Containers

Contents

1. Container runtime

- A container runtime is a software component which works with operating system kernels and is responsible for both creating and running containers.

Creating Containers

- **Image Handling**:
  - The container runtime pulls the necessary images from a registry (e.g., Docker Hub).
- **Container Creation**:
  - It takes the image and creates a container from it, setting up the necessary environment and filesystem for the container.

Running Containers

- **Execution**:
  - The container runtime starts the container, executing the processes defined in the image.
- **Management**:
  - Manages the container's lifecycle, including starting, stopping, and restarting containers as needed.
- **Resource Allocation**:
  - It handles resource allocation (CPU, memory, I/O) to ensure that containers run efficiently and remain isolated from one another.
- **Networking**:
  - Manages the network settings and configurations for containers.

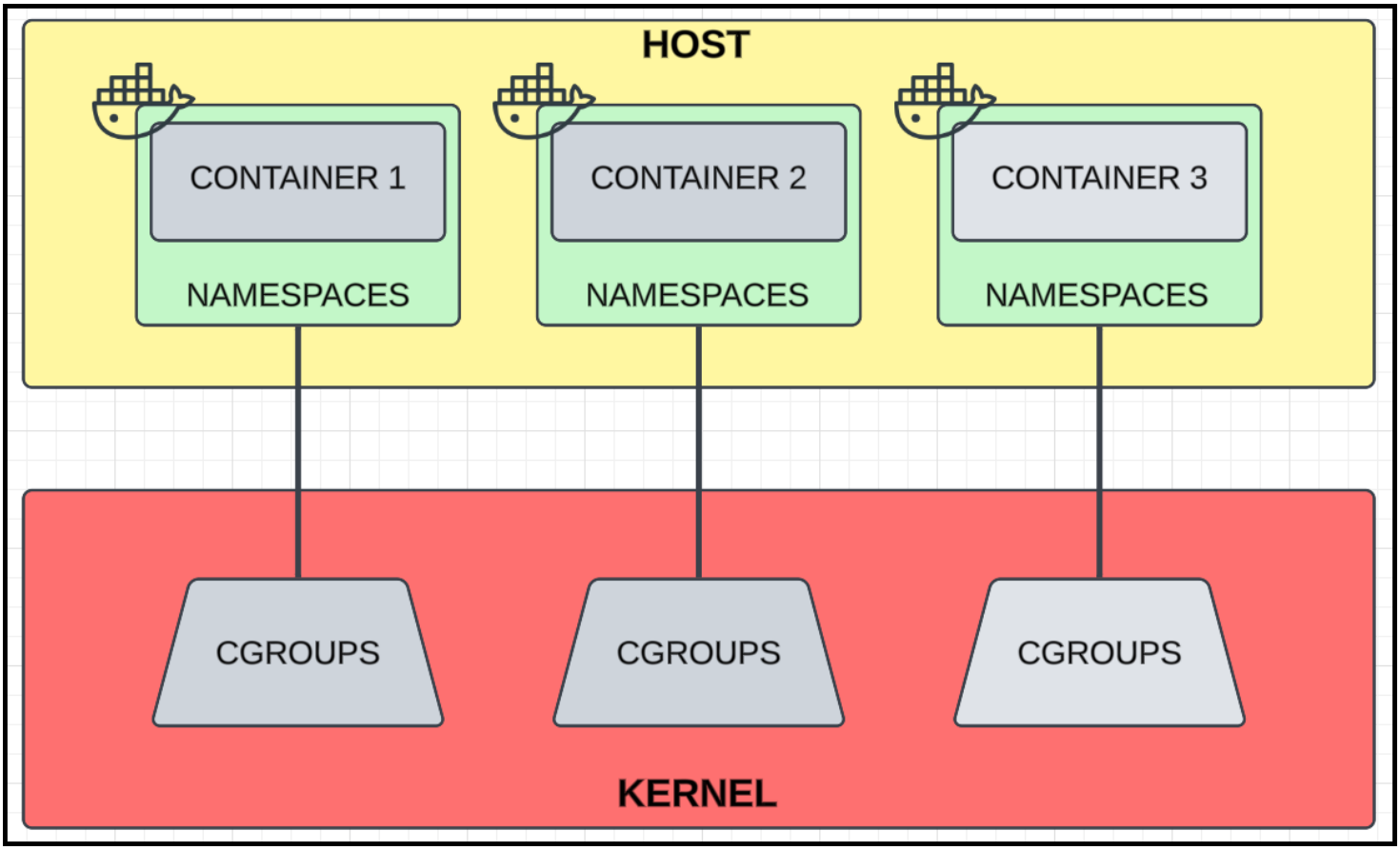
Key Responsibilities of a Container Runtime

1. Container Lifecycle Management
   - Create, start, stop, and delete containers.
   - Manage containerized processes.

2. Isolation
    - Set up namespaces and control groups (cgroups) for resource and process isolation (This we will see in the next section).
    - Ensure containers are sandboxed and cannot interfere with each other or the host.
3. Filesystem Management
    - Set up the container's filesystem using techniques like union filesystems (e.g., OverlayFS).
    - Mount container-specific directories and manage filesystem changes.
4. Networking
    - Set up the container's network stack (e.g., virtual Ethernet interfaces, bridges).
    - Handle container IP assignment and network isolation.
5. Interfacing with the Kernel
    - Use Linux kernel features like namespaces, cgroups, and capabilities to implement container functionalities.
6. Image Management
    - Pull container images from registries and unpack them.
    - Layer and prepare the image as a container-ready filesystem.

2. Anatomy of a Container

- A container is composed of:
  a. Linux Namespace
  b. Linux Control Group (CGROUPS)



a. Linux Namespace

- A **namespace** is a mechanism provided by the Linux kernel to create isolated environments.
- Processes running in a namespace perceive a restricted or virtualized view of system resources. This allows an application to think it has access to the entire file system and hardware, even if it's restricted.
  - For example, an app can run as a superuser but actually only access a limited file system as a specific user.

Types of Namespaces

Namespace Type	Description	Use in Containers
PID (Process ID)	Isolates process IDs. Each container has its own process ID space, so the processes inside a container don't see or interfere with processes in other containers or the host.	Ensures that processes inside a container only see other processes in the same container.
NET (Network)	Isolates network interfaces, IP addresses, routing tables, ports, etc.	Allows containers to have their own network stack, virtual interfaces, and IP addresses. Containers can be networked together or isolated.
MNT (Mount)	Isolates the filesystem mount points.	Provides each container with its own view of the filesystem, typically mapped to a specific directory on the host.
UTS (Unix Timesharing System)	Isolates hostname and domain name.	Enables containers to have their own hostname and domain name.
IPC (Inter-Process Communication)	Isolates IPC resources like shared memory and message queues.	Prevents interference in IPC mechanisms between containers or between containers and the host.
USER (User)	Isolates user and group IDs.	Allows mapping container users to host users (e.g., root in the container may map to a non-root user on the host).
CGROUP (Control Group)	Not technically a namespace, but often used alongside namespaces to control resource allocation (CPU, memory, etc.).	Ensures containers use only their allocated resources.
TIME	Ability to change time.	NA

Note: Docker uses all namespaces **except** the **TIME** namespace (time cannot be changed in Docker containers).

How Namespaces Work in Containers

- When a container is started (e.g., using Docker), the container runtime sets up namespaces for the container:

- Each container gets its own **isolated view** of the resources specified by the namespace types.
- For instance, a container's **PID namespace** means the container can have its own set of process IDs starting from 1, while being unaware of processes on the host or in other containers.
- Similarly, a **NET namespace** ensures that each container has its own network stack, which can be bridged, NATed, or even directly exposed to the host network.

b. Linux Control Groups (CGROUPS)

- Control groups are another Linux kernel feature that restricts how much hardware each process can use.
- Docker uses cgroups to:
  - Monitor and restrict **CPU usage**.
  - Monitor and restrict **network and disk bandwidth**.
  - Monitor and restrict **memory consumption**.
- Benefits:
  - Prevents resource-hogging containers from affecting others.
  - Avoids significant memory partitioning as seen with virtual machines.
- **Limitation**: Control groups cannot assign disk quotas to containers. However, container-native storage solutions can address this.

Isolation Mechanism

- **Namespaces**: Provide isolation by exposing different system views to each container.
- **Control Groups**: Provide resource isolation by restricting hardware usage.

Caveats

- **Linux-Only Features**
  - Control groups and namespaces are native to Linux, meaning Docker runs natively on Linux and some newer versions of Windows.

- **Kernel Dependency**
    - Containers can run on any system, but container images are tied to the kernel they were created from.
    - Containers created from Linux-configured images can only run on Linux, and those from Windows-configured images can only run on Windows.

- To run Docker on Windows, you typically need to install the Windows Subsystem for Linux (WSL). WSL allows you to run a Linux environment directly on Windows without the need for a virtual machine. Docker relies on Linux kernel features such as control groups and namespaces, so having WSL installed helps Docker operate smoothly on Windows.