Docker Important interview Question

Docker Interview

Docker is a good topic to ask in DevOps Engineer Interviews, mostly for freshers. One must surely try these questions in order to be better in Docker

Questions

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

An image is a pre-configured file that contains all the necessary dependencies and settings to run a specific application or service. It is used to create and run containers.

A container is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files. Containers run within an Operating System, and share the host kernel.

An engine is a software that is responsible for creating and managing containers. It is the component that handles the low-level details of container creation, networking, storage, and other operations. Popular examples of container engines include Docker.

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

The COPY and ADD commands in Docker are used to add files and directories to a container image.

The COPY command is used to copy local files and directories to the container file system. It is a more basic command and only supports local file and directory paths.

The ADD command also copies local files and directories to the container file system, but it has some additional features. It can automatically decompress files and supports copying files from remote URLs.

In general, it is recommended to use COPY command for most cases as it's more secure and predictable, as ADD command has some features that may increase the chance of introducing security vulnerabilities.

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

The CMD and RUN commands in Docker are used to execute commands in a container.

The RUN command is used to execute commands during the image build process. It runs the command and commits the result, creating a new layer in the image. This allows you to install software, make configuration changes, and perform other tasks when building an image.

The CMD command is used to set the default command that will be executed when a container is run from the image. It is used to specify the command that will be run when the container starts up, and it can be overridden when starting a container. Unlike RUN, CMD doesn't create a new layer on the image, but you can use multiple CMD commands to set defaults for an executable.

In summary, the RUN command is used during the image building process to execute commands and commit the results, while CMD command sets the command that will be run when the container starts up.

4. How Will you reduce the size of the Docker image?

There are several ways to reduce the size of a Docker image:

1. Multi-stage builds: Multi-stage builds allow you to use multiple FROM statements in your Dockerfile. This allows you to use one image as a builder image and then copy only the necessary files to a smaller image.
2. Use official images: Official images are images that are maintained by the upstream software maintainers. These images are usually smaller in size and more secure than images built by other parties.
3. Remove unnecessary files: Remove unnecessary files and dependencies from the image by using the RUN apt-get autoremove, RUN apt-get clean and RUN rm commands in your Dockerfile.

4. Use Alpine Linux: Alpine Linux is a lightweight Linux distribution that is popular for creating small Docker images. It is smaller than most other Linux distributions and has a smaller attack surface.
5. Use Squash: Squash is a technique that allows you to combine all the layers of an image into a single layer. This can significantly reduce the size of an image.
6. Use .dockerignore file: A .dockerignore file allows you to specify files and directories that should be excluded from the build context sent to the Docker daemon. This helps to exclude unnecessary files from the build context, which in turn reduces the size of the image.

It is important to note that reducing the size of a Docker image is not only beneficial for performance and storage, but also for security as small images have a smaller attack surface.

5. Why and when to use Docker?

Docker is a popular tool for creating, deploying, and running applications in containers. Containers are isolated environments that include all the dependencies and configurations required to run an application, making it easy to run the same application consistently across different environments.

Here are a few reasons why you might want to use Docker:

1. **Consistency**: Docker containers provide a consistent environment for running applications, regardless of the host system. This makes it easy to move applications between development, testing, and production environments.
2. **Isolation:** Containers isolate applications from one another and from the host system, providing an additional layer of security.
3. **Portability**: Containers are lightweight and portable, making it easy to move them between different systems and environments.

4. **Scalability**: With Docker, it is easy to scale applications up and down as needed, by running multiple instances of a container.
5. **Ease of use**: Docker provides a simple and consistent interface for creating, deploying, and running containers.

Docker is used in various scenarios, such as:

- Developing, testing and deploying microservices.
- Automating the deployment of complex applications.
- Simplifying the configuration management of systems.
- Building and testing software in a consistent and reliable environment.
- Managing and scaling applications in a cloud-based infrastructure.
- Running applications in a lightweight, portable and consistent environment.

In short, Docker can be used in any scenario where you need to run a consistent, isolated environment for your applications, and where you want the ability to easily move those applications between different systems and environments.

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

Docker consists of several components that work together to create, deploy, and manage containers. These components include:

1. **The Docker daemon**: This is the background process that manages containers, images, networks, and storage on a host. It listens for API requests and performs the necessary operations.
2. **The Docker client**: This is the command-line interface (CLI) that allows users to interact with the Docker daemon. It communicates with the daemon via REST API calls to perform various operations, such as creating and managing containers.

3. **The Docker registry**: This is a service that stores and distributes Docker images. It is used to share images with others and to retrieve images for use on a host. The most popular registry is Docker Hub, which is maintained by Docker, Inc.
4. **The Docker engine**: The Docker engine is the layer between the host and the container. This is the component that provides the low-level functionality for creating, starting, and stopping containers. It is built on top of the Docker daemon.
5. **The Docker image:** An image is a pre-configured file that contains all the necessary dependencies and settings to run a specific application or service. It is used to create and run containers.

The Docker client and daemon interact together, the client sends commands to the daemon, which performs the requested operations. The client and daemon can run on the same host, or the daemon can be running on a remote host and the client can communicate with it via the Docker API.

Users interact with the Docker client to create, start, and stop containers, pull images from a registry and push images to a registry. The Docker daemon uses the images to create and run the containers. The Docker registry is used to store and distribute images. The Docker engine is the layer between the host and the container, it provides the low-level functionality for creating, starting and stopping containers.

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

1. **Docker Compose**: Docker Compose is a tool for defining and running multi-container applications. It uses a YAML file to configure the application's services, networks, and volumes. With Compose, you can define your application's services, networks, and volumes in a single file, and then start and stop your application using a single command.

2. **Dockerfile**: A Dockerfile is a script that contains instructions for building a Docker image. It is a simple text file that contains commands, such as FROM, RUN, COPY, EXPOSE, ENV, etc. These commands are executed by the Docker daemon during the build process to create an image.
3. **Docker Image**: An image is a pre-configured file that contains all the necessary dependencies and settings to run a specific application or service. It is used to create and run containers. An image is a read-only template that can be used to create new containers.
4. **Docker Container**: A container is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files. Containers run within an Operating System, and share the host kernel. Once an image is run, it becomes a container. The container is a running instance of an image, and it can be started, stopped, moved, or deleted.

In summary, Docker Compose is a tool to define and run multi-container applications, a Dockerfile is a script to build a Docker image, an image is a pre-configured file to run a specific application or service, and a container is a lightweight, stand-alone, executable package that runs an image.

8. Docker vs Hypervisor?

Docker and hypervisors are both used for virtualization, but they work in different ways and have different use cases.

A hypervisor is a software or hardware-based solution that creates and runs virtual machines (VMs). Each VM runs its own operating system and has its own resources, such as CPU, memory, and storage. Hypervisors abstract the underlying hardware and provide a layer of isolation between the VMs and the host. Examples of hypervisors include VMware, Hyper-V, and VirtualBox.

Docker, on the other hand, uses a different virtualization approach called containerization. Containers are a lightweight alternative to VMs, they share the host operating system kernel and therefore don't require a separate operating system for each container. Instead, containers use the host's resources and abstract the application and its dependencies into a single package.

Docker allows for more efficient use of resources, as containers don't require a separate operating system for each instance, and allows for faster startup times and smaller disk space usage. Hypervisors, on the other hand, provide a higher level of isolation between instances and are generally considered more secure.

In general, Docker is best suited for applications that have a high degree of dependency on the host system and that have a high degree of similarity between instances, while Hypervisors are best suited for applications that have a high degree of isolation requirements and that can run on multiple instances with different operating systems.

9. What are the advantages and disadvantages of using docker?

Advantages:

1. Consistency: Docker provides a consistent environment for running applications, regardless of the host system. This makes it easy to move applications between development, testing, and production environments.
2. Isolation: Containers isolate applications from one another and from the host system, providing an additional layer of security.
3. Portability: Containers are lightweight and portable, making it easy to move them between different systems and environments.
4. Scalability: With Docker, it is easy to scale applications up and down as needed, by running multiple instances of a container.

5. Ease of use: Docker provides a simple and consistent interface for creating, deploying, and running containers.
6. Resource Efficiency: Docker utilizes the host resources more efficiently as it doesn't require a full-fledged operating system for each container, thus reducing the footprint of the application.

Disadvantages:

1. Security: Docker containers share the host kernel and therefore, a vulnerability in the host can compromise all the containers running on that host.
2. Performance overhead: Docker introduces a small amount of overhead due to the additional abstraction layer.
3. Complexity: Docker can be complex to use, especially when running multiple containers or deploying large applications.
4. Limited support for GUI-based applications: Docker is primarily designed to run command-line applications, and it may not be as well-suited for GUI-based applications.
5. Limited Windows support: Docker has limited support for Windows and Windows containers may have some limitations compared to Linux containers.

10. What is a Docker namespace?

A Docker namespace is a feature of the Docker engine that allows for the isolation of resources within a single host. Namespaces provide a way to divide a single host into multiple isolated environments, called namespaces. Each namespace has its own set of resources, such as network interfaces, process IDs, and filesystems.

There are several different types of namespaces in Docker:

1. pid namespace: isolates the process ID space, which means that processes in one namespace cannot see or signal processes in another namespace.
2. net namespace: isolates network interfaces, IP addresses, and routing tables, which means that containers in one namespace cannot communicate with containers in another namespace using their IP addresses.
3. ipc namespace: isolates inter-process communication resources, such as System V semaphores and message queues, which means that processes in one namespace cannot communicate with processes in another namespace using these resources.
4. mnt namespace: isolates the filesystem, which means that processes in one namespace cannot access files in another namespace.
5. user namespace: isolates the user and group IDs, which means that processes in one namespace cannot access resources that are owned by users or groups in another namespace.

Docker uses these namespaces to provide isolation between containers running on a single host, which allows for multiple containers to run on the same host without interfering with each other. Namespaces also enable the use of multiple instances of the same service on a single host, and also provide a way to limit the resources that a container can use, ensuring stability, security and control on the host.

11. What is a Docker registry?

A Docker registry is a service for storing and distributing Docker images. It is used to share images with others and to retrieve images for use on a host. The most popular registry is Docker Hub, which is maintained by Docker, Inc.A registry can be either public or private. Public registries, like Docker Hub, allow anyone to pull images, while private registries are only accessible to a specific group of users.Registries also provide versioning and tagging functionality, which allows users to keep track of different versions of an image and to easily switch between them.

12. What is an entry point?

In the context of Docker, an entry point is a command that is executed when a container is started from an image. It is used to specify the command that will be run when the container starts up.An entry point can be specified in a Dockerfile using the ENTRYPOINT instruction. The ENTRYPOINT instruction sets the command that will be run when the container starts, and it can be overridden when starting a container.

For example, a Dockerfile might have the following ENTRYPOINT instruction:

```
ENTRYPOINT ["/usr/local/bin/my-application"]
```

13. How to implement CI/CD in Docker?

CI/CD (Continuous Integration and Continuous Deployment) in Docker can be implemented using the following steps:

1. **Create a Dockerfile**: A Dockerfile is a script that contains instructions for building a Docker image. It is a simple text file that contains commands such as FROM, RUN, COPY, EXPOSE, ENV, etc. These commands are executed by the Docker daemon during the build process to create an image.
2. **Create a build pipeline**: Set up a build pipeline that automatically builds the image from the Dockerfile whenever there is a change in the source code. This can be done using tools like Jenkins, CircleCI, TravisCI, etc.
3. **Automate testing**: Set up automated testing for the image, such as unit tests, integration tests, and acceptance tests, to ensure that the image is working as expected.
4. **Push the image to a registry**: Once the image is built and tested, it can be pushed to a Docker registry, such as Docker Hub, so that it can be easily distributed to other systems.
5. **Deploy the image to production:** Use a container orchestration tool like Kubernetes, Docker Swarm, or Amazon ECS to deploy the image to a production environment.

6. **Monitor and scale**: Monitor the deployed image and scale it as needed to handle increased

14. Will data on the container be lost when the docker container exits?

By default, data stored within a container is not persistent and will be lost when the container exits or is removed. However, there are ways to make the data persistent and to persist data between container restarts:

1. Volume mounts
2. Data Volumes
3. Volume Plugins
4. Bind Mounts

15. What is a Docker swarm?

A Docker swarm is a feature of the Docker engine that allows for the orchestration of multiple Docker containers across multiple hosts. It provides native clustering capabilities for Docker, allowing containers to be scheduled and managed across multiple hosts.

A swarm consists of a manager node, which is responsible for managing the swarm, and worker nodes, which are responsible for running the containers. The manager node handles tasks such as scheduling containers, maintaining the desired state of the swarm, and providing a centralized point of management. The worker nodes run the containers and communicate with the manager node to receive tasks and report their status.

16. 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 <name> <image>
- **command to export a docker:** docker export <container_id or name> > <filename>.tar

- **command to import an already existing docker image:**
- `docker import <filename>.tar <repository>:<tag>`
- **commands to delete a container:** `docker rm <container_name or ID>`
- **command to remove all stopped containers, unused networks, build caches, and dangling images:** `docker system prune -a`

Thank you for reading!