Docker interview questions

1. What is docker ?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you.

1. what is difference between vm's and docker ?

Hypervisors(VM’s) ,guestos(VM’s), libraries sharable (docker), more resources(docker)

1. what is dockerfile?
   * Plain Text file
   * That contains different command to build a Image
   * (FROM, MAINTAINER, RUN, CMD, LABEL, EXPOSE, ENV,ADD, COPY, ENTRYPOINT, VOLUME, USER, WORKDIR)

[https://kapeli.com/cheat\_sheets/Dockerfile.docset/Contents/Resources/Docu](https://kapeli.com/cheat_sheets/Dockerfile.docset/Contents/Resources/Documents/index) [ments/index](https://kapeli.com/cheat_sheets/Dockerfile.docset/Contents/Resources/Documents/index)

1. what is difference between the kill and stop?

The ***docker stop*** command stops the container gracefully and provides a safe way out. If a ***docker stop*** command fails to terminate a process within the specified timeout, the Docker issues a kill command implicitly and immediately.

1. what is difference between and ADD and COPY commands in a dockerfile ?

copying of local files into the container

* + Add (download Tar file and unzip it)
  + Add (use to download remote sources)
  + CoPy (use to multistage dockerfile )

1. what is difference between CMD and ENTRYPOINT ?
   * Both are used to Execution time or Run time.
   * CMD can override Parameter during Run time
   * If multiple CMD’s are there only latest

One will execute(Last one)

1. which is the base image you will prefer your docker file ?

Alpine(size 5MB)

1. how login a running container  docker ps
2. how to check live logs in docker containers?

 docker logs --tail 5 container\_name\_or\_ID

1. how to check no.of containers running your machine ? --> docker ps
2. how to check stop and running containers ?  docker ps -a
3. what is port forwarding ? -p 8181:9090
4. How to run a container in background?  docker run -d <Img/tag>
5. what is first line in a dockerfile ?  ARG,FROM <baseImg>
6. what are the commands you have used in your docker file ?

FROM RUN CMD ENTRYPOINT label workdir COPY ADD EXPOSE VOLUME ENV ARG HEALTHCHECK

1. what is label in dockerfile?  Label <meta@data>
2. what is docker tag command ?

 <Image Names> docker build -t <t name>, docker tag

1. How you store data persistently ? volumes [docker volume create <volumeName>] 2types:

/var/lib/docker/volumes/

Bind mounts may be stored anywhere on the host system.

tmpfs mounts are stored in the host system’s memory only

1. have you worked on docker volumes? -v hostmachine/path:container/data/path
2. what is docker network ? types of networks ?

There are three common Docker network types – **bridge networks, used within a single host, overlay networks**, for multi-host

communication, and macvlan networks which are used to connect Docker containers directly to host network interfaces.

1. I have a one web application is running on one container and database is running in one container how you communicate these two containers? -> --link , properties files,[database-info]
2. where you are store docker images?--> ecr, nexus , docker Hub
3. how to run multiple containers at a time?  docker compose
4. what is difference between exec and attach ? 

# The docker exec command runs a new command in a running container. execute the below command after restarting container nodeapi.

// start the container

docker start nodeapi// execute the exec command docker exec -it nodeapi bash

once attached, you can do all the cli commands like you can check the node version, run the bundle file, exit etc..

**Docker attach**

# This command to attach your terminal’s standard input, output, and error (or any combination of the three) to a running container using the container’s ID or name.

# Attach the running container with the following command. you need to press enter once execute this command, docker is waiting for your input.

docker attach nodeapi

1. what is save and load commands in docker?

**docker commit**

**Create a new image from a container’s changes**

* **General Questions**

**What is Docker, and how does it differ from virtual machines?**

Docker is a containerization platform that packages apps and dependencies in isolated containers.

Unlike VMs, containers share the host OS, making them lightweight and faster to start.

**Explain the key components of Docker: Dockerfile, Image, Container, Volume, and Network.**

Dockerfile: Script to build images.

Image: Read-only template with app code and dependencies.

Container: Running instance of an image.

Volume: Persistent storage.

Network: Enables container communication.

**What is the purpose of a Dockerfile?**

A Dockerfile automates the creation of Docker images using instructions to install dependencies, copy files, and configure the environment.

**How do you create and run a container from an image?**

Use **docker build -t image-name** . to create the image, and

**docker run -d --name container-name image-name** to run the container.

* **Scenario-Based Questions**

**Ensure app runs same in all environments – how Docker helps?**

Docker packages the app with all dependencies, ensuring consistency across dev, test, and prod without "it works on my machine" issues.

**Deploy two Node.js apps on same host – how to isolate?**

Use separate containers and Docker networks to isolate them. Assign different ports or use reverse proxies if needed.

**New dev setup quickly – how Docker helps?**

Provide a Dockerfile or Compose file; devs can run a single command (docker compose up) to get the full environment.

**Keep image minimal – how?**

Use lightweight base images like Alpine and combine RUN steps. Remove unnecessary files to reduce image size.

**Review a Dockerfile with manual package installs – improvements?**

Recommend multi-stage builds, minimal base images, and using apt-get clean to avoid caching issues and reduce size.

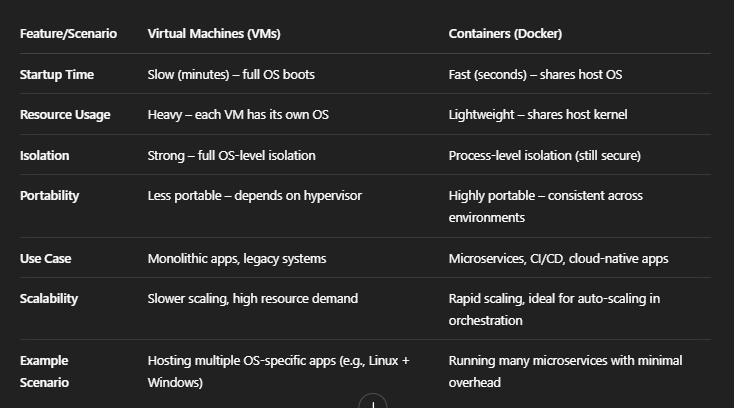
**Explain Docker architecture to non-technical person.**

Docker runs small, isolated environments (containers) on the same machine, like apps on a smartphone – all sharing the same base OS.

**VMs vs Containers – scenario comparison.**

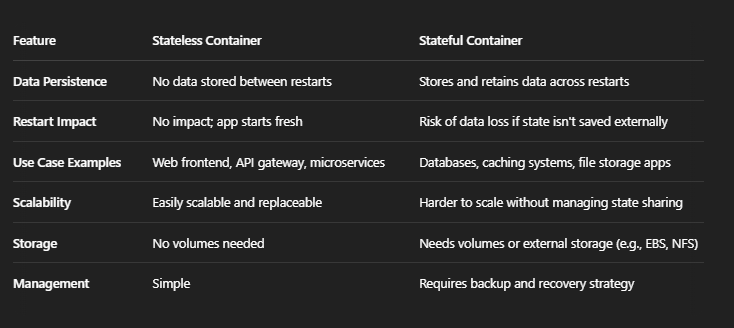
Containers start in seconds, consume fewer resources, and are portable.

VMs take longer to boot and have larger footprints.

****

**Deleted a production container – now what?**

If it was stateless, redeploy from the image. If stateful, check for data volumes; recover from backups or persistent storage.



**Install & configure Docker in new Linux system – how?**

Use official Docker installation script or package manager.

Add user to Docker group, test with docker run hello-world.

**App works locally, not in Docker – troubleshooting?**

Check **image build logs**, **container logs** (docker logs), **mounted volumes**, **exposed ports**, and **base image dependencies**.

* **Docker Images & Containers – Answers**

**General Questions-**

**How do you list, start, stop, and remove Docker containers?**

**List**: docker ps

**Start**: docker start <container-id>

**Stop**: docker stop <container-id>

**Remove**: docker rm <container-id>

**What happens when you delete a running container?**

Deleting a running container stops it, but does not remove its image or associated volumes. You can remove it with docker rm -f.

**How do you commit changes in a running container to a new image?**

Use docker commit <container-id> <new-image-name>. This creates a new image based on the current state of the container.

**What is the difference between COPY and ADD in a Dockerfile?**

COPY copies files from the host to the image, while ADD can also extract tar files and fetch files from URLs.

**Explain the ENTRYPOINT and CMD instructions in a Dockerfile.**

ENTRYPOINT: Ensures the container always runs the Python app as the main process.

CMD: Provides default arguments, like the port number, but can be overridden when running the container.

**# Step 5: Set ENTRYPOINT to run the Python application**

ENTRYPOINT ["python", "app.py"]

docker run myflaskapp --port 8080

**# Step 6: Provide default arguments (e.g., default port)**

CMD ["--host", "0.0.0.0", "--port", "5000"]

* **Scenario-Based Questions**

**Your container crashes immediately after starting – steps to diagnose and fix?**

Check the container logs with docker logs <container-id>, inspect the Dockerfile for issues, and test the image locally with docker run -it.

**New version of your image is larger and takes longer to deploy – what can you do?**

Optimize the Dockerfile by removing unnecessary files, using multi-stage builds, and choosing a smaller base image (e.g., Alpine).

**Build a CI/CD pipeline with Docker images – how to structure Dockerfile for caching?**

Arrange the Dockerfile to leverage caching by placing less frequently changing instructions (e.g., installing dependencies) before more frequently changing ones.

Install dependencies (this layer is cached unless requirements.txt changes)

RUN pip install --no-cache-dir -r requirements.txt

**Container should run with non-root privileges – how to modify Dockerfile?**

Add the USER instruction in the Dockerfile after installing dependencies and avoid running processes as root inside the container.

# Switch to non-root user

**USER appuser**

**Service exposes ports but they’re not accessible externally – troubleshooting approach?**

Ensure the container’s ports are mapped correctly to the host using the -p flag in docker run.

Verify firewall settings and ensure the service inside the container is listening on the correct port.

**Need to pass secrets to container securely – how?**

|  |
| --- |
| Use Docker secrets for sensitive data or pass secrets via environment variables using the -e flag, ensuring they’re not stored in Dockerfiles.  DOCKER\_BUILDKIT=1 docker build --secret id=mysecret,src=secret.txt . --> bash command to store secreat |

|  |
| --- |
| DockerFile :  # syntax=docker/dockerfile:1.2  FROM alpine  # Access the secret at build time (not saved in image)  RUN --mount=type=secret,id=mysecret cat /run/secrets/mysecret  docker run -e DB\_PASSWORD=SuperSecret123 myapp  docker run -v /secrets/db:/run/secrets/db myapp |

**How to handle an app with multiple processes running in one container?**

It’s better to run each service in its own container, but you can use process managers like supervisor

if necessary to manage multiple processes.

**Need to test different version of app in parallel – how to do it with Docker?**

Run multiple containers with different image tags (versions) or use Docker Compose to

spin up separate environments for each version.

**Reduce number of layers in Docker image – how to restructure Dockerfile?**

Combine RUN commands into a single layer where possible,

remove unnecessary files (e.g., build dependencies), and avoid creating redundant intermediate images.

|  |
| --- |
| # Combine RUN steps  RUN apt-get update && \  apt-get install -y curl && \  apt-get clean && \  rm -rf /var/lib/apt/lists/\* |

**Build takes too long due to frequent image rebuilds – how to optimize?**

Use Docker’s cache effectively by minimizing the number of layers,

combining similar RUN instructions, and using specific versioned

dependencies to avoid unnecessary updates.

A screenshot of a computer

AI-generated content may be incorrect.

**Build takes too long due to frequent image rebuilds – how to optimize?**

Reorder Dockerfile to Maximize Cache

Use .dockerignore

Use Multi-Stage Builds

Minimize Image Layers

Use Lightweight Base Images

Cache Dependencies

**Docker Networking –**

**What are the different types of Docker networks?**

Docker supports bridge, host, overlay, macvlan, and none. Each has a specific use case for communication or isolation.

**What is the default network driver used by Docker?**

Docker uses the **bridge network** as the default driver for standalone containers.

**How does the bridge network work in Docker?**  
It creates an internal virtual network where containers can communicate using IP addresses and DNS names.

**Difference between Bridge Host Overlay Macvlan None**

A screenshot of a computer

AI-generated content may be incorrect.

**What is an overlay network in Docker and when is it used?**

Overlay networks enable container communication across multiple hosts and are commonly used in Docker Swarm.

**How does container DNS resolution work in Docker?**

Docker provides an internal DNS server that resolves container names to IP addresses within user-defined networks.

**How can you list all Docker networks on a host?**

Run **docker network ls** to see all existing networks on the host.

**How do you inspect a specific Docker network?**

Use **docker network inspect <network-name>** to view connected containers, subnets, and configurations.

**Can you connect a container to multiple networks?**

Yes, with docker network connect, a container can join additional networks after creation.

**What happens if two containers on different custom networks need to communicate?**

They can’t by default. You must connect both containers to a common user-defined network.

**What is the difference between ingress and overlay networks in Docker Swarm?**

Ingress handles external requests to services, while overlay is used for internal service-to-service communication.

**What is MACVLAN and when would you use it?**

MACVLAN gives containers their own MAC addresses, useful for network-level separation or legacy integration.

**How can you restrict container access to external networks?**

Use **--network=none** or firewall rules to block internet access from the container.

**How does port mapping (-p) work in Docker?**

Port mapping binds a host port to a container port, e.g., -p 8080:80 exposes container port 80 on host port 8080.

🔌 Docker Networking - Scenario-Based Questions Explained

**You have two containers that need to talk to each other using service names. How would you configure networking for that?**

Explanation: Containers can communicate using service names if they are part of the same user-defined network. A user-defined bridge network allows containers to discover each other by their service names. You would create a custom bridge network and run the containers on that network.

**A container is unreachable from your host machine. What steps would you take to troubleshoot the networking issue?**

Explanation: First, ensure the container has the **correct port mapping (-p)** set. Next, verify **the container is running (docker ps).** Then check if the container is on the correct network (**docker inspect <container-name>)**. If using custom networks, make sure the container is attached to the proper network.

**You want to isolate multiple containers into different networks for security. How would you achieve that using Docker?**

Explanation: You can create multiple user-defined networks (e.g., **docker network create <network-name>**), and attach containers to different networks. This ensures containers in one network cannot directly communicate with containers in another unless explicitly connected.

**A container needs access to an external database server on a private subnet. How would you configure networking and routing?**

Explanation: For external access, you can connect the container to the host network (using --network host) or configure a custom bridge network with routing to the private subnet. You may also use macvlan to allow the container to have its own IP on the same network as the external database.

**How does Docker’s bridge network work internally? What happens if you run multiple containers without specifying a custom network?**

Explanation: The bridge network creates a private internal network on the host, where containers communicate using IP addresses assigned by Docker. If containers are run without a specified network, they are connected to the default bridge network, where they can communicate with each other but not the host or external networks directly.

**How can you connect a container to multiple networks? Give a real-world use case where this is required.**

Explanation: You can connect a container to multiple networks using the command docker network connect <network-name>. A real-world use case might be a web application container connected to one network for accessing internal services and another for external access (e.g., for security or performance reasons).

**You need to expose a container to a specific port range on the host. How would you do this using Docker networking options?**

Explanation: Use the -p option to expose specific ports or a range. For example, -p 8000-8100:8000-8100 will map the host port range 8000-8100 to the container’s corresponding port range. This allows multiple ports to be mapped at once.

**A container in a swarm is not accessible via its published port. What could be the networking-related causes?**

Explanation: First, check that the overlay network is correctly configured for swarm communication. Ensure that firewall settings on the nodes don’t block the port. Verify the container is running (docker service ps <service-name>) and that the published ports are correctly mapped.

**Explain how DNS-based service discovery works in a Docker user-defined bridge network.**

Explanation: Docker provides DNS service discovery within a user-defined bridge network. Containers can communicate with each other using their container names or the service name in Docker Compose. Docker automatically updates the DNS resolution as containers are added or removed from the network.

**Can two containers on the same host but different user-defined networks communicate directly? How would you enable or restrict it?**

Explanation: By default, containers on different networks cannot communicate unless explicitly connected. To enable communication, you can use docker network connect to connect the containers to a common network. To restrict communication, ensure containers are on separate networks and don't connect them unless necessary.

**Docker security interview questions :-**

1. **What is Docker Content Trust (DCT), and how does it work?**

DCT leverages the Notary framework to sign images with cryptographic keys. When enabled, `docker pull` and `docker push` verify signatures, ensuring only trusted images are used.

2**. How do you ensure containers run with the least privilege?**

Create and switch to non‑root users in the Dockerfile (`USER appuser`), drop unnecessary Linux capabilities, and avoid privileged or `--cap-add` flags unless absolutely required.

**3. What mechanisms does Docker provide for image signing and verification?**

Docker uses Notary (via `docker trust`) to sign manifests and enforce signature verification at pull time. You can manage keys locally or via a root/targets key hierarchy.

**4. How can you scan Docker images for known vulnerabilities?**

Integrate tools like Docker Scout, Clair, Trivy, or Anchore into your CI pipeline to analyze image layers against vulnerability databases and fail builds on critical findings.

**5. What’s the difference between user namespaces and POSIX capabilities in Docker?**

User namespaces remap container UIDs to unprivileged host UIDs, isolating root. POSIX capabilities selectively grant or drop kernel privileges (e.g. `CAP\_NET\_ADMIN`) without full root rights.

**6. How would you secure the Docker daemon socket (`/var/run/docker.sock`)?**

Restrict access via UNIX socket permissions or group membership, avoid mounting it into containers, and consider using a TLS‑protected TCP socket instead of the default socket.

**7. Explain how to use seccomp profiles with Docker containers.**

Seccomp filters system calls at the kernel level. You can supply a custom JSON profile (`--security-opt seccomp=profile.json`) or use Docker’s default profile to block dangerous syscalls.

**8. What are AppArmor and SELinux, and how do they integrate with Docker?**

AppArmor (Ubuntu) and SELinux (RHEL/CentOS) enforce Mandatory Access Control policies. Docker labels containers and applies profiles (`--security-opt apparmor=…` or relies on SELinux labels) to restrict actions.

**9. How do you prevent a container from accessing sensitive host resources?**

Use read‑only mounts (`--read-only`), drop capabilities, set resource and path restrictions, and avoid bind‑mounting host directories unless explicitly needed.

**10. Describe strategies to manage and rotate secrets for containerized applications.**

Use Docker secrets (Swarm), Kubernetes Secrets, or external vaults (HashiCorp, AWS Secrets Manager). Mount secrets at runtime in-memory and rotate them through automated tooling with minimal downtime.

**11. How can you enforce TLS for Docker daemon and client communication?**

Generate server and client certificates, configure the daemon with `--tlsverify --tlscacert --tlscert --tlskey`, and point clients to the CA and client certs to require mutual TLS.

**12. What is the purpose of the `no-new-privileges` flag in Docker?**

`--security-opt no-new-privileges` ensures neither the container’s processes nor their children can gain additional privileges via setuid binaries or kernel exploits.

**13. How would you isolate containers at the network level for enhanced security?**

Use user-defined networks with restrictive policies, apply network policies in orchestrators (Swarm/Kubernetes), or leverage CNI plugins that support micro‑segmentation and firewall rules.

**14. What are the security implications of running containers in privileged mode?**

`--privileged` grants all host capabilities and disables many isolation features, effectively giving container root access to the host and greatly increasing risk if compromised.

**15. How do you implement runtime security monitoring for Docker containers?**

Deploy agents like Falco, Aqua Security, or Sysdig to watch system calls, file changes, and network activity in real time, alerting or blocking suspicious behaviors.

1. Docker Engine

Ye main service hai jo system pe chalta hai.

Components:

Docker Daemon (dockerd): Background service, sab kuch handle karta hai.

Docker CLI (docker): Command line se jo tu command deta hai, woh yahin se jata hai.

REST API: Docker internal communication ke liye.