DOCKER INTERVIEW QUESTIONS

1. **What is the Difference between an Image, Container and Engine?**

- **An image** is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and dependencies.
- **A container** is an instance of an image that runs as a process on the host machine. It provides an isolated and portable environment for running applications with consistent behavior across different environments.
- **The Docker engine** is the runtime environment that allows the creation and management of containers. It is responsible for building, running, and managing Docker containers.

2) **What is the Difference between the Docker command COPY vs ADD?**

- The **COPY** command is used to copy files or directories from the host machine to the container. It works with local files or directories and has a straightforward syntax.
- The **ADD** command is similar to COPY but has some additional features. It can also copy remote URLs and automatically extract compressed files, such as tarballs, during the copy process. However, because of its complexity, it is recommended to use COPY for most cases unless you specifically need the extra functionality provided by ADD.

3) **What is the Difference between the Docker command CMD vs RUN?**

- The **RUN** command is used to execute commands during the build process of an image. It allows you to install dependencies, run scripts, or perform any other actions necessary to prepare the image.
- The **CMD** command is used to specify the default command to run when a container is started from an image. It provides the default behavior for the container but can be overridden with command-line arguments when starting the container.

4) **How will you reduce the size of the Docker image?**

To reduce the size of a Docker image, you can consider the following practices:

- Use a minimal base image.
- Minimize the number of layers in the image by consolidating related operations into a single command.
- Remove unnecessary files and dependencies after they are no longer needed.
- Use multi-stage builds to separate build dependencies from the final image.
- Compress and optimize files within the image.
- Use Alpine Linux or other lightweight distributions as the base image.
- Utilize Docker image layer caching and build cache.

5) **Why and when to use Docker?**

Docker offers several advantages, making it beneficial in various scenarios:

- **Achieving consistency**: Docker allows you to package applications with all their dependencies, ensuring consistency across different environments.
- **Simplifying deployment**: Docker provides a lightweight and portable containerization platform, making it easier to deploy applications on different servers or cloud environments.
- **Scalability and resource efficiency**: Docker enables easy scaling of applications by running multiple containers on a single host, optimizing resource utilization.
- **Isolation**: Containers provide process isolation, ensuring that applications do not interfere with each other or the underlying host system.
- **DevOps integration**: Docker facilitates the adoption of DevOps practices by enabling consistent development and deployment environments and supporting automated workflows.
- **Faster software delivery**: Docker's container-based approach accelerates the software development and delivery process, promoting faster iteration cycles and reducing time to market.

6) **Explain the Docker components and how they interact with each other.**

Docker consists of the following key components:

- **Docker daemon**: The Docker daemon, or Docker engine, is a background process responsible for building, running, and managing

Docker containers. It receives commands from the Docker client and manages container images, networks, and volumes.

- **Docker client:** The Docker client is the primary interface for users to interact with the Docker daemon. It sends commands to the daemon and receives information and output from it.
- **Docker images:** Docker images are read-only templates used to create Docker containers. They include the application code, runtime, system tools, libraries, and dependencies required to run the application.
- **Docker containers**: Docker containers are instances of Docker images that run as isolated processes on the host machine. Each container has its own file system, network, and process space, providing a lightweight and consistent runtime environment.
- **Docker registry**: Docker registry is a centralized repository for storing and distributing Docker images. It allows users to share and download images from the registry.
- **Docker Compose**: Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to specify the services, networks, and volumes required for the application’s containers to communicate and work together.

7) **Explain the terminology: Docker Compose, Docker File, Docker Image, Docker Container?**

- Docker Compose: Docker Compose is a tool that allows you to define and manage multi-container Docker applications. It uses a YAML file to specify the services, networks, and volumes required for the application’s containers.
- Dockerfile: A Dockerfile is a text file that contains instructions for building a Docker image. It specifies the base image, dependencies, configuration settings, and commands required to create the image.
- Docker Image: A Docker image is a read-only template that contains the application code, runtime, system tools, libraries, and dependencies required to run the application within a container.
- Docker Container: A Docker container is a running instance of a Docker image. It is an isolated and lightweight runtime environment that encapsulates the application and its dependencies, allowing it to run consistently across different environments.

8) **In what real scenarios have you used Docker?**

Docker is widely used in various scenarios, including:

- Microservices architecture: Docker enables the deployment and management of microservices-based applications, where each microservice runs in its own container.
- Continuous integration and delivery (CI/CD): Docker simplifies the packaging and deployment of applications, making it easier to implement CI/CD pipelines for automated testing, building, and deployment.
- Development environments: Docker provides a consistent and reproducible environment for development teams, ensuring that the application runs the same way on developers' machines as it does in production.
- Scaling applications: Docker's containerization allows applications to be easily scaled by running multiple instances of the same container on different hosts or clusters.
- Hybrid and multi-cloud deployments: Docker facilitates the portability of applications across different cloud providers and on-premises environments, enabling hybrid and multi-cloud deployments.

9) **Docker vs Hypervisor?**

Docker and hypervisors serve similar purposes but have different approaches:

- **Docker**: Docker utilizes containerization, which operates at the operating system (OS) level. It allows multiple containers to run on a single host, sharing the host's OS kernel. This approach provides lightweight and efficient isolation, making it suitable for deploying microservices and application components.
- **Hypervisor**: Hypervisors, such as VMware and Hyper-V, provide hardware-level virtualization. They create virtual machines (VMs) that run on a host machine, with each VM having its own OS and resources. Hypervisors offer strong isolation but consume more resources compared to Docker containers.

10) **What are the advantages and disadvantages of using docker?**

Advantages of using Docker:

- Portability: Docker allows applications to run consistently across different environments, making them highly portable.
- Scalability: Docker’s containerization enables easy scaling of applications by running multiple containers on a single host or across a cluster of hosts.
- Efficiency: Docker containers are lightweight and share the host OS kernel, resulting in efficient resource utilization and faster startup times.
- Isolation: Containers provide process-level isolation, ensuring that applications do not interfere with each other or the underlying host system.

11) What is a Docker namespace?

A Docker namespace is a feature that provides process isolation and resource separation for containers. Namespaces allow each container to have its own view of system resources, such as the process tree, network interfaces, file systems, and user IDs. Docker uses various namespaces, such as the PID namespace, network namespace, mount namespace, and user namespace, to ensure that containers have their isolated environments and cannot access or interfere with resources outside their namespace.

12) What is a Docker registry?

A Docker registry is a centralized repository that stores Docker images. It acts as a distribution hub for Docker images, allowing users to share, download, and manage container images. Docker Hub is the default public Docker registry, but private registries can also be set up for internal use within organizations.

13) What is an entry point?

In Docker, the entry point is the command or script that is executed when a container is started from an image. It specifies the default executable or action for the container. The entry point can be set in the Dockerfile or overridden during container runtime by passing command-line arguments.

14) How to implement CI/CD in Docker?

Implementing CI/CD (Continuous Integration/Continuous Deployment) with Docker involves the following steps:

- Set up a CI/CD pipeline using a CI/CD tool like Jenkins, GitLab CI/CD, or CircleCI.
- Configure the pipeline to listen for changes in the source code repository.
- Use a Dockerfile to define the build environment and dependencies for the application.
- Build a Docker image in the CI/CD pipeline, typically using a Docker build command.
- Push the built Docker image to a Docker registry.
- Deploy the Docker image to the desired environment, such as development, staging, or production, using tools like Docker Compose or Kubernetes.

15) Will data on the container be lost when the docker container exits?

By default, data within a container will be lost when the container exits. Containers are designed to be ephemeral and disposable. However, you can persist data by mounting volumes in the container or using Docker's data persistence mechanisms such as bind mounts or Docker volumes. These techniques allow data to be stored outside the container and persist even after the container exits.

16) What is a Docker swarm?

Docker Swarm is a native clustering and orchestration solution provided by Docker. It allows you to create and manage a swarm of Docker nodes, forming a cluster. Swarm enables you to deploy and scale applications across multiple Docker hosts, providing high availability, load balancing, and automatic service discovery.

17) What are the docker commands for the following:
— view running containers

```
docker ps
```

— command to run the container under a specific name

```
docker run — — name <container>
```

— command to export a docker

```
docker export <container_id> > <file_name.tar>
```

— command to import an already existing docker image

```
docker import <file_name.tar>
```

— commands to delete a container

```
docker rm <container_id>
```

— command to remove all stopped containers, unused networks, build caches, and dangling images?

```
docker system prune
```

18) What are the common docker practices to reduce the size of Docker Image?

- Use a minimal base image, such as Alpine Linux.
- Minimize the number of layers in the image by combining related operations.
- Remove unnecessary files and dependencies after they are no longer needed.
- Utilize multi-stage builds to separate build dependencies from the final image.
- Optimize and compress files within the image, if applicable.
- Use .dockerignore files to exclude unnecessary files and directories during the build process.
- Leverage Docker image layer caching and build cache for faster builds.
- Consider using smaller, specialized images for different components of your application.