**Docker Interview Questions**

**Q1. What is Docker?**

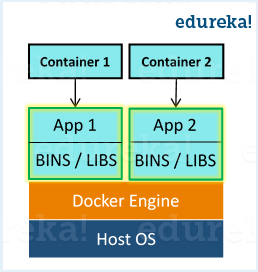
I will suggest you to start with a small definition of Docker.

Docker is a containerization platform which packages your application and all its dependencies together in the form of containers so as to ensure that your application works seamlessly in any environment be it development or test or production.

Now you should explain Docker containers.

Docker containers, wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries etc. anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

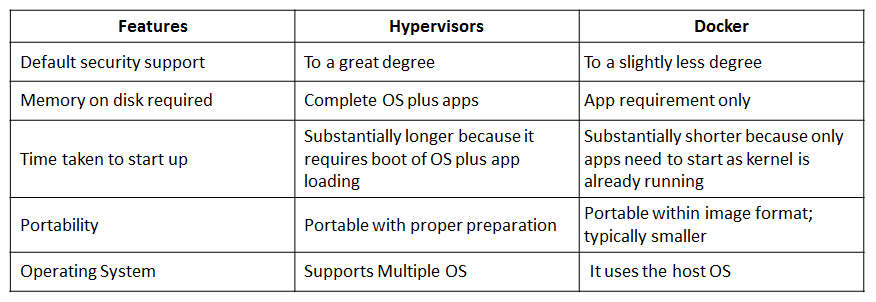
You can refer the diagram shown below, as you can see that containers run on a single machine share the same operating system kernel, they start instantly as only apps need to start as the kernel is already running and uses less RAM.

**Note: Unlike Virtual Machines which has its own OS Docker containers uses the host OS**  

*As you have mentioned about Virtual Machines in your previous answer, so the next question in this Docker Interview Questions blog will be related to the differences between the two.*

**Q2. What are the differences between Docker and Hypervisors?**

You can refer the below differences:



*Next set of Docker interview questions will focus on various components of Docker.*

**Q3. What is Docker image?**

I will suggest you to go with the below mentioned flow:

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

**Q4. What is 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.

**Q5 What is Docker hub?**

Answer to this question is pretty direct.

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

[**Learn Docker With DevOps Now**](https://www.edureka.co/devops/?utm_source=blog&utm_medium=button&utm_campaign=docker-interview-questions-learn-docker-with-devops-now)

**Q6.** **How is Docker different from other container technologies?**

According to me, below, points should be there in your answer:

Docker containers are easy to deploy in a cloud. It can get more applications running on the same hardware than 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 the above explanation is there in your answer.

**Q7.** **What is Docker Swarm?**

You should start this answer by explaining Docker Swarn.

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.

I will also suggest you to include some supported tools:

* Dokku
* Docker Compose
* Docker Machine
* Jenkins

**Q8. What is Dockerfile used for?**

This answer, according to me should begin by explaining the use of Dockerfile.

Docker can build images automatically by reading the instructions from a Dockerfile.

Now I will suggest you to give a small definition of Dockerfle.

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.

*Now, the next set of Docker interview questions will test your experience with Docker.*

**Q9.** **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 filename to use for eg:  
**docker-compose -f docker-compose.json up**

**Q10.** **Tell us how you have used Docker in your past position?**

Explain how you have used Docker to help rapid deployment. Explain how you have scripted Docker and used Docker with other tools like Puppet, Chef or Jenkins.

If you have no past practical experience in Docker and have past 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.

**Q11. How to create Docker container?**

I will suggest you to give a direct answer to this.

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

|  |  |
| --- | --- |
| 1 | docker run -t -i command name |

This command will create and start a container.

You should also add, If you want to check the list of all running container with the status on a host use the below command:

|  |  |
| --- | --- |
| 1 | docker ps -a |

**Q12. How to stop and restart the Docker container?**

In order to stop the Docker container you can use the below command:

|  |  |
| --- | --- |
| 1 | docker stop container ID |

Now to restart the Docker container you can use:

|  |  |
| --- | --- |
| 1 | docker restart container ID |

Q13 How far do Docker containers scale?

Large web deployments like Google and Twitter, and platform providers such as Heroku and dotCloud all run on container technology, at a scale of hundreds of thousands or even millions of containers running in parallel.

Q14. What platforms does Docker run on?

I will start this answer by saying Docker runs on only Linux and Cloud platforms and then I will mention the below vendors of Linux:

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

Cloud:

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

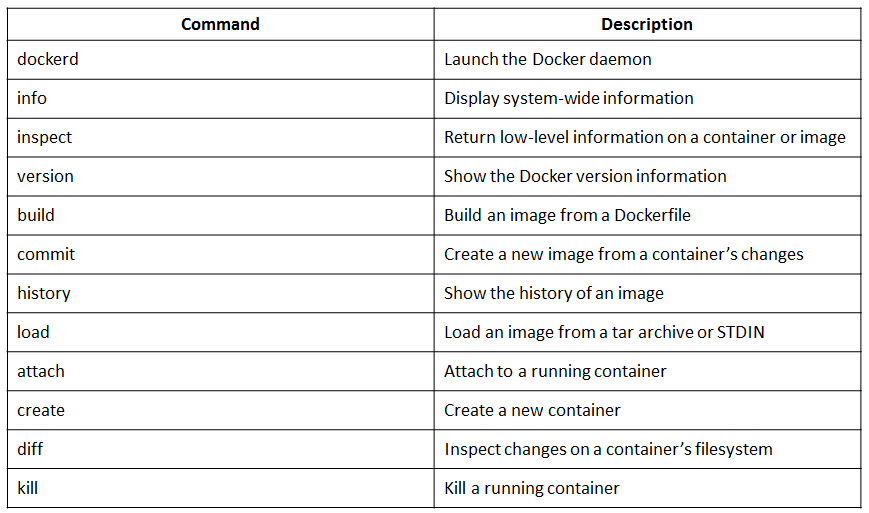
**Note that Docker does not run on Windows or Mac.**

**Q15. Do I lose my data when the Docker container exits?**

You can answer this by saying, no I won’t lose my data when Docker container exits, any data that your application writes to disk gets preserved in its container until you explicitly delete the container. The file system for the container persists even after the container halts.

**Q16. Mention some commonly used Docker command?**

Below are some commonly used Docker commands:



I have included the frequently asked Docker interview questions. If you have more questions in your mind just type it in the comment box below and we will reply you ASAP. Before going for the interview I will suggest you to [***check out this Docker blog series***](https://www.edureka.co/blog/docker-tutorial).

**What is Docker?**

Docker is a platform to run each application isolated and securely. Internally it achieves it by using kernel containerization feature.

**What is the advantage of Docker over hypervisors?**

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

**What is Docker Container?**

[Docker Container](https://tekslate.com/tutorials/docker) is the instantiation of docker image. In other words, it is the run time instance of images. Images are set of files whereas containers are the one who run the image inside isolated.

**Is Container technology new?**

No, it is not. Different variations of containers technology were out there in \*NIX world for a long time.Examples are:-Solaris container (aka [Solaris Zones](http://en.wikipedia.org/wiki/Solaris_Containers))-FreeBSD Jails-AIX Workload Partitions (aka WPARs)-Linux [OpenVZ](http://openvz.org/" \o "OpenVZ" \t "_blank)

**How is Docker different from other container technologies?**

Well, Docker is a quite fresh project. It was created in the Era of Cloud, so a lot of things are done much nicer than in other container technologies. Team behind Docker looks to be full of enthusiasm, which is of course very good. I am not going to list all the features of Docker here, but I will mention those which are important to me.

Docker can run on any infrastructure, you can run docker on your laptop, or you can run it in the cloud.

Docker has a Container HUB, it is a repository of containers which you can download and use. You can even share containers with your applications.

Docker is quite well documented.

**Difference between Docker Image and container?**

Docker container is the runtime instance of docker image.

Docker Image does not have a state, and its state never changes as it is just set of files whereas docker container has its execution state.

**What is the use case for Docker?**

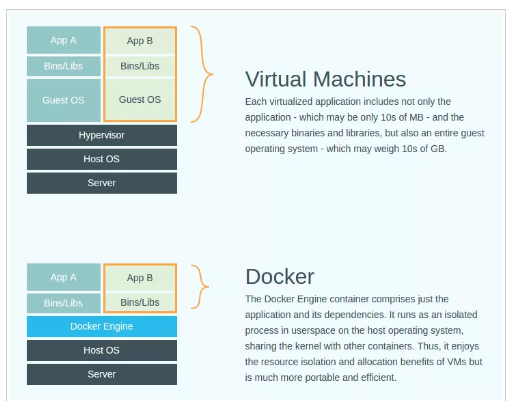
Well, I think, docker is extremely useful in development environments. Especially for testing purposes. You can deploy and re-deploy apps in a blink of an eye.

Also, I believe there are use cases where you can use Docker in production. Imagine you have some Node.js application providing some services on the web. Do you need to run full OS for this?

Eventually, if docker is good or not should be decided on an application basis. For some apps, it can be sufficient, for others not.

**How exactly are containers (Docker in our case) different from hypervisor virtualization (vSphere)? What are the benefits?**

To run an application in a virtualized environment (e.g., vSphere), we first need to create a VM, install an OS inside and only then deploy the application. To run the same application in docker, all you need is to deploy that application in Docker. There is no need for additional OS layer. You just deploy the application with its dependent libraries, docker engine (kernel, etc.) provides the rest.This table from a Docker official website shows it in a quite clear way.



Another benefit of Docker, from my perspective, is speed of deployment. Let’s imagine a scenario:

ACME inc. needs to virtualize application GOOD APP for testing purposes.

Conditions are:

Application should run in an isolated environment.

Application should be available to be redeployed at any moment in a very fast manner.

***Solution 1.***  
In vSphere world what we would usually do, is:

Deploy OS in a VM running on vSphere.

Deploy an application inside OS.

Create a template.

Redeploy the template in case of need. Time of redeployment around 5-10 minutes.

Sounds great! Having app up and running in an hour and then being able to redeploy it in 5 minutes.

***Solution 2.***

-Deploy Docker.

-Deploy the app GOODAPP in container.

-Redeploy the container with an app when needed.

***Benefits:*** No need of deploying full OS for each instance of the application. Deploying a container takes seconds.

**How did you become involved with the Docker project?**

I came across Docker not long after Solomon open sourced it. I knew a bit about LXC and containers (a past life includes working on Solaris Zones and LPAR on IBM hardware too), and so I decided to try it out. I was blown away by how easy it was to use. My prior interactions with containers had left me with the feeling they were complex creatures that needed a lot of tuning and nurturing. Docker just worked out of the box. Once I saw that and then saw the CI/CD-centric workflow that Docker was building on top I was sold.

**Docker is the new craze in virtualization and cloud computing. Why are people so excited about it?**

I think it’s the lightweight nature of Docker combined with the workflow. It’s fast, easy to use and a developer-centric DevOps-ish tool. Its mission is basically: make it easy to package and ship code. Developers want tools that abstract away a lot of the details of that process. They just want to see their code working. That leads to all sorts of conflicts with Sys Admins when code is shipped around and turns out not to work somewhere other than the developer’s environment. Docker turns to work around that by making your code as portable as possible and making that portability user-friendly and simple.

**What, in your opinion, is the most exciting potential use for Docker?**

It’s the build pipeline. I mean I see a lot of folks doing hyper-scaling with containers, indeed you can get a lot of containers on a host, and they are blindingly fast. But that doesn’t excite me as much as people using it to automate their dev-test-build pipeline.

**How is Docker different from standard virtualization?**

Docker is operating system level virtualization. Unlike hypervisor virtualization, where virtual machines run on physical hardware via an intermediation layer (“the hypervisor”), containers instead run user space on top of an operating system’s kernel. That makes them very lightweight and very fast.

### Docker Container Interview Questions And Answers For Experienced

**Do you think open source development has heavily influenced cloud technology development?**

I think open source software is closely tied to cloud computing. Both in terms of the software running in the cloud and the development models that have enabled the cloud. Open source software is cheap, it’s usually low friction both from an efficiency and a licensing perspective.

**How do you think Docker will change virtualization and cloud environments? Do you think cloud technology has a set trajectory, or is there still room for significant change?**

I think there are a lot of workloads that Docker is ideal for, as I mentioned earlier both in the hyper-scale world of many containers and in the dev-test-build use case. I fully expect a lot of companies and vendors to embrace Docker as an alternative form of virtualization on both bare metal and in the cloud.

As for cloud technology’s trajectory. I think we’ve seen a significant change in the last couple of years. I think they’ll be a bunch more before we’re done. The question of OpenStack and whether it will succeed as an IAAS alternative or DIY cloud solution. I think we’ve only touched on the potential for PAAS and there’s a lot of room for growth and development in that space. It’ll also be interesting to see how the capabilities of PAAS products develop and whether they grow to embrace or connect with consumer cloud-based products.

**Can you give us a quick rundown of what we should expect from your Docker presentation at OSCON this year?**

It’s very much a crash course introduction to Docker. It’s aimed at Developers and SysAdmins who want to get started with Docker in a very hands on way. We’ll teach the basics of how to use Docker and how to integrate it into your daily workflow.

**Your bio says “for a real job” you’re the VP of Services for Docker. Do you consider your other open source work a hobby?**

That’s mostly a joke related to my partner. Like a lot of geeks, I’m often on my computer, tapping away at a problem or writing something. My partner jokes that I have two jobs: my “real” job and my open source job. Thankfully over the last few years, at places like Puppet Labs and Docker, I’ve been able to combine my passion with my paycheck.

**Why is Docker the new craze in virtualization and cloud computing?**

It’s OSCON time again, and this year the tech sector is abuzz with talk of cloud infrastructure. One of the more interesting startups is Docker, an ultra-lightweight containerization app that’s brimming with potential

I caught up with the VP of Services for Docker, James Turnbull, who’ll be running a Docker crash course at the con. Besides finding out what Docker is anyway, we discussed the cloud, open source contributing and getting a real job.

Your bio says “for a real job” you’re the VP of Services for Docker. Do you consider your other open source work a hobby?

That’s mostly a joke related to my partner. Like a lot of geeks, I’m often on my computer, tapping away at a problem or writing something. My partner jokes that I have two jobs: my “real” job and my open source job. Thankfully over the last few years, at places like Puppet Labs and Docker, I’ve been able to combine my passion with my paycheck.

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

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

There has already been a lot written about this problem of processes handling signals in containers.

To fix this issue, try the following:

Make sure you’re using the JSON form of CMD and ENTRYPOINT in your [Dockerfile](https://tekslate.com/save-docker/" \t "_blank).

For example use ["program", "arg1", "arg2"] not"program arg1 arg2". Using the string form causes Docker to run your process using bash which doesn’t handle signals properly. Compose always uses the JSON form, so don’t worry if you override the command or entrypoint in your Compose file.

-If you are able, modify the application that you’re running to add an explicit signal handler for SIGTERM.

-Set the stop\_signal to a signal which the application knows how to handle:

-web: build: . stop\_signal: SIGINT

-If you can’t modify the application, wrap the application in a lightweight init system (like s6) or a signal proxy (like dumb-init or tini). Either of these wrappers take care of handling SIGTERM properly.

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

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

### Docker Container Interview Questions

**What’s the difference between up, run, and start?**

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

The docker-compose run command is for running “one-off” or “ad-hoc” tasks. It requires the service name you want to run and only starts containers for services that the running service depends on. Use run to run tests or perform an administrative task such as removing or adding data to a data volume container. The [run command](https://tekslate.com/docker-commands/) acts like docker run -ti in that it opens an interactive terminal to the container and returns an exit status matching the exit status of the process in the container.

The docker-compose start command is useful only to restart containers that were previously created but were stopped. It never creates new containers.

**Can I use json instead of yaml for my Compose file?**

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

docker-compose -f docker-compose.json up

**Should I include my code with COPY/ADD or a volume?**

You can add your code to the image using COPY or ADD directive in a Dockerfile. This is useful if you need to relocate your code along with the Docker image, for example when you’re sending the code to another environment (production, CI, etc).

You should use a volume if you want to make changes to your code and see them reflected immediately, for example when you’re developing code and your server supports hot code reloading or live-reload.

There may be cases where you’ll want to use both. 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.

**Where can I find example compose files?**

There are many examples of Compose files on github.

**Compose documentation**

-Installing Compose

-Get started with Django

-Get started with Rails

-Get started with WordPress

-Command line reference

-Compose file reference

**Are you operationally prepared to manage multiple languages/libraries/repositories?**

Last year, we encountered an organization that developed a modular application while allowing developers to “use what they want” to build individual components. It was a nice concept but a total organizational nightmare — chasing the ideal of modular design without considering the impact of this complexity on their operations.

The organization was then interested in Docker to help facilitate deployments, but we strongly recommended that this organization not use Docker before addressing the root issues. Making it easier to deploy these disparate applications wouldn’t be an antidote to the difficulties of maintaining several different development stacks for long-term maintenance of these apps.

**Do you already have a logging, monitoring, or mature deployment solution?**

Chances are that your application already has a framework for shipping logs and backing up data to the right places at the right times. To implement Docker, you not only need to replicate the logging behavior you expect in your virtual machine environment, but you also need to prepare your compliance or governance team for these changes. New tools are entering the Docker space all the time, but many do not match the stability and maturity of existing solutions. Partial updates, rollbacks, and other common deployment tasks may need to be reengineered to accommodate a containerized deployment.

If it’s not broken, don’t fix it. If you’ve already invested the engineering time required to build a continuous integration/continuous delivery (CI/CD) pipeline, containerizing legacy apps may not be worth the time investment.

### Docker Questions And Answers

**Will cloud automation overtake containerization?**

At AWS Re:Invent last month, Amazon chief technology officer Werner Vogels spent a significant portion of his keynote on AWS Lambda, an automation tool that deploys infrastructure based on your code. While Vogels did mention AWS’ container service, his focus on Lambda implies that he believes dealing with zero infrastructure is preferable to configuring and deploying containers for most developers.

Containers are rapidly gaining popularity in the enterprise, and are sure to be an essential part of many professional CI/CD pipelines. But as technology experts and CTOs, it is our responsibility to challenge new methodologies and services and properly weigh the risks of early adoption. I believe Docker can be extremely effective for organizations that understand the consequences of containerization — but only if you ask the right questions.

**You say that Ansible can take up to 20x longer to provision, but why?**

Docker uses cache to speed up builds significantly. Every command in Dockerfile is building in another docker container, and its results are stored in a separate layer. Layers are built on top of each other.

Docker scans Dockerfile and try to execute each steps one after another, before executing it probes if this layer is already in cache. When a cache is hit, building step is skipped, and from the user perspective is almost instant.

When you build your Dockerfile in a way that the most changing things such as application source code are on the bottom, you will experience instant builds.

You can learn more about caching in docker in this article.

Another way of amazingly fast building docker images is using a good base image – which you specify inFROMcommand, you can then only make necessary changes, not rebuild everything from scratch. This way, the build will be quicker. It’s especially beneficial if you have a host without the cache like Continuous Integration server.

Summing up, building Docker images with Dockerfile is faster than provisioning with Ansible, because of using docker cache and good base images. Moreover, you eliminate provisioning, by using ready to use configured images such stgresus.

$ docker run --name some-postgres -d postgres No installing postgres at all - it's ready to run.

**Also, you mention that docker allows multiple apps to run on one server.**

It depends on your use case. You probably should split different components into separate containers. It will give you more flexibility.

Docker is very lightweight and running containers is cheap, especially if you store them in RAM – it’s possible to spawn new container for every http callback, however, it’s not very practical.

At work, I develop using a set of five different types of containers linked together.

In production some of them are replaced by real machines or even clusters of machine – however, settings on application level don’t change.

Here you can read more about linking containers.

It’s possible because everything is communicating over the network. When you specify links in dockerrun command – docker bridges containers and injects environment variables with information about IPs and ports of linked children into the parent container.

This way, in my app settings file, I can read those values from the environment. In python it would be:

import os VARIABLE = os.environ.get('VARIABLE')

There is a tool which greatly simplifies working with docker containers, linking included. It’s called fig, and you can read more about it here.

Some of the popular Docker interview questions are:

* What is Docker?
* What is the difference between Docker image and Docker container?
* How will you remove an image from Docker?
* How is a Docker container different from a hypervisor?
* Can we write compose file in json file instead of yaml?
* Can we run multiple apps on one server with Docker?
* What are the common use cases of Docker?
* What are the main features of Docker-compose?
* What is the most popular use of Docker?
* What is the role of open source development in the popularity of Docker?
* What is the difference between Docker commands: up, run and start?
* What is Docker Swarm?
* What are the features of Docker Swarm?
* What is a Docker Image?
* What is a Docker Container?
* What is Docker Machine?
* Why do we use Docker Machine?
* How will you create a Container in Docker?
* Do you think Docker is Application-centric or Machine-centric?
* Can we lose our data when a Docker Container exits?
* Can we run more than one process in a Docker container?
* What are the objects created by Docker Cloud in Amazon Web Services (AWS) EC2?
* How will you take backup of Docker container volumes in AWS S3?
* What are the three main steps of Docker Compose?
* What is Pluggable Storage Driver architecture in Docker based containers?
* What is Docker Hub?
* What are the main features of Docker Hub?
* What are the main security concerns with Docker based containers?
* What are the security benefits of using Container based system?
* How can we check the status of a Container in Docker?
* What are the main benefits of using Docker?
* How does Docker simplify Software Development process?
* What is the basic architecture behind Docker?
* What are the popular tasks that you can do with Docker Command line tool?
* What type of applications- Stateless or Stateful are more suitable for Docker Container?
* How can Docker run on different Linux distributions?
* Why do we use Docker on top of a virtual machine?
* How can Docker container share resources?
* What is the difference between Add and Copy command in a Dockerfile?
* What is Docker Entrypoint?
* What is ONBUILD command in Docker?
* What is Build cache in Docker?
* What are the most common instructions in Dockerfile?
* What is the purpose of EXPOSE command in Dockerfile?
* What are the different kinds of namespaces available in a Container?
* How will you monitor Docker in production?
* What are the Cloud platforms that support Docker?
* How can we control the startup order of services in Docker compose?
* Why Docker compose does not wait for a container to be ready before moving on to start next service in dependency order?
* How will you customize Docker compose file for different environments?