Docker Interview Questions and Answers

# 1. What is the difference between Virtual Machines and Containers?

Virtual Machines (VMs) emulate entire physical machines including the operating system, allowing multiple OS environments to run on a single physical machine. Containers, on the other hand, share the host OS kernel and isolate applications at the process level, making them lightweight and faster to start. VMs are suitable for running different OS types, while containers are ideal for deploying microservices and applications with consistent environments.

# 2. Explain the Docker lifecycle.

The Docker lifecycle includes the following stages:  
- Build: Create a Docker image using a Dockerfile.  
- Pull: Download an image from a registry.  
- Run: Start a container from an image.  
- Pause/Unpause: Temporarily stop/resume container processes.  
- Stop/Kill: Gracefully or forcefully stop a container.  
- Restart: Restart a stopped container.  
- Remove: Delete a container or image.

# 3. Write some Docker commands.

Some common Docker commands:  
- docker build -t image\_name .  
- docker run -d -p 80:80 image\_name  
- docker ps (list running containers)  
- docker stop container\_id  
- docker rm container\_id  
- docker images (list images)  
- docker rmi image\_id

# 4. Write a Dockerfile for one application. Explain each layer in it.

Example Dockerfile for a Python Flask app:  
FROM python:3.9-slim # Base image with Python  
WORKDIR /app # Set working directory  
COPY requirements.txt . # Copy dependencies file  
RUN pip install -r requirements.txt # Install dependencies  
COPY . . # Copy application code  
CMD ["python", "app.py"] # Command to run the app  
Each layer adds a step to build the image. FROM sets the base, WORKDIR sets the working directory, COPY and RUN install dependencies and copy code, CMD specifies the default command.

# 5. What is a docker-compose file? Explain what it does. Write one sample file if you can.

A docker-compose file defines and runs multi-container Docker applications. It uses YAML syntax to configure services, networks, and volumes.  
Example docker-compose.yml:  
version: '3'  
services:  
 web:  
 build: .  
 ports:  
 - "5000:5000"  
 redis:  
 image: "redis:alpine"  
This file defines two services: a web app and a Redis instance.

# 6. By default, which Docker network is present?

Docker creates three default networks: bridge, host, and none. The 'bridge' network is the default for containers if no network is specified.

# 7. What is the purpose of a multi-stage Dockerfile? How does it reduce the image size?

Multi-stage Dockerfiles allow you to use multiple FROM statements to build and copy only necessary artifacts to the final image. This reduces image size by excluding build tools and intermediate files.

# 8. Write the multi-stage Dockerfile for the same.

Example multi-stage Dockerfile:  
FROM python:3.9-slim AS builder  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --user -r requirements.txt  
  
FROM python:3.9-slim  
WORKDIR /app  
COPY --from=builder /root/.local /root/.local  
COPY . .  
ENV PATH=/root/.local/bin:$PATH  
CMD ["python", "app.py"]

# 9. How does container-to-container communication happen? Explain it.

Containers can communicate via Docker networks. By default, containers in the same bridge network can communicate using container names as hostnames. For complex setups, user-defined networks are preferred.

# 10. Mention some Docker network types and explain their real-world use cases.

- Bridge: Default network for standalone containers.  
- Host: Shares host's network stack; useful for performance.  
- None: No networking; used for isolated containers.  
- Overlay: Used in Docker Swarm for multi-host communication.  
- Macvlan: Assigns MAC addresses; used for legacy applications.

# 11. What is the difference between CMD and ENTRYPOINT?

CMD provides default arguments for ENTRYPOINT or the command to run if ENTRYPOINT is not specified. ENTRYPOINT sets the executable. ENTRYPOINT is preferred for fixed commands, CMD for default parameters.

# 12. Where are Docker volumes stored?

Docker volumes are stored in /var/lib/docker/volumes on the host system. They persist data independently of containers.

# 13. What is the difference between COPY and ADD?

COPY copies files/directories from source to destination. ADD does the same but also supports remote URLs and automatic extraction of tar archives.

# 14. How many containers can we run in Docker exactly?

There is no fixed limit. The number of containers depends on system resources like CPU, memory, and disk space.

# 15. What happens to the data inside a container when you delete the running container?

Data stored inside the container is lost when the container is deleted unless volumes are used. Volumes persist data beyond container lifecycle.

**You’re running an app using docker-compose with low traffic. As traffic grows, how do you scale the application in AWS? What services will you choose — EKS, ECS or EC2? Why?**

Step-by-Step Scaling Strategy

1. Containerize Your App: Since you're already using Docker Compose, your app is containerized. You’ll need to convert your docker-compose.yml into a format compatible with your chosen AWS service.
2. Choose an Orchestration Platform: Let’s compare the three options:

1. Amazon EC2 (Elastic Compute Cloud)

* Pros:
  + Full control over the infrastructure.
  + Can run Docker directly on EC2 instances.
  + Suitable for custom setups or legacy applications.
* Cons:
  + Requires manual scaling, load balancing, and orchestration.
  + Higher operational overhead.
* Use Case: Good for low-level control or when you need custom networking/storage setups.

2. Amazon ECS (Elastic Container Service)

* Pros:
  + Fully managed container orchestration.
  + Integrates well with AWS services (ALB, CloudWatch, IAM).
  + Easier migration from Docker Compose using tools like ecs-cli or Compose-X.
* Cons:
  + Less flexible than Kubernetes (EKS) for complex workloads.
* Use Case: Ideal for most applications scaling from Docker Compose. Easier to manage and cost-effective.

3. Amazon EKS (Elastic Kubernetes Service)

* Pros:
  + Full Kubernetes support.
  + Highly flexible and powerful for complex microservices.
  + Large ecosystem and community support.
* Cons:
  + Steeper learning curve.
  + More expensive and complex to manage than ECS.
* Use Case: Best for large-scale, enterprise-grade applications with complex orchestration needs.

✅ Recommended Choice: ECS

For most Docker Compose-based applications scaling in AWS, Amazon ECS is the best starting point because:

* It’s easier to migrate from Compose.
* It’s cost-effective and integrates well with AWS.
* It supports auto-scaling and load balancing.
* You can use Fargate (serverless containers) to avoid managing EC2 instances.

**Application works via localhost but not over the web — how will you troubleshoot**

1. Check Network Accessibility

* Is the server publicly accessible?
  + Ensure the server is not bound only to localhost (127.0.0.1). It should bind to 0.0.0.0 or the public IP.
  + Example: In Docker, use EXPOSE 80 and run with -p 80:80.

🌐 2. Verify DNS and IP

* Are you using the correct domain or IP?
  + Try accessing via public IP first.
  + If using a domain, check DNS propagation and correctness using nslookup or dig.

🔒 3. Firewall and Security Groups

* Are ports open?
  + On AWS, check Security Groups and Network ACLs.
  + Ensure inbound rules allow traffic on the required port (e.g., 80 or 443).

🐳 4. Docker Configuration

* Is the container exposing the correct port?
  + Check docker-compose.yml or docker run command.
  + Example:

🧱 5. Web Server Configuration

* Is the web server (e.g., Nginx, Apache) configured correctly?
  + Ensure it listens on the correct interface and port.
  + Check for any redirect or proxy misconfigurations.

🔁 6. Reverse Proxy or Load Balancer

* Is there a proxy in front of your app?
  + Check if it's forwarding requests correctly.
  + Look at logs for errors or misrouting.

📜 7. Application Logs

* Are there errors in the logs?
  + Check both Docker container logs (docker logs <container\_id>) and app logs.
  + Look for binding errors, permission issues, or request failures.

🧪 8. Test with Curl or Browser

* Try:
  + This helps isolate whether the issue is with the browser, DNS, or server.

🧰 Tools You Can Use

* netstat or ss to check listening ports.
* curl, telnet, or nc to test connectivity.
* AWS CloudWatch for logs and metrics**.**

# Kubernetes Interview Questions and Answers

## 1. Why is Kubernetes considered over Docker? Mention the advantages of it.

Kubernetes is considered over Docker Swarm for container orchestration due to its advanced features, scalability, and community support. Advantages include:  
- Automated bin packing: Efficient resource utilization.  
- Self-healing: Automatically replaces failed containers.  
- Horizontal scaling: Scale applications up/down easily.  
- Service discovery and load balancing.  
- Rolling updates and rollbacks.  
- Secret and configuration management.  
Kubernetes supports complex deployments and is cloud-agnostic, making it ideal for production-grade systems.

## 2. Explain the architecture of Kubernetes. Mention each component’s role.

Kubernetes architecture includes:  
- Master Node:  
 - API Server: Frontend for Kubernetes control plane.  
 - Scheduler: Assigns workloads to nodes.  
 - Controller Manager: Maintains cluster state.  
 - etcd: Key-value store for configuration data.  
- Worker Node:  
 - Kubelet: Communicates with master, manages containers.  
 - Kube-proxy: Handles networking and load balancing.  
 - Container Runtime: Runs containers (e.g., Docker, containerd).

## 3. What are Services in Kubernetes? Explain.

Services in Kubernetes expose pods to other pods or external systems. Types include:  
- ClusterIP: Default, accessible within cluster.  
- NodePort: Exposes service on each node’s IP.  
- LoadBalancer: Uses cloud provider’s load balancer.  
- ExternalName: Maps service to external DNS name.  
Services enable stable networking and load balancing for dynamic pod IPs.

## 4. What is a Namespace? What is its role?

Namespaces in Kubernetes provide a way to divide cluster resources between multiple users. They are useful for:  
- Environment separation (dev, test, prod).  
- Resource isolation and quota management.  
- Access control via RBAC.  
Default namespaces include default, kube-system, and kube-public.

## 5. What is Autoscaling and its types? When can we use vertical scaling?

Autoscaling adjusts resources based on demand. Types:  
- Horizontal Pod Autoscaler (HPA): Scales pod count based on CPU/memory.  
- Vertical Pod Autoscaler (VPA): Adjusts pod resource limits.  
- Cluster Autoscaler: Adds/removes nodes.  
Vertical scaling is used when a pod needs more CPU/memory but scaling out is not feasible.

## 6. What is the difference between StatefulSet and Deployment?

Deployment manages stateless applications with interchangeable pods. StatefulSet manages stateful applications with persistent identity and storage.  
Use Deployment for web servers, APIs. Use StatefulSet for databases, queues.

## 7. Difference between StatefulSet, DaemonSet, and ReplicaSet. Explain use cases of each with real-world examples.

- StatefulSet: Maintains identity and storage. Example: MongoDB cluster.  
- DaemonSet: Ensures one pod per node. Example: Log collector like Fluentd.  
- ReplicaSet: Ensures specified number of pod replicas. Example: Backend API service.

## 8. Write a YAML file for a simple nginx pod.

apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx  
 ports:  
 - containerPort: 80

## 9. Write an imperative command to create a deployment with image nginx and replica count of 3.

kubectl create deployment nginx-deployment --image=nginx --replicas=3

## 10. What does node affinity do? Mention its rules and what it does.

Node affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on labels on nodes. Types:  
- RequiredDuringSchedulingIgnoredDuringExecution: Mandatory rules.  
- PreferredDuringSchedulingIgnoredDuringExecution: Preferred rules.  
It helps in placing pods on specific nodes for performance, compliance, or hardware needs.

## 11. What is the difference between CMD and ENTRYPOINT?

CMD provides default arguments for the ENTRYPOINT or the container. ENTRYPOINT sets the command to run and is not overridden by Docker run arguments.  
Use ENTRYPOINT for fixed commands and CMD for default parameters.

## 12. Where are Docker volumes stored?

Docker volumes are stored in the host filesystem under /var/lib/docker/volumes by default.

## 13. What is the difference between COPY and ADD?

COPY copies files/directories from source to destination. ADD does the same but also supports remote URLs and auto-extraction of tar archives.

## 14. How many containers can we run in Docker exactly?

There is no hard limit; it depends on system resources like CPU, memory, and disk. Docker can run thousands of containers on powerful hardware.

## 15. What happens to the data inside a container when you delete the running container?

If the data is not stored in a volume or bind mount, it is lost when the container is deleted. Volumes persist data beyond container lifecycle.