# Week 7 — Container & Cloud-Native Forensics (Docker, containerd, Kubernetes)

Digital Forensics Course

Updated: October 2, 2025

- Focus: Docker/containerd artifacts, Kubernetes node/pod logs, live triage, memory & filesystem capture, supply-chain validation, eBPF runtime telemetry.
- Alignment: builds on Week 6 environment prep and host/network fundamentals.

# Outline

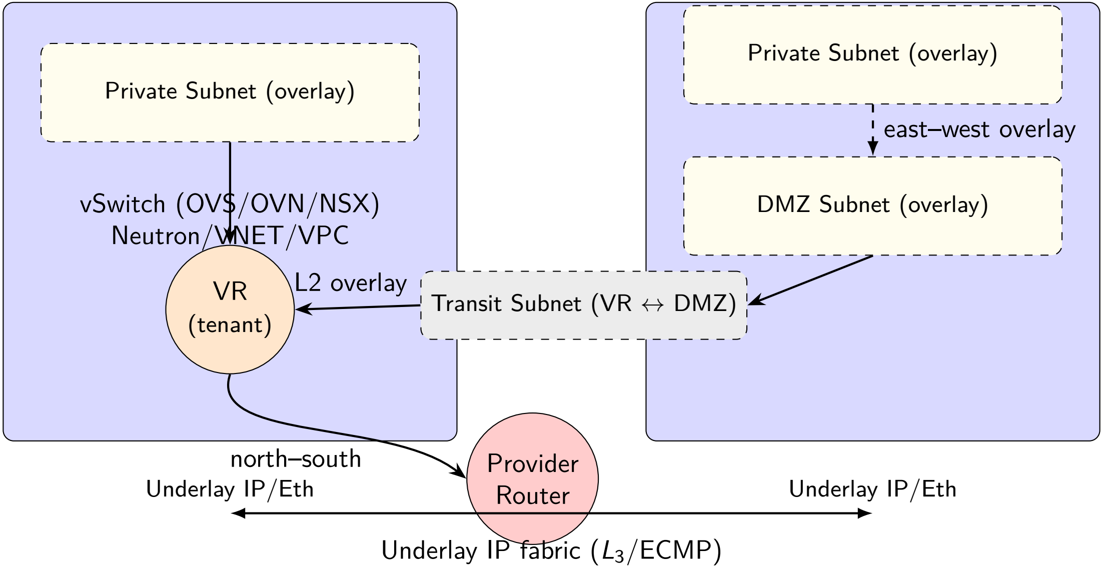
1. Foundations & Warm-up

2. Setup prerequisites

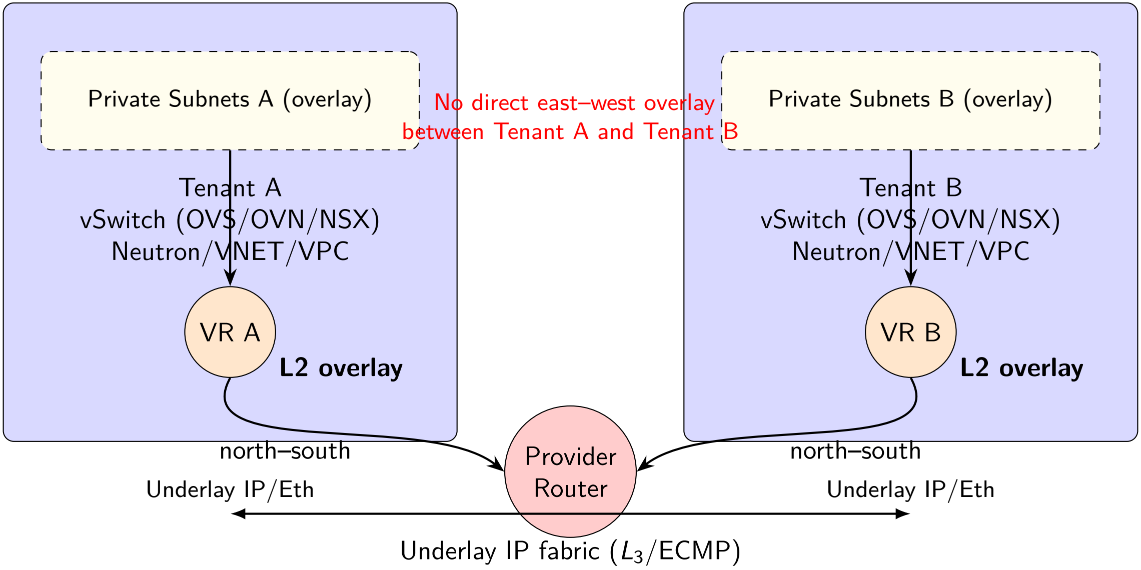
# Learning outcomes

- Explain container runtime architecture (Docker Engine vs. containerd/CRI-O) and implications for evidence.
- Locate and preserve core artifacts: images, layers, snapshots, volumes, logs, network state.
- Perform live triage (`docker`/`nerdctl`/`crictl`/`kubectl`) without tampering evidence.
- Acquire memory (process- or container-scoped) and filesystem snapshots (export, checkpoint).
- Reconstruct timelines across host → container → cluster.
- Validate image integrity (signatures) and generate SBOMs for supply-chain analysis.
- Instrument runtime telemetry with eBPF-based sensors (Falco/Tetragon) and interpret alerts.

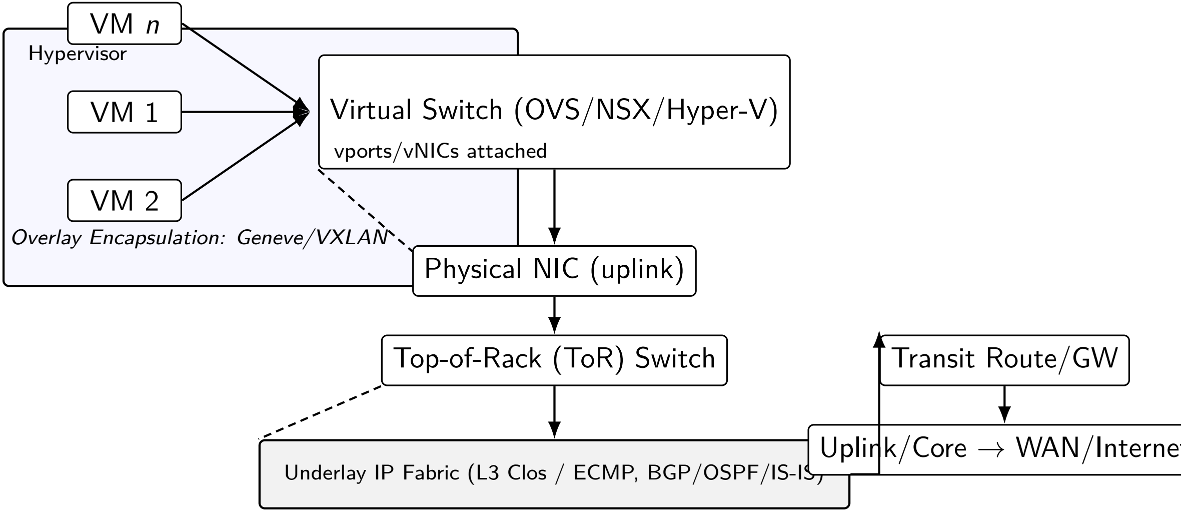
# Big Picture: Underlay vs. Overlay (Single tenant with Transit Subnet)



# Big Picture: Underlay vs. Overlay (Multi-Tenant Isolation)

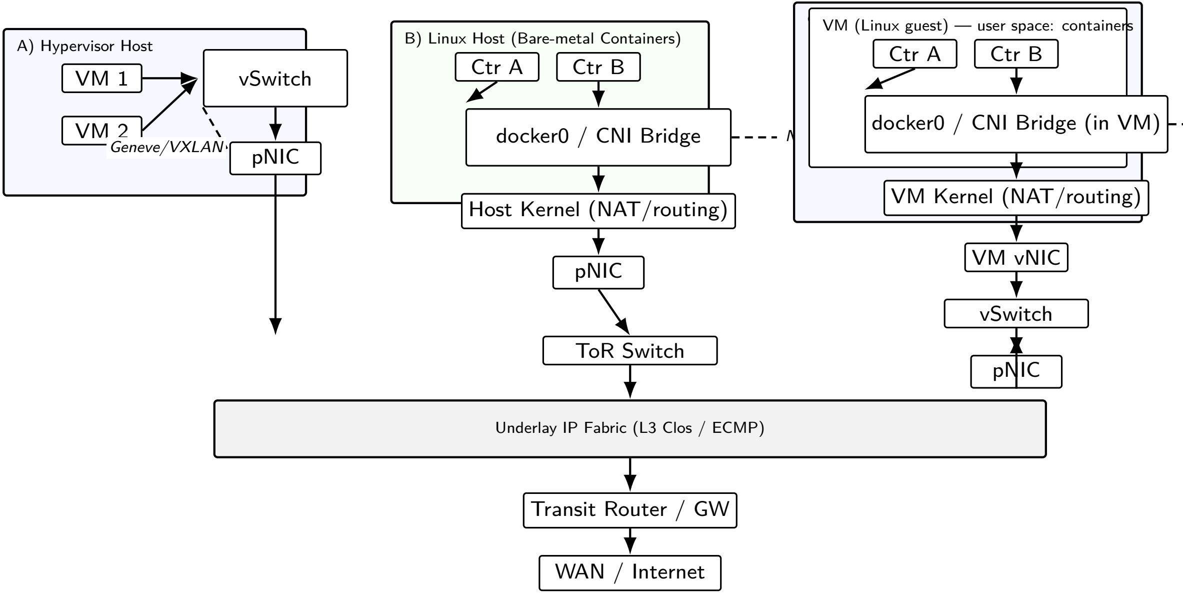


# Underlay vs. Overlay (Hypervisor → Fabric → Transit)


VM *n*
Hypervisor
VM 1
VM 2
Overlay Encapsulation: Geneve/VXLAN
Virtual Switch (OVS/NSX/Hyper-V)
vports/vNICs attached
Physical NIC (uplink)
Top-of-Rack (ToR) Switch
Transit Route/GW
Uplink/Core → WAN/Interne
Underlay IP Fabric (L3 Clos / ECMP, BGP/OSPF/IS-IS)

**Control planes (examples):** OpenStack Neutron / VMware NSX / AWS VPC / Azure VNet / GCP VPC
**Roles:** vSwitch encapsulates tenant traffic (overlay) → pNIC → ToR →
Underlay fabric forwards outer IP (no tenant state) → Transit GW for N–S or E–W between VRFs/tenants.

# VMs vs Containers (bare-metal) vs Containers-in-VM



A) Hypervisor Host

VM 1

VM 2

vSwitch

Geneve/VXLAN

pNIC

B) Linux Host (Bare-metal Containers)

Ctr A

Ctr B

docker0 / CNI Bridge

M

Host Kernel (NAT/routing)

pNIC

ToR Switch

VM (Linux guest) — user space: containers

Ctr A

Ctr B

docker0 / CNI Bridge (in VM)

VM Kernel (NAT/routing)

VM vNIC

vSwitch

pNIC

Underlay IP Fabric (L3 Clos / ECMP)

Transit Router / GW

WAN / Internet

# Core distinction: containers *are not* VMs

| | **Containerized app** | **Virtual Machine (VM)** |
|---|---|---|
| Isolation boundary | Namespaces & cgroups on host *kernel* | Full guest OS w/ *virtual hardware* |
| Kernel | Shared with host | Own kernel (guest) |
| FS artifacts | Image layers, RW diff, volumes | Guest disk image (`qcow2/vhdx/ebs`), guest FS |
| Process view | Host sees processes (`pid`) in namespaces | Host sees VM process; guest processes internal |
| Network identity | veth/bridge on host; per-pod/task ENI (cloud) | vNICs owned by guest; hypervisor bridges/switches |
| Evidence scope | Runtime logs, layers, container logs, node events | Guest disk/mem snapshots + in-guest artifacts |

# Warm-up: Setups

**Linux (lab hosts)**

- Kernel with cgroups v1/v2, namespaces; overlayfs enabled.
- Packages: `docker` (or `moby`), `containerd`, `jq`, `tar`, `iptables`, `conntrack`, `tcpdump`.
- Time sync (chrony/systemd-timesyncd), NTP reachable.
- Disk: separate `/var/lib/docker` volume if possible.

**Windows (WSL2 Docker Desktop)**

- Enable WSL2; install Docker Desktop (Linux containers).
- Ensure Hyper-V not conflicting with other hypervisors.
- Space for `%LOCALAPPDATA%\Docker\wsl\data\ext4`
- For Windows containers mode, enable HNS + Windows features.

# Forensic lab toggles (when you are the operator)

- Daemon: `live-restore=true`, log-level `info` (or `debug` temporarily), rotate logs conservatively.
- Always run with pinned **digests**; keep a registry mirror for reproducibility.
- Disable `-privileged`; use `-cap-drop=ALL` and add only what you need.
- Prefer **read-only** rootfs and tmpfs for runtime state; isolate writable data to a dedicated volume.
- Centralize logs to a durable sink; collect `/var/log/containers/`, `/var/log/pods/` on K8s nodes.

# Common topologies & what to snapshot first

- **A. Bare-metal Linux host** (Docker/containerd) ⇒ collect from `/var/lib/docker` or `/var/lib/containerd` on the host.
- **B. Docker *inside* a Linux VM** (KVM/ESXi/Hyper-V) ⇒ snapshot/export the *VM disk* first (qcow2 external), then collect container artifacts *inside* the guest copy.
- **C. Windows + WSL2 (Docker Desktop)** ⇒ containers live in a Linux utility VM; backing disk at `%LOCALAPPDATA%\Docker\wsl\data\ext4.vhdx`.
- **D. Windows containers** (no Linux) ⇒ HNS networks + `C:\ProgramData\docker\...`.
- **E. AWS EC2 (Nitro)** with ECS/EKS ⇒ take EBS snapshots; pull CloudWatch/VPC Flow Logs/CloudTrail; collect node/container artifacts if accessible.

# Acquisition decision tree (high level)

1. **Identify context**: containers vs VMs; Linux vs Windows; cluster (K8s/ECS) vs single host.
2. **Snapshot at the correct boundary first**: qcow2/avhdx/EBS as applicable.
3. **Work on copies**: mount/export snapshots; then collect container layers, logs, and configs.
4. **Cloud-native** timelines: add CloudWatch/Flow Logs/CloudTrail (ECS/EKS) or equivalent.
5. **Escalate** to memory/ephemeral debug only if necessary and approved.

# Warm-up roadmap

1. Docker on **Windows host (Hyper-V/WSL2)** — where the bits & logs live; how Hyper-V/WSL2 change handling.
2. Docker on **Linux host (bare-metal or KVM/QEMU VM)** — layering, storage, libvirt snapshots.
3. Docker on **AWS EC2 (Nitro/Xen legacy)** — what the Nitro System means; ECS/EKS basics.
4. **Simple orchestration for tasks** — Compose, systemd, K8s Jobs/CronJobs, and an ECS Task Definition.

# Docker on Windows host (Hyper-V/WSL2) — architecture

- **Two modes**: *Linux containers* run in a Linux VM (WSL2 default; Hyper-V backend possible). *Windows containers* share the Windows kernel with HNS-managed networking.
- **HNS networking modes**: `nat`, `overlay`, `transparent`, `l2bridge`, `l2tunnel` via Hyper-V vSwitch.
- **Evidence pointers**:
  - Docker Desktop WSL2 disk: `%LOCALAPPDATA%\Docker\wsl\data\ext4.vhdx`.
  - Windows engine logs: `C:\ProgramData\docker\containers\<id>\<id>-json.log` or in

`wsl`$\backslash docker - desktop - data\backslash data\backslash docker\backslash containers\backslash < id > .Daemonconfig :$ `C:\ProgramData\docker\config\daemon.json`.

# Windows warm-up: quick triage & snapshots

```
# Discover Docker root & containers
docker info --format ’{{.DockerRootDir}}’
docker ps --no-trunc
docker inspect <cid> | jq ’.[0] | {Id,Image,LogPath,Mounts}’

# Check HNS networks
Get-HnsNetwork | ConvertTo-Json -Depth 5

# If Hyper-V backend: locate VHDX / checkpoints
Get-VM; Get-VHD -VMName <vm> | ft Path,FileSize
Get-VMSnapshot -VMName <vm> | ft Name,CreationTime
```

# Docker on Linux host (bare-metal or KVM/QEMU guest)

- **Engine/runtime**: dockerd → containerd; overlay2 snapshotter; default logging driver `json-file`.
- **Container artifacts**:
  - Root: `/var/lib/docker/` (images/layers/containers/volumes).
  - **Volumes**: `/var/lib/docker/volumes/` (named volumes; bind mounts live on host paths).
  - Logs: `/var/lib/docker/containers/<id>/<id>-json.log`.
  - Config: `/etc/docker/daemon.json` (logging rotation, driver).
  - **Rootless Docker**: data root under `$HOME/.local/share/docker/` (adjust paths accordingly).
- **If host runs under KVM/QEMU**: guest disks in `/var/lib/libvirt/images/*.qcow2`; use *external* snapshots (`-disk-only -atomic`).

# Linux warm-up: libvirt quick actions

```
# List pools and locate guest images
sudo virsh pool-list --all
sudo virsh vol-list default
# Create external snapshot while VM runs
sudo virsh snapshot-create-as --domain vm1 snap1 \
  --disk-only --atomic \
  --diskspec vda,file=/var/lib/libvirt/images/vm1-snap1.qcow2

# Inspect and export container FS (work on copies)
sudo docker ps --no-trunc
sudo docker export <cid> -o /evidence/<cid>.tar
```

# Docker on AWS EC2 (Nitro / Xen legacy)

- **Substrate**: EC2 is *Nitro-first*; treat Xen as *legacy* (migrate where possible).
- **EKS/ECS AMIs**: Prefer *AL2023* or *Bottlerocket*. Note: EKS AL2-optimized AMIs stop publishing on **Nov 26, 2025**. Defaults: `containerd` + `kubelet`; Windows EKS AMIs include `containerd`, `kube-proxy`.
- **ECS networking (awsvpc)**: *1 ENI/IP per task* for precise attribution; consider *ENI trunking* to raise task density; mind ENI/SG/subnet limits.
- **Forensic acquisition**: take *EBS snapshots* (volumes); correlate *CloudWatch Logs*, *VPC Flow Logs*, and *CloudTrail*; collect node/container artifacts if instance access is allowed.
- **Logging/telemetry**: use `awslogs` or `firelens` (to CloudWatch/S3/OpenSearch). EKS commonly ships with *Fluent Bit* / *Container Insights* (or OTel) for durability.

# AWS warm-up: evidence hooks

```
# Map tasks to ENIs (ECS, awsvpc)
aws ecs list-tasks --cluster <name>
aws ecs describe-tasks --tasks <ids> --cluster <name> \
  --query ’tasks[].attachments[].details[?name==‘networkInterfaceId‘].value’

# VPC Flow Logs & CloudWatch (identity & timing source of truth)
aws logs describe-log-groups --log-group-name-prefix /aws/ecs/
aws ec2 describe-network-interfaces --filters Name=group-id,Values=<sg-id>
aws ec2 describe-network-interfaces --network-interface-ids <eni-id>
```

# Simple orchestration for tasks: options & trade-offs

- **Docker Compose**: quick multi-container orchestration on a single host (compose v2). Good for labs and small services.
- **systemd**: ensure containers/compose stacks start on boot; control logs via journald or redirect to log drivers.
- **Kubernetes Jobs/CronJobs**: reliable batch/periodic tasks with retries/backoff; declarative.
- **AWS ECS Task Definitions**: container specs, roles, networking (`awsvpc`), logging (`awslogs`/`awsfirelens`).
- **Logging durability note**: json-file rotation alone is not durable; forward to centralized sinks (Fluent Bit/Vector/ELK).

# Compose: minimal warm-up

```
# docker-compose.yml
services:
  web:
    image: nginx:stable
    ports: ["8080:80"]
    logging:
      driver: local
      options: {max-size: "10m", max-file: "3"}
  sidecar:
    image: busybox
    command: ["sh","-c","while true; do date; sleep 10; done"]
    restart: unless-stopped
```

# systemd unit for Compose stack

```
# /etc/systemd/system/myapp.service
[Unit]
Description=My Compose Stack
Requires=docker.service
After=docker.service

[Service]
Type=oneshot
WorkingDirectory=/opt/myapp
ExecStart=/usr/bin/docker compose up -d
ExecStop=/usr/bin/docker compose down
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

# Kubernetes: Job & CronJob (warm-up)

```
# job.yaml
apiVersion: batch/v1
kind: Job
metadata: {name: demo-job}
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - name: task
        image: busybox
        command: ["sh","-c","echo HELLO && sleep 2 && exit 0"]

# cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata: {name: demo-cron}
spec:
  schedule: "*/15 * * * *"
  jobTemplate:
    spec:
```

# AWS ECS: task definition (snippet)

```
{
  "family": "sample-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256", "memory": "512",
  "containerDefinitions": [{
     "name": "web",
     "image": "public.ecr.aws/nginx/nginx:latest",
     "portMappings": [{"containerPort": 80}],
     "logConfiguration": {
       "logDriver": "awslogs",
       "options": {
         "awslogs-group": "/ecs/sample",
         "awslogs-region": "ap-southeast-1",
         "awslogs-stream-prefix": "web"
       }
     }
  }]
}
```

# Recap & Lab Environment

**Prereqs (Week 6):**

- VMs on OpenStack (or local hypervisor) with Linux hosts and a small K8s lab (kind/minikube/microk8s/cluster).
- Analyst WS with: `docker`/`nerdctl`, `ctr`, `crictl`, `kubectl`, `jq`, `tar`, `gdb`/`gcore`, `tcpdump`, `ss`, `conntrack`, `bpftrace` (optional).
- Optional: `falco` or `tetragon`; `cosign`, `syft`, `grype`, `dive`.

# Container architecture & runtimes

- **Docker Engine**: CLI ↔ daemon; overlay2 storage; `docker0` bridge.
- **containerd**: CRI-compatible; tools `ctr`, `nerdctl`; snapshotters (overlayfs).
- **CRI-O**: Lightweight CRI runtime used by K8s.
- **K8s CRI**: Kubelet → (containerd/CRI-O); use `crictl` for low-level queries.
- **Key**: containers share host kernel ⇒ evidence spans namespaces (mnt, pid, net).

## Terminology

Image → Read-only layers
Container → RW layer + namespaces
Snapshot → runtime view for fs
Volume → persistent bind/managed storage

# Host artifacts — Docker (typical paths)

- `/var/lib/docker/` ⇒ images, layers, containers, **volumes**.
- **Volumes**: `/var/lib/docker/volumes/` (named); *bind mounts* live on host paths.
- Logs (json-file): `/var/lib/docker/containers/<id>/<id>-json.log`.
- Config: `/etc/docker/daemon.json` (logging/rotation).
- Network: `docker0` bridge, iptables `DOCKER` chain, DNS per network.
- API socket: `/var/run/docker.sock` (privileged; preserve carefully).
- **Rootless Docker**: data root at `$HOME/.local/share/docker/` (paths shift accordingly).

# Host artifacts — containerd / CRI-O

- containerd data: `/var/lib/containerd/`
  - Content blobs: `io.containerd.content.v1.content/`
  - Snapshots (overlayfs): `io.containerd.snapshotter.v1.overlayfs/`
- CRI-O / Podman (rootful): `/var/lib/containers/` (images, overlay, volumes)
- Tools: `ctr -n k8s.io images ls`, `ctr -n k8s.io snapshots ls`, `crictl ps -a`, `nerdctl inspect`.

# Kubernetes node & pod logs (overview)

- Node-level files: `/var/log/containers/*.log` (often symlinked into `/var/log/pods/...`).
- **Multi-container pods**: always specify `-c <container>` with `kubectl logs`.
- CLI vs node files: prefer grabbing node log files as well to avoid truncation/rotation gaps.
- Useful CLI: `kubectl logs <pod> [-c <container>] [-previous]` for crash loops.
- Ephemeral debug: `kubectl debug` (workload or node profile). Forward logs off-node (ELK/Vector/Fluent Bit).

# Live triage — minimally invasive commands

```
# Identify containers and metadata
docker ps --no-trunc
docker inspect <cid> | jq ’.[0] | {Id,Name,Image,Mounts,LogPath,State,Config}’
docker logs --since 15m <cid>

# containerd / CRI: prefer crictl / nerdctl
crictl ps -a
crictl inspect <cid>
nerdctl -n k8s.io ps -a
nerdctl -n k8s.io inspect <cid>

# Kubernetes
kubectl get pods -A -o wide
kubectl logs -n <ns> <pod> -c <container> --previous
kubectl get events -A --sort-by=.lastTimestamp
```

# FS acquisition: export vs save vs commit

```
# export: snapshot of a CONTAINER’s filesystem (no history; excludes named volumes)
docker export <cid> -o container_fs.tar

# save: archive of IMAGE + LAYERS (preserves history)
docker save <image:tag> -o image_layers.tar

# commit: create new IMAGE from running container (mutates state; avoid in forensics)
docker commit <cid> case123/findings:forensic-freeze
```

# Filesystem acquisition (containers/images)

```
# containerd snapshots / images
ctr -n k8s.io images export image_layers.tar <image:tag>
ctr -n k8s.io snapshots ls

# Copy specific paths (avoid modifying mtime)
docker cp <cid>:/app/logs ./case123/app_logs
kubectl cp -n <ns> <pod>:/var/log ./case123/pod_logs
```

# Memory & process capture (situational)

```
# Process-level dump with gcore (PID from docker/crictl top)
sudo gcore -o /evidence/p12345.core 12345

# Kubernetes ephemeral debugger (attach tools; then gcore/tcpdump/strace)
kubectl debug -n <ns> <pod> -it --image=nicolaka/netshoot --target=<container>
# Node debug pod (host namespaces; for net/iptables/conntrack capture)
kubectl debug node/<node> -it --image=nicolaka/netshoot --profile=general

# CRIU (if enabled) - checkpoint a container
docker checkpoint create <cid> chkpt1
```

# Network state & packet capture

```
# Host view: bridges, NAT, conntrack
ip link; ip addr; ip route
sudo iptables -S; sudo iptables -t nat -S
sudo conntrack -L | head

# Container netns capture (Linux)
nsenter -t $(docker inspect -f ’{{.State.Pid}}’ <cid>) -n ss -tpna
nsenter -t $(docker inspect -f ’{{.State.Pid}}’ <cid>) -n tcpdump -i any -w /evidence/cid.
    pcap

# Kubernetes service context
kubectl get svc,endpoints -A -o wide
```

# Supply chain: image integrity & SBOM

- **Signatures**: verify with `cosign verify <ref>` (Sigstore) or registry-native signatures.
- **SBOM**: generate with `syft <ref> -o spdx-json` or `cyclonedx-json`; scan with `grype <ref>`.
- **Layer inspection**: `dive <ref>` to review files, history, and potential secret leakage.
- Record hashes (digest), signer identity, SBOM files in your evidence manifest.

# Runtime telemetry with eBPF

- **Falco**: rule-driven syscall monitoring; detect shells in containers, sensitive file access, privilege escalation patterns.
- **Tetragon**: eBPF-based observability and enforcement (process/file/network), K8s identity-aware policies.
- Guidance: run sensors in *observe* mode during evidence collection; export alerts with timestamps for timeline correlation.

# Incident timeline reconstruction

- Correlate: container lifecycle (create/start/stop), image pulls, log entries, host auth events, K8s events, runtime alerts.
- Derive attacker actions: exec sessions, outbound connections, dropped binaries, modified configs, new crons.
- Validate persistence: bind-mounted volumes, hostPath mounts, startup commands, sidecars, init containers.

## Common pitfalls & anti-patterns

- Using `exec` to poke around first (modifies state); prefer exports/copies, then work on copies.
- Deleting pods/containers before collecting node logs and events.
- Ignoring `-previous` logs for crash loops & not specifying `-c` for multi-container pods.
- Missing rotated logs due to small `max-size`/`max-file` policy or no forwarding pipeline.
- Overlooking runtime CVEs and not checking host for lateral movement.

# Hands-on lab (60–75 min)

**Scenario**: web app in a container exploited to drop a miner.

1. Identify suspect container; confirm topology; snapshot at the correct boundary.
2. Export FS and collect logs (node files + CLI); include volumes if relevant.
3. Generate SBOM; verify signature; scan for known vulns.
4. Inspect network connections (namespaces), capture short PCAP.
5. If safe, attach ephemeral debug container; dump target process memory.
6. Build a timeline across host logs, container logs, K8s events, Falco/Tetragon alerts.
7. Write a short incident note: IOCs, root cause, containment recommendations.

# Assessment & deliverables

- Evidence bundle: `container_fs.tar`, `image_layers.tar`, `*.log`, `*.pcap`, volumes (if used), SBOM (`spdx.json`/`cdx.json`), signature report.
- Timeline with at least 8 correlated events and supporting artifacts.
- 1–2 page analysis: attack path, mitigation, hardening checklist.

# Reference commands (quick sheet)

- `docker ps/inspect/logs/export/save/cp/commit`
- `ctr -n k8s.io images/snapshots`
- `nerdctl -n k8s.io ps/inspect/logs/cp`
- `crictl ps/inspect/top/logs/exec`
- `kubectl get/describe/logs -previous/cp/debug`
- `gcore`, `tcpdump`, `ss`, `conntrack`
- `cosign verify`, `syft`, `grype`, `dive`
- `falco`, `tetragon`

# Further reading & hardening (selected)

- Docker logging drivers & rotation, rootless Docker implications.
- Kubernetes logging architecture & ephemeral containers for troubleshooting.
- CRI runtimes (containerd/CRI-O) and node-level artifacts.
- Runtime security with Falco/Tetragon; eBPF observability.
- Image signing (Sigstore Cosign) and SBOMs (SPDX/CycloneDX).

Questions & discussion