Docker Interview Cheatsheet

55 Essential Questions & Answers
Quick Reference Guide for 2026

✓ 55 Real-World Interview Questions
✓ Quick-Reference Commands
✓ Troubleshooting Scenarios
✓ Best Practices Checklist
✓ Architecture Diagrams
✓ Interview Preparation Timeline

Updated: February 2026

Table of Contents

1. Beginner Questions (1-15)

Q1. What is Docker?

Docker is a containerization platform that packages applications and dependencies into lightweight, portable containers that run consistently across environments.

Q2. Difference between image and container?

An image is a read-only template (blueprint), while a container is a running instance of that image with its own writable layer.

Q3. How to create a Docker image?

Create a Dockerfile with build instructions, then run: `docker build -t image-name .`

Q4. What is a Dockerfile?

A text file containing instructions (FROM, RUN, COPY, CMD, etc.) that Docker uses to build an image automatically.

Q5. Explain Docker architecture

Docker uses client-server architecture: Client (CLI) sends commands to Daemon (server) which manages containers, images, networks. Registry stores images.

Q6. CMD vs ENTRYPOINT?

CMD provides default command (can be overridden). ENTRYPOINT defines the main executable (arguments appended). Often used together: ENTRYPOINT for program, CMD for default args.

Q7. What are Docker volumes?

Volumes are Docker's mechanism for persisting data outside containers. They survive container deletion and can be shared between containers.

Q8. How to view container logs?

`docker logs [container-name]`. Use `-f` to follow, `--tail N` for last N lines.

Q9. What is Docker Compose?

Tool for defining and running multi-container applications using YAML file. Start entire stack with `docker compose up`.

Q10. List running containers?

`docker ps` (running only), `docker ps -a` (all containers including stopped).

Q11. COPY vs ADD in Dockerfile?

COPY simply copies files. ADD can extract tar files and download from URLs. Best practice: use COPY unless you need ADD's extra features.

Q12. Stop and remove container?

`docker stop [name]` then `docker rm [name]`. Or `docker rm -f [name]` to force-remove running container.

Q13. What is Docker Hub?

Default public registry for Docker images. Hosts official images and community images. Like GitHub for containers.

Q14. Docker networking basics?

Bridge (default, private network), Host (use host network), Overlay (multi-host), None (no network). Custom bridge networks enable DNS resolution.

Q15. How to expose ports?

In Dockerfile: `EXPOSE 8080` (documentation). At runtime: `-p 8080:8080` (actual mapping). Format: `host-port:container-port`

2. Intermediate Questions (16-35)

Q16. Optimize Docker image size?

Use Alpine base images, multi-stage builds, combine RUN commands, use .dockerignore, order layers by change frequency.

Q17. What are Docker layers?

Each Dockerfile instruction creates a read-only layer. Layers are cached and reused. Changed layer invalidates cache for all subsequent layers.

Q18. docker run vs docker start?

`docker run` creates new container from image. `docker start` starts existing stopped container.

Q19. Handle environment variables?

Runtime: `-e VAR=value` or `--env-file .env`. Dockerfile: `ENV VAR=value`. Never hardcode secrets in images.

Q20. Volume vs bind mount?

Volumes: Docker-managed, portable, for production. Bind mounts: map host path, for development (live reload). tmpfs: in-memory, temporary.

Q21. Debug failing container?

1) Check logs 2) `docker inspect` 3) `docker exec -it` to enter 4) Override entrypoint 5) Check resource limits 6) Review events

Q22. Multi-stage builds?

Multiple FROM statements in Dockerfile. Build in one stage, copy artifacts to minimal runtime stage. Reduces final image size dramatically.

Q23. Resource limits in Docker?

Memory: `-m 512m`. CPU: `--cpus='1.5'`. Prevents containers from monopolizing host resources.

Q24. Copy-on-write strategy?

Containers share read-only image layers. When modifying a file, Docker copies it to container's writable layer first, then modifies.

Q25. Docker network drivers?

bridge (default, single-host), host (no isolation), overlay (multi-host Swarm), macvlan (physical network), none (isolated).

Q26. Dockerfile HEALTHCHECK?

Tests if container is functional: `HEALTHCHECK CMD curl -f http://localhost/health || exit 1`. Checks every interval, marks unhealthy after retries.

Q27. Share data between containers?

Use shared named volume: `docker volume create shared`, mount to multiple containers with `-v shared:/data`.

Q28. docker pause vs stop?

pause: Freeze processes (cgroups), stays in memory, instant resume. stop: Graceful shutdown (SIGTERM), then SIGKILL after timeout.

Q29. Clean up Docker resources?

`docker system prune -a` (all unused). Or targeted: `container/image/volume/network prune`.

Q30. Docker caching for builds?

Put stable instructions first (dependencies), changing code last. Use .dockerignore. Separate dependency install from code copy.

Q31. Docker secrets?

Encrypted data for Swarm services. Create: `docker secret create`. Appears in container at `/run/secrets/`. Never use ENV for secrets.

Q32. Zero-downtime updates?

Rolling updates (orchestrator), blue-green deployment (run both versions), health checks, graceful shutdown, connection draining.

Q33. Public vs private registry?

Public: Anyone can pull (Docker Hub). Private: Requires auth (ECR, GCR, ACR, Harbor). Always use private for production/proprietary code.

Q34. Monitor containers in production?

Prometheus + Grafana for metrics, ELK/Loki for logs, health checks, cAdvisor for container stats. Track CPU, memory, restarts, network I/O.

Q35. Registry mirroring?

Local cache of Docker Hub/registries. Faster pulls, reduced bandwidth, resilience against outages, compliance. Set in daemon.json.

3. Advanced Questions (36-55)

Q36. Docker storage drivers?

overlay2 (recommended, default), devicemapper (legacy), btrfs/zfs (snapshots), vfs (slow, no CoW). Use overlay2 on ext4/xfs.

Q37. Container security best practices?

Scan images, minimal base images, run as non-root, drop capabilities, read-only filesystem, secrets management, user namespaces, regular updates.

Q38. Docker Swarm vs Kubernetes?

Swarm: Simple, Docker-native, smaller scale. K8s: Industry standard, complex, massive ecosystem, better for large-scale. K8s has won for enterprise.

Q39. Persistent storage for databases?

Named volumes in production, never container layer. Cloud: EBS/Persistent Disks. Backup regularly, test restores, backward-compatible migrations.

Q40. Docker namespace isolation?

Linux namespaces isolate: PID (processes), Network (IP/ports), Mount (filesystem), UTS (hostname), IPC (shared memory), User (UIDs).

Q41. CI/CD with Docker?

Build image with version tag, run tests in container, scan vulnerabilities, push to registry, deploy with orchestrator. Tag with git SHA for rollbacks.

Q42. What are cgroups?

Linux kernel feature limiting container resources (CPU, memory, I/O). Docker uses cgroups to enforce limits, prevent resource starvation.

Q43. Troubleshoot container networking?

1) Verify same network 2) Test connectivity (ping/curl) 3) Check DNS 4) Verify ports 5) Check firewall 6) Review network mode 7) Packet capture

Q44. Docker logging drivers?

json-file (default), syslog, journald, gelf, fluentd, awslogs. Configure in daemon.json. Set rotation to prevent disk fill.

Q45. Graceful shutdown in containers?

Handle SIGTERM signal, stop accepting requests, finish current work, close connections, timeout as fallback. Use exec form in CMD.

Q46. Docker build context?

All files sent to daemon for build. Use .dockerignore to exclude. Large contexts slow builds. Copy specific files, not entire directory.

Q47. tmpfs mount use cases?

In-memory storage for: temp files, caches, session data, sensitive data that shouldn't touch disk. Fast (RAM speed), ephemeral.

Q48. Handle image vulnerabilities?

Scan with Trivy/Snyk/Scout. Block CRITICAL/HIGH in CI/CD. Update base images regularly, use minimal images, multi-stage builds.

Q49. Docker IPv6 support?

Enable in daemon.json: `ipv6: true`. Create IPv6 network. Most production runs dual-stack or IPv4 still.

Q50. Health checks: Docker vs K8s?

Docker: Single HEALTHCHECK. K8s: Liveness (restart if fails), Readiness (traffic control), Startup (slow apps). K8s more sophisticated.

Q51. What is BuildKit?

Next-gen build engine. Parallel builds, cache mounts, secret mounting, SSH access, better caching, cross-platform. Default since Docker 23.

Q52. Handle timezones in containers?

Set TZ env var or mount /etc/localtime. Best practice: Keep containers UTC, handle timezone in app logic.

Q53. User namespace remapping?

Maps container root to non-root host UID. Major security improvement. Enable in daemon.json. Reduces container escape impact.

Q54. Zero-downtime deployment strategy?

Health checks, rolling updates, multiple replicas, graceful shutdown, connection draining, backward-compatible DB changes, feature flags.

Q55. Docker Scout and Extensions?

Scout: Security/CVE scanning, SBOM, policy enforcement. Extensions: Add tools to Docker Desktop (Portainer, Trivy, Snyk, etc.).

Quick Command Reference

Container Management

`docker run -d --name app -p 8080:80 nginx`	Run container detached with port mapping
`docker ps / docker ps -a`	List running / all containers
`docker stop [name] / docker start [name]`	Stop / start container
`docker restart [name]`	Restart container
`docker rm [name] / docker rm -f [name]`	Remove container (force if running)
`docker exec -it [name] bash`	Execute command in running container
`docker logs -f --tail 100 [name]`	Follow logs, last 100 lines
`docker inspect [name]`	Detailed container info

Image Management

`docker build -t image:tag .`	Build image from Dockerfile
`docker images / docker image ls`	List images
`docker pull image:tag`	Download image from registry
`docker push image:tag`	Upload image to registry
`docker rmi image:tag`	Remove image
`docker tag source:tag target:tag`	Tag image
`docker image prune -a`	Remove all unused images

Volume & Network

`docker volume create myvolume`	Create volume
`docker volume ls / docker volume rm`	List / remove volumes
`docker network create mynetwork`	Create network
`docker network ls`	List networks
`docker network inspect bridge`	Inspect network

Docker Compose

`docker compose up -d`	Start services in background
`docker compose down`	Stop and remove containers

`docker compose logs -f`	Follow logs
`docker compose ps`	List containers
`docker compose restart`	Restart services

System Commands

`docker system df`	Show disk usage
`docker system prune -a --volumes`	Clean everything
`docker stats`	Live resource usage
`docker version / docker info`	Version / system info

Common Troubleshooting Scenarios

Problem: Container keeps restarting

Solutions:

- Check logs: docker logs [container]
- Inspect exit code: docker inspect --format='{{.State.ExitCode}}'
- Verify health check isn't failing
- Check resource limits (OOM killer?)
- Ensure required services are available
- Override entrypoint for debugging

Problem: Cannot connect to container

Solutions:

- Verify port mapping: docker port [container]
- Check if app binds to 0.0.0.0 (not 127.0.0.1)
- Test from host: curl localhost:port
- Verify firewall rules (iptables)
- Check network mode and DNS resolution
- Ensure container is on correct network

Problem: Build fails with cache issues

Solutions:

- Clear build cache: docker builder prune
- Force rebuild: docker build --no-cache
- Check .dockerignore is working
- Verify layer ordering (dependencies first)
- Ensure sufficient disk space
- Check for file permission issues

Problem: Disk space full

Solutions:

- Check usage: docker system df
- Prune unused: docker system prune -a --volumes
- Remove specific: container/image/volume prune
- Set up log rotation in daemon.json
- Monitor with: df -h /var/lib/docker
- Consider moving Docker root to larger disk

Problem: Performance degradation

Solutions:

- Check resource usage: docker stats

- Review storage driver (use overlay2)
- Inspect container limits: docker inspect
- Optimize image layers and size
- Check host resources (CPU, memory, I/O)
- Review networking overhead

Problem: Permission denied errors

Solutions:

- Run container with correct user: --user
- Check volume permissions: ls -la
- Use user namespace remapping
- Avoid running as root in container
- Adjust host directory ownership
- Check SELinux/AppArmor policies

Best Practices Checklist

Image Building

- ■ Use official base images from verified publishers
- ■ Use specific version tags, not :latest
- ■ Implement multi-stage builds for production
- ■ Minimize layers by combining RUN commands
- ■ Use .dockerignore to exclude unnecessary files
- ■ Order Dockerfile instructions by change frequency
- ■ Use Alpine or distroless for smaller images
- ■ Scan images for vulnerabilities regularly

Security

- ■ Never run containers as root (USER directive)
- ■ Use secrets management, not ENV for sensitive data
- ■ Implement health checks in Dockerfile
- ■ Drop unnecessary capabilities (--cap-drop)
- ■ Use read-only filesystem where possible
- ■ Scan for CVEs in CI/CD pipeline
- ■ Keep base images and dependencies updated
- ■ Enable user namespace remapping

Production Deployment

- ■ Always set resource limits (CPU, memory)
- ■ Use named volumes for persistent data
- ■ Implement proper logging strategy
- ■ Set up container orchestration (K8s/Swarm)
- ■ Use private registries for images
- ■ Implement blue-green or rolling deployments
- ■ Configure restart policies appropriately
- ■ Monitor container metrics and logs

Development

- ■ Use Docker Compose for local development
- ■ Leverage BuildKit for faster builds
- ■ Use bind mounts for live code reload
- ■ Create custom networks for service isolation
- ■ Document environment variables
- ■ Version control Dockerfile and compose files
- ■ Test builds in CI before deployment
- ■ Use consistent naming conventions

Docker Architecture Diagrams

Docker Client-Server Architecture

```
■■■■■■■■■■■■■■■■■■■■■■ ■ Docker Client ■ ← (docker CLI) ■ (docker run) ■
■■■■■■■■■■■■■■■■■■■■■■ ■ REST API ▼ ■■■■■■■■■■■■■■■■■■■■■■ ■ Docker Daemon ■ ←
(dockerd) ■ - Containers ■ ■ - Images ■ ■ - Networks ■ ■ - Volumes ■
■■■■■■■■■■■■■■■■■■■■■■ ■ ▼ ■■■■■■■■■■■■■■■■■■■■■■ ■ Docker Registry ■ ← (Docker
Hub, ECR) ■ - Image Store ■ ■■■■■■■■■■■■■■■■■■■■
```

Container Layer Architecture

```
■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Container Layer ■ ← Writable (ephemeral)
■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Image Layer 3 (APP) ■ ← Read-only
■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Image Layer 2 (DEPS) ■ ← Read-only (cached)
■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Image Layer 1 (BASE) ■ ← Read-only (cached)
■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Docker Networking Types

```
Bridge Network (default): Container A ■■ ■■ Bridge ■■■ Host ■■■ External Container B
■■ Host Network: Container ■■■ Host Network (no isolation) Overlay Network (Swarm): Node
1: Container A ■■ ■■ Overlay ■■■ External Node 2: Container B ■■
```

Interview Preparation Timeline

A 2-week preparation plan to ace your Docker interview:

Week 1: Foundations

- **Days 1-2:** Master basics: containers, images, Dockerfile, common commands
- **Days 3-4:** Deep dive: Networking, volumes, Docker Compose
- **Days 5-6:** Practice: Build multi-container apps, troubleshoot issues
- **Day 7:** Review: Answer all beginner questions without looking

Week 2: Advanced & Practice

- **Days 8-9:** Study: Security, optimization, production best practices
- **Days 10-11:** Advanced topics: CI/CD, orchestration, monitoring
- **Day 12:** Mock interviews: Practice explaining concepts out loud
- **Day 13:** Review cheatsheet, practice troubleshooting scenarios
- **Day 14:** Final review: Focus on your weak areas, relax

Pro Tips for Interview Day:

1. Always explain the 'why' behind your answers, not just 'what'
2. Mention real-world experience if you have it
3. Be honest about what you don't know - explain how you'd find out
4. Use the STAR method for behavioral questions
5. Ask clarifying questions before diving into technical answers
6. Keep answers concise but complete - practice 2-3 minute responses
7. Bring up security and best practices when relevant
8. Have 2-3 questions ready to ask your interviewer