Docker FAQ for Interviews

Basic Questions

Basic Questions

1. What is Docker?
Docker is a containerization platform that allows developers to build, package, ship, and run applications in containers. Containers ensure consistency across environments, are lightweight, portable, and faster compared to traditional virtual machines.

---

2. What is a Docker container?
A Docker container is a lightweight, standalone, and executable unit that includes everything required to run an application — application code, runtime, system tools, libraries, and configuration files. Containers run in isolated environments but share the host operating system kernel.

---

3. What is a Docker image?
A Docker image is a read-only template used to create containers. It contains application code, dependencies, environment settings, and instructions required to run the container.

---

4. Difference between a Docker image and a Docker container?

- Docker Image: A static, read-only blueprint for creating containers. Can be versioned and shared.
- Docker Container: A running instance of an image. It can be started, stopped, moved, or deleted and only exists while running.

---

5. What is Dockerfile?
A Dockerfile is a script-like text file containing a set of instructions to build a Docker image. It defines the base image, dependencies, environment setup, files to copy, ports to expose, and the default command to run when the container starts.

---

6. How is Docker different from a virtual machine (VM)?

- Docker: Uses the host OS kernel, lightweight, faster to start, and shares resources efficiently.
- VM: Each VM runs its own OS, heavier, resource-intensive, and slower to boot compared to containers.

---

7. What is Docker Hub?
Docker Hub is a public cloud-based registry where developers can publish, share, and download

Docker images. It acts as the default central repository for Docker images but private registries can also be used.

Intermediate Questions

8. How do you manage persistent data in Docker?
Persistent data in Docker can be managed using:

- Volumes → Managed by Docker, stored outside the container lifecycle, recommended for most use cases.
- Bind mounts → Directly use files/directories from the host machine.

---

9. How do you link containers?
Containers can communicate through Docker networks such as:

- Bridge network (default for single-host communication).
- Overlay network (multi-host, used in Swarm).
  You can create custom networks using `docker network create` and attach containers using the `--network` option.

---

10. How do you optimize Docker images?

- Use minimal base images like Alpine Linux.
- Combine multiple commands into fewer RUN instructions to reduce layers.
- Use a `.dockerignore` file to exclude unnecessary files.
- Avoid installing unused dependencies.

---

11. Explain Docker Compose.
Docker Compose is a tool used to define and manage multi-container applications. It uses a docker-compose.yml file to specify services, networks, and volumes. With a single command (`docker-compose up`), you can bring up the entire stack.

---

12. Difference between CMD and ENTRYPOINT in Dockerfile?

- CMD: Specifies default command/arguments for a container. It can be overridden at runtime.
- ENTRYPOINT: Specifies the main executable that always runs; arguments can be passed during container startup.

---

13. What is a multi-stage build?
Multi-stage builds allow you to create smaller, optimized Docker images by using multiple FROM statements in the Dockerfile. You build in one stage (e.g., compile code) and then copy only the required artifacts into the final lightweight image.

Advanced Questions

14. How does Docker networking work?
Docker networking enables container communication:

- Bridge network → Default single-host container communication.
- Host network → Shares the host's network namespace (no isolation).
- Overlay network → Enables communication between containers across multiple Docker hosts (used in Swarm).

---

15. How do you handle secrets in Docker?

- Docker secrets (Swarm mode) → Securely store and share sensitive information like passwords, tokens, or certificates.
- Environment variables or bind mounts can also be used, but they are less secure.

---

16. How do you monitor Docker containers?

- Built-in: `docker stats` command provides real-time container metrics.
- External tools: Prometheus, Grafana, ELK Stack, Datadog, cAdvisor, Sysdig for detailed monitoring, logging, and alerting.

---

17. How do you scale containers?

- Docker Compose: Use `docker-compose up --scale service=n` to scale specific services.
- Docker Swarm: Use `docker service scale service_name=n` to scale services across multiple nodes.

---

18. What is a Docker registry?
A Docker registry is a system for storing and distributing Docker images. Examples include:

- Public: Docker Hub (default).
- Private: Harbor, AWS Elastic Container Registry (ECR), Azure Container Registry (ACR), Google Artifact Registry.

---

19. What is the difference between COPY and ADD in Dockerfile?

- COPY: Copies files/folders from host to container.
- ADD: Works like COPY but also supports extracting compressed files (tar) and fetching files from URLs.

---

20. How do you reduce image size for production?

- Use multi-stage builds.
- Choose minimal base images (Alpine, BusyBox).
- Remove unnecessary packages and cache.
- Use `.dockerignore` to exclude unwanted files.

==

21. How is Docker different from Kubernetes?
Docker is a containerization platform that runs single or multiple containers. Kubernetes is an orchestration platform that manages the deployment, scaling, and networking of containers across multiple hosts.

---

22. Can you explain container lifecycle commands?

- `docker create` – creates a container without starting it
- `docker run` – creates and starts a container
- `docker start` / `docker stop` – starts or stops a container
- `docker restart` – restarts a container
- `docker rm` – removes a container

---

23. How does Docker ensure process isolation?
Docker uses Linux namespaces for process, network, and mount isolation. It also uses cgroups to limit resource usage such as CPU, memory, and I/O, ensuring containers cannot interfere with each other.

---

24. How do you monitor Docker container performance in production?
You can monitor containers using `docker stats` for real-time metrics. For more advanced monitoring, integrate tools like Prometheus, Grafana, cAdvisor, or Datadog to track CPU, memory, network, and disk usage.

---

25. How do you perform rolling updates with Docker Swarm?
Use `docker service update --image <new_image> <service_name>` to update

services. Control the update process with flags such as `--update-parallelism` and `--update-delay` to manage how many containers update at a time and the interval between updates.

---

26. How can you reduce security vulnerabilities in Docker images?

- Use minimal base images such as Alpine or BusyBox
- Regularly update images and dependencies
- Scan images using tools like Trivy, Clair, or Anchore to detect vulnerabilities

---

27. What are Docker namespaces and how do they ensure isolation?
Namespaces provide process, network, mount, and PID isolation. For example, PID namespaces isolate process IDs, while network namespaces isolate container networking, ensuring containers operate independently.

---

28. Explain cgroups in Docker.
Control Groups (cgroups) limit and monitor CPU, memory, and I/O usage for containers. They ensure that no single container can consume all the host resources and affect other containers.

---

29. How do you debug network issues between Docker containers?
Check existing networks using `docker network ls` and `docker network inspect`. Use `ping` or `curl` from one container to another. Verify firewall rules, port exposure, and DNS resolution to ensure connectivity.

---

30. How do you handle image cleanup and prevent disk space issues?
Use `docker system prune` to remove unused images, containers, volumes, and networks. Employ image tagging and automated CI/CD cleanup policies to manage disk space proactively.

---

31. How do you connect Docker containers to a database securely?
Use environment variables or Docker secrets to manage database credentials. Connect containers via internal networks rather than exposing ports publicly. Ensure the database is secured with access control and authentication.

---

32. How can you handle multi-container dependencies in Docker Compose?
Use `depends_on` in `docker-compose.yml` to define the start order of services. Combine this with healthchecks to ensure dependent services are fully ready before other services start.

---

33. What is the difference between Docker Swarm mode and standalone Docker?
Swarm mode provides native clustering, service discovery, and load balancing across multiple hosts. Standalone Docker runs containers on a single host without orchestration features.

---

34. How do you handle container restarts automatically?
Use the `--restart` flag (options: always, on-failure, unless-stopped) during `docker run` or in Docker Compose files to automatically restart containers on failure or system reboot.

---

35. Explain overlay networks in Docker.
Overlay networks enable multi-host container communication in Swarm clusters. They are built on top of host networks using VXLAN encapsulation to allow containers on different hosts to communicate securely.

---

36. How can you reduce Docker image build time in CI/CD pipelines?
Use layer caching effectively, implement multi-stage builds, and avoid including unnecessary files and dependencies in the image to speed up the build process.

---

37. How do you integrate Docker with CI/CD pipelines?
Build Docker images in the CI/CD pipeline using Docker commands. Push images to a registry like Docker Hub, ECR, or ACR. Deploy the images to staging or production environments using Docker Compose, Swarm, or Kubernetes.

---

38. How do you handle logging for multiple containers in production?
Use centralized logging solutions like ELK Stack, Fluentd, or Splunk to aggregate logs. Configure log rotation and retention policies to prevent disk space issues and ensure long-term monitoring.

---

39. How do you roll back a failed deployment in Docker Swarm?
Use `docker service update --rollback` to revert to the previous image. Monitor service health to detect failures early and automatically trigger rollback if necessary.

---

40. What is the difference between docker exec and docker attach?

- `docker exec` runs a new command inside a running container without affecting the main process.
- `docker attach` connects to the main process of a running container, sharing its input/output streams.

---

41. How do you manage Docker images across multiple environments?
Use tagging conventions such as dev, test, and prod. Use private registries for controlled image distribution. Automate image builds and pushes via CI/CD pipelines.

---

42. How can Docker help in microservices architecture?
Each microservice runs in its own container, which is isolated and portable. Docker simplifies deployment, scaling, and versioning of microservices. It works well with orchestration tools like Kubernetes for service management.

---

43. How do you secure inter-container communication?
Use encrypted overlay networks in Swarm. Limit network scope to only necessary containers. Avoid exposing container ports publicly unless required.

---

44. Scenario: You need to deploy a multi-tier web application using Docker. How would you organize containers?
Use Docker Compose to define services for frontend, backend, and database. Define separate networks for communication. Use volumes for persistent database storage. Set dependencies using `depends_on` and healthchecks.

---

45. Scenario: A container keeps restarting in production. How would you troubleshoot?
Check container logs using `docker logs <container>`. Inspect the container status with `docker inspect <container>`. Verify resource limits, dependencies, and environment variables. Temporarily disable the restart policy to debug.

---

46. Scenario: You need to run a containerized application across multiple hosts. How would you handle networking?
Use overlay networks for multi-host communication. Ensure all hosts are part of a Swarm or Kubernetes cluster. Use service discovery (DNS-based) for container-to-container communication.

---

47. Scenario: You want to reduce the image size of a Python application. What steps would you take?
Use a lightweight base image such as Python-Alpine. Use multi-stage builds to separate build and runtime dependencies. Remove unnecessary files and cache, including pip cache and logs.

---

48. Scenario: Your application requires sensitive environment variables in containers. How do you handle this securely?
Use Docker secrets (Swarm mode). Avoid hardcoding variables in Dockerfiles or Compose files. Store secrets in environment files or external secret management tools.

---

49. Scenario: A containerized service depends on a database that may start slowly. How do you ensure reliability?
Use healthchecks in Compose or Dockerfile. Combine with `depends_on` along with retries in scripts or entrypoints. Implement graceful retry logic in the application itself.

---

50. Scenario: You need zero-downtime deployment of a critical service. How would you achieve this?
Use blue-green deployment with two versions of the service. Switch traffic gradually using load balancers or service routing. Keep the old version until the new version is verified.

---

51. Scenario: You notice high CPU usage in one container. How would you limit resources?
Set CPU limits with `--cpus` or `--cpu-shares`. Monitor usage using `docker stats`. Optimize application code or split the workload into multiple containers.

---

52. Scenario: You need to migrate containers from one host to another without downtime. How?
Use Swarm mode or Kubernetes for orchestrated deployment. Pull images on the new host and start containers there. Gradually switch traffic using a load balancer or DNS.

---

53. Scenario: You want to version and rollback container images in CI/CD. What strategy would you use?
Use semantic versioning for Docker images. Keep previous versions in a registry. Use automated deployment scripts to rollback by pulling previous versions.

---

54. Scenario: Your containerized application requires persistent logging. How would you handle it?
Use Docker volumes or bind mounts to store logs outside containers. Integrate with centralized logging tools such as ELK or Fluentd. Implement log rotation to prevent disk exhaustion.

---

55. Scenario: You have multiple containers that need to communicate securely over a public network. What would you do?
Use encrypted overlay networks. Limit container ports to the internal network whenever possible. Use TLS certificates for inter-service communication.

---

56. Scenario: A container image build fails in CI/CD but works locally. How do you debug?
Compare the environment differences between local and CI/CD. Check for caching issues or missing build dependencies. Use verbose build logs and temporarily disable caching.

---

57. Scenario: You want to ensure all deployed images are free from vulnerabilities. How would you implement this?
Use image scanning tools like Trivy, Clair, or Anchore. Integrate scanning into CI/CD pipelines before deployment. Automate alerts for new vulnerabilities in base images.

---

58. Scenario: You need to deploy a stateful application like a database in Docker Swarm. How would you handle persistence and scaling?
Use Docker volumes or external storage (NFS, cloud volumes). Limit scaling to avoid data corruption. Backup volumes regularly and restore in case of failure.

---

59. Scenario: You have a monolithic application that you want to containerize and migrate to microservices. How would you approach this?
Identify modules and separate them into individual services. Containerize each service with its own Dockerfile. Use Docker Compose or Kubernetes for orchestration. Ensure proper networking, logging, and monitoring between services.

---

60. Scenario: Your containerized app crashes intermittently with "out of memory" errors. How would you troubleshoot and fix it?
Check resource limits with `docker stats`. Increase memory limits using `--memory`. Optimize application memory usage or implement caching. Use smaller, optimized base images to reduce footprint.

---

61. Scenario: You want to run multiple versions of the same service for testing. How would you manage this?
Tag Docker images with version numbers. Use separate Compose services or different container names. Use environment variables to configure ports, database connections, or other dependencies.

---

62. Scenario: Your CI/CD pipeline builds images frequently. Disk space is running out on the Docker host. What would you do?
Implement automated image cleanup using `docker system prune -a`. Use registry retention policies to remove old images. Use multi-stage builds to reduce final image size.

---

63. Scenario: You need to run a container with real-time updates to a database on the host. How would you handle data persistence?
Use bind mounts or Docker volumes to persist database files. Ensure proper permissions for host directories. Back up volume data regularly.

---

64. Scenario: You are deploying multiple containers on different hosts, and service discovery fails. How do you troubleshoot?
Check overlay network configuration. Verify Swarm nodes are properly joined. Inspect DNS

resolution between containers using `docker exec`. Ensure firewall rules allow container communication.

---

65. Scenario: You want to implement automatic failover for a containerized service. How would you achieve this?
Deploy multiple replicas using Swarm or Kubernetes. Use healthchecks to detect failures. Use load balancers to route traffic to healthy containers.

---

66. Scenario: You have a legacy app that must run with root privileges, but you want to enhance security. How would you handle it?
Limit container capabilities using `--cap-drop`. Use user namespaces to map root to non-root host users. Run only the required processes as root and everything else as non-root.

---

67. Scenario: You want to monitor container logs and metrics across multiple hosts. How would you implement this?
Centralize logs using ELK Stack, Fluentd, or Splunk. Collect metrics using cAdvisor, Prometheus, and Grafana. Set up alerts for critical thresholds.

---

68. Scenario: You want to deploy a stateless service globally across multiple cloud regions. How would Docker help?
Build a container image once and deploy it to all regions. Use orchestrators like Kubernetes or Swarm for cluster management. Use load balancers or DNS routing for global traffic management.

---

69. Scenario: Your containerized service depends on external APIs. How do you ensure reliability?
Implement retry logic in the application. Use healthchecks to detect API failures. Use a circuit breaker pattern for graceful degradation.

===

70. Scenario: You need to ensure that containers are reproducible across environments (dev, test, prod). How?
Answer:

- Use **Docker images** with pinned versions.
- Use consistent base images and dependency versions.
- Include environment-specific configs via `.env` files or secrets.

==

71. Scenario: You want to deploy a containerized application on a host without installing Docker. What options do you have?
Answer:

- Use **Docker-in-Docker (DinD)** in CI/CD pipelines.
- Use **Podman** as a daemonless alternative.
- Use **Kubernetes CRI (Container Runtime Interface)** with supported runtimes.

==

72. Scenario: You need to rollback a failed update in a production containerized environment. How would you do it?
Answer:

- Use Swarm: `docker service update --rollback`.
- Use Kubernetes: `kubectl rollout undo deployment <name>`.
- Keep previous container images tagged for easy rollback.

==

73. Scenario: You need to optimize Docker container startup time in a microservices environment. How?
Answer:

- Use **minimal base images**.
- Use **multi-stage builds** to reduce image size.
- Pre-pull images to hosts before deployment.
- Optimize application initialization scripts.

===

74. Scenario: You need to deploy a highly available database in Docker. How would you design it?
Answer:

- Use **stateful containers** with persistent volumes.
- Deploy multiple replicas with **leader-follower replication**.
- Use orchestrators (Swarm/Kubernetes) for failover and automated recovery.
- Backup volumes regularly and test restore procedures.

==

75. Scenario: You need to integrate Docker with Kubernetes for a CI/CD pipeline. How would you do it?
Answer:

- Build Docker images in CI pipeline and push to a container registry.
- Deploy images to Kubernetes using **kubectl** or **Helm charts**.

- Use rolling updates and healthchecks to ensure zero-downtime deployments.
- Automate rollback in case of deployment failures using Kubernetes deployment strategies.

===

76. Scenario:

Your production container fails after restart due to missing environment variables. How do you prevent this?
Answer:

- Store env vars in `.env` files or secret managers (Vault, AWS Secrets Manager).
- Use `--env-file` or Compose configs.
- Ensure CI/CD pipelines inject required variables at runtime.

---

77. Scenario:

You need to limit a container to **500MB RAM and 1 CPU core**. How do you enforce it?
Answer:

- Use resource flags: `--memory=500m --cpus=1`.
- Monitor with `docker stats`.
- Enforce resource governance in production orchestrators (Swarm/K8s).

---

78. Scenario:

A CI/CD pipeline rebuilds the same Docker image multiple times. How do you speed it up?
Answer:

- Use **Docker layer caching**.
- Reorder Dockerfile commands (dependencies before code).
- Cache package downloads (pip, npm, Maven).
- Use remote build cache (BuildKit + registry cache).

---

79. Scenario:

You need to allow containers to access the host system securely. How would you achieve this?
Answer:

- Use **bind mounts** for controlled directories.
- Use `--network host` cautiously.
- Apply **AppArmor/SELinux** to restrict access.

---

80. Scenario:

A containerized microservice needs to scale independently. How do you implement this?
Answer:

- Use Docker Compose or Swarm with `replicas`.
- Place microservices in separate Compose services.
- Implement autoscaling with Kubernetes HPA or external monitoring.

---

81. Scenario:

You want to enforce compliance by ensuring only signed images run in production. How do you do it?
Answer:

- Enable **Docker Content Trust (DCT)** for image signing.
- Use Notary or Cosign for signing/verifying.
- Block unsigned images in CI/CD and orchestrator policies.

---

82. Scenario:

Your Docker host is running out of inodes even though disk usage is low. How do you fix it?
Answer:

- Check for too many small files in `/var/lib/docker`.
- Clean up unused containers, images, and layers (`docker system prune`).
- Use overlay2 storage driver for efficiency.

---

83. Scenario:

You want to securely expose a containerized API over HTTPS. How do you handle SSL/TLS?
Answer:

- Use reverse proxies (Nginx, Traefik, Caddy) with SSL certificates.
- Automate certs with Let’s Encrypt.
- Avoid managing TLS inside app containers (use sidecars or ingress).

---

84. Scenario:

A container starts but immediately exits. How do you debug?
Answer:

- Check logs with `docker logs <id>`.
- Verify entrypoint and CMD configuration.
- Run in interactive mode (`docker run -it`).
- Ensure background daemons use `CMD ["sleep","infinity"]` or proper supervisors.

---

85. Scenario:

You need to run containers on an **air-gapped environment (no internet access)**. How do you handle images?
Answer:

- Export/import images with `docker save` and `docker load`.
- Set up an offline private registry.
- Mirror base images locally.

---

86. Scenario:

Your security team demands all Docker images be scanned for vulnerabilities before deployment. How do you implement this?
Answer:

- Integrate scanning tools (Trivy, Anchore, Clair) in CI/CD.
- Fail builds on high-severity vulnerabilities.
- Continuously rescan base images for new CVEs.

---

87. Scenario:

You want to run Docker in production without root privileges. How?
Answer:

- Use **rootless Docker** mode.
- Run with Podman (daemonless, rootless alternative).
- Use user namespaces for container isolation.

---

88. Scenario:

Your application in Docker is not handling timezone correctly. How do you fix it?
Answer:

- Pass timezone environment variables (`TZ=UTC`).
- Mount host timezone files (`/etc/localtime`).
- Use UTC in production for consistency.

---

89. Scenario:

How would you handle **data migration** when upgrading containerized databases?
Answer:

- Backup volumes before upgrade.
- Run migrations in separate migration containers.
- Test schema changes in staging before production rollout.

---

90. Scenario:

Your containerized service must process **large files (GBs)**. How do you optimize?
Answer:

- Use bind mounts for fast disk I/O.
- Avoid copying large files inside images.
- Stream processing instead of full file loads.
- Allocate proper memory and CPU.

---

91. Scenario:

How would you enforce **container immutability** in production?
Answer:

- Make containers stateless.
- Store state in external DBs/volumes.
- Rebuild and redeploy containers instead of modifying them live.

---

92. Scenario:

A team pushes untagged “latest” images. How do you enforce proper versioning?
Answer:

- Enforce **tagging policies** in CI/CD.
- Block “latest” in production.
- Use semantic versioning or Git commit SHAs.

---

93. Scenario:

You want to optimize **Docker for high-performance CI builds**. What would you do?
Answer:

- Use SSD storage for `/var/lib/docker`.
- Enable BuildKit for parallel builds.
- Cache dependencies in shared volumes.
- Run multi-stage builds for smaller images.

---

94. Scenario:

Your Docker Swarm cluster experiences network latency between nodes. How do you troubleshoot?
Answer:

- Check overlay network health.
- Verify MTU settings and VXLAN encapsulation.
- Use `docker service logs` and `docker network inspect`.
- Ensure firewalls allow Swarm ports.

---

95. Scenario:

How would you design a **multi-cloud container deployment** with Docker?
Answer:

- Store images in a multi-cloud registry (Harbor, JFrog).
- Use Terraform/Ansible for provisioning infra.
- Orchestrate with Kubernetes across regions.
- Implement DNS-based traffic routing.

---

96. Scenario:

Your containerized service needs **low-latency communication with GPUs**. How do you configure Docker?
Answer:

- Use `--gpus` flag with NVIDIA runtime.
- Ensure drivers match container CUDA versions.
- Use GPU monitoring tools for resource governance.

---

97. Scenario:

You need to run **1000+ containers** on a single host. What challenges and solutions?
Answer:

- Challenges: CPU/memory contention, inode exhaustion, networking limits.
- Solutions: Use Kubernetes/Swarm for scheduling, resource quotas, and autoscaling.
- Spread workloads across nodes.

---

98. Scenario:

Your security team requires **audit logging of all container activity**. How do you achieve this?
Answer:

- Enable Docker daemon auditing (`auditd`).
- Use centralized logging with Fluentd/ELK.
- Use eBPF-based monitoring for syscall tracing.

---

99. Scenario:

You want **zero-downtime upgrades** of Docker hosts. How do you do it?
Answer:

- Run in orchestrator (Swarm/K8s).
- Drain host before upgrade (`docker node update --availability drain`).
- Migrate workloads to other nodes.
- Automate host patching with rolling updates.

---

100. Scenario:

Your company wants to move from **Docker Swarm to Kubernetes**. How would you plan migration?
Answer:

- Containerize workloads with Kubernetes-compatible configs.
- Convert `docker-compose.yml` to Kubernetes manifests (Kompose, Helm).

- Migrate persistent storage and secrets.
- Run hybrid Swarm + K8s during transition.

101. Scenario: Your Docker builds are non-deterministic — the same Dockerfile sometimes produces slightly different images. How do you fix this?

Answer:

- Pin dependency versions (apt-get, pip, npm).
- Use digest-based base images instead of floating tags.
- Leverage Docker BuildKit for reproducible builds.
- Store build artifacts in cacheable layers.

---

102. Scenario: You need to run containers with different kernel versions. How do you achieve this?

Answer:

- Docker itself cannot emulate kernel versions.
- Use VMs (with different kernels) running Docker inside.
- Alternative: Use Kubernetes with mixed OS node pools.

---

103. Scenario: A container's clock is drifting compared to the host. What do you do?

Answer:

- Sync host with NTP service.
- Mount `/etc/localtime` and `/etc/timezone` inside containers.
- Use `--privileged` + `hwclock` if container needs hardware clock sync.

---

104. Scenario: You want to prevent developers from running privileged containers in production. How do you enforce it?

Answer:

- Use Docker daemon security options (`--userns-remap`).
- Apply admission controllers in Kubernetes.
- Enforce PodSecurityPolicies/OPA Gatekeeper for non-root rules.
- Audit logs for `--privileged` usage.

---

105. Scenario: Your containerized Java app shows memory leaks. How do you distinguish between app leaks and Docker cgroup limits?

Answer:

- Compare JVM heap usage inside container vs. `docker stats`.
- Enable JVM flags (`-XX:+UseContainerSupport`).
- If OOM matches cgroup limits → increase memory.
- If JVM heap keeps growing → fix app leak.

---

106. Scenario: A containerized service must pass compliance (PCI DSS, HIPAA). How do you harden Docker for compliance?

Answer:

- Run CIS Docker Bench scans.
- Restrict root privileges.
- Use signed and verified images only.
- Encrypt data at rest and in transit.
- Ensure audit logging for all container activity.

---

107. Scenario: You notice container startup latency due to huge images. How do you reduce startup delays?

Answer:

- Use **multi-stage builds**.
- Store dependencies in shared base layers.
- Pre-pull images to nodes.
- Use slim or Alpine base images.

---

108. Scenario: You want to run Docker containers across ARM & x86 architectures. How do you build images?

Answer:

- Use `docker buildx` with `--platform` flag.
- Push multi-arch manifests to registry.
- Test on both architectures (QEMU for emulation).

---

109. Scenario: A production container becomes unresponsive but logs show nothing. How do you debug?

Answer:

- Use `docker exec` with debugging tools (bash, strace, tcpdump).
- Inspect resource limits (`docker inspect`).
- Capture syscalls with `sysdig` or eBPF.
- If still unresponsive → check kernel logs (`dmesg`).

---

110. Scenario: Your Docker host runs out of PID limits due to too many short-lived containers. How do you prevent PID exhaustion?

Answer:

- Set `--pids-limit` per container.
- Tune host PID limits (`/proc/sys/kernel/pid_max`).
- Batch jobs into fewer containers.
- Use orchestrators with job queues (K8s CronJobs).

---

111. Scenario: Your containerized ML workloads require GPUs and TPUs. How do you configure Docker?

Answer:

- NVIDIA: use `--gpus all` + NVIDIA runtime.
- TPU: not directly supported → use Kubernetes device plugins.
- Validate driver + CUDA/cuDNN compatibility.

---

112. Scenario: You must share container images with partners securely. How do you design image distribution?

Answer:

- Use **private registries** (Harbor, ECR, ACR).
- Enable TLS + authentication.
- Sign images with Notary/Cosign.
- Use RBAC for access control.

---

113. Scenario: Docker containers keep failing due to DNS resolution errors. How do you troubleshoot?

Answer:

- Check `/etc/resolv.conf` inside container.
- Verify Docker daemon DNS settings (`--dns`).
- Ensure firewall isn't blocking UDP 53.
- Use internal DNS (CoreDNS, Consul) for reliability.

---

114. Scenario: You need to run real-time audio/video processing inside containers. How do you minimize latency?

Answer:

- Use **host networking** for lower latency.
- Assign CPU pinning (`--cpuset-cpus`).
- Tune kernel with low-latency options.
- Disable unnecessary logging.

---

115. Scenario: Your CI/CD pipeline runs untrusted user code inside Docker. How do you sandbox securely?

Answer:

- Use rootless Docker or Podman.
- Apply seccomp, AppArmor, SELinux profiles.
- Run builds in isolated VMs (Firecracker, Kata Containers).
- Limit syscalls and device access.

---

116. Scenario: A production container is consuming too many open file descriptors. How do you handle it?

Answer:

- Increase ulimits (`--ulimit nofile=65535:65535`).
- Use connection pooling in app.
- Monitor with `lsof` inside container.
- Tune OS kernel if required.

---

117. Scenario: You need to migrate petabytes of Docker volumes to a new cluster. How do you do it?

Answer:

- Use snapshot-based storage migration (EBS snapshots, Ceph RBD).
- Sync volumes with `rsync` or `restic`.
- For live migration, use replication before cutover.
- Validate with checksum/hash verification.

---

118. Scenario: You need fine-grained RBAC for Docker usage. How do you implement it?

Answer:

- Docker itself lacks RBAC → integrate with orchestrators (Swarm/K8s).
- Use authorization plugins for Docker.
- Enforce policies via OPA (Open Policy Agent).

---

119. Scenario: You discover that some containers in production are running outdated vulnerable images. How do you enforce updates?

Answer:

- Enable **image scanning** in CI/CD.
- Automate rebuilds when base images change.
- Use `watchtower` or K8s imagePullPolicy=Always.
- Block old tags with registry policies.

---

120. Scenario: Leadership asks for a disaster recovery plan for all Dockerized apps. What would you propose?

Answer:

- Regular backups of images (registry) + volumes.
- Multi-region private registry replication.
- Infrastructure-as-Code for rapid redeployments.
- Runbooks for restoring images + data.
- Chaos testing to validate DR readiness.

121. Scenario: Your company runs thousands of containers daily. The Docker daemon becomes a bottleneck. How do you scale beyond single-node Docker?

Answer:

- Docker Swarm or Kubernetes for multi-node orchestration.
- CRI-O or containerd instead of Docker runtime.
- Split workloads across clusters (multi-cluster strategy).

---

122. Scenario: A critical production container exits without logs. What are your steps to investigate?

Answer:

- Run with `--restart always` + `--log-driver=json-file`.
- Check exit code with `docker inspect`.
- Enable systemd journal logs.
- Use `--init` to handle zombie processes.

---

123. Scenario: Your Docker host gets hacked via a malicious container. What mitigation steps would you take?

Answer:

- Run containers as non-root (`USER`).
- Enable seccomp, AppArmor, SELinux.
- Patch Docker daemon regularly.
- Restrict access to `/var/run/docker.sock`.

---

124. Scenario: You want to run Docker inside Docker (DinD) for CI pipelines. What issues can arise?

Answer:

- Privilege escalation risks.
- High disk I/O usage.
- Inconsistent caching.
- Better solution: use Kaniko or BuildKit for rootless builds.

---

125. Scenario: How would you optimize Docker for IoT edge devices with low CPU/memory?

Answer:

- Use Alpine or scratch images.
- Cross-compile for ARM.
- Use BuildKit caching.
- Prune old images frequently.

---

126. Scenario: How do you prevent image sprawl in a large organization?

Answer:

- Enforce naming conventions.
- Use registry cleanup policies.
- Automate scans for unused images.
- Implement centralized governance with Harbor.

---

127. Scenario: A container runs out of ephemeral storage due to excessive logs. How do you prevent it?

Answer:

- Configure `--log-opt max-size` and `--log-opt max-file`.
- Ship logs to ELK/Fluentd.
- Use tmpfs for ephemeral files.
- Monitor with `docker system df`.

---

128. Scenario: Your compliance team requires image provenance tracking. How do you achieve this?

Answer:

- Sign images with Cosign/Notary.
- Maintain SBOM (Software Bill of Materials).
- Store build metadata in registry.
- Automate policy checks with OPA.

---

129. Scenario: You need multi-tenant isolation in Docker. How do you architect it?

Answer:

- Separate users into namespaces (userns-remap).
- Apply seccomp profiles per tenant.
- Use dedicated Docker daemons or VM isolation.
- Orchestrators (K8s) with node pools per tenant.

---

130. Scenario: You have inconsistent network performance between Docker containers across different hosts. How do you fix it?

Answer:

- Switch from default bridge to overlay network.
- Tune MTU size.
- Use CNI plugins like Calico, Flannel.
- Verify firewall rules.

---

131. Scenario: You want to run serverless functions on Docker. What approach would you take?

Answer:

- Use lightweight container runtimes (Firecracker, gVisor).
- Deploy OpenFaaS, Knative, or AWS Lambda runtime with Docker.
- Optimize cold-start by pre-pulling images.

---

132. Scenario: What are the risks of mounting the host Docker socket inside a container?

Answer:

- Full root access to host.
- Attackers can start privileged containers.
- Can escape container sandbox.
- Solution: never mount `/var/run/docker.sock`.

---

133. Scenario: Your build pipeline creates huge Docker images (>2GB). How do you slim them down?

Answer:

- Use `.dockerignore`.
- Optimize multi-stage builds.
- Choose minimal base images.
- Clean temp files in the same layer.

---

134. Scenario: You need to migrate from Docker Swarm → Kubernetes. How do you approach it?

Answer:

- Convert Docker Compose to Helm charts.
- Map Swarm services → K8s Deployments.
- Map Secrets, Configs → K8s equivalents.
- Migrate networking (Swarm overlay → K8s CNI).

---

135. Scenario: Docker Hub rate limits are slowing your CI/CD builds. What's your solution?

Answer:

- Use authenticated pulls with tokens.
- Mirror images in a private registry.
- Cache images locally with proxy registries.

---

136. Scenario: You detect image tampering in your registry. How do you secure it?

Answer:

- Enable content trust (Notary).
- Enforce digest-based deployments.
- Monitor registry logs.
- Use vulnerability scanners (Clair, Trivy).

---

137. Scenario: Your ML training containers require terabytes of data. How do you efficiently mount data?

Answer:

- Use NFS/CephFS for shared mounts.
- Leverage object storage with FUSE drivers.
- Use volume plugins for high IOPS.
- Cache frequently accessed data locally.

---

138. Scenario: You want to run air-gapped Docker deployments (no internet). How do you handle dependencies?

Answer:

- Mirror base images locally.
- Pre-download dependencies in build stage.
- Use portable registries (Harbor, Nexus).
- Automate sync scripts for updates.

---

139. Scenario: You must run Docker workloads across multiple clouds. What's your strategy?

Answer:

- Push images to multi-cloud registries (ECR, ACR, GCR).
- Use Kubernetes Federation or Rancher for orchestration.
- Enforce image consistency with digest-based pulls.
- Implement centralized CI/CD pipelines.

---

140. Scenario: Leadership asks: *"Can Docker fully replace VMs?"* How do you answer?

Answer:

- No, containers share host kernel, unlike VMs.
- VMs provide stronger isolation/security.
- Use containers for app portability, VMs for infra boundaries.
- Hybrid approach (VMs + containers) is common.

141. Scenario: Your containers randomly crash with `Segmentation Fault`. How do you debug at the Docker level?

Answer:

- Use `docker run --cap-add=SYS_PTRACE` for debugging.
- Attach `gdb` or `strace` inside container.
- Check kernel logs (`dmesg`).
- Ensure app binary matches container architecture.

---

142. Scenario: You notice orphaned Docker volumes eating storage. How do you manage and prevent them?

Answer:

- Run `docker volume prune`.
- Enforce cleanup policies in CI/CD.
- Tag volumes properly.
- Use external volume drivers with lifecycle hooks.

---

143. Scenario: A containerized app depends on kernel modules not available in the host. How do you solve it?

Answer:

- Load required modules on host.
- Use privileged containers with module mounts.
- Run container inside VM with required modules.

---

144. Scenario: Your company needs auditable Docker usage for SOC 2 compliance. How do you implement it?

Answer:

- Enable Docker daemon audit logs.
- Forward logs to SIEM.
- Enforce RBAC for `docker` commands.
- Periodically run CIS Docker Bench.

---

145. Scenario: A security team requires rootless containers. What challenges will you face?

Answer:

- Limited networking options.
- Volume mount restrictions.
- GPU and device access harder.
- Some legacy apps may fail without root.

---

146. Scenario: How do you optimize Docker image pulls in high-latency environments?

Answer:

- Use registry mirrors closer to nodes.
- Pre-pull images in advance.
- Compress images with BuildKit.
- Enable layer caching across builds.

---

147. Scenario: You want to limit network bandwidth per container. How do you achieve it?

Answer:

- Use `--network` with CNI plugins supporting bandwidth.
- Apply Linux `tc` (traffic control).
- Kubernetes: use NetworkPolicies with QoS.

---

148. Scenario: Docker containers fail under ultra-high file I/O workloads. What's your approach?

Answer:

- Use `--mount` with direct host volumes.
- Tune storage drivers (prefer `overlay2`).
- Place volumes on SSD-backed storage.
- Separate logs from data paths.

---

149. Scenario: You need deterministic builds for regulated industries. How do you enforce them?

Answer:

- Lock dependency versions.
- Use `--build-arg` for fixed timestamps.
- Rebuild from source regularly.
- Generate SBOM for compliance.

---

150. Scenario: A developer accidentally runs `docker system prune -a` on a production host. How do you mitigate impact?

Answer:

- Maintain image backups in registry.
- Enable `DOCKER_OPTS="--icc=false"` for extra safety.
- Restrict Docker access with RBAC.
- Automate redeployment from IaC (Terraform/Ansible).

---

151. Scenario: You want to analyze syscalls inside containers for performance debugging. How do you do it?

Answer:

- Use `strace -p <PID>`.
- Use `sysdig` or eBPF-based tools.
- Monitor `/proc/<PID>/syscall`.
- Aggregate syscall data for hotspots.

---

152. Scenario: How do you enforce data encryption inside Docker volumes?

Answer:

- Use encrypted volume drivers (RexRay, Portworx).
- OS-level encryption (dm-crypt, LUKS).
- Application-level encryption before writes.

---

153. Scenario: A Docker image build is too slow due to network bottlenecks. How do you fix it?

Answer:

- Cache dependencies.
- Use `--mount=type=cache` in BuildKit.
- Host local package mirrors (apt, pip, npm).
- Parallelize multi-stage builds.

---

154. Scenario: You want zero-downtime updates for Dockerized apps without Kubernetes. What’s your approach?

Answer:

- Run containers behind HAProxy/Nginx.
- Use `docker-compose up --no-deps --build -d`.
- Shift traffic gradually (blue-green).

---

155. Scenario: A financial service requires FIPS-compliant cryptography in Docker. How do you ensure this?

Answer:

- Use FIPS-enabled base images.
- Configure OpenSSL with FIPS mode.
- Validate compliance with automated tests.

---

156. Scenario: How do you detect and kill rogue containers launched outside of your orchestration system?

Answer:

- Monitor Docker events API.
- Use audit logs for container start/stop.
- Run periodic scans against registry policies.
- Alert on containers not managed by CI/CD.

---

157. Scenario: You must integrate Docker logs with centralized observability (ELK/Prometheus/Grafana). How do you architect it?

Answer:

- Use logging drivers (`fluentd`, `gelf`, `syslog`).
- Sidecar logging containers.
- Export Docker metrics via cAdvisor.
- Ship logs to ELK with Filebeat.

---

158. Scenario: A single container consumes all CPU cores. How do you restrict it?

Answer:

- Use `--cpus="2"` or `--cpu-quota`.
- Pin to cores with `--cpuset-cpus`.
- Monitor with `docker stats`.

---

159. Scenario: You must deploy Docker workloads on 5G edge networks with ultra-low latency. What optimizations do you apply?

Answer:

- Use minimal images.
- Co-locate workloads near edge nodes.
- Pre-warm containers to reduce cold start.
- Use DPDK-enabled networking containers.

---

160. Scenario: Your team wants full Docker observability (infra → container → app). How do you achieve this?

Answer:

- Use cAdvisor + Prometheus for metrics.
- Fluentd/Logstash for logs.
- Jaeger/Zipkin for tracing.
- Correlate data with Grafana dashboards.

161. Scenario: Your CI/CD builds keep failing due to Docker Hub outage. How do you make your pipelines resilient?

Answer:

- Use mirrored private registries (Harbor, ECR, ACR, GCR).
- Cache base images locally.
- Implement fallback registry pull policies.

---

162. Scenario: You must guarantee regulatory audit trails for all container actions. How do you enforce this?

Answer:

- Enable `dockerd --log-level=debug`.
- Collect Docker events (`docker events`).
- Stream logs to SIEM (Splunk, ELK).
- Integrate with OPA Gatekeeper.

---

163. Scenario: Your Docker host crashes due to extreme inode usage. What happened?

Answer:

- Containers writing millions of small files.
- Overlay2 storage driver exhaustion.
- Fix: mount volumes on ext4/xfs with higher inode count, clean unused files.

---

164. Scenario: How would you run deterministic multi-arch builds without emulation overhead?

Answer:

- Use `docker buildx` with native ARM & x86 build nodes.
- Run CI pipelines on both hardware types.
- Push a multi-arch manifest list.

---

165. Scenario: A production container keeps restarting due to OOM, but memory usage looks normal. Why?

Answer:

- Application hitting kernel memory (not user memory).
- Memory spikes faster than monitoring interval.
- Hidden leaks in native libraries.
- Fix: enable cgroup memory accounting + kernel tuning.

---

166. Scenario: How do you handle Docker upgrades across hundreds of production hosts?

Answer:

- Use config management (Ansible, Chef, Puppet).
- Canary upgrades.
- Validate with `docker version` drift reports.
- Automate rollback procedures.

---

167. Scenario: How do you secure secrets management inside Docker?

Answer:

- Use Docker Swarm secrets.
- Mount secrets from external vault (HashiCorp Vault, AWS Secrets Manager).
- Avoid baking secrets into images.

---

168. Scenario: Your legal team requires a Software Bill of Materials (SBOM) for Docker images. How do you generate it?

Answer:

- Use Syft, Trivy, Anchore.
- Store SBOMs in registry metadata.
- Automate generation during CI/CD builds.

---

169. Scenario: You want to run immutable Docker containers that can’t be modified after start. How?

Answer:

- Mount root filesystem as read-only (`--read-only`).
- Store writable paths in tmpfs/volumes.
- Disable exec into running containers.

---

170. Scenario: How do you optimize Docker for real-time trading apps that require microsecond latency?

Answer:

- Use CPU pinning.
- Enable `--network=host`.
- Use tuned kernel with low-latency profile.
- Minimize syscalls and logging.

---

171. Scenario: A developer accidentally pushes gigabytes of sensitive data into a public Docker image. What do you do?

Answer:

- Revoke compromised credentials.
- Delete public image (may remain cached).
- Rotate secrets.
- Automate scanning to prevent recurrence.

---

172. Scenario: Containers on the same host experience port conflicts. How do you fix it?

Answer:

- Use Docker bridge networking with unique ports.
- Use `--network=overlay` for multi-container apps.
- Apply reverse proxies (Nginx, Traefik).

---

173. Scenario: You must throttle disk I/O per container. How do you configure it?

Answer:

- Use `--device-read-bps` and `--device-write-bps`.
- Apply blkio cgroup limits.
- Monitor via `iotop` inside container.

---

174. Scenario: Your containers rely on ephemeral cloud metadata services (AWS IMDS, GCP metadata). How do you secure access?

Answer:

- Use metadata proxy with IAM role scoping.
- Block host-level metadata IP from non-trusted containers.
- Rotate temporary tokens.

---

175. Scenario: How do you run sandboxed untrusted workloads inside Docker (like online IDEs)?

Answer:

- Use gVisor or Kata Containers for isolation.
- Enforce seccomp & AppArmor profiles.
- Limit CPU/memory aggressively.
- Network isolation with CNI plugins.

---

176. Scenario: Docker host is under DDoS attack via containers spawning excessive processes. How do you defend?

Answer:

- Apply `--pids-limit`.
- Enable Linux cgroup process accounting.
- Rate-limit container spawns with orchestrator.

---

177. Scenario: How do you monitor container lifecycle events at scale?

Answer:

- Subscribe to `docker events` API.
- Export metrics with Prometheus.
- Send event streams to Kafka for analysis.

---

178. Scenario: What’s your strategy for Docker image supply chain security?

Answer:

- Pin digest-based pulls.
- Use signed images.

- Automate scans at build & deploy.
- Monitor CVEs continuously.

---

179. Scenario: You must deploy Docker workloads to an HPC cluster. What challenges do you face?

Answer:

- MPI communication requires low-latency networking.
- GPU/FPGA passthrough.
- Large shared volumes (Lustre/GlusterFS).
- Scheduler integration with Slurm.

---

180. Scenario: Future of Docker — do you think containers will replace unikernels or co-exist?

Answer:

- Containers: great for general-purpose workloads.
- Unikernels: ultra-light, but niche.
- Likely co-existence: containers for portability, unikernels for ultra-secure microservices.

181. Scenario: Your Docker host kernel panics due to container workloads. How do you debug?

Answer:

- Collect kernel logs (`dmesg`, `/var/log/messages`).
- Use crash dumps with `kdump`.
- Correlate container cgroups to failing processes.
- Isolate workload and test on hardened kernel.

---

182. Scenario: A containerized service must run on air-gapped military infrastructure. How do you deliver images?

Answer:

- Use `docker save` / `docker load` for image tarballs.
- Mirror a registry offline.
- Use signed and hashed image tarballs for verification.

---

183. Scenario: How do you secure Docker against kernel privilege escalation exploits?

Answer:

- Drop all capabilities except required ones.
- Use seccomp, AppArmor, SELinux profiles.
- Regularly patch host kernel.
- Consider gVisor/Kata for strong isolation.

---

184. Scenario: Your organization requires FIPS-compliant cryptography for Docker. How do you enforce it?

Answer:

- Run Docker on FIPS-enabled Linux distributions (RHEL, Ubuntu FIPS).
- Use FIPS-certified OpenSSL builds in base images.
- Verify compliance with scanning tools.

---

185. Scenario: You want to run thousands of containers per host. What Linux tuning do you apply?

Answer:

- Increase `ulimit` (file descriptors, processes).
- Tune sysctl: `vm.max_map_count`, `fs.inotify.max_user_instances`.
- Enable cgroup v2 for better resource isolation.
- Use lightweight runtimes like containerd instead of full Docker.

---

186. Scenario: Your CI/CD builds fail due to network instability when pulling large images. How do you solve it?

Answer:

- Use smaller base images.
- Layer caching.
- Deploy local registry mirror.
- Enable retries with exponential backoff.

---

187. Scenario: You must ensure data at rest encryption for container volumes. How?

Answer:

- Use encrypted storage drivers (dm-crypt, LUKS).
- Cloud-managed encrypted volumes (EBS, Azure Disk, GCP PD).
- Application-level encryption for sensitive data.

---

188. Scenario: How do you enforce policy-as-code for Docker security?

Answer:

- Integrate Open Policy Agent (OPA) with Docker.
- Use Conftest for Dockerfile linting.
- Block non-compliant images in CI/CD.

---

189. Scenario: Containers must use dedicated GPUs for ML training. How do you configure scheduling?

Answer:

- Use NVIDIA Docker runtime (`--gpus`).
- Limit access to specific GPU indices.
- Schedule via Kubernetes device plugins.

---

190. Scenario: How do you troubleshoot container DNS resolution failures?

Answer:

- Inspect `/etc/resolv.conf` inside container.
- Check Docker network DNS config.
- Ensure host `/etc/docker/daemon.json` has correct DNS settings.
- Debug with `dig`/`nslookup` inside container.

---

191. Scenario: A containerized microservice leaks file descriptors, leading to failures. How do you detect and fix?

Answer:

- Use `lsof` to track open files.
- Monitor descriptor usage with Prometheus.

- Apply ulimit limits per container.
- Fix application-level leaks.

---

192. Scenario: You must enforce immutable container deployments (no manual exec into containers). How do you do it?

Answer:

- Block `docker exec` via RBAC.
- Enforce CI/CD-based redeployment only.
- Audit container changes with intrusion detection tools.

---

193. Scenario: You need live-patching of Docker host kernel without downtime. How do you achieve it?

Answer:

- Use tools like Ksplice, kpatch, or KernelCare.
- Run containers on multiple hosts with rolling upgrades.
- Automate live-patching via orchestration layer.

---

194. Scenario: How do you run Docker workloads with real-time video streaming requirements?

Answer:

- Use host networking.
- Tune kernel buffer sizes.
- Pin CPU cores for video encoding/decoding.
- Use GPU passthrough for acceleration.

---

195. Scenario: You want to audit all container syscalls for security monitoring. How do you implement it?

Answer:

- Enable seccomp audit logging.
- Use eBPF-based tools (Falco, Tracee).
- Send logs to SIEM for anomaly detection.

---

196. Scenario: How do you handle multi-cloud Docker deployments with compliance restrictions?

Answer:

- Use multi-cloud registries (Harbor, JFrog).
- Encrypt and replicate images across regions.
- Apply consistent security scanning policies across providers.

---

197. Scenario: Your Docker Swarm cluster must meet PCI-DSS compliance. What steps do you take?

Answer:

- Encrypt all secrets and cardholder data.
- Restrict network access with firewalls.
- Enable container activity monitoring.
- Maintain audit logs for every container lifecycle event.

---

198. Scenario: A containerized AI workload requires terabytes of shared training data. How do you design storage?

Answer:

- Use distributed storage (Ceph, GlusterFS, NFS).
- Mount as volumes across containers.
- Enable caching layers to reduce read bottlenecks.

---

199. Scenario: Your organization wants Docker for 5G edge workloads. What challenges arise?

Answer:

- Ultra-low latency networking (SR-IOV, DPDK).
- Resource constraints on edge nodes.
- Synchronization across distributed nodes.
- Security at remote locations.

---

200. Scenario: What’s the next evolution beyond Docker in container technology?

Answer:

- Rootless containers (Podman, CRI-O).
- MicroVMs (Firecracker, Kata Containers).
- Wasm (WebAssembly) for ultra-light secure workloads.
- Likely hybrid environments: Docker for dev, K8s + Wasm for prod.