# Week 7: Container Fundamentals

## NT524 — Cloud Architecture and Security

PhD. Nguyen Ngoc Tu

October 22, 2025

*Thuật ngữ:* container fundamentals = *nền tảng container*; cloud architecture = *kiến trúc đám mây*

# Outline

# Learning Objectives

- Application Isolation models
- Containers and VMs: Compare and contrast
- Container usage: portability, lower overhead, microservices
- Container building blocks: container **runtime** and **platform** *(orchestrator boundary only; details in Week 8)*

---

*Thuật ngữ:* platform = *nền tảng container*; orchestrator = *bộ điều phối (để tuần sau)*

# Review — Isolation Mindset (Compute → Network → Storage)

- **Compute / Hardware (VMs):** Hypervisor isolation; tenancy boundaries; vCPU/NUMA pinning; I/O isolation (IOMMU, SR-IOV) to contain blast radius and noisy neighbors.
- **Network (SDN):** Tenant segmentation (VRF/VLAN/VXLAN/Geneve), Security Groups/ACLs, microsegmentation, controlled exposure (LB/NAT), and policy enforcement.
- **Storage:** Ephemeral vs persistent; block/object/shared filesystems; snapshots/clones; encryption-at-rest; access modes (RWO/RWX); quotas & I/O QoS to bound interference.

---

*Thuật ngữ:* VRF (Virtual Routing and Forwarding) = *định tuyến & chuyển tiếp ảo*; VLAN (Virtual LAN) = *mạng LAN ảo*; VXLAN/Geneve = *mạng phủ đóng gói L2 trên L3*; Security Group = *nhóm bảo mật*; ACL (Access Control List) = *danh sách kiểm soát truy cập*; microsegmentation = *vi phân đoạn*; IOMMU (Input–Output Memory Management Unit) = *bộ ánh xạ bộ nhớ I/O*; SR-IOV (Single Root I/O Virtualization) = *ảo hoá I/O đơn gốc*; LB/NAT (Load Balancer / Network Address Translation) = *cân bằng tải / biên dịch địa chỉ mạng*; RWO/RWX (ReadWriteOnce / ReadWriteMany) = *đọc–ghi độc quyền / đọc–ghi chia sẻ*; QoS (Quality of Service) = *chất lượng dịch vụ*

# Motivation — Application Isolation (Why we care)

- **Contain failures and cross-tenant impact:** isolate each service/tenant so one compromise or crash cannot cascade.
- **Portable, safe artifacts:** ship one minimal, reproducible, signed image that behaves the same across environments.
- **Enable multi-tenant PaaS on one host:** enforce resource limits and admission policy to prevent interference and keep SLOs predictable.

---

*Thuật ngữ:* multi-tenant = *đa thuê*; non-root = *không quyền root*; LSM = *mô-đun bảo mật Linux*; SLO (Service Level Objective) = *mục tiêu mức dịch vụ*

# Review: Software-Defined Networking (SDN) in Cloud Platforms

- **Concept:** SDN separates the **control plane** (policy/intent) from the **data plane** (packet forwarding). In cloud, this is implemented by **Neutron + OVN/OVS**.
- **Control Plane:** define tenant networks, subnets, routers, ACL/QoS via APIs; store intent in OVN NB DB.
- **Data Plane:** virtual switches (OVS) and tunnels (VLAN/VXLAN/GRE/Geneve) stitch compute nodes.
- **Why it matters:** isolation between tenants/projects; programmable connectivity for any workload model.

---

*Thuật ngữ:* control plane = *mặt phẳng điều khiển*; data plane = *mặt phẳng dữ liệu*; OVN/OVS = *Open Virtual Network / Open vSwitch*; ACL/QoS = *kiểm soát truy cập/chất lượng dịch vụ*

# Review: Cloud Storage Models (Block / Object / File)

| Model | Access & Protocol | Typical Services | Strengths | Limitations |
|---|---|---|---|---|
| **Block** | Attached volume to host; iSCSI/NVMe | Cinder, EBS, PD, Azure MD | Low latency; consistent IOPS; boot/system disks, DBs | Per-host attach; AZ affinity; snapshot lifecycle mgmt |
| **Object** | HTTP(S) REST; key–value objects; multipart | Swift, S3, GCS, Azure Blob | Massive scale; durability (e.g. "11 nines"); lifecycle tiers | Higher latency; eventual consistency (class-dependent); non-POSIX |
| **File** | Shared NFS/SMB namespace; multi-client mount | Manila, EFS/FSx, Azure Files, Filestore | Shared tree; lift-and-shift; simple app portability | Throughput/latency tier-bound; scaling limits; cost at scale |

*Thuật ngữ:* block/object/file = *khối/đối tượng/tệp*; IOPS = *số thao tác I/O/giây*; AZ = *khu vực sẵn sàng*; POSIX = *chuẩn giao diện hệ tệp*

# Review OpenStack case study: Block / Object / File Storage

- **Block (Cinder):** low-latency *volumes* for OS/DB; snapshots for backup/clone.
- **Object (Swift):** HTTP objects for backups/logs/media/data lake; versioning + lifecycle.
- **File (Manila):** shared NFS/SMB for multi-host app state or legacy lift-and-shift.
- **Operator model:** types/classes, quotas, retention, replication/QoS policies.

*Thuật ngữ:* retention = *lưu giữ*; replication = *sao chép*; quota = *hạn ngạch*

# Review OpenStack case study: Cinder (Block Storage)

**Goal:** create, attach, and mount a persistent block volume to a VM.

## 1) Inspect volume types & quotas

```
openstack volume type list
openstack limits show --absolute | grep VOLUME
```

## 2) Create a 10GiB volume (choose a type)

```
openstack volume create --size 10 --type lvmdriver --availability-zone nova demo-vol
openstack volume show demo-vol -f table
```

## 3) Attach to a server and format/mount in-guest

```
openstack server add volume <VM_NAME> demo-vol
# Inside VM:
lsblk
sudo mkfs.ext4 /dev/vdb
sudo mkdir -p /data && sudo mount /dev/vdb /data
```

## 4) Persist mount (optional)

```
echo "/dev/vdb /data ext4 defaults 0 0" | sudo tee -a /etc/fstab
```

# Review OpenStack Swift — Public URL (readable container)

**Goal:** create a container, upload/download objects, and fetch via a public URL.

## 1) Create a container and list

```
openstack container create demo-container
openstack container list
```

## 2) Upload & list objects

```
openstack object create demo-container report.pdf
openstack object list demo-container
```

## 3) Make container publicly readable (demo use)

```
openstack container set --public demo-container
```

## 4) Construct & fetch the URL

```
SWIFT_URL=$(swift stat -v | awk -F': ' '/StorageURL/ {print $2}')
curl -I "$SWIFT_URL/demo-container/report.pdf"
```

*Thuật ngữ:* public read=*công khai đọc*; endpoint=*điểm cuối*

# Review OpenStack Swift — Temp URL (time-limited)

**Goal:** Keep container private but share a signed, time-limited URL.

## 1) Set an account-level Temp-URL key

```
KEY=$(openssl rand -hex 16)
openstack object store account set --property Temp-URL-Key="$KEY"
```

## 2) Gather account  storage URL

```
ACCOUNT=$(swift stat -v | awk -F': ' '/Account/ {print $2}')
SWIFT_URL=$(swift stat -v | awk -F': ' '/StorageURL/ {print $2}')
```

## 3) Generate a signed URL (valid 3600s)

```
SIG=$(swift tempurl GET 3600 "/v1/$ACCOUNT/demo-container/report.pdf" "$KEY")
```

## 4) Fetch via Temp URL

```
curl -I "$SWIFT_URL/demo-container/report.pdf?$SIG"
```

---

*Thuật ngữ:* Temp URL=*liên kết tạm thời*; signed URL=*liên kết có chữ ký*

# Review OpenStack case study: Manila (File Storage)

**Goal:** create an NFS share, allow access, mount from a VM.

## 1) Inspect share types

```
openstack share type list
```

## 2) Create a 10GiB NFS share

```
openstack share create nfs 10 --name demo-share
OpenStack share show demo-share -f table
```

## 3) Allow access to a VM IP (adjust IP)

```
openstack share access allow demo-share ip 192.168.1.10
```

## 4) Mount in the VM

```
sudo apt-get update && sudo apt-get install -y nfs-common
sudo mkdir -p /mnt/share
# Replace <MANILA_EXPORT> with 'server:/export/path' from share show
sudo mount -t nfs <MANILA_EXPORT> /mnt/share
```

---

*Thuật ngữ:* share = *chia sẻ tệp*; export (NFS) = *điểm xuất khẩu*; allow access = *cấp quyền truy cập*

# Security & Operations Tips (Block/Object/File)

**Block (Cinder)**

- Prefer *encrypted volume types*; snapshot/backup routinely; detach before delete.
- Keep volume & VM in same AZ unless replication is configured.

**Object (Swift)**

- Enable versioning/lifecycle; use *temp URLs* for limited sharing; consider CDN.
- Set metadata: content-type, cache-control for static hosting.

**File (Manila)**

- Limit by IP/user/AD; per-share quotas; monitor throughput/latency.
- For many writers, validate file-locking vs. app semantics.

---

*Thuật ngữ:* encrypted volume = *ổ mã hoá*; lifecycle policy = *chính sách vòng đời*; quota = *hạn ngạch*

# Review OpenStack case study: Quick Comparison

| Service | Model | Access/Protocol | Best For | Watch Out |
|---|---|---|---|---|
| Cinder | Block | iSCSI/NVMe (to host) | DBs, boot/system, low latency | AZ/attach affinity; snapshot backups |
| Swift | Object | HTTP REST (SDK/CLI) | Backups, logs, media, data lake | Higher latency; not POSIX; class consistency |
| Manila | File | NFS/SMB (multi-host) | Shared content, home dirs | Throughput tiering; file locking semantics |

*Thuật ngữ:* iSCSI = *kết nối khối qua IP*; NVMe = *giao diện lưu trữ nhanh*; POSIX = *chuẩn hệ tệp*

# Enhanced Tasks

1. Cinder: create 10GiB volume; attach to VM; format & mount at `/data`.
2. Swift: create container; upload a file; verify via CLI and show URL/endpoint.
3. Manila: create NFS share; allow VM IP; mount at `/mnt/share`; write/read test.

# VM vs. Container Architecture

| Aspect | Virtual Machine (VM) | Container |
|---|---|---|
| **Kernel and Isolation** | Each VM has its own guest kernel; isolation boundary is the *hypervisor*. | Share the host kernel; isolation via *namespaces* and *cgroups*, hardened by capabilities, seccomp, and *LSMs*. |
| **Provisioning and Boot** | High provisioning cost (allocate full OS, boot kernel); startup in seconds–minutes. | Lightweight provisioning; startup in milliseconds–seconds. |
| **Filesystem (Overlay; CoW)** | Independent disk images (e.g., QCOW2, VMDK). | Union/Copy-on-Write layers (e.g., `overlay2`); layer caching; use *bind mounts* or *block volumes* for write-heavy I/O. |
| **Performance vs. Security** | Strong isolation and stability, but higher resource overhead. | Higher density and efficiency; larger shared-kernel attack surface — requires defense-in-depth. |

# Cloud Application Isolation Models (I)

| Workload Type | How it is Deployed | Isolation & Scalability | Key Drawbacks |
|---|---|---|---|
| **Monolithic** | Whole stack in one unit (single VM / host). | Simple deployment; vertical scaling. | Tightly coupled; difficult partial updates; poor fault isolation. |
| **System-Containerized** | System containers (LXC, systemd-nspawn) provide a full userspace. | Better process isolation than raw services; suitable for legacy stacks. | Heavier than app containers; limited orchestration support; shared kernel. |
| **Application-Containerized (OCI)** | Microservices packaged as OCI images; run via container runtimes and orchestrators. | Fast startup, horizontal scaling, CI/CD friendly, fine-grained isolation. | State must be externalized; kernel exposure risk; networking/storage integration complexity. |

*Thuật ngữ:* OCI = *chuẩn container mở*; externalize = *tách trạng thái ra ngoài*

# Cloud Application Isolation Models (II)

| Model | Deployment / Runtime | Advantages | Limitations |
|---|---|---|---|
| **Micro-container / Sandboxed** | Micro-VMs or syscall sandboxes (Firecracker, gVisor, Kata). | VM-like isolation with near-container performance; good for multi-tenant workloads. | Increased complexity and some performance overhead. |
| **Ephemeral / FaaS** | Short-lived function containers (event-driven). | Extreme elasticity; cost-efficient for bursty tasks. | Cold-start latency; unsuitable for long-running stateful jobs. |

**Progression:** Installed Host → Isolated Service → Containerized App → Sandboxed Micro-container → Ephemeral Function

*Thuật ngữ:* ephemeral = *tạm thời*; sandbox = *môi trường cách ly*

# Isolation Boundaries (Security & Fault Containment)

- **Process vs VM boundary:** containers = shared kernel (process-level). For untrusted tenants use micro-VMs / Kata / Firecracker.
- **Namespace isolation:** PID/NET/MNT/IPC/UTS/User separate process trees, network stacks, mounts, and identities.
- **Resource isolation:** enforce cgroups limits (`cpu/memory/io/pids`) to avoid noisy-neighbor and DoS.
- **Privilege reduction:** drop unneeded capabilities; run non-root user; use `seccomp` filters; enable AppArmor/SELinux; mount rootfs read-only.
- **Network isolation:** per-namespace veths, network policies, vSwitch security groups (OVS/OVN), and optional eBPF packet/flow filtering.
- **Storage isolation:** place secrets on tmpfs or secret stores; mount volumes with `ro,nodev,nosuid,noexec` where applicable.
- **Fault containment:** use liveness/readiness probes, restart policies, and non-shared PID namespaces to limit propagation of failures.

---

*Thuật ngữ:* seccomp = *lọc syscall*; capabilities = *tập quyền Linux*; eBPF = *bộ lọc gói mở rộng*

# Scalability Mechanics (Density & Elasticity)

- **Horizontal scale (replicas):** immutable images allow fast cloning; stateless tiers scale linearly behind a load balancer.
- **Externalize state:** keep compute ephemeral; use managed DBs, object storage, and caches for session/state persistence.
- **Bin-packing & QoS:** declare resource requests/limits; use scheduler policies and QoS classes to meet p95/p99 latency SLOs.
- **Deployment strategies:** rolling, blue–green, and canary deployments to reduce blast radius during updates.
- **Autoscaling signals:** CPU/memory, queue length, RPS, latency percentiles; support scale-to-zero where appropriate.
- **Failure domains:** disperse replicas across hosts/availability zones; prefer anti-affinity for critical services.
- **Density vs safety:** increase density with quotas, limits and monitoring to avoid resource contention and noisy neighbors.

---

*Thuật ngữ:* SLO = *mục tiêu mức dịch vụ*; replica = *bản sao*

# Execution Substrates

| Substrate | How it Runs | When to Use | Notes |
|---|---|---|---|
| **Linux host (containerd/runc)** | Containers share host kernel; namespaces + cgroups + LSMs. | High-density stateless services; fast start. | Harden with non-root, seccomp, capability drops. |
| **Rootless containers** | User-namespace remapping, no host root. | Multi-tenant dev/CI and lower-privilege environments. | Feature limitations (net namespace, privileged operations). |
| **gVisor / Firecracker** | Syscall interception (gVisor) or micro-VMs (Firecracker). | Multi-tenant and untrusted workloads. | Stronger isolation; moderate performance overhead. |
| **Kata / light VMs** | Each pod runs in a minimal VM. | Zero-trust tenants, PCI/regulated workloads. | Lower density, longer cold-starts. |
| **GPU / NUMA tuned nodes** | Pin CPU/GPU and tune hugepages, I/O isolation. | ML training/inference, HPC. | Scheduler coordination needed (device plugins / topology hints). |

*Thuật ngữ:* NUMA(Non-uniform memory access) = *bộ nhớ bất đồng nhất*; device plugin = *trình cắm thiết bị cho scheduler*

# Container Stack, OCI and Supply Chain Security

- **Image model:** layered images, manifests, digests (sha256), tags, and multi-arch manifests.
- **Registry and policies:** private registries (Harbor, ECR, GHCR) with admission/gate policies, TTLs, and vulnerability scanning.
- **Runtime alignment:** prefer CRI-compatible runtimes (containerd / CRI-O) and validated low-level engines (`runc`, `crun`).
- **Supply-chain controls:** sign images (cosign), publish SBOMs (syft), and require attestation/provenance before deployment.
- **CI/CD integration:** automated build/test/sign $\rightarrow$ push $\rightarrow$ gated deploy pipeline with automated rollbacks on failures.

---

*Thuật ngữ:* SBOM(Software Bill of Materials) = *danh sách thành phần phần mềm*; cosign = *ký số image*

# Practical Hardening — Image & Execution

- **Least privilege (runtime):** run as non-root; drop Linux capabilities; read-only rootfs; enable `no-new-privileges`.
- **Syscall surface:** attach a minimal `seccomp` profile per workload class.
- **LSM enforcement:** AppArmor/SELinux in enforcing mode; mask/read-only sensitive paths; restrict device mounts.
- **Image provenance & SBOM:** pin `@sha256` digests; sign & verify images; publish SBOMs; scan before run.

---

*Thuật ngữ:* seccomp=*lọc syscall*; LSM (Linux Security Modules)=*mô-đun bảo mật Linux*; SBOM=*danh sách thành phần phần mềm*

# Practical Hardening — Runtime & Operations

- **Runtime choice by risk:** `containerd+runc` for trusted/high-performance; `gVisor`/`Kata` for untrusted multi-tenant workloads.
- **Resource controls (cgroup v2):** `cpu.max/weight`, `memory.high/max`, `pids.max`, I/O throttling; (on K8s) set requests/limits & QoS to protect SLOs.
- **Network & secrets:** default-deny policy; constrain egress; eBPF filters for lateral-movement detection; use Vault or KMS-backed secrets; mount secrets as files (tmpfs).
- **Observability & resilience:** immutable logs; metrics & audit (seccomp/LSM denials); alert on anomalous syscalls/spikes; health probes, canary/rolling, auto-rollback.

---

*Thuật ngữ:* cgroup v2=*nhóm điều khiển phiên bản 2*; KMS (Key Management Service)=*dịch vụ quản lý khóa*; SLO (Service Level Objective)=*mục tiêu mức dịch vụ*

# Trade-offs and Decision Guidance

- **Performance vs Isolation:** pure containers give best density/perf; micro-VMs/sandboxes give stronger isolation at cost of density.
- **Operational complexity:** stronger isolation increases surface area for ops (image signing, attestation, orchestration complexity).
- **When to choose what:**
  - High-trust internal services → containerd/runc on hardened hosts.
  - Multi-tenant or untrusted code → gVisor / Kata / Firecracker.
  - Event-driven short jobs → FaaS / ephemeral containers.
- **Rule of thumb:** start with secure defaults, progressively relax only when evidence (benchmarks / telemetry) justifies it.

---

*Thuật ngữ:* ephemeral = *tạm thời*; FaaS = *Function-as-a-Service*

# runc

- **runc** is a lightweight, portable container runtime and the reference implementation of the OCI Runtime Spec.
- Originally extracted from Docker's *libcontainer*; now maintained under the **opencontainers** (OCI) org.
- Executes containers by setting up:
  - Linux **namespaces** (PID, network, mount, etc.)
  - **cgroups** for resource limits
  - Container filesystem and process entrypoint
- Invoked by higher-level runtimes (e.g., `containerd`, `CRI-O`) to spawn containers.

---

*Thuật ngữ:* `runc` = lớp runtime OCI thấp nhất, trực tiếp tạo tiến trình cách ly

# containerd

- **containerd** is a **daemon-level runtime** that manages container lifecycle.
- Originated from Docker; now a **CNCF** project.
- Provides:
  - Image pull/push and **content store**
  - Snapshot management (e.g., `overlayfs`, `btrfs`, `zfs`)
  - Create/start/stop via OCI runtimes (e.g., `runc`, `crun`)
- No built-in networking; integrates with **CNI** via higher layers (e.g., Kubernetes CRI plugin).

---

*Thuật ngữ:* containerd = runtime trung gian điều phối ảnh, snapshot, và gọi `runc`/`crun`

# CRI-O

- **CRI-O** is a lightweight runtime designed specifically for **Kubernetes**.
- Implements the **Container Runtime Interface (CRI)** natively.
- Uses:
  - `runc` or `crun` for low-level execution (OCI runtime).
  - `conmon` to supervise container processes and I/O.
- Image and storage via `containers/image` and `containers/storage` libraries (Podman/Buildah are sibling tools, not dependencies).
- Provides a minimal, secure, and fast runtime for Kubernetes nodes.

---

*Thuật ngữ:* CRI-O = runtime cho Kubernetes (CRI), dùng `runc`/`crun` + `conmon`

# crun

- **crun** is a fast, lightweight alternative to `runc`.
- Written in `C` (vs Go for `runc`), often lower memory and faster startup.
- Fully compliant with the OCI Runtime Specification.
- Common in Fedora/RHEL; widely used with `CRI-O`.

---

*Thuật ngữ:* crun = thay thế `runc`, tối ưu hiệu năng/bộ nhớ

# gVisor

- **gVisor** is a user-space kernel for containers (Google).
- Intercepts syscalls (via `runsc`) to sandbox workloads—adds an extra isolation layer.
- Trade-off: lower performance vs native `runc`.
- Suited for multi-tenant/untrusted workloads (e.g., GKE Sandbox, Cloud Run).

---

*Thuật ngữ:* gVisor = runtime an toàn, mô phỏng/syscall trong user space

# Kata Containers

- Combines **lightweight VMs** with container interfaces (via a VMM such as KVM/Firecracker/QEMU).
- Each container runs inside a minimal VM $\rightarrow$ strong isolation boundary.
- Integrates with `containerd` and `CRI-O`.
- Used in multi-tenant cloud environments for stronger isolation.

---

*Thuật ngữ:* Kata = hybrid: container trong VM nhỏ, bảo mật cao

# Docker Engine (Runtime Layer)

- **Docker Engine** includes:
  - `dockerd` — daemon/API
  - `containerd` — lifecycle management
  - `runc` — low-level OCI runtime
- Handles image build/pull (BuildKit) and local container lifecycle.
- Still widely used outside Kubernetes for standalone deployments.

---

*Thuật ngữ:* Docker Engine = `dockerd` + `containerd` + `runc`

# Docker

- **Dockerfile:** `FROM/RUN/COPY/ENV/WORKDIR/USER/EXPOSE/HEALTHCHECK/CMD`.
- **CLI:** `build/run/exec/ps/logs/push/pull`; pin `@sha256` instead of floating tags.
- **Best practices:** multi-stage builds; non-root `USER`; distroless; cache layers; `HEALTHCHECK`.
- **Secrets/Config:** never bake secrets; prefer files mounted from a secret store or Docker/K8s secrets; avoid plain env for sensitive data.

---

*Thuật ngữ:* distroless = *ảnh tối giản không shell*; pin digest = *ghim theo băm*

# Container Platform — Overview

- A **container platform** provides full lifecycle for *single-host* containers:
  - Build, store, run, and secure containers (optionally compose multi-container apps).
  - Integrates with *orchestrators* for cluster scheduling/scaling; ties into CI/CD and monitoring.
- Built **on top of container runtimes** (e.g., `containerd`, `CRI-O`).
- Examples: `Docker`, `Podman`, `OpenShift (platform product built on Kubernetes)`.

*Thuật ngữ:* Platform = tầng quản lý build/run/single-host, dựa trên runtime; scale cụm do orchestrator

# Platform vs Orchestration (Concept)

- **Container Platform:** developer/runtime tooling for *build, image management, local run, developer workflows* (CLI, build, registry, compose). Examples: Docker, Podman, OpenShift (platform product).
- **Orchestration:** cluster control plane that *schedules, scales, networks, discovers, and heals* workloads across many hosts. Examples: Kubernetes, Docker Swarm (legacy).
- **Short rule:** Platform = "build + ship + run (single host/node)" — Orchestration = "schedule + scale + operate (cluster)".

*Thuật ngữ:* Platform = *nền tảng*; Orchestration = *điều phối cụm*

# Docker — Orchestration (note)

- **Docker Swarm:** built-in orchestrator mode in Docker Engine (services, simple scheduling) — now legacy/maintenance.
- **Positioning:** Swarm is an orchestration feature, not the same as Docker the platform. Modern clusters predominantly use Kubernetes.
- **Implication:** when you say "Docker", be explicit: developer platform (Engine/Compose) vs. orchestration mode (Swarm).

---

*Thuật ngữ:* Swarm = *chế độ điều phối của Docker (legacy)*; phân biệt rõ để tránh nhầm lẫn

# Comparison — Platform vs Orchestrator

| **Role** | **Platform (build/run/dev)** | **Orchestrator (cluster operate)** |
|---|---|---|
| Primary concerns | developer workflows, image build, local run, registries, single-node lifecycle | scheduling, scaling, service discovery, cluster networking, reconciliation |
| Examples | Docker Engine, Docker Desktop, Podman, OpenShift (platform product) | Kubernetes (kube-apiserver, scheduler), Docker Swarm (legacy) |
| Typical outputs | images, Compose files, OCI artifacts | Pods, Deployments, Services, NetworkPolicies |
| Interaction | runs workloads on a node via a runtime (containerd/runc) | delegates execution to CRI runtimes across nodes |

*Thuật ngữ:* Nói rõ “platform” hay “orchestrator” khi thiết kế kiến trúc

# Example: Multi-stage Dockerfile (Node.js)

```
# Build stage (needs dev deps)
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
# Runtime stage (production deps only)
FROM gcr.io/distroless/nodejs20-debian12
WORKDIR /app
COPY --from=build /app/dist ./dist
# Option B (simpler): add an intermediate "deps" stage to produce prod-only node_modules.
USER 10001:10001
ENV NODE_ENV=production
EXPOSE 3000
CMD ["dist/server.js"]
```

*Thuật ngữ:* multi-stage build = *xây nhiều giai đoạn*; runtime image = *ảnh chạy*

# Container Networking Basics

- **Drivers:** `bridge`, `host`, `macvlan`, overlay (CNI concepts).
- **Port mapping:** `-p host:container`; DNAT via iptables/nft; outbound SNAT.
- **Service discovery:** embedded DNS; Compose/K8s Services; health/readiness probes.
- **North/South vs. East/West:** publish/LB vs. service-to-service mesh/policies.

---

*Thuật ngữ:* CNI = *giao diện mạng container*; DNAT/SNAT = *dịch đích/nguồn*

# Container Storage & Configuration

- **Volumes:** bind mounts vs. named volumes; local/NFS/CIFS; avoid heavy writes on overlay2.
- **State:** separate data to persistent volumes; use snapshot/backup cycles.
- **Secrets/Config:** Swarm/K8s Secrets; never commit secrets; rotate periodically.

*Thuật ngữ:* bind mount = *gắn kết đường dẫn*; persistent volume = *ổ bền vững*; rotate = *xoay vòng*

# Platform Comparison

| Runtime/Tool | Strengths | Notes |
|---|---|---|
| Docker Engine | Rich ecosystem, UX, Compose | K8s prod uses containerd/CRI; rootless available |
| containerd | CRI standard, light, K8s-native | Needs CLI/UX (nerdctl) |
| CRI-O | K8s-focused, minimal surface | Narrow scope (K8s-centric) |
| Podman | Daemonless, rootless, Docker-like CLI | Compose via podman-compose |

*Thuật ngữ:* rootless = *không đặc quyền root*; daemonless = *không nền dịch vụ*

# Operational Considerations & Security

- **Provenance:** cosign signatures, SBOM (syft), attestations; reject unsigned images.
- **Least privilege:** non-root `USER`; drop caps; seccomp; AppArmor/SELinux; read-only FS.
- **Resource limits:** `-cpus`, `-memory`, `-pids-limit`; avoid noisy neighbors/DoS.
- **Policies:** admission control (OPA/Conftest); pin-by-digest; base-image allowlists.

---

*Thuật ngữ:* attestation = *xác nhận tạo tác*; allowlist = *danh sách cho phép*; admission control = *kiểm soát nhập cụm*

# Practice Overview

- **Task 1:** Provision OpenStack VM; install Docker via Ansible (idempotent).
- **Task 2:** Build multi-stage image; push to private registry.
- **Task 3:** Run container with limits; health checks; non-root user.
- **Task 4 (optional):** Integrate Neutron SGs, Cinder volumes; optionally Magnum/Zun.

---

*Thuật ngữ:* idempotent = *bất biến theo lặp*; registry = *kho ảnh*

# Ansible Snippet: Install Docker on OpenStack VM

```
- hosts: app_vms
  become: true
  tasks:
    - name: Install prerequisites
      package:
        name: [curl, ca-certificates, gnupg, lsb-release]
        state: present

    - name: Install Docker Engine
      shell: |
        install -m 0755 -d /etc/apt/keyrings
        curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo $ID)/gpg | \
          gpg --dearmor -o /etc/apt/keyrings/docker.gpg
        echo \
          "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
          https://download.docker.com/linux/$(. /etc/os-release; echo $ID) \
          $(. /etc/os-release; echo $VERSION_CODENAME) stable" | \
          tee /etc/apt/sources.list.d/docker.list > /dev/null
        apt-get update
        apt-get install -y docker-ce docker-ce-cli containerd.io
```

# Ansible Snippet: Build & Push Image

```yaml
- hosts: app_vms
  become: true
  vars:
    registry: "ghcr.io/yourorg"
    image_name: "wk7-sample"
    image_tag: "v1"
  tasks:
    - name: Copy app sources
      synchronize:
        src: ./app/
        dest: /opt/app/

    - name: Build image
      community.docker.docker_image:
        name: "{{ registry }}/{{ image_name }}:{{ image_tag }}"
        build:
          path: /opt/app
        push: no

    - name: Login and push
      community.docker.docker_login:
```

# Ansible Snippet: Run Container with Limits

```yaml
- hosts: app_vms
  become: true
  tasks:
    - name: Run container with resource limits
      community.docker.docker_container:
        name: wk7-web
        image: "ghcr.io/yourorg/wk7-sample:v1"
        state: started
        published_ports:
          - "80:3000"
        restart_policy: unless-stopped
        memory: "512m"
        cpus: "0.5"
        env:
          NODE_ENV: production
```

*Thuật ngữ:* published ports = *cổng công bố*; restart policy = *chính sách khởi động lại*

# Integrating with OpenStack

- **Neutron:** SG mở 80/443; Floating IP để publish; inbound via LBaaS khi cần.
- **Cinder:** mount volume làm thư mục dữ liệu (`-v /data:/var/lib/app`); snapshot/backup.
- **Glance/Packer:** base image đã cài Docker để boot nhanh và nhất quán.
- **Magnum/Zun (tuỳ chọn):** Container/K8s as-a-service trong OpenStack.

---

*Thuật ngữ:* SG = *nhóm bảo mật*; LBaaS = *cân bằng tải như dịch vụ*

# Mini Lab (Tóm tắt)

1. Cài Docker trên VM OpenStack; cấu hình SG cho 22/80.
2. Build image multi-stage; push lên registry riêng (pin digest).
3. Chạy container non-root; giới hạn CPU/RAM; thiết lập `HEALTHCHECK`.
4. Cấp Floating IP; kiểm tra HTTP từ Internet.
5. (Tuỳ chọn) Mount Cinder volume cho dữ liệu; thử backup/restore nhanh.

---

*Thuật ngữ:* backup/restore = *sao lưu/khôi phục*; non-root = *không đặc quyền root*

# Assessment & Deliverables

- **Báo cáo (2 trang):** so sánh VM vs. container (perf, security), kiến trúc triển khai, kết quả kiểm thử.
- **Minh chứng:** ảnh cài Docker; ảnh image trong registry; `docker ps`; endpoint công khai.
- **File nộp:** Dockerfile, playbooks Ansible, inventory, compose (nếu dùng).
- **Rubric (10 điểm):** kỹ thuật (4), bảo mật (3), tự động hoá (2), tài liệu (1).

---

*Thuật ngữ:* inventory = *danh mục máy*; endpoint = *điểm cuối*

# Threat Model Lite (STRIDE) cho Containers

- **Spoofing:** rò rỉ registry creds ⇒ secret-manager/KMS, token scope hẹp.
- **Tampering:** sửa image ⇒ ký số, pin digest, admission policy.
- **Repudiation:** thiếu audit ⇒ log build/push/pull, provenance/SBOM.
- **Info Disclosure:** secrets trong image ⇒ build-args/secret mounts.
- **DoS:** hog CPU/RAM ⇒ `-cpus`, `-memory`, `-pids-limit`.
- **EoP:** `-privileged` ⇒ tránh; drop caps; seccomp profile chặt.

---

*Thuật ngữ:* KMS = *hệ quản lý khoá*; EoP = *leo thang đặc quyền*

# Pitfalls & Anti-patterns

- **Floating tags:** dùng `:latest` gây drift $\Rightarrow$ luôn pin `@sha256`.
- **Run as root:** root + volume RW $\Rightarrow$ tăng rủi ro EoP.
- **overlay2 write-heavy:** workload I/O ngẫu nhiên $\Rightarrow$ chuyển sang volumes chuyên dụng.
- **Bloat images:** base image quá lớn $\Rightarrow$ multi-stage + distroless.

*Thuật ngữ:* drift = *trôi lệch cấu hình*; RW = *đọc-ghi*

# Wrap-up

- Cloud host workloads differ by **isolation boundary** and **scalability mechanics**.
- SDN/Storage choices shape connectivity & persistence; container practice shapes density & speed.
- Safe defaults: non-root, signed images, SBOM, pin digest, strict limits, secrets outside images.
- Next: **Orchestration with Kubernetes (K8s basics)**.

*Thuật ngữ:* isolation boundary = *ranh giới cô lập*; density = *mật độ*