Operating Systems

Beyond Docker

Noah Lehmann

Instiute of Information Systems, Hof University

March 22, 2025

Table of Contents

Introduction

About me

- Noah Lehmann
- Research Assistant @ iisys
  `https://www.iisys.de/profile/noah-lehmann/`
- Starting orientation for PHD (Probably something with containers)
- Obsessed with automation

`https://github.com/noahlehmann/talk-beyond-docker`
`https://github.com/noahlehmann/talk-beyond-docker`

Introduction

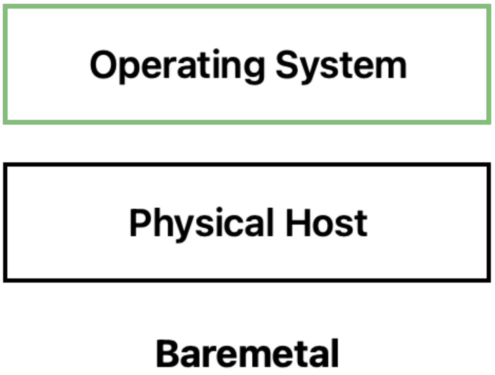
Prerequisites

You should have heard of the following:

- **Docker**
  `docker run|build|exec|ps|stop|rm|...`
- **Linux**
  What are processes, basic file system structure...
- **Virtualization**
  What are VMs, how are they used.

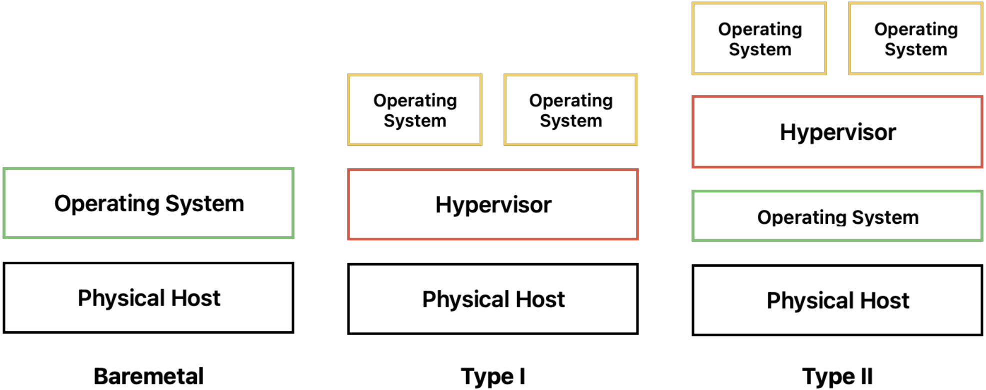
Containerization under the hood

Bare Metal



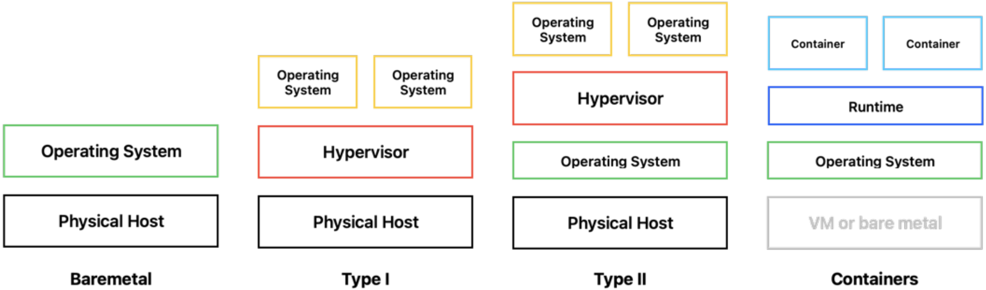
Containerization under the hood

Hypervisors



Containerization under the hood

Containers I



Containerization under the hood

Containers II

- Containers are isolated processes
- No need for virtualized hardware

You can see this quite obviously with the commands on the following slide.

Containerization under the hood

Containers III

On a Linux system hosting VMs run:

```
ps aux | grep kvm
```

On a Linux machine with docker installed run:

```
docker run -dit --name alpine alpine:latest /bin/ash
ps -fp $(docker inspect -f '{{.State.Pid}}' alpine)
```

Containerization under the hood

Containers IV

The container likely only runs one single process.
This process likely hast the ID `1`.

```
docker exec -it alpine ps aux
```

Containerization under the hood

rootfs I

Notice we started the container with `/bin/ash`.
This does not ship standard with most distributions.
So where does this binary come from?

```
ROOT_FS=$(docker inspect \
    $(docker ps -q --filter "name=alpine") \
    | jq '.[0].GraphDriver.Data.MergedDir' \
    | sed 's/"//g' \
)
```

Containerization under the hood

rootfs II

We now have the path to the containers filesystem saved in the variable `Root_FS`.
We can now search if the `ash` binary is present.

```
sudo ls $ROOT_FS/bin | grep ash
```

Containerization under the hood

Namespaces I

- Namespaces are borders in which a process can operate.
- They isolate processes and therefore containers from another.
- We can check the namespaces of a process under the `/proc` directory.

Containerization under the hood

Namespaces II

The following command shows us the namespaces of our Alpine container.

```
ls -l /proc/$(docker inspect \
    --format '{{.State.Pid}}' alpine)/ns
```

Containerization under the hood

Namespaces III

- Processes can have the same ID as long as they are in different namespaces.
- This is why the process ID of the container process is `1`.
- The host system most likely runs `/sbin/init` with process ID `1`.

Containerization under the hood

Control Groups I

- Cgroups are a way to limit resources of a process.
- They are used to limit CPU, memory, and other resources.
- We can check the cgroups of a process under the `/proc` directory.

Containerization under the hood

Control Groups II

Check the cgroups of our Alpine container.

```
sudo cat /proc/$(docker inspect \
    --format '{{.State.Pid}}' alpine)/cgroup
```

```
0::/system.slice/docker-xxx.scope
```

Containerization under the hood

Control Groups III

- `0::`
  Root `cgroup`
- `system.slice`
  Broadly speaking this handles processes started by `systemd`
- `docker-589c53b502....scope`
  The container specific `cgroup`

The hierarchy allows the host system to control resources of service.
Systemd can control the resources of all services started by it.
Docker can control the resources of all containers started by it.

Levels of Control

Runtimes I

- OS Level Container Management is complex.
- Docker is a collection of products making this happen.
- Control of Cgroups and Namespaces is done by Container Runtimes,

Levels of Control

Runtimes II

Low Level Runtimes

- Handle Kernel feature like `cgroups` and `namespaces`.
- Typically don't handle container and image management.
- Not very user friendly.

Docker uses `runc` for this.

Levels of Control

Runtimes III

High Level Runtimes

- Call low-level runtimes to use Kernel features.
- Manage container and image lifecycles.
- Usually command line based and lack advanced features.

Docker uses `containerd` for this.

Interfaces

To use these runtimes more user friendly, implementations like Dockers add custom interfaces for this.

- Docker CLI for advanced CLI usage (`docker run`).
- Docker Dashboard for a graphical user interface.
- Add features for ease of use like e.g.:
  - Compose
  - Registry management
  - Auto pulling of images

Not necessary in production environment.

Note

Windows and MacOS

- Non Linux OSs usually don't offer Kernel features like `cgroups` and `namespaces`.
- They use optimized virtualization of the Linux Kernel instead.
- Virtualization allows them to use Kernel features.

Example on MacOS

```
ps aux | grep docker
```

```
/Applications/Docker.app/Contents/MacOS/
com.docker.virtualization --kernel /Applications/
Docker.app/Contents/Resources/linuxkit/kernel
```

Open Container Initiative

Standardization for Containers

The Open Container Initiative (OCI) is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

OCI - `https://opencontainers.org/about/overview/`

Beyond Docker

What exactly is Docker?

Docker Engine
Powerful container runtime

The Docker Engine powers your containerized applications with high performance and reliability. It provides the core technology for building and running containers, ensuring efficient and scalable operations.

Docker CLI
Flexible command-line interface

The Docker CLI offers a robust command-line tool for precise control over your containers. Execute complex commands, automate tasks, and integrate Docker seamlessly into your workflows.

Docker Compose
Streamlined multi-container management

Docker Compose simplifies the process of managing multi-container applications. Define and run complex setups with a single configuration file, making it easier to deploy and scale your applications.

Docker Build
Simplified container building

Docker Build is a powerful tool within Docker Desktop that simplifies the process of creating container images. It enables you to package and build your code to ship it anywhere while integrating seamlessly into your development pipeline.

Docker Kubernetes
Built-in container orchestration

Docker Kubernetes provides built-in Kubernetes support within Docker Desktop, allowing you to orchestrate and manage containers efficiently. Supporting both multi-node clusters and developer-selected versions, Docker Kubernetes simplifies deploying, scaling, testing, and managing containerized applications locally without needing an external cluster.

Volume Management
Effective data management

Docker Volumes provides a robust solution for managing and sharing container data. This feature allows you to easily and securely manage volumes for backup, sharing, or migration purposes, enhancing data management and portability.

Synchronized File Shares
Seamless data synchronization

Synchronized File Shares enable real-time sharing and synchronization of files between your host and containers. This feature ensures that file updates are instantly reflect on the host and container, improving collaboration and consistency.

Docker Debug
Advanced troubleshooting tools

Docker Debug provides comprehensive tools for diagnosing and resolving issues within your containers and images. This CLI command lets you create and work with slim containers that would otherwise be difficult to debug.

Hardened Docker Desktop
Enhanced container isolation

Hardened Docker Desktop includes advanced security features to safeguard your development environment. With enhanced container isolation, registry and image access management, and compliance with industry standard, you can confidently build and deploy secure applications.

VDI Support
Virtual desktop integration

VDI Support allows Docker to seamlessly integrate with virtual desktop infrastructure (VDI) environments. This feature ensures that Docker runs smoothly on virtualized desktops, providing a consistent experience regardless of where you access your containers.

Docker Private Extensions Marketplace
Custom extensions for your needs

The Docker Private Extensions Marketplace offers a curated selection of extensions tailored to your specific requirements. Customize and enhance your Docker environment with specialized tools and integrations available exclusively through the marketplace.

Beyond Docker

Alternatives

- **Podman**
  A daemonless container engine with full OCI compatibility.
  `https://podman.io/`

- **LXC**
  A stateful container solution with focus on system containers (not application containers).
  `https://linuxcontainers.org/`

- **Kata Containers**
  A container runtime running MicroVMs for stronger isolation.
  `https://katacontainers.io/`

Beyond Docker

Escalating container usage

- Simple containers
- Docker compose
- Docker swarm mode
- Container orchestration

Applications

Usage examples - Demo

- Test software
- No local installs
- Local environments
- Devcontainers
- CI/CD
- AI/ML and data science
- Kubernetes/ OpenShift
- Serverless

And lots more.

Security Concerns

Isolation and sharing

- **Kernel sharing**
  Escaping a isolated process could allow manipulation of system.
- **Privilege Escalation**
  Some container features require `root` privileges. Escaping the process isolation allows access to sensible OS features.
- **Persistent data exposure**
  Mounting and mapping file systems can leak data to wrong container if misconfigured.

Bleeding Edge Research

What's coming next?

- **Checkpoint and restore**
  Dumping containers states and restarting or migrating them stateful.
  `https://criu.org/Main_Page`

- **Machine learning capabilities**
  Leveraging container efficiency for ML acceleration (GPU usage efficiencies).
  `https://www.nvidia.com/en-us/technologies/multi-instance-gpu/`

Q&A

Any Questions?