**Interview question for docker**

### 1. Difference between an Image, Container, and Engine:

**Image:** Think of it as a snapshot of a file system and the parameters needed to run a piece of software.

**Container:** An instance of an image, including the application and all its dependencies, running in isolation.

**Engine:** The core of Docker, facilitating the creation and management of containers.

Understanding these distinctions sets the foundation for comprehending Docker's functionality.

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**

5. Name and explain the various Docker components.

The three main [Docker components](https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker" \o "Docker components" \t "_blank) 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" \o "Docker images" \t "_blank) 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" \o "Docker Hub" \t "_blank) and Docker Cloud are two public registries available for use by anyone.

### **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

### **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>

### **20. Can you lose data stored in a container?**

Any data stored in a container remains there unless you delete the container.

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.

### **26. Finally, how do you create a Docker swarm?**

Use the following command: docker swarm init –advertise-addr <manager IP>

### 4. Reducing Docker Image Size:

Efficient image size is crucial. Strategies include using a minimal base image, combining RUN commands, cleaning up unnecessary files, and leveraging multi-stage builds to discard unneeded artifacts.

**6. Explaining Docker Components:**

* **Docker Compose:** A tool for defining and running multi-container Docker applications.
* **Docker File:** A script with build instructions for a Docker image.
* **Docker Image:** A self-contained package with everything needed to run an application.
* **Docker Container:** An instantiated and runnable Docker image.

### 7. Real Scenarios of Using Docker:

Docker finds application in microservices architecture, CI/CD pipelines, and the isolation and packaging of applications and dependencies.

**9. Advantages and Disadvantages of Docker:**

* **Advantages:** Portability, efficiency, rapid deployment, and optimized resource utilization.
* **Disadvantages:** Security concerns, learning curve, and potential complexity in larger deployments.
* **9.**
* **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 ngin
* **10.**
* **What is Docker Compose? What can it be used for?**
* Hide answer
* 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.