Top 26 Docker Interview Questions and Answers for 2025

Discover the top Docker interview questions and answers to help you prepare for your upcoming job interview!

Contents

Nov 24, 2024 · 15 min read

Laiba Siddiqui
I'm a content strategist who loves simplifying complex topics.

TOPICS

Docker

MLOps

Docker has become the most popular containerization tool in modern software development, especially within DevOps and CI/CD workflows. It simplifies application deployment and management through containers, which enables delivering of software quickly and consistently.

Its scalability and flexibility features make Docker a requirement for data-related roles such as data engineering, MLOps, and even data science. That's why I've compiled commonly asked Docker interview-related questions, covering core concepts and real-world scenarios.

What is Docker?

Docker is a container platform that developers use to package applications with all their dependencies so they can run smoothly across different environments.

Although containers share the same OS kernel, each one operates in its own isolated environment. This setup minimizes compatibility issues, reduces delays, and improves communication between development, testing, and operations teams.

Docker logo. Image source

In 2023, **Docker led the containerization market with over 32% share**. This highlights its importance in modern software development! That's why you can expect recruiters to test your Docker expertise in data-related job interviews.

Basic Docker Interview Questions

First, familiarize yourself with some fundamental Docker concepts. These basic questions will help you build your understanding and prepare for the initial phase of the interview.

1. What is a Docker image?

A Docker image is like a blueprint that creates containers. It has everything a developer needs to run an application, such as:

- Code
- Libraries
- Settings

When you use a Docker image, Docker turns it into a container, which is a fully isolated environment. That's where the application runs independently.

2. What is a Docker host?

A Docker host is the system where we install Docker. It acts as the primary environment that runs and manages Docker containers. We can set up a Docker host on a local device or in a virtual or cloud environment.

3. How is a Docker client different from a Docker daemon? Can you share an example?

The Docker client and Docker daemon work side-by-side but have separate roles. The Docker client is the tool that sends commands and the Docker daemon is the engine that acts on those commands.

For example, if we type the `docker run` command to start a container, the client will take the request and send it to the Docker daemon. The Docker daemon will then handle the real work by starting the container.

4. Can you explain what Docker networking is and which commands can create a bridge and an overlay network?

Docker networking allows containers to connect and communicate with other containers and hosts. The `docker network create` command allows us to set up user-defined networks.

- **Bridge network**: Creates a local network for communication between containers on the same Docker host.
    - Command: `docker network create -d bridge my-bridge-network`
    - This sets up a bridge network called `my-bridge-network` for containers on the same host.

- **Overlay network**: Enables communication between containers across multiple Docker hosts, often used in a Swarm setup.
    - Command: `docker network create --scope=swarm --attachable -d overlay my-multihost-network`
    - This creates an attachable overlay network called `my-multihost-network` for containers running on different hosts in a Docker Swarm.

5. Explain how Docker bridge networking works.

Bridge networking is the default setup Docker uses to connect containers. If we don't specify a network, Docker links it to the bridge network. This bridge connects all containers on the same Docker host. Each container has a unique IP address, which allows containers to communicate directly.

Intermediate Docker Interview Questions

These questions are asked to test your knowledge of intermediate-level Docker concepts.

6. What is a Dockerfile? Explain how you would write it.

A Dockerfile is a script that defines instructions for building a Docker image. Each command in the Dockerfile sets up a specific part of the environment. When we run these commands, Docker builds an image layer-by-layer. Here's how we can write it:

1. First, choose a base image. It contains essential tools for the app.
2. Next, specify a working directory inside the container. That's where app files will be stored and run.
3. In the third step, use `COPY . .` command to copy all project files into the container's working directory.
4. Use the `RUN` command to install dependencies.
5. Use the `EXPOSE` command to specify the port on which your app runs.
6. Now, define the command Docker should run when it starts the container.

Here's a simple example of a Dockerfile for a Python web application:

```
# Step 1: Choose a base image
FROM python:3.9-slim
# Step 2: Specify the working directory
WORKDIR /app
# Step 3: Copy project files into the container
COPY . .
# Step 4: Install dependencies
RUN pip install -r requirements.txt
# Step 5: Expose the port the app runs on
EXPOSE 5000
# Step 6: Define the default command
CMD ["python", "app.py"]
```


Using the Dockerfile above, you can build an image with `docker build -t my-python-app .` and run a container using `docker run -p 5000:5000 my-python-app .`

7. What is Docker Compose, and how is it different from Dockerfile?

Docker Compose is a tool for defining and managing multi-container Docker applications using a YAML file (docker-compose.yml). It allows us to configure services, networks, and volumes in a single file, making it easier to manage complex applications.

Differences from Dockerfile:

- A Dockerfile is used to build a single Docker image by defining its layers and dependencies.
- Docker Compose is used to run and orchestrate multiple containers that may rely on each other (e.g., a web app container and a database container).

For example, a `docker-compose.yml` file might look like this:

```
version: '3.9'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - db
  db:
    image: postgres
    volumes:
      - db-data:/var/lib/postgresql/data
volumes:
  db-data:
```


This file defines two services, `web` and `db`, with networking and volume configurations.

8. Why do we use volumes in Docker?

We use Docker volumes to keep data safe outside of Docker containers. They provide a separate location on hosts where data lives even if the container gets removed. Also, it's easier to manage, back up, and share the volumes among containers.

9. What are Docker bind mounts, and why do we prefer volumes over bind mounts?

With Docker bind mounts, we can share files between the host machine and a container. They connect a specific file on the host system to a location in the container. If we make any changes to files, it'll instantly show up inside the container.

Docker mounts are suitable for real-time file sharing — but they rely on the host OS, which raises security issues.

On the contrary, since Docker volumes work independently, they're more secure than mounts.

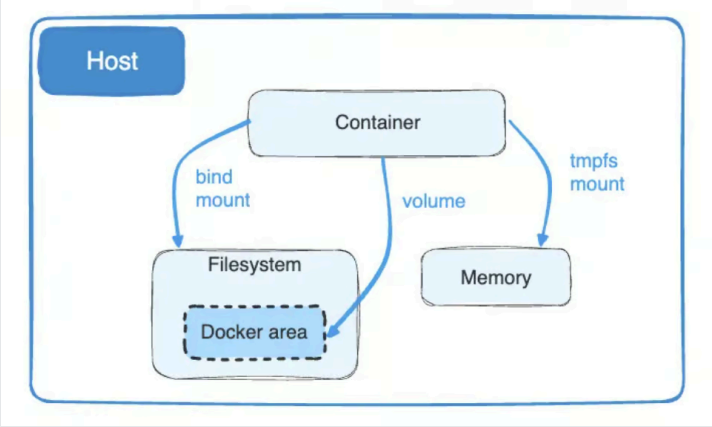


Diagram of docker bind mounts and volumes. Image source: Docker

10. What is Docker Swarm?

Docker Swarm is a container orchestration tool that manages and deploys services across a cluster of Docker nodes. It enables high availability, scalability, and load balancing, allowing multiple hosts to act as a single virtual Docker engine.

11. Can we autoscale Docker Swarm?

No, Docker Swarm does not natively support automatic autoscaling. To achieve autoscaling, we need to integrate monitoring tools and use scripts to manually adjust the number of instances. Here's how:

- Install a monitoring tool, like Prometheus or Grafana, to track resource usage, like CPU and memory.
- Set the scaling triggers. For instance, we can define that CPU usage over 82% will trigger scaling up.
- Next, write a script with the `docker service scale` command to adjust the number of replicas. For example, to scale a service to 5 replicas: `docker service scale <service_name>=5`

By combining monitoring tools, triggers, and scripts, you can implement a form of autoscaling in Docker Swarm, even though it's not built-in.

12. How would you use Docker Compose to scale services?

To scale services using Docker Compose, we can use the `--scale` flag with the `docker-compose up` command. This is typically used for stateless services like web servers. For example, to scale a web service to 3 instances:

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

It's important to ensure the `docker-compose.yml` file has properly defined services and uses an external load balancer or supports scaled instances. Scaling stateful services (e.g., databases) requires additional configuration to ensure data consistency.

13. Can a container restart by itself? Define its default and always policies.

Yes, a container can restart by itself. However, we need to set a restart policy for that.

Docker has different restart policies that control when and how containers should restart. The default policy is no, which means a container won't restart if it stops. With the always policy, Docker will automatically restart the container whenever it stops.

We can use this command to apply always policy:

```
docker run --restart=always <container-name>
```

Advanced Docker Interview Questions

Now, let's move on to advanced Docker interview questions!

14. Explain the lifecycle of Docker containers.

A Docker container passes through a lifecycle that defines the states a container can be in and how it operates in those states. The stages in a Docker container lifecycle are:

- **Create:** In this state, we set up a container from an image with the `docker create` command.
- **Run**: Here, we use the `docker start` command to run the container, which performs the tasks until we stop or pause it.
- **Pause**: We use the `docker pause` command to stop the process. This state keeps the memory and disk intact. If you want to resume the container, use the `docker unpause` command.
- **Stop**: If the container is inactive, it enters the stop stage, but this can happen due to multiple reasons:

- **Immediate stop**: The `docker kill` command stops the container without cleanup.
- **Process completion**: When the container finishes the task, it stops automatically.
- **Out of memory**: Container stops when it consumes too much memory.

- **Delete**: In the final stage, we remove the stopped or created container with the `docker rm` command.

15. What is a Docker image repository?

A Docker image repository stores and shares multiple container images of the same name with the clients or the community. We can label them with tags to differentiate their different versions. For example, `app/marketing_campaign:v1` will be the first version of a marketing app, and `app/marketing_campaign:v2` will be the second version.

Docker Hub, the most popular Docker image repository, allows users to host, share, and pull container images publicly or privately. Other alternatives include Amazon ECR, Google Artifact Registry, and GitHub Container Registry.

16. Tell me about 3 best practices to keep a Docker container safe.

To enhance container security and minimize common vulnerabilities, I follow these best practices:

1. **Choose lightweight images**: Use minimal base images like Alpine to reduce the attack surface.
2. **Limit system calls**: Since Docker containers can access unnecessary calls, use tools like Seccomp to restrict these calls.
3. **Secure sensitive data**: Use Docker secrets to manage API keys or passwords. They encrypt the secrets and make them available only during runtime.

17. Why do Docker containers need health checks?

Docker containers rely on health checks to ensure they run smoothly. Deploying a container that is running but doesn't process requests can create issues for deployment teams. Health checks monitor these issues in real-time and informs us instantly.

For example, a health check can be added in a Dockerfile like this:

```
HEALTHCHECK --interval=30s --timeout=10s --retries=3 CMD curl -f http://localhost:8080/health ‖ exit 1
```

This health check pings the container's health endpoint every 30 seconds and marks the container as unhealthy if it fails three consecutive attempts. This proactive monitoring helps identify and resolve issues promptly.

18. What are dangling images in Docker, and how do you remove them?

Dangling images in Docker are unused image layers that no longer have any tags associated with them. They often build up when you create new images with the same name and tag, leaving the old layers without references. These images can consume significant disk space, so it's important to clean them up. Here's how:

1. Run the `docker images -f dangling=true` command to find dangling images.
2. Then, run the `docker image prune -f` command to delete all images in one go.
3. If you want to remove images manually, use the `docker rmi -f $(docker images -f dangling=true -q)` command.

These steps help keep your system clean and free up storage efficiently.

Docker and Kubernetes Interview Questions

Docker and **Kubernetes** are often used together, so finding some Kubernetes questions in a Docker interview wouldn't be unexpected, particularly if the role is oriented to **DevOps**. Here are some questions you may be asked:

19. What is the primary difference between Docker and Kubernetes?

Docker is a containerization platform that allows you to build, ship, and run containers. It focuses on creating and managing individual containers.Kubernetes, on the other hand, is an orchestration platform designed to manage multiple containers at scale. It handles deployment, scaling, load balancing, and self-healing across clusters of nodes.

Learn more about the differences in the **Kubernetes vs Docker** blog post.

20. Compare Docker Swarm with Kubernetes.

Kubernetes and Docker Swarm manage containers, but they work differently:

- **Kubernetes** manages large and complex container setups. Its self-healing and built-in monitoring features make it a more suitable option for complex environments.
- **Docker Swarm** is suitable for smaller or less complex setups as it doesn't offer any built-in features like Kubernetes. We can easily integrate it with Docker tools like Docker CLI and Docker Compose.

21. How does Kubernetes manage a large number of Docker containers?

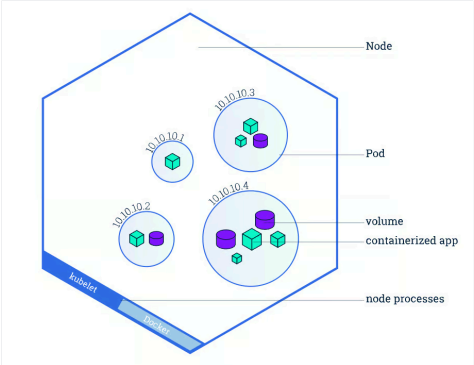
While Docker is great for creating and running containers, managing a large number of them requires Kubernetes. Kubernetes orchestrates containers efficiently by:

- **Setting resource limits**: It allocates CPU, memory, and other resources to each container to prevent overconsumption.
- **Scheduling containers**: Kubernetes decides where to run each container, optimizing resource utilization across nodes in a cluster.
- **Scaling automatically**: Based on workload demand, it scales pods (groups of one or more containers) up or down to maintain performance and efficiency.

By automating these processes, Kubernetes ensures smooth operation, even when managing thousands of containers. While occasional errors can occur, its self-healing capabilities, like restarting failed containers, minimize disruptions.

22. What is a Pod in Kubernetes, and how is it different from a container?

A Pod is the smallest deployable unit in Kubernetes and represents a group of one or more containers that share the same network namespace, storage, and configuration.Unlike individual containers, Pods allow multiple tightly coupled containers to work together as a single unit (e.g., a web server and a sidecar logging container).



*Overview of a Kubernetes node, highlighting pods and containers. Image source: **Kubernetes**.*

23. How can you manage sensitive data like passwords in Docker and Kubernetes?

- **In Docker:** We can use Docker secrets, which encrypt sensitive data and make it accessible only to authorized containers at runtime.
- **In Kubernetes:** We use Secrets objects, which store sensitive data like passwords, tokens, and API keys. Secrets can be mounted as volumes or exposed as environment variables to pods securely.

Example in Kubernetes:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  password: cGFzc3dvcmQ=  # Base64-encoded "password"
```


Scenario-Based Docker Interview Questions

The interviewer asks scenario-based and problem-solving questions to test how you approach real-world problems. Let's take a look at some questions to give you an idea:

24. Imagine you're creating an image of a Maven-based API. You've already set up the Dockerfile with basic configurations. Now, you notice the image size is large. How would you reduce it?

Example answer:

To reduce the size of a Docker image for a Maven-based API, I would follow these steps:

Create a .dockerignore file in the project directory to specify files and folders that should not be included in the Docker build context. This prevents unnecessary files from being added to the image, reducing its size. For example, I'd add the following to .dockerignore :

```
.git        # Version control files
target      # Compiled code and build artifacts
.idea       # IDE configuration files
```


Optimize the Dockerfile using multi-stage builds. I'd build the Maven project in one stage and copy only the necessary artifacts (e.g., compiled JAR files) into the final stage to keep the image small. Example Dockerfile with multi-stage build:

```
# Stage 1: Build the application
FROM maven:3.8.5-openjdk-11 AS build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package
# Stage 2: Create a lightweight runtime image
FROM openjdk:11-jre-slim
WORKDIR /app
COPY --from=build /app/target/my-api.jar .
CMD ["java", "-jar", "my-api.jar"]
```

By ignoring unnecessary files and using multi-stage builds, one can significantly reduce the image size while maintaining efficiency.

25. Imagine you need to push a Docker container image to the Docker Hub with Jenkins. How would you do that?

Example answer:

Here's how I'd push a Docker container image to Docker Hub with Jenkins:

1. **Configure a Jenkins pipeline**: Create a multi-branch pipeline job in Jenkins and link it to the repository containing the Dockerfile and Jenkinsfile.
2. **Define the pipeline in my Jenkinsfile**: The `Jenkinsfile` would include the following steps:
    - Build the Docker image
    - Log in to Docker Hub (using credentials stored securely in Jenkins)
    - Push the image to Docker Hub
3. **Run the pipeline**: Trigger the Jenkins job. It will build the image, log in to Docker Hub, and push the image automatically.

26. Imagine you have to migrate a WordPress Docker container to a new server without losing any data. How would you do that?

Example answer:

Here's how I'd migrate a WordPress Docker container:

1. **Backup the WordPress data**: Export the container's persistent data (WordPress files and database). I'd use `docker cp` or a volume backup tool to back up the necessary volumes, typically the `html` directory for WordPress files and the database volume.
2. **Transfer the backup files**: I'd use `scp` to securely copy the backup files to the new server.
3. **Set up WordPress on the new server**: I'd deploy a new WordPress container and database container on the new server.
4. **Restart and verify**: Lastly, I'd restart the containers to apply the changes and verify that the WordPress site is running correctly.

By backing up volumes and restoring them to a new server, one can migrate WordPress without losing data. This method avoids reliance on specific extensions and provides more control over the migration process.

Tips for Preparing for a Docker Interview

If you're reading this guide, you've already taken an important step to excel in your upcoming interview! But for complete beginners, preparing for an interview can be overwhelming. That's why I've put together some tips:

Master the basics of Docker

To excel in a Docker interview, start with a solid understanding of its core concepts.

- Learn how Docker images serve as the blueprint for containers, and practice creating, running, and managing containers to get familiar with their lightweight, isolated environments.
- Explore Docker volumes to handle persistent data effectively, and dive into networking by experimenting with bridge, host, and overlay networks to facilitate container communication.

- Study Dockerfiles to understand how images are built, focusing on key instructions like `FROM`, `RUN`, and `CMD`.
- Additionally, get hands-on with Docker Compose to manage multi-container applications and understand how Docker registries, such as Docker Hub, store and share images.

DataCamp offers plenty of other resources to guide you throughout your learning journey:

- **For Docker introductory concepts:** Introduction to Docker Course
- **For Docker intermediate concepts:** Intermediate Docker Course
- **For learning containerization and virtualization:** Containerization and Virtualization Concepts Course

Get hands-on experience with Docker

Once you've learned essential Docker topics, it's time to challenge yourself with some practical work. Here are 10 excellent Docker project ideas for beginners and more advanced learners. When working on these projects, use DataCamp's Docker cheat sheet to keep key commands at your fingertips.

Document Your Experience

Be ready to discuss your Docker experience in interviews. Prepare examples of:

- **Projects**: Highlight the Dockerized applications you've built or contributed to.
- **Challenges**: Describe issues you encountered, such as debugging containers or optimizing images, and how you resolved them.
- **Optimization**: Share how you improved build times, reduced image sizes, or streamlined workflows with Docker Compose.
- **Collaboration**: If you worked in a team, explain how you used Docker to improve collaboration, testing, or deployment processes.

Your real-world examples will demonstrate your practical knowledge and problem-solving skills!

Conclusion

As you prepare for your interview, remember these questions are a starting point. While memorizing answers can help, interviewers value candidates who can demonstrate practical experience and a deep understanding of containerization concepts. You should practice implementing these concepts in real-world scenarios and build your projects.

If you're a beginner, start with our Introduction to Docker course. In the end, success in your interview will come from combining theoretical knowledge with hands-on experience and the ability to articulate your problem-solving approach!

FAQs

Do I need to learn Kubernetes to use Docker?

No, you don't need to learn Kubernetes to use Docker. Docker does a completely different job than Kubernetes. It's used to build, run, and manage containers on a single machine.

Does Docker need coding?

How long will it take to prepare for the Docker interview?

AUTHOR
Laiba Siddiqui

I'm a content strategist who loves simplifying complex topics. I've helped companies like Splunk, Hackernoon, and Tiiny Host create engaging and informative content for their audiences.

TOPICS

Docker MLOps

Learn more about Docker with these courses!

TRACK

Containerization and Virtualization with Docker and Kubernetes

13 hr

Learn the power of Docker and Kubernetes, this interactive track will allow you to build and deploy applications in modern environments.

See Details → Start Course

See More →

Related

BLOG

Top 30 Cloud Computing Interview Questions and...

BLOG

Top 30 Machine Learning Interview Questions For 2025

BLOG

The Top 39 Data Engineering Interview Questions and...

See More →