

# Containers 101

---

**Author:** JJ Quinlivan

**Date:** November 7, 2019

# Containers 101 Agenda

- What is a Container?
  - Containers vs Virtual Machines
- Where are Container Technologies used today?
- Container Technologies
- Docker/OCI vs Linux Containers (LXC)
- Docker/OCI Overview
- Orchestration Overview

# Where are Container Technologies used Today?

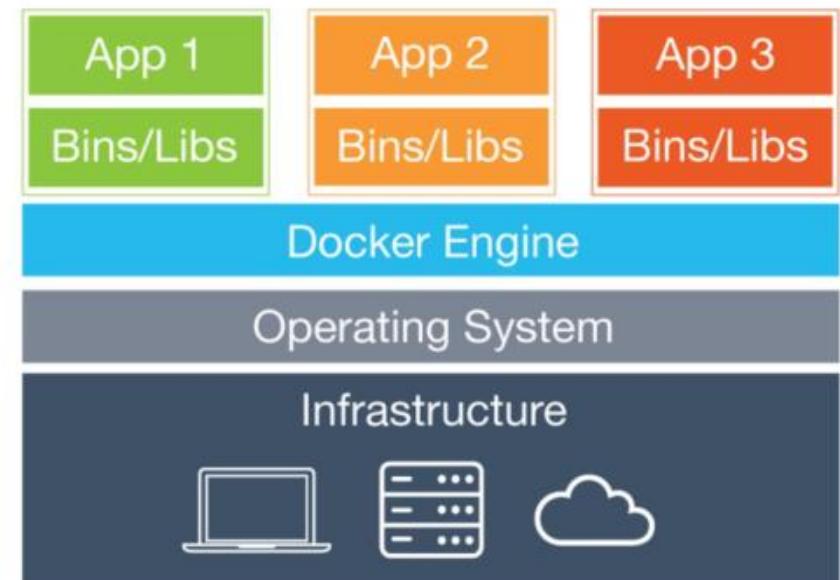
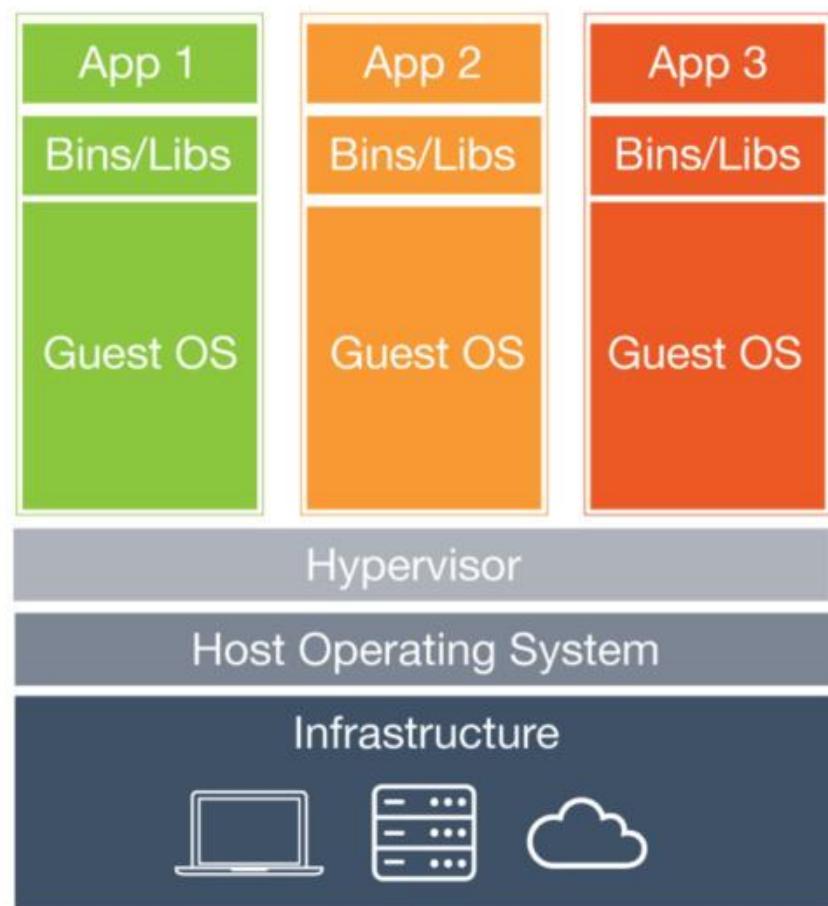
- Docker/OCI Containers
- Linux Containers (LXC)
- Snaps/Flatpaks
- FireJail
- The dreaded Google Chrome
- Kubernetes, Amazon ECS etc.

# What is a Container

- Sandbox for a process
- Uses technologies built-in to the Linux Kernel



# Containers vs Virtual Machines



# Three Little Pigs by Dan Walsh



Once upon a time there were Three Little Pigs, who had different types of homes to choose from...

# Standalone Homes

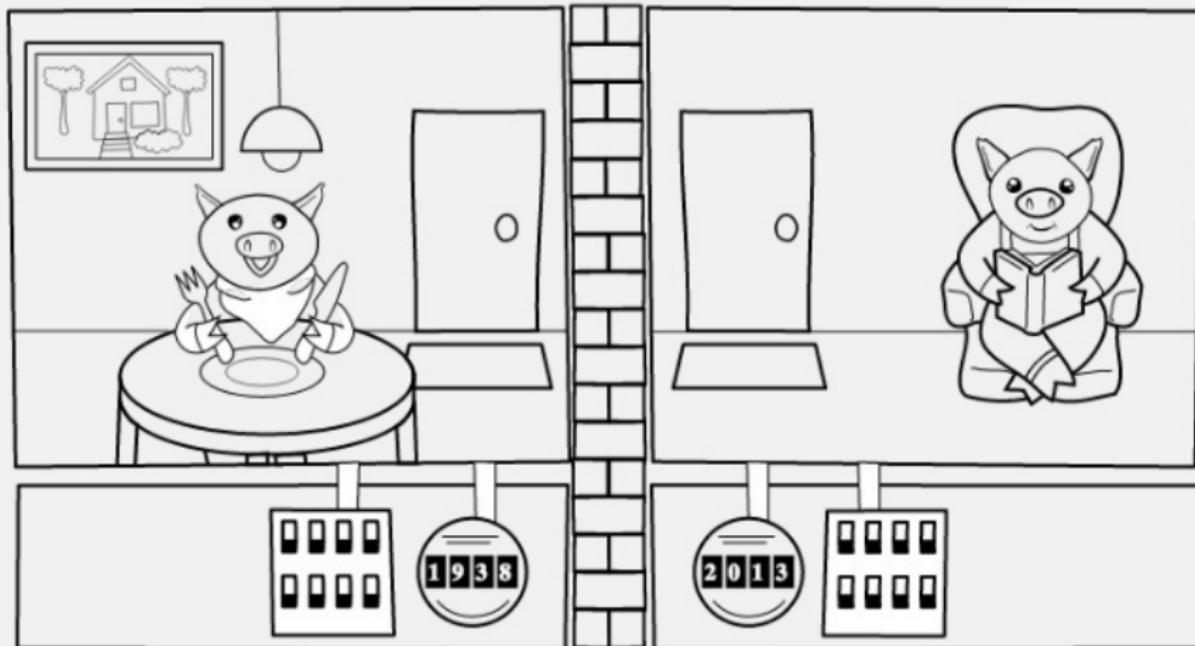
## Separate Physical Machines



If one house is broken into, the rest remain secure...a lot more maintenance though!

# Duplex Home

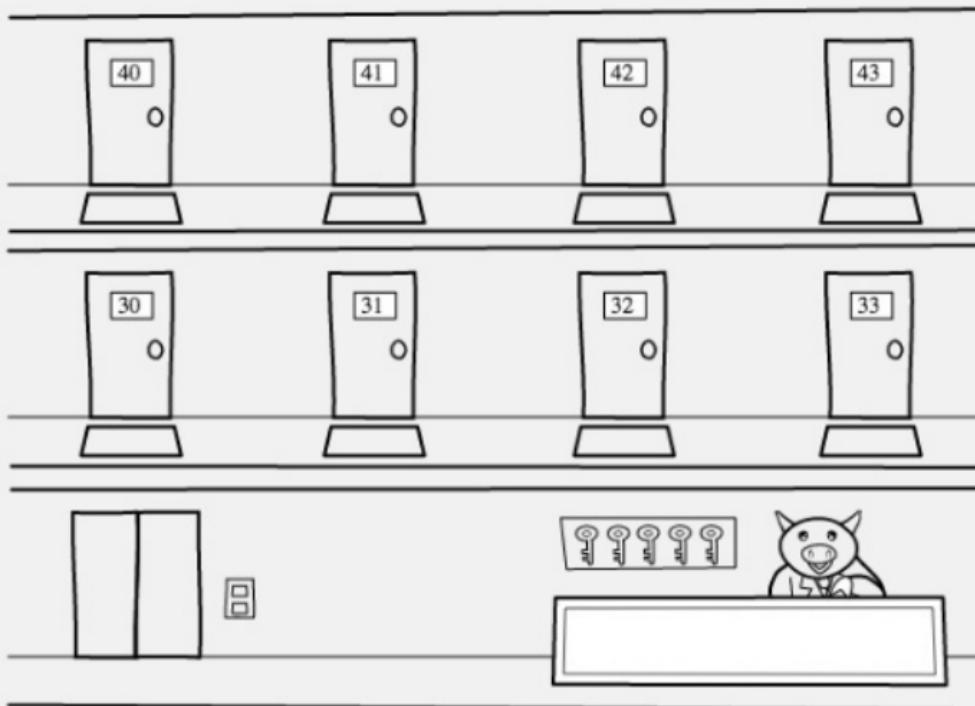
## Virtual Machines



Hypervisor, sVirt and the host kernel provide separation..but still cost of maintaining separate OSes

# Apartment Building

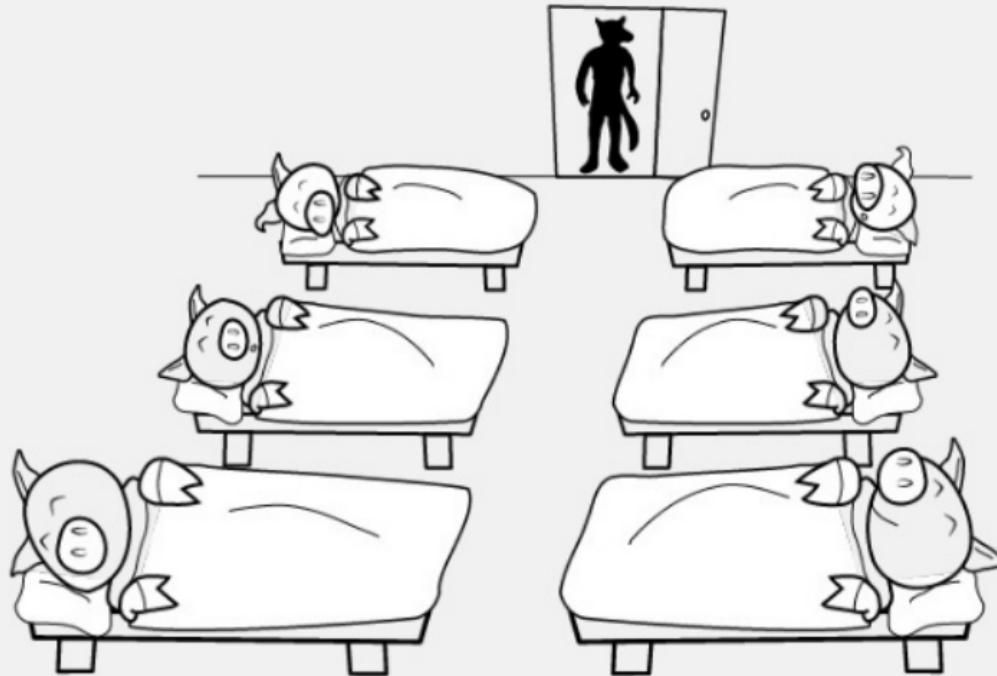
## Containers



Excellent sharing of services, lower cost of maintenance and decent separation

# Hostel

Services on the same machine



Limited isolation between services, better if you have SELinux

# Park Bench

# setenforce 0



Don't let this be you...

# Pigs in Apartment Building

The best combination



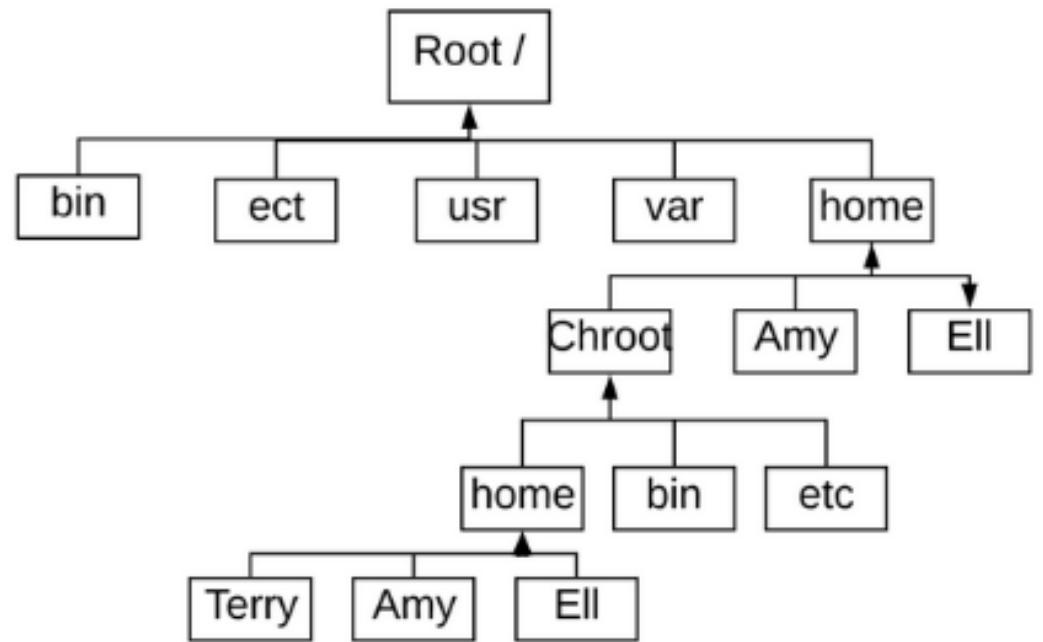
Best combination of resource sharing, ease of maintenance & security.

# What's a Container Made of

- Chroot
- Linux Namespaces
- Cgroups
- Linux Capabilities
- SECCOMP

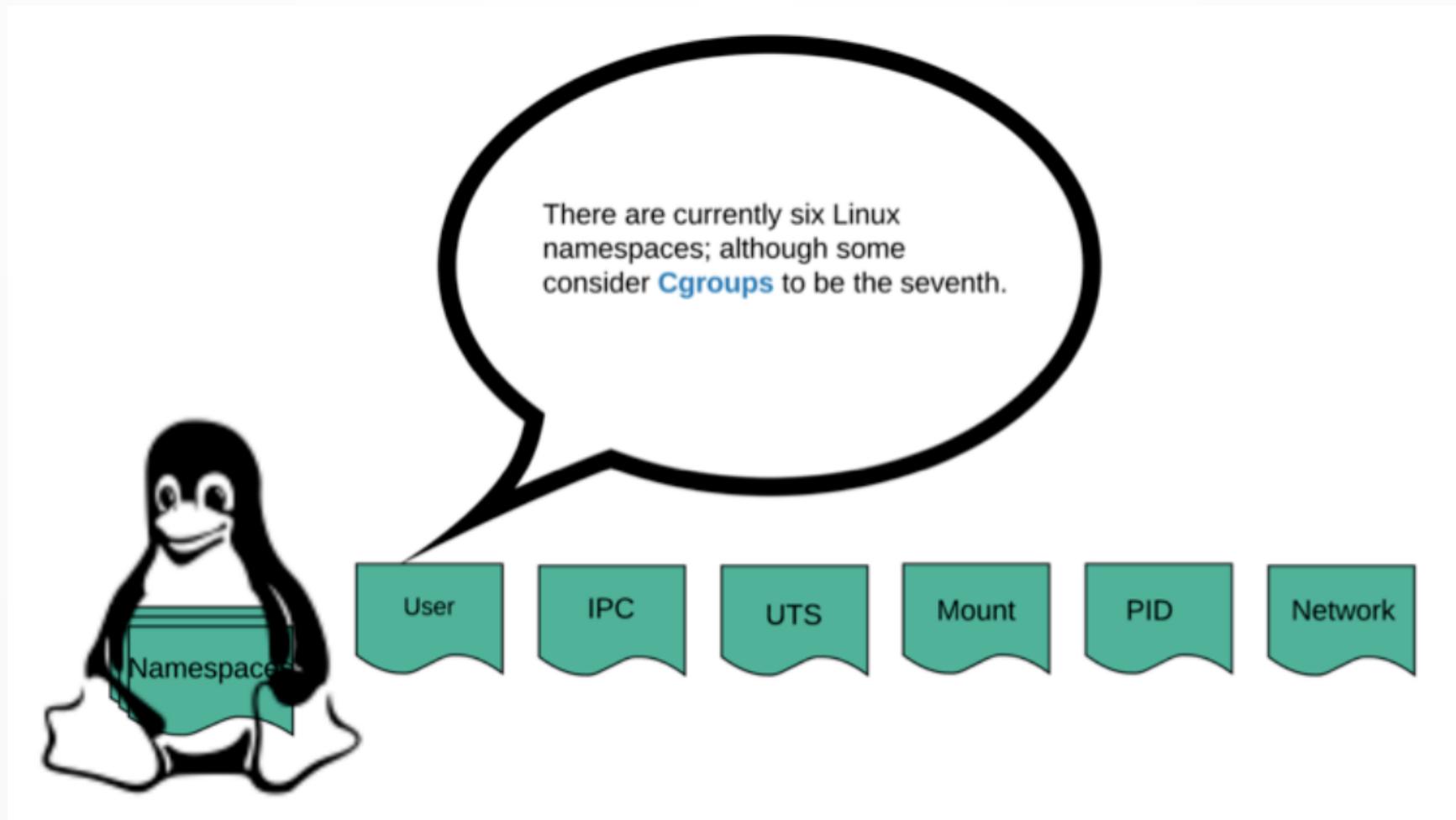
# Chroot

- Chroot = Change root
  - Changes the apparent root directory.
  - A new “root” directory becomes the root directory for both the current and running processes and all of the children processes.
  - A new “root” directory is known as a jailed directory or “Chroot jail”.
  - Chroot must be run as privileged user.



# Chroot Demo

# Linux Namespaces

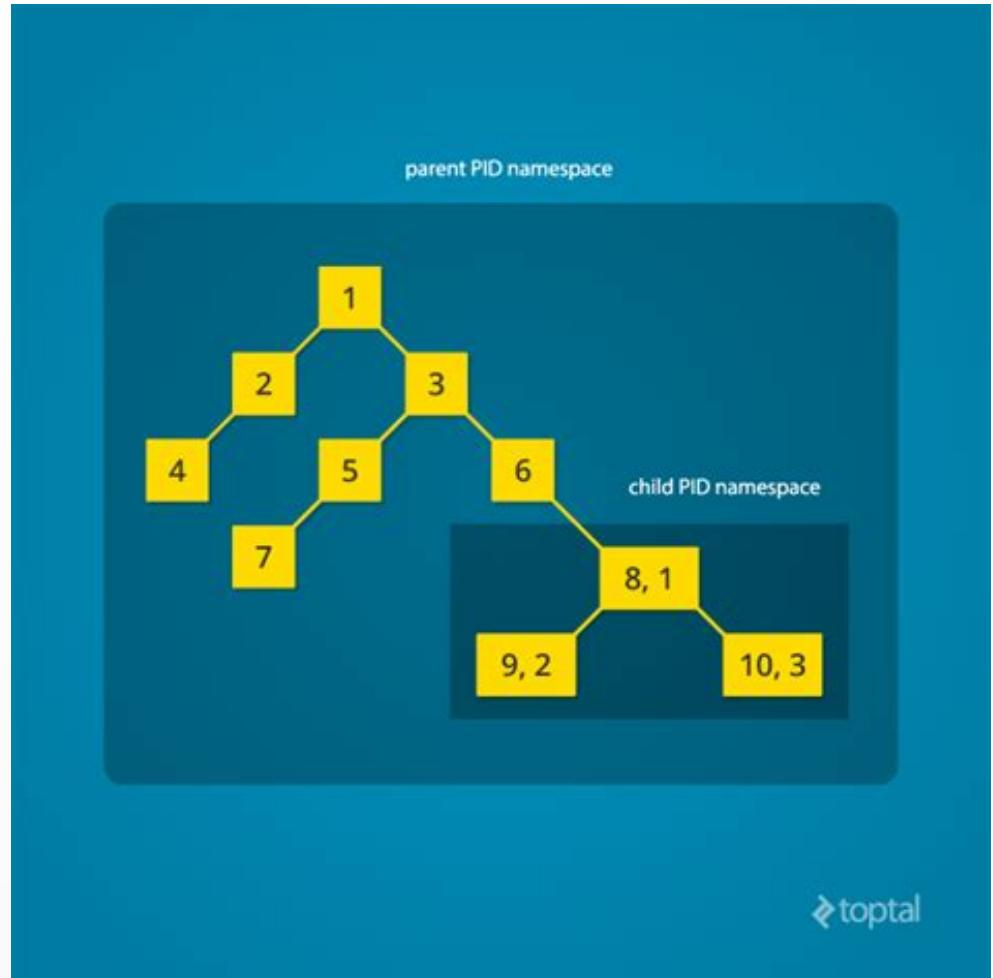


# Linux Namespace

- User
  - Maps non-root user ID on the host to root user ID in the namespace
  - Can map non-existent user ID on the host (i.e. ID above 65000)
- IPC – Interprocess communication
- UTS – different host and domain names
- Mount
  - Isolates the mount table
  - Mount / Unmount operations isolated to the namespace
- PID – Process IDs
- Network
  - Virtualize the network stack
  - Each namespace can have its own virtual or physical network interface, resources and firewall rules.

# PID Namespace

- Allows a container its own isolated instances of global resources
  - I.E. PID namespace can only see processes running in the container



# Network Namespace Demo

# Cgroups – Control Groups

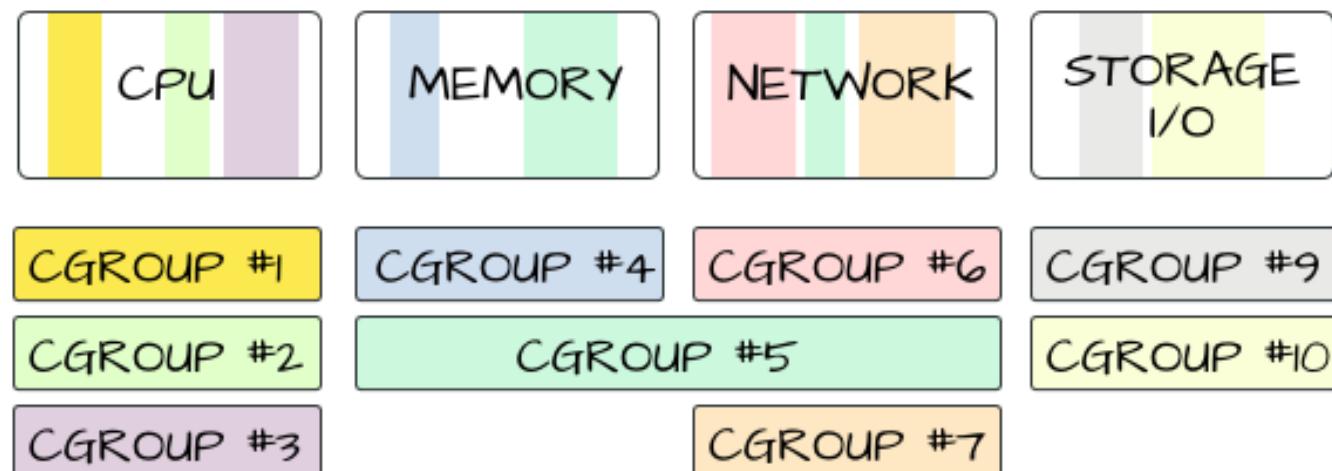
- Limits the usage of resources to the container/process
  - Number of CPU cores
  - % of CPU usage
  - Amount of memory
- Control which device nodes can be created within a namespace
  - i.e. /dev/console, /dev/tty, /dev/random used by kernel and by default not available to containers
- Access in /sys/fs/cgroup

# Cgroups – Control Groups

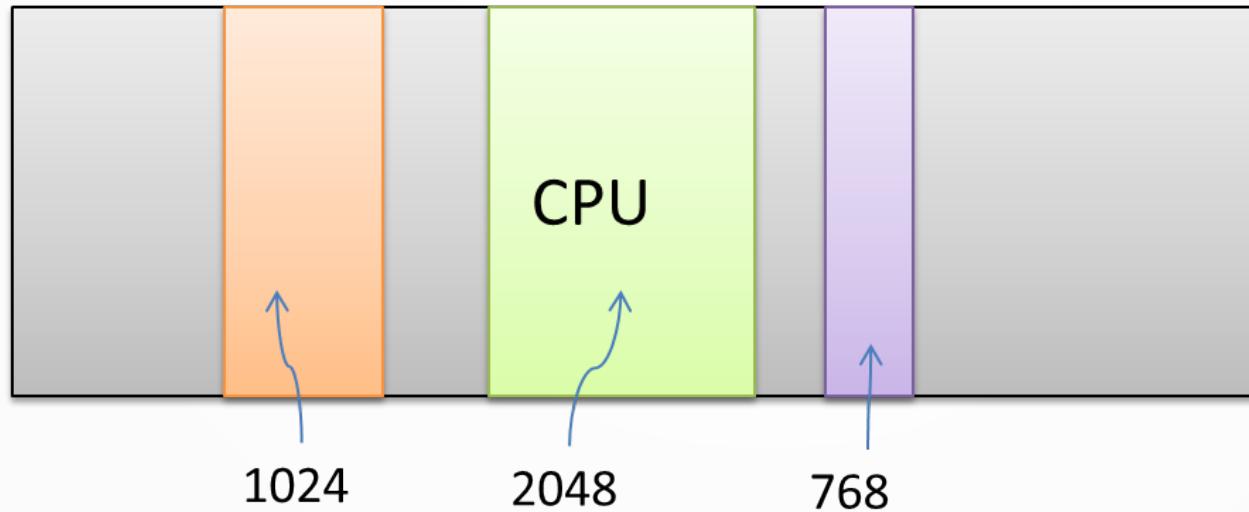
## Docker Grounds up: Resource Isolation

### Cgroups : Isolation and accounting

- cpu
- memory
- block i/o
- devices
- network
- numa
- freezer



# Cgroups – Control Groups



Cgroup #1

Cgroup #1 gets more CPU than Cgroup #3

Cgroup #2

Cgroup #2 gets twice CPU than Cgroup #1

Cgroup #3

Cgroup #3 gets less CPU than both other Cgroups

# Linux Capabilities

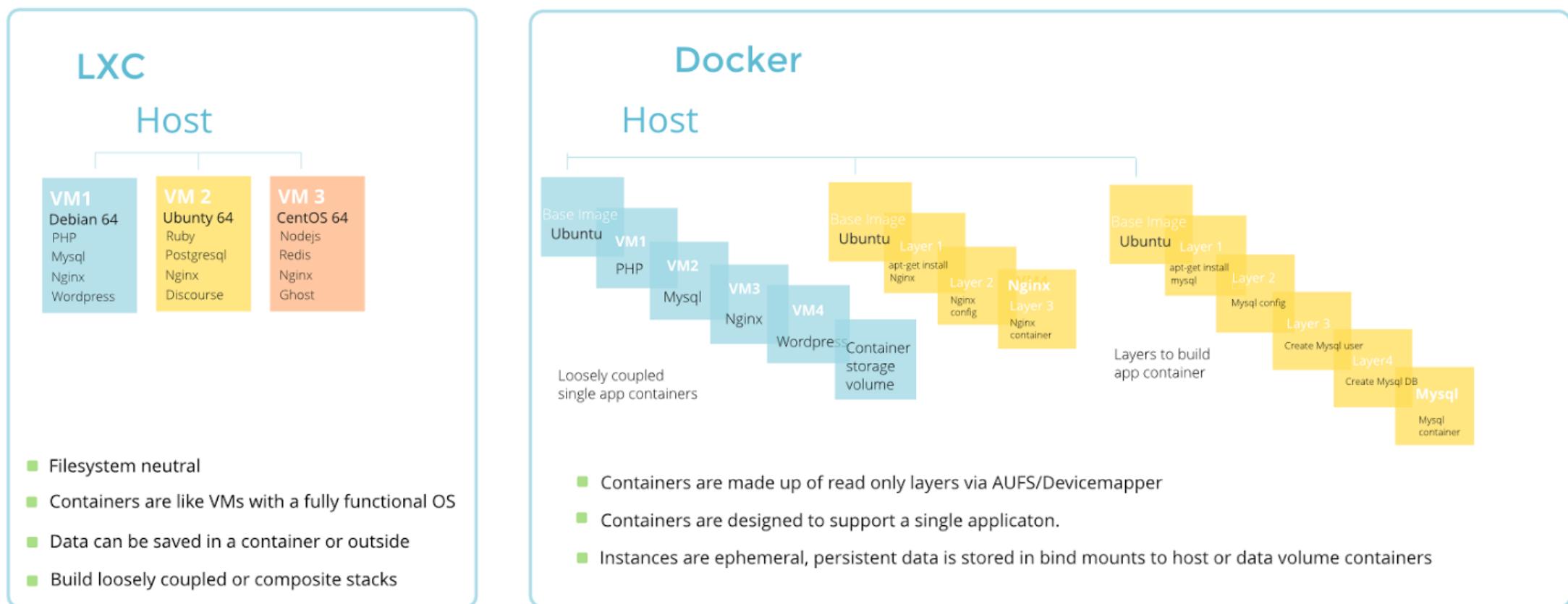
- Linux root privileges broken down into 32 capabilities
- Capabilities can be disabled per container.
- Two capabilities always dropped:
  - CAP\_NET\_ADMIN: capability to configure network (i.e. ip routes, firewall rules, etc.)
  - CAP\_SYS\_ADMIN: catch all capability, used to control mounts

# SECCOP

- Containers can interact with kernel system calls (syscalls)
  - i.e. bind, accept, fork
- SECCOMP limits the number of syscalls a container can access
- Docker disables around 44 of 300+ system calls with default options
  - i.e. mount, settimeofday

# Docker/OCI vs Linux Containers/LXC

## Key differences between LXC and Docker



# Benefits of Docker/OCI Containers

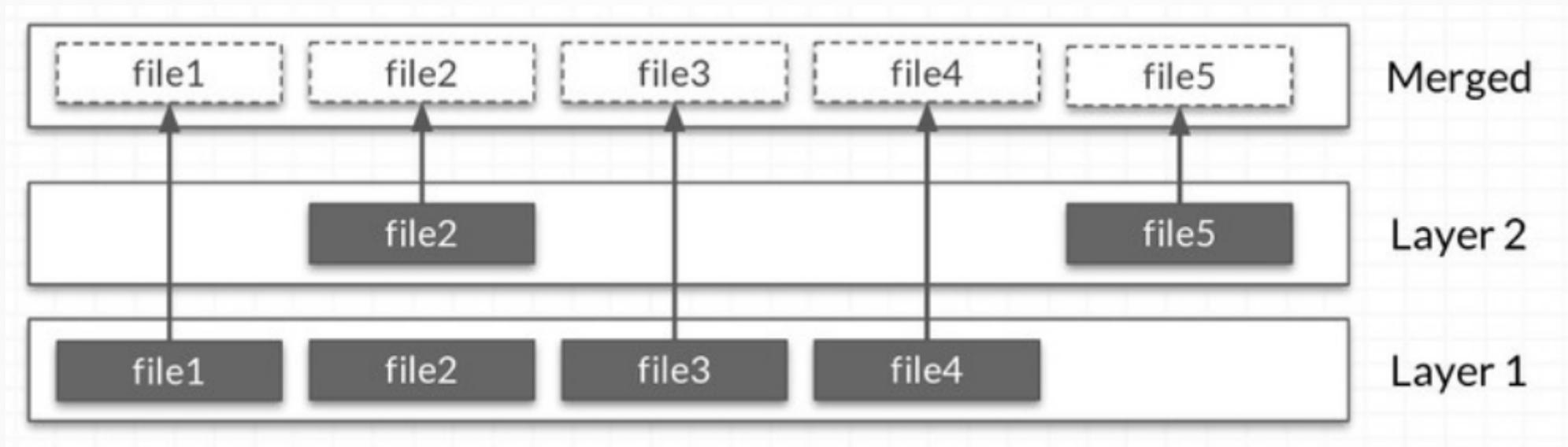
- Portable Deployment Across Machines
- App Centric
  - Sometimes referred to application containers vs machine containers
- Automatic Build
- Versioning
- Component re-use
- Sharing

# Docker/OCI: Images

- What is an Image?
  - Fully self-contained “thing” that contains everything an app needs to run
    - Application source code
    - All runtime dependencies, config files, and binaries
  - At the end of the day, it’s basically just a transportable file system
  - Images are Stateless and Immutable!
  - Share images by pushing to a registry
    - Docker Hub is the default Docker registry
    - Many other third-party registries exist
    - Once shared, others can pull the image

# Docker/OCI: Image Layers

- Images are composed of layers of filesystem changes
- Each layer can add or remove from the previous layer
- Each layer's filesystem changes are stored as a single tar file
- Joined together layers provide a full filesystem
- Container uses the merged view



# Docker/OCI: DockerFile

- Text file that contains a script to create an image
  - Each line creates a layer
  - Chain RUN commands together
- Allows various commands:
  - FROM – specify the parent image
  - COPY – copy files from the host into the image
  - RUN – run a command using binaries inside container (i.e. install)
  - CMD – specify the default command

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python python-pip
RUN pip install awscli
RUN apt-get autoremove --purge -y python-pip
```

Verses

```
FROM ubuntu
RUN apt-get update && \
    apt-get install -y python python-pip && \
    pip install awscli && \
    apt-get autoremove --purge -y python-pip && \
    rm -rf /var/lib/apt/lists/*
```

# Docker/OCI: Volumes

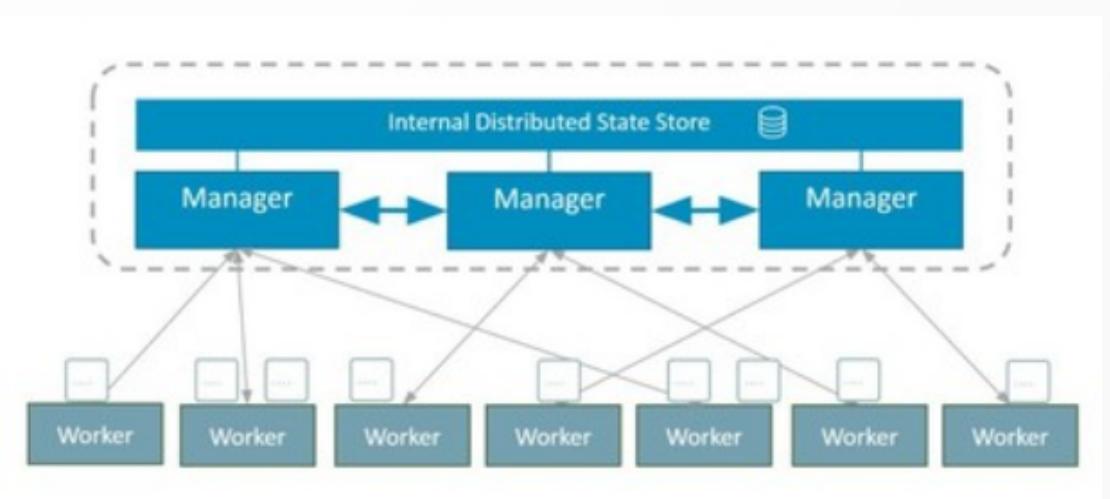
- Volumes provide the ability to persist/supply data in containers
- Bind mount volumes
  - You choose where to persist the data
  - Example: `-v /docker-data/mysql:/var/lib/mysql`
- Named volumes:
  - Let Docker choose where to persist the data
  - Example: `-v mysql-data:/var/lib/mysql`
- Issues with user permissions when accessing same files from both host and the container
  - Map a user ID in container to a host user ID

# Orchestration Overview

- Provides the ability to manage the running of container workloads, often over a fleet of machines
- As an administrator, you define the expected state, or the desired state
  - Includes service definitions, (images, ports, volumes, etc.), replica counts (how many of each to run), update policies, and more.
- The system then tries to make the actual state reflect the expected state
  - If a container exits, it'll try to restore it. If you change the expected state, it'll try to reconcile it.

# Actors in Orchestrators

- Every orchestrator has the concept of two types of nodes
- Managers
  - Serve the brains of the cluster
  - Maintain state & schedule work
  - Sometimes called masters
- Worker nodes
  - Perform the actual work, as instructed by a manager
  - Sometimes called agents or nodes



# Different type of Orchestrators

- Docker Swarm
  - Shipped with the Docker Engine
  - Very user friendly and easy to get up and running
  - Satisfies most needs, though not all; theoretically extensible, but takes a little work
- Kubernetes
  - Spun out of the work done within Google and contributed to CNCF
  - Think of it as a toolkit – so not as easy to get setup and running
  - Very configurable and extensible
- Amazon ECS
- Hashicorp – Separate Products for easier management
  - Terraform – Provision and Manage Bare Metal and Virtual Machines
  - Nomad – Provision and Manage Containers and legacy applications
  - Vault – Handle password, secret, and key management
  - Consul – Orchestrate and Manage worker nodes and provide connections to outside world