Docker Important Interview Questions:

1. **What is the Difference between an Image, Container and Engine?**
   a. **Docker Engine:** Docker Engine is the core component of the Docker platform which provides the runtime environment for building and running the containers. It is a lightweight, open-source containerization technology that enables developers to package applications, dependencies and libraries into containers. It uses CLI client docker.

   b. **Docker Image:** Docker Image contains application code, libraries, tools, dependencies and other files needed to run the application on system. It generates by using Dockerfiles & Dockerfiles contains the executable commands to create Docker Images. It acts as a set of instructions to build the container like to say *a template*. It also acts as the starting point when using docker. It is similar to a snapshot in VM machine environments. It has multiple layers, each originates from previous one. It is immutable.

   c. **Docker Container:** Docker Container is a virtualized runtime environment used in application deployment. It is used to create, run and deploy applications that are isolated from the underlaying hardware. It can easily be moved between different environments.

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

   o **COPY** and **ADD** command are both Dockerfile instructions that has same purpose i.e. to copy files from a specific location into a Docker image.
   o **COPY** takes in a source and destination. It only let us copy in a local or directory from our host into the Docker image itself whereas **ADD** does the same work but in addition it also supports 2 other sources.
   1. A URL instead of a local file/directory.
   2. Extract tar from the source directory into the destination.

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

   In Docker, **RUN** is used to execute a command during the image build process, while **CMD** specifies the default command that should be executed when a container is started from the image.

   RUN is used to execute any command, including installing software or updating system configurations. The changes made during **RUN** are stored in the image, and are used whenever a container is created from that image.

   CMD, on the other hand, provides a default command for an image, but it can be overridden at runtime when the container is started. If multiple **CMD** instructions are specified in a Dockerfile, only the last one will be used.

4. **How will you reduce the size of the Docker image? Or What are the common docker practices to reduce the size of Docker Image?**

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

   o **Use a smaller base image:** Select a lightweight base image such as Alpine Linux, instead of using a full-fledged operating system image.
   o **Use multi-stage builds:** Use multiple Dockerfiles to build the application in stages, where each stage builds on top of the previous one.

- **Minimize the number of layers:** Each command in a Dockerfile creates a new layer, which adds to the size of the image. Try to combine commands whenever possible to reduce the number of layers.
- **Remove unused files:** Remove any unnecessary files, packages, or libraries from the image by using the "RUN apt-get clean" and "RUN rm -rf /var/lib/apt/lists/*" commands.
- **Use compression:** Compress large files and directories to reduce their size by using gzip or other compression algorithms.
- **Keep the image up-to-date:** Regularly updating the image to the latest version can reduce its size, as newer images often include bug fixes and optimizations.
- **Using Docker ignore command:** Docker can ignore the files present in the working directory if configured in the .dockerignore file.
- **Keeping application data elsewhere:** Highly recommended to use the volume feature of the container runtimes to keep image separate from the data.

5. Why and when to use Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers for improved efficiency, reliability, and portability. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Docker is used for several reasons:

- **Isolation:** Docker containers provide a clean and isolated environment for applications, which reduces the risk of conflicts between different applications or dependencies.
- **Portability:** Docker containers can run on any system that has Docker installed, making it easier to move an application between development, testing, and production environments.
- **Scalability:** Docker containers can be easily scaled up or down as needed, making it easier to handle changes in load and demand.
- **Cost savings:** Docker containers can be run on shared infrastructure, reducing the need for dedicated resources and resulting in cost savings.
- **Continuous integration and delivery (CI/CD):** Docker can be used to automate the build and deployment process, making it easier to release new features and bug fixes faster.
- **Ease of deployment:** Docker containers can be deployed quickly and easily, reducing the time and effort required to get an application up and running.

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

Docker follows Client-Server architecture, which includes the three main components: Docker Client, Docker Host & Docker Registry.

- **Docker Client:** Docker client uses commands and REST APIs to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.
- **Docker Host:** Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.
- **Docker Registry:** Docker Registry manages and stores the Docker images. There are two types of registries in the Docker -
  Pubic Registry - Public Registry is also called as Docker hub.
  Private Registry - It is used to share images within the enterprise.

Docker is a platform that consists of a Docker Daemon (core component), a CLI, Docker Images (executable packages), Docker Containers (running instances of images), and a Docker Hub registry, that interact with each other to easily build, package, and deploy applications in containers, managing the application lifecycle from development to production.

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

   Docker Compose: Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to define the application's services, networks, and volumes in a single file called a “*docker-compose.yml*” file, and then start and stop all services using a single “*docker-compose*” command. With Docker Compose, you can easily configure and run applications consisting of multiple containers, reducing the time and effort required to set up complex applications.
   Docker File: A Dockerfile is a text file that contains instructions to build a Docker image. It specifies how to set up the environment and configure the application in a container. The file contains a list of commands that are executed in order to create a Docker image. The resulting image can be run as a container to host the application.
   Docker Images: Docker images are the read-only binary templates used to create Docker Containers. It uses a private container registry to share container images within the enterprise and also uses public container registry to share container images within the whole world.
   Docker Containers: Containers are used to hold the entire package that is needed to run the application. The advantage of containers is that it requires very less resources..

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

   Docker are mostly used for creating a container used to run anything from a small microservice or software process to a larger application. Inside a container are all the necessary executables, binary code, libraries, and configuration files.

   Containers are a streamlined way to build, test, deploy, and redeploy applications on multiple environments from a developer’s local laptop to even the cloud. Dockerfiles and Docker compose files are used to build images and containers.

   Benefits are numerous: Less overhead, Increased portability, More consistent operation, Greater efficiency, Better application development.

9. **Docker vs Hypervisor?**

   Hypervisor: Hypervisors are of two types – the bare metal works directly on the hardware while type two hypervisor works on top of the operating system. Hypervisors allows the users to generate multiple instances of complete operating systems. They need dedicated resources for any particular instance among the shared hardware which the hypervisor allocates during boot. A hypervisor might consume up to a minute to boot the OS and get up and running. Hypervisors are OS agnostic. They can run across Windows, Mac, and Linux.

   Docker: Docker works on the host kernel itself. Dockers can run multiple applications or multiple instances of a single application. It does this with containers. Dockers do not need any dedicated resources. One can create as many containers as needed. Based on the application requirement and availability of processing power, the Docker provides it to the containers. Docker can create containers in seconds, and users can get started in no time. Dockers, on the other hand, are limited to Linux only.

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

Advantages:

1. Light weight because it does not require any resource pre-allocation (RAM). Whenever it needs resources it acquires them from host OS.
2. Run on physical/virtual hardware/cloud.
3. Continuous integration efficiency – Docker enables you to build a container image and use that same image across every step of the deployment process.
4. Re-use the image.
5. Simplicity, faster configuration & rapid Deployment.
6. Cost saving.

Disadvantages:

1. Docker is not good for application that requires rich GUI
2. It is difficult to manage large amount of containers
3. Docker does not provide cross-platform compatibility means if an application is designed to run in a Docker container on windows, then it cannot run on Linux Docker container
4. Docker is suitable when development OS and testing OS are same
5. It does not provide any solution for data backup and recovery

11. **What is a Docker namespace?**

Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker namespace types:

- ➢ **Process ID (pid namespace):** Process isolation
- ➢ **Mount (mnt namespace):** managing filesystem mount points
- ➢ **Interprocess communication (ipc namespace):** managing access to IPC resources
- ➢ **Network (net namespace):** managing network interfaces
- ➢ **Unix timesharing system (uts namespace):** Isolates kernel and version identifiers

12. **What is a Docker registry?**

- ➢ A Docker registry is a system for storing and distributing Docker images with specific names. There may be several versions of the same image, each with its own set of tags.
- ➢ A Docker registry is separated into Docker repositories, each of which holds all image modifications. The registry may be used by Docker users to fetch images locally and to push new images to the registry (given adequate access permissions when applicable).
- ➢ The registry is a server-side application that stores and distributes Docker images. It is stateless and extremely scalable.
- ➢ There are two types of registries in the Docker -
  Pubic Registry - Public Registry is also called as Docker hub.
  Private Registry - It is used to share images within the enterprise.

13. What is an entry point?

- **ENTRYPOINT** is one of the many instructions that we write in a Dockerfile.
- ENTRYPOINT instruction is used to configure the executables that will always run after the container is initiated.
- We can mention a script to run as soon as the container is started. Note that the ENTRYPOINT commands cannot be overridden or ignored, even when we run the container with command line arguments.
- Docker ENTRYPOINT instructions can be written in both shell and exec forms such as
  - Shell form: ENTRYPOINT node app.js
  - Exec form: ENTRYPOINT ["node", "app.js"]

14. How to implement CI/CD in Docker?

CI/CD implementation in Docker can be done using following steps:

- Create a Dockerfile, a script having instructions for building Docker images. It includes commands such as FROM, RUN, COPY, EXPOSE, ENV, etc. The Docker Daemon is responsible for executing this commands during the build process for image execution.
- Build the Pipeline which automates the build image process from the Dockerfile whenever there is any change in code. It can be done by using Jenkins, CircleCI.
- Setting automatic testing like unit tests, integration tests, acceptance tests to ensure image is working or not.
- Once the image is build and tested, it can be pushed to the docker registry say DockerHub for further distribution to other systems.
- For deploying image to production environment, we can use Docker swarm, Kubernetes (container orchestration tools).
- Monitor deployed image and scale it as needed to handle increased.

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

Not at all! The data cannot be lost if the container is stopped and we can view this docker container by using “docker ps -a” command. We can restart this container also. The data of a container remains in it until and unless you delete the container.

If Docker runs on Linux, tmpfs mount is used to store files in the host's system memory. If Docker runs on Windows, named pipe is used to store files in the host's system memory. Volumes are the most preferred way to store container data as they provide efficient performance and are isolated from the other functionalities of the Docker host.

16. What is a Docker swarm?

A **Docker Swarm** is a group of either physical or virtual machines that are running the Docker application and that have been configured to join together in a cluster.

The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster are referred to as nodes.

There are typically several **worker nodes** and at least one **manager node** that is responsible for handling the worker nodes' resources efficiently and ensuring that the cluster operates efficiently.

As if the Manager Node gets down or become unavailable due to some reason, the leadership will transferred to another Node

One of the key benefits associated with the operation of a docker swarm is the high level of availability offered for applications.

Docker Swarm lets you connect containers to multiple hosts similar to Kubernetes.

Docker Swarm has two types of services: replicated and global.

17. **What are the docker commands for the following:**
    a. **view running containers**

       "docker ps" or "docker container ls"

    b. **command to run the container under a specific name**

       docker run –name <container-name> <docker-image>

    c. **command to export a docker**

       docker export <container-ID or name> <filename.tar>

    d. **command to import an already existing docker image**

       docker import <filename.tar> <repository> <tag>

    e. **commands to delete a container**

       docker rm <container-ID or name>

    f. **command to remove all stopped containers, unused networks, build caches, and dangling images?**

       Docker system prune **-**a