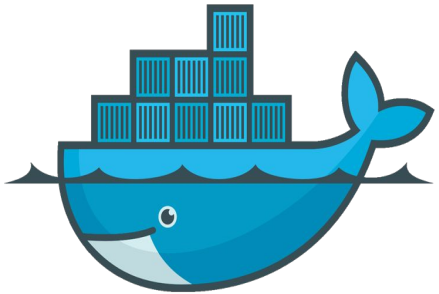


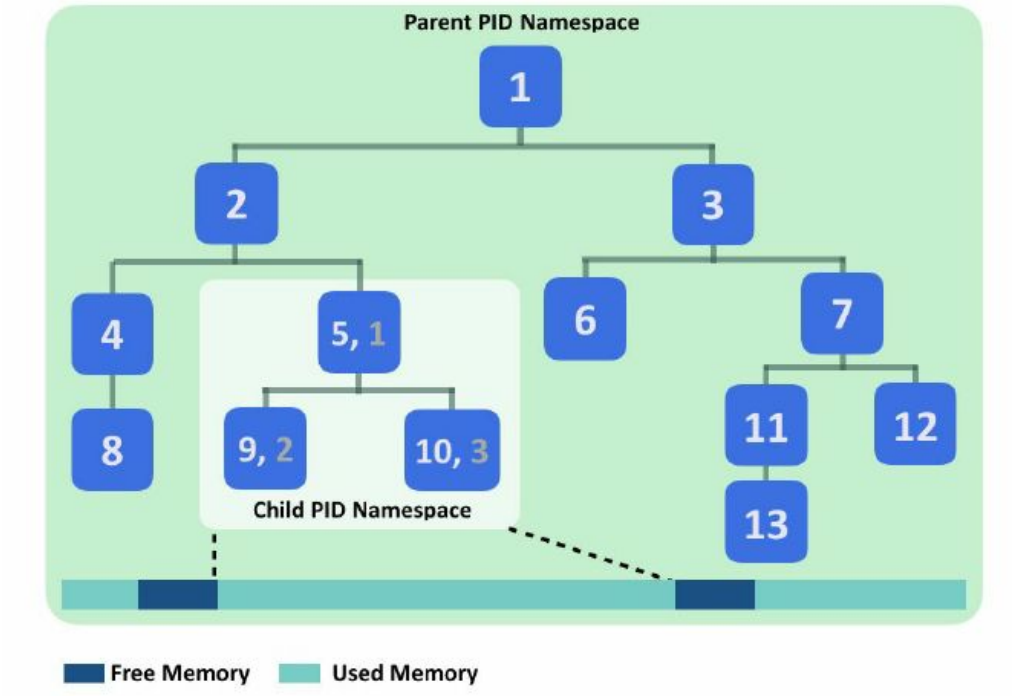
Agenda

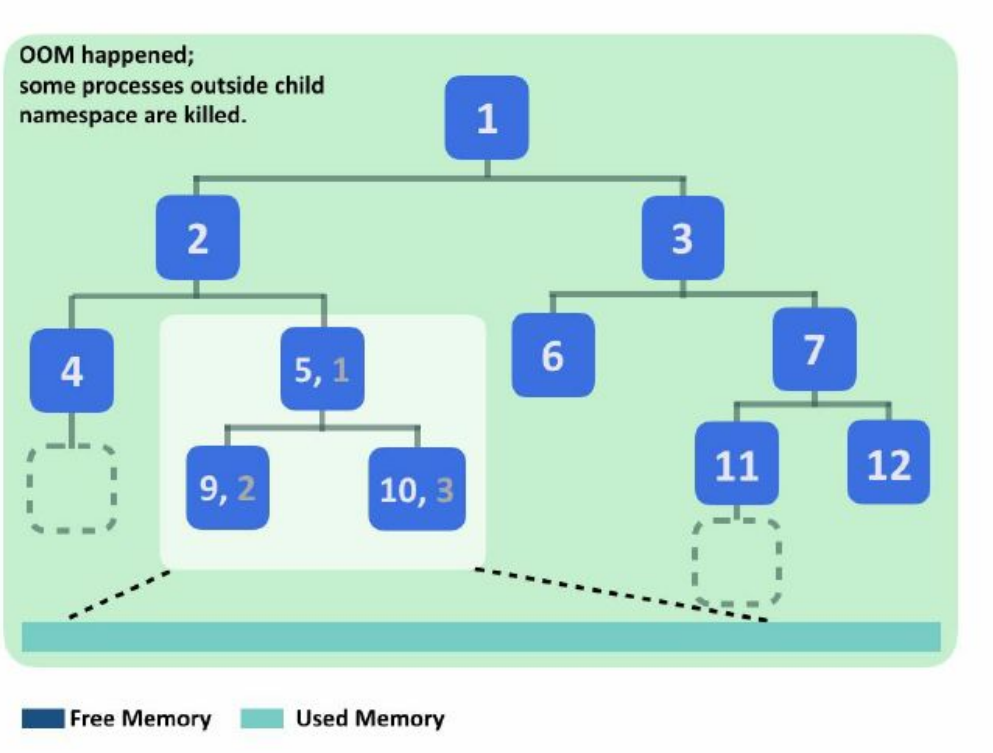
1. NameSpaces
2. Control Groups
3. Containers
4. Virtual Machines vs Containers
5. What is a Dockerfile?
6. What is an Image?
7. Docker Registry
8. EXTRA: AppArmor (Security Profiles)

Let's see some examples of a Name Space.

NameSpaces

Namespaces provide logical partitions of certain kinds of system resources, such as mounting point (mnt), process ID (PID), network (net), and so on.


Parent PID Namespace
1
2
3
4
5, 1
6
7
8
9, 2
10, 3
11
12
13
Child PID Namespace
Free Memory
Used Memory


OOM happened;
some processes outside child
namespace are killed.
1
2
3
4
5, 1
6
7
9, 2
10, 3
11
12
Free Memory
Used Memory

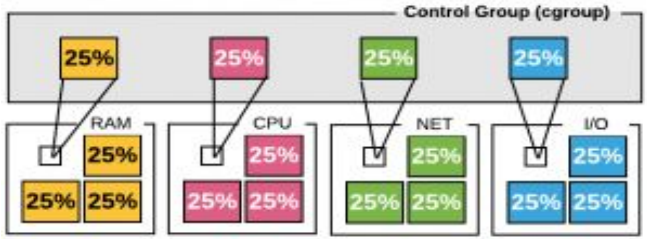
Let's see an example limiting the resources of a NameSpace.

Control Groups (a.k.a CGroups)

Control Groups can set constraint on different kinds of system resources, such as, memory, CPU, CPU Sets, Disk, I/O Blocks.

Control Groups (a.k.a CGroups)

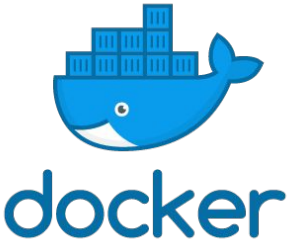
Cgroups, can be used to slice an entire operating system into buckets, similarly to how virtual machines slice up their host system into buckets, but without having to go so far as replicating an entire set of hardware.

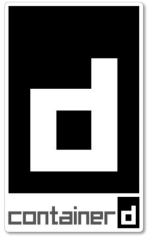


What is a container?

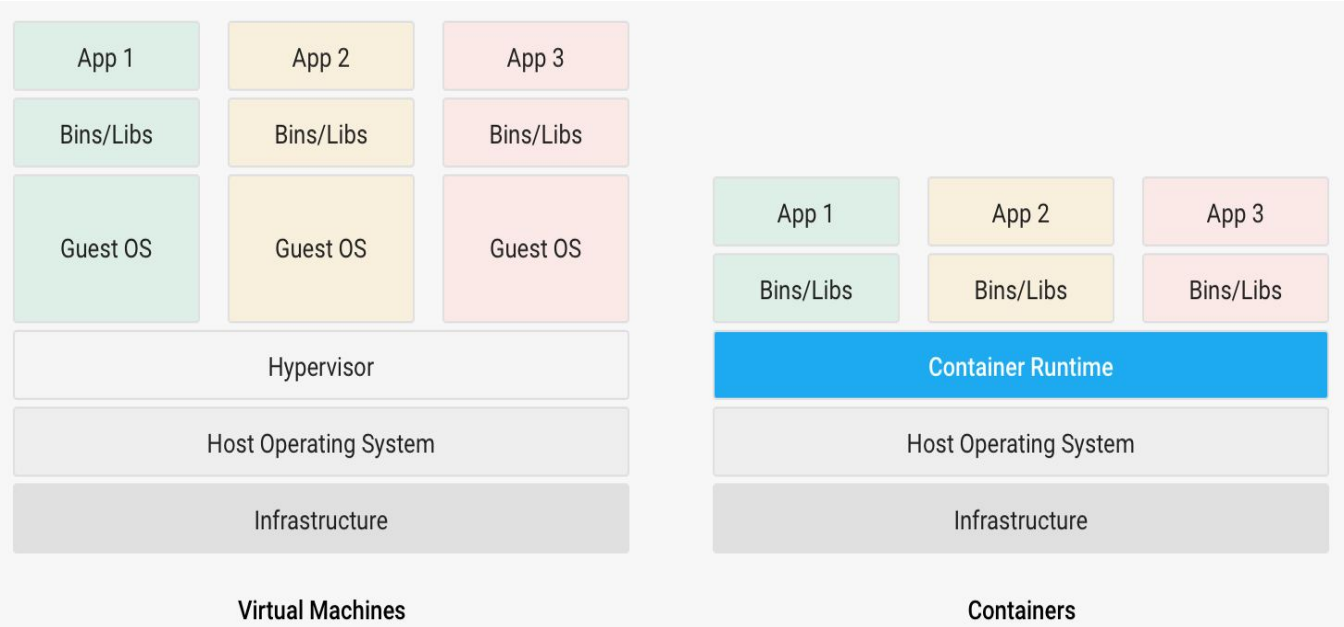
Containers are like normal operating system processes, isolated through Name Spaces and limited by Control Groups.

Since containers are nothing other than OS processes, this is the reason why lifting a container takes seconds and lifting a virtual machine takes minutes.





VMs vs Containers



What is a Dockerfile?

A Dockerfile is a text file which contains a series of commands or instructions. These instructions are executed in the order in which they are written. Execution of these instructions takes place on a base image.

Let's write a DockerFile

We can use Docker’s reserved, minimal image, **scratch**, as a starting point for building containers

```
FROM scratch
ADD hello /
CMD ["/hello"]
```

Docker containers are Linux Containers

- Kernel Feature
- Groups Processes
- Control Resource Allocation
  - CPU, CPU Sets
  - Memory
  - Disk
  - Block I/O

- The real magic behind containers
- It creates barriers between processes
- Different Namespaces
  - PID Namespace
  - Net Namespace
  - IPC Namespace
  - MNT Namespace
- Linux Kernel Namespace introduced between kernel 2.6.15 – 2.6.26

- Images
  - Not a File System
  - Not a VHD
  - Basically a tar file
  - Has a Hierarchy
    - Arbitrary Depth
  - Fits into Docker Registry

lxc-start = docker run

… & What about the FS?

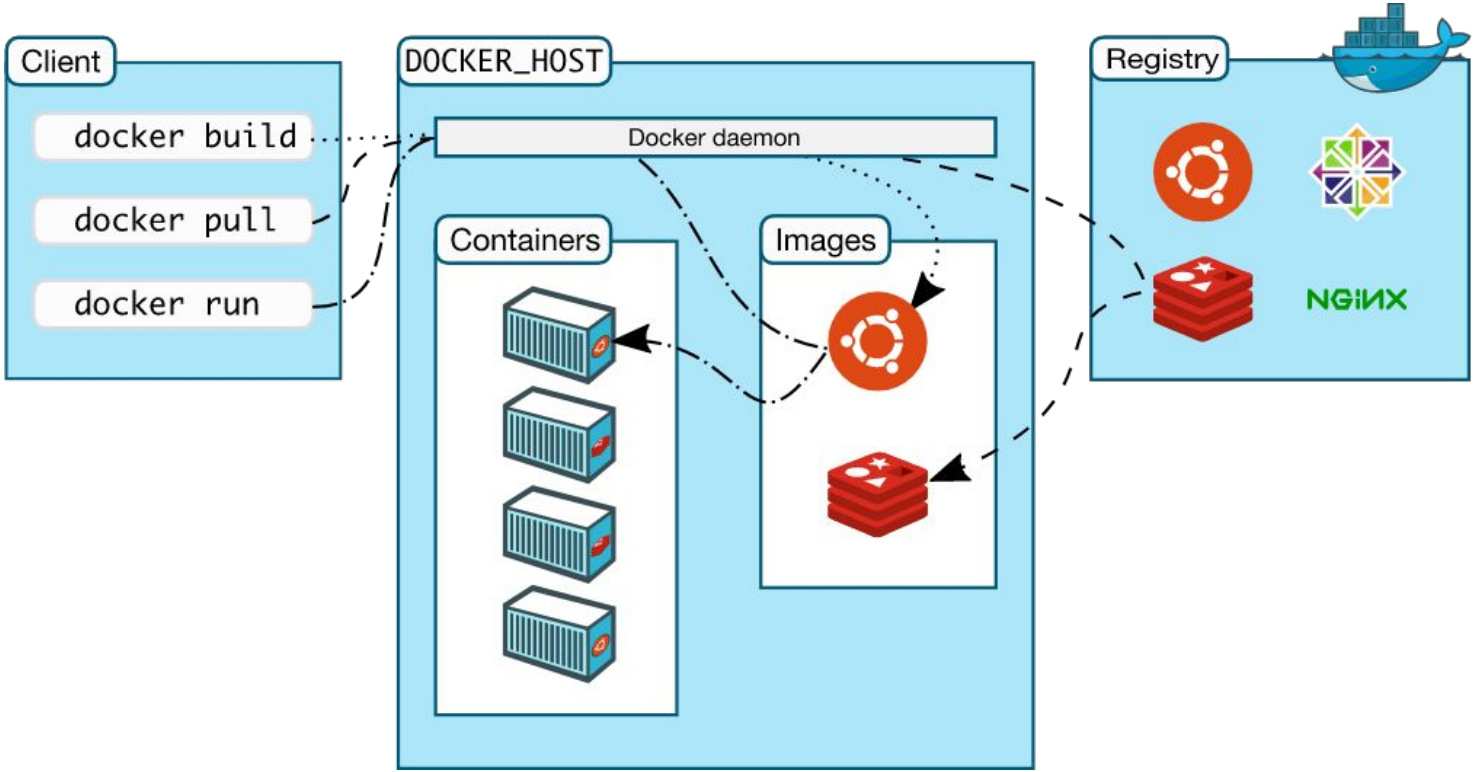
The data doesn’t persist when the container no longer exists, and it can be difficult to get the data out of the container if another process needs it…

Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: **volumes**, and **bind mounts**

Docker Registry

A **Docker registry** is a storage and distribution system for named Docker images. The same image might have multiple different versions, identified by their tags.

Docker Registry



Docker Registry

A Docker registry is organized into **Docker repositories** , where a repository holds all the versions of a specific image.

By default, the Docker engine interacts with **DockerHub** , Docker's public registry instance.

EXTRA

AppArmor (Application Armor)

AppArmor is a Linux Security Module (LSM). It protects the operating system by applying profiles to individual applications or containers.

In contrast to managing *capabilities* with CAP_DROP and syscalls with *seccomp*, AppArmor allows for much finer-grained control.

For example, AppArmor can restrict file operations on specified paths.

AppArmor (Application Armor)

Docker automatically generates and loads a default profile for containers named `docker-default`.

Note: This profile is used on containers, *not* on the Docker Daemon.

Let's see an App Armor Profile and launch a Nginx container with the profile loaded.