CMD ["/path/dedicated_command"] as you will be able to override the setting by supplying arguments to docker run
- CMD command mentioned inside Dockerfile file can be overridden via docker run command while ENTRYPOINT cannot be.
- entrypoint behaves similarly to cmd. And in addition, it allows us to customize the command executed at startup.
- Like with cmd, in case of multiple entrypoint entries, only the last one is considered.

```
FROM ubuntu
MAINTAINER vijay
RUN apt-get update
ENTRYPOINT ["echo", "Hello"]
CMD ["World"]
```

docker build .

docker run [container_name]

It will return the message Hello World. docker run

[container_name] [your_name]

The output has now changed to Hello [your_name]

- This is because you cannot override ENTRYPOINT instructions, whereas with CMD you can easily do so.

85. What is ONBUILD command in Docker?

We use ONBUILD command in Docker to run the instructions that have to execute after the completion of current Dockerfile build.

It is used to build a hierarchy of images that have to be build after the parent image is built.

A Docker build will execute first ONBUILD command and then it will execute any other command in Child Dockerfile. **Q. What is Build cache in Docker?**

When we build an Image, Docker will process each line in Dockerfile.

It will execute the commands on each line in the order that is mentioned in the file.

But at each line, before running any command, Docker will check if there is already an existing image in its cache that can be reused rather than creating a new image.

This method of using cache in Docker is called Build cache in Docker.

We can also specify the option –no-cache=true to let Docker know that we do not want to use cache for Images. With this option, Docker will create all new images. **Q. What are the most common instructions in Dockerfile?**

Some of the common instructions in Dockerfile are as follows:

FROM : We use FROM to set the base image for subsequent instructions. In every valid Dockerfile, FROM is the first instruction.

LABEL : We use LABEL to organize our images as per project, module, licensing etc. We can also use LABEL to help in automation.

In LABEL we specify a key value pair that can be later used for programmatically handling the Dockerfile.

RUN : We use RUN command to execute any instructions in a new layer on top of the current image. With each RUN command we add something on top of the image and use it in subsequent steps in Dockerfile.

CMD : We use CMD command to provide default values of an executing container. In a Dockerfile, if we include multiple CMD commands, then only the last instruction is used. **Q. What is the purpose of EXPOSE command in Dockerfile?**

We use EXPOSE command to inform Docker that Container will listen on a specific network port during runtime.

But these ports on Container may not be accessible to the host. We can use –p to publish a range of ports from Container.

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

In a Container we have an isolated environment with namespace for each resource that a kernel provides. There are mainly six types of namespaces in a Container.

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

Mount Namespace : 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. **PID Namespace** : This namespace contains all the processes that run within a Container. We can run ps command to see the processes that are running within a Docker container.

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.

User Namespace : This namespace contains the users and groups that are defined within a container.

Network Namespace : With this namespace, container provides 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.

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.

Q. What are the Cloud platforms that support Docker?

Some of the popular cloud platforms that support Docker are:

- Amazon AWS
- Google Cloud Platform
- Microsoft Azure
- IBM Bluemix

Q. How can we control the start-up order of services in Docker compose?

- In Docker compose we can use the depends on option to control the start-up order of services.
- With compose, the services will start in the dependency order.
- Dependencies can be defined in the options like- depends_on, links, volumes_from, network_mode etc But Docker does not wait for until a container is ready.

Q. Why Docker compose does not wait for a container to be ready before moving on to start next service in dependency order?

- The problem with waiting for a container to be ready is that in a Distributed system, some services or hosts may become unavailable sometimes. Similarly, during startup also some services may also be down.
- Therefore, we have to build resiliency in our application. So that even if some services are down we can continue our work or wait for the service to become available again.

We can use wait-for-it or dockerize tools for building this kind of resiliency.

Q. How will you customize Docker compose file for different environments?

- In Docker compose there are two files docker-compose.yml and docker-compose.override.yml.
- We specify our base configuration in docker- compose.yml file. For any environment specific customization we use docker-compose.override.yml file.
- We can specify a service in both the files.
- Docker compose will merge these files based on following rules:

1. For single value options, new value replaces the old value.
2. For multi-value options, compose will concatenate the both set of values.

- We can also use extends field to extend a service configuration to multiple environments.
- With extends, child services can use the common configuration defined by parent service.

Development command:

docker-compose -f docker-compose.yml -f docker-compose.dev.yml up **Production command:**

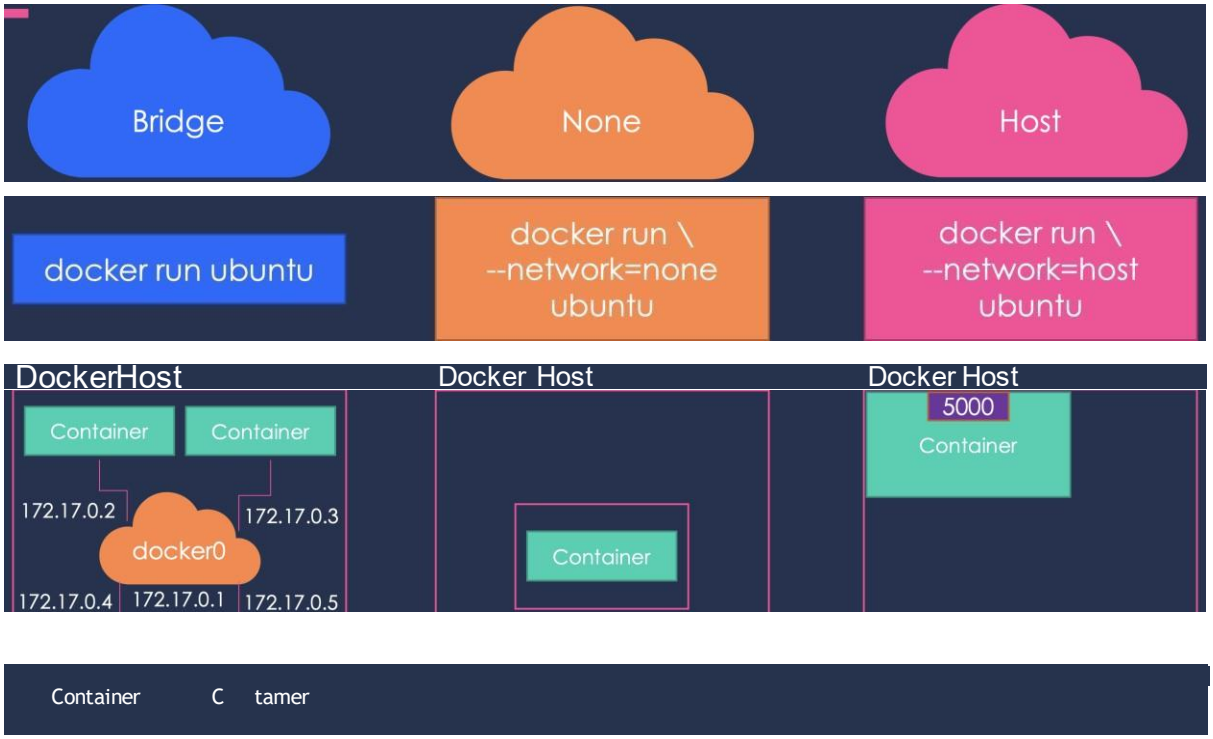
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up

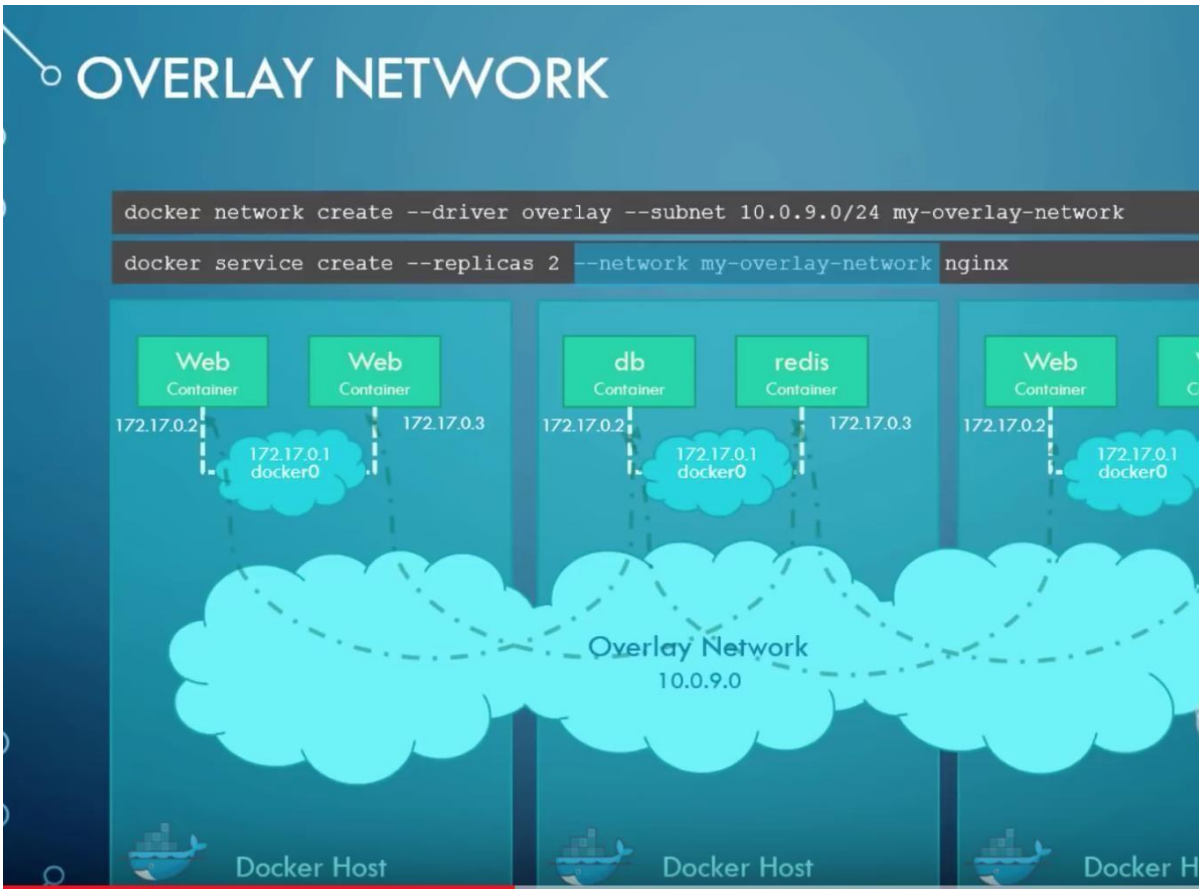
Note: If you name your second dockerfile `docker-compose.override.yml`, a simple docker-compose up would read the overrides automatically. But in your case, a name based on the environment is clearer.

Network Drivers

There are mainly 5 network drivers: Bridge, Host, None, Overlay, Macvlan Docker

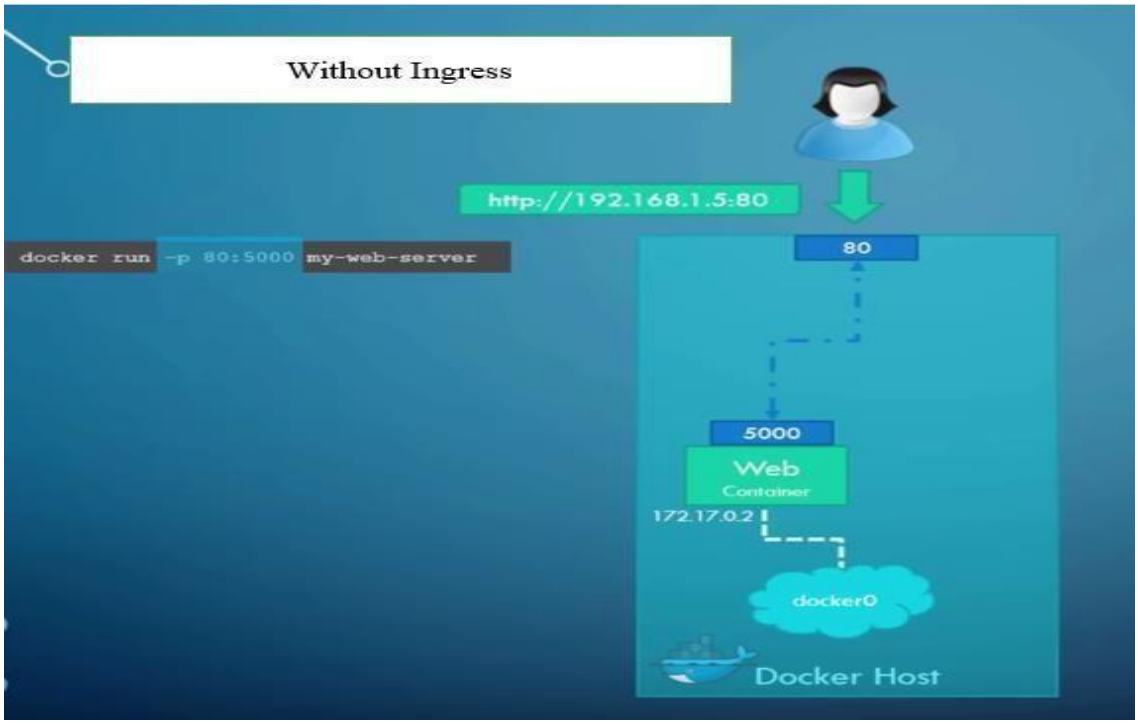
Networking

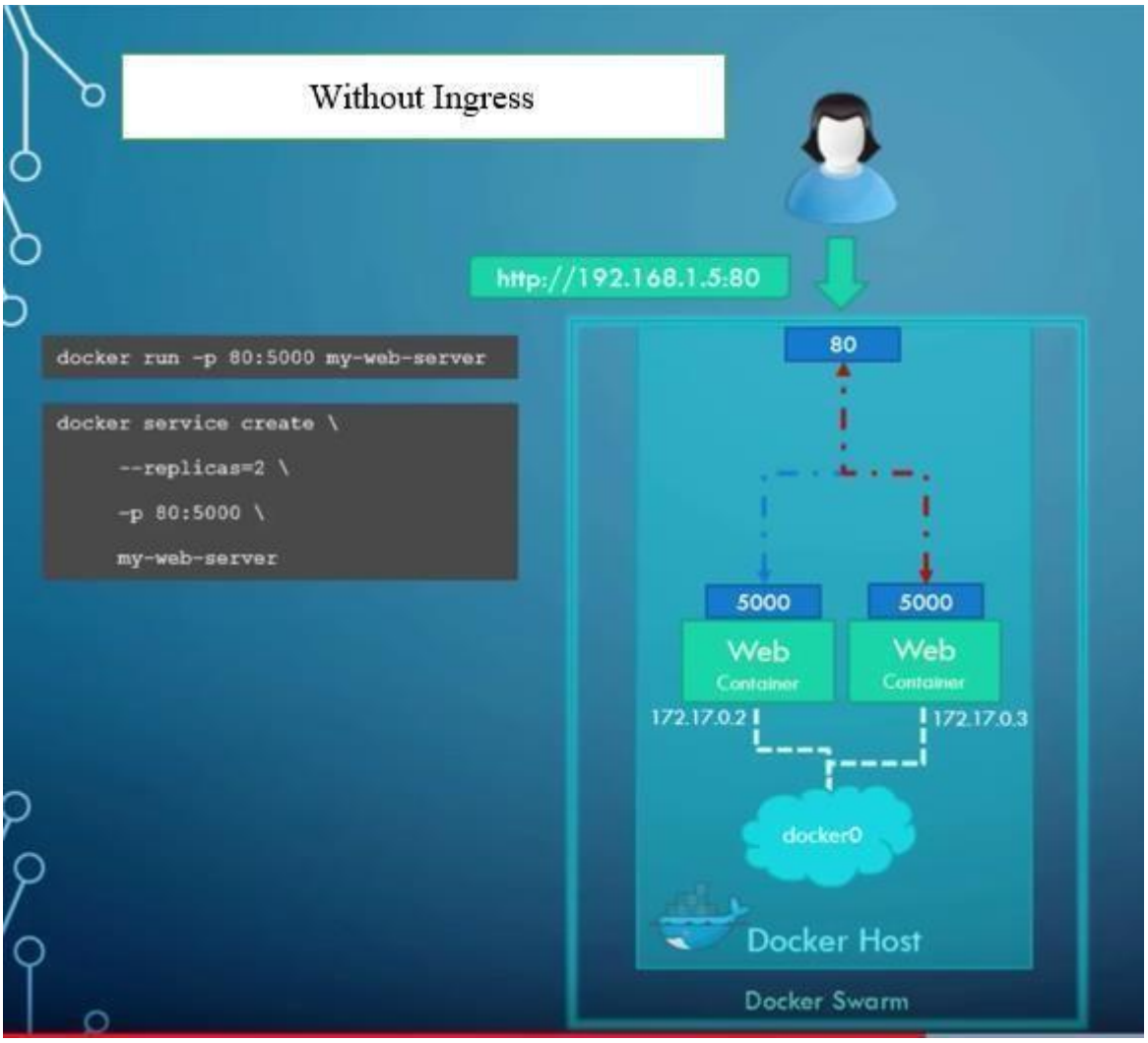




Q. How do multiple containers publish on the same port?

- Mapping of replica ports to the same host port denied Without ingress

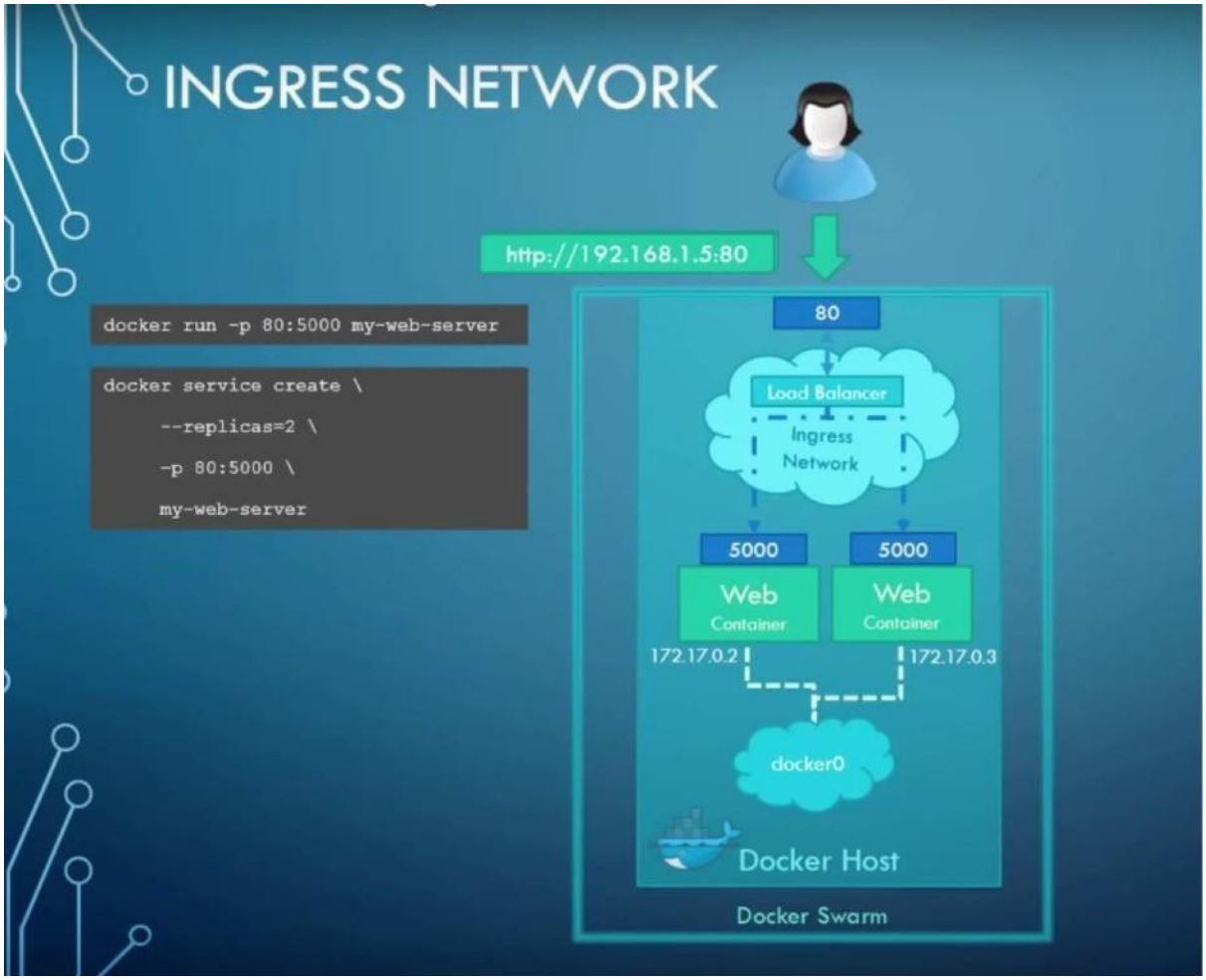




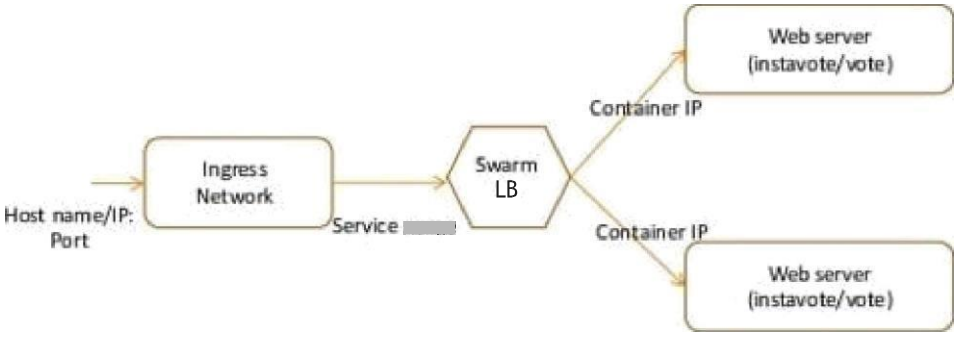
How do multiple containers publish on the same port?

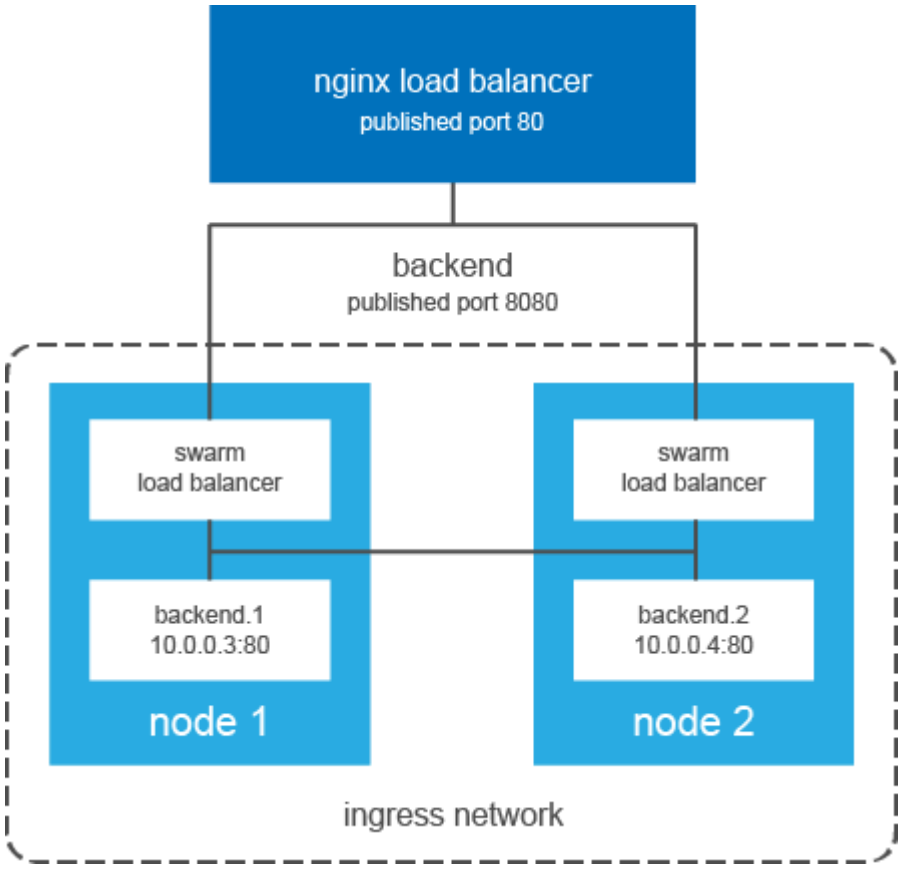
Mapping of replica ports to the same host port allowed with ingress

- When you create a docker swarm cluster, it automatically creates an ***ingress network***. The ingress network has a built-in ***load balancer*** that redirects traffic from the published port, which in this case is the port 80.
- All the mapped ports are the port 5000 on each container. Since the ingress network is created automatically there is no configuration that you have to do.



Ingress Load balancer





What is routing mesh under docker swarm mode

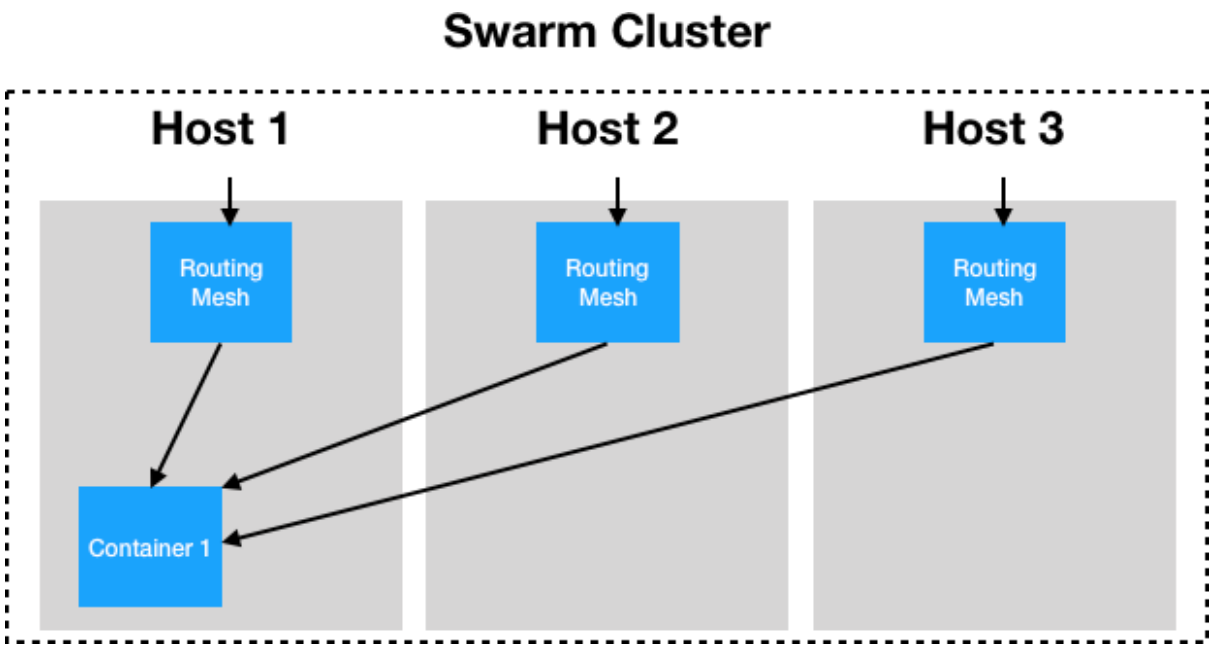
- Routing Mesh is a feature which make use of Load Balancer concepts.
- It provides global publish port for a given service.
- The routing mesh uses port-based service discovery and load balancing. So to reach any service from outside the cluster you need to expose ports and reach them via the Published Port.
- Docker Engine swarm mode makes it easy to publish ports for services to make them available to resources outside the swarm.
- All nodes participate in an ingress routing mesh.
- The routing mesh enables each node in the swarm to accept connections on published ports for any service running in the swarm, even if there's no task running on the node.
- The routing mesh routes all incoming requests to published ports on available nodes to an active container.
- To use the ingress network in the swarm, you need to have the following ports open between the swarm nodes before you enable swarm mode:

1. Port 7946 TCP/UDP for container network discovery.

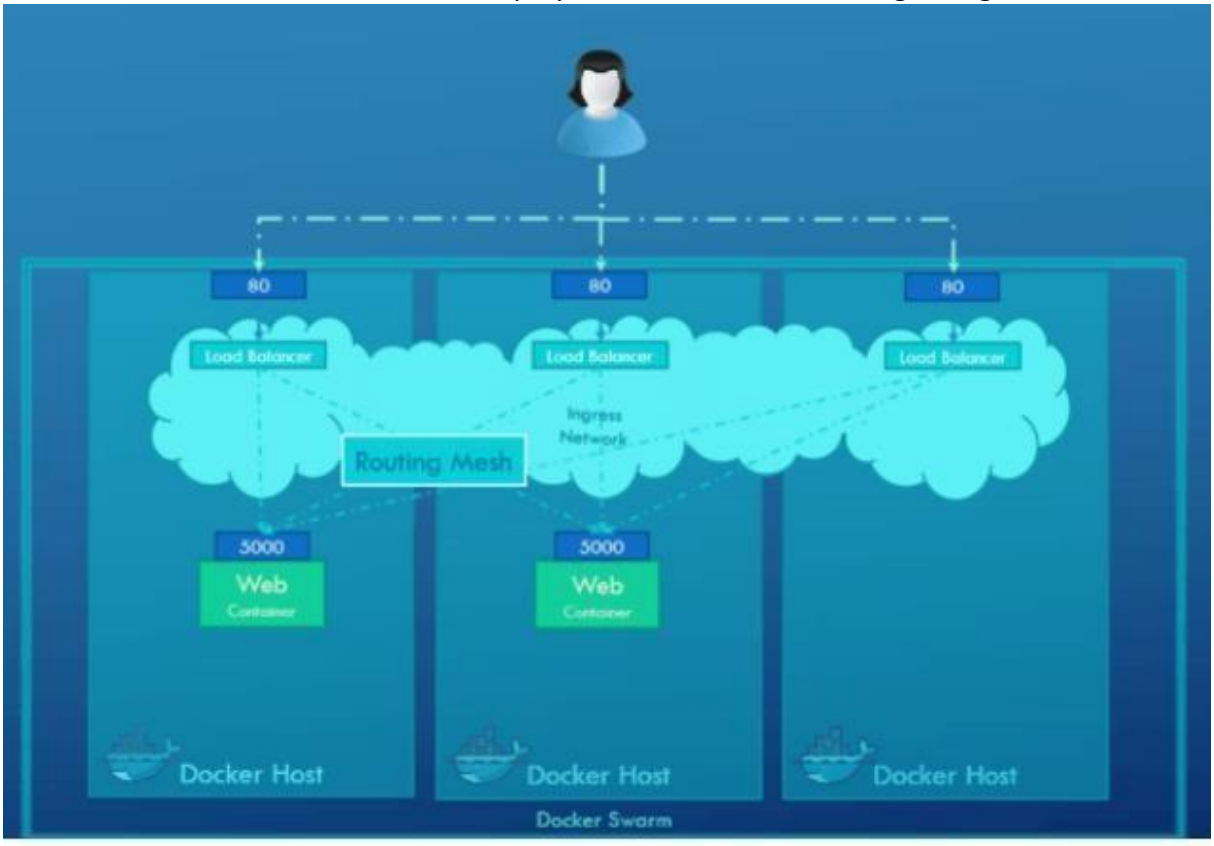
2. Port 4789 UDP for the container ingress network.

How do you access a service that could be started anywhere in your cluster?

- Docker Swarm has a very useful tool to solve this problem called the Swarm routing mesh.
- The routing mesh manages ingress into your running containers. By default, Swarm makes all services accessible via their published port on each Docker host.



- The Swarm routing mesh has its pros and cons. This default configuration has its limitations, but it is designed to make getting started as easy as possible. As your applications get more complex, the routing mesh can be configured to behave differently, and different services can be deployed to use different routing configurations



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.

Docker – version

You can check the currently used Docker version on your system through this command - $ docker –v

Docker pull

This command can pull the images from docker's hub or repository that is hub.docker.com $ docker pull ubuntu

All the images of the hub will be cached and stored from docker's hub.

Docker run

You can create a container from the image through this command.

$ docker run –it –d ubuntu **Docker**

ps

To check the running containers or to know that how many containers are running right now, you can use this command:

$ docker ps

Docker ps –a

To view all the running and exited containers, you can use this command: $ docker ps –a

Docker exec

To access the running container, you can use this command:

$ docker exec it <container id> bash

Docker stop

To stop the running container, we can use this command:

$ docker stop <container id>

Docker kill

The containers get killed after getting stopped by this command. In Docker, stop command container gets the full time to shut down, but when you need to shut down any container immediately then you can kill the Docker container through kill command.

$ docker kill <container id>

Docker commit

To create a new image of the edited container on the local system, you can use this command: $ docker commit <container id> <username/imagename>

Docker login

To login the docker hub repository, you can use this command:

$ docker login

Docker push

You can push a new image into Docker hub through this command:

$ docker push <username/image name>

Docker images

All locally stored images in docker hub will be listed through this command:

$ docker images

Docker rm

If you want to delete any stopped container then this command can help you:

$ docker rm <container id>

Docker build

If you want to build an image from a Docker file then you can use this command:

$ docker build <path to docker file>

Apart from the above-listed Docker commands cheat sheet, one can also use other commands for Docker like 'docker export' command that can export a container's filesystem as an archive file or 'docker attach' that can attach any running container, etc.

Docker network

To view Docker networks, run: docker network ls

To get further details on networks, run: docker network inspect

docker network inspect bridge

docker network create --driver bridge <bridge_network_name>

docker run --net=<bridge_network_name> --name=my_psql_db postgres

my_multi_host_network

Create the overlay network in a similar manner to the bridge network (network name):

docker network create --driver overlay <bridge_network_name>

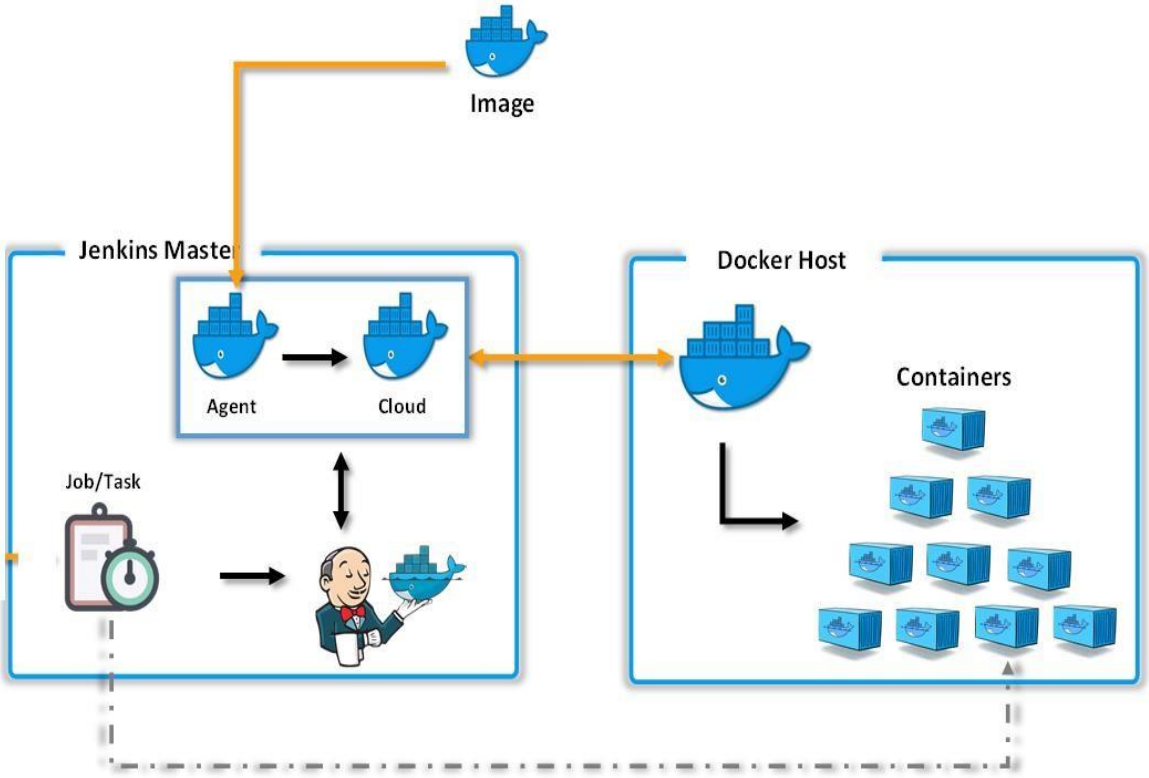
Launch containers on each host; make sure you specify the network name:

docker run -itd -net=<bridge_network_name> my_python_app we can use host network directly.

docker run -d --name web1 --net=host <image_name>
docker run -d --name web1 -net=host nginx

docker run -d --name web2 --net=none <image_name>

How jenkins will manage the docker containers



RT_Technologies

+91-8985255875