

DOCKER

COMPLETE GUIDE TO DOCKER FOR BEGINNERS
AND INTERMEDIATES

Questions and Answers



DAY 1

MASTER DEVOPS WITH VINAY

1: What is Docker, and what are its primary use cases?

Answer:

Docker is an open-source platform that simplifies the deployment, scaling, and management of applications by using containers. A container packages an application along with all its dependencies, libraries, and configuration files into a single lightweight unit. This ensures the application runs reliably across different environments, from development to production.

Why Docker is Used:

1. Portability: Containers can run on any system with Docker installed, avoiding environment mismatch issues.
2. Consistency: Ensures identical environments across development, testing, and production stages.
3. Efficiency: Shares the host OS kernel, making containers faster and more resource-efficient than traditional virtual machines.
4. Scalability: Easily scale applications by running multiple instances (containers) in parallel.

2: How does a Docker image differ from a Docker container?

Answer:

- Docker Image is a read-only template that contains the application code, dependencies, libraries, environment variables, and configuration files needed to run the application. Think of it as a blueprint or snapshot used to create containers.
- Docker Container is a runtime instance of a Docker image. It is the actual running process that includes the image and a writable layer on top. Containers are isolated environments that run the application based on the image.

3: Can you list some key advantages of Docker in application development and deployment?

Answer:

Docker offers several key advantages that make it a popular choice for application development, deployment, and operations:

1. Portability:

- Docker containers can run on any system with Docker installed, ensuring consistent behavior across development, testing, and production environments.

2. Lightweight:

- Containers share the host OS kernel, making them more efficient and faster to start than traditional virtual machines.

3. Consistency & Reproducibility:

- Docker images ensure the same setup and environment every time you deploy, reducing "it works on my machine" issues.

4. Rapid Deployment:

- Containers start almost instantly, enabling faster development, testing, and deployment cycles.

5. Isolation:

- Each container runs in its own isolated environment, improving security and reducing the risk of conflicts between applications.

6. Scalability:

- Docker makes it easy to scale applications horizontally by spinning up multiple container instances using orchestration tools like Kubernetes or Docker Swarm.

7. Simplified CI/CD Integration:

- Docker integrates well with modern DevOps pipelines, improving automation, testing, and delivery.

8. Environment Management:

- Easily define and manage different environments (dev, test, prod) with Docker Compose and Dockerfiles.

4: What occurs when a running Docker container is stopped?

Answer:

When you stop a running Docker container:

1. Docker first sends a SIGTERM signal to the container's main process, giving it a chance to gracefully shut down and perform any necessary cleanup.
2. If the process doesn't stop within the default grace period (10 seconds), Docker sends a SIGKILL signal to forcefully terminate it.

🔴 Command:

```
docker stop <container-id>
```

State Transition:

Running → Stopped

5: How does Docker ensure application isolation?

Answer:

Docker ensures application isolation by leveraging several key Linux kernel features:

1. Namespaces:

- Isolate system resources such as processes (PID), network interfaces, and file systems.
- Each container gets its own isolated environment, preventing interference with others.

2. Control Groups (cgroups):

- Manage and limit resource usage like CPU, memory, and disk I/O per container.
- Ensures one container doesn't consume all system resources.

3. Union File Systems:

- Uses layered file systems like AUFS or OverlayFS to build efficient, modular images.
- Supports image versioning and reusability while keeping containers lightweight.

4. Container Runtime:

- The container runtime (e.g., containerd) manages container lifecycle and enforces isolation at the OS level.


Together, these components ensure that each container runs securely and independently from others on the same host.

6: Explain the lifecycle of a Docker container.

Answer:

The lifecycle of a Docker container consists of the following key stages:

1. Create:

- A container is created from a Docker image but not yet running.
-  Command:
docker create <image-name>


2. Start:

- The container starts, and the application/process inside begins execution.
-  Command:
docker start <container-id>


3. Running:

- The container is actively running until the application completes or it's manually stopped.


4. Stop:

- Gracefully stops the container by sending a SIGTERM signal.
-  Command:
docker stop <container-id>

5. Kill:

- Immediately and forcefully stops the container using a SIGKILL signal (no cleanup).
-  Command:
docker kill <container-id>


6. Remove:

- Deletes the container from the system (must be stopped first).
-  Command:
docker rm <container-id>

7: How can you restart a Docker container?

Answer:

Yes, a Docker container can be restarted using the docker restart command.

 Command:

docker restart <container-id>

Explanation:

- This command stops the container (if it's running) and then starts it again.
- It's commonly used to apply configuration changes or recover from temporary errors without removing the container.

8: What is the purpose of Docker tags in an image?

Answer:

Docker tags are used to label and identify specific versions of a Docker image. They help manage, reference, and deploy the correct version of an image with clarity and consistency.

 Syntax:

`<image-name>:<tag>`

◆ Examples:

1. `nginx:1.21` → Refers to version 1.21 of the NGINX image.
2. `ubuntu:latest` → Points to the most recent stable version of Ubuntu.

 Purpose:

- Manage and differentiate multiple versions of the same image.
- Ensure consistent deployments by referencing specific versions.
- Simplify image updates and rollbacks during CI/CD workflows.

9: How can you check the resource usage of a Docker container?

Answer:

You can monitor the real-time resource usage of Docker containers using the `docker stats` command.

 Command:

`docker stats`

 Output Includes:

- CPU usage
- Memory consumption
- Network I/O
- Disk I/O
- Container name or ID

 To monitor a specific container:

`docker stats <container-id>`

This is especially useful for diagnosing performance issues or analyzing container resource consumption in live environments.

10: What is the method to give a specific name to a Docker container?

Answer:

You can assign a custom name to a Docker container using the `--name` option with the `docker run` command.

 Command:

`docker run --name <custom-name> -d <image-name>`

◆ Example:

`docker run --name my-nginx -d nginx`

 Explanation:

- Assigning a custom name makes it easier to identify, manage, and reference containers, especially in multi-container or production environments.
- It eliminates the need to use automatically generated container IDs.