A **container runtime** is the software component responsible for running containers on a host system. It provides the low-level functionality needed to:

- **Start and stop containers** – launch containerized applications from images and manage their lifecycle.
- **Set up namespaces and cgroups** – isolate process IDs, networking, storage, and resource usage (CPU, memory, I/O) so containers don't interfere with each other.
- **Handle container images** – pull images from registries, unpack them, and prepare their filesystems.
- **Provide networking and storage hooks** – integrate containers with networks and volumes.

Think of it as the "engine" that actually executes containers.

Types of container runtimes

There are two broad categories:

1. **Low-level runtimes** – directly interact with the OS kernel to create containers.
    - Examples: **runc**, **crun**, **Kata Containers**
2. **High-level/container management runtimes** – sit above low-level runtimes and provide more features like image management, orchestration hooks, and tooling.
    - Examples: **containerd**, **CRI-O**, **Docker Engine**

For example, Docker uses `containerd` under the hood, which in turn uses `runc` to start containers.

Docker uses a **stack of runtimes**:

1. **Docker Engine** – the high-level runtime and CLI/API layer you interact with (`docker run`, `docker build`, etc.).

2. **containerd** – the mid-level runtime responsible for managing container lifecycle (create, start, stop, delete) and pulling/storing images.

3. **runc** – the low-level runtime that actually interfaces with the Linux kernel to set up namespaces, cgroups, and start the containerized process.

Flow:

You (docker run)

↓

Docker Engine

↓

containerd

↓

runc

↓

Linux kernel (namespaces, cgroups)

In short: **Docker uses `containerd` and `runc` under the hood** to run containers

---

A **shim is a middleman process** that manages a running container's lifecycle and I/O after it has been launched, so the main runtime (`runc`) doesn't need to stay attached.

ps aux | grep containerd-shim

Role of the shim

- **Keeps the container alive** even if `containerd` restarts (since the shim holds the process).
- **Collects exit codes** from the container process.
- **Manages I/O** (standard input/output) so logs don't get lost.
- **Allows detached execution** (container keeps running in the background).

When a container starts:

- `containerd` uses a low-level runtime (like `runc`) to create the container.
- Once the container is running, you don't want `runc` to hang around forever—it's only needed during startup.
- But you still need a way to:
    - Keep the container's **stdin/stdout/stderr** connected.
    - Handle **exit status** when the container stops.
    - Support features like **`docker exec`** (running commands inside a running container).

That's where the **shim** comes in.