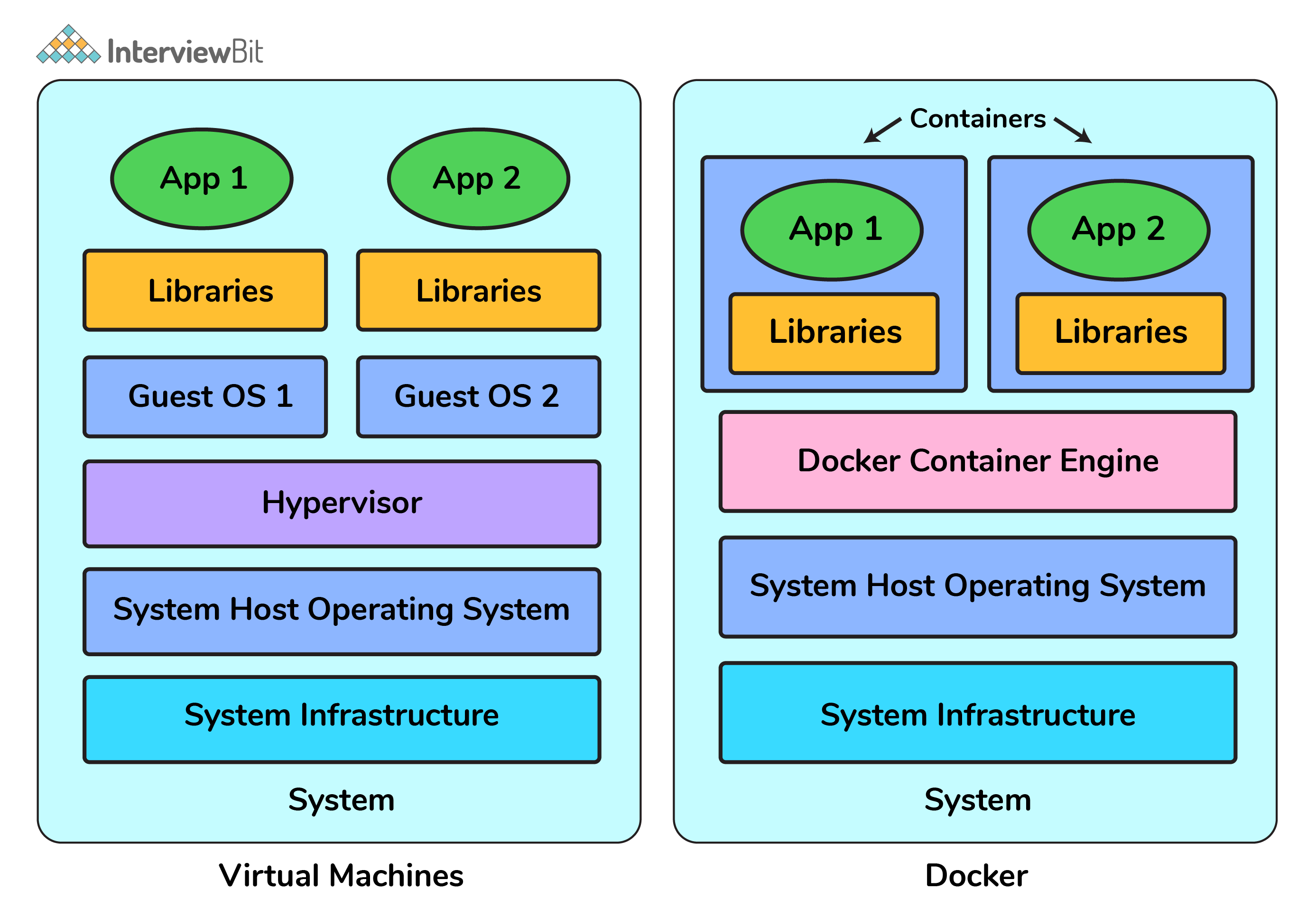
**Introduction to Docker:**  
Docker is a very popular and powerful open-source containerization platform that is used for building, deploying, and running applications. Docker allows you to decouple the application/software from the underlying infrastructure.  
  
**What is a Container?**  
A container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

U,{2fc42bf8-d97b-429b-ba54-9284607f2bf7}{216},3.125,3.125

### **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.

U,{54f6bdfc-caa1-4e7b-8e17-262eb0c57626}{30},3.125,3.125

* 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.

U,{ce90ffea-22c1-4603-9226-f58ab667674e}{13},3.125,3.125

### **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.

U,{f3afe45d-f888-4dc4-b042-0a98137e7ae3}{67},3.125,3.125

### **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.

## Docker Interview MCQs

1. What command is used for stopping a running container?

docker stop <container\_id>

docker rm <container\_id>

docker kill <container\_id>

docker terminate <container\_id>

2.What command is used for running images as a container?

docker ps -a

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

sudo docker run <container\_name>

All of the above

3.What is a cloud-hosted service of Docker providing registry capabilities for public and private content?

Docker Images

Docker Alpine

Docker Hub

Docker Compose

4.What is used for converting a pool of Docker hosts into a single, virtual Docker host?

Docker Images

Docker Compose

Docker Hub

Docker Swarm

5.What command is used for viewing all running containers?

docker ps

docker ps -a

docker rm

docker show

6.What is a template used for describing a build of an image?

Docker Logs

Docker Documents

DockerFile

Docker Stream

7.Multiple containers running on a single machine all share the same resources such as the operating system kernel for instant boot and efficient utilization of RAM. True or False?

True

False

Doesn't make sense

8.What is the default IP address of the Docker host?

0.0.0.0

127.0.0.1

172.17. 0.0/16

192.127.99.1

9.What command is used to go inside a running container?

docker sh <container\_id>

docker exec -it <container\_id> sh

docker access <container\_id>

docker ps -a

10.What command is used for remove all stopped containers, unused networks, build caches, and dangling images?

docker system kill -all

docker kill -a

docker system prune

docker system rm -a

## **Basic Docker Interview Questions for Beginners**

### 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.

#### Prepare Yourself to Answer All Questions!

### 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

#### Get the Must-Have Skills of a Web Developer

### 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

## **Advanced Docker Interview Questions for Experienced Professionals**

### 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>

# **Top 50 Docker Interview Questions You Must Prepare In 2023**

**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:

* A server which is a type of long-running program called a daemon process (the docker command).
* A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface (CLI) client (the docker command). 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.

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.

### **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.

## **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.

### **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.

## **Docker Basic Interview Questions and Answers**

### **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.

## **Docker Advanced Interview Questions**

### **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.

## **Docker Interview Questions For Experienced**

### **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 the possible ways of using insecure Docker image registries?**

In some projects, you might choose private Docker registries rather than Docker Hub or any cloud provider’s registry. This might take the form of deploying a Docker registry server, or perhaps a third-party on-premise registry server like Nexus.

When you want to connect these private registries, your registry should be secured with an SSL certificate in accordance with best practices.

You can also elect to use a private registry insecurely if you want to use self-signed SSL certificates—note, this should only be done for testing purposes. To do this, add your private test registry to an array as the value for the "insecure-registries" key in your daemon.json config file.

**What is the use of the** docker save **and** docker load **commands?**

A Docker image can be exported as an archive via the docker save command. For example:

docker save -o <container-export-path>.tar <container-name>

The exported Docker image can then be imported to another Docker host via the docker load command:

docker load -i <container-path>.tar

Note that this does not export data from any containers that were based on the image, just the image itself.

**What is the default Docker network driver, and how can you change it when running a Docker image?**

Docker provides different network drivers like bridge, host, overlay, and macvlan. bridge is the default.

Sometimes you might want to use Docker Swarm or connect your containers to your host network directly. In these cases, you’ll need to change your default network driver.

First, you have to create a new network with the new network driver by using the --driver or -d parameter with your docker network create command. Then you’ll need to run your Docker image with the --network parameter to use your newly-created network.

**What is container orchestration and why should we use it?**

When you have to manage large and dynamic environments, the docker command alone does not suffice. You will face many problems automating scaling and health checks for containers. In this case, software teams use container orchestration tools like Kubernetes. Such software enables another level of automation:

* Deploy or scale your containers easily, securely, and with high availability
* Provide a service (internally or externally) from a container group
* Move your containers from one host to another when there’s a host-specific problem
* Manage your configuration data—like environment variables—easily

**What are a Docker container’s possible states, and what do they mean?**

**Created**: If your docker container is newly created, you will see this state for your container. In this state, the container is not yet started.

**Restarting**: When you restart your docker container—or container restarts itself due to a problem—you will see this state.

Docker has four different restart policies. The default is called no. With this policy, the Docker daemon will never try to restart your container (unless you tell it to manually.)

The second policy is on-failure. With this policy, the Docker daemon will try to restart containers if any problem exists, that is, if any startup script returns a non-zero exit code.

The third policy is always. With this policy, the Docker daemon will try restart containers if:

1. Any problem exists,
2. You stop them manually, or
3. The docker daemon was itself stopped and restarted

The fourth policy is unless-stopped, where the Docker daemon will always try to restart containers unless you stop them manually.

**Running**: Running is the main state you’ll see for containers. It means it has started, and there is no problem detected with the container itself.

**Paused**: If you temporarily stop your running Docker container via docker pause, this is what you’ll see until you unpause it.

**Exited**: If your container has stopped because of a problem or you stopped your container manually, you will see your container in this state, depending on your restart policy as described above.

**What is a Docker image? What is a Docker image registry?**

A Docker image consists of many layers. Each layer corresponds to a command in an image’s Dockerfile. This image provides isolation for an application when you run a Docker image as a container.

You can run many containers from a single Docker image. Docker images can be built from a Dockerfile.

A Docker image registry is a storage area for Docker images. You can get images from them instead of building them.

An image registry is either public or private. The best-known public registry is Docker Hub.

**What features are provided by Docker Enterprise Edition instead of Docker Community Edition?**

Docker Enterprise Edition provides certified Docker images and plugins. With this certification, Docker Inc. ensures that the images in question pass security and best-practice checks. In other words, they guarantee a certain baseline of reliability.

Docker Enterprise Edition also provides Active Directory or LDAP user integration, continuous vulnerability and security scans, and container app and image management features.

**What is Docker Swarm and which network driver should be used with it?**

Docker Swarm is an open-source container orchestration tool that is integrated with the Docker engine and CLI. If you want to use Docker Swarm, you should use the overlay network driver. Using an overlay network enables the Swarm service by connecting multiple docker host daemons together.

**Is there any problem with just using the** latest **tag in a container orchestration environment? What is considered best practice for image tagging?**

If you’re running your image via the latest tag with a container orchestration environment like Kubernetes, it may cause a problem.

The problem is if you push a new image with just the latest tag, you lose your old image and your deployments will use the new image. If the new image has any problem, your deployments might fail, resulting in downtime.

When you use explicit version numbers to tag Docker images instead, you can roll back to old images easily. Also, when you push a new image to your private registry, your deployments will continue to use the old version number due to your tag until you’re ready to switch each of them over.

The best practice of Docker image tagging is to use both types of tagging. First, tag your Docker images with latest and a version number, then push twice, separately for each tag. For example:

docker tag nginx:latest nginx:0.0.1  
  
docker push nginx:latest  
docker push nginx:0.0.1

**What is Docker Compose? What can it be used for?**

Docker Compose is a tool that lets you define multiple containers and their configurations via a YAML or JSON file.

The most common use for Docker Compose is when your application has one or more dependencies, e.g., MySQL or Redis. Normally, during development, these dependencies are installed locally—a step that then needs re-doing when moving to a production setup. You can avoid these installation and configuration parts by using Docker Compose.

Once set up, you can bring all of these containers/dependencies up and running with a single docker-compose up command.

**What does the volume parameter do in a** docker run **command?**

The volume parameter syncs a directory in a container with a host directory.

For example:

docker run -v nginx-sites:/etc/nginx/sites-available nginx

This command mounts the nginx-sites directory in the host to the /etc/nginx/sites-available directory. In this way, you can sync nginx sites without restarting the container they’re in. Also, you can protect your data that is generated in your container using a directory in the host. Otherwise, if you delete your container, your data that was generated and stored in your container will naturally be deleted.

When you use the volume parameter, you can use the same data that was generated in a previous container using the same command.

**What is the main difference between the approaches of Docker and standard hypervisor virtualization?**

With standard virtualization using a hypervisor like vSphere, an operating system is necessary for each app. A host operating system is at the bottom of your infrastructure, and a hypervisor has to be installed on your host OS. Then on top of the hypervisor, you install operating systems for each of your applications.

With Docker, the Docker daemon sits between your host operating system and your Docker images, in place of a hypervisor. Docker images reuse parts of the host operating system—thus a separate OS is not necessary for each app—but your apps are still isolated like they would be with a standard hypervisor.

## [**What is a Volume in docker?**](https://www.knowledgehut.com/interview-questions/docker#collapse-beginner-2875)

A data volume is a specially-designated directory that is located outside of the root filesystem of a container (i.e. created on the host), designed to persist data, independent of the container’s life cycle. This allows sharing data within containers by importing volume directory in other containers.

Data volumes provide several useful features:

* Data volumes persist even if the container itself is deleted.
* Data volumes can be shared and reused among containers.
* Changes to a data volume can be made directly.
* Volumes can be initialized when a container is created.

## [**When is .dockerignore file used?**](https://www.knowledgehut.com/interview-questions/docker#collapse-beginner-2876)

A typical Dockerfile contains one or more COPY commands to copy files and/or folders from the developer machine to the docker image, which eventually become part of the container. While copying folders to a docker image, it is quite possible that some unwanted files are also copied to the image. This may create a bulky image and hence cause performance issues in the container.

In order to avoid this, we can create a file named **.***dockerignore* along with Dockerfile in the same directory. This file is used to list all the files and directories that need to be excluded while copying folders onto the image. It contains a pattern and none of the files matching it is added to the image. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images.

## [**What are the steps involved while using Docker for application development?**](https://www.knowledgehut.com/interview-questions/docker#collapse-beginner-2880)

All the steps below are based on the prerequisite that Docker is already installed on the machine:

* **Create Dockerfile:**

The initial step is to create a Dockerfile file using a suitable base image along with all the required steps/commands, like setting environment variables, adding application jar, etc. This creates several layers on the existing base image.

* **Build image:**

Once the Dockerfile is ready, we can either use docker command or via a Gradle task to generate a docker image. This image contains all the application dependencies required to run the application in a container.

*docker build -t-test/security tool.*

* **Run the image:**

Once the docker image is built, we can create and start the container using command.

*docker run --name rest\_tool test/security tool.*

* **Start Containers using Compose:**

In case we have multiple containers constituting an application like database, messaging queue, etc.; then it is advisable to use docker-compose to run multiple containers simultaneously. It is also useful in the CI pipeline for running the application and performing tests.

*docker-compose up*

* **Test the Application:**

After the containers are up and running, the application is ready for Integration or Acceptance tests to be performed. Ideally, it is integrated into the CI pipeline for determining if the new code changes are affecting the existing flows.

* **Push image:**

Typically in a container-based development environment, the deliverable artifact is a docker image. This image needs to be published to an internal Registry like Artifactory so that it can be propagated to next levels like Continuous Delivery and Deployment pipelines.

*docker push test/securityTool*

* **Production orchestration:**

Ideally, organizations need to use orchestration tools like Kubernetes to run the containers in a Pod to perform load balancing, service discovery, etc in a production environment. It also helps in providing scalability and high availability of the application.

## [**Explain basic Docker usage workflow.**](https://www.knowledgehut.com/interview-questions/docker#collapse-beginner-2879)

Everything starts with the Dockerfile. The Dockerfile is the source code of the Image.  
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.

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.

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).

aaaa