Day 21 Task: Docker Important interview Questions.

Questions

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

→ Docker Engine: - Docker Engine is a software that lets you create and run containers, which are isolated environments for your applications. using Docker, we can bundle up all application dependencies inside a container which is then run on Docker Engine.

Docker Engine acts as a client-server application and

A server with a long-running daemon process called as dockerd. Docker Engine uses
A command line interface (CLI) client docker

→ Docker Images: - **Docker** Image is an executable package that includes everything needed to run an application. This image informs how a container should instantiate, determining which software components will run and how. The Docker images are created using the Dockerfiles.The Dockerfiles contains all the executables commands to create a docker image.

→ **Docker Container** is a virtual environment that bundles application code with all the dependencies required to run the application. The application runs quickly and reliably from one computing environment to another.

Also, The Docker container is a runtime instance of a Docker image.DOckerContainer is created using the Docker Image.

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

→ **COPY** and **ADD** are both Dockerfile commands that serve a similar purpose. Both are used to copy the data from one location to docker images.

The COPY Command is used to copy in a local or directory from your host into the Docker image itself.

ADD Command also behave the same but in addition, it also supports 2 others sources.

1. A URL instead of a local file/directory.
2. Extract tar from the source directory into the destination.

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

→ RUN: - RUN is command used in Dockerfile to create a Docker Image. The RUN Command is capable of Run the command or install needed dependencies also it creates an intermediate layer of current image.

There are multiple RUN Commands are used in a Dockerfile and all commands are executed one after another.

→ CMD: - CMD is command used in Dockerfile to create a Docker Image. The CMD Command is capable of Execute the command also this command is not capable of install needed dependencies.

There are multiple Only One Commands are used in a Dockerfile ,The Reason is CMD command overrides all previous one CMD commands are execute lats CMD command.

- **How Will you reduce the size of the Docker image?**

→ Docker images are a core component in our development and production lifecycles. Having a large image can make every step of the process slow and tedious. The large size is affected when we spin up new instances that run our code. Reducing the size of the image can have benefits both for developers and your users.

There are some methods to reducing the Docker image size: -

1. Using minimal base images - choosing the right base image with a minimal OS footprint.
2. Multistage builds - we use different Dockerfiles for building and packaging the application code.
3. Minimizing the number of layers (more use of RUN Command)
4. Understanding caching - add the lines which are used for installing dependencies & packages earlier inside the Dockerfile – before the COPY commands
5. Using Docker ignore Command - Docker can ignore the files present in the working directory if configured in the .dockerignore file.
6. Keeping application data elsewhere - It’s highly recommended to use the volume feature of the container runtimes to keep the image separate from the data.

- **Why and when to use Docker?**

→ Containerization is a technology that is enjoying huge popularity in the tech world – and Docker is a renowned player of it.With Docker, your development environment will be exactly the same as your production environment, and exactly the same as everyone else's development environment, alleviating the problem of "it's broken on my machine!"

If you wanted to add another server to your cluster, you wouldn't have to worry about reconfiguring that server and reinstalling all the dependencies you need. Once you build a container, you can share the container file with anyone, and they could easily have your app up and running with a few commands. Docker makes running multiple servers very easy, especially with orchestration engines like Kubernetes and Docker Swarm.

Here are few Reasons, why you might want to use Docker:

1] Consistent & Isolated Environment: - In simple words, what the consistent environment here means is that the Docker image created by you during any development stage will work similarly in other SDLC phases also such as testing, production, etc.

2] Rapid Application Deployment: - The docker containers come up with the minimal runtime requirements of the application that allows them to deploy faster. Here, you're not required to set up a new environment – all you need to do is download the Docker image to run it on different environments. And let us tell you these images are quite smaller in size that further prompts rapid application deployment.

3] Ensures Scalability & Flexibility: - Docker allows you to rapidly create replications for redundancy reasons, and it makes you enable to start and terminate the application or services promptly to make things much easier.

4] Better Portability

5] Cost-Effective: - Docker subsequently requires a smaller team of professionals compared to the traditional workflow that also leads to minimized workforce costs for the organization.

- **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.

The architecture Docker uses is a client-server model. It consists of major components such as Docker's Client, Docker Host, Network and Storage components, and the Docker Registry/Hub.

1. Docker's Client: - Docker users can interact with the docker daemon through a docker client. Client is nothing but the commands which the user fires in order to process requests. Docker client uses the docker REST Api which mentioned earlier to sends requests to daemon.

2. Docker Host: - Docker host/ daemon listens for requests coming from the client and manages the docker objects. Docker objects such as images, containers, volumes and network infrastructure.

3. Docker Registry:-Docker registry is the place from where you can download and upload the docker images. In other words, Docker registry contains the docker repositories from where you can download the official and self-made docker images to make a suitable environment for your application. Docker has its own public registry named docker hub and docker cloud from where you can download the official images. Apart from public repositories you can make your own private repositories to manage the private self-designed images.

4. Docker image :- are read-only templates with instructions to create a docker container. Docker image can be pulled from a Docker hub and used as it is, or you can add additional instructions to the base image and create a new and modified docker image. You can create your own docker images also using a dockerfile. Create a dockerfile with all the instructions to create a container and run it; it will create your custom docker image.

- **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 Docker applications**. Using a YAML configuration file, Docker Compose allows us to configure multiple containers in one place. We can then start and stop all of those containers at once using a single command.

Additionally, **Docker Compose allows us to define common objects shared by containers**. For example, we can define a volume once and mount it inside every container, so that they share a common file system. Or, we can define the network used by one or more containers to communicate.

→ simple Docker Compose file:

```
version : '3.3'
services :
  frontend :
      container_name : "flask-container1"
      build : .
      ports :
        - '8888:8888'
      volumes :
        - my-flask-volume:/app

  bakend :
    container_name : "flask-container2"
    image : mysql:latest
    ports :
       - '3306:3306'
    environment:
                  MYSQL_ROOT_PASSWORD: PASSWORD

volumes :
     my-flask-volume :
       external : true
```

2] Dockerfile :-A Dockerfile is a plain text file that contains instructions for building Docker images. There's a Dockerfile standard they follow, and **the Docker daemon is ultimately responsible for executing the Dockerfile and generating the image**.

A typical Dockerfile usually starts by including another image. For example, it might build on a specific operating system or Java distribution.

From there, a Dockerfile can perform various operations to build an image:

Copy files from the host system into the container. For example, we might want to copy a JAR file that contains our application code.

Run arbitrary commands relative to the image. For example, we might want to run typical Unix commands to change file permissions or install new packages using a package manager.

Define the command that should be executed when a container is created. For example, a *java* command that loads our JAR file and starts the desired main method.

```
FROM openjdk:17-alpine
ARG JAR_FILE=target/my-app.jar
COPY ${JAR_FILE} my-app.jar
ENTRYPOINT ["java","-jar","/my-app.jar"]
```

3]Docker image: -
A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, l like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.

Docker images have multiple layers, each one originates from the previous layer but is different from it. The layers speed up Docker builds while increasing reusability and decreasing disk use. Image layers are also read-only files. Once a container is created, a writable layer is added on top of the unchangeable images, allowing a user to make changes.

4] Docker Container: -
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

- **In what real scenarios have you used Docker?**

Let’s get our hands dirty and code something already! You can see Docker as a way to pack you code in a nice little container that contains everything it needs to run it; it containerizes

your code. The benefits are numerous: containers are scalable, cost-effective and are isolated from each other. This part focusses on the docker elements:

→ **Dockerfile:** Specifications for how the image should be built
→ **Image:** Like a CD: it contains all code but it doesn't do anything yet.
→ **Container:** A running image. Think of this as the CD that you've just put in the CD-player. It's executing the image.

- ## Docker vs Hypervisor?

Hypervisors and Dockers are currently two of the most used software in the industry. This is due to the surge in the use of virtual machines and applications.

1] The most significant difference between hypervisors and Dockers is the way they boot up and consume resources. Hypervisors are of two types – the bare metal works directly on the hardware while type two hypervisor works on top of the operating system.

Docker, on the other hand, works on the host kernel itself. Hence, it does not allow the user to create multiple instances of operating systems.

Instead, they create containers that act as virtual application environments for the user to work on.

2] A hypervisor allows the users to generate multiple instances of complete operating systems.

Dockers can run multiple applications or multiple instances of a single application. It does this with containers.

3] Hypervisors enable users to run multiple instances of complete operating systems. This makes them resource hungry. They need dedicated resources for any particular instance among the shared hardware which the hypervisor allocates during boot.

Dockers, however, do not have any such requirements. 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.

- **What are the advantages and disadvantages of using docker?**

Advantages of Docker

- **Docker produces an API for container management** – Docker produces an API for container management in an image format and a chance to use a remote registry for sharing containers. This scheme serves both developers and system administrators with advantages for instance.
- **Fast application deployment** – containers carry the minimal runtime requirements of the application, decreasing their size and enabling them to be deployed instantly.
- **Transferable across machines** – an application and all its dependencies can be grouped into a separate container that is autonomous from the host version of Linux kernel, platform configuration, or deployment type. This container can be assigned to another machine that runs Docker and performed there without adaptability issues.
- **Version control and component retain** – you can pursue succeeding versions of a container, inspect irregularities or go back to previous versions. Containers reuse segments from the preceding layers, which makes them remarkably light.
- **Sharing** – you can use a distant repository to share your container with others. Red Hat provides a registry for this purpose, and it is also desirable to configure your own individual repository.
- **Light and minimal overhead** – Docker images are typically very small, which promotes rapid delivery and reduces the time to deploy new application containers.

Disadvantages of Docker

- **Containers don't work at bare-metal rates –** Containers utilize resources more efficiently than virtual machines. But containers are however directed to performance overhead due to overlay networking, interfacing within containers and the host system and so on. If you want 100% bare-metal performance, you want to apply bare metal, not containers.
- **The container ecosystem is split –** But the core Docker platform is open source, some container products don't work with other ones.
- **Data storage is intricate –** By design, all of the data inside a container leaves forever when it closes down except you save it somewhere else first. There are ways to store data tenaciously in Docker, such as Docker Data Capacities, but this is arguably a test that still has yet to be approached in a seamless manner.
- **Graphical applications do not operate well –** Docker was created as a solution for deploying server applications that don't need a graphical interface. While there are some

creative approaches that one can practice to run a GUI app inside a container, these solutions are solid at best.

- **Few applications do not benefit from Docker Containers** – In common, the applications that are intended to work as a collection of thoughtful microservices attain to get the most from containers. Contrarily, Docker's one real benefit is that it can interpret application delivery by giving an easy packaging mechanism.

What is a Docker namespace?

Docker uses a technology called namespaces to provide the isolated work space 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 are limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

- **The pid namespace:** Process isolation (PID: Process ID).
- **The net namespace:** Managing network interfaces (NET: Networking).
- **The ipc namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
- **The mnt namespace:** Managing filesystem mount points (MNT: Mount).
- **The uts namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

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.

By default, the Docker engine interacts with **DockerHub** , Docker's public registry instance. However, it is possible to run on-premise the open-source Docker registry/distribution, as well as a commercially supported version called **Docker Trusted Registry.** There are other public registries available online.

- ## What is an entry point?

Docker entrypoint is a Dockerfile directive or instruction that is used to specify the executable which should run when a container is started from a Docker image. It has two forms, the first one is the 'exec' form and the second one is the 'shell' form. If there is no entrypoint or CMD specified in the Docker image, it starts and exits at the same time that means container stops automatically so, we must have to specify entrypoint or CMD so that when we will start the container it should execute something rather than going to stop.

Syntax: - ENTRYPOINT ["executable", "param1", "param2", ...]

- How to implement CI/CD in Docker?

CI/CD (continuous integration and continuous Deployment or Delivery) in Docker can be implemented using the following steps:

1] Create a Dockerfile :- A Dockerfile isa script that contains instructions for Buildng a Docker image. It is a simple text file that contains commands such as FFROM,RUN,COPY,EXPOSE,ENV,etc., These commands are executed by the Docker Daemon during the build process to execute an image.

2]create a build pipeline: - setup 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 jenkins,CircleCI.

3] AuttomateTesting :- setup automated testing for the image such as unit tests, integration tests,acceptence tests. To ensure image is working as expected.

4] push the images to a registery :-Once the image is built and tested, it can be pushed to a docker registry such as dockerhub.si 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 AmazonnECS to deploy the image to a production environment.

6] Monitor a scale: - Monitor the deployed image and scale it as needed to handle increased.

- Will data on the container be lost when the docker container exits?

If the container still exists and stopped "can be viewed by docker ps -a", you can restart it without losing the container data.

Also, if you are mounting the container data directory to a directory on the host machine, then you still have the data even if the container got removed.

Also, if you are using any storage plugins then you shouldn't lose the container data regardless of the container status.

The only case you would lose the container data is when the container doesn't use any data persistence mechanism and the container got removed either because you started it with "—rm" option or because any sort of garbage collector that removes the exit containers.

- ## What is a Docker swarm?

A Docker Swarm is a container orchestration tool running the Docker application. It has 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 three types of docker swarm nodes which are given below.

1. Leader Node: - The leader node takes care of tasks such as task orchestration decisions for the swarm, managing swarm. If the leader node gets down or becomes unavailable due to any reason, the leadership is transferred to another Node using the same algorithm.
2. Worker Node: In general, all Nodes are the worker nodes even the manager node is also a worker node and capable of performing the task/operations when required resources are available for them.

How does Docker swarm work?
When you want to deploy a container in the swarm first, you have to launch services. Service consists of multiple containers of the same image. These services are deployed inside a node so to deploy a swarm at least one node has to be deployed.

API in the manager is the medium between the manager node and the worker node to communicate with each other by using the HTTP protocol. The service of one cluster can be used by the other. All the execution of the task is performed by the worker node.

- ## 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-name> <docker-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 ID od Name>

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

docker system prune -a