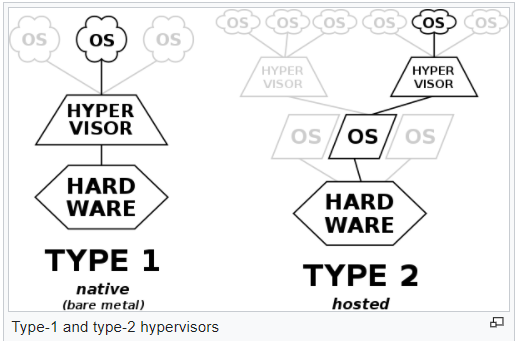
**Docker Basic Questions**

1. **What is Hypervisor?**

* A hypervisor is software that creates and runs virtual machines.
* A hypervisor is also called as Virtual Machine Monitor.
* A hypervisor allows one host computer to support multiple guest virtual machines (VMs) by virtually sharing its resources such as memory and processing

There are two types of Hypervisors:

* **Type 1 Hypervisor:** Hypervisor that runs directly on the system hardware -A bare metal embedded hypervisor.
* **Type 2 Hypervisor:** Hypervisors that runs on a host operating system that provides virtualization services, such as I/O device support and memory management.



1. **What is virtualization?**

* Virtualization is the process of running the virtual systems or resources on top a sing physical machine. These resources could be a storage device, network or even an operating 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.

1. **What is containerization?**

* Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package.
* Application Containerization is an OS-level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each app.
* 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. Most famous containerization environments are Docker and Kubernetes.

1. **Difference between virtualization and containerization**

* 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.

|  |  |
| --- | --- |
| **Virtualization** | **Containerization** |
| Virtualizes hardware resources | Virtualizes only OS resources |
| Requires the complete OS installation for every VM | Installs the container only on a host OS |
| A kernel is installed for every virtualized OS | Uses only the kernel of the underlying host OS |
| Heavyweight | Lightweight |
| Limited performance | Native performance |
| Fully isolated | Process-level isolation |

1. **What is Docker?**

* Docker is a computer program that performs operating-system-level virtualization also known as containerization
* Docker is used to run the software packages called containers.
* Docker is an open-source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools.
* 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.

1. **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.

1. **What are Docker Images?**

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

1. **What is the difference between Docker container and image?**

* Docker images are read-only templates used to build containers.
* Containers are deployed instances created from those templates.
* Images and containers are closely related and are essential in powering the Docker software platform.

1. **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.

1. **What is a Dockerfile?**

* 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.
* A Dockerfile contains all the instructions, e.g., the Linux commands to install and configure the software. Dockerfile creation, as we already know, is the primary way of generating a Docker image. When we use the build command to create an image, it can refer to a Dockerfile available on our path or to a URL such as the [GitHub](https://intellipaat.com/blog/what-is-github-github-tutorial/) repository.

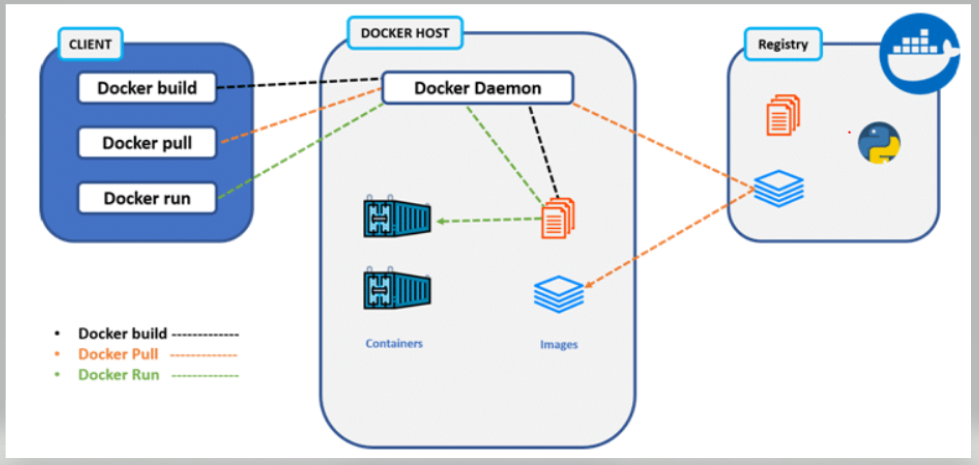
1. **What is a Docker file Instructions?**

* The instructions in a Dockerfile are executed in the same order as they are found in the Dockerfile.
* There can also be comments starting with the # character in the Dockerfile.
* The following table contains the list of instructions available:

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| FROM | The first instruction in the Dockerfile, it identifies an image to inherit from |
| MAINTAINER | This instruction provides visibility as well as credit to the author of the image |
| RUN | This instruction executes a Linux command to install and configure |
| ENTRYPOINT | The final script or application which is used to bootstrap the container and make it an executable application |
| CMD | This instruction uses a JSON array to provide default arguments to the ENTRYPOINT |
| LABEL | This instruction contains the name/value metadata about the image |
| ENV | This instruction sets the environment variables |
| COPY | This instruction copies files into the container |
| ADD | This instruction is basically an alternative to the COPY instruction |
| WORKDIR | This sets a working directory for RUN, CMD, ENTRYPOINT, COPY, and/or ADD instructions |
| EXPOSE | The ports on which the container listens |
| VOLUME | This instruction is to create a mount point |
| USER | An instruction to run RUN, CMD, and/or ENTRYPOINT instructions |

1. **Explain Docker Architecture?**

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



The basic architecture in Docker consists of three parts:

* 1. Docker Client
  2. Docker Host
  3. Docker Registry

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

In Docker host, we have Docker daemon and Docker objects such as containers and images.

1. **Docker Objects:**

* **What is a Docker image?** A Docker image is a type of recipe/template that can be used for creating Docker containers. It includes steps for creating the necessary software.
* **What is a Docker container?** A type of virtual machine created from the instructions found within the Docker image. It is a running instance of a Docker image that consists of the entire package required to run an application.

1. **Docker Daemon:**

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

1. **Docker Registry**
   * Docker registry is a repository for Docker images which is used for creating Docker containers.
   * We can use a local/private registry or the Docker hub, which is the most popular social example of a Docker repository.
2. **What is the lifecycle of a Docker Container?**

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

1. **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

1. **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.

1. **What is 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.

1. **What is Docker Swarm?**

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

**Docker Advanced Questions**

1. **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.

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

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.

1. **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.

1. **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

1. **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.

1. 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 dot Cloud, 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.

1. 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

1. 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, the following command:

$ docker ps -a

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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

1. 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.

1. 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.

**Docker Basic Commands**

$ docker version 🡪 To check Docker client and Docker Sever Version

$ docker info 🡪 To get the number of containers running, paused and stopped

$ docker –help 🡪 is very useful as it gives you help on how to use a command, the syntax, etc.

$ docker <command> --help 🡪 Lists all Docker commands. Need help with one specific command

$ docker login 🡪 **To login into docker repository? (**hub.docker.com)

$ docker pull <image\_name> 🡪 To pull an image from docker hub onto your local system

$ docker run -it -d <image\_name> 🡪 To run containers from their image name

$ docker ps 🡪 To lists down all the running containers

$ docker ps -a 🡪 If there are any stopped containers they can be seen by adding -a flag

$ docker exec -it <container id> bash🡪For logging into/accessing the container

$ docker start <container\_id> 🡪 To start a docker container

$ docker stop <container\_id> 🡪 For stopping a running container

$ docker kill <container\_id> 🡪 kill a container with the following command:

$ docker commit <container id> <username/imagename> 🡪 To commit the container

$ docker push <username/image name> 🡪 To push it to docker hub

$ docker rm <container id> 🡪 To delete as stopped container

$ docker rmi <image-id> 🡪 To delete an image from the local storage system

$ docker build <path to docker file> 🡪 To Build a docker file

$ docker system prune 🡪 It 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.

**Management Commands:**

config 🡪 Manage Docker configs

container 🡪 Manage containers

image 🡪 Manage images

network 🡪 Manage networks

node 🡪 Manage Swarm nodes

plugin 🡪 Manage plugins

secret 🡪 Manage Docker secrets

service 🡪 Manage services

stack 🡪 Manage Docker stacks

swarm 🡪 Manage Swarm

system 🡪 Manage Docker

trust 🡪 Manage trust on Docker images (experimental)

volume 🡪 Manage volumes

**Commands:**

Attach 🡪 Attach local standard input, output, and error streams to a running container

build 🡪 Build an image from a Dockerfile

commit 🡪 Create a new image from a container's changes

cp 🡪 Copy files/folders between a container and the local filesystem

create 🡪 Create a new container

diff 🡪 Inspect changes to files or directories on a container's filesystem

events 🡪 Get real time events from the server

exec 🡪 Run a command in a running container

export 🡪 Export a container's filesystem as a tar archive

history 🡪 Show the history of an image

images 🡪 List images

import 🡪 Import the contents from a tarball to create a filesystem image

info 🡪 Display system-wide information

inspect 🡪 Return low-level information on Docker objects

kill 🡪 Kill one or more running containers

load 🡪 Load an image from a tar archive or STDIN

login 🡪 Log in to a Docker registry

logout 🡪 Log out from a Docker registry

logs 🡪 Fetch the logs of a container

pause 🡪 Pause all processes within one or more containers

port 🡪 List port mappings or a specific mapping for the container

ps 🡪 List containers

pull 🡪 Pull an image or a repository from a registry

push 🡪 Push an image or a repository to a registry

rename 🡪 Rename a container

restart 🡪 Restart one or more containers

rm 🡪 Remove one or more containers

rmi 🡪 Remove one or more images

run 🡪 Run a command in a new container

save 🡪 Save one or more images to a tar archive (streamed to STDOUT by default)

search 🡪 Search the Docker Hub for images

start 🡪 Start one or more stopped containers

stats 🡪 Display a live stream of container(s) resource usage statistics

stop 🡪 Stop one or more running containers

tag 🡪 Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

top 🡪 Display the running processes of a container

unpause 🡪 Unpause all processes within one or more containers

update 🡪 Update configuration of one or more containers

version 🡪 Show the Docker version information

wait 🡪 Block until one or more containers stop, then print their exit codes

A self-sufficient runtime for containers

**Options:**

--config string Location of client config files (default "/root/.docker")

-D, --debug Enable debug mode

-H, --host list Daemon socket(s) to connect to

-l, --log-level string Set the logging level("debug"|"info"|"warn"|"error"|"fatal")(default info")

--tls Use TLS; implied by --tlsverify

--tlscacert string Trust certs signed only by this CA (default "/root/.docker/ca.pem")

--tlscert string Path to TLS certificate file (default "/root/.docker/cert.pem")

--tlskey string Path to TLS key file (default "/root/.docker/key.pem")

--tlsverify Use TLS and verify the remote

-v, --version Print version information and quit