Docker Interview Questions and Answers

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

Docker is an open-source containerization platform that allows developers to package applications and their dependencies into lightweight containers. It works by using OS-level virtualization to run multiple isolated environments on the same host machine.

2. What are the differences between a container and a virtual machine?

- **Containers** share the host OS kernel, making them lightweight and faster to start.
- **Virtual Machines (VMs)** include a full OS with a hypervisor, making them more resource-intensive and slower to boot.

3. What are the advantages of using Docker?

- Portability across different environments
- Faster startup and scaling compared to VMs
- Efficient resource utilization
- Version control and repeatability using Docker images

4. What is a Docker Image? How is it different from a Container?

A **Docker Image** is a blueprint that contains all dependencies and configurations needed to run an application. A **Container** is a running instance of an image.

5. What is a Dockerfile, and how do you use it?

A **Dockerfile** is a script containing instructions to build a Docker image. You use it by running:

```
docker build -t my-image .
```

6. Explain the different Docker object types.

- **Image**: A read-only template used to create containers.
- **Container**: A running instance of an image.
- **Volume**: A persistent storage mechanism for containers.
- **Network**: Allows communication between containers.

7. What is the purpose of `docker-compose`? How is it different from `docker run`?

`docker-compose` is a tool for defining and running multi-container applications using a `docker-compose.yml` file. Unlike `docker run`, which starts a single container, `docker-compose up` can start multiple containers at once.

8. What is a Docker Volume, and why is it used?

A Docker Volume is a persistent storage mechanism that allows data to persist even when containers are deleted. It is useful for databases and logs.

9. Explain the difference between `COPY` and `ADD` in a Dockerfile.

- `COPY` simply copies files from the host to the container.
- `ADD` can also copy files but supports automatic extraction of compressed files and fetching remote URLs.

10. How do you persist data in Docker containers?

By using **Docker Volumes** or **Bind Mounts**:

```
docker volume create my_volume
docker run -v my_volume:/data my-container
```

11. What is the difference between `CMD` and `ENTRYPOINT` in a Dockerfile?

- `CMD` provides default arguments that can be overridden at runtime.
- `ENTRYPOINT` defines the main executable of the container and is not easily overridden.

12. How do you expose a port in Docker?

- In the Dockerfile: `EXPOSE 80`
- When running a container: `docker run -p 8080:80 my-image`

13. What is a Multi-Stage Build in Docker?

Multi-Stage Builds allow creating small, optimized images by using intermediate build stages.
Example:

```
FROM golang:alpine AS builder
WORKDIR /app
COPY . .
RUN go build -o myapp

FROM alpine
COPY --from=builder /app/myapp /myapp
CMD ["/myapp"]
```

Bhuvan

14. What happens when a container exits?

It enters the `Exited` state and can be restarted using `docker start <container_id>`.

15. How do you check running containers and their logs?

- List running containers:

  ```
  docker ps
  ```

- Check logs:

  ```
  docker logs <container_id>
  ```

16. How do you manage secrets in Docker?

Docker provides **secrets management** via **Docker Swarm**, but in standalone mode, environment variables or external secret management tools (like HashiCorp Vault) can be used.

17. What is the difference between a bridge network, host network, and overlay network in Docker?

- **Bridge Network**: Default network mode, allowing isolated communication between containers.
- **Host Network**: Containers share the host's network, improving performance.
- **Overlay Network**: Used in Docker Swarm for multi-host communication.

18. Explain the concept of Docker namespaces and cgroups.

- **Namespaces** provide isolation for processes, networks, and mounts.
- **cgroups** control resource allocation (CPU, memory, I/O) for containers.

19. How do you optimize Docker images to reduce their size?

- Use **alpine-based images**.
- Minimize layers in Dockerfile.
- Remove unnecessary files using `.dockerignore`.
- Use multi-stage builds.

20. How do you troubleshoot issues in a running Docker container?

- Check logs: `docker logs <container_id>`
- Check processes: `docker top <container_id>`
- Inspect container: `docker inspect <container_id>`
- Access shell: `docker exec -it <container_id> sh`

21. How does Docker handle networking, and what is `docker network create` used for?

Docker provides **bridge, host, and overlay networks** to manage container communication. The `docker network create` command allows creating custom networks for better control.

22. What are the key components of a Dockerfile?

- `FROM`: Specifies the base image.
- `WORKDIR`: Sets the working directory.
- `COPY` / `ADD`: Copies files into the container.
- `RUN`: Executes commands during image build.
- `CMD` / `ENTRYPOINT`: Defines the main command.
- `EXPOSE`: Declares ports to expose.
- `ENV`: Sets environment variables.

23. What are the benefits of using Docker Compose?

- Allows defining multi-container applications in a single YAML file.
- Simplifies managing dependencies between containers.
- Provides an easy way to scale services.

24. How do you scale services in Docker Compose?

Use the `--scale` flag when running `docker-compose up`:

```
docker-compose up --scale web=3
```

This starts 3 instances of the `web` service.

25. What is the difference between `depends_on` and `links` in Docker Compose?

- `depends_on`: Ensures a container starts after another but doesn't wait for readiness.
- `links`: Creates a deprecated legacy network link between services.

Scenario-based Docker Questions

26. What is your experience with Docker?

A common interview question is about the candidate's experience with Docker, specifically asking for a situation where Docker was used to solve a problem. An example response could involve addressing inconsistency issues between development and production environments by deploying an application with Kubernetes, thus demonstrating practical knowledge of Docker.

Bhuvan

27. How do you clean up Docker resources?

Using the `docker prune` command allows for the deletion of images, volumes, and containers to free up space. The `docker system prune` command removes all unused resources permanently, and caution is advised in production environments.

28. Is there a limit on the number of Docker containers that can run simultaneously?

There is no explicit limit on container numbers; it is primarily constrained by the hardware specifications of the host machine. Factors such as CPU capacity and memory size directly influence the number of containers that can be effectively run.

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

Resource constraints can be applied using the `docker run` command with the `--cpu` and `--memory` options. Example:

```
docker run --memory=512m --cpus=1 nginx
```

30. How do you scale Docker containers horizontally?

Horizontal scaling involves running multiple instances of Docker containers, which can be managed using orchestration tools like Kubernetes. Kubernetes allows users to define the number of replicas of a specific image in a manifest file, facilitating efficient scaling of applications.

31. What is the difference between Docker containers and Kubernetes pods?

A Docker container is an isolated environment running a single application instance, while a Kubernetes pod can contain one or more containers, providing a higher-level abstraction.

32. How do you debug failing Docker containers?

Common methods include checking container logs using the `docker logs` command and accessing the container shell for direct inspection. Health checks can also be defined for containers to monitor their status and automate recovery actions based on predefined conditions.

33. How do you optimize a Dockerfile for efficient builds?

Strategies include using smaller base images, minimizing the number of layers by combining commands, and employing multi-stage builds to exclude unnecessary files. These optimizations enhance performance and reduce resource consumption.

34. How do you update a Docker container without data loss?

The process typically involves backing up the container's data, stopping the container, pulling the updated image, and then starting the new container. Verifying the integrity of the data post-update is crucial.

35. How do you secure Docker containers?

Best practices include using trusted base images, regularly updating Docker and its underlying host, and scanning images for vulnerabilities. Implementing access controls and monitoring container activity are also key components of a robust security strategy.

36. How is Docker used in CI/CD?

An example could include using Jenkins to build Docker images during the CI phase and deploying them to production using Kubernetes in the CD phase. This integration ensures consistency across development, testing, and production environments.

37. How do you use Docker in AWS?

An example could involve using Amazon ECS for deploying Docker containers, which simplifies the management of containerized applications. AWS Fargate can be utilized for serverless compute options.

38. How do you monitor Docker containers?

Docker commands like `docker stats` and `docker top` provide built-in monitoring capabilities. Open-source tools such as Prometheus and Grafana can also be employed for more comprehensive monitoring solutions.

39. What are the best practices for Docker in production?

Best practices include using lightweight containers, regularly updating software, and implementing orchestration platforms like Kubernetes. Monitoring resource usage and applying security measures are also critical for maintaining production stability.

40. How does Docker integrate with Kubernetes?

Kubernetes orchestrates multiple Docker containers, providing benefits like load balancing and automated scaling. Docker manages the containers on each node, while Kubernetes oversees the overall deployment and lifecycle management.

41. How do you use Terraform for Docker deployment?

Terraform can be used to automate Docker container deployments by defining ECS services and tasks, specifying Docker images from ECR. Terraform allows for easy updates to infrastructure by applying changes to the defined scripts.

42. How do you implement Canary Deployments with Docker and Kubernetes?

Canary deployments involve deploying a new version alongside the current version and directing a portion of traffic to it for testing. Kubernetes configurations can be used to gradually increase traffic to the new version before a full rollout.