(Linux) Containers introduction

WASA: Web and Software Architecture

Enrico Bassetti

Containers

Virtualization technologies

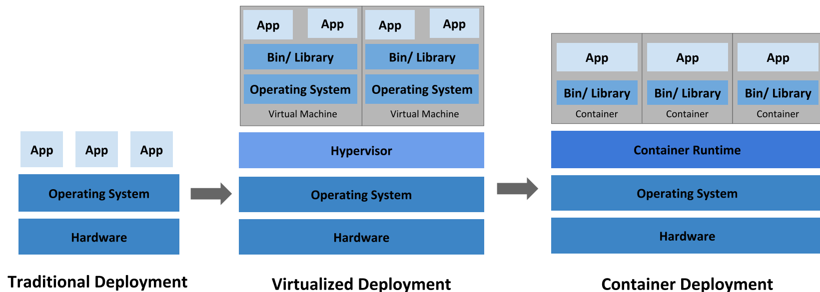


Image courtesy of The Kubernetes Authors / The Linux Foundation - CC BY 4.0

Standard deployment

Standard deployment:

- Processes aware of other processes
  - they can interact each other (POSIX signals, shared mem)
- They share kernel, libraries, filesystem, hardware
- Little to no constraints, limited “supervision” (restart-on-crash, etc.)

Virtual machines

Virtualization:

- World divided in two roles:
  - *host* is the O.S. running on real hardware
  - *guest* is the O.S. running in the virtual machine
- Guest machine instructions are executed/translated by the hypervisor
- Direct hardware access not needed
  - guests have “virtual hardware” devices
  - but it might be granted if needed
- VMs are isolated: crashes, security incidents, etc. in guests are not affecting other guests

Containers

Containers:

- World is still divided in:
  - *host* is the O.S. running on real hardware
  - *containerized process(es)* is the process(es) running in a *container*
- A *container* is a set of restrictions placed on **syscall**
  - A container may have one or more processes
- Containerized processes runs on the same kernel of the *host*
- Restricted environment
  - Filesystem is not shared by default
  - processes cannot interact by default
  - other restrictions are in place
- Less isolation than the VM:
  - a problem in a containers is not affecting other containers
  - **a problem in a syscall will affect the whole system**

Linux containers

Containers

In general, many operating systems supports containers in various shapes:

- `chroot` in most UNIXes (since 1982!)
- FreeBSD jails (2000)
- Solaris Containers/Zones (2000)
- LXC on Linux (2005)
- Docker/Podman on GNU/Linux (2013), Windows and recently FreeBSD
  - macOS is not in this list!
- Other technologies: LXD, vServer, OpenVZ, Virtuozzo, etc.

Linux containers

In Linux, containers are built over **cgroups** (v2 currently). cgroups where designed by Google in 2006, and published in 2008.

They provide:

- Resource limitings
  - limits on CPUs, RAM, filesystem, I/O
- Prioritization
  - some processes may have priority
- Accounting
  - resource usage measurements (e.g., billing, issue detection)
- Lifecycle control
  - restart on crash, checkpoint, schedule, etc.

Linux containers

cgroups are responsible for resource management.
It makes sense, to grant some daemon exclusive
access to this functionality to avoid lots of
problems.

systemd-nspawn

If your sound card can do hardware mixing, and
your Linux device driver supports this feature,
then multiple programs can access your sound
card at the same time and you hear them all
simultaneously!
PulseAudio daemon does software mixing.
Without hardware or software mixing, only one
program can access the sound card; as a result,
you cannot have Audacious AND VLC put out sound
at the same time!
JACK daemon does the same but targets professional
audio editors.

DRM manages the GPU
KMS manages the display controller (CRTC)
The display controller usualy sits on the
die of the GPU, and communicates with
the monitor, e.g. changes the resolution
or the refresh rate.

David Herman split DRM and KMS,
then added "render nodes" to the DRM.

X.Org doen't need to be root any longer,
but its still wise (technically necessary?)
to grant it exclusive access to the KMS.

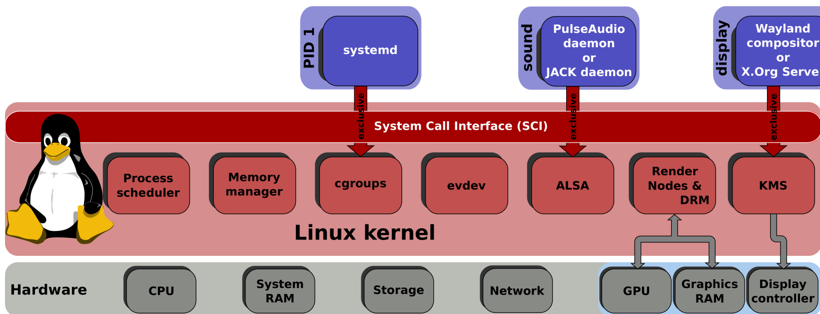


Image courtesy of Wikipedia author ScotXW - CC BY 4.0

Namespace isolation

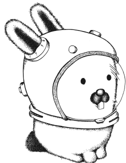
Each container runs in *namespaces*. A *namespace* may contain one or more containers.

Processes/containers that runs in the same namespace can share their resources with others.

There are different namespaces:

- Process ID (PID)
- Network
- UTS / hostname
- Mount
- IPC
- User
- Cgroup

Linux namespaces came from Plan 9, which is an experimental operating system from Bell Labs.



Plan 9 was invented by the same group that developed UNIX and many programming languages: B, C, and GO!

OCI containers

Docker

Docker (2013) is a set of tool for managing Linux containers, and, lately, Windows containers.

Docker containers have been standardized under OCI (Open Container Initiative), and different implementations exists (e.g., ***podman***).

OCI/Docker glossary

- **Container**: a Linux containers (or Windows containers on Windows)
- **(Container) Image**: an archive with files and configurations for creating a new container
- **Registry**: a *forge* for sharing/storing container images (e.g., DockerHub)
- **Dockerfile/Containerfile**: a file that specifies how to build a container image
- **(Docker) compose**: a YAML that specifies how to run a container
- **Volume**: an external storage space that is mounted inside the container
    - containers may share a volume
    - volume are persistent

OCI/Docker container volumes

Docker Container are started from an image. The image contains the new "root directory" for the container. **This new root will be ephemeral** (data will be deleted at exit).

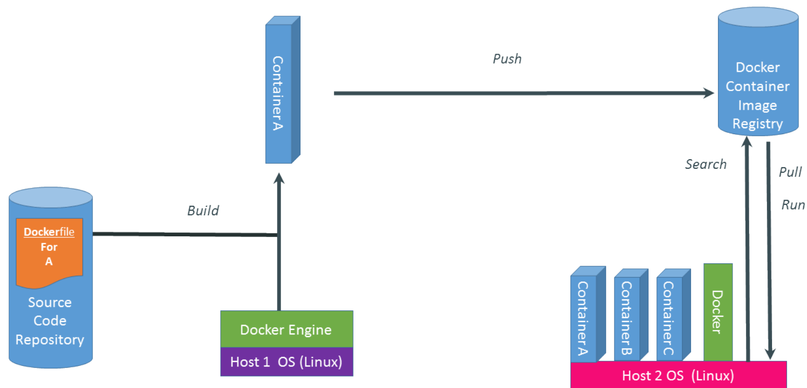
Processes cannot access files outside that root. However, docker can be instructed to mount an external volume in a specific path inside the container.

Container image are built using *`Dockerfile/Containerfile`*. The file contains instructions; each instruction creates a *layer*.

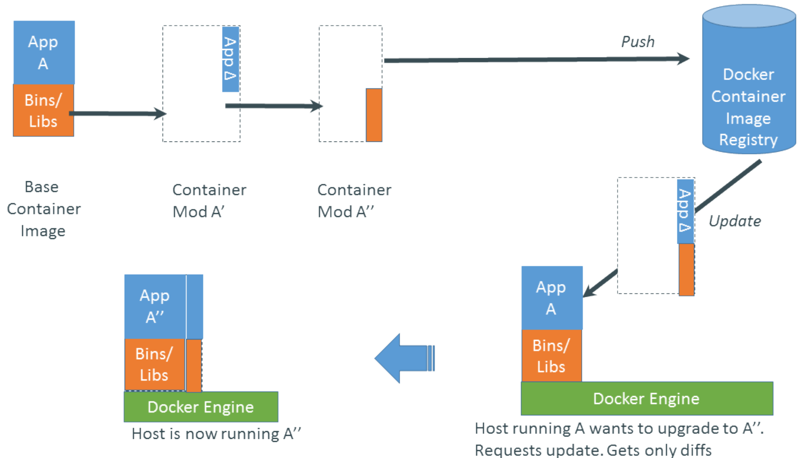
Layers contains the difference since previous layers. They can be cached and reused.

Usually, container images starts from a *base image*, which contains some tools. E.g., *`debian:stable`* contains a basic Debian stable system; *`python:3`* contains a basic Python 3 interpreter (with all dependencies!).

OCI/Docker: flow



OCI/Docker: share layers



Containers in the cloud

Once a container is built, it can run on any Linux platform, regardless of the underlying system (provided that it supports cgroups and other container-related technologies).

Run your first container

To run a container with Docker, you can use a simple command:

```
docker run -it --rm debian:bullseye
```

This command creates a new interactive (flag `-it`) container using the image `debian:bullseye`, and remove the container at exit (flag `--rm`).