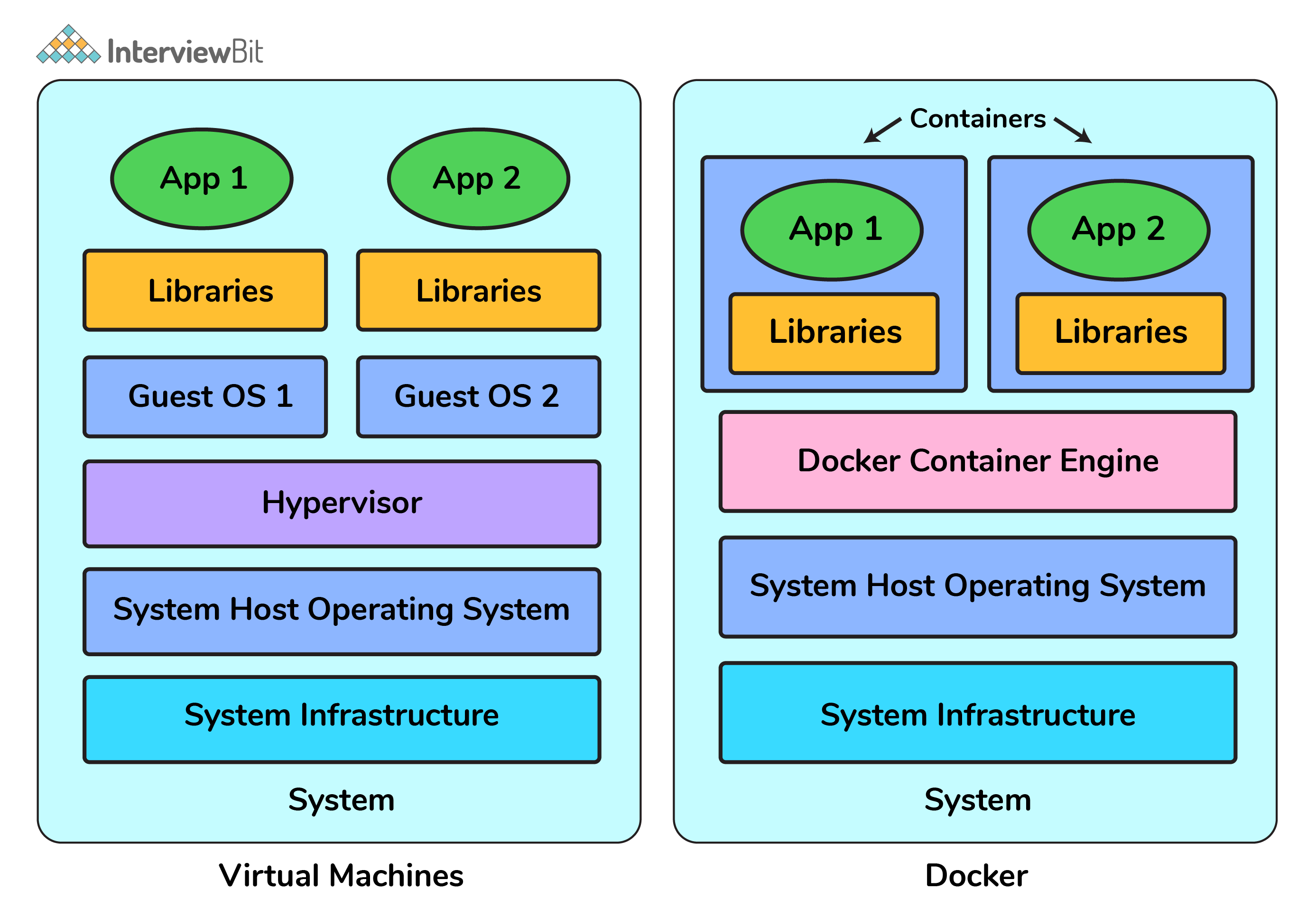
* **[Questions](https://www.interviewbit.com/docker-interview-questions/" \l "docker-basic-questions" \o "Docker Basic Interview Questions)**
  + [1.Can you tell something about docker container?](https://www.interviewbit.com/docker-interview-questions/#describe-docker-container)
  + [2.What are docker images?](https://www.interviewbit.com/docker-interview-questions/#docker-images)
  + [3.What is a DockerFile?](https://www.interviewbit.com/docker-interview-questions/#what-is-a-dockerfile)
  + [4.Can you tell what is the functionality of a hypervisor?](https://www.interviewbit.com/docker-interview-questions/#hypervisor-functionality)
  + [5.What can you tell about Docker Compose?](https://www.interviewbit.com/docker-interview-questions/#docker-compose)
  + [6.Can you tell something about docker namespace?](https://www.interviewbit.com/docker-interview-questions/#docker-namespace)
  + [7.What is the docker command that lists the status of all docker containers?](https://www.interviewbit.com/docker-interview-questions/#docker-command-that-lists-docker-containers)
  + [8.On what circumstances will you lose data stored in a container?](https://www.interviewbit.com/docker-interview-questions/#cause-for-data-loss-in-container)
  + [9.What is docker image registry?](https://www.interviewbit.com/docker-interview-questions/#docker-image-registry)
  + [10.How many Docker components are there?](https://www.interviewbit.com/docker-interview-questions/#how-many-docker-components-are-there)
  + [11.What is a Docker Hub?](https://www.interviewbit.com/docker-interview-questions/#what-is-docker-hub)
  + [12.What command can you run to export a docker image as an archive?](https://www.interviewbit.com/docker-interview-questions/#command-to-export-docker-image-as-an-archive)
  + [13.What command can be run to import a pre-exported Docker image into another Docker host?](https://www.interviewbit.com/docker-interview-questions/#command-to-import-pre-exported-docker-image-into-another-docker-host)
  + [14.Can a paused container be removed from Docker?](https://www.interviewbit.com/docker-interview-questions/#can-a-paused-container-be-removed-from-docker)
  + [15.What command is used to check for the version of docker client and server?](https://www.interviewbit.com/docker-interview-questions/#command-to-check-version-of-docker-client-and-server?)
* [**Docker Intermediate Interview Questions**](https://www.interviewbit.com/docker-interview-questions/#docker-intermediate-questions)
  + [16.Differentiate between virtualization and containerization.](https://www.interviewbit.com/docker-interview-questions/#difference-between-virtualization-and-containerization)
  + [17.Differentiate between COPY and ADD commands that are used in a Dockerfile?](https://www.interviewbit.com/docker-interview-questions/#dockerfile-add-vs-copy)
  + [18.Can a container restart by itself?](https://www.interviewbit.com/docker-interview-questions/#can-container-restart-by-itself)
  + [19.Can you tell the differences between a docker Image and Layer?](https://www.interviewbit.com/docker-interview-questions/#docker-image-vs-layer)
  + [20.What is the purpose of the volume parameter in a docker run command?](https://www.interviewbit.com/docker-interview-questions/#volume-parameter-in-docker-run-command)
  + [21.Where are docker volumes stored in docker?](https://www.interviewbit.com/docker-interview-questions/#docker-volumes-stored-in-docker)
  + [22.What does the docker info command do?](https://www.interviewbit.com/docker-interview-questions/#docker-info-command)
  + [23.Can you tell the what are the purposes of up, run, and start commands of docker compose?](https://www.interviewbit.com/docker-interview-questions/#up-run-start-commands-of-docker-compose)
  + [24.What are the basic requirements for the docker to run on any system?](https://www.interviewbit.com/docker-interview-questions/#can-docker-run-on-any-system)
  + [25.Can you tell the approach to login to the docker registry?](https://www.interviewbit.com/docker-interview-questions/#docker-registry-login)
  + [26.List the most commonly used instructions in Dockerfile?](https://www.interviewbit.com/docker-interview-questions/#commonly-used-instructions-in-dockerfile)
  + [27.Can you differentiate between Daemon Logging and Container Logging?](https://www.interviewbit.com/docker-interview-questions/#daemon-logging-vs-container-logging)
  + [28.What is the way to establish communication between docker host and Linux host?](https://www.interviewbit.com/docker-interview-questions/#communication-between-docker-host-and-linux-host)
  + [29.What is the best way of deleting a container?](https://www.interviewbit.com/docker-interview-questions/#delete-docker-containers)
  + [30.Can you tell the difference between CMD and ENTRYPOINT?](https://www.interviewbit.com/docker-interview-questions/#docker-cmd-vs-entrypoint)
* [**Docker Advanced Interview Questions**](https://www.interviewbit.com/docker-interview-questions/#docker-advanced-questions)
  + [31.Can we use JSON instead of YAML while developing docker-compose file in Docker?](https://www.interviewbit.com/docker-interview-questions/#possible-use-json-instead-of-yaml-for-docker-compose-file)
  + [32.How many containers you can run in docker and what are the factors influencing this limit?](https://www.interviewbit.com/docker-interview-questions/#docker-container-memory-limit)
  + [33.Describe the lifecycle of Docker Container?](https://www.interviewbit.com/docker-interview-questions/#docker-container-lifecycle)
  + [34.How to use docker for multiple application environments?](https://www.interviewbit.com/docker-interview-questions/#docker-multiple-application-environments)
  + [35.How will you ensure that a container 1 runs before container 2 while using docker compose?](https://www.interviewbit.com/docker-interview-questions/#docker-compose-container-scenario)
* [**Conclusion**](https://www.interviewbit.com/docker-interview-questions/#docker-conclusion)
  + [36.Conclusion](https://www.interviewbit.com/docker-interview-questions/#docker-conclusion)
* [**Docker Interview MCQs**](https://www.interviewbit.com/docker-interview-questions/#docker-interview-mcqs)

**Introduction to Docker:**  
Docker is a very popular and powerful open-source containerization platform that is used for building, deploying, and running applications. Docker allows you to decouple the application/software from the underlying infrastructure.  
  
**What is a Container?**  
A container is a standard unit of software bundled with dependencies so that applications can be deployed fast and reliably b/w different computing platforms.

* Docker can be visualized as a big ship (docker) carrying huge boxes of products (containers).
* Docker container doesn’t require the installation of a separate operating system. Docker just relies or makes use of the kernel’s resources and its functionality to allocate them for the CPU and memory it relies on the kernel’s functionality and uses resource isolation for CPU and memory, and separate namespaces to isolate the application’s view of the OS (operating system).



**Why Learn Docker?**  
Application development is a lot more than just writing code! They involve a lot of behind-the-scenes things like usage of multiple frameworks and architectures for every stage of its lifecycle which makes the process more complex and challenging. Using the nature of containerization helps developers to simplify and efficiently accelerate the application workflow along with giving them the liberty to develop using their own choice of technology and development environments.

* All these aspects form the core part of DevOps which becomes all the more important for any developer to know these in order to improve productivity, fasten the development along with keeping in mind the factors of application scalability and more efficient resource management.
* Imagine containers as a very lightweight pre-installed box with all the packages, dependencies, software required by your application, just deploy to production with minimal configuration changes.
* Lots of companies like PayPal, Spotify, Uber, etc use Docker to simplify the operations and to bring the infrastructure and security closer to make more secure applications.
* Being portable, Containers can be deployed on multiple platforms like bare instances, virtual machines, Kubernetes platform etc. as per requirements of scale or desired platform.

## Docker Basic Interview Questions

### 1. Can you tell something about docker container?

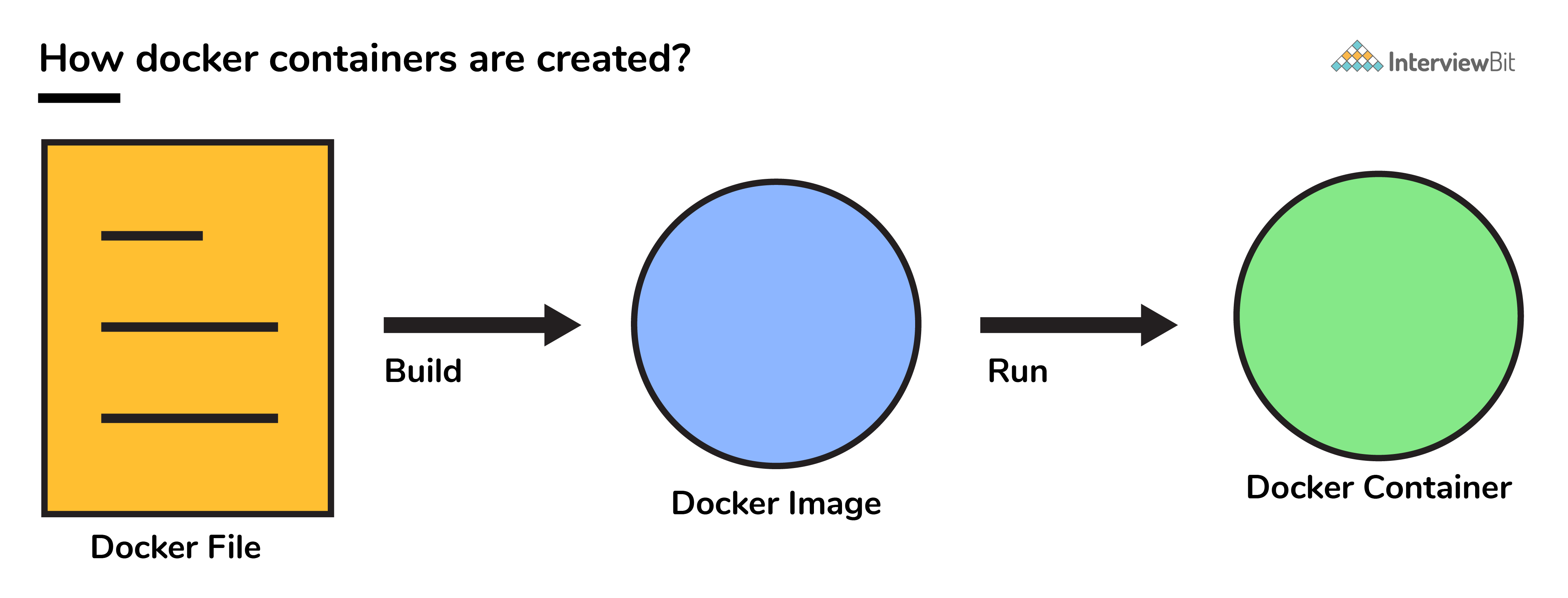
* In simplest terms, docker containers consist of applications and all their dependencies.
* They share the kernel and system resources with other containers and run as isolated systems in the host operating system.
* The main aim of docker containers is to get rid of the infrastructure dependency while deploying and running applications. This means that any containerized application can run on any platform irrespective of the infrastructure being used beneath.
* Technically, they are just the runtime instances of docker images.

### 2. What are docker images?

They are executable packages(bundled with application code & dependencies, software packages, etc.) for the purpose of creating containers. Docker images can be deployed to any docker environment and the containers can be spun up there to run the application.

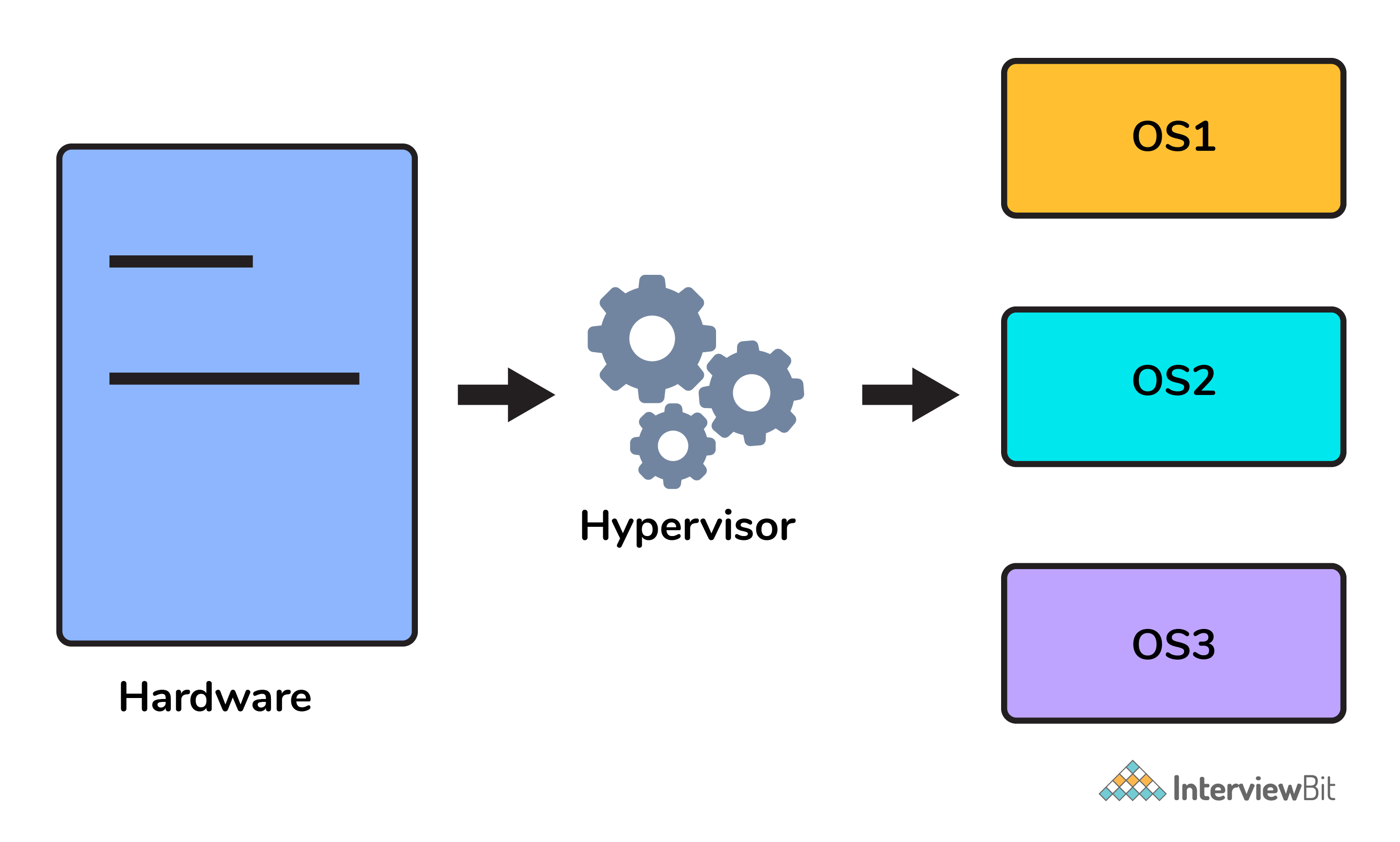
### 3. What is a DockerFile?

* It is a text file that has all commands which need to be run for building a given image.



### 4. Can you tell what is the functionality of a hypervisor?

A hypervisor is a software that makes virtualization happen because of which is sometimes referred to as the Virtual Machine Monitor. This divides the resources of the host system and allocates them to each guest environment installed.



* This means that multiple OS can be installed on a single host system. Hypervisors are of 2 types:  
    
  1. **Native Hypervisor:** This type is also called a Bare-metal Hypervisor and runs directly on the underlying host system which also ensures direct access to the host hardware which is why it does not require base OS.  
  2. **Hosted Hypervisor:** This type makes use of the underlying host operating system which has the existing OS installed.

### 5. What can you tell about Docker Compose?

It is a YAML file consisting of all the details regarding various services, networks, and volumes that are needed for setting up the Docker-based application. So, docker-compose is used for creating multiple containers, host them and establish communication between them. For the purpose of communication amongst the containers, ports are exposed by each and every container.

### 6. Can you tell something about docker namespace?

A namespace is basically a Linux feature that ensures OS resources partition in a mutually exclusive manner. This forms the core concept behind containerization as namespaces introduce a layer of isolation amongst the containers. In docker, the namespaces ensure that the containers are portable and they don't affect the underlying host. Examples for namespace types that are currently being supported by Docker – PID, Mount, User, Network, IPC.

### 7. What is the docker command that lists the status of all docker containers?

In order to get the status of all the containers, we run the below command: docker ps -a

### 8. On what circumstances will you lose data stored in a container?

The data of a container remains in it until and unless you delete the container.

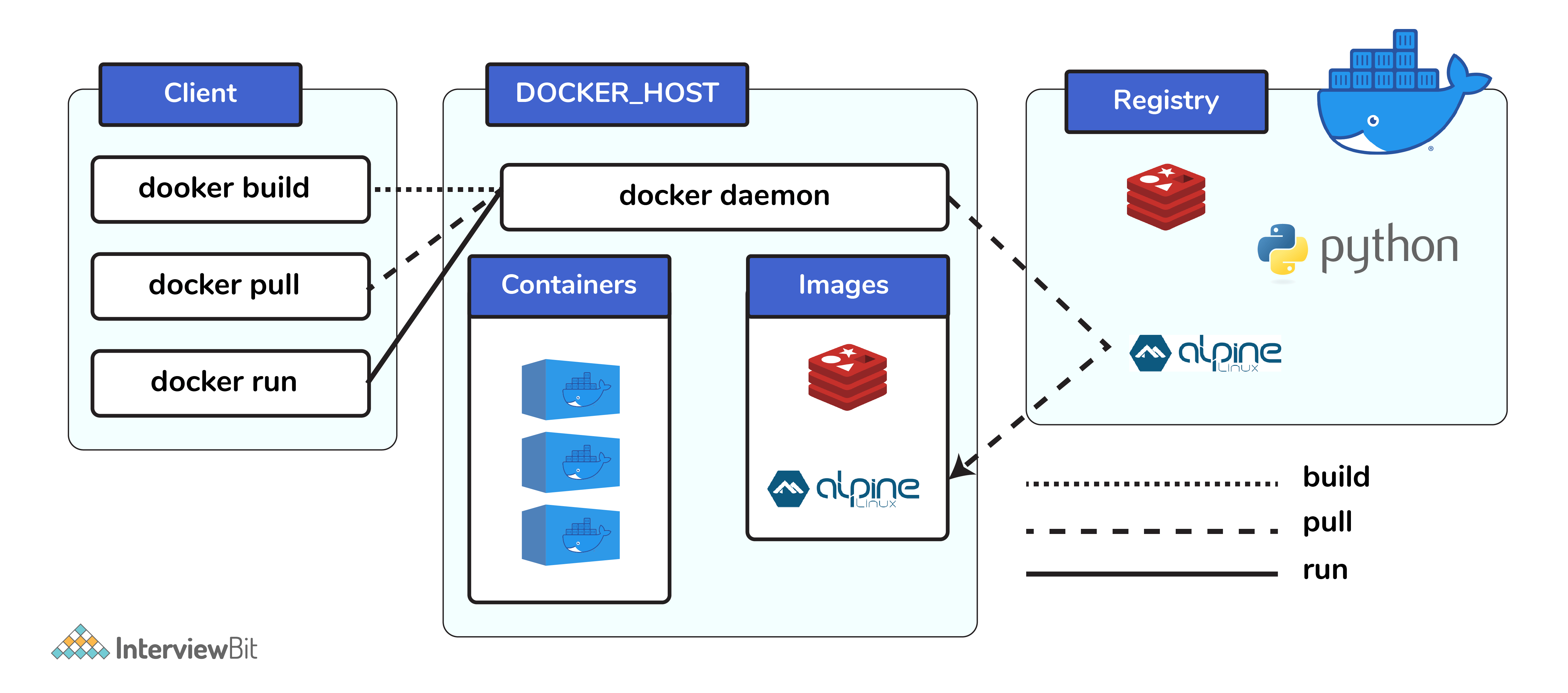
### 9. What is docker image registry?

* A Docker image registry, in simple terms, is an area where the docker images are stored. Instead of converting the applications to containers each and every time, a developer can directly use the images stored in the registry.
* This image registry can either be public or private and Docker hub is the most popular and famous public registry available.

### 10. How many Docker components are there?

There are three docker components, they are - Docker Client, Docker Host, and Docker Registry.

* **Docker Client:** This component performs “build” and “run” operations for the purpose of opening communication with the docker host.
* **Docker Host:** This component has the main docker daemon and hosts containers and their associated images. The daemon establishes a connection with the docker registry.
* **Docker Registry:** This component stores the docker images. There can be a public registry or a private one. The most famous public registries are Docker Hub and Docker Cloud.



### 11. What is a Docker Hub?

* It is a public cloud-based registry provided by Docker for storing public images of the containers along with the provision of finding and sharing them.
* The images can be pushed to Docker Hub through the docker push command.

### 12. What command can you run to export a docker image as an archive?

This can be done using the docker save command and the syntax is: docker save -o <exported\_name>.tar <container-name>

### 13. What command can be run to import a pre-exported Docker image into another Docker host?

This can be done using the docker load command and the syntax is docker load -i <export\_image\_name>.tar

### 14. Can a paused container be removed from Docker?

No, it is not possible! A container MUST be in the stopped state before we can remove it.

### 15. What command is used to check for the version of docker client and server?

* The command used to get all version information of the client and server is the docker version.
* To get only the server version details, we can run docker version --format '{{.Server.Version}}'

## Docker Intermediate Interview Questions

### 16. Differentiate between virtualization and containerization.

The question indirectly translates to explaining the difference between virtual machines and Docker containers.

| **Virtualization** | **Containerization** |
| --- | --- |
| This helps developers to run and host multiple **OS** on the hardware of a single physical server. | This helps developers to deploy multiple **applications** using the same operating system on a single virtual machine or server. |
| **Hypervisors** provide overall virtual machines to the guest operating systems. | **Containers** ensure isolated environment/ user spaces are provided for running the applications. Any changes done within the container do not reflect on the host or other containers of the same host. |
| These virtual machines form an **abstraction of the system hardware** **layer**this means that each virtual machine on the host acts like a physical machine. | Containers form **abstraction of the application** **layer** which means that each container constitutes a different application. |

### 17. Differentiate between COPY and ADD commands that are used in a Dockerfile?

Both the commands have similar functionality, but COPY is more preferred because of its higher transparency level than that of ADD.  
COPY provides just the basic support of copying local files into the container whereas ADD provides additional features like remote URL and tar extraction support.

### 18. Can a container restart by itself?

* Yes, it is possible only while using certain docker-defined policies while using the docker run command. Following are the available policies:  
    
  1. **Off:** In this, the container won’t be restarted in case it's stopped or it fails.  
  2. **On-failure**: Here, the container restarts by itself only when it experiences failures not associated with the user.  
  3. **Unless-stopped:** Using this policy, ensures that a container can restart only when the command is executed to stop it by the user.  
  4. **Always:** Irrespective of the failure or stopping, the container always gets restarted in this type of policy.  
    
  These policies can be used as:  
  docker run -dit — restart [restart-policy-value] [container\_name]

### 19. Can you tell the differences between a docker Image and Layer?

**Image:** This is built up from a series of read-only layers of instructions. An image corresponds to the docker container and is used for speedy operation due to the caching mechanism of each step.  
  
**Layer:** Each layer corresponds to an instruction of the image’s Dockerfile. In simple words, the layer is also an image but it is the image of the instructions run.  
  
Consider the example Dockerfile below.  
FROM ubuntu:18.04 COPY . /myapp RUN make /myapp CMD python /myapp/app.py Importantly, each layer is only a set of differences from the layer before it.   
  
- The result of building this docker file is an image. Whereas the instructions present in this file add the layers to the image. The layers can be thought of as intermediate images. In the example above, there are 4 instructions, hence 4 layers are added to the resultant image.

### 20. What is the purpose of the volume parameter in a docker run command?

* The syntax of docker run when using the volumes is: docker run -v host\_path:docker\_path <container\_name>
* The volume parameter is used for syncing a directory of a container with any of the host directories. Consider the below command as an example: docker run -v /data/app:usr/src/app myapp  
  The above command mounts the directory  /data/app in the host to the usr/src/app directory. We can sync the container with the data files from the host without having the need to restart it.
* This also ensures data security in cases of container deletion. This ensures that even if the container is deleted, the data of the container exists in the volume mapped host location making it the easiest way to store the container data.

### 21. Where are docker volumes stored in docker?

Volumes are created and managed by Docker and cannot be accessed by non-docker entities. They are stored in Docker host filesystem at /var/lib/docker/volumes/

### 22. What does the docker info command do?

The command gets detailed information about Docker installed on the host system. The information can be like what is the number of containers or images and in what state they are running and hardware specifications like total memory allocated, speed of the processor, kernel version, etc.

### 23. Can you tell the what are the purposes of up, run, and start commands of docker compose?

* Using the up command for keeping a docker-compose up (ideally at all times), we can start or restart all the networks, services, and drivers associated with the app that are specified in the docker-compose.yml file. Now if we are running the docker-compose up in the “attached” mode then all the logs from the containers would be accessible to us. In case the docker-compose is run in the “detached” mode, then once the containers are started, it just exits and shows no logs.
* Using the run command, the docker-compose can run one-off or ad-hoc tasks based on the business requirements. Here, the service name has to be provided and the docker starts only that specific service and also the other services to which the target service is dependent (if any).  
  - This command is helpful for testing the containers and also performing tasks such as adding or removing data to the container volumes etc.
* Using the start command, only those containers can be restarted which were already created and then stopped. This is not useful for creating new containers on its own.

### 24. What are the basic requirements for the docker to run on any system?

Docker can run on both Windows and Linux platforms.

* For the Windows platform, docker atleast needs Windows 10 64bit with 2GB RAM space. For the lower versions, docker can be installed by taking help of the toolbox. Docker can be downloaded from <https://docs.docker.com/docker-for-windows/> website.
* For Linux platforms, Docker can run on various Linux flavors such as Ubuntu >=12.04, Fedora >=19, RHEL >=6.5, CentOS >=6 etc.

### 25. Can you tell the approach to login to the docker registry?

Using the docker login command credentials to log in to their own cloud repositories can be entered and accessed.

### 26. List the most commonly used instructions in Dockerfile?

* **FROM:** This is used to set the base image for upcoming instructions. A docker file is considered to be valid if it starts with the FROM instruction.
* **LABEL:** This is used for the image organization based on projects, modules, or licensing. It also helps in automation as we specify a key-value pair while defining a label that can be later accessed and handled programmatically.
* **RUN:** This command is used to execute instructions following it on the top of the current image in a new layer. Note that with each RUN command execution, we add layers on top of the image and then use that in subsequent steps.
* **CMD:** This command is used to provide default values of an executing container. In cases of multiple CMD commands the last instruction would be considered.

### 27. Can you differentiate between Daemon Logging and Container Logging?

* In docker, logging is supported at 2 levels and they are logging at the Daemon level or logging at the Container level.
* **Daemon Level:** This kind of logging has four levels- Debug, Info, Error, and Fatal.  
  - Debug has all the data that happened during the execution of the daemon process.  
  - Info carries all the information along with the error information during the execution of the daemon process.  
  - Errors have those errors that occurred during the execution of the daemon process.  
  - Fatal has the fatal errors that occurred during the execution.
* **Container Level:**  
  - Container level logging can be done using the command: sudo docker run –it <container\_name> /bin/bash  
  - In order to check for the container level logs, we can run the command: sudo docker logs <container\_id>

### 28. What is the way to establish communication between docker host and Linux host?

This can be done using networking by identifying the “ipconfig” on the docker host. This command ensures that an ethernet adapter is created as long as the docker is present in the host.

### 29. What is the best way of deleting a container?

We need to follow the following two steps for deleting a container:  
- docker stop <container\_id>  
- docker rm <container\_id>

### 30. Can you tell the difference between CMD and ENTRYPOINT?

* CMD command provides executable defaults for an executing container. In case the executable has to be omitted then the usage of ENTRYPOINT instruction along with the JSON array format has to be incorporated.
* ENTRYPOINT specifies that the instruction within it will always be run when the container starts.   
  This command provides an option to configure the parameters and the executables. If the DockerFile does not have this command, then it would still get inherited from the base image mentioned in the FROM instruction.  
  - The most commonly used ENTRYPOINT is /bin/sh or /bin/bash for most of the base images.
* As part of good practices, every DockerFile should have at least one of these two commands.

## Docker Advanced Interview Questions

### 31. Can we use JSON instead of YAML while developing docker-compose file in Docker?

Yes! It can be used. In order to run docker-compose with JSON, docker-compose -f docker-compose.json up can be used.

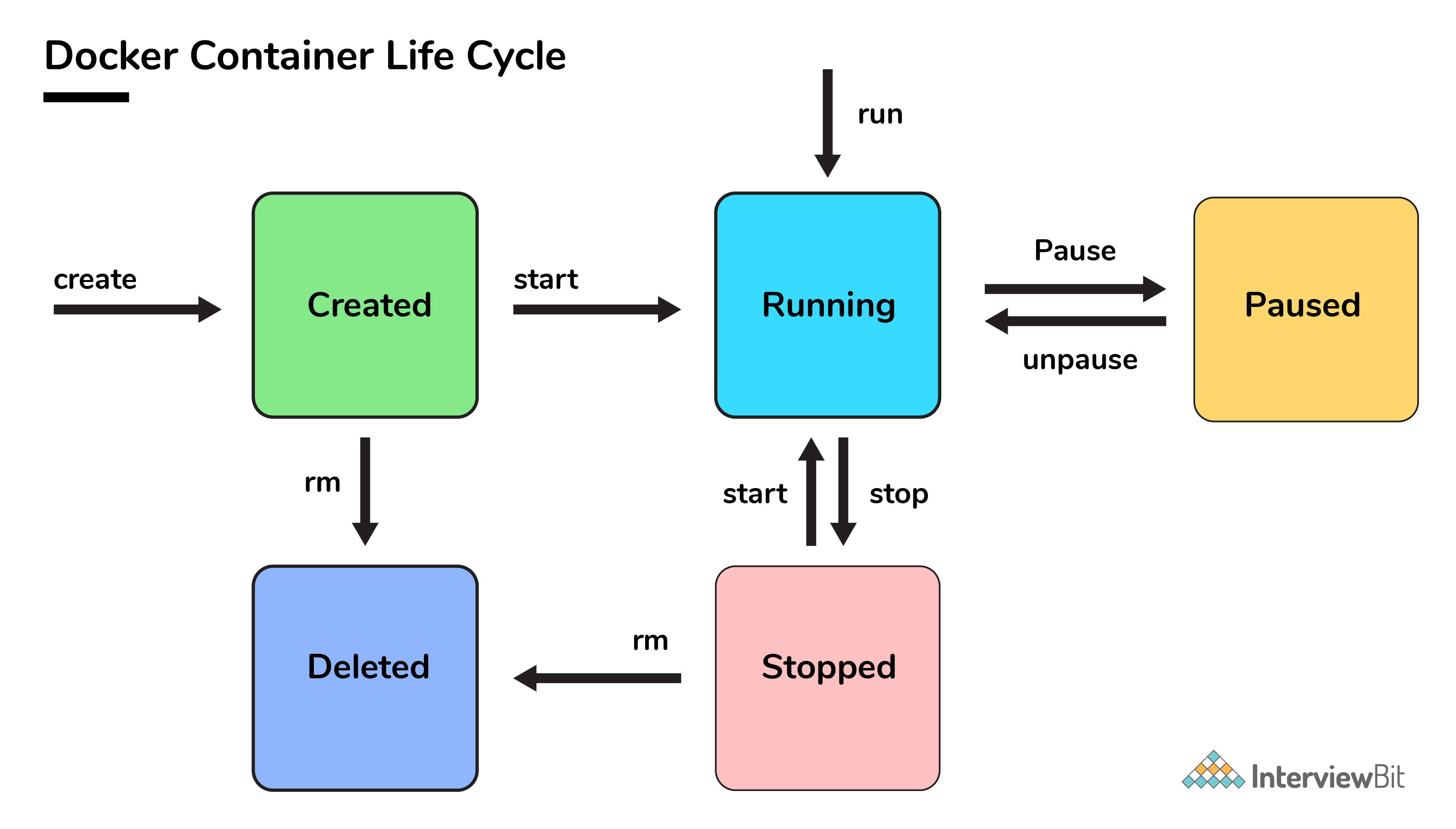
### 32. How many containers you can run in docker and what are the factors influencing this limit?

There is no clearly defined limit to the number of containers that can be run within docker. But it all depends on the limitations - more specifically hardware restrictions. The size of the app and the CPU resources available are 2 important factors influencing this limit. In case your application is not very big and you have abundant CPU resources, then we can run a huge number of containers.

### 33. Describe the lifecycle of Docker Container?

The different stages of the docker container from the start of creating it to its end are called the docker container life cycle.   
The most important stages are:

* **Created:** This is the state where the container has just been created new but not started yet.
* **Running:** In this state, the container would be running with all its associated processes.
* **Paused:** This state happens when the running container has been paused.
* **Stopped:** This state happens when the running container has been stopped.
* **Deleted:** In this, the container is in a dead state.



### 34. How to use docker for multiple application environments?

* Docker-compose feature of docker will come to help here. In the docker-compose file, we can define multiple services, networks, and containers along with the volume mapping in a clean manner, and then we can just call the command “docker-compose up”.
* When there are multiple environments involved - it can be either dev, staging, uat, or production servers, we would want to define the server-specific dependencies and processes for running the application. In this case, we can go ahead with creating environment-specific docker-compose files of the name “docker-compose.{environment}.yml” and then based on the environment, we can set up and run the application.

### 35. How will you ensure that a container 1 runs before container 2 while using docker compose?

Docker-compose does not wait for any container to be “ready” before going ahead with the next containers. In order to achieve the order of execution, we can use:

* The “depends\_on” which got added in version 2 of docker-compose can be used as shown in a sample docker-compose.yml file below:

version: "2.4"

services:

backend:

build: .

depends\_on:

- db

db:

image: postgres

The introduction of service dependencies has various causes and effects:

* The docker-compose up command starts and runs the services in the dependency order specified. For the above example, the DB container is started before the backend.
* docker-compose up SERVICE\_NAME by default includes the dependencies associated with the service. In the given example, running docker-compose up backend creates and starts DB (dependency of backend).
* Finally, the command docker-compose stop also stops the services in the order of the dependency specified. For the given example, the backend service is stopped before the DB service.

### **1. What is Docker?**

[Docker](https://www.simplilearn.com/tutorials/docker-tutorial/getting-started-with-docker) is an open-source containerization platform. It is used to automate the deployment of any application, using lightweight, portable containers.

### **2. What are Docker’s most notable features?**

Docker’s most essential features include:

* Application agility
* Developer productivity
* Easy modeling
* Operational efficiencies
* Placement and affinity
* Version control

### **3. Why should anyone use Docker? What does it offer?**

Docker gives users many incentives for adoption, such as:

* An efficient and easy initial set up experience
* The means to describe an application lifecycle in detail
* Simple configuration and smooth interaction with [Docker Compose](https://www.simplilearn.com/tutorials/docker-tutorial/docker-compose)
* Complete and well-detailed documentation
* Ability to run on a PC or enterprise IT system with equal ease

### **4. What about the opposite? Does Docker have any downsides?**

Docker isn’t perfect. It comes with its share of drawbacks, including:

* Lacks a storage option
* Monitoring options are less than ideal
* You can’t automatically reschedule inactive nodes
* Automatic horizontal scaling set up is complicated

### **5. Name and explain the various Docker components.**

The three main [Docker components](https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker) are:

1. **Docker Client.** Performs Docker build pull and run operations to open up communication with the Docker Host. The Docker command then employs Docker API to call any queries to run.
2. **Docker Host.** Contains Docker daemon, containers, and associated images. The Docker daemon establishes a connection with the Registry. The stored images are the type of metadata dedicated to containerized applications.
3. **Registry.** This is where [Docker images](https://www.simplilearn.com/tutorials/docker-tutorial/docker-images) are stored. There are two of them, a public registry and a private one. [Docker Hub](https://www.simplilearn.com/tutorials/docker-tutorial/docker-hub) and Docker Cloud are two public registries available for use by anyone.

### **6. What is a container?**

Containers are deployed applications bundled with all necessary dependencies and configuration files. All of the elements share the same OS kernel. Since the container isn’t tied to any one IT infrastructure, it can run on a different system or the cloud.

### **7. Explain virtualization.**

[Virtualization](https://www.simplilearn.com/virtualization-in-cloud-computing-article) is the means of employing software (such as Hypervisor) to create a virtual version of a resource such as a server, [data storage](https://www.simplilearn.com/big-data-era-data-storage-rules-article), or application. Virtualization lets you divide a system into a series of separate sections, each one acting as a distinct individual system. The virtual environment is called a virtual machine.

### **8. What’s the difference between virtualization and containerization?**

Virtualization is an abstract version of a physical machine, while containerization is the abstract version of an application.

### **9. Last simple question…Describe a Docker container’s lifecycle.**

Although there are several different ways of describing the steps in a Docker container’s lifecycle, the following is the most common:

1. Create container
2. Run container
3. Pause container
4. Unpause container
5. Start container
6. Stop container
7. Restart container
8. Kill container
9. Destroy container

We will next look at the intermediate-level docker interview questions and answers.

## Exclusive Intermediate Interview Questions on Docker

### **10. Name the essential Docker commands and what they do.**

The most critical [Docker commands](https://www.simplilearn.com/tutorials/docker-tutorial/docker-commands) are:

* **Build.** Builds a Docker image file
* **Commit.** Creates a new image from container changes
* **Create.** Creates a new container
* **Dockerd.** Launches Docker daemon
* **Kill.** Kills a container

### **11. What are Docker object labels?**

Labels are the mechanism for applying metadata to Docker objects such as containers, images, local daemons, networks, volumes, and nodes.

### **12. How do you find stored Docker volumes?**

Use the command: /var/lib/docker/volumes

### **13. How do you check the versions of Docker Client and Server?**

This command gives you all the information you need: $ docker version

### **14. Show how you would create a container from an image.**

To create a container, you pull an image from the Docker repository and run it using the following command: $ docker run -it -d <image\_name>

### **15. How about a command to stop the container?**

Use the following command: $ sudo docker stop container name

### **16. How would you list all of the containers currently running?**

Use the command: $ docker ps

### **17. What’s involved in scaling a Docker container?**

Docker containers have the potential to be scaled to any level needed. Thanks to the platform’s flexibility, you can have anything from a few hundred to a few thousand, to millions of containers, providing they all have continual, unconstrained access to the required memory and OS.

### **18. What do you know about the Docker system prune?**

It’s a command used to remove all stopped containers, unused networks, build caches, and dangling images. Prune is one of the most useful commands in Docker. The syntax is:  $ docker system prune

### **19. List some of the more advanced Docker commands and what they do.**

Some advanced commands include:

* **Docker info.** Displays system-wide information regarding the Docker installation
* **Docker pull.** Downloads an image
* **Docker stats.** Provides you with container information
* **Docker images.** Lists downloaded images

### **20. Can you lose data stored in a container?**

Any data stored in a container remains there unless you delete the container.

### **21. What platforms can you run Docker on?**

The Linux platforms are:

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

Docker can also run on the following cloud-based platforms:

* Amazon EC2
* Amazon ECS
* Google Compute Engine
* Microsoft Azure
* Rackspace

### **22. Which is the best method for removing a container: the command “stop container” followed by the command “remove the container,” the rm command by itself?**

Stop the container first, then remove it. Here’s how:

* $ docker stop <coontainer\_id>
* $ docker rm -f <container\_id>

### **23. Can a container restart on its own?**

Since the default flag -reset is set to false, a container cannot restart by itself.

### **24. How do Docker daemon and the Docker client communicate with each other?**

You use a combination of Rest API, socket.IO, and TCP to facilitate communication.

### **25. Can you implement continuous development (CD) and continuous integration (CI) in Docker?**

Yes, you can. You can run [Jenkins](https://www.simplilearn.com/tutorials/jenkins-tutorial/what-is-jenkins) on Docker and use Docker Compose to run integration tests.

### **26. Finally, how do you create a Docker swarm?**

Use the following command: docker swarm init –advertise-addr <manager IP>

**1. What is Hypervisor?**

A hypervisor is a software that makes virtualization possible. It is also called Virtual Machine Monitor. It divides the host system and allocates the resources to each divided virtual environment. You can basically have multiple OS on a single host system. There are two types of Hypervisors:

* Type 1: It’s also called Native Hypervisor or Bare metal Hypervisor. It runs directly on the underlying host system. It has direct access to your host’s system hardware and hence does not require a base server operating system.
* Type 2: This kind of hypervisor makes use of the underlying host operating system. It’s also called Hosted Hypervisor.

### ****2. What is virtualization?****

Virtualization is the process of creating a software-based, virtual version of something(compute storage, servers, application, etc.). These virtual versions or environments are created from a single physical hardware system. Virtualization lets you split one system into many different sections which act like separate, distinct individual systems. A software called Hypervisor makes this kind of splitting possible. The virtual environment created by the hypervisor is called Virtual Machine.

### ****3. What is containerization?****

Let me explain this is with an example. Usually, in the software development process, code developed on one machine might not work perfectly fine on any other machine because of the dependencies. This problem was solved by the containerization concept. So basically, an application that is being developed and deployed is bundled and wrapped together with all its configuration files and dependencies. This bundle is called a container. Now when you wish to run the application on another system, the container is deployed which will give a bug-free environment as all the dependencies and libraries are wrapped together. Most famous containerization environments are Docker and Kubernetes.

**4. Difference between virtualization and containerization**

Once you’ve explained containerization and virtualization, the next expected question would be differences. The question could either be differences between virtualization and containerization or differences between virtual machines and containers. Either way, this is how you respond.

Containers provide an isolated environment for running the application. The entire user space is explicitly dedicated to the application. Any changes made inside the container is never reflected on the host or even other containers running on the same host. Containers are an abstraction of the application layer. Each container is a different application.

Whereas in Virtualization, hypervisors provide an entire virtual machine to the guest(including Kernal). Virtual machines are an abstraction of the hardware layer. Each VM is a physical machine.

### ****5. What is Docker?****

Since its a Docker interview, there will be an obvious question about what is Docker. Start with a small definition.

Docker is a containerization platform which packages your application and all its dependencies together in the form of containers so as to ensure that your application works seamlessly in any environment, be it development, test or production. Docker containers, wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries, etc. It wraps basically anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

### ****6. What is a Docker Container?****

Docker containers include the application and all of its dependencies. It shares the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud. Docker containers are basically runtime instances of Docker images.

### ****7. What are Docker Images?****

When you mention Docker images, your very next question will be “what are Docker images”.

Docker image is the source of Docker container. In other words, Docker images are used to create containers. When a user runs a Docker image, an instance of a container is created. These docker images can be deployed to any Docker environment.

### ****8. What is Docker Hub?****

Docker images create docker containers. There has to be a registry where these docker images live. This registry is Docker Hub. Users can pick up images from Docker Hub and use them to create customized images and containers. Currently, the [Docker Hub](https://hub.docker.com/) is the world’s largest public repository of image containers.

### ****9. Explain Docker Architecture?****

Docker Architecture consists of a Docker Engine which is a client-server application with three major components:

1. A server which is a type of long-running program called a daemon process (the docker command).
2. A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
3. A command line interface (CLI) client (the docker command).
4. The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

### ****10. What is a Dockerfile?****

Let’s start by giving a small explanation of Dockerfile and proceed by giving examples and commands to support your arguments.

Docker can build images automatically by reading the instructions from a file called Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build, users can create an automated build that executes several command-line instructions in succession.

### ****11. Tell us something about Docker Compose.****

Docker Compose is a YAML file which contains details about the services, networks, and volumes for setting up the Docker application. So, you can use Docker Compose to create separate containers, host them and get them to communicate with each other. Each container will expose a port for communicating with other containers.

### ****12. What is Docker Swarm?****

You are expected to have worked with Docker Swarm as it’s an important concept of Docker.

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

### ****13. What is a Docker Namespace?****

A namespace is one of the Linux features and an important concept of containers. Namespace adds a layer of isolation in containers. Docker provides various namespaces in order to stay portable and not affect the underlying host system. Few namespace types supported by Docker – PID, Mount, IPC, User, Network

### ****14. What is the lifecycle of a Docker Container?****

This is one of the most popular questions asked in Docker interviews. Docker containers have the following lifecycle:

* Create a container
* Run the container
* Pause the container(optional)
* Un-pause the container(optional)
* Start the container
* Stop the container
* Restart the container
* Kill the container
* Destroy the container

### ****15. What is Docker Machine?****

Docker machine is a tool that lets you install Docker Engine on virtual hosts. These hosts can now be managed using the docker-machine commands. Docker machine also lets you provision Docker Swarm Clusters.

### ****16. How to check for Docker Client and Docker Server version?****

The following command gives you information about Docker Client and Server versions:

$ docker version

### ****17. How do you get the number of containers running, paused and stopped?****

You can use the following command to get detailed information about the docker installed on your system.

$ docker info

You can get the number of containers running, paused, stopped, the number of images and a lot more.

### ****18. If you vaguely remember the command and you’d like to confirm it, how will you get help on that particular command?****

The following command is very useful as it gives you help on how to use a command, the syntax, etc.

$ docker --help

The above command lists all Docker commands. If you need help with one specific command, you can use the following syntax:

$ docker <command> --help

### ****19. How to login into docker repository?****

You can use the following command to login into hub.docker.com:

$ docker login

You’ll be prompted for your username and password, insert those and congratulations, you’re logged in.

### ****20. If you wish to use a base image and make modifications or personalize it, how do you do that?****

You pull an image from docker hub onto your local system

It’s one simple command to pull an image from docker hub:

$ docker pull <image\_name>

### ****21. How do you create a docker container from an image?****

Pull an image from docker repository with the above command and run it to create a container. Use the following command:

$ docker run -it -d <image\_name>

Most probably the next question would be, what does the ‘-d’ flag mean in the command?

**-d** means the container needs to start in the detached mode. Explain a little about the detach mode. Have a look at [this](https://www.edureka.co/blog/docker-commands/) blog to get a better understanding of different docker commands.

### ****22. How do you list all the running containers?****

The following command lists down all the running containers:

$ docker ps

### ****23. Suppose you have 3 containers running and out of these, you wish to access one of them. How do you access a running container?****

The following command lets us access a running container:

$ docker exec -it <container id> bash

The exec command lets you get inside a container and work with it.

**24. How to start, stop and kill a container?**

The following command is used to start a docker container:

$ docker start <container\_id>

and the following for stopping a running container:

$ docker stop <container\_id>

kill a container with the following command:

$ docker kill <container\_id>

### ****25. Can you use a container, edit it, and update it? Also, how do you make it a new and store it on the local system?****

Of course, you can use a container, edit it and update it. This sounds complicated but its actually just one command.

$ docker commit <conatainer id> <username/imagename>

### ****26. Once you’ve worked with an image, how do you push it to docker hub?****

$ docker push <username/image name>

### ****27. How to delete a stopped container?****

Use the following command to delete a stopped container:

$ docker rm <container id>

### ****28. How to delete an image from the local storage system?****

The following command lets you delete an image from the local system:

$ docker rmi <image-id>

### ****29. How to build a Dockerfile?****

Once you’ve written a Dockerfile, you need to build it to create an image with those specifications. Use the following command to build a Dockerfile:

$ docker build <path to docker file>

The next question would be when do you use “.dockerfile\_name” and when to use the entire path?

Use “.dockerfile\_name” when the dockerfile exits in the same file directory and you use the entire path if it lives somewhere else.

### ****30. Do you know why****docker system prune****is used? What does it do?****

$ docker system prune

The above command is used to remove all the stopped containers, all the networks that are not used, all dangling images and all build caches. It’s one of the most useful docker commands.

### ****31. Will you lose your data, when a docker container exists?****

No, you won’t lose any data when Docker container exits. Any data that your application writes to the container gets preserved on the disk until you explicitly delete the container. The file system for the container persists even after the container halts.

### ****32. Where all do you think Docker is being used?****

When asked such a question, respond by talking about applications of Docker. Docker is being used in the following areas:

* Simplifying configuration: Docker lets you put your environment and configuration into code and deploy it.
* Code Pipeline Management: There are different systems used for development and production. As the code travels from development to testing to production, it goes through a difference in the environment. Docker helps in maintaining the code pipeline consistency.
* Developer Productivity: Using Docker for development gives us two things – We’re closer to production and development environment is built faster.
* Application Isolation: As containers are applications wrapped together with all dependencies, your apps are isolated. They can work by themselves on any hardware that supports Docker.
* Debugging Capabilities: Docker supports various debugging tools that are not specific to containers but work well with containers.
* Multi-tenancy: Docker lets you have multi-tenant applications avoiding redundancy in your codes and deployments.
* Rapid Deployment: Docker eliminates the need to boost an entire OS from scratch, reducing the deployment time.

### ****33. How is Docker different from other containerization methods?****

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.

If you have some more points to add you can do that but make sure the above explanation is there in your answer.

### ****34. Can I use JSON instead of YAML for my compose file in Docker?****

You can use JSON instead of YAML for your compose file, to use JSON file with compose, specify the JSON filename to use, for eg:

$ docker-compose -f docker-compose.json up

### ****35. How have you used Docker in your previous position?****

Explain how you have used Docker to help rapid deployment. Explain how you have scripted Docker and used it with other tools like Puppet, Chef or Jenkins. If you have no past practical experience in Docker and instead have experience with other tools in a similar space, be honest and explain the same. In this case, it makes sense if you can compare other tools to Docker in terms of functionality.

### ****36. 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.

### ****37. What platforms does docker run on?****

Docker runs on various Linux administration:

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

It can also be used in production with Cloud platforms with the following services:

* Amazon EC2
* Amazon ECS
* Google Compute Engine
* Microsoft Azure
* Rackspace

### ****38. Is there a way to identify the status of a Docker container?****

There are six possible states a container can be at any given point – Created, Running, Paused, Restarting, Exited, Dead.

Use the following command to check for docker state at any given point:

$ docker ps

The above command lists down only running containers by default. To look for all containers, use the following command:

$ docker ps -a

### ****39. Can you remove a paused container from Docker?****

The answer is no. You cannot remove a paused container. The container has to be in the stopped state before it can be removed.

### ****40. Can a container restart by itself?****

No, it’s not possible for a container to restart by itself. By default the flag -restart is set to false.

### ****41. Is it better to directly remove the container using the rm command or stop the container followed by remove container?****

Its always better to stop the container and then remove it using the remove command.

$ docker stop <coontainer\_id>  
$ docker rm -f <container\_id>

Stopping the container and then removing it will allow sending SIG\_HUP signal to recipients. This will ensure that all the containers have enough time to clean up their tasks. This method is considered a good practice, avoiding unwanted errors.

### ****42. Will cloud overtake the use of Containerization?****

Docker containers are gaining popularity but at the same time, Cloud services are giving a good fight. In my personal opinion, Docker will never be replaced by Cloud. Using cloud services with containerization will definitely hype the game. Organizations need to take their requirements and dependencies into consideration into the picture and decide what’s best for them. Most of the companies have integrated Docker with the cloud. This way they can make the best out of both the technologies.

### ****43. How many containers can run per host?****

There can be as many containers as you wish per host. Docker does not put any restrictions on it. But you need to consider every container needs storage space, CPU and memory which the hardware needs to support. You also need to consider the application size. Containers are considered to be lightweight but very dependant on the host OS.

### ****44. Is it a good practice to run stateful applications on Docker?****

The concept behind stateful applications is that they store their data onto the local file system. You need to decide to move the application to another machine, retrieving data becomes painful. I honestly would not prefer running stateful applications on Docker.

### ****45. Suppose you have an application that has many dependant services. Will docker compose wait for the current container to be ready to move to the running of the next service?****

The answer is yes. Docker compose always runs in the dependency order. These dependencies are specifications like depends\_on, links, volumes\_from, etc.

### ****46. How will you monitor Docker in production?****

Docker provides functionalities like docker stats and docker events to monitor docker in production. Docker stats provides CPU and memory usage of the container. Docker events provide information about the activities taking place in the docker daemon.

### ****47. Is it a good practice to run Docker compose in production?****

Yes, using docker compose in production is the best practical application of docker compose. When you define applications with compose, you can use this compose definition in various production stages like CI, staging, testing, etc.

### ****48. What changes are expected in your docker compose file while moving it to production?****

These are the following changes you need make to your compose file before migrating your application to the production environment:

* Remove volume bindings, so the code stays inside the container and cannot be changed from outside the container.
* Binding to different ports on the host.
* Specify a restart policy
* Add extra services like log aggregator

### ****49. Have you used Kubernetes? If you have, which one would you prefer amongst Docker and Kubernetes?****

Be very honest in such questions. If you have used Kubernetes, talk about your experience with Kubernetes and Docker Swarm. Point out the key areas where you thought docker swarm was more efficient and vice versa. Have a look at [this](https://www.edureka.co/blog/kubernetes-vs-docker/) blog for understanding differences between Docker and Kubernetes.

You Docker interview questions are not just limited to the workarounds of docker but also other similar tools. Hence be prepared with tools/technologies that give Docker competition. One such example is Kubernetes.

### ****50. Are you aware of load balancing across containers and hosts? How does it work?****

While using docker service with multiple containers across different hosts, you come across the need to load balance the incoming traffic. Load balancing and HAProxy is basically used to balance the incoming traffic across different available(healthy) containers. If one container crashes, another container should automatically start running and the traffic should be re-routed to this new running container. Load balancing and HAProxy works around this concept.

This brings us to the end of the Docker Interview Questions article. With increasing business competition, companies have realized the importance of adapting and taking advantage of the changing market. Few things that kept them in the game were faster scaling of systems, better software delivery, adapting to new technologies, etc. That’s when docker swung into the picture and gave these companies boosting support to continue the race.

### 1. What is Docker?

Docker is an open-source lightweight containerization technology. It has gained widespread popularity in the cloud and application packaging world. It allows you to automate the deployment of applications in lightweight and portable containers.

### 2. What are the advantages of using Docker container?

Here, are a major advantage of using [Docker](https://www.guru99.com/docker-tutorial.html).

* Offers an efficient and easy initial set up
* Allows you to describe your application lifecycle in detail
* Simple configuration and interacts with Docker Compose.
* Documentation provides every bit of information.

### 3. What are the important features of Docker?

Here are the essential features of Docker:

* Easy Modeling
* Version control
* Placement/Affinity
* Application Agility
* Developer Productivity
* Operational Efficiencies

### 4. What are the main drawbacks of Docker?

Some notable drawbacks of Docker are:

* Doesn’t provide a storage option
* Offer a poor monitoring option.
* No automatic rescheduling of inactive Nodes
* Complicated automatic horizontal scaling set up

### 5. What is Docker image?

The Docker image help to create Docker containers. You can create the Docker image with the build command. Due to this, it creates a container that starts when it begins to run. Every docker images are stored in the Docker registry.

### 6. What is Docker Engine?

Docker daemon or Docker engine represents the server. The docker daemon and the clients should be run on the same or remote host, which can communicate through command-line client binary and full RESTful API.

### 7. Explain Registries

There are two types of registry is

* Public Registry
* Private Registry

Docker’s public registry is called Docker hub, which allows you to store images privately. In Docker hub, you can store millions of images.

### 8. What command should you run to see all running container in Docker?

$ docker ps

### 9. Write the command to stop the docker container

$ sudo docker stop container name

### 10. What is the command to run the image as a container?

$ sudo docker run -i -t alpine /bin/bash

### 11. What are the common instruction in Dockerfile?

The common instruction in Dockerfile are: FROM, LABEL, RUN, and CMD.

### 12. What is memory-swap flag?

Memory-swap is a modified flag that only has meaning if- memory is also set. Swap allows the container to write express memory requirements to disk when the container has exhausted all the RAM which is available to it.

### 13. Explain Docker Swarm?

Docker Swarm is native gathering for docker which helps you to a group of Docker hosts into a single and virtual docker host. It offers the standard docker application program interface.

### 14. How can you monitor the docker in production environments?

Docker states and Docker Events are used to monitoring docker in the production environment.

### 15. What the states of Docker container?

Important states of Docker container are:

* Running
* Paused
* Restarting
* Exited

### 16. What is Docker hub?

Docker hub is a cloud-based registry that which helps you to link to code repositories. It allows you to build, test, store your image in Docker cloud. You can also deploy the image to your host with the help of Docker hub.

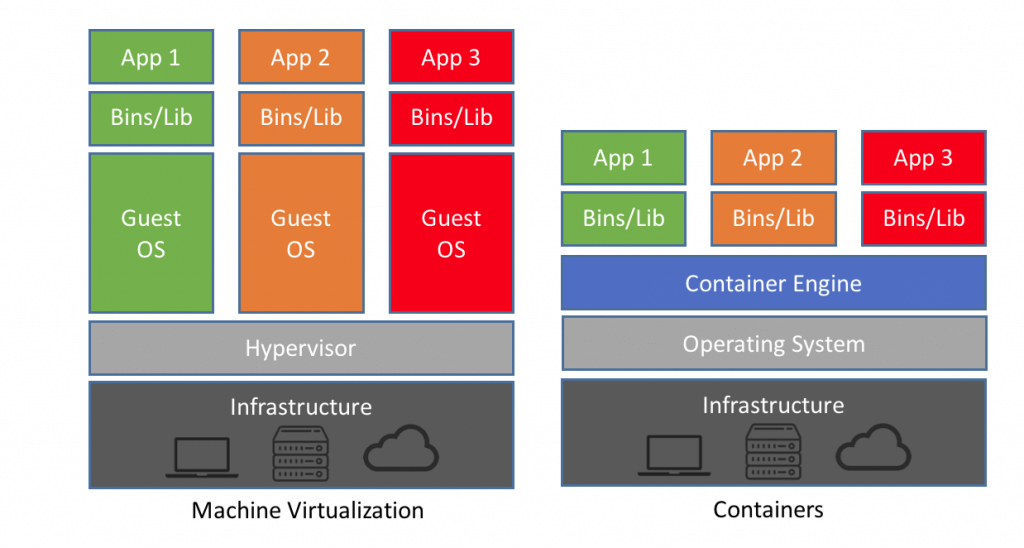
### 17. What is Virtualization?

Virtualization is a method of logically dividing mainframes to allow multiple applications to run simultaneously.

However, this scenario changed when companies and open source communities were able to offer a method of handling privileged instructions. It allows multiple OS to run simultaneously on a single x86 based system.

### 18. What is Hypervisor?

The hypervisor allows you to create a virtual environment in which the guest virtual machines operate. It controls the guest systems and checks if the resources are allocated to the guests as necessary.



Virtualization in Docker vs Hypervisor

### 19. Explain Docker object labels

Docker object labels is a method for applying metadata to docker objects including, images, containers, volumes, network, swam nodes, and services.

### 20. Write a Docker file to create and copy a directory and built it using python modules?

FROM pyhton:2.7-slim

WORKDIR /app

COPY . /app

docker build –tag

### 21. Where the docker volumes are stored?

You need to navigate:

/var/lib/docker/volumes

### 22. List out some important advanced docker commands

| **Command** | **Description** |
| --- | --- |
| docker info | Information Command |
| docker pull | Download an image |
| docker stats | Container information |
| Docker images | List of images downloaded |

### 23. How does communication happen between Docker client and Docker Daemon?

You can communicate between Docker client and Docker Daemon with the combination of Rest API, socket.IO, and TCP.

### 24. Explain Implementation method of Continuous Integration(CI) and Continues Development (CD) in Docker?

You need to do the following things:

* Runs Jenkins on docker
* You can run integration tests in Jenkins using docker-compose

### 25. What are the command to control Docker with Systemd?

systemctl start/stop docker

service docker start/stop

### 26. How to use JSON instead of YAML compose file?

docker-compose -f docker-compose.json up

### 27. What is the command you need to give to push the new image to Docker registry?

docker push myorg/img

### 28. How to include code with copy/add or volumes?

In docker file, we need to use COPY or ADD directive. This is useful to relocate code. However, we should use a volume if we want to make changes.

### 29. Explain the process of scaling your Docker containers

The Docker containers can be scaled to any level starting from a few hundred to even thousands or millions of containers. The only condition for this is that the containers need the memory and the OS at all times, and there should not be a constraint when the Docker is getting scaled.

### 30. What is the method for creating a Docker container?

You can use any of the specific Docker images for creating a Docker container using the below command.

docker run -t -i command name

This command not only creates the container but also start it for you.

### 31. What are the steps for the Docker container life cycle?

Below are the steps for Docker life cycle:

* Build
* Pull
* Run

### 32. How can you run multiple containers using a single service?

By using docker-compose, you can run multiple containers using a single service. All docker-compose files uses yaml language.

### 33. What is CNM?

CNM stands for Container Networking Model. It is a standard or specification from Docker, Inc. that forms the basis of container networking in a Docker environment. This docker’s approach provides container networking with support for multiple network drivers.

### 34. Does Docker offer support for IPV6?

Yes, Docker provides support IPv6. IPv6 networking is supported only on Docker daemons runs on Linux hosts. However, if you want to enable IPv6 support in the Docker daemon, you need to modify /etc/docker/daemon.json and set the ipv6 key to true.

### 35. Can you lose data when the container exits?

No, any data that your application writes to disk get stored in container. The file system for the contain persists even after the container halts.

### 36. What are a different kind of volume mount types available in Docker?

Bind mounts- It can be stored anywhere on the host system

### 37. How to configure the default logging driver under Docker?

To configure the Docker daemon to default to a specific logging driver. You need to set the value of log-driver to the name of the logging drive the daemon.jason.fie.

### 38. Explain Docker Trusted Registry?

Docker Trusted Registry is the enterprise-grade image storage toll for Docker. You should install it after your firewall so that you can securely manage the Docker images you use in your applications.

### 39. What are Docker Namespaces?

The Namespace in Docker is a technique which offers isolated workspaces called the Container. Namespaces also offer a layer of isolation for the Docker containers.

### 40. What are the three components of Docker Architecture

* Client
* Docker-Host
* Registry

### 41. What is client?

Docker provides Command Line Interface tools to the client to interact with Docker daemon.

### 42. What is the purpose of Docker\_Host?

It contains container, images, and Docker daemon. It offers a complete environment to execute and run your application.

### 43. How do I run multiple copies of Compose file on the same host?

Compose uses the project name which allows you to create unique identifiers for all of a project’s containers and other resources. To run multiple copies of a project, set a custom project name using the -a command-line option or using COMPOSE\_PROJECT\_NAME environment variable.

### 1. Docker Vs VM (Virtual Machine)

|  |  |
| --- | --- |
| **Virtual Machines** | **Docker Containers** |
| Need more resources | Less resources are used |
| Process isolation is done at the 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 a Virtual Machine | The creation of docker is very quick |
| Booting takes minutes | Booting is done within seconds. |

### 2. What is Docker?

[Docker](https://mindmajix.com/what-is-docker-how-docker-works) can be defined as a Containerization platform that packs all your applications, and all the necessary dependencies combined to form containers. This will not only ensure the applications work seamlessly given any environment but also provides better efficiency to your Production-ready applications. Docker wraps up bits and pieces of software with all the needed filesystems containing everything that needs to run the code, provide the runtime, system tools/libraries. This will ensure that the software is always run and executed the same, regardless of the environment.

Containers run on the same machine sharing the same Operating system Kernel, this makes it faster – as starting the applications is the only time that is required to start your Docker container (remember that the OS Kernel is already UP and running and uses the least of the RAM possible).

### 3. What is the advantage of Docker over hypervisors?

Docker is lightweight and more efficient in terms of resource uses because it uses the host underlying kernel rather than creating its own hypervisor.

### 4. How is Docker different from other container technologies?

To start with, Docker is one of the upcoming and is a fresh project. Since its inception has been done in the Cloud era, it been way better than many of the other competing Container technologies which have ruled their way until Docker came into existence. There is an active community that is running towards the better upbringing of Docker and it has also started extending its support to Windows and Mac OSX environments in recent days. Other than these, below are the best possible reasons to highlight Docker as one of the better options to choose from than the existing Container technologies.

* 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.

### 5. What is Docker's image?

A Docker image can be understood as a template from which Docker containers can be created as many as we want out of that single Docker image. Having said that, to put it in layman's terms, Docker containers are created out of Docker images. Docker images are created with the build command, and this produces a container that starts when it is run. Docker images are stored in the Docker registry such as the public Docker registry (registry.hub.docker.com) as these are designed to be constituted with layers of other images, enabling just the minimal amount of data over the network.

### 6. What is a Docker container?

This is a very important question so just make sure you don’t deviate from the topic and I will advise you to follow the below mentioned format:

* Docker containers include the application and all of its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.
* Now explain how to create a Docker container, Docker containers can be created by either creating a Docker image and then running it or you can use Docker images that are present on the Dockerhub.
* Docker containers are basically runtime instances of Docker images.

### 7. What is a Docker hub?

Docker hub is a cloud-based registry service that allows you to link to code repositories, build your images and test them, store manually pushed images, and link to the Docker cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

### 8. What is Docker Swarm?

Docker Swarm can be best understood as the native way of Clustering implementation for Docker itself. Docker Swarm turns a pool of Docker hosts into a single and virtual Docker host. It serves the standard Docker API or any other tool that can already communicate with a Docker daemon and can make use of Docker Swarm to scale in a transparent way to multiple hosts. Following is a list of some of the supported tools that will be helpful in achieving what we have discussed just now.

* Dokku
* Docker Compose
* Docker Machine
* Jenkins.

### 9. What is Dockerfile used for?

Dockerfile is nothing but a set of instructions that have to be passed on to Docker itself so that it can build images automatically by reading these instructions from that specified Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

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

YES, you can very comfortably use JSON instead of the default YAML for your Docker compose file. In order to use JSON file with composing, you need to specify the filename to use as the following:  
docker-compose -f docker-compose.json up

### 11. Tell us how you have used Docker in your past position?

This is a question that you could bring upon your whole experience with Docker and if you have used any other Container technologies before Docker. You could also explain the ease that this technology has brought in the automation of the development to production lifecycle management. You can also discuss any other integrations that you might have worked on along with Docker such as Puppet, Chef, or even the most popular of all technologies – Jenkins. If you do not have any experience with Docker itself but similar tools from this space, you could convey the same and also show your interest in learning this leading containerization technology.

### 12. How to create Docker container?

You can create a [Docker Container](https://mindmajix.com/docker/application-platforms-and-continuous-integration) out of any specific Docker image of your choice and the same can be achieved using the command given below:

docker run -t -i command name

The command above will create the container and also starts it for you. In order 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 their statuses on the host that the Docker container runs.  
docker ps -a

### 13. How to stop and restart the Docker container?

The following command can be used to stop a certain Docker container with the container id as

CONTAINER\_ID:  
docker stop CONTAINER\_ID

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

CONTAINER\_ID:  
docker restart CONTAINER\_ID

### 14. How far do Docker containers scale?

Best examples in the Web deployments like Google, Twitter and best examples in the Platform Providers like Heroku, and dotCloud run on Docker which can scale from the ranges of hundreds of thousands to millions of containers running in parallel, given the condition that the OS and the memory don’t run out from the hosts which runs all these innumerable containers hosting your applications.

### 15. 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 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.

### 16. Do I lose my data when the Docker container exits?

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.

### 17. 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.

### 18. 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.

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

A docker-compose stop will attempt to stop a specific Docker container by sending a SIGTERM message. Once this message is delivered, it waits for the default timeout period of 10 seconds and once the timeout period is crossed, it then sends out a SIGKILL message to the container – in order to kill it forcefully. If you are actually waiting for the timeout period, then it means that the containers are not shutting down on receiving SIGTERM signals/messages.

In an attempt to solve this issue, the following is what you can do:

* You can ensure that you are using the JSON form of the CMD and also the ENTRYPOINT in your docker file.
* Use [“program”, “argument1”, “argument2”] instead of sending it as a plain string as like this – “program argument1 argument2”.
* Using the string form, makes Docker run the process using a bash that can’t handle signals properly. Compose always uses the JSON form.
* If it is possible then modify the application which you intend to run by adding an explicit signal handler for the SIGTERM signal
* Also, set the stop\_signal to a proper signal that the application can understand and also know how to handle it.

### 20. 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.

### 21. What’s the difference between up, run, and start?

In any given scenario, you would always want your docker-compose 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 the 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 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.

### 22. What’s the benefit of “Dockerizing?”

Dockerizing enterprise environments helps teams to leverage the Docker containers to form a service platform like CaaS (Container as a Service). It gives teams the necessary agility, and portability and also lets them control staying within their own network/environment.

Most of the developers opt to use Docker and Docker alone because of the flexibility and also the ability that it provides to quickly build and ship applications to the rest of the world. Docker containers are portable and these can run in any environment without making any additional changes when the application developers have to move between Developer, Staging, and Production environments. This whole process is seamlessly implemented without the need of performing any recoding activities for any of the environments. These not only help reduce the time between these lifecycle states but also ensures that the whole process is performed with utmost efficiency. There is every possibility for the Developers to debug any certain issue, fix it and also update the application with it and propagate this fix to the higher environments with the utmost ease.

The operations teams can handle the security of the environments while also allowing the developers to build and ship the applications in an independent manner. The CaaS platform that is provided by the Docker framework can deploy on-premise and is also loaded with full of enterprise-level security features such as role-based access control, integration with LDAP or any Active Directory, image signing and etc. Operations teams have heavily relied on the scalability provided by Docker and can also leverage the Dockerized applications across any environment.

Docker containers are so portable that it allows teams to migrate workloads that run on an Amazon’s [AWS](https://mindmajix.com/aws-tutorial) environment to [Microsoft Azure](https://mindmajix.com/microsoft-azure-tutorial) without even having to change its code and also with no downtime at all. Docker allows teams to migrate these workloads from their cloud environments to their physical datacenters and vice versa. This also enables the Organizations to focus on the infrastructure from the gained advantages both monetarily and also the self-reliability over Docker. The lightweight nature of Docker containers compared to traditional tools like virtualization, combined with the ability for Docker containers to run within VMs, allows teams to optimize their infrastructure by 20X, and save money in the process.

### 23. 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, and available resources (like CPU, and memory) will decide on the number of containers that can run on an environment. Though containers create newer CPUs on their own 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.

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

This can be easily achieved by adding either the COPY or the ADD directives in your docker file. This will count to be useful if you want to move your code along with any of your Docker images, for example, sending your code an environment up the ladder – The development environment to the Staging environment or from the Staging environment to the Production environment.

Having said that, you might come across situations where you’ll need to use both approaches. You can have the image include the code using a COPY, and use a volume in your Compose file to include the code from the host during development. The volume overrides the directory contents of the image.

### 25. Will cloud automation overtake containerization any sooner?

Docker containers are gaining popularity each passing day and definitely will be a quintessential part of any professional 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. In my humble opinion, Docker will be extremely effective in Organizations that appreciate the consequences of Containerization.

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

We can identify the status of a Docker container by running the command ‘docker ps –a’, which will in turn list down all the available docker containers with its corresponding statuses on the host. From there we can easily identify the container of interest to check its status correspondingly.

### 27. What are the differences between the ‘docker run’ and the ‘docker create’?

The most important difference that can be noted is that, by using the ‘docker create’ command we can create a Docker container in the Stopped state. We can also provide it with an ID that can be stored for later usages as well.  
This can be achieved by using the command ‘docker run’ with the option –cidfile FILE\_NAME as like this:  
‘docker run –cidfile FILE\_NAME’

### 28. 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

### 29. Can you remove a paused container from Docker?

To answer this question blatantly, 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.

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

To answer this question blatantly, No, it is not possible. The default –restart flag is set to never restart on its own. If you want to tweak this, then you may give it a try.

### 31. 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](https://docs.docker.com/engine/reference/commandline/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 cleanup tasks. Once this activity is completed, we can then comfortably remove the container using the ‘docker rm’ command from Docker and thereby update the docker registry as well.

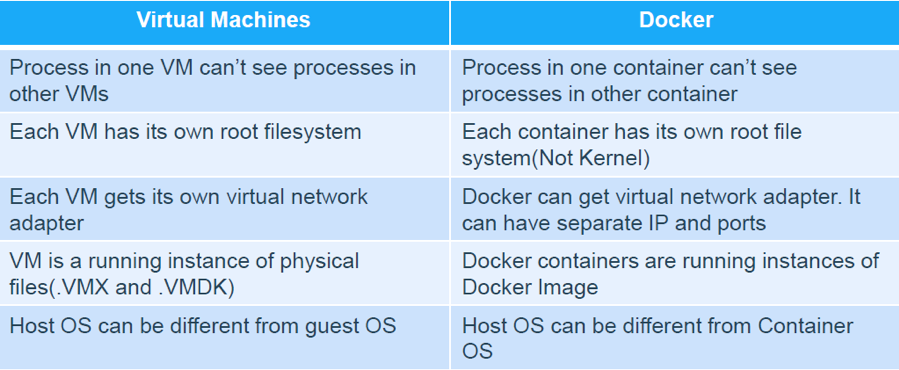
### 32. Difference between Docker Image and container?

Docker container is the runtime instance of the docker image.

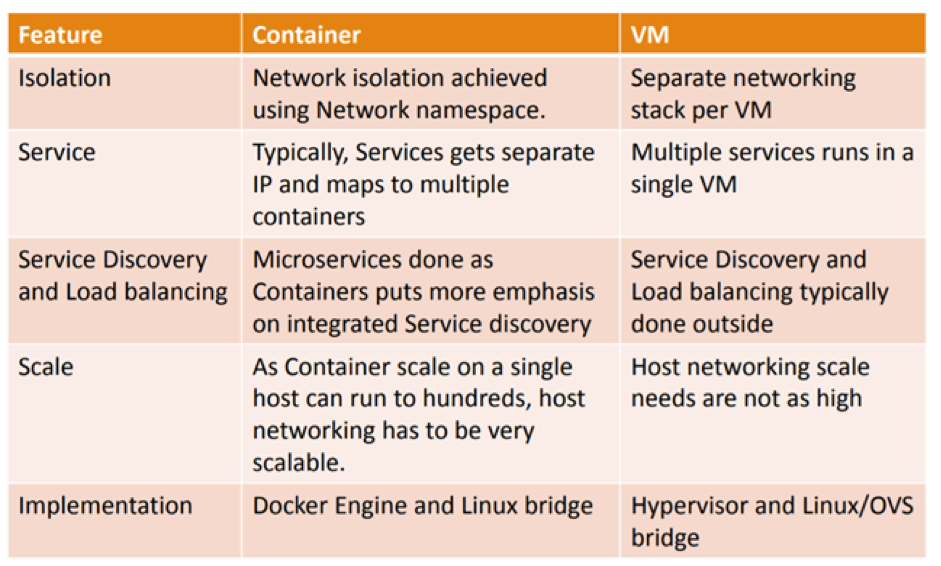
Docker Image doesn't have a state and its state never changes as it is just a set of files whereas the docker container has its execution state.

# What are 5 similarities between Docker & Virtual Machine?

Docker is not quite like a VM. It uses the host kernel & can’t boot a different operating system. Below are 5 similarities between Docker & VIrtual Machine:

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/Picture1.png)

# What is the difference between Container Networking & VM Networking?

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-3.png)

# Is it possible to run multiple process inside Docker container?

Yes, you can run multiple processes inside Docker container. This approach is discouraged for most use cases. For maximum efficiency and isolation, each container should address one specific area of concern. However, if you need to run multiple services within a single container, you can use tools like supervisor. Supervisor is a moderately heavy-weight approach that requires you to package supervisord and its configuration in your image (or base your image on one that includes supervisord), along with the different applications it manages. Then you start supervisord, which manages your processes for you. Example: Here is a Dockerfile using this approach, that assumes the pre-written supervisord.conf, my\_first\_process, and my\_second\_process files all exist in the same directory as your Dockerfile.

# Does Docker run on Linux, macOS and Windows?

You can run both Linux and Windows programs and executables in Docker containers. The Docker platform runs natively on Linux (on x86-64, ARM and many other CPU architectures) and on Windows (x86-64). Docker Inc. builds products that let you build and run containers on Linux, Windows and macOS.

# What is DockerHub?

DockerHub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

# What is Dockerfile?

Docker builds images automatically by reading the instructions from a text file called Dockerfile. It contains all commands, in order, needed to build a given image. A Dockerfile adheres to a specific format and set of instructions which you can find here.

# How is Dockerfile different from Docker Compose?

A Dockerfile is a simple text file that contains the commands a user could call to assemble an image whereas Docker Compose is a tool for defining and running multi-container Docker applications. Docker Compose define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment. It get an app running in one command by just running docker-compose up. Docker compose uses the Dockerfile if one add the build command to your project’s docker-compose.yml. Your Docker workflow should be to build a suitable Dockerfile for each image you wish to create, then use compose to assemble the images using the build command.

# Can I use JSON instead of YAML for my Docker Compose file?

Yes. Yaml is a superset of json so any JSON file should be valid Yaml. To use a JSON file with Compose, specify the filename to use, for example: docker-compose -f docker-compose.json up

You can use json instead of yaml for your compose file, to use json file with compose, specify the filename to use for eg: docker-compose -f docker-compose.json up

# How to create Docker container?

We can use Docker image to create Docker container by using the below command:

$ docker run -t -i command name

This command will create and start a container.If you want to verify the list of all running container with the status on a host use the below command:

$ docker ps -a

# What is maximum number of container you can run per host?

This really depends on your environment. The size of your applications as well as the amount of available resources (i.e like CPU) will all affect the number of containers that can be run in your environment. Containers unfortunately are not magical. They can’t create new CPU from scratch. They do, however, provide a more efficient way of utilizing your resources. The containers themselves are super lightweight (remember, shared OS vs individual OS per container) and only last as long as the process they are running.

# Is it possible to have my own private Docker registry?

Yes, it is possible today using Docker own registry server. if you want to use 3rd party tool, see Portus. TBA

# Does Docker container package up the entire OS?

Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.

# Describe how many ways are available to configure Docker daemon?

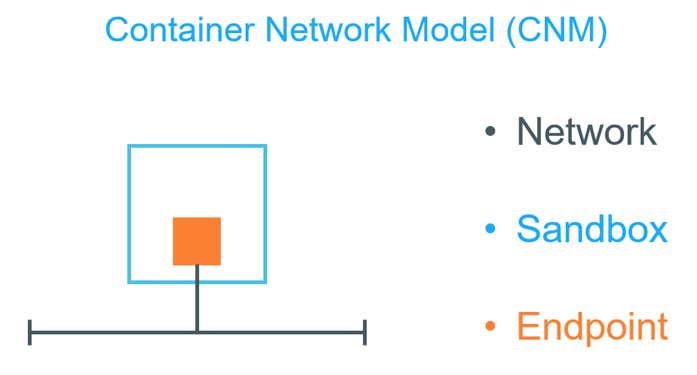
There are two ways to configure the Docker daemon:

* Using a JSON configuration file. This is the preferred option, since it keeps all configurations in a single place.
* Using flags when starting dockerd. You can use both of these options together as long as you don’t specify the same option both as a flag and in the JSON file. If that happens, the Docker daemon won’t start and prints an error message. $ dockerd –debug –tls=true –tlscert=/var/docker/server.pem –tlskey=/var/docker/serverkey.pem  
  –host tcp://<Host\_IP>:2376

1. Can you list reasons why Container Networking is so important? Below are top 5 reasons why we need container networking:

* Containers need to talk to external world.
* Reach Containers from external world to use the service that Containers provides.
* Allows Containers to talk to host machine.
* Inter-container connectivity in same host and across hosts.
* Discover services provided by containers automatically.
* Load balance traffic between different containers in a service.
* Provide secure multi-tenant services.

# What does CNM refers to? What are its components? ![img](

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-5.png)

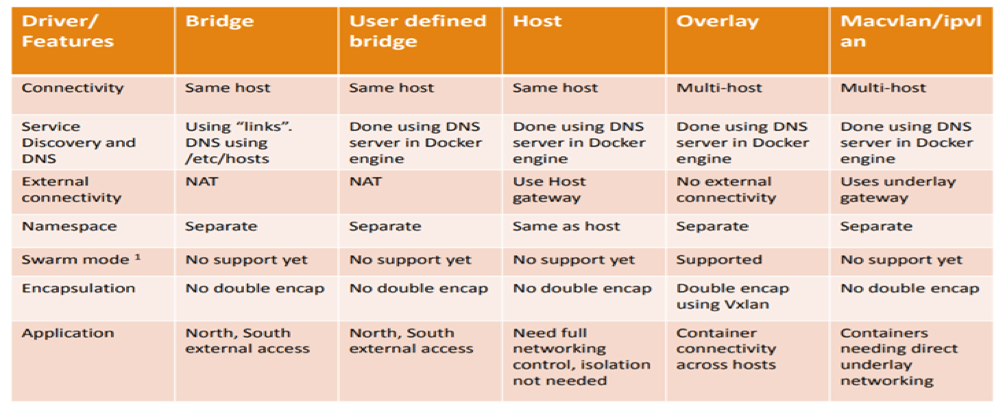
CNM refers to Container Networking Model. The Container Network Model (CNM) is a standard or specification from Docker, Inc. that forms the basis of container networking in a Docker environment.It is Docker’s approach to providing container networking with support for multiple network drivers. The CNM provides the following contract between networks and containers:

* All containers on the same network can communicate freely with each other
* Multiple networks are the way to segment traffic between containers and should be supported by all drivers
* Multiple endpoints per container are the way to join a container to multiple networks
* An endpoint is added to a network sandbox to provide it with network connectivity

The major components of the CNM are:

* Network,
* Sandbox and
* Endpoint. Sandbox is a generic term that refers to OS specific technologies used to isolate networks stacks on a Docker host. Docker on Linux uses kernel namespaces to provide this sandbox functionality. Networks “stacks” inside of sandboxes include interfaces, routing tables, DNS etc. A network in CNM terms is one or more endpoints that can communicate.All endpoints on the same network can communicate with each other.Endpoints on different networks cannot communicate without external routing.

# What are different types of Docker Networking drivers?

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-7.png)

Docker’s networking subsystem is pluggable using drivers. Several drivers exist by default, and provide core networking functionality. Below is the snapshot of difference of various Docker networking drivers.

Below are details of Docker networking drivers: Bridge: The default network driver. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.

Host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly. host is only available for swarm services on Docker 17.06 and higher.

Overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See overlay networks.

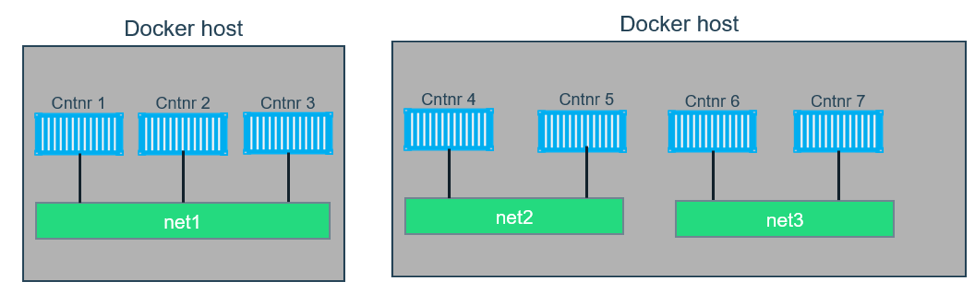
MacVLAN: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host’s network stack.

None: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services.

# What features are possible only under Docker Enterprise Edition in comparison to Docker Community Edition?

The following two features are only possible when using Docker EE and managing your Docker services using Universal Control Plane (UCP): The HTTP routing mesh allows you to share the same network IP address and port among multiple services. UCP routes the traffic to the appropriate service using the combination of hostname and port, as requested from the client. Session stickiness allows you to specify information in the HTTP header which UCP uses to route subsequent requests to the same service task, for applications which require stateful sessions.

# How is Docker Bridge network different from traditional Linux bridge ?

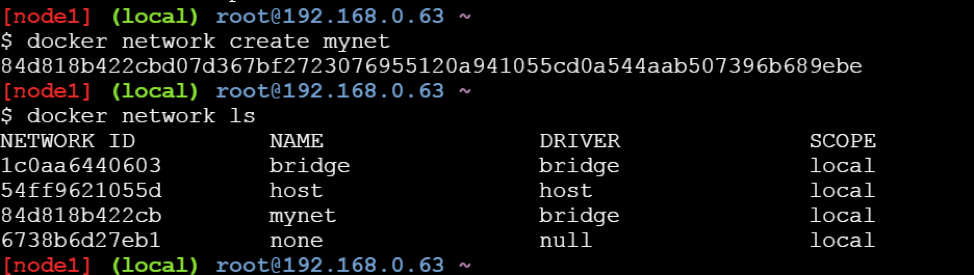
[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-8.png)

In terms of networking, a bridge network is a Link Layer device which forwards traffic between network segments. A bridge can be a hardware device or a software device running within a host machine’s kernel. In terms of Docker, a bridge network uses a software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network. The Docker bridge driver automatically installs rules in the host machine so that containers on different bridge networks cannot communicate directly with each other.

# How to create a user-defined Bridge network ?

To create a user-defined bridge network, one can use the docker network create command –

$ docker network create mynet

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-9.png)

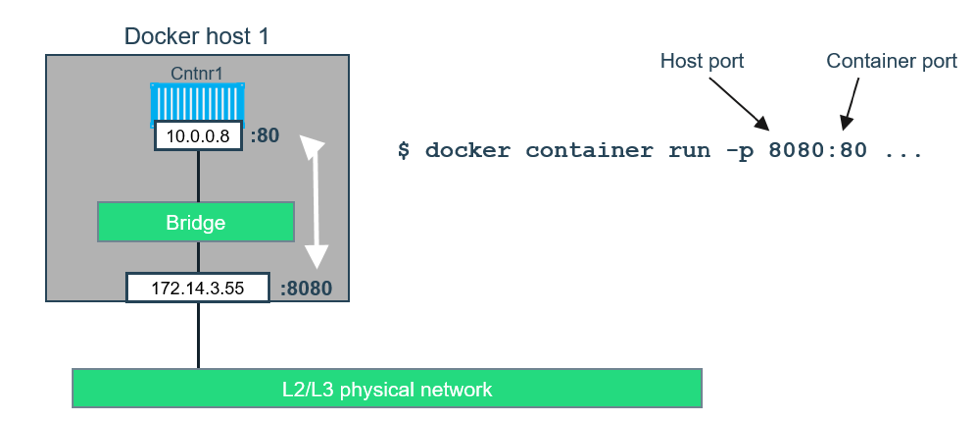
You can specify the subnet, the IP address range, the gateway, and other options. See the docker network create reference or the output of docker network create –help for details.

# How to delete a user-defined Bridge network ?

Use the docker network rm command to remove a user-defined bridge network. If containers are currently connected to the network, disconnect them first.

$ docker network rm mynet

# How to connect Docker container to user-defined bridge network?

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-11.png)

When you create a new container, you can specify one or more –network flags. This example connects a Nginx container to the my-net network. It also publishes port 80 in the container to port 8080 on the Docker host, so external clients can access that port. Any other container connected to the my-net network has access to all ports on the my-nginx container, and vice versa.

$ docker create --name my-nginx \

--network my-net \

--publish 8080:80 \

nginx:latest

To connect a running container to an existing user-defined bridge, use the docker network connect command. The following command connects an already-running my-nginx container to an already-existing my-net network:

$ docker network connect my-net my-nginx

# Does Docker support IPv6?

Yes, Docker support IPv6. IPv6 networking is only supported on Docker daemons running on Linux hosts.Support for IPv6 address has been there since Docker Engine 1.5 release.To enable IPv6 support in the Docker daemon, you need to edit /etc/docker/daemon.json and set the ipv6 key to true.

{

"ipv6": true

}

Ensure that you reload the Docker configuration file.

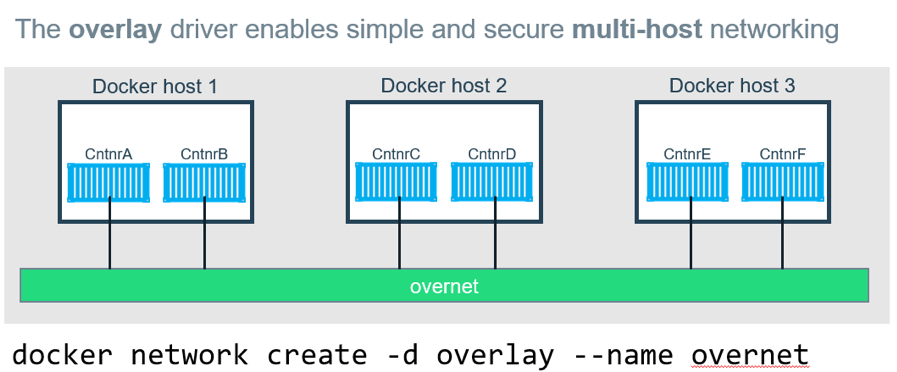
$ systemctl reload docker

You can now create networks with the--ipv6flag and assign containers IPv6 addresses using the --ip6 flag.

# Does Docker Compose file format support IPv6 protocol?

Yes.

# How is overlay network different from bridge network?

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-13.png)

Bridge networks connect two networks while creating a single aggregate network from multiple communication networks or network segments, hence the name bridge. Overlay networks are usually used to create a virtual network between two separate hosts. Virtual, since the network is build over an existing network. Bridge networks can cater to single host, while overlay networks are for multiple hosts.

1. What networks are affected when you join a Docker host to an existing Swarm? When you initialize a swarm or join a Docker host to an existing swarm, two new networks are created on that Docker host:

* an overlay network called ingress, which handles control and data traffic related to swarm services. When you create a swarm service and do not connect it to a user-defined overlay network, it connects to the ingress network by default.
* a bridge network called docker\_gwbridge, which connects the individual Docker daemon to the other daemons participating in the swarm.

# How shall you disable the networking stack on a container?

 If you want to completely disable the networking stack on a container, you can use the –network none flag when starting the container. Within the container, only the loopback device is created. The following example illustrates this.

# How can one create MacVLAN network for Docker container?

To create a Macvlan network which bridges with a given physical network interface, once can use –driver macvlan with the docker network create command. You also need to specify the parent, which is the interface the traffic will physically go through on the Docker host.

$ docker network create -d macvlan \

--subnet=172.16.86.0/24 \

--gateway=172.16.86.1 \

-o parent=eth0 collabnet

# Is it possible to exclude IP address from being used in MacVLAN network?

If you need to exclude IP addresses from being used in the Macvlan network, such as when a given IP address is already in use, use --aux-addresses:

$ docker network create -d macvlan \

--subnet=192.168.32.0/24 \

--ip-range=192.168.32.128/25 \

--gateway=192.168.32.254 \

--aux-address="my-router=192.168.32.129" \

-o parent=eth0 collabnet32

# Do I lose my data when the container exits?

Not at all! Any data that your application writes to disk gets preserved in its container until you explicitly delete the container. The file system for the container persists even after the container halts.

# Does Docker Enterprise Edition support Kubernetes?

Yes, Docker Enterprise Edition(rightly called EE) support Kubernetes. EE 2.0 allows users to choose either Kubernetes or Swarm at the orchestration layer.

# What is Docker Swarm?

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

# What is --memory-swap flag?

--memory-swap is a modifier flag that only has meaning if --memory is also set. Using swap allows the container to write excess memory requirements to disk when the container has exhausted all the RAM that is available to it. There is a performance penalty for applications that swap memory to disk often.

# Can you explain different volume mount types available in Docker?

There are three mount types available in Docker · Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker. · Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time. · tmpfs mounts are stored in the host system’s memory only, and are never written to the host system’s filesystem.

# How to share data among DockerHost?

Ways to achieve this when developing your applications. One is to add logic to your application to store files on a cloud object storage system like Amazon S3. Another is to create volumes with a driver that supports writing files to an external storage system like NFS or Amazon S3. Volume drivers allow you to abstract the underlying storage system from the application logic. For example, if your services use a volume with an NFS driver, you can update the services to use a different driver, as an example to store data in the cloud, without changing the application logic.

# How to Backup, Restore, or Migrate data volumes under Docker container?

Steps to Backup a container

1. Launch a new container and mount the volume from the dbstore container
2. Mount a local host directory as /backup
3. Pass a command that tars the contents of the dbdata volume to a backup.tar file inside our /backup directory.

$ docker run --rm --volumes-from dbstore -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata Restore container from backup With the backup just created, you can restore it to the same container, or another that you made elsewhere. For example, create a new container named dbstore2: $ docker run -v /dbdata --name dbstore2 ubuntu /bin/bash

Then un-tar the backup file in the new container`s data volume:

$ docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1

# How to Configure Automated Builds on DockerHub

You can build your images automatically from a build context stored in a repository. A build context is a Dockerfile and any files at a specific location. For an automated build, the build context is a repository containing a Dockerfile.

# How to configure the default logging driver under Docker?

To configure the Docker daemon to default to a specific logging driver, set the value of log-driver to the name of the logging driver in the daemon.json file, which is located in /etc/docker/ on Linux hosts or C:\ProgramData\docker\config\ on Windows server hosts. The default logging driver is json-file.

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

Compose stop attempts to stop a container by sending a SIGTERM. It then waits for a default timeout of 10 seconds. After the timeout, a SIGKILL is sent to the container to forcefully kill it. If you are waiting for this timeout, it means that your containers aren’t shutting down when they receive the SIGTERM signal.

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

Compose uses the project name to create unique identifiers for all of a project’s containers and other resources. To run multiple copies of a project, set a custom project name using the -command line option or the COMPOSE\_PROJECT\_NAME environment variable.

# What’s the difference between up, run, and start under Docker Compose?

Typically, you want docker-compose up. Use up to start or restart all the services defined in a docker-compose.yml. In the default “attached” mode, you see all the logs from all the containers. In “detached” mode (-d), Compose exits after starting the containers, but the containers continue to run in the background.

The docker-compose run command is for running “one-off” or “adhoc” tasks. It requires the service name you want to run and only starts containers for services that the running service depends on. Use run to run tests or perform an administrative task such as removing or adding data to a data volume container. The run command acts like docker run -ti in that it opens an interactive terminal to the container and returns an exit status matching the exit status of the process in the container. The docker-compose start command is useful only to restart containers that were previously created, but were stopped. It never creates new containers.

# What is Docker Trusted Registry?

Docker Trusted Registry (DTR) is the enterprise-grade image storage solution from Docker. You install it behind your firewall so that you can securely store and manage the Docker images you use in your applications.

# How to declare default environment variables under Docker Compose?

Compose supports declaring default environment variables in an environment file named .env placed in the folder where the docker-compose command is executed (current working directory). Example: The below example demonstrate how to declare default environmental variable for Docker Compose.

When you run docker-compose up, the web service defined above uses the image alpine:v3.4. You can verify this with the docker-compose config command which prints your resolved application config to the terminal:

# Can you list out ways to share Compose configurations between files and projects under Docker Compose?

Compose supports two methods of sharing common configuration:

1. Extending an entire Compose file by using multiple Compose files
2. Extending individual services with the extends field

# What is the role of .dockerignore file?

To understand the role of .dockerignore file, let us take a practical example. You may have noticed that if you put a Dockerfile in your home directory and launch a docker build you will see a message uploading context. Right? This means docker creates a .tar with all the files in your home and in all the subdirectories, and uploads this tar to the docker daemon. If you have some huge files, this may take a long time. In order to avoid this, you might need to create a specific directory, where you put your Dockerfile, and all what is needed for your build. It becomes necessary to tell docker to ignore some files during the build. Hence, you need to put in the .dockerignore all the files not needed for your build Before the docker CLI sends the context to the docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.

# What is the purpose of EXPOSE command in Dockerfile?

When writing your Dockerfiles, the instruction EXPOSE tells Docker the running container listens on specific network ports. This acts as a kind of port mapping documentation that can then be used when publishing the ports.

EXPOSE <port> [<port>/<protocol>...]

You can also specify this within a docker run command, such as:

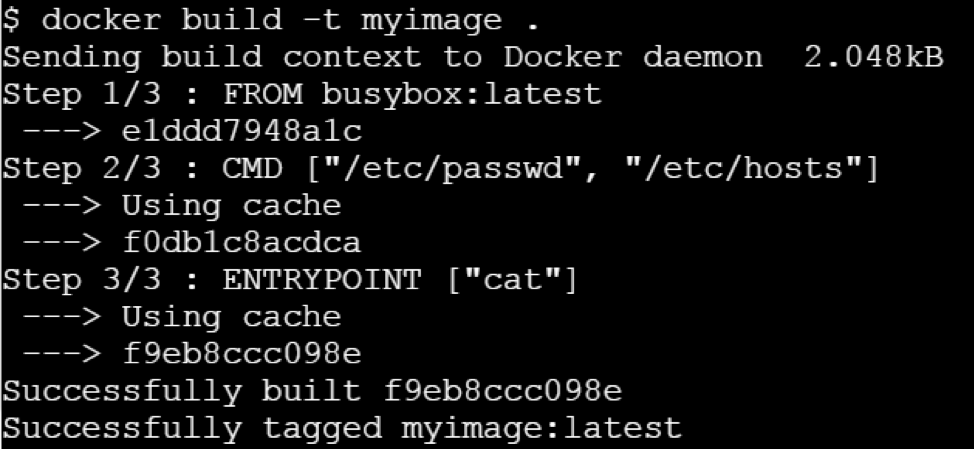
docker run --expose=1234 my\_app

Please note that EXPOSE will not allow communication via the defined ports to containers outside of the same network or to the host machine. To allow this to happen you need to publish the ports.

# How is ENTRYPOINT instruction under Dockerfile different from RUN instruction?

ENTRYPOINT is meant to provide the executable while CMD is to pass the default arguments to the executable. To understand it clearly, let us consider the below Dockerfile:

If you try building this Docker image using docker build command –

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-21.png)

Let us run this image without any argument.

Let’s run it passing a command line argument

This clearly state that ENTRYPOINT is meant to provide the executable while CMD is to pass the default arguments to the executable.

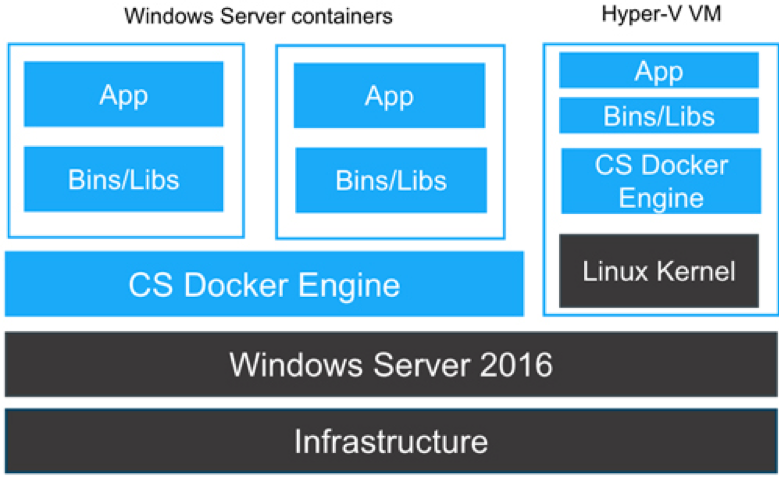
# Why Build cache in Docker is so important?

If the objects on the file system that Docker is about to produce are unchanged between builds, reusing a cache of a previous build on the host is a great time-saver. It makes building a new container really, really fast. None of those file structures have to be created and written to disk this time — the reference to them is sufficient to locate and reuse the previously built structures.

# Why Docker Monitoring is necessary?

● Monitoring helps to identify issues proactively that would help to avoid system outages. ● The monitoring time-series data provide insights to fine-tune applications for better performance and robustness. ● With full monitoring in place, changes could be rolled out safely as issues will be caught early on and be resolved quickly before that transforms into root-cause for an outage. ● The changes are inherent in container based environments and impact of that too gets monitored indirectly.

# Difference between Windows Containers and Hyper-V Containers

[](https://raw.githubusercontent.com/collabnix/dockerlabs/master/docker/img/docker-interview-24.png)

Underlying is the architecture laid out by the Microsoft for the Windows and Hyper-V Containers

Here are few of the differences bet

### ****1. What is Docker?****

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Docker** | **Virtual Machines** |
| Use of OS | All containers share the host OS | Each VM runs on its own OS |
| Startup time | Very fast | Slow |
| Isolation | Process-level isolation | Full isolation |
| Security | Low | High |

We can define Docker as a containerization platform that combines all our applications in a package so that we have all the dependencies to run our applications in any environment. This means, our application will run seamlessly on any environment, and this makes it easy for having a product-ready application. What Docker does is wrap the software needed in a file system that has everything for running the code, providing the runtime and all the necessary libraries and system tools. Containerization technology like Docker will share the same operating system kernel with the machine, and due to this it is extremely fast. This means that we have to run Docker only at the beginning and after that, since our OS is already running, we will have a smooth and seamless process.

### ****2. Define Containerization.****

Containerization is a form of virtualization through which applications are run in containers (isolated user spaces) all using a shared OS. It packs or encapsulates software code and all its dependencies for it to run in a consistent and uniform manner on any infrastructure.

### ****3. What is the benefit of using a Docker over a Hypervisor?****

Though Docker and Hypervisor might do the same job overall, there are many differences between them in terms of how they work. Docker can be thought of as lightweight since it uses very few resources and also the host kernel rather than creating it like a Hypervisor.

### ****4. What are the unique features of Docker over other containerization technologies?****

Some of the most important and unique features of Docker are as follows:

* We can run our Docker container either on our PC or on our enterprise IT system.
* Along with the Docker Hub, which is a repository of all containers, we can deploy and download all our applications from a central location.
* We can even share our applications with the containers that we create.

### ****5. What is a Docker image?****

A Docker image helps in creating Docker containers. We can create the Docker image with the build command; due to this, it creates a container that starts when it begins to run. All the Docker images are stored in the Docker registry such as the public Docker registry. These have minimal amounts of layers within the image so that there is a minimum amount of data on the network.

Top of Form



**6. What is a Docker container?**

It is a comprehensive set of applications including all its dependencies which share the same OS kernel, along with the other containers running in separate processes within the operating system in a user space. Docker is not tied to any IT infrastructure, and thus it can run on any computer system or on the cloud. We can create a Docker container using Docker images and then run it, or we can use the images that are already created in the Docker Hub. To simplify things, let’s say that the Docker containers are just runtime instances of the Docker image.

### ****7. What is a Docker Hub?****

We can think of Docker Hub as a cloud registry that lets us link the code repositories, create the images, and test them. We can also store our pushed images, or we can link to the Docker Cloud, so that the images can be deployed to the host. We have a centralized container image discovery resource that can be used for the collaboration of our teams, automating the workflow and distribution, and changing management by creating the development pipeline.

### ****8. What is a Docker Swarm?****

We can think of a Docker Swarm as the way of orchestrating the Docker containers. We will be able to implement Dockers in a cluster. We can convert our Docker pools into a single Docker Swarm for easy management and monitoring.

### ****9. What is the use of a Dockerfile?****

A Dockerfile is a set of specific instructions that we need to pass on to Docker so that the images can be built. We can think of the Dockerfile as a text document which has all the commands that are needed for creating a Docker image. We can create an automated build that lets us execute multiple command lines one after the other.

### ****10. What is Docker Compose?****

Docker Compose defines and runs multi-container Docker applications. With Compose, a YAML file is used to configure an application’s services. All the services from the configuration can be created and started with a single command.

### ****11. Why use Docker?****

Following are the reasons why one should use Docker:

* It allows the use of system resources more efficiently
* Software delivery cycles are faster with it
* Application portability is possible and easy
* It is great for the microservices architecture

### ****12. What are the drawbacks of Docker?****

Docker has a few drawbacks as listed below:

* No storage option
* Poor monitoring
* Unable to automatically reschedule inactive nodes
* Has a complicated automatic horizontal scaling setup

### ****13. What is Docker Engine?****

Docker Engine is an open-source containerization technology that facilitates the development, assembling, shipping, and running of applications with the help of the following components:

* Docker Daemon
* Docker Engine REST API
* Docker CLI

### ****14. What are registries?****

Docker registries provide locations for storing and downloading images. There are two types of registries

* Public registry
* Private registry

Public registries include Docker Hub and Docker Cloud.

### ****15. What are Docker namespaces?****

When a container is run, Docker creates a set of isolated workspaces for the container called namespaces.

### ****16. How to use Docker?****

The Docker command syntax looks like this:

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

To run a container, we must incorporate the image on which it is based:

docker run [docker\_image]

We can run containers if the Docker images are locally stored. If they are not, the software will take it from the online registry.

Docker can be used to run a container:

* Under a specific name
* Interactively
* And publish container ports
* In the background in the detached mode
* And mount host volumes
* And remove it once the process is complete

### ****17. Is it possible to use JSON instead of YAML for Docker Compose?****

We can use JSON instead of YAML for a Docker Compose file. When we are using the JSON file for composing, we have to specify the filename with the following command:

docker-compose -f docker-compose.json up

### ****18. What is the process for creating a Docker container?****

We can use any specific Docker image for creating a Docker container using the below command:

docker run -t -i command name

This command not only creates the container but also starts it for us. If we want to check whether the Docker container has been created or not, then we need to have the following command that will list all the Docker containers, along with the host on which the Docker containers run:

docker ps -a

### ****19. What is the process for stopping and restarting a Docker container?****

To stop a Docker container, we need to use the following command:

docker stop CONTAINER\_ID

To restart a Docker container, we need to use the following command:

docker restart CONTAINER\_ID

### ****20. How to create a Docker image?****

A Docker image can be built using the following command:

$ docker build .

**21. How to stop a Docker container?**

We can use the following to stop one or more running Docker containers:

docker container stop [OPTIONS] CONTAINER [CONTAINER...]

### ****22. How to remove Docker images?****

Use **docker images** with the **-a** flag to get the image IDs for removal. Then, pass their IDs or tags to

docker rmi:

**List:**

docker images -a

**Remove:**

docker rmi Image Image

### ****23. Where are Docker images stored?****

It depends on which system the Docker is running and the Docker storage driver is being used.

For example, on Windows, Docker images are stored by default in:

C:\ProgramData\DockerDesktop

On a Mac, Docker images are stored by default in:

~/Library/Containers/com.docker.docker/Data/vms/0/

### ****24. How to run a Docker image?****

The ‘docker run’ command runs an image inside the container. It will require the image name.

docker run [docker\_image]

### ****25. How to start a Docker container?****

The following command starts a Docker container:

docker container start [OPTIONS] CONTAINER [CONTAINER...]

### ****26. How to use Docker Compose?****

Docker Compose typically includes a three-step process:

1. Using a **dockerfile** to define the app’s environment to facilitate reproduction anywhere
2. Defining the app services in**docker-compose.yml** so that they can run in an isolated environment together
3. Running**docker-compose up**

### ****27. What command should be run to view all the running Docker containers?****

To view all the running containers in Docker, we can use the following:

$ docker ps

### ****28. What is Docker daemon?****

Docker daemon is a service that manages Docker containers, images, storage volumes, and the network. It constantly listens to Docker API requests and processes them. A daemon can communicate with other daemons as well for the management of Docker services.

### ****29. Name and explain the states of a Docker container.****

* **Created:** We see this Docker container state when a container is newly created.
* **Restarting:** When the Docker container is restarted due to any issues, this state is observed.
* **Running:** It is the main state for the container after it has started.
* **Paused:** When a running Docker container is temporarily stopped via docker pause, this is the status that we will see.
* **Exited:** If a container has stopped due to some issue or stopped manually, this will be the state of the container.
* **Dead:**When the daemon has tried but failed to stop a container (mostly because of a busy device or resource), this state will be seen.

### ****30. What is Docker Hub?****

Docker Hub helps with linking to code repositories. This cloud-based registry enables the building, testing, and storing of images in Docker Cloud. Images can also be deployed to the host with it.

### ****31. Define Virtualization.****

Virtualization is the process of logically dividing mainframes to enable more than one application to run at a time.

### ****32. What is a Hypervisor?****

A hypervisor helps in the creation of a virtual environment, in which the guest virtual machines run. It manages the guest systems and checks the required resource allocations to the guests.

### ****33. What are Docker object labels?****

Docker object labels help us add metadata to Docker objects, including containers, images, Swarm nodes, network, volumes, and services.

### ****34. What are the steps in a Docker container life cycle?****

* Build
* Pull
* Run

### ****35. Name the three components of the Docker architecture.****

* Client
* Docker host
* Registry

### ****36. How do you scale your Docker containers?****

Docker containers can be scaled to any level, starting from a few hundreds to even thousands or millions of containers. The only condition is that the containers need the memory and the OS all the time, and there should not be a constraint on these when the Docker is getting scaled.

### ****37. How does communication happen between Docker client and Docker daemon?****

The communication between Docker client and Docker daemon happens with the help of the combination of TCP, Rest API, and Socket.IO.

### ****38. Explain the implementation method of continuous integration (CI) and continuous deployment (CD) in Docker.****

* Run [**Jenkins**](https://intellipaat.com/blog/tutorial/devops-tutorial/jenkins-tutorial/) on Docker
* Using**docker-compose**, run integration tests in Jenkins

### ****39. How to run a Docker container?****

The Docker run command manages the running of containers in Docker.

**Running a container under a specific name:**

The command for running a container under a specific name is:

docker container run --name [container\_name] [docker\_image]

**Running a container in the background in the detached mode:**

The command for running a container in the background is:

docker container run -d [docker\_image]

**Running a container interactively:**

The following command is run for running a container interactively:

docker container run -it [docker\_image] /bin/bash

**Running a container and publishing container ports:**

We have to include **-p** to the **docker run** command, along with the following:

-p [host\_ip]:[host\_port]:[container\_port]

Here,**host\_ip**is optional. It is not mandatory to specify this while we run the command.

**Running a container and mounting host volumes:**

The **docker container run** command looks like this:

docker container run -v [/host/volume/location]:[/container/storage] [docker\_image]

### ****40. Tell us how you have used Docker in your past position.****

This is a question wherein we could bring upon our whole experience with Docker and any other Container technologies we have used prior to Docker. We could also explain the ease that this technology has brought in the automation of the development-to-production life cycle management. We can also discuss any other integrations that we might have worked, along with Docker, such as Puppet, Chef, or even the most popular of all technologies—Jenkins. If we do not have any experience with Docker but we have it with similar tools from this space, we could convey the same and also show our interest in learning this leading containerization technology.

## *Q1*:

## What is Docker

**Answer**

* Docker is a containerization platform which packages your application and all its dependencies together in the form of containers so as to ensure that your application works seamlessly in any environment be it development or test or production.
* Docker containers, wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries etc. anything that can be installed on a server.
* This guarantees that the software will always run the same, regardless of its environment.

## *Q2*:

## What is the need for DevOps?

Nowadays instead of releasing big sets of features, companies are trying to see if small features can be transported to their customers through a series of release trains. This has many advantages like quick feedback from customers, better quality of software etc. which in turn leads to high customer satisfaction. To achieve this, companies are required to:

1. Increase deployment frequency
2. Lower failure rate of new releases
3. Shortened lead time between fixes
4. Faster mean time to recovery in the event of new release crashing

DevOps fulfills all these requirements and helps in achieving seamless software delivery.

## How to build envrionment-agnostic systems with Docker?

There are three main features helping to achieve that:

* Volumes
* Environment variable injection
* Read-only file systems

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

We can identify the status of a Docker container by running the command

docker ps –a

which will in turn list down all the available docker containers with its corresponding statuses on the host. From there we can easily identify the container of interest to check its status correspondingly.

## What are the advantages of DevOps?

Technical benefits:

* Continuous software delivery
* Less complex problems to fix
* Faster resolution of problems

Business benefits:

* Faster delivery of features
* More stable operating environments
* More time available to add value (rather than fix/maintain)

## 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.

## 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

## What is Docker container?

**Docker containers** include the application and all of its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

## What is Docker hub?

**Docker hub** is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

## What is Docker image?

**Docker image** is the source of Docker container. In other words, Docker images are used to create containers. Images are created with the build command, and they’ll produce a container when started with run. Images are stored in a Docker registry such as registry.hub.docker.com because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network.

## What is the difference between the COPY and ADD commands in a Dockerfile?

Although ADD and COPY are functionally similar, generally speaking, COPY is preferred.

That’s because it’s more transparent than ADD. COPY only supports the basic copying of local files into the container, while ADD has some features (like local-only tar extraction and remote URL support) that are not immediately obvious. Consequently, the best use for ADD is local tar file auto-extraction into the image, as in ADD rootfs.tar.xz /.

## What is the function of CI (Continuous Integration) server?

CI server function is to continuously integrate all changes being made and committed to repository by different developers and check for compile errors. It needs to build code several times a day, preferably after every commit so it can detect which commit made the breakage if the breakage happens.

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

It is preferable to create Stateless application for Docker Container. 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. Also a stateless application is much easier to scale with Docker Containers than a stateful application.

## Explain basic Docker usage workflow

1. Everything starts with the **Dockerfile**. The Dockerfile is the source code of the Image.
2. Once the Dockerfile is created, you build it to create the **image** of the container. The image is just the "compiled version" of the "source code" which is the Dockerfile.
3. Once you have the image of the container, you should redistribute it using the **registry**. The registry is like a git repository -- you can push and pull images.
4. Next, you can use the image to run **containers**. A running container is very similar, in many aspects, to a virtual machine (but without the hypervisor).

+------------+ docker build +--------------+ docker run -dt +-----------+ docker exec -it +------+

| Dockerfile | --------------> | Image | ---------------> | Container | -----------------> | Bash |

+------------+ +--------------+ +-----------+ +------+

^

| docker pull

|

+--------------+

| Registry |

+--------------+

## 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.

## What is Docker Swarm?

**Docker Swarm** is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

## What is Hypervisor?

The **hypervisor** handles creating the virtual environment on which the guest virtual machines operate. It supervises the guest systems and makes sure that resources are allocated to the guests as necessary. The hypervisor sits in between the physical machine and virtual machines and provides virtualization services to the virtual machines. To realize it, it intercepts the guest operating system operations on the virtual machines and emulates the operation on the host machine's operating system.

The rapid development of virtualization technologies, primarily in cloud, has driven the use of virtualization further by allowing multiple virtual servers to be created on a single physical server with the help of hypervisors, such as Xen, VMware Player, KVM, etc., and incorporation of hardware support in commodity processors, such as Intel VT and AMD-V.

## What is the difference between Docker Image and Layer?

* **Image**: A Docker image is built up from a series of **read-only** layers
* **Layer**: Each layer represents an instruction in the image’s Dockerfile.

The below Dockerfile contains four commands, each of which creates a layer.

FROM ubuntu:15.04

COPY . /app

RUN make /app

CMD python /app/app.py

Importantly, each layer is only a set of differences from the layer before it.

## What is virtualisation?

In its conceived form, **virtualisation** was considered a method of logically dividing mainframes to allow multiple applications to run simultaneously. However, the scenario drastically changed when companies and open source communities were able to provide a method of handling the privileged instructions in one way or another and allow for multiple operating systems to be run simultaneously on a single x86 based system.

The net effect is that virtualization allows you to run two completely different OS on same hardware. Each guest OS goes through all the process of bootstrapping, loading kernel etc. You can have very tight security, for example, guest OS can't get full access to host OS or other guests and mess things up.

The virtualization method can be categorized based on how it mimics hardware to a guest operating system and emulates guest operating environment. Primarily, there are three types of virtualization:

* Emulation
* Paravirtualization
* Container-based virtualization