Docker Lab 4 — Container Lifecycle, Namespaces & cgroups

Author: Dr. Sandeep Kumar Sharma

Lab Description

Lab 4 introduces the low-level Linux primitives that power Docker: **namespaces**, **control groups (cgroups)**, and the **container lifecycle**. You will perform hands-on demonstrations to understand how Docker creates isolation for processes, file systems, users, networking, and resource limits. All examples use `ubuntu` or `nginx` containers with names like `sandeep-ns-demo`.

Topics Covered

- How a container starts and stops (the real lifecycle)
- Linux namespaces (PID, NET, UTS, IPC, MOUNT, USER)
- How Docker applies namespace isolation
- Understanding cgroups for CPU & memory limits
- Inspecting container namespaces using `nsenter`
- Resource constraint experiments

Learning Objectives

- Understand how Docker uses Linux primitives for isolation.
- Inspect and enter container namespaces manually.
- Apply CPU & memory limits using Docker.
- Analyze the lifecycle of a running container.

Learning Outcomes

- Ability to debug container resource issues.
- Ability to trace namespace isolation.
- Ability to configure production-grade resource limits.

Section 1 — Start an Ubuntu Container

```
docker run -d --name sandeep-ns-demo ubuntu sleep infinity
```

List processes inside the container:

```
docker exec -it sandeep-ns-demo ps aux
```

Section 2 — Inspect the Container's Namespaces

```
docker inspect sandeep-ns-demo | grep -i Pid
```

Use `nsenter` to enter namespaces:

```
sudo nsenter --target <PID> --mount --uts --ipc --net --pid bash
```

You are now in all namespaces of the container.

Section 3 — Demonstrate PID Namespace Isolation

Inside container:

```
ps -ef
```

PID 1 will be your process (sleep).

Outside the container:

```
ps -ef | grep sleep
```

PID is different → namespaces isolate processes.

Section 4 — Apply CPU & Memory Limits

Run container with limits:

```
docker run -d --name sandeep-cgroup-demo
  --cpus="0.5" --memory="256m" ubuntu sleep infinity
```

Inspect cgroup paths:

```
docker inspect sandeep-cgroup-demo | grep -i cgroup
```

Section 5 — Memory Stress Test

Enter container:

```
docker exec -it sandeep-cgroup-demo bash
```

Install stress tool:

```
apt update && apt install -y stress
```

Run test:

```
stress --vm 1 --vm-bytes 500M --timeout 10s
```

Container will be OOM-killed due to memory limit.

Section 6 — Cleanup

```
docker rm -f sandeep-ns-demo sandeep-cgroup-demo
```

Docker Lab 5 — Docker Overlay Networks (Multi-Host & Swarm Simulated)

Author: Dr. Sandeep Kumar Sharma

Lab Description

Lab 5 introduces **overlay networks**, the foundation of multi-host container networking. Even if Swarm mode is not enabled, you will simulate multi-host communication, understand VXLAN encapsulation, and create custom overlay networks. Container names will follow the pattern `sandeep-app1`, `sandeep-app2`, etc.

Topics Covered

- What is an overlay network?
- How overlay networking works (VXLAN, encapsulation)
- Service discovery in overlay networks
- Creating overlay networks
- Attaching services to overlay networks

Learning Objectives

- Understand how containers communicate across multiple hosts.
- Create and manage overlay networks.
- Inspect and test multi-container connectivity.

Learning Outcomes

- Ability to build distributed applications using overlay networks.
- Ability to debug multi-host networking issues.

Section 1 — Enable Swarm Mode (Required for Overlay)

```
docker swarm init
```

An overlay network requires Swarm.

Section 2 — Create an Overlay Network

```
docker network create -d overlay sandeep-overlay-net
```

Inspect:

```
docker network inspect sandeep-overlay-net
```

Section 3 — Deploy Services on Overlay Network

```
docker service create --name sandeep-app1
  --network sandeep-overlay-net nginx

docker service create --name sandeep-app2
  --network sandeep-overlay-net ubuntu sleep infinity
```

List services:

```
docker service ls
```

Section 4 — Test Connectivity

Get a container from service:

```
CID=$(docker ps | grep sandeep-app2 | awk '{print $1}')
docker exec -it $CID bash
```

Ping nginx service:

```
apt update && apt install -y iputils-ping
ping sandeep-app1
```

Service discovery works automatically.

Section 5 — Deploy Multi-Replica Services

```
docker service update --replicas 3 sandeep-app1
```

List tasks:

```
docker service ps sandeep-app1
```

Multiple tasks receive VIP-based load balancing.

Section 6 — Cleanup

```
docker service rm sandeep-app1 sandeep-app2
```

```
docker network rm sandeep-overlay-net
```

```
docker swarm leave --force
```

Lab 4 & Lab 5 Summary

- Lab 4 covered namespaces, lifecycle, and cgroups.
- Lab 5 introduced overlay networking and distributed communication.

You may now say: **"Create Lab 6"**.