Contents

Introduction

Control groups and namespaces are foundational features in the Linux kernel that enable containerization by providing resource control and isolation. Together they form the building blocks for container runtimes like Docker, Podamn and Kubernetes. To appreciate their role it is essential to understand each in depth and how they synergize to create the container abstraction.

Control groups (cgroups)

Control groups are a Linux kernel feature that provides mechanisms for limiting prioritizing and monitoring the usage of system resource such as CPU, memory and disk I/O, and network bandwidth. Introduced in 2007, cgroups allow processes to be grouped hierarchically and to apply resource constraints or quotas at at the group level. Cgroups work by organizing processes into "control groups", which are hierarchical structures where resource limits and accounting are defined. This hierarchy is represented as virtual filesystem (often mounted at `/sys/fs/cgroup`), with directories corresponding to cgroups. Each cgroup can enforce specific limits to priorities for a particular type of resource. For instance:

1. CPU and CPU Shares: Cgroups allow limiting the amount of CPU time a process or group of processes can use. For example, if two containers are running on the same host, you can allocate more CPU time to one container over the other.
2. Memory Limits: By setting memory constraints, cgroups ensure that a process cannot exceed a specific memory threshold, thereby preventing memory exhaustion on the host system.
3. Block in/out network bandwidth: these can be throttled to ensure fair sharing among processes or to priorities critical workloads.

Cgroups are hierarchical, meaning resource limits propagate downward in the tree. This hierarchy enables complex configurations, such as allocating a fixed percentage of CPU to a parent cgroup and then subdividing that allocation among child cgroups.

Without cgroups processes running on the same host would contend for resources without constraints leading to potential resource starvation or overuse. In the context of containers, cgroups ensure that each container

operates within its allocated resources, making them predictable, and performant even in multi tenant environment.

Namespaces

Namespaces, another Linux kernel feature isolate global system resources for groups of processes, by scoping resources, namespaces provide the illusion that a process or group of processes is running on its own system, separate from the others. This isolation is the backbone of containerization, as it ensures that processes inside a container are unaware of and unaffected by the processes and resources outside their namespace.

There are several types of namespaces, each targeting a specific system resource or function:

1. PID Namespace: Provides isolation for process ID. Processes inside a PID namespace see a separate process tree starting from PID 1, often the init system within the container. This ensures that processes in one container cannot see or signal processes in another container or the host
2. Mount namespace: isolates filesystem mount points. Each container can have its own root filesystem and mount points separate from the host or other containers. This is critical for providing a private file system view for containers.
3. UTS namespace: isolates system identifiers such as host name domain name. This allows containers to have their own hostname decoupled from the host system's identity
4. Network namespace: isolates network resources, including IP addresses routing tables and sockets. Containers can have their own virtual network interfaces and IP address, managed independently of the host or other containers.
5. `IPC` namespace: isolates inter process communication resources, such as shared memory and semaphores ensuring that contains cannot inadvertently interfere with one another's IPC mechanisms
6. User namespace: provides user and group ID isolation , allowing process inside a container to have different user ID from the host.

By leveraging namespaces each container can function as if it is running on a standalone system with a private set of resources. However, the host kernel orchestrates and manages these namespaces allowing containers to coexist on a shared kernel.

Cgroup and Namespaces

Cgroups and namespace complement each other to create the container abstraction:

1. Isolation - namespaces ensure that a container processes , network, filesystem and other system resources are isolated from the host and from the other containers. This isolation is crucial for security and the lightweight vitalization illusion that containers offer.
2. Resource control - control groups regulate how much of the resources isolated by the namespaces a container can consume, this combination prevents resource contention ensuring fair and predictable usage.

For example a container running a web server might operate within its own network namespace, with its own virtual NIC and IP address, mount namespace, with a private filesystem view, and PID namespace, with its own process tree. Simultaneously, cgroups ensure that the container cannot exceed 512MB of RAM and 50% of the CPU usage.

How they enable containers

Containers are not independent operating systems like virtual machines, they are processes running on the host operating system, constrained and isolated by cgroups and namespaces. These technologies allow containers to share the host kernel while appearing isolated, enabling the following key benefits

1. Lightweight - since containers share the host kernel, the require far fewer resources than VM which emulate hardware and run separate OS instances.
2. Fast start and shutdown - containers start quickly because they do not boot an entire operating system. They are essentially just processes with extra isolation, running on the host, just any other user level user process
3. Scalability - with cgroups enforcing resource limits, many containers can run concurrently on the same host without over provisioning

Modern container runtimes like Docker encapsulate these capabilities by automating the creation and management of namespaces and cgroups for each container, Tools like Kubernetes further build on this foundation to orchestrate containers across distributes systems, ensuring high availability and scalability.

In conclusion cgroups and namespaces by isolating and managing resources, transform the Linux kernel into a powerful platform for containerization. They encapsulate processes in a lightweight portable and secure manner, paving the way for the cloud native ecosystem that underpins much of today's software industry and infrastructure

More kernel components

OverlayFS

That is a union filesystem that allows multiple layers of filesystems to be stacked. It plays a key role in containerization by enabling the creation of lightweight writable layers on top of read-only image layers. The way it works is that when a container is created, a writable layer is added on top of a resad only base image. Any modification made by the container are written to this top layer, while the base image remains unchanged. This enables:

- Efficient sharing of base images between containers.
- Minimal disk usage since only changes are stored.
- Fast container creation by simply stacking layers

In containers the `OverlayFS` underpins the storage driver mechanism in Docker and other container runtimes, making image building sharing and running highly efficient.

Seccomp

Seccomp is a Linux kernel security feature that restricts the system calls (syscalls) a process can make. Containers use seccomp to reduce the attack vector / surface by preventing them from invoking potentially dangerous syscalls. The way this works is that seccomp operates as syscall filter. A process or container is provided with a list of allowed syscalls, any syscall not on the list is blocked. This is typically configured using a seccomp profile.

In containerization this play key role in protecting the host from container exploits by limiting access to sensitive syscalls, enables fine grained control over what containers can and cannot do or execute

Capabilities

Linux capabilities break down the all powerful root privileges into discrete units of authority. Containers often run as root within their own namespaces but they are stripped of unnecessary capabilities to mitigate risks. The way this works is by having the kernel allowing processes to drop or retain specific capabilities, like CAP_NET_ADMIN, for network management. For example a container may have the ability to bind to privileged ports but not modify kernel parameters

In containerization this enhances security by limiting what containers can do, even when running as root, reduces the risk of privilege escalation.

AppArmor and SELinux

These are two kernel frameworks for enforcing mandatory access control (MAC) policies. These frameworks restrict how applications or containers interact with the system. The way they work is that AppArmor defines file and process level access policies for containers, it is path-based meaning access is controlled based on file paths. SELinux uses labels to define granular access policies for processes, files and other resources.

In containerization ensures that containers can only access files, directories and resources explicitly allowed by their profiles or labels, also protects the host system from compromised or malicious containers.

Linux capabilities and No New Privileges (NNP)

Beyond capabilities the "No New Privileges" is a kernel flag feature that ensures a process cannot gain additional privileges through mechanism like `setuid` binaries. The way this works is that when `NNP` is enabled, even if a containerized process runs a binary with `setuid` permissions, it cannot escalate its privileges.

`setuid - short for set user identity allow users to run an executable with the file system permissions of the executable's owner, and to change behavior in directories. They are often used to allow users on a computer system to run programs with a temporarily elevated privileges to perform a specific task.`

Device control

The Linux kernel provides mechanism for controlling access to hardware devices through the device cgroup and the `mknod` syscall. The way this works is that the device cgroup allows fine grained control over which devices a container can access - block devices, character devices etc. Containers can also be restricted from creating new device nodes using `mknod`

In containerization prevents unauthorized access to host devices like storage or network interfaces, limits the potential for containers to interact directly with sensitive hardware

Network virtualization

Linux networking stack provides features that are critical for container networking, including virtual Ethernet (`veth`) pairs, network bridges and like `iptables`. The way this works containers are typically connected to the host network through virtual Ethernet pairs, where one end resides in the container's network namespace and the other in the host's namespace or bridge. Network bridges like `docker0` or `CNI` plugins create virtual networks to interconnect containers

Resource limits

Resource limits - `rlimits` provide per-process controls over resources like file descriptors, stack size, and CPU time. While not as flexible as cgroups, `rlimits` play a complementary role in containerized environments. The

way this works is that `rlimits` are set using the `setrlimit` syscall, constraining individual process behavior.

In containerization it is used to impose additional resource restrictions at the process level, prevents runaway resource consumption within a container by a rogue process