Orchestration

Docker Certified Associate

KNOWLEDGE CHEATSHEET

DESIRED STATE AND RECOVERY

Desired state → **Schedule tasks** → **Check health** → **Reconcile**

01 Orchestration

Swarm membership and quorum

- A **node** is a Docker Engine host participating in the swarm, on a physical or virtual machine. **Managers** maintain cluster state and schedule work; **workers** execute tasks.
- **Raft quorum** requires a majority of managers: floor(N/2) + 1. Three managers tolerate one failure; five tolerate two; seven tolerate three. Two managers cannot tolerate losing either manager.
- Use an **odd number of managers** and spread them across failure domains. With seven managers in three zones, **3-2-2** retains quorum after losing any one zone; 4-2-1 and 5-1-1 do not.
- Run **docker node ls** on a manager to list swarm nodes. A CLI connection alone does not make a machine a swarm node. Worker count does not increase manager quorum.

Services, placement and health

- A **service** declares desired workload state; a **task** is one scheduled container instance. Replicated services request a replica count; global services run one task on each eligible node.
- Use **node labels** and **placement constraints** for mandatory separation: docker node update --label-add env=prod NODE, then docker service create --constraint 'node.labels.env==prod' IMAGE.
- **Placement preferences** spread tasks on a best-effort basis. Environment variables configure the application; they do not restrict eligible nodes. Missing capacity or matching nodes can leave tasks pending.
- **docker service ps http** lists current and retained historical tasks, including task errors. **docker service inspect http** shows service configuration; **docker service logs http** shows logs with supported logging drivers.
- **HEALTHCHECK** defines a command, interval, timeout, retries and optional startup grace period. Consecutive failures can mark a container unhealthy. Swarm can replace unhealthy service tasks; a standalone health check does not itself restart a container.

Kubernetes workloads and maintenance

- A **Pod** groups containers sharing networking and declared volumes. A **Deployment** manages replicas through ReplicaSets. A **namespace** groups namespaced objects such as Deployments, Secrets and ConfigMaps; Pods do not contain these API objects.
- **Labels** identify objects; **selectors** match current labels, including on objects created later. Use kubectl get pods -A -l env=development for all namespaces; omitting -A uses the current namespace.
- A **liveness probe** can trigger restart of the failing container according to its restart policy. A failed **readiness probe** removes it from normal Service traffic. A **startup probe** delays liveness and readiness checks until startup succeeds.

Orchestration / Images & Registry

Docker Certified Associate

KNOWLEDGE CHEATSHEET

IMAGE DELIVERY

Dockerfile → **Build image** → **Registry** → **Deploy**

- **kubectl cordon NODE** prevents new scheduling. **kubectl drain NODE** also evicts eligible Pods for maintenance, respecting disruption controls; DaemonSets and local data can need explicit handling. **kubectl uncordon NODE** restores scheduling.
- Use **kubectl describe deployment api** to inspect details and related events. **kubectl get** summarizes objects; **kubectl logs** reads container output. Check Pod events and status when a Deployment cannot become ready.

02 Image creation, management and registry

Dockerfiles and efficient builds

- A **Dockerfile** describes how to build an image. An **image** supplies read-only filesystem layers and runtime defaults; a container adds a writable layer. Persistent application data belongs in external storage.
- **COPY** copies files from the build context or a selected build stage. **ADD** also supports remote URL sources and automatic extraction of recognized local tar archives. Wildcard patterns are not regular expressions.
- Use **multi-stage builds** with multiple FROM instructions and **COPY --from=build** to transfer only runtime artifacts. This separates build tools from the final image and can reduce size and attack surface.
- Place **stable dependencies early** and frequently changed application files late. Cache reuse depends on the instruction and its inputs; copying all source before installing dependencies often causes avoidable cache invalidation.
- Use **.dockerignore** to exclude unnecessary build-context files. Keep environment-specific settings and certificates outside the image; build once and supply configuration at deployment time.

DTR access and image lifecycle

- **Docker Trusted Registry (DTR)** stores and distributes images and controls repository access. Organization membership alone does not imply write access; assign appropriate **team or user repository permissions**.
- **Pull/read** and **push/write** are separate permissions. Grant access to the intended repository; copying or mirroring an image elsewhere does not grant permission to the original repository.
- Enable repository **tag immutability** to prevent reassignment of existing tags. Naming a tag 'immutable' has no enforcement effect. A **digest** identifies image content independently of a mutable tag.

Images & Registry / Installation & Configuration

Docker Certified Associate

KNOWLEDGE CHEATSHEET

CONFIGURE THE RIGHT LAYER

- **Engine** — Daemon settings
- **Workload** — Runtime config
- **Logs** — Application output
- **Audit** — API requests

- **Pruning policies** select image references according to configured rules and retention. **Garbage collection** reclaims unreferenced registry blobs after deletion; layers still referenced by other images must remain.
- **Security scanning** identifies supported components, known vulnerabilities and license information. Coverage depends on the scanner database and supported packages; it does not guarantee detection of embedded private keys or every insecure image configuration.

03 Installation and configuration

Engine and enterprise control plane

- The **Docker client** calls the **daemon API**; the daemon manages images, containers, networks and volumes. **docker version** reports client/server versions; **docker info** reports daemon configuration and capabilities.
- On Linux, **/etc/docker/daemon.json** configures daemon options. Validate JSON and avoid conflicting command-line settings. Apply changes through the service's supported reload or restart behavior.
- An **edition upgrade** requires the supported engine packages, licensing and upgrade procedure. Stopping the service alone does not change the installed edition. Check supported host OS, architecture and container platform before installation.
- **Universal Control Plane (UCP)** provides enterprise cluster management, authentication and authorization across orchestrated resources. **DTR** provides image repository management. Keep cluster deployment permissions separate from registry push/pull permissions.
- Use **local accounts** or the configured external identity integration for UCP authentication. Identity-provider integration establishes who a user is; **RBAC grants** determine which resources and actions that identity can access.

Logs, audits and runtime settings

- Set daemon **log-driver** and **log-opts** in daemon.json for defaults on newly created containers. For Splunk, use the **splunk** driver and its destination/token options. Existing containers retain their logging configuration.
- Override defaults at creation with **--log-driver** and **--log-opt**. **docker logs** reads available container output; **docker events** reports engine events. Neither command configures forwarding to a central log service.
- Enable **UCP API audit logging** before an incident. **Metadata** captures request identity, operation and resource information; **Request** adds request-body detail. Plan retention and access because request bodies may contain sensitive data.

Installation & Configuration / Networking

Docker Certified Associate

KNOWLEDGE CHEATSHEET

SERVICE TRAFFIC PATH

Client → **Service IP : port** → **Selected endpoint** → **Pod : targetPort**

- Use **Swarm configs** for nonsecret runtime files and **Swarm secrets** for sensitive values such as private keys. Grant each service only the objects it needs; changing an image's build instructions is not runtime configuration.
- In Kubernetes, create a **ConfigMap** and reference it through a Pod volume and container **volumeMounts**, or supported environment references. Use **Secrets** for sensitive data and restrict their access; base64 encoding is not encryption.

04 Networking

Network drivers and published ports

- A **bridge** connects containers on one Docker host; a user-defined bridge supplies container-name DNS. **Host networking** shares the host network namespace. An **overlay** connects workloads across swarm nodes.
- **Macvlan** and **ipvlan** attach containers through the underlying network and require suitable routing or switching configuration. A local bridge does not become a multi-host network merely because it is user-defined.
- **EXPOSE** documents intended container ports. **--publish / -p** creates host port mappings, for example docker run -p 8080:80 nginx. Bind a host address when exposure should be restricted; the app must listen on a reachable container address.
- Use **docker port CONTAINER** or **docker container inspect CONTAINER** to examine published mappings. **docker network inspect NETWORK** examines membership and network settings. Connecting a container to a network does not publish a port.
- Publish UDP explicitly: **docker service create --name dns-cache -p 53:53/udp IMAGE**. The short form is published:target/protocol; TCP is the default. Publishing a UDP mapping does not prevent the application from listening on other internal ports.
- Encrypt eligible overlay data traffic with **docker network create -d overlay --opt encrypted appnet**, then attach services to appnet. Overlay data-plane encryption is separate from swarm control-plane TLS.

Kubernetes Services and policy

- Containers in one **Pod share its IP address and port space** and communicate over localhost. Address planning counts Pod IPs, not container count. A /26 contains 64 addresses; usable Pod capacity also depends on networking reservations and node limits.
- A **ClusterIP Service** exposes a stable virtual IP and **port**, forwarding to eligible selected Pod endpoints at **targetPort**. A Service with port 80 and targetPort 8080 sends requests to the Pod's port 8080.
- A Service's **selector** normally matches Pods in its own namespace. Labels determine membership; readiness normally determines whether an endpoint receives traffic. Do not assume a specific balancing algorithm or infer direction from label names.

Networking / Security

Docker Certified Associate

KNOWLEDGE CHEATSHEET

SEPARATE SECURITY CONTROLS

- **Namespaces** — Isolation
- **cgroups** — Resource limits
- **TLS** — Transport trust
- **Signatures** — Image trust

- A **NetworkPolicy** selects Pods within its namespace and controls specified ingress or egress when the network plugin enforces it. Policies are additive; selected Pods accept the union of allowed traffic for the isolated direction.
- For Pod-to-Pod traffic, check **source egress** and **destination ingress**. Both must allow the flow when each side is isolated. Evaluate podSelector, namespaceSelector, ports and protocol; 'frontend' and 'backend' labels have no built-in allow/deny meaning.

05 Security

Linux isolation and privilege

- **Namespaces** isolate views of resources: PID for processes, network for interfaces and ports, mount for filesystems, IPC for interprocess communication, and UTS for hostname. They do not set CPU or memory quotas.
- **cgroups** account for and limit resource use, including CPU and memory. Docker resource flags configure these controls. **Capabilities** split privileged operations; **seccomp** filters system calls; **SELinux/AppArmor** enforce mandatory access policies.
- Changing system time requires **CAP_SYS_TIME**, which Docker does not grant by default. Container root does not automatically possess every host privilege. Diagnose the specific capability, syscall and security-policy restrictions.
- **User namespaces** map container user IDs to different host IDs. Docker's **userns-remap** is an explicit daemon configuration; it reduces host privilege exposure without replacing other isolation controls.
- **--privileged** greatly expands capabilities and device access and relaxes security confinement. It is not a general instruction to remove configured CPU or memory limits. Prefer narrowly scoped capabilities and device grants.

Registry TLS and content trust

- **TLS certificate trust** authenticates the registry endpoint and protects transport. An unknown-authority error means the client lacks a trusted certificate chain; image signatures address a separate trust decision.
- Trust a private registry CA at **/etc/docker/certs.d/REGISTRY/ca.crt** on each engine host that pulls images. REGISTRY includes the port when used. Alternatively, install the CA in the host trust store using the platform's supported procedure.
- If explicitly required, configure the daemon's **insecure-registries** list for a registry lacking trusted TLS. This weakens transport security; prefer installing its CA. A client environment variable does not replace daemon registry configuration.

Security / Storage & Volumes

Docker Certified Associate

KNOWLEDGE CHEATSHEET

PERSISTENT STORAGE ATTACHMENT

PV Storage → **PVC** Claim → **Pod** Volume → **Container** Mount path

- **Docker Content Trust (DCT)** uses signed tag metadata to establish image publisher authenticity and integrity. **DOCKER_CONTENT_TRUST=1** enables verification for supported client workflows; local image inspection is not a signature-verification operation.
- With trust enabled, a build needing an **unsigned base image** can fail when resolving that base through a registry. **UCP signed-image enforcement** applies cluster deployment policy. A registry's 'official image' label is not equivalent to a trusted signature.

06 Storage and volumes

Docker mounts and persistence

- A container's **writable layer** belongs to that container and is lost when it is removed. A **named volume** is managed separately by Docker. A **bind mount** exposes a specific host path inside the container.
- Mount a host directory read-only with **docker run --volume /data:/mydata:ro ubuntu**. The equivalent long form is **--mount type=bind,src=/data,dst=/mydata,readonly**. The read-only setting applies to that container mount.
- Use **docker container inspect nginx** and inspect **Mounts** for volume names, sources, destinations and write permissions. **docker volume ls** lists managed volumes; **docker volume inspect NAME** reports one volume's metadata.
- A **local volume** is local to its host unless its driver provides shared storage. Swarm task rescheduling does not automatically move local data to another node; plan shared storage or placement around the data's location.

Storage & Volumes

Docker Certified Associate

KNOWLEDGE CHEATSHEET

PERSISTENT STORAGE ATTACHMENT

PV Storage → **PVC** Claim → **Pod** Volume → **Container** Mount path

Kubernetes persistent storage

- A **PersistentVolume (PV)** is cluster-scoped storage; a **PersistentVolumeClaim (PVC)** is a namespaced request. A Pod's **volumes** references the PVC, and its container's **volumeMounts** selects the mount path.
- A **StorageClass** specifies provisioning behavior through a provisioner and parameters. A PVC using a suitable class can trigger dynamic provisioning; the provisioner, topology and requested capacity/access modes must all be satisfiable.
- An explicit **storageClassName: ""** requests no class and disables dynamic provisioning for that claim. If no compatible classless PV exists, the PVC stays **Pending**. Omitting the field can allow default-class assignment.
- Static **PV/PVC binding** must match class, capacity, access modes and other requested constraints. A **WaitForFirstConsumer** binding mode can defer provisioning or binding until scheduling supplies topology information.
- For a local development mount, declare **hostPath.path: /data** in a Pod volume and reference it with volumeMounts. The path is on the **Kubernetes node**, so a VM-based local cluster needs that laptop path made available inside its node.
- A **hostPath-backed PV** can expose a node path through a PVC, but it does not create portable shared storage. The PV's **reclaim policy** controls whether released backing storage is retained or deleted.