

# Understanding Namespaces and Cgroups in Docker

How Linux Kernel Features  
(Namespaces and Cgroups) Enable  
Containerization

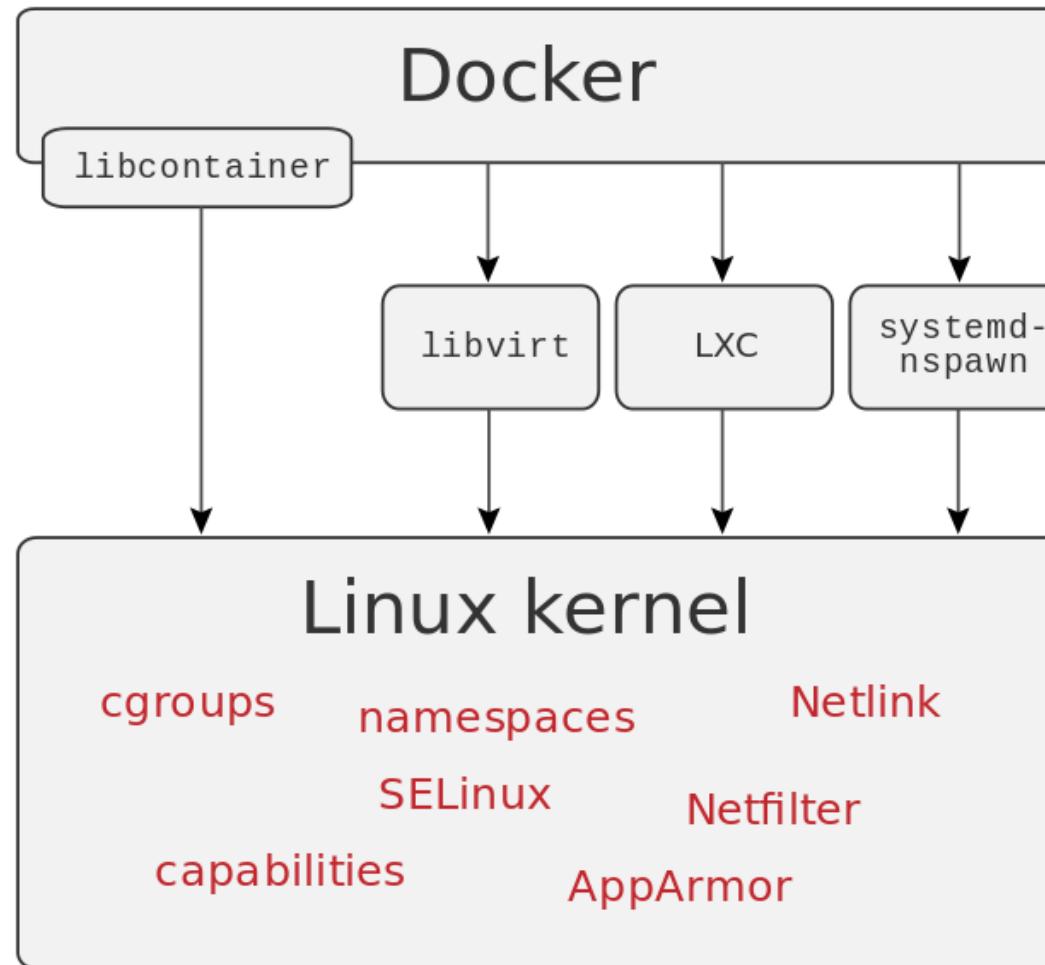
# Introduction to Containerization

- Docker containers are lightweight, portable, and isolated environments.
- Containers **share the host OS kernel**, unlike virtual machines.
- Docker takes advantage of **several features of the Linux kernel** to **deliver its functionality**.
- Two Linux kernel features used widely:
  - Namespaces (for isolation)
  - Cgroups (for resource limitation)

# Introduction to Containerization

- Namespaces and Cgroups are the basis of lightweight process virtualization.
  - As such, they forms the basis of Linux containers.
- What is **lightweight process virtualization** ?
  - A process that gives the user an illusion that he runs a linux operating system. You can run many such processes on a machine, and all such processes in fact share a single Linux kernel which runs on the machine.
  - This is opposed to hypervisor-based solutions, like Xen or KVM, where you run another instance of the kernel.

# Docker | Namespace | Cgroup



# What Are Namespaces?

- Linux kernel is a feature that isolates system resources for a process.
- Each container gets its own set of namespaces.
- Ensures isolation of processes, network, filesystems, etc.

# Types of Namespaces in Docker

| Namespace | Description                    | Isolated Resource          |
|-----------|--------------------------------|----------------------------|
| PID       | Process ID number space        | Process trees              |
| NET       | Network interfaces and routing | Interfaces, ports          |
| MNT       | Mount points                   | Filesystems                |
| UTS       | Hostname and domain name       | Hostname, domain           |
| IPC       | Inter-process communication    | Message queues, semaphores |
| USER      | User and group IDs             | UID/GID mappings           |
| CGROUP    | Cgroup access control          | Access to cgroup tree      |

# Visualizing Namespace Isolation

- Container A: PID 1 = bash, Hostname = web1, sees only its own processes
- Container B: PID 1 = nginx, Hostname = web2, sees only its own processes
- Host system sees both containers

# What Are Cgroups (Control Groups)?

- The cgroups (control groups) subsystem is a **Resource Management and Resource Accounting/Tracking solution**, providing a generic process-grouping framework.
- Linux kernel feature for limiting, accounting, and isolating resource usage.
- Docker uses Cgroups to enforce **resource constraints** on containers.
- Controls CPU, memory, block I/O, and devices.



# What Are Cgroups (Control Groups)?

- Cgroups are a Linux kernel feature that enable the **management and partitioning** of system resources by controlling the resources for a collection of processes.
- Administrators can use Cgroups to allocate resources, set limits, and prioritize processes.
- Docker utilizes Cgroups to **control** and **limit** the resources available to containers.

# What Are Cgroups (Control Groups)?

- Different types of available Cgroups include CPU Cgroup, memory Cgroup, block I/O Cgroup, and device Cgroup.
- While Cgroups are not explicitly designed for security, they play a crucial role in controlling and monitoring the resource usage of processes.



# How Cgroups Work

- Organizes processes into hierarchical groups.
- Resources can be:
- - Limited (e.g., max memory)
- - Prioritized (e.g., CPU shares)
- - Accounted (e.g., memory usage logs)

# Cgroup Controllers Used by Docker

| Controller | Resource Controlled              |
|------------|----------------------------------|
| cpu        | CPU usage                        |
| cpuset     | CPU cores a container can run on |
| memory     | RAM and swap usage               |
| blkio      | Disk I/O read/write rates        |
| devices    | Access to specific devices       |
| freezer    | Suspend/resume processes         |

# Example - Resource Limiting with Cgroups

- Example command:
  - `docker run -m 256m --cpus=1 ubuntu`
- Docker uses memory and cpu cgroup controllers to enforce limits.

# Combining Namespaces and Cgroups

- Namespaces isolate what a container can see.
- Cgroups control how much a container can use.
- Together provide:
  - Process & file system isolation
  - Network isolation
  - Resource-controlled environments

# Namespaces vs Cgroups

- Although namespaces and Cgroups may appear similar in definition, they are fundamentally different and serve different purposes.
  - Namespaces perform **isolation** by creating separate environments for processes that prevent one process from accessing or affecting other processes and/or the system.
  - In contrast, Cgroups **distribute and limit** resources like CPU, memory, and I/O among groups of processes. Often, namespaces and Cgroups are used together for process isolation and resource management.



# How Docker Uses these Internally

- When you run a container:
  - Docker creates namespaces for it
  - Docker assigns it to a new cgroup
  - Docker executes the container process in this isolated environment



# Docker Architecture Overview

- Docker Engine
- Namespaces → Isolate: PID, NET, MNT, etc.
- Cgroups → Limit: CPU, Memory, Disk I/O
- Container = Process + Namespaces + Cgroups + Filesystem



# Practical Use Case

- Running multiple microservices on the same host
- Each service in a separate container
- Namespaces ensure isolation
- Cgroups ensure fair resource allocation



# Conclusion

- Docker relies on Linux namespaces and cgroups.
- These are kernel-level features, not Docker-specific.
- Enable secure, isolated, resource-controlled containers.
- Essential knowledge for understanding Docker internals.



innovate

achieve

lead



# Docker Containers on Windows: Isolation Concepts

- Docker uses Linux namespaces and cgroups for isolation and resource control.
- On Windows, Docker Desktop uses WSL2 or Hyper-V to run a Linux kernel.
- Containers use Linux kernel features even on Windows, inside WSL2 VM.



# Windows vs Linux Containers

- Windows containers use Windows kernel; Linux containers use Linux kernel.
- Docker Desktop supports both, but only one mode at a time.
- Default is Linux containers (via WSL2 or Hyper-V).



# How to Switch Container Mode

- Right-click Docker Desktop tray icon to switch between Linux and Windows containers.
- Option may not appear if:
  - You're on Windows Home
  - Windows container support is disabled
- Alternate: Use PowerShell: DockerCli.exe - SwitchDaemon

# Why Ubuntu Image in Docker?

- Ubuntu image is a minimal userland snapshot — not a full OS.
- Used for package manager (apt) and developer familiarity.
- Still lightweight (~29MB), no GUI or systemd.
- Aligned with container philosophy: only needed tools included.

# Cross-Platform Docker Image Compatibility

- Docker images are OS kernel-specific.
- Windows container image ≠ compatible with Linux Docker engine.
- Error: 'image operating system "windows" cannot be used on this platform.'
- Solution: Rebuild from a Linux base (e.g., ubuntu, alpine).



# Summary: Key Points

- Docker Desktop runs Linux containers by default using WSL2.
- Windows containers require Windows Pro/Enterprise and are not portable.
- Ubuntu image is used for tooling, not for OS features.
- OS kernel mismatch prevents Windows images from running on Linux Docker hosts.

# References

- Linux man pages: `man namespaces`, `man cgroups`
- Docker Docs – Engine Internals:  
<https://docs.docker.com/engine/>
- Linux Kernel Docs – Control Groups:  
<https://www.kernel.org/doc/Documentation/cgroup-v1/>