DOCKER

50 interview questions/answers

Basic Docker Questions

1. What is Docker, and how does it work?

Answer: Docker is an open-source platform that automates the deployment of applications inside lightweight, portable containers. Containers package the application along with its dependencies, ensuring it runs consistently in any environment.

Key Components:

- **Docker Engine:** Core engine running containers.
- **Docker Images:** Read-only templates for containers.
- **Docker Containers:** Instances of Docker images.
- **Docker CLI:** Command-line interface for managing Docker.

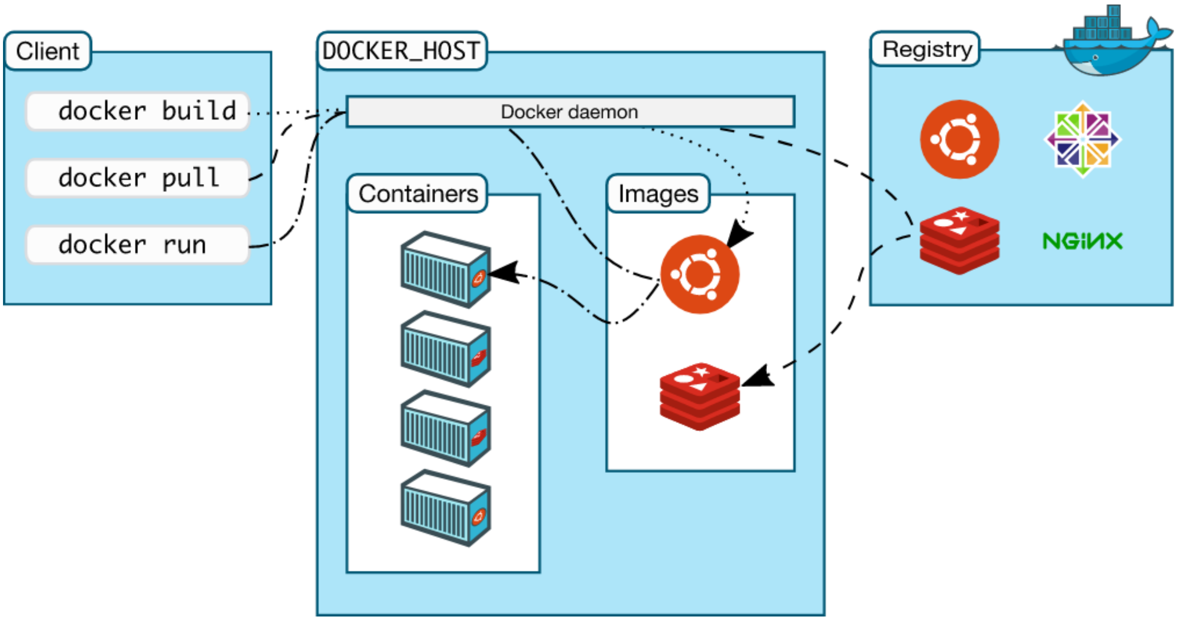
Example:

```
docker run -d -p 8080:80 nginx
```

This runs an Nginx web server accessible at http://localhost:8080.

---

2. What are the key components of Docker architecture?



Answer: Docker architecture consists of:

1. **Docker Engine:** Core of Docker, consisting of:
    - **Docker Daemon:** Executes commands and manages Docker objects.
    - **Docker CLI:** Interface for user interaction.
2. **Docker Images:** Templates for creating containers.
3. **Docker Containers:** Isolated environments for running applications.
4. **Docker Registries:** Store and distribute Docker images (e.g., Docker Hub).

---

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

Answer:

- **Docker Image:** A static, read-only template with the application and dependencies.
- **Docker Container:** A dynamic, runtime instance of a Docker image.

Example:

```
# Build an image

docker build -t myapp:1.0 .

# Run a container from the image

docker run -d myapp:1.0
```

---

4. Explain the Docker lifecycle.

Answer: The Docker lifecycle consists of:

1. **Create:** Define a container from an image.
2. **Start:** Launch the container.
3. **Run:** Combines create and start into one step.
4. **Pause/Unpause:** Temporarily halt/resume a container.
5. **Stop/Kill:** Gracefully or forcefully stop a container.
6. **Remove:** Delete a container.

Example:

```
docker create nginx

docker start <container_id>

docker stop <container_id>

docker rm <container_id>
```

5. What is a Dockerfile, and how is it used?

Answer: A Dockerfile is a text file containing instructions to assemble a Docker image. Each instruction adds a layer to the image.

Example Dockerfile:

```
FROM python:3.8

COPY app.py /app/

WORKDIR /app

RUN pip install flask

CMD ["python", "app.py"]
```

Commands:

```
# Build an image

docker build -t flask-app .

# Run the container

docker run -d -p 5000:5000 flask-app
```

6. What are the main benefits of using Docker?

Answer:

1. **Portability:** Consistent environments across development, testing, and production.
2. **Efficiency:** Lightweight and resource-friendly compared to virtual machines.
3. **Scalability:** Easy scaling of applications.
4. **Speed:** Faster deployment and startup times.
5. **Isolation:** Isolates applications from each other and the host OS.

7. How does Docker differ from virtual machines?

Answer:

Feature	Docker (Containers)	Virtual Machines
Architecture	Shares host OS kernel	Includes guest OS
Startup Time	Seconds	Minutes
Performance	Near-native	Overhead due to hypervisor

Size	Lightweight (MBs)	Heavyweight (GBs)

---

8. Explain the role of the Docker Engine.

Answer: Docker Engine is a client-server application with:

1. **Server (Daemon):** Manages Docker objects like containers and images.
2. **REST API:** Enables communication between client and server.
3. **Client:** CLI for user interaction with Docker Daemon.

---

9. How do you list all running containers in Docker?

Answer: Use the docker ps command:

```
docker ps
```

To list all containers, including stopped ones:

```
docker ps -a
```

---

10. What is the command to stop a running container?

Answer: To stop a running container gracefully:

```
docker stop <container_id>
```

To force stop:

```
docker kill <container_id>
```

Docker Commands

11. How do you build a Docker image?

Answer: Use the docker build command:

```
docker build -t myapp:1.0 .
```

Here:

- -t specifies the image tag.
- . specifies the directory containing the Dockerfile.

---

12. Explain the docker run command with an example.

Answer: The docker run command creates and starts a container.

Example:

```
docker run -d -p 8080:80 nginx
```

- -d: Run in detached mode.
- -p: Map host port 8080 to container port 80.
- nginx: Image to use.

---

13. What is the difference between COPY and ADD in Dockerfile?

Answer:

- **COPY:** Copies files from the host to the container.
- **ADD:** Similar to COPY, but also supports remote URL downloads and automatic extraction of tar archives.

Example:

```
COPY index.html /usr/share/nginx/html/

ADD https://example.com/config.tar.gz /app/
```

---

14. How do you tag a Docker image?

Answer: Tagging helps version Docker images.

```
docker tag <image_id> username/repo:tag
```

Example:

```
docker tag myapp:1.0 myrepo/myapp:latest
```

---

15. What is the use of the docker exec command?

Answer: The docker exec command runs a new process inside an existing container.

Example:

```
docker exec -it <container_id> /bin/bash
```

This opens an interactive shell in the container.

---

16. Explain the purpose of the docker-compose command.

Answer: docker-compose is used to define and manage multi-container Docker applications using a YAML file.

Example docker-compose.yml:

```
version: '3'

services:
  web:
    image: nginx
    ports:
      - "8080:80"
  app:
    build:
      context: .
```

Run with:

```
docker-compose up
```

---

17. How can you remove all stopped containers in Docker?

Answer: Use the following command:

```
docker container prune
```

This removes all stopped containers.

---

18. How do you inspect a running container?

Answer: Use the docker inspect command to view container details:

```
docker inspect <container_id>
```

This outputs information like network settings, volumes, and environment variables.

---

19. What is the docker logs command used for?

Answer: To view logs of a running container:

```
docker logs <container_id>
```

Add -f to follow live logs:

```
docker logs -f <container_id>
```

---

20. How can you monitor Docker containers?

Answer: Docker provides several tools for monitoring:

- **docker stats:** Shows real-time resource usage.

```
docker stats
```

- Use third-party tools like Prometheus, Grafana, or Datadog for advanced monitoring.

---

21. What is Docker Compose, and how is it used?

Answer: Docker Compose is a tool for defining and running multi-container Docker applications using a YAML file (docker-compose.yml). It simplifies managing related containers.

Example:

```
version: '3'
services:
  db:
    image: postgres
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
  web:
    image: nginx
    ports:
      - "8080:80"
    depends_on:
      - db
```

Run:

```
docker-compose up
```

This will spin up a PostgreSQL database and a Nginx web server.

---

22. How do you create a multi-container application with Docker Compose?

Answer: Multi-container applications are defined in a docker-compose.yml file. Each service represents a container.

Example:

```
version: '3'
services:
app:
    build:
    context: .
    ports:
      - "5000:5000"
  redis:
    image: redis
```

Run:

```
docker-compose up
```

This example builds a web app and connects it to a Redis container.

---

23. Explain Docker Swarm and its components.

Answer: Docker Swarm is Docker's native clustering and orchestration tool. It manages a cluster of Docker nodes as a single virtual system.

Components:

1. **Manager Node:** Orchestrates tasks and services.
2. **Worker Node:** Executes tasks assigned by the manager.
3. **Services:** High-level abstractions for running containers.

Commands:

```
# Initialize Swarm
docker swarm init
# Add nodes
docker swarm join --token <token>
```

---

24. What is a Docker Registry?

Answer: A Docker Registry stores and distributes Docker images. Examples include Docker Hub (public) and private registries.

Key Commands:

```
# Login to Docker Hub
docker login
# Pull an image
docker pull nginx
# Push an image
docker push myrepo/myapp:1.0
```

---

25. What is the difference between a public and private Docker registry?

Answer:

- **Public Registry:** Open to everyone, e.g., Docker Hub.
- **Private Registry:** Restricted access, often hosted on-premise or in a private cloud.

Example:

```
# Run a private registry

docker run -d -p 5000:5000 --name registry registry:2

# Tag and push an image

docker tag myapp:1.0 localhost:5000/myapp

docker push localhost:5000/myapp
```

26. How do you push an image to Docker Hub?

After tagging an image:

bash

```
docker push myrepo/myimage:v1
```

27. What are Docker volumes, and how are they managed?

Docker volumes are used to persist data outside the container filesystem. Volumes can be managed using docker volume commands.

28. Explain the concept of networking in Docker.

Docker uses virtual networks to allow containers to communicate with each other. Types of networks include:

- **Bridge**: Default network for containers.
- **Host**: Containers share the host's network stack.
- **Overlay**: Used for multi-host communication.

29. What are bridge networks in Docker?

A **bridge network** allows containers on the same Docker host to communicate with each other.

30. What is a Docker overlay network?

An **overlay network** is used for communication between containers on different Docker hosts (e.g., in a Swarm cluster).

Docker Security

31. **How do you secure a Docker container?**

To secure a Docker container:

- **Use minimal base images**: Reduce the attack surface by using smaller images.
- **Avoid running as root**: Use the USER directive in the Dockerfile to set a non-root user.
- **Scan images**: Use tools like Clair or Trivy to scan images for vulnerabilities.
- **Limit container privileges**: Use --cap-drop and --cap-add to drop unnecessary capabilities.

Example:

Dockerfile

```
FROM ubuntu

USER nonroot
```

32. **What is Docker Content Trust (DCT)?**

Docker Content Trust (DCT) is a security feature that uses digital signatures to verify the authenticity of images. When DCT is enabled, Docker only pulls and pushes signed images.

Command to enable DCT:

bash

```
export DOCKER_CONTENT_TRUST=1
```

33. **How do you implement authentication in Docker Hub?**

Docker Hub uses username/password authentication, but you can also use **two-factor authentication (2FA)** for added security. To authenticate via the CLI:

bash

```
docker login
```

34. **What is a rootless container in Docker?**

A **rootless container** runs without root privileges, enhancing security by preventing container breakouts. Docker supports rootless mode starting from version 20.10.

Command to enable rootless mode:

bash

```
dockerd-rootless-setuptool.sh install
```

35. **Explain how you can limit container resources in Docker.**

Docker allows you to limit resources like CPU and memory using the following flags:

- --memory: Limit memory usage.
- --cpu-shares: Limit CPU shares.
- --cpus: Limit CPU usage.

Example:

bash

```
docker run -d --memory="512m" --cpus="1.0" myimage
```

Real-world Scenarios

36. **How would you debug a failing Docker container?**

Steps to debug a failing container:

- **View logs**: Use docker logs <container_id> to check for errors.
- **Enter the container**: Use docker exec -it <container_id> bash to access the container and diagnose issues.
- **Check container status**: Use docker inspect <container_id> to view the container's status and details.

37. **What happens when a Docker container crashes?**

When a container crashes:

- Docker tries to restart the container based on the restart policy (--restart=always, --restart=on-failure).
- Logs can be accessed using docker logs <container_id>.

Example:

bash

```
docker run --restart=on-failure:5 myimage
```

38. **How do you handle persistent storage in Docker?**

Persistent storage is handled using **Docker volumes**, which are outside the container's filesystem. Volumes persist even when containers are stopped or removed.

Example:

bash

```
docker run -v /host/path:/container/path myimage
```

39. **How do you handle logging for multiple Docker containers?**

Docker provides centralized logging using tools like **ELK stack (Elasticsearch, Logstash, and Kibana)**, **Fluentd**, or **Prometheus**. These tools aggregate and analyze logs from multiple containers.

40. **Explain how you can update a running container.**

To update a running container:

- **Stop the old container**: docker stop <container_id>.
- **Remove the old container**: docker rm <container_id>.
- **Recreate the container with the updated image**: docker run <new_image>.

Integration and Ecosystem

41. **How does Docker integrate with Kubernetes?**

Docker is used as the container runtime in Kubernetes. Kubernetes schedules and manages the containers, while Docker builds and runs the containers.

Flow Diagram:

text

```
Kubernetes → Docker → Containers
```

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

- **Docker Swarm**: A native Docker tool for container orchestration. Simpler and more integrated into Docker.
- **Kubernetes**: A more complex and feature-rich orchestration platform that supports more advanced scaling, load balancing, and high availability features.

43. **How do you use Docker in a CI/CD pipeline?**

Docker is used in CI/CD pipelines to automate the testing, building, and deployment of applications. A typical flow:

- **Build**: A Docker image is built from source code.
- **Test**: The image is tested within a container.
- **Deploy**: The image is deployed to production.

Example using Jenkins:

groovy

```
pipeline {
  agent any
```

```
    stages {
        stage('Build') {
            steps {
                script {
                    docker.build('myimage')
                }
            }
        }
        stage('Deploy') {
            steps {
                script {
                    docker.image('myimage').push()
                }
            }
        }
    }
}
```

44. **What tools can be used to monitor Docker containers in production?**

Popular monitoring tools include:

- **Prometheus**: For metrics collection.
- **Grafana**: For visualization.
- **cAdvisor**: For container monitoring.
- **Docker stats**: Provides basic container metrics.

45. **How do you migrate a legacy application to Docker?**

Steps:

1. **Create a Dockerfile** for the legacy application.
2. **Build a Docker image**.
3. **Run the application** in a container.
4. **Test** the application for compatibility.
5. Optionally, use **Docker Compose** for multi-container setups.

Performance and Optimization

46. **How can you optimize the size of a Docker image?**

- **Use a smaller base image**: E.g., alpine instead of ubuntu.
- **Remove unnecessary files**: Clean up after installing dependencies.
- **Leverage multi-stage builds**: Use separate stages for building and running.

Example of multi-stage build:

Dockerfile

```
# Build Stage
FROM node:14 AS builder
WORKDIR /app
COPY . .
RUN npm install

# Production Stage
FROM node:14-slim
COPY --from=builder /app /app
WORKDIR /app
CMD ["npm", "start"]
```

47. **What are best practices for writing a Dockerfile?**

- **Minimize layers**: Combine commands to reduce the number of layers.
- **Use official images**: Base your images on official, trusted images.
- **Clean up after installing dependencies**: Remove temporary files to keep the image small.

48. **How do you analyze the performance of a Docker container?**

You can use the docker stats command to monitor the performance of containers. Additionally, use external tools like **cAdvisor** or **Prometheus** for more detailed metrics.

bash

```
docker stats
```

49. **What is Docker BuildKit, and why is it used?**

Docker BuildKit is an advanced build system for Docker that improves build performance, supports caching, and enables features like multi-stage builds and better control over build contexts.

Enable BuildKit:

bash

```
export DOCKER_BUILDKIT=1
```

50. **How do you manage a large number of containers in production?**

Use orchestration tools like **Docker Swarm** or **Kubernetes** to manage and scale large numbers of containers. These tools provide features like auto-scaling, load balancing, and health checks.

Example (Kubernetes):

bash

```
kubectl apply -f deployment.yaml
```