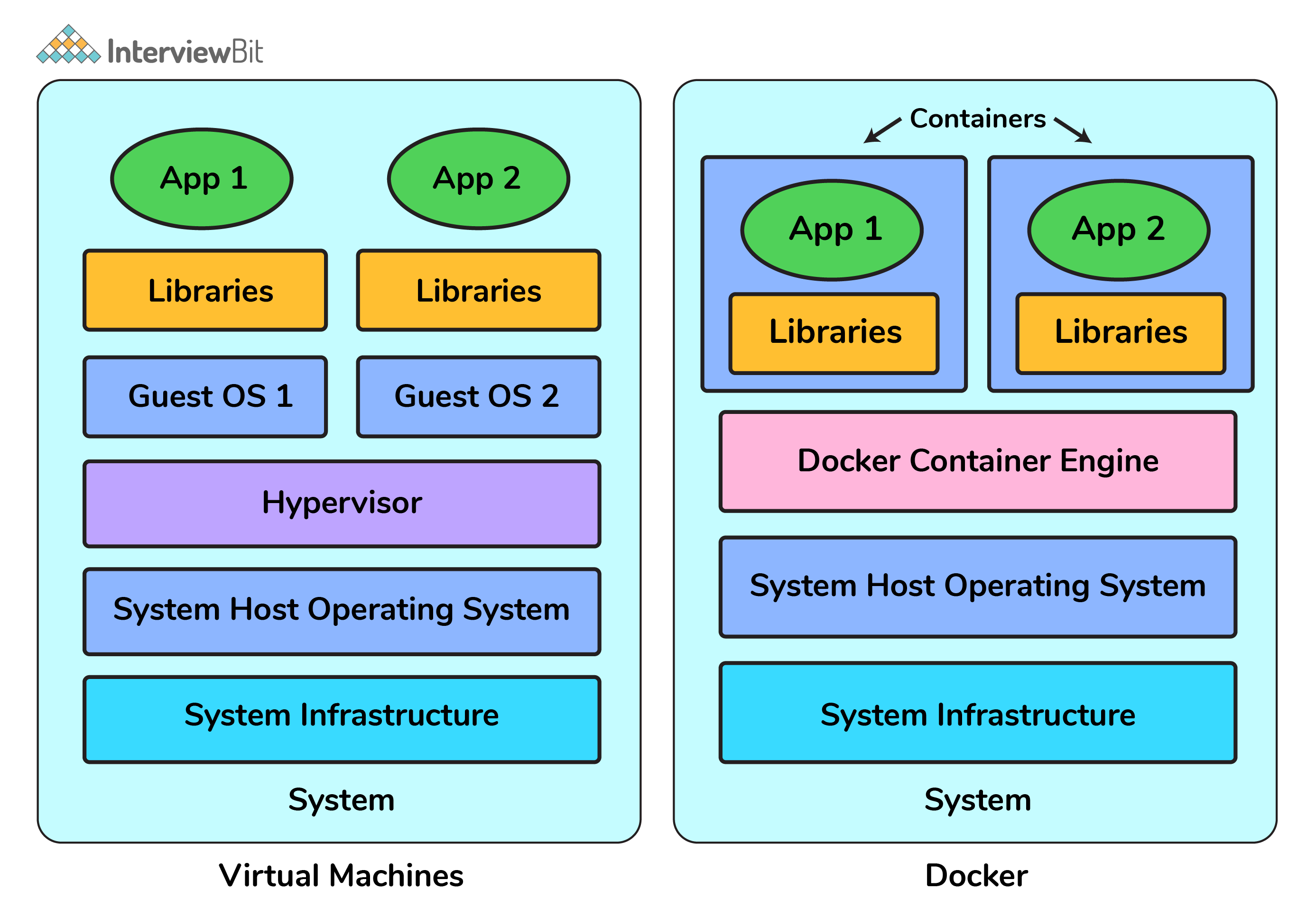
**Introduction to Docker:**  
Docker is a very popular and powerful open-source containerization platform that is used for building, deploying, and running applications. Docker allows you to decouple the application/software from the underlying infrastructure.  
  
**What is a Container?**  
A container 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?

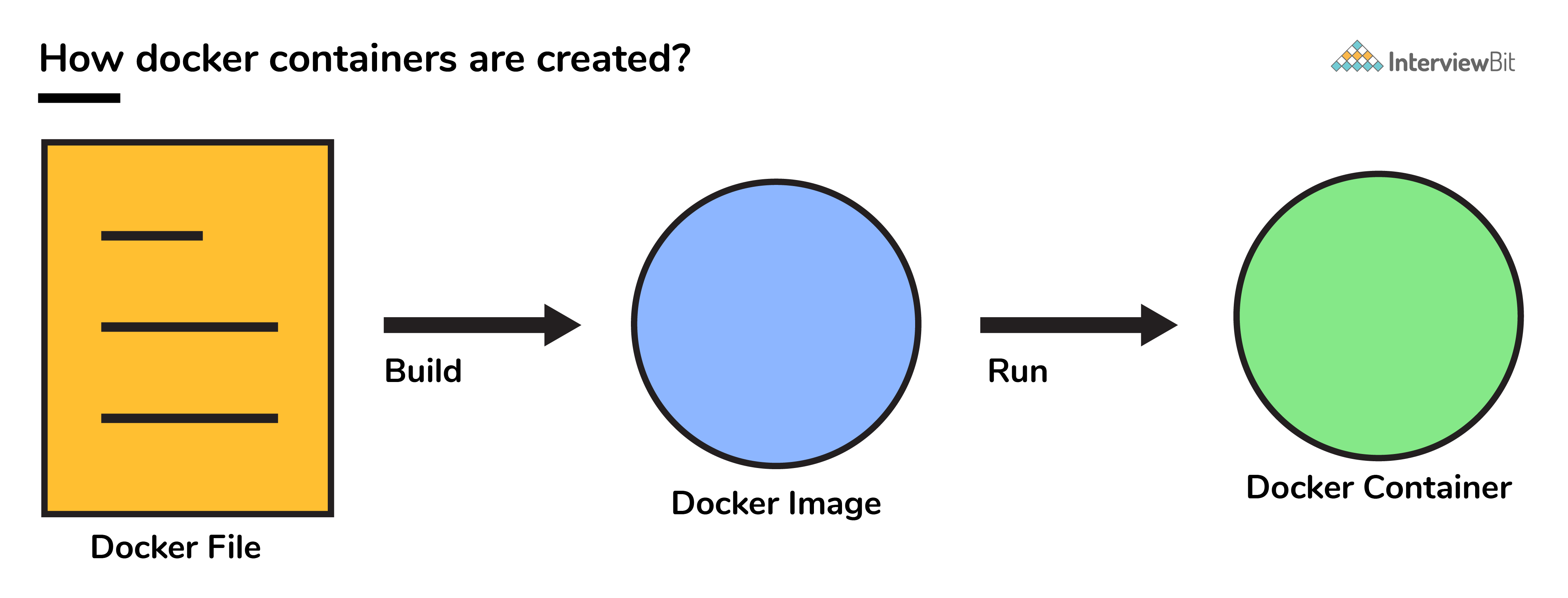
* In simplest terms, docker containers consist of applications and all their dependencies.
* They share the kernel and system resources with other containers and run as isolated systems in the host operating system.
* The main aim of docker containers is to get rid of the infrastructure dependency while 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.

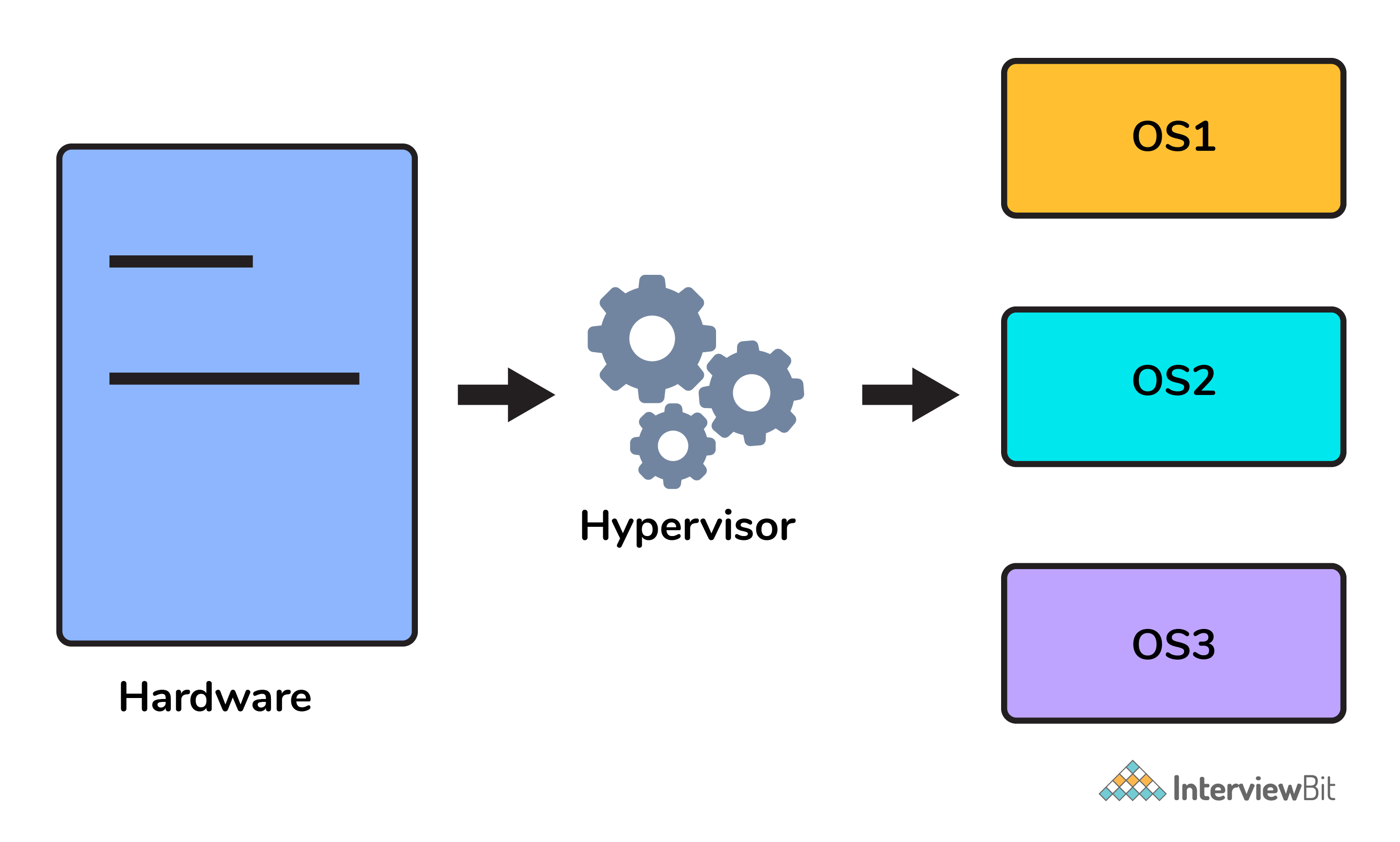


**You can download a PDF version of Docker Interview Questions.**

**Download PDF**

### **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.

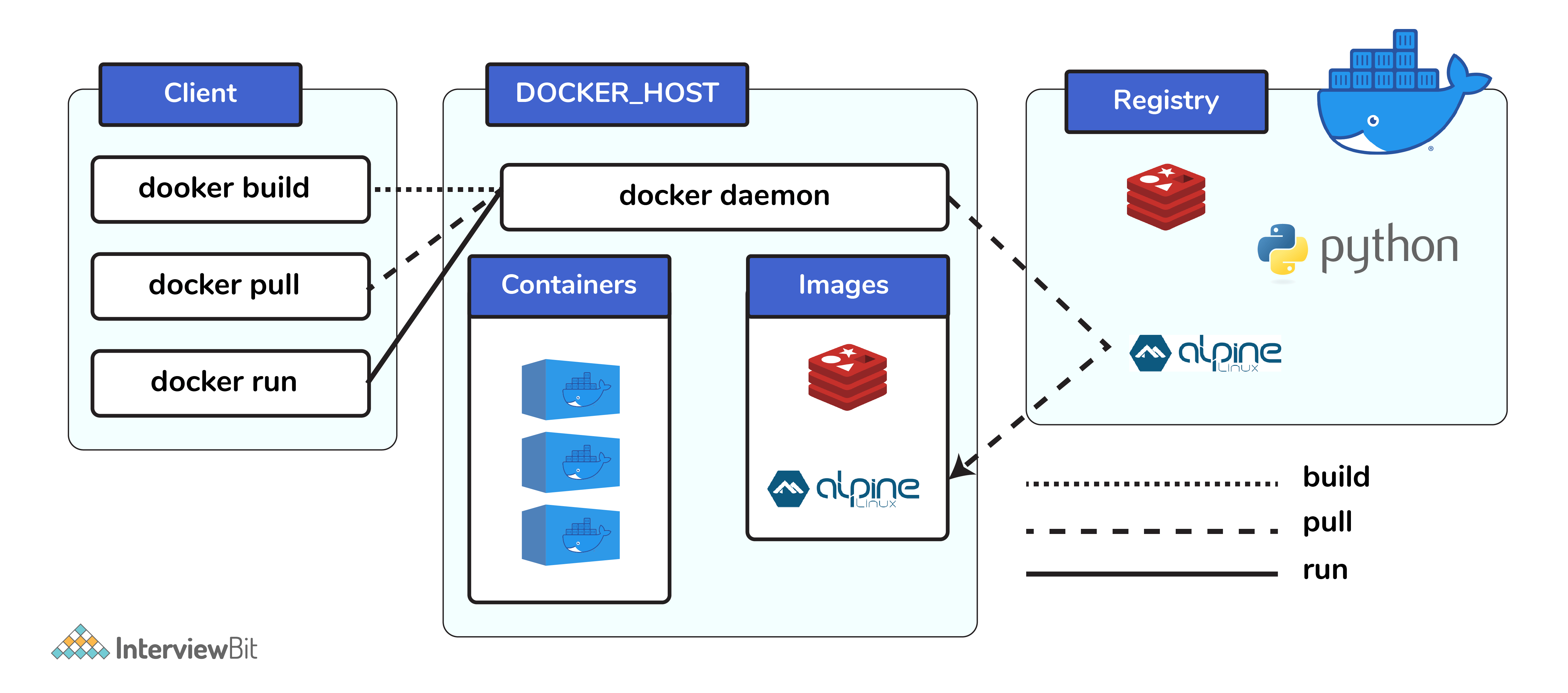
### 9. What is docker image registry?

* 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.

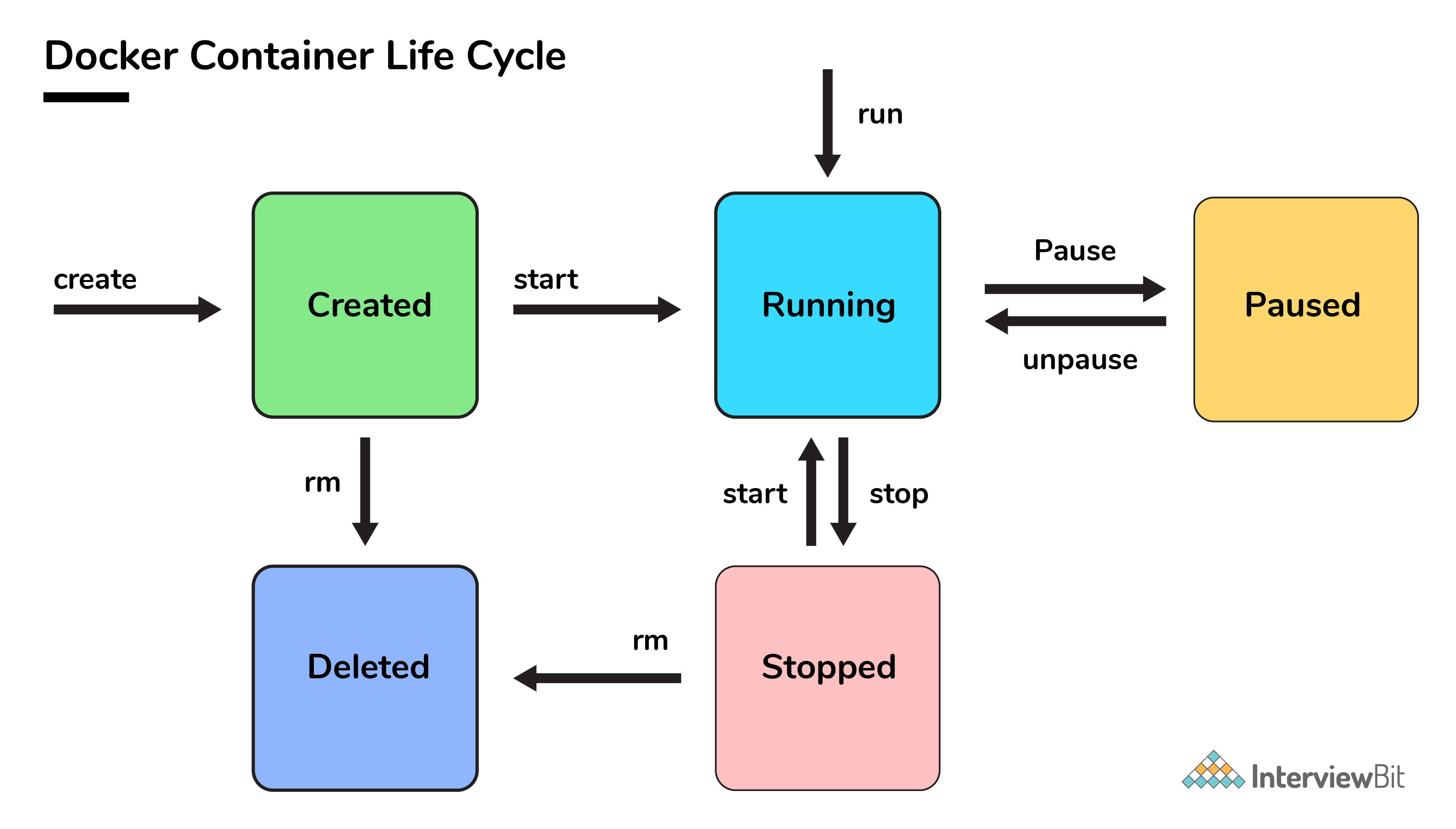
### 32. How many containers you can run in docker and what are the factors influencing this limit?

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.

# Top 50 Docker Interview Questions You Must Prepare In 2023

Last updated on Nov 18,2022*373.5K Views*

Introduced in 2013, Docker hit the IT industry. It turned out to be a big hit with 13 billion + container image downloads per month in 2022. Increasing demand for docker showed an exponential increase in job openings. Go ahead and take advantage of all the new job openings with this article which lists down 50 most important Docker Interview Questions.

I have categorized these 50 questions into:

* [Docker Basic Questions](https://www.edureka.co/blog/interview-questions/docker-interview-questions/#DockerBasicQuestions)
* [Docker Basic Commands](https://www.edureka.co/blog/interview-questions/docker-interview-questions/#DockerBasicCommands)
* [Docker Advanced Questions](https://www.edureka.co/blog/interview-questions/docker-interview-questions/#DockerAdvancedQuestions)

## ****Docker Basic Questions****

This category of Docker Interview Questions consists of questions that you’re expected to know. These are the most basic questions. An interviewer will start with these and eventually increase the difficulty level. Let’s have a look at them.

**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.

Refer to this blog, to read more about [**Docker Architecture**](https://www.edureka.co/blog/what-is-docker-container).

### **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.

The interviewer does not just expect definitions, hence explain how to use a Dockerfile which comes with experience. Have a look at [this](https://www.edureka.co/blog/docker-explained/) tutorial to understand how Dockerfile works.

### **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.

## ****Docker Basic Commands****

Once you’ve aced the basic conceptual questions, the interviewer will increase the difficulty level. So let’s move on to the next section of this Docker Interview Questions article. This section talks about the commands that are very common amongst docker users.

### **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

[[](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=curriculum)](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=curriculum" \t "_blank)

### [Docker Certification Training Course](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=curriculum" \t "_blank)

[Explore Curriculum](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=curriculum" \t "_blank)

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>

### DevOps Training

### [DEVOPS CERTIFICATION TRAINING COURSE](https://www.edureka.co/devops-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "DevOps Certification Training Course" \t "_blank)

### [DevOps Certification Training Course](https://www.edureka.co/devops-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "DevOps Certification Training Course" \t "_blank)

*[Reviews](https://www.edureka.co/devops-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "DevOps Certification Training Course" \t "_blank)*

**[5](https://www.edureka.co/devops-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "DevOps Certification Training Course" \t "_blank)**[(151336)](https://www.edureka.co/devops-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "DevOps Certification Training Course" \t "_blank)

### [KUBERNETES CERTIFICATION TRAINING COURSE: ADMINISTRATOR (CKA)](https://www.edureka.co/kubernetes-certification?utm_source=blogbanner&utm_campaign=trending_course" \o "Kubernetes Certification Training Course: Administrator (CKA)" \t "_blank)

### [Kubernetes Certification Training Course: Administrator (CKA)](https://www.edureka.co/kubernetes-certification?utm_source=blogbanner&utm_campaign=trending_course" \o "Kubernetes Certification Training Course: Administrator (CKA)" \t "_blank)

*[Reviews](https://www.edureka.co/kubernetes-certification?utm_source=blogbanner&utm_campaign=trending_course" \o "Kubernetes Certification Training Course: Administrator (CKA)" \t "_blank)*

**[5](https://www.edureka.co/kubernetes-certification?utm_source=blogbanner&utm_campaign=trending_course" \o "Kubernetes Certification Training Course: Administrator (CKA)" \t "_blank)**[(11650)](https://www.edureka.co/kubernetes-certification?utm_source=blogbanner&utm_campaign=trending_course" \o "Kubernetes Certification Training Course: Administrator (CKA)" \t "_blank)

### [AWS DEVOPS ENGINEER CERTIFICATION TRAINING COURSE](https://www.edureka.co/aws-certified-devops-training?utm_source=blogbanner&utm_campaign=trending_course" \o "AWS DevOps Engineer Certification Training Course" \t "_blank)

### [AWS DevOps Engineer Certification Training Course](https://www.edureka.co/aws-certified-devops-training?utm_source=blogbanner&utm_campaign=trending_course" \o "AWS DevOps Engineer Certification Training Course" \t "_blank)

*[Reviews](https://www.edureka.co/aws-certified-devops-training?utm_source=blogbanner&utm_campaign=trending_course" \o "AWS DevOps Engineer Certification Training Course" \t "_blank)*

**[5](https://www.edureka.co/aws-certified-devops-training?utm_source=blogbanner&utm_campaign=trending_course" \o "AWS DevOps Engineer Certification Training Course" \t "_blank)**[(8299)](https://www.edureka.co/aws-certified-devops-training?utm_source=blogbanner&utm_campaign=trending_course" \o "AWS DevOps Engineer Certification Training Course" \t "_blank)

### [DOCKER CERTIFICATION TRAINING COURSE](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Docker Certification Training Course" \t "_blank)

### [Docker Certification Training Course](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Docker Certification Training Course" \t "_blank)

*[Reviews](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Docker Certification Training Course" \t "_blank)*

**[5](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Docker Certification Training Course" \t "_blank)**[(7354)](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Docker Certification Training Course" \t "_blank)

### [JENKINS CERTIFICATION TRAINING COURSE](https://www.edureka.co/jenkins-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Jenkins Certification Training Course" \t "_blank)

### [Jenkins Certification Training Course](https://www.edureka.co/jenkins-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Jenkins Certification Training Course" \t "_blank)

*[Reviews](https://www.edureka.co/jenkins-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Jenkins Certification Training Course" \t "_blank)*

**[5](https://www.edureka.co/jenkins-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Jenkins Certification Training Course" \t "_blank)**[(9071)](https://www.edureka.co/jenkins-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Jenkins Certification Training Course" \t "_blank)

### [GIT CERTIFICATION TRAINING](https://www.edureka.co/git-certification-training-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Git Certification Training" \t "_blank)

### [Git Certification Training](https://www.edureka.co/git-certification-training-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Git Certification Training" \t "_blank)

*[Reviews](https://www.edureka.co/git-certification-training-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Git Certification Training" \t "_blank)*

**[5](https://www.edureka.co/git-certification-training-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Git Certification Training" \t "_blank)**[(4728)](https://www.edureka.co/git-certification-training-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Git Certification Training" \t "_blank)

### [ANSIBLE CERTIFICATION TRAINING COURSE](https://www.edureka.co/ansible-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Ansible Certification Training Course " \t "_blank)

### [Ansible Certification Training Course](https://www.edureka.co/ansible-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Ansible Certification Training Course " \t "_blank)

*[Reviews](https://www.edureka.co/ansible-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Ansible Certification Training Course " \t "_blank)*

**[5](https://www.edureka.co/ansible-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Ansible Certification Training Course " \t "_blank)**[(696)](https://www.edureka.co/ansible-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Ansible Certification Training Course " \t "_blank)

Next

### **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.

## ****Docker Advanced Questions****

Once the interviewer knows that you’re familiar with the Docker commands, he/she will start asking about practical applications This section of Docker Interview Questions consists of questions that you’ll only be able to answer when you’ve gained some experience working with Docker.

### **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?**

This is a very straightforward question but can get tricky. Do some company research before going for the interview and find out how the company is using Docker. Make sure you mention the platform company is using in this answer.

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.

[[](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=batches)](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=batches" \t "_blank)

### [Docker Certification Training Course](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=batches" \t "_blank)

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/docker-training?utm_source=blogbanner&utm_campaign=batches" \t "_blank)

### **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.

**Commands**

1. docker --version
2. docker run -p 5000:5000 in28min/hello-world-python:0.0.1.RELEASE
3. docker run -p 5000:5000 in28min/hello-world-java:0.0.1.RELEASE
4. docker run -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE
5. docker run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE
6. docker run -d -p 5001:5000 in28min/hello-world-python:0.0.1.RELEASE
7. docker logs 04e52ff9270f5810eefe1f77222852dc1461c22440d4ecd6228b5c38f09d838e
8. docker logs c2ba
9. docker images
10. docker container ls
11. docker container ls -a
12. docker container stop f708b7ee1a8b
13. docker run -d -p 5001:8080 in28min/hello-world-rest-api:0.0.1.RELEASE
14. docker pull mysql
15. docker search mysql
16. docker image history in28min/hello-world-java:0.0.1.RELEASE
17. docker image history 100229ba687e
18. docker image inspect 100229ba687e
19. docker image remove mysql
20. docker image remove in28min/hello-world-java:0.0.1.RELEASE
21. docker container rm 3e657ae9bd16
22. docker container ls -a
23. docker container pause 832
24. docker container unpause 832
25. docker container stop 832
26. docker container inspect ff521fa58db3
27. docker container prune
28. docker system
29. docker system df
30. docker system info
31. docker system prune -a
32. docker top 9009722eac4d
33. docker stats 9009722eac4d
34. docker container run -p 5000:5000 -d -m 512m in28min/hello-world-java:0.0.1.RELEASE
35. docker container run -p 5000:5000 -d -m 512m --cpu-quota=50000 in28min/hello-world-java:0.0.1.RELEASE
36. docker system events
38. docker container stats 4faca1ea914e3e4587d1d790948ec6cb8fa34f26e900c12632fd64d4722fd59a
39. docker stats 42f170966ce613d2a16d7404495af7b3295e01aeb9142e1fa1762bbdc581f502
41. cd /in28Minutes/git/devops-master-class/projects/hello-world/hello-world-python
42. docker build -t in28min/hello-world-python:0.0.2.RELEASE .
43. docker run -p 5000:5000 -d in28min/hello-world-python:0.0.2.RELEASE
44. docker history e66dc383f7a0
45. docker push in28min/hello-world-python:0.0.2.RELEASE
47. cd ../hello-world-nodejs/
48. docker build -t in28min/hello-world-nodejs:0.0.2.RELEASE .
49. docker container run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.2.RELEASE
50. docker push in28min/hello-world-nodejs:0.0.2.RELEASE
52. cd ../hello-world-java/
53. docker build -t in28min/hello-world-java:0.0.2.RELEASE .
54. docker run -d -p 5000:5000 in28min/hello-world-java:0.0.2.RELEASE
55. docker push in28min/hello-world-java:0.0.2.RELEASE
57. docker run -d -p 5001:5000 in28min/hello-world-nodejs:0.0.3.RELEASE ping google.com

60. docker run -d -p 8000:8000 --name=currency-exchange in28min/currency-exchange:0.0.1-RELEASE
61. docker run -d -p 8100:8100 --name=currency-conversion in28min/currency-conversion:0.0.1-RELEASE
63. docker network ls
64. docker network inspect bridge
66. docker run -d -p 8100:8100 --env CURRENCY\_EXCHANGE\_SERVICE\_HOST=http://currency-exchange --name=currency-conversion --link currency-exchange in28min/currency-conversion:0.0.1-RELEASE
68. docker network create currency-network
69. docker container stop currency-exchange
70. docker container stop currency-conversion
71. docker run -d -p 8000:8000 --name=currency-exchange --network=currency-network in28min/currency-exchange:0.0.1-RELEASE
72. docker run -d -p 8100:8100 --env CURRENCY\_EXCHANGE\_SERVICE\_HOST=http://currency-exchange --name=currency-conversion --network=currency-network in28min/currency-conversion:0.0.1-RELEASE
74. docker-compose --version
75. cd ../../microservices/
76. docker-compose up
77. docker-compose up -d
78. docker container ls
79. docker network ls
80. docker network inspect microservices\_currency-compose-network
81. docker-compose down
82. docker container ls -a
83. docker system prune -a
84. docker-compose config
85. docker-compose images
86. docker-compose ps
87. docker-compose top
89. docker build -t in28min/hello-world-java:0.0.1.RELEASE .
90. docker push in28min/hello-world-java:0.0.1.RELEASE
92. docker build -t in28min/hello-world-python:0.0.1.RELEASE .
93. docker push in28min/hello-world-python:0.0.1.RELEASE
95. docker build -t in28min/hello-world-nodejs:0.0.1.RELEASE .
96. docker push in28min/hello-world-nodejs:0.0.1.RELEASE