**1. What is Docker, and how is it different from a virtual machine?**

**Answer:** Docker is a platform for developing, shipping, and running applications in lightweight containers.

* Docker containers share the host system's kernel and are more efficient in terms of performance and resource utilization.
* Docker **Demon** is heart of docker, we are giving instruction to docker demon ex docker run sujay (demon receiving a instruction and creating that)

**Scenario-based Example:**

* **Scenario:** **You are deploying a microservice-based application. Why would you use Docker instead of a virtual machine**?
* **Answer:** Docker is preferred because it allows each **microservice** to run in its isolated container, sharing the host OS kernel.
* This reduces overhead compared to virtual machines, which require separate guest OS installations. It speeds up deployment and conserves system resources.

**1.1 Docker Architecture**

**Docker file 🡪 Docker Image 🡪 Docker Container**

**1.2 Drawbacks**

**Docker demon run with root user**

**2. How do you troubleshoot a failing Docker container?**

**Answer:** Troubleshooting involves:

1. Checking the logs using **docker logs** <container\_id>. to check error messages.
2. Inspecting the container with **docker inspect** <container\_id>. to check if the container configuration
3. Checking the status with **docker ps -a.**
4. Verifying network connectivity using docker network ls and docker network inspect.

**2.1 Docker Registry**

**Docker hub is used to storing images**

**3. How do you optimize Docker images to make them smaller?**

**Answer:**

1. Use a smaller base image (e.g., alpine instead of ubuntu).
2. Combine RUN statements in the Dockerfile.
3. Remove unnecessary files during build using --no-cache.
4. Use multi-stage builds

**4. What is a Docker volume, and why is it used?**

A **Docker volume** is a way to store data created by a container so that the data persists even after the container is stopped or deleted. It's commonly used to share data between the host and the container or between multiple containers.

**5. How do you handle environment-specific configurations in Docker?**

**Answer:** Environment variables can be passed at runtime using -e or --env-file. Docker Compose can also manage environment configurations.

**6. How would you handle container networking in Docker?**

**Answer:** Docker provides three networking modes:

1. **Bridge:** Default for standalone containers.
2. **Host:** Shares the host network stack.
3. **None:** No network connectivity.

**7. What is the difference between Docker Compose and Docker Swarm?**

**Answer:**

* **Docker Compose:** Tool for defining and running multi-container applications locally using a YAML file.
* **Docker Swarm:** Orchestrates containers across a cluster for high availability.

**8. How do you secure Docker containers?**

**Answer:**

1. Use trusted images from Docker Hub.
2. Limit container privileges (--cap-drop).
3. Use read-only file systems.
4. Scan images for vulnerabilities.

**9. How do you handle logging in Docker?**

**Answer:** Docker uses drivers like json-file, syslog, or third-party logging tools like ELK.

**10. How do you debug Dockerfile issues during the build process?**

**Answer:** Use the --progress and --no-cache options with docker build. Add RUN echo statements to debug intermediate steps.

**11. What is the difference between a Docker image and a Docker container?**

**Answer:** **Docker Image:** A blueprint or template for creating containers.

* It contains the application code, dependencies, and configurations.
* **Docker Container:** A running instance of a Docker image.
* It’s the execution environment for the application.

**12. How do you copy files from a container to the host system?**

**Answer:** Use the docker cp command.

**13**. **How do you check the status of a container?**

**Answer:** Use the docker ps command:

* **docker ps:** Shows running containers.
* **docker ps -**a: Shows all containers (running or stopped).

**14.** **How can you ensure that a container restarts automatically if it crashes?**

**Answer:** Use the --restart flag when running the container.

**15. What is the difference between CMD and ENTRYPOINT in a Dockerfile?**

**Answer:**

* CMD: Specifies default arguments for the container at runtime but can be overridden.
* ENTRYPOINT: Specifies the main command that always runs, and additional arguments are appended.

16. **How do you limit CPU and memory usage for a Docker container?**

**Answer:** Use the --memory and --cpus flags.

**17. What is a multi-stage build in Docker?**

**Answer:** A multi-stage build allows you to use multiple FROM statements in a Dockerfile to create smaller and optimized images.

**18. How do you clean up unused Docker resources?**

docker container **prune**

**19. What is Docker Compose, and how do you use it?**

**Answer:** Docker Compose is a tool to define and run multi-container applications using a YAML file.

**20. How do you connect two containers together?**

**Answer:** Use a Docker network.

**docker network create my\_network**

**docker run --network my\_network --name db postgres**

**docker run --network my\_network --name backend my\_backend**

**21. How do you update a running container with new changes?**

**Answer:** Stop and remove the old container, then run a new one with the updated image.

docker stop my\_app

docker rm my\_app

docker run -d –name my\_app my\_image:latest

**22. How do you debug a container that isn’t working as expected?**

**Answer:**

* Use docker logs to check logs.
* Use docker exec to access the container’s shell for debugging.

docker logs <container\_id>

**23. What is a Dockerfile, and how is it used?**

**Answer:** A Dockerfile is a text file containing instructions to build a Docker image.

FROM node:14

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

CMD ["npm", "start"]

**24. What are Docker namespaces, and why are they important?**

**Answer:** Docker uses **namespaces** to provide isolation for containers.

Each container gets its own isolated namespace for processes, networking, and file systems, ensuring they don’t interfere with other containers or the host.

**25. What is the difference between a bind mount and a volume in Docker?**

**Answer:**

* **Bind Mount:** Links a directory on the host to a directory in the container. The host fully controls it.

**(it will create bind repository that exists on container to local host)**

* **Volume:** Managed by Docker and more portable across systems. Best for persistent data.

**CMD:** docker volume inspect (volume name) it will give all details

* **docker volume** **rm (volme name)**
* **docker volume create (volme name)**
* **docker run -d --mount source= volume name,target=/app image name**

**26. What are Docker labels, and how are they used?**

**Answer:** Docker labels are metadata applied to images, containers, volumes, or networks.

They are key-value pairs used for identification, automation, or organizing resources.

**27. How do you secure Docker containers?**

**Answer:**

1. Use the **least privilege principle** (e.g., run as non-root user).
2. Keep your Docker images updated.
3. Use **Docker Content Trust (DCT)** to verify the integrity of images.
4. Limit container resources (CPU, memory).
5. Use network isolation (bridge, host, or custom networks)

**28. How do you manage secrets in Docker?**

**Answer** Secrets can be securely managed using Docker Swarm secrets or external tools like HashiCorp Vault.

**29. What is the difference between Docker Swarm and Kubernetes?**

**Answer:**

* **Docker Swarm:** A simple container orchestration tool built into Docker.
* **Kubernetes:** A powerful, feature-rich container orchestration platform with advanced scheduling, scaling, and networking.

**30. How do you monitor Docker containers?**

**Answer:** You can monitor containers using Docker CLI, logging tools, or external monitoring solutions like **Prometheus** and **Grafana**.

**31. What are Docker networking modes?**

**Answe:** Docker provides the following network modes:

1. **Bridge (default):** Containers communicate within a private network.
2. **Host:** Containers share the host’s network stack.
3. **None:** Containers have no network access.
4. **Overlay:** Used in Swarm mode for multi-host communication.
5. **Custom:** User-defined networks for better isolation.

Docker Compose

[docker/awesome-compose: Awesome Docker Compose samples](https://github.com/docker/awesome-compose)

services:

web:

build:

context: app

target: builder

ports:

- '80:80'

volumes:

- ./app:/var/www/html/