

## **DOCKER QUESTIONS**

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

Docker is a containerization platform that combines all the dependencies of our applications in a package so that we can ship this package in any environment and run our application seamlessly. This means, our application will run seamlessly on any environment, and this makes it easy to have a product-ready application. Containerization technology like Docker will share the same operating system kernel with the machine, and due to this, it is extremely fast. Docker and Virtual Machines are both used to isolate and run applications on a host machine, but they differ in their approach and the level of isolation they provide.

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

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

### **3. Define Virtualization.**

- Virtualization in Layman terms refers to running multiple operating systems on a single machine.
- While most computers only have one operating system installed, virtualization allows a computer to run several operating systems on top of the same physical machine.

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

- A hypervisor helps in the creation of a virtual environment, in which the guest virtual machines run. It manages the guest systems and checks the required resource allocations to the guests.
- Prepare yourself for the DevOps interview by learning from these Top DevOps Interview Questions and Answers!

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

Benefits of using Docker over a Hypervisor:

- More efficient in resource management
- Less overhead so faster startup time.
- Highly portable across diverse environments

### **6. What is the difference between Docker and Virtual Machines?**

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

## 7. Why use Docker?

Following are the reasons why one should use Docker:

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

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

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

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

## 9. How to install Docker on Linux Operating System?

```
sudo apt-get update
```

```
sudo apt-get install docker.io -y
```

## 10. How to check if Docker is installed on a system?

You can use the following command: `docker --version`

## 11. What are the main components of Docker architecture?

The Docker follows a client-server architecture which include:

- Docker Client
- Docker Host
- Network and Storage Components
- Docker Hub/Registry

## 12. What is a Docker image?

A Docker image helps in creating a Docker container. It is a static file with executable code that can create a Docker container. We can create the Docker image with the `docker build` command and a `Dockerfile`. All the Docker images are stored in the Docker registry such as DockerHub. These have minimal amounts of layers within the image so that there is a minimum amount of data on the network.

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

A Docker container is the running instance of a Docker image. Docker is not tied to any IT infrastructure, and thus it can run on any computer system or on the cloud. We can create a Docker container using Docker images and then run it, or we can use the images that are already created in the Docker Hub. To simplify things, let's say that the Docker containers are just runtime instances of the Docker image.

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

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

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

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

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

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

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

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

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

Docker has a few drawbacks as listed below:

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

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

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

- Docker Daemon
- Docker Engine REST API
- Docker CLI

## **20. What are registries?**

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

- Public registry
- Private registry
- Public registries include Docker Hub and Docker Cloud.

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

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

## **22. What are Docker's most notable features?**

Docker's most key features include:

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

## **23. 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.

## **24. Describe a Docker container's lifecycle.**

The most common steps in a Docker container's lifecycle are as follows:

- Create container
- Run container
- Pause container
- Unpause container
- Start container
- Stop container
- Restart container
- Kill container
- Destroy container

## **INTERMEDIATE QNS:**

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

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

```
docker-compose -f docker-compose.json up
```

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

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

```
docker pull ubuntu
```

```
docker run -it -d --name <container_name> <image_name>
```

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

```
docker ps -a
```

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

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

```
sudo docker stop CONTAINER_ID
```

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

```
docker restart CONTAINER_ID
```

### **28. How to create a Dockerfile?**

In order to create a Dockerfile, follow these steps:

- Create a new file and name it "Dockerfile"
- Write the below script inside this Dockerfile

```
FROM ubuntu
```

```
RUN apt-get update
```

```
RUN apt-get install -y apache2
```

Save this file. This Dockerfile script pulls the Ubuntu base image, updates the package lists, and attempts to install the Apache2 server.

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

A Docker image can be built using the following command:

```
$ docker build . -t <image_name>
```

### 30. How to stop a Docker container?

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

```
docker container stop container-id
```

### 31. How to list all the Docker images?

We can use the following command to list all the Docker images:

```
Removing intermediate container ec5004b30270
--> 568a48b8b307
Successfully built 568a48b8b307
Successfully tagged apache_image:latest
root@ip-172-31-10-161:/home/ubuntu# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
apache_image	latest	568a48b8b307	About a minute ago	231MB
ubuntu	latest	e4c58958181a	3 weeks ago	77.8MB

### 32. How to remove Docker images?

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

```
docker rmi <image_name>
```

List:

```
docker images -a
```

Remove:

```
docker rmi <image_name>
```

### 33. Where are Docker images stored?

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

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

```
C:\ProgramData\DockerDesktop
```

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

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

### 34. What is host port and container port?

A host port specifies a port on the host to bind to whereas a container port specifies a port within a container.

### 35. How to run a Docker container?

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

Running a container under a specific name:

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

- `docker container run -it --name [container_name] [docker_image]`

```
root@ip-172-31-10-161:/home/ubuntu# docker run -it --name First_container ubuntu
root@78d8fa593bd7:/#
```

- `docker container run -it --name [container_name] [docker_image]`

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

The command for running a container in the background is:

`docker container run -it -d [docker_image]`

```
root@ip-172-31-10-161:/home/ubuntu# docker run -it -d ubuntu
```

### Running a container interactively:

The following command is run for running a container interactively:

`docker container run -it [docker_image]`

```
root@ip-172-31-10-161:/home/ubuntu# docker run -it ubuntu
root@90e4be75b21a:/#
```

### Running a container and publishing container ports:

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

`-p [host_ip]:[host_port]:[container_port]`

```
root@ip-172-31-10-161:/home/ubuntu# docker run -it -d --name hosted_container -p 81:80 ubuntu
4bff209fe37027aa1f9489436b0761195e6ad752d6ce1b6681b77f4a48b1fd4e
root@ip-172-31-10-161:/home/ubuntu#
```

Here, `host_ip` is optional. It is not mandatory to specify this while we run the command.

### Running a container and mounting host volumes:

The docker container run command looks like this:

`docker container run -v [/host/volume/location]:[/container/storage] [docker_image]`

## 36. How to start a Docker container?

The following command starts a Docker container:

`docker container start container_name`

## 37. How to use Docker Compose?

Docker Compose typically includes a three-step process:

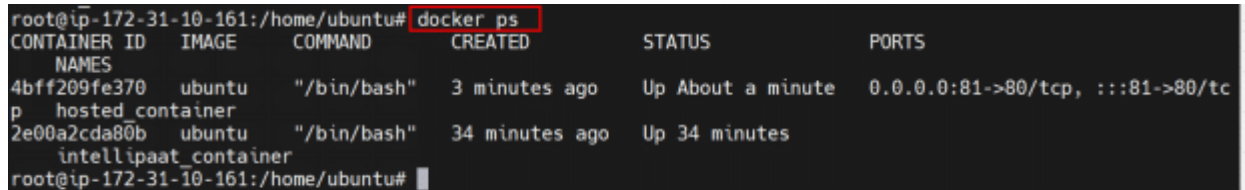
- Using a `dockerfile` to define the app's environment to facilitate reproduction anywhere
- Defining the app services in `docker-compose.yml` so that they can run in an isolated environment together

- Running docker-compose up

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

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

\$ docker ps



```

root@ip-172-31-10-161:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
4bfff209fe370  ubuntu   "/bin/bash"   3 minutes ago   Up About a minute   0.0.0.0:81->80/tcp, :::81->80/tc
2e00a2cda80b   ubuntu   "/bin/bash"   34 minutes ago   Up 34 minutes

```

### 39. What is Docker daemon?

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

### 40. Name and explain the states of a Docker container.

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

### 41. What is Docker Hub?

Docker Hub helps with linking to code repositories. This cloud-based registry enables the building, testing, and storing of images in Docker Cloud. Images can also be deployed to the host with it. You can visit <https://hub.docker.com/>

### 42. How to rename a docker image?

To rename a Docker image, use docker tag to create a new tag for the existing image with the desired name and tag. Then, use docker rmi to remove the old tag. For example:

```

docker tag old_image:new_tag new_image:new_tag</b>

<b>docker rmi old_image:new_tag

```



```

root@ip-172-31-10-161:/home/ubuntu# docker image tag new_image docker6767/rename_image
root@ip-172-31-10-161:/home/ubuntu# docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
docker6767/rename_image latest      e4c58958181a  3 weeks ago   77.8MB
new_image            latest      e4c58958181a  3 weeks ago   77.8MB
ubuntu               latest      e4c58958181a  3 weeks ago   77.8MB
root@ip-172-31-10-161:/home/ubuntu#

```

#### 43. How to push images to DockerHub?

Try the following commands to push the images to your DockerHub repository:

1. Login to your Dockerhub account using the below command:  
docker login
2. Enter the Username and the password.

```

root@ip-172-31-10-161:/home/ubuntu# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: docker6767
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@ip-172-31-10-161:/home/ubuntu#

```

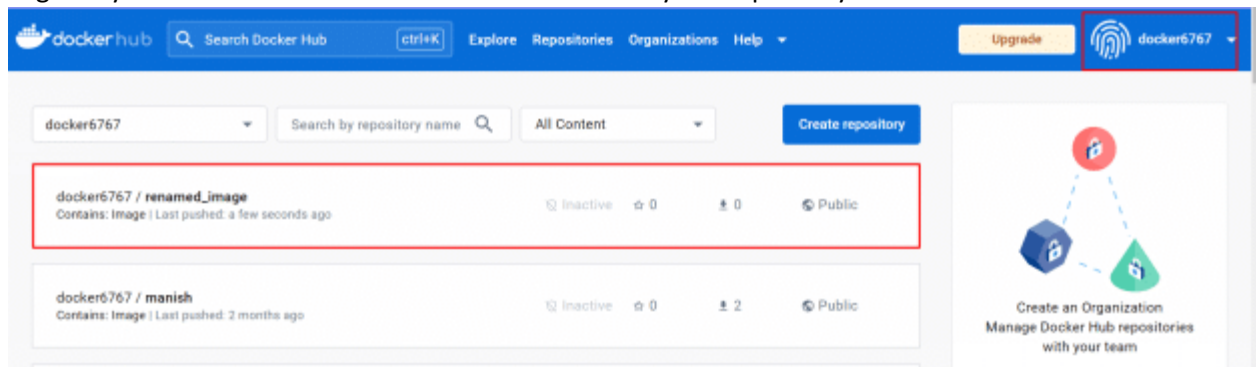
3. Run: docker push <image\_name>

```

root@ip-172-31-10-161:/home/ubuntu# docker push docker6767/rename_image
Using default tag: latest
The push refers to repository [docker.io/docker6767/rename_image]
256d88da4185: Mounted from library/ubuntu
latest: digest: sha256:c0eef2c2053b7e7a644da62acce5c26fdd655d581d3e52007bb97f567fbcd444 size: 529
root@ip-172-31-10-161:/home/ubuntu#

```

4. Login to your DockerHub account via browser and verify the repository.



#### 44. What are Docker object labels?

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

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

Build

Pull

Run

## ADVANCED QNS:

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

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

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

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

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

Run Jenkins on Docker

Using docker-compose, run integration tests in Jenkins

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

This is a question wherein we could bring upon our whole experience with Docker and any other Container technologies we have used prior to Docker. We could also explain the ease that this technology has brought in the automation of the development-to-production life cycle management.

We can also discuss any other integrations that we might have worked, along with Docker, such as Puppet, Chef, or even the most popular of all technologies—Jenkins. If we do not have any experience with Docker but we have it with similar tools from this space, we could convey the same and also show our interest in learning this leading containerization technology.

### **50. List and explain the key commands for interacting with Docker, including what purpose each serves in working with containers.**

The key Docker commands and their purposes are:

1. **docker run**: Creates and runs a new container from an existing Docker image.
2. **docker ps**: Lists all running containers, along with their container IDs, images, names, ports, and statuses.
3. **docker stop**: Stops a specified running container, gracefully terminating all processes within the container.
4. **docker rm**: Removes a specified stopped container, eliminating its existence on the system.
5. **docker inspect**: Displays detailed information about a specific container, including its configuration, environment variables, network settings, and mount points.
6. **docker exec**: Executes a specified command in a running container, providing direct access to the container's environment.
7. **docker logs**: Retrieves the logs generated by a specific container, both running and stopped, offering insights into its behavior.

These commands enable creating, managing, and interacting with Docker containers, helping in deployment and maintenance tasks.

## 51. What's the difference between virtualization and containerization?

Virtualization and containerization are both methods for isolating and managing software applications, but they differ in their approach:

**Virtualization:** Creates a complete virtual machine (VM) that includes its own operating system, kernel, and applications. This provides a high degree of isolation and resource allocation, but it can be more resource-intensive and complex to manage.

**Containerization:** Packages an application and its dependencies into a lightweight container that shares the host's operating system and kernel. This approach is more efficient and resource-friendly, but it may lead to conflicts if containers from different sources use incompatible versions of libraries.

## 52. What is the lifecycle of a Docker Container?

The lifecycle of a Docker container consists of the following stages:

1. **Create:** The container image, a blueprint for the container, is downloaded from a registry or built locally.
2. **Start:** The container is initialized and its processes begin executing, bringing the application to life.
3. **Run:** The container is actively running and its processes are providing services or performing tasks as intended.
4. **Stop:** The container is halted, terminating all its running processes and gracefully shutting it down.
5. **Remove:** The container is completely eliminated from the system, removing its files, processes, and associated metadata.
6. **Remove:** The container is completely eliminated from the system, removing its files, processes, and associated metadata.

## 53. Explain what the Docker system prune command does and its benefits in container management.

The Docker system prune cleans unused data like stopped containers, images, and cache. This command reclaims disk space, enhances system performance, and simplifies Docker's environment, maintaining efficient resource utilization and management.

### **DOCKER SENARIO BASED QNS:**

## 54. What are the differences between Docker Swarm and Kubernetes?

Both are container orchestration platforms, but there are some key differences:

Docker Swarm is Docker's native orchestration tool and is easier to set up and use compared to Kubernetes. Swarm has a smaller learning curve and is ideal for smaller-scale deployments.

On the other hand, Kubernetes provides advanced features like automatic scaling, rolling updates, service discovery, and load balancing. It is better suited for larger-scale and more complex deployments.

## 55. How do you achieve load balancing with Docker Swarm?

It can be performed by using the built-in load-balancing feature. When a service is deployed in a Docker Swarm cluster, multiple containers are created to run the service.

The Swarm's load balancer then automatically distributes incoming requests across all available containers running the service, Thus, making sure that the load is evenly distributed. This delivers better availability and scalability for the application.

Also, you can set advanced load balancing options such as session stickiness and routing modes to adjust the behaviour of the load balancer according to your application's needs.

## **56. How do you troubleshoot issues with Docker containers?**

Troubleshooting Docker containers involves several steps:

1. Check the container's logs for any error messages or abnormalities using the `docker logs` command.
2. Inspect the container's metadata and runtime details with commands like `docker inspect` or `docker stats`.
3. Verify the container's resource allocation and constraints such as CPU and memory limits.
4. Check the host system's logs for any related issues or resource constraints.
5. If networking issues are suspected, examine the container's network configuration and connectivity.
6. Ensure that the Docker daemon and related services are running correctly.
7. Consult the Docker documentation, community forums, and relevant online resources for specific error messages or known issues.
8. If necessary, recreate or redeploy the container to rule out any configuration or state-related issues.

## **57. How do you limit the resources consumed by a Docker container?**

Resource constraints can be set during container creation or updated dynamically using the `docker update` command. For example:

### **Limiting CPU usage:**

Use the `--cpu-period` and `--cpu-quota` options to set the CPU share and the maximum amount of CPU time a container can use.

### **Limiting memory usage:**

Use the `--memory` and `--memory-swap` options to limit the amount of memory a container can use and prevent it from exceeding the set limit.

### **Limiting disk I/O and network bandwidth:**

Docker provides options like `--device-read-bps`, `--device-write-bps`, `--device-read-iops`, and `--device-write-iops` to control the rate of I/O operations a container can perform.