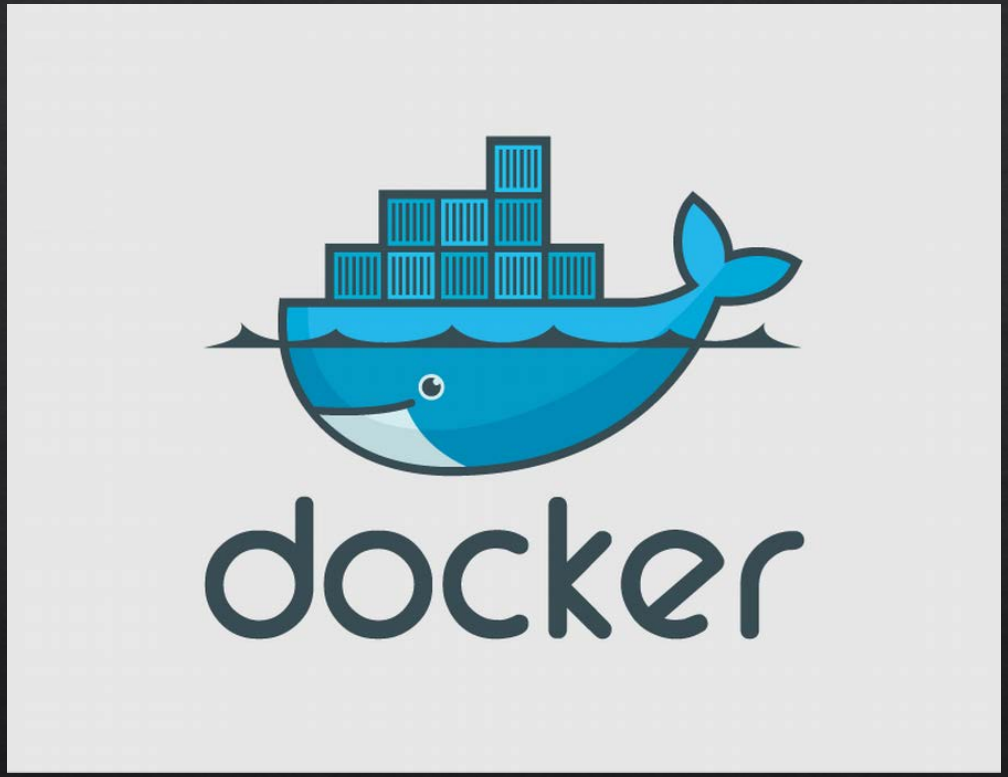


https://www.docker.io/

What is it?

- Docker is an open-source project to easily create lightweight, portable, self-sufficient containers from any application
- Makes fast running, portable **containers**

History

- Dotcloud, Inc creates PaaS service
- January 2013, work starts on docker internally
- March 2013, first public release
- Statistics:
  - 16853 stars on github
  - 3343 forks
  - 684 contributors
- Massive community interest
- Created by Solomon Hykes (French engineer ;)

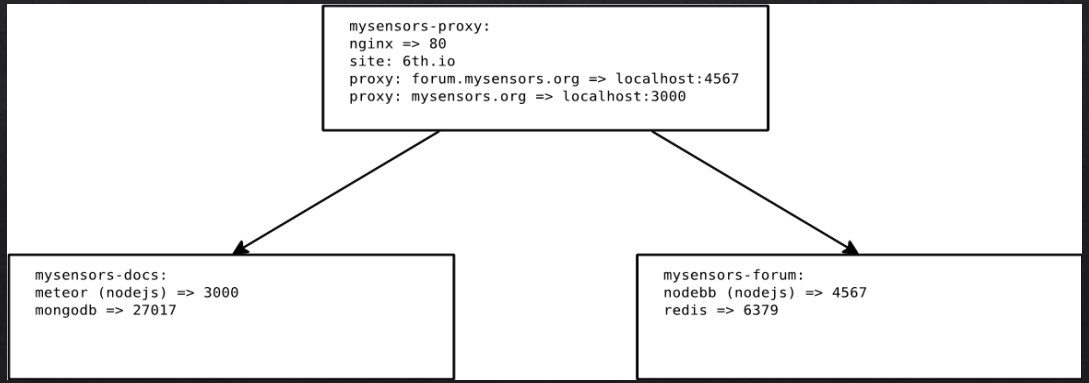
Why this hype?

- Solves an important problem
- Easy to use
- Efficient

Why docker?

Problem

- An application have many dependencies that needs to coexist
- Deployments with different demands:
  - Development/CI
  - Test
  - Production



Application Deployment got complex

Application Stack

- Basic OS
- JVM
- Static web server
- Front-end platform
- Database layer
- Application code

X

Deployment environments

- Development VM
- QA Server
- Customer Data Center
- Public Cloud
- Contributors Laptop
- Production Servers
- Production Clusters

Problems

- Installing software
- Software versoning
- Configuration
- Testing

What is docker?

Analogy

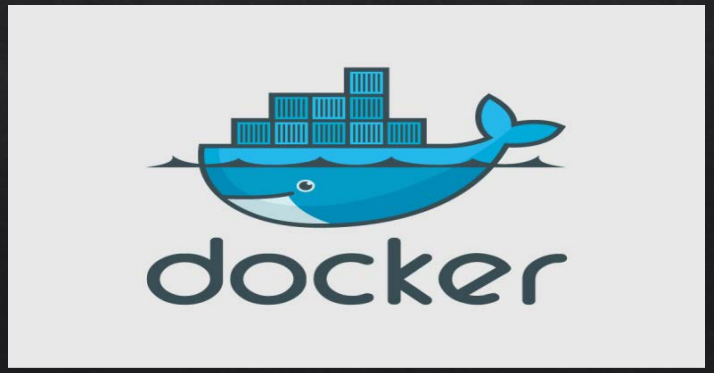
- Transporting of goods before 1960

Solution: Shipping container

- Separation of concerns
  - User cares about packing the inside
  - Shipper cares about moving the container
- Standardized interface

What is docker?

Solution
- Containers



Docker containers

Standardized interface for software container

Developer concerns

- Code
- Libraries
- Services
- Configuration
- Data

All servers look the same

Ops concerns

- Moving containers
- Starting/Stopping containers
- Logging
- Monitoring
- Network configuration

All containers look the same

Isolation

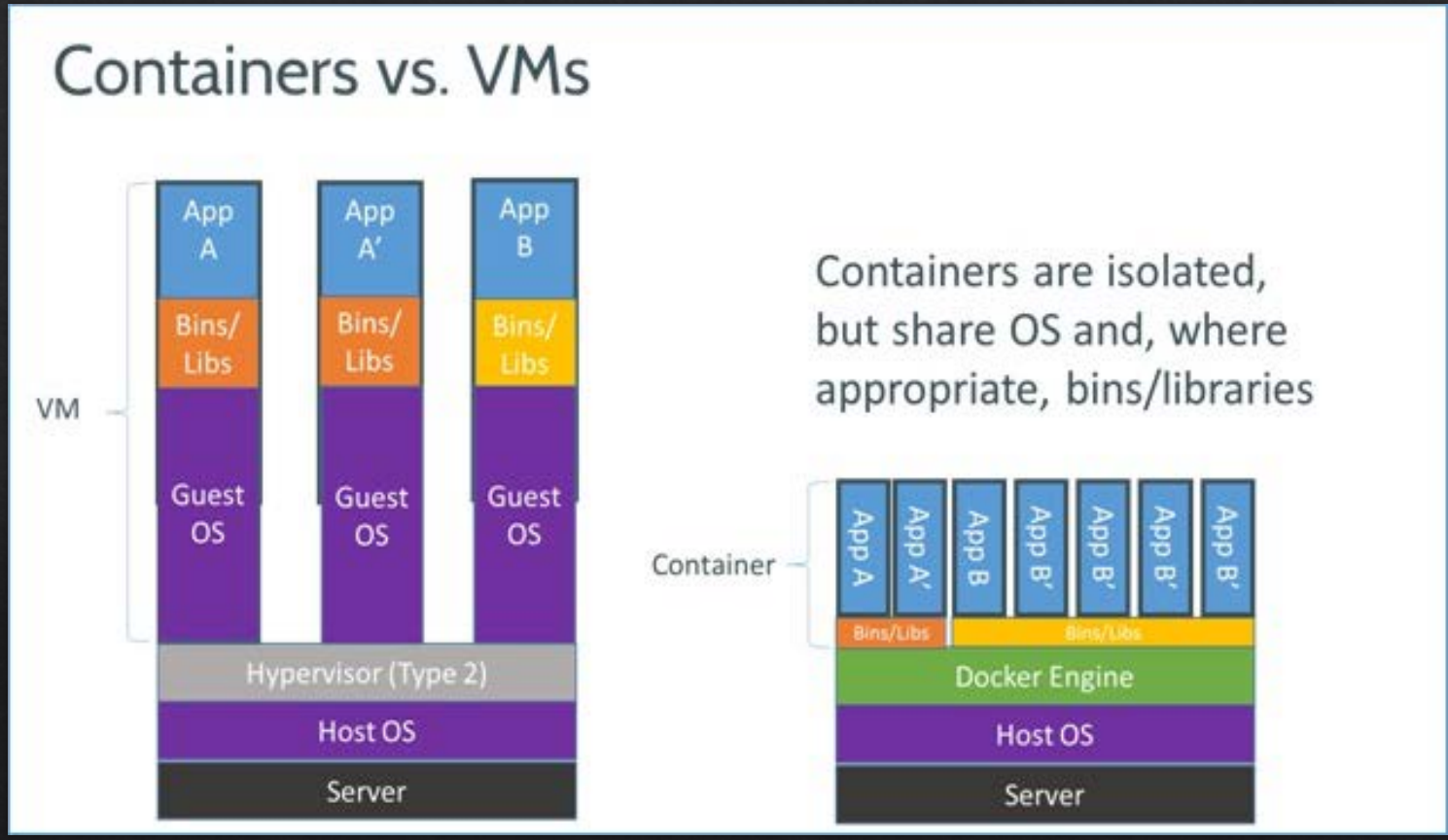
What is a Container?

- **Operating system–level virtualization**
- Operating system–level virtualization is a server virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one
- Differs from traditional virtual machines

Containers vs. VMs

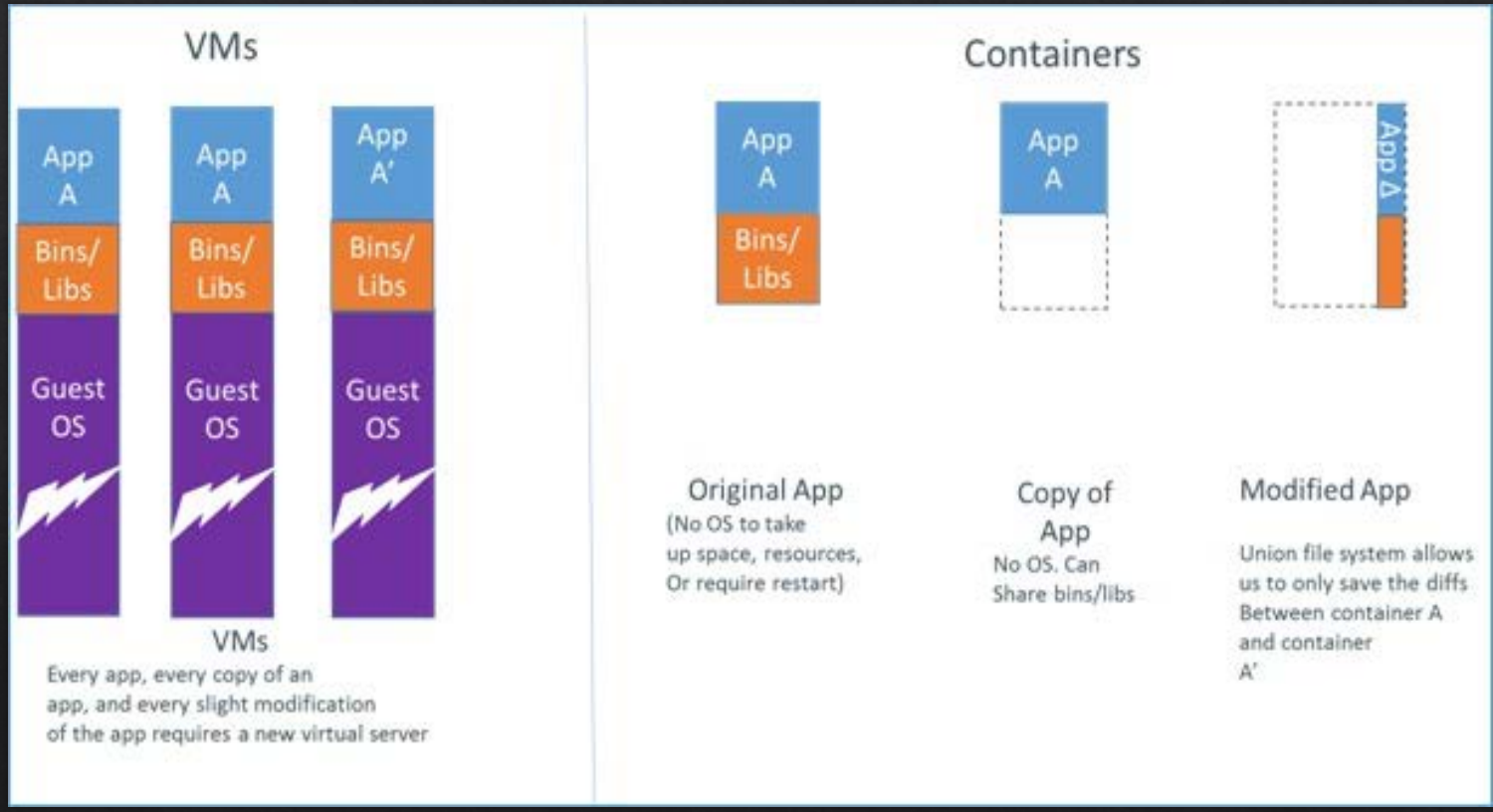
- Container
  - Shares the host OS and kernel
  - Zero boot time
  - Cannot run any OS (strictly Linux for Docker)
  - Little to no start-up or performance penalty. Technically native.
- Virtualization
  - Full OSes on top of a host OS via hypervisor
  - Full software stack
  - Each VM has it's own kernel
  - Full boot process for each VM (slow)
  - Can run any OS (Windows and BSD included)

Containers vs. VMs (Pretty Picture)



Pretty picture from https://www.docker.io/the_whole_story/

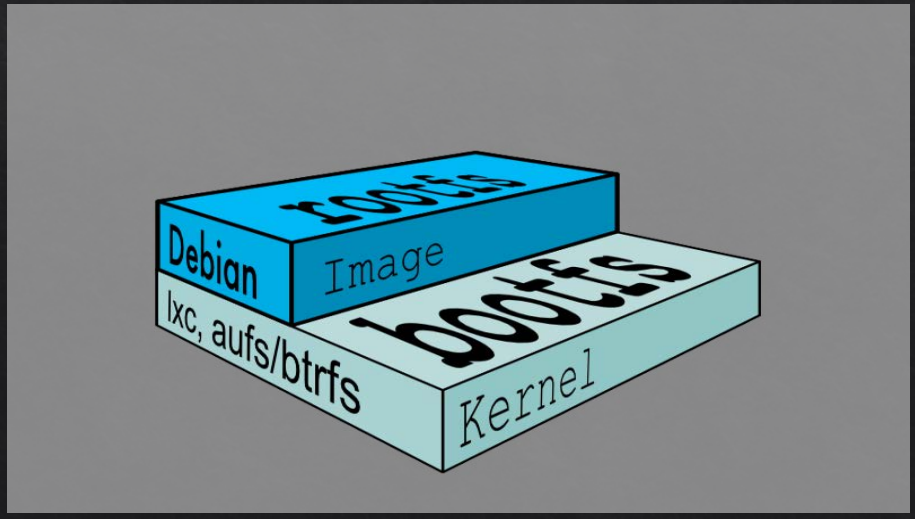
Containers vs. VMs (Another Pretty Picture)



Another pretty picture from https://www.docker.io/the_whole_story/

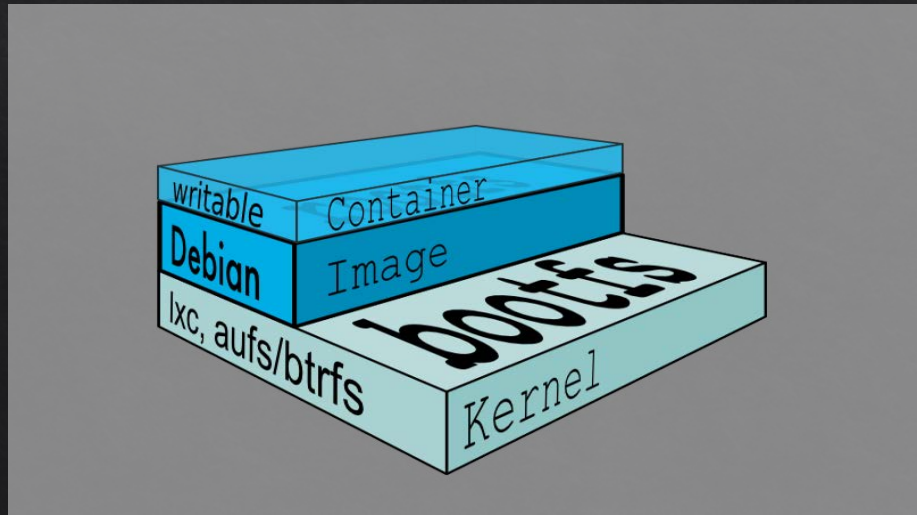
Docker glossary

- Image
  - Read-only template for a container
  - Includes all files required for application to run
  - Has additional metadata
    - Exposed network ports
    - Binary to start



Docker glossary

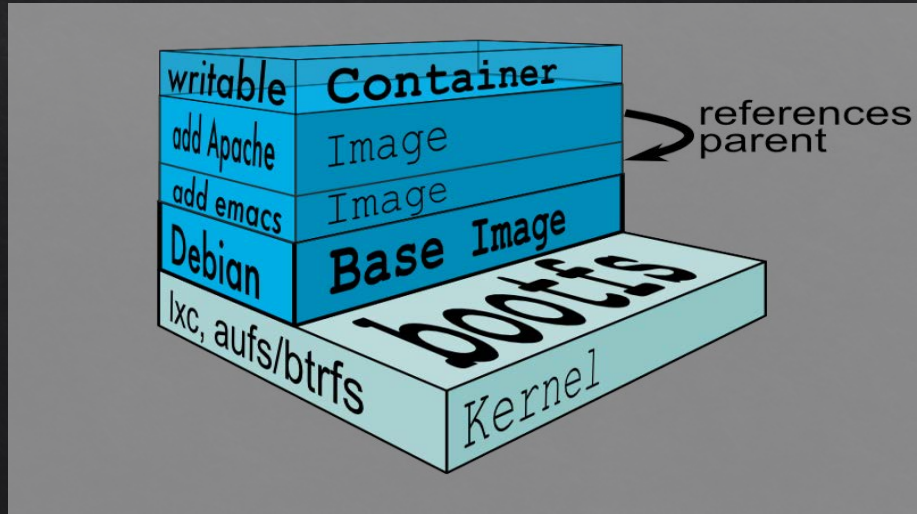
- Container
  - Running processes
  - Based on a particular image
  - Typically a single process
  - Isolated from host system
  - Cheap
  - Can write to filesystem
  - Commit creates new Image

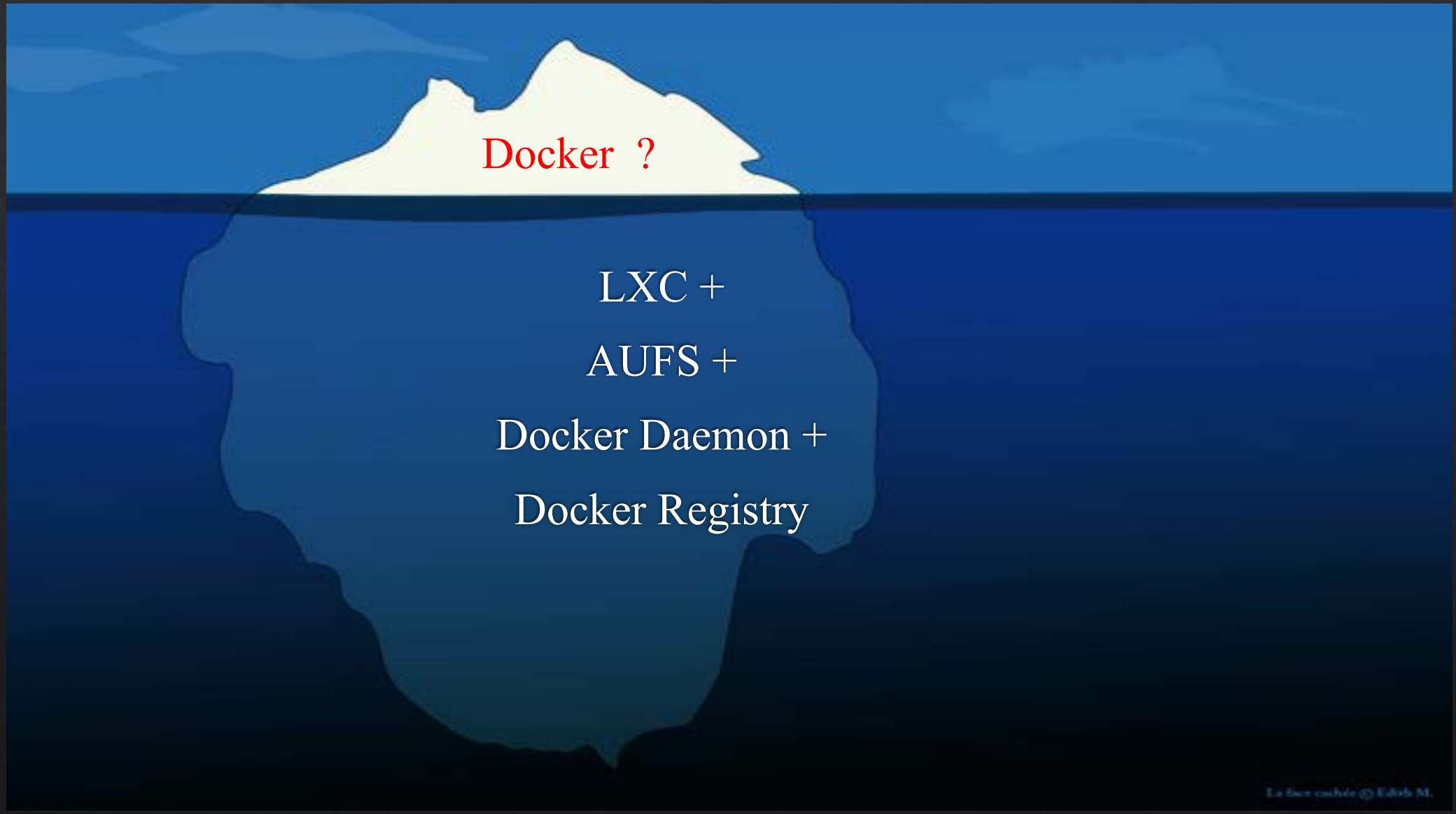


Docker glossary

Layers

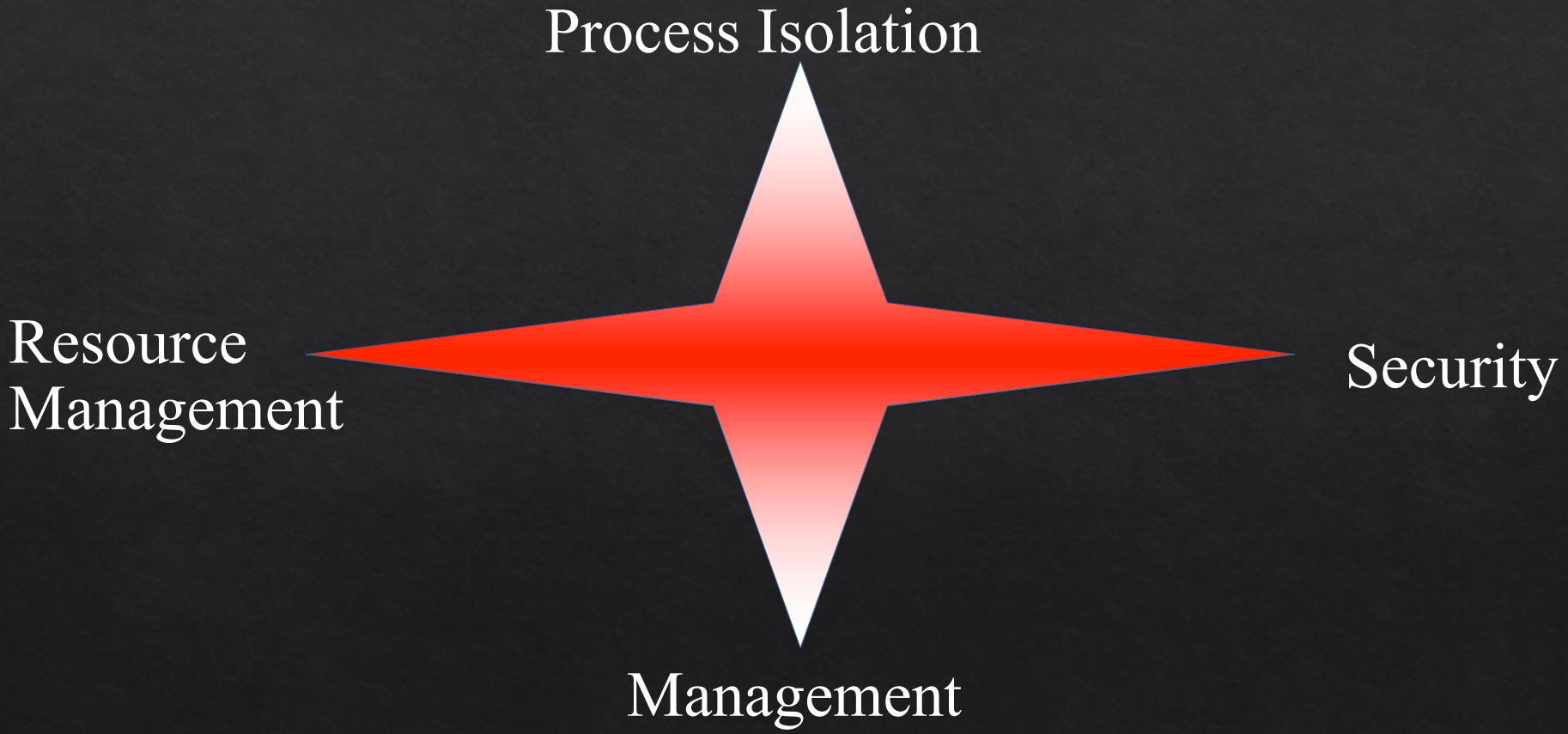
- Images are based on a parent
- The layers stack on top
- Files in base layers are shared between Images
- Each commit creates a layer
- Base image has no parent




Docker ?
LXC +
AUFS +
Docker Daemon +
Docker Registry

Containers are a userspace concept that takes advantage of several Kernel Subsystems

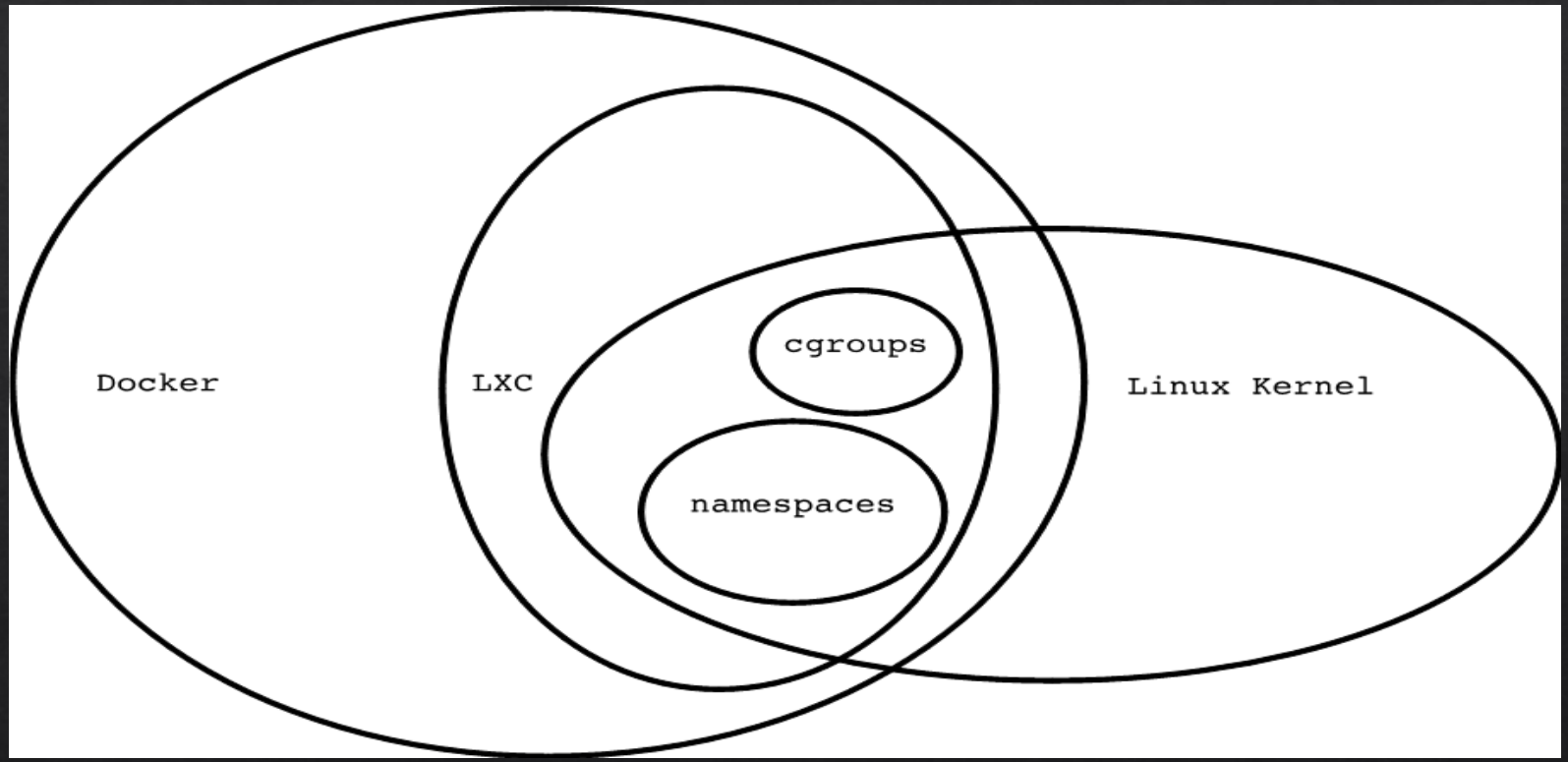
Key elements of Linux Containers



How does this work

- cgroups
  - Group trees of processes
  - Allows control of resources for the group
- Namespaces
  - Isolate processes from host
  - Network, filesystem, pids, etc
- AuFS/DeviceMapper/Btrfs
  - CoW snapshoting of filesystem images

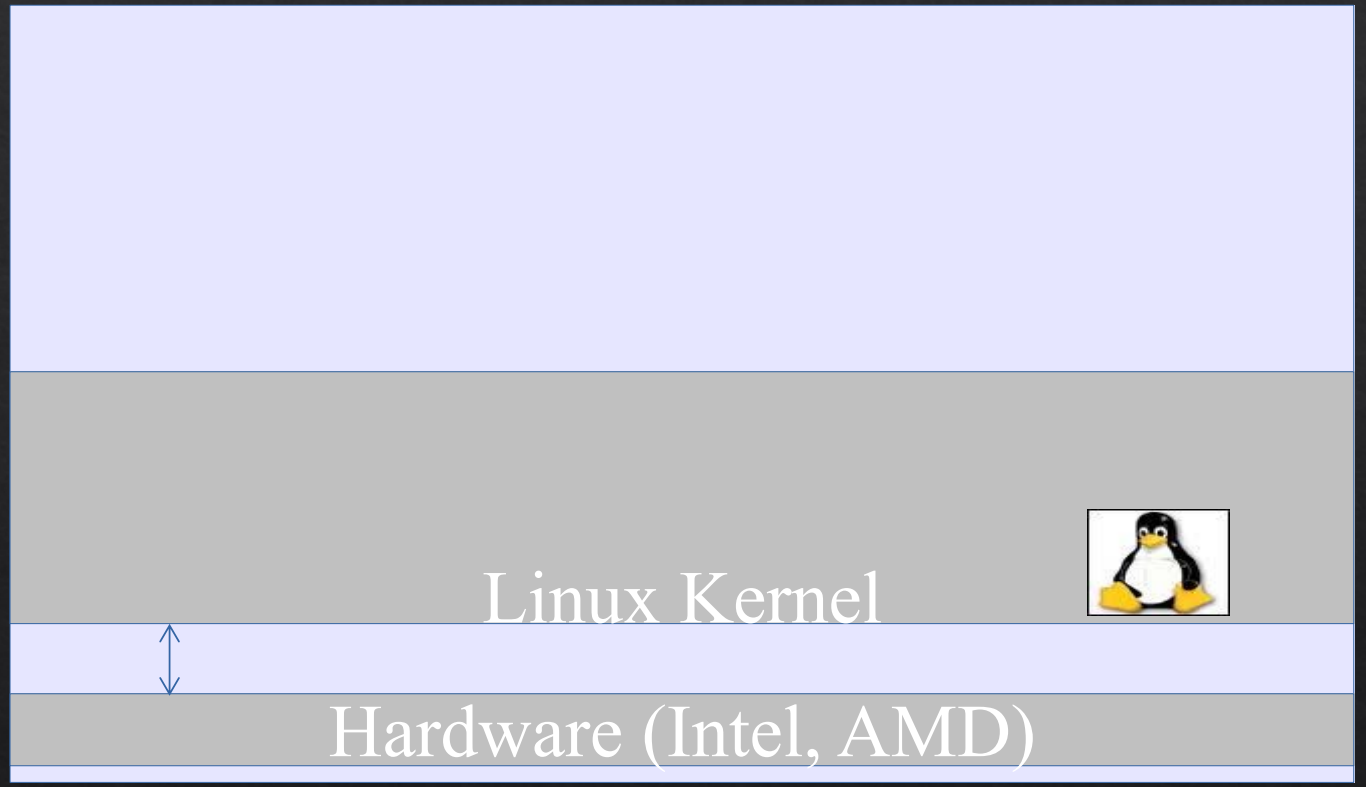
How does it work?



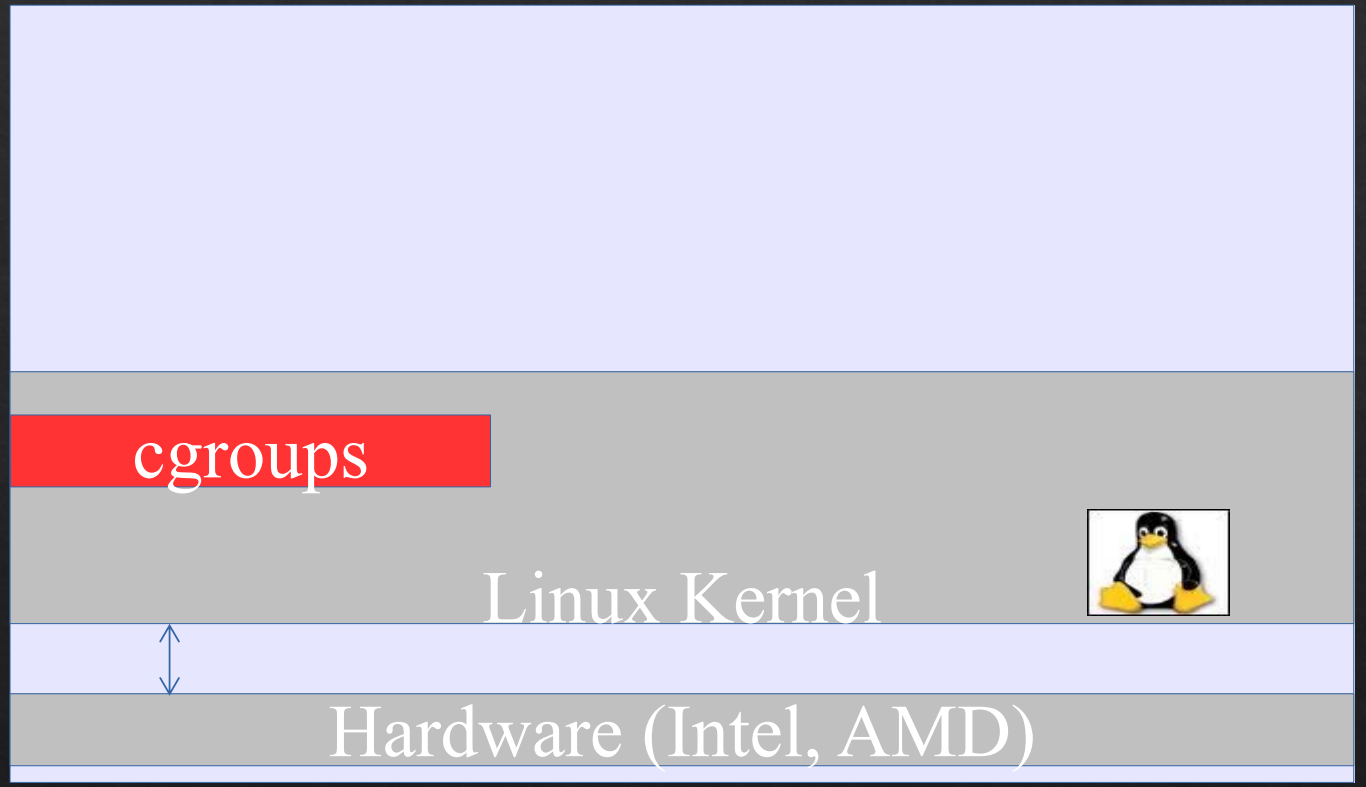
LXC (Linux Containers)

- Capability is already built into the kernel
- Utilizes cgroups (control groups) to limit, account and isolate resource usage (CPU, memory, disk I/O, etc.).
- **Stuff runs isolated from the rest of the host OS**

Linux Container Architecture



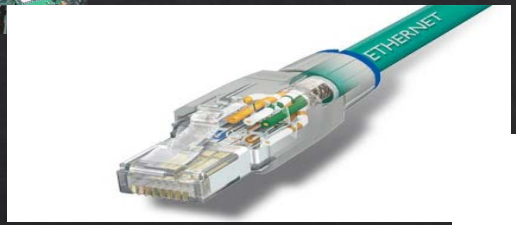
Linux Container Architecture



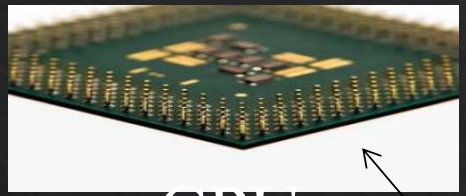
Cgroups

Resource Management

Memory



Network



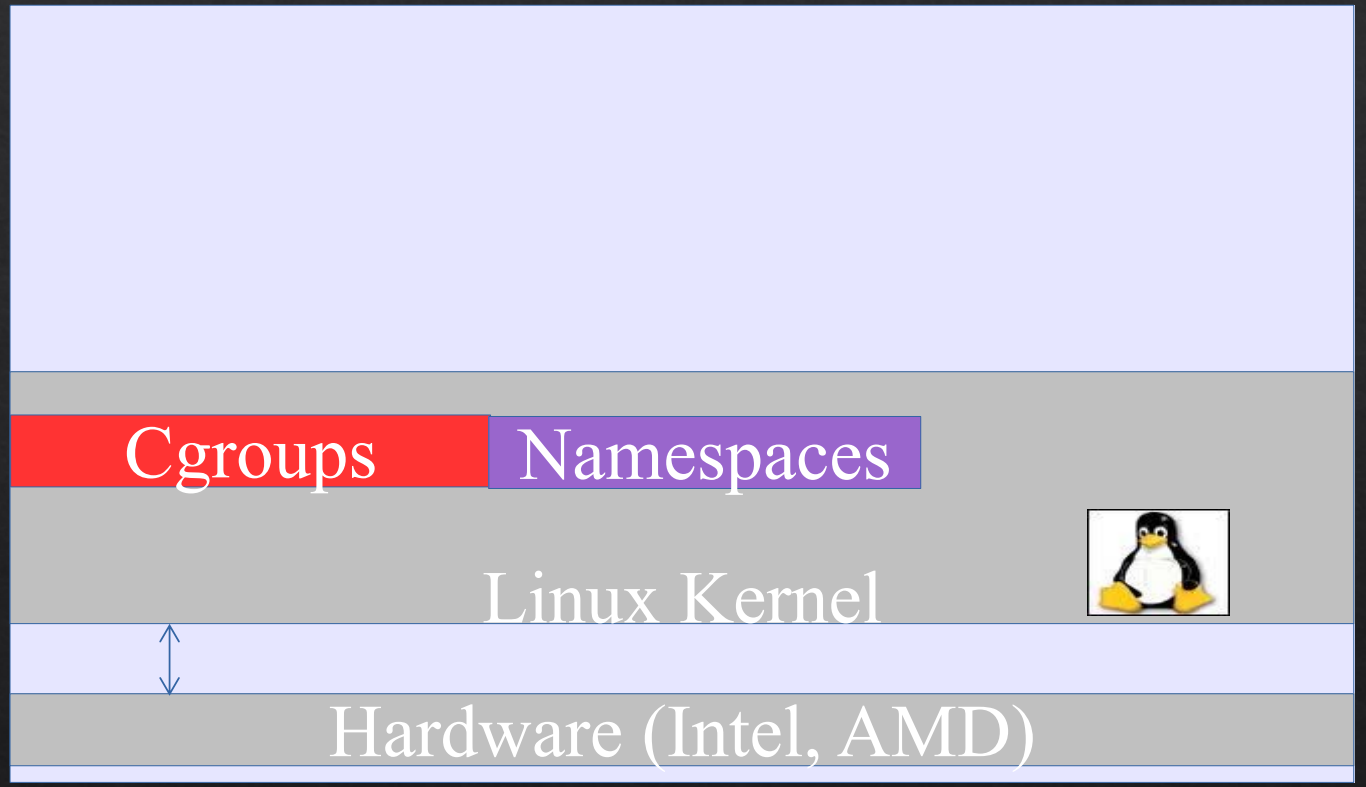
CPU

Block IO

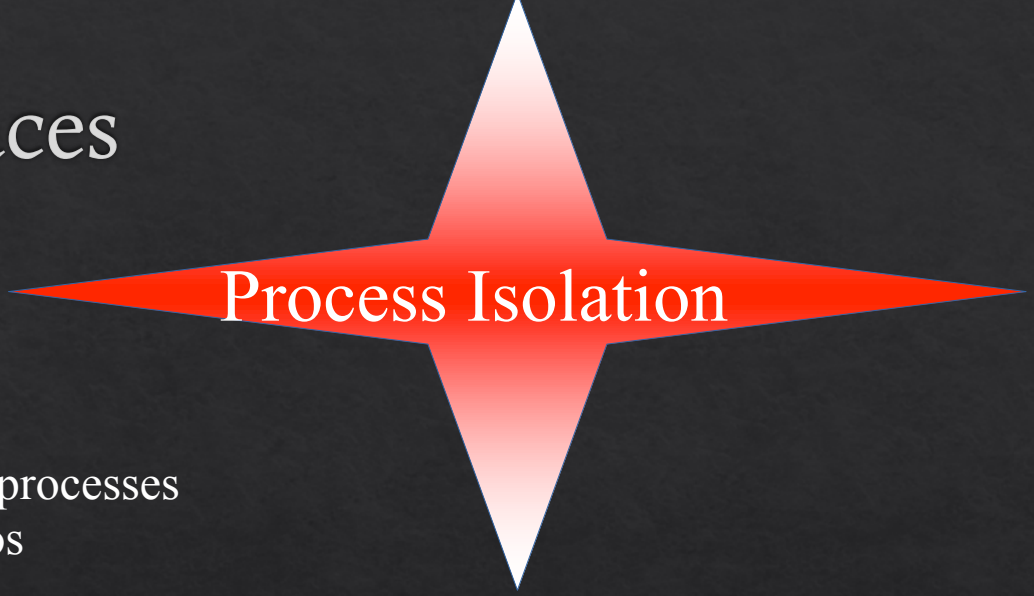
Linux Kernel

Hardware (Intel, AMD)

Linux Container Architecture



Namespaces



- Isolate processes

–Create a new environment with a

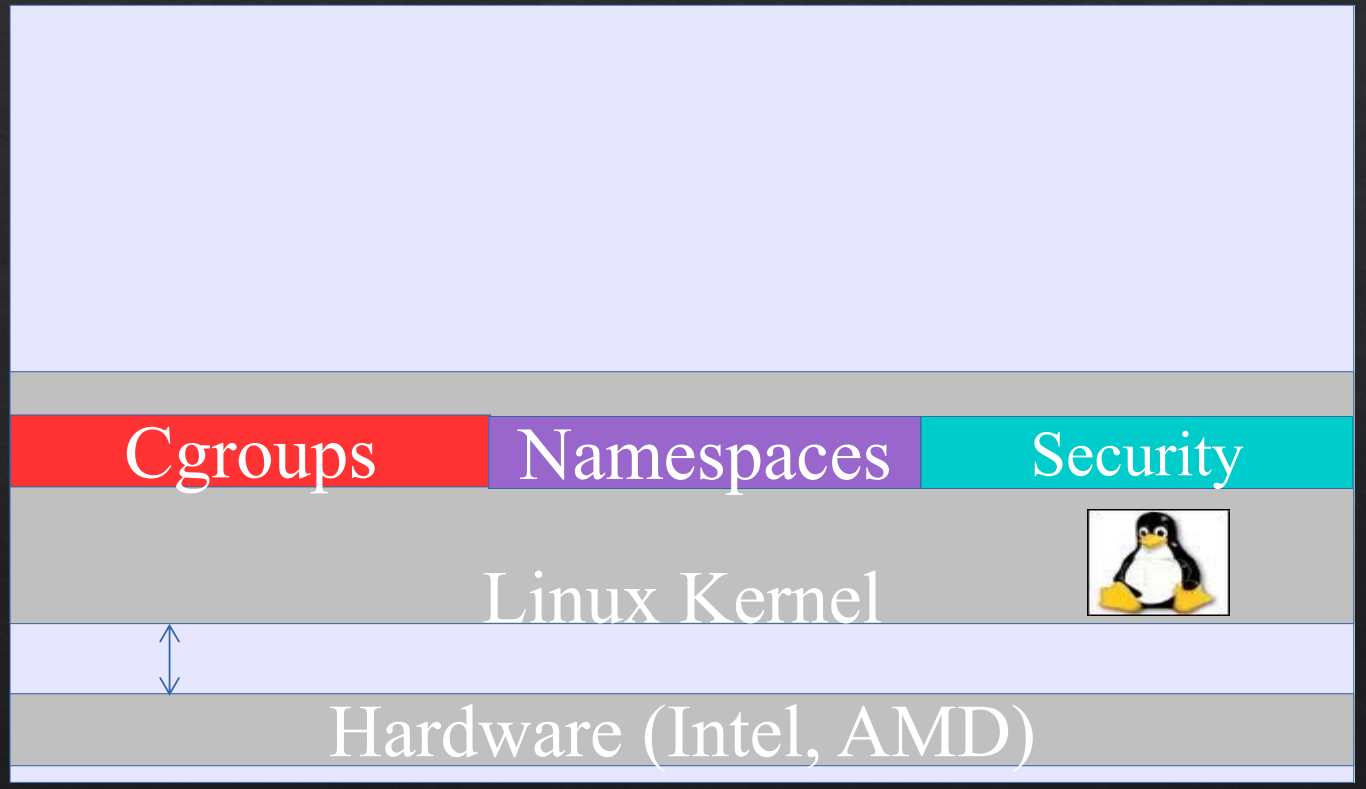
–Subset of the resources

- Once set up, namespaces are transparent for processes
- Can be used in custom and complex scenarios
- Supported Namespaces

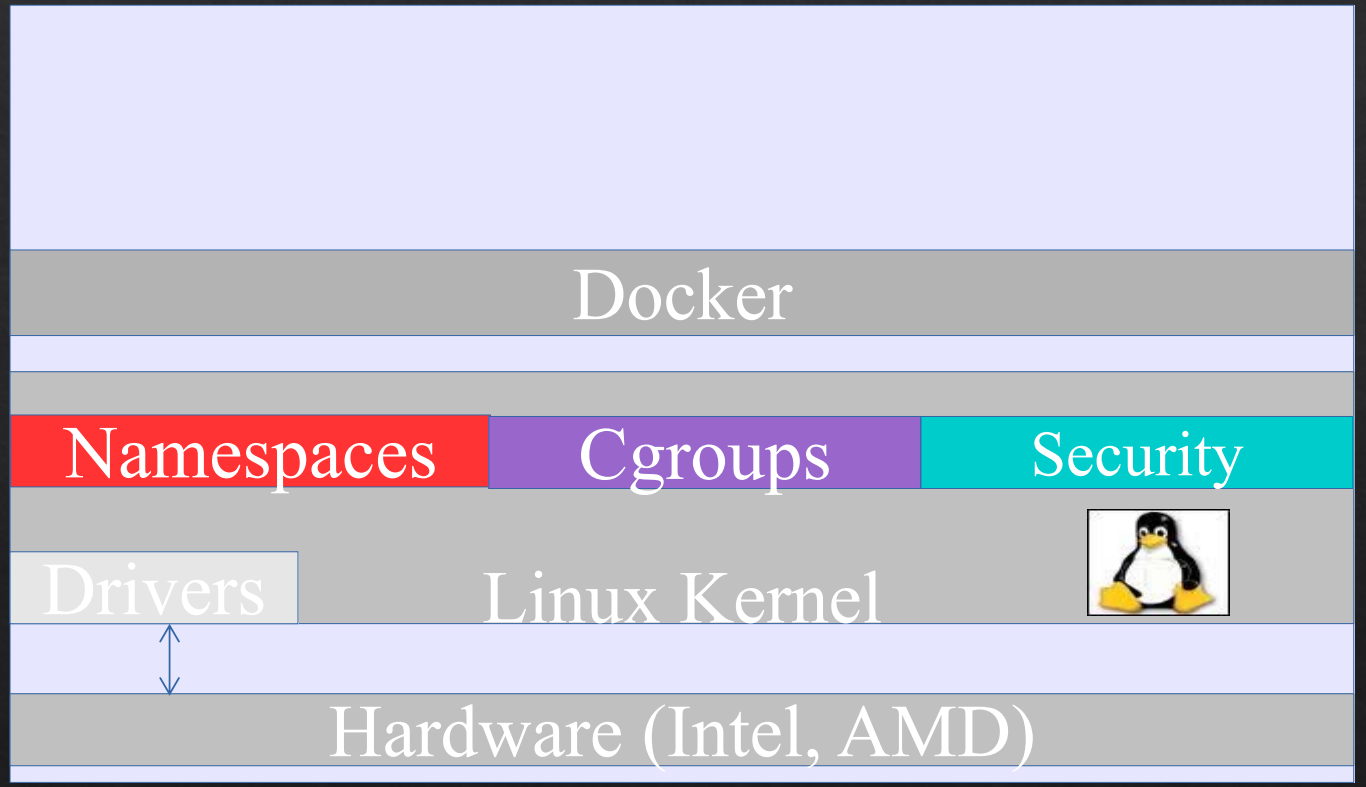
–ipc, pid, mnt, net, uts

–Future Red Hat Enterprise Linux 7: user

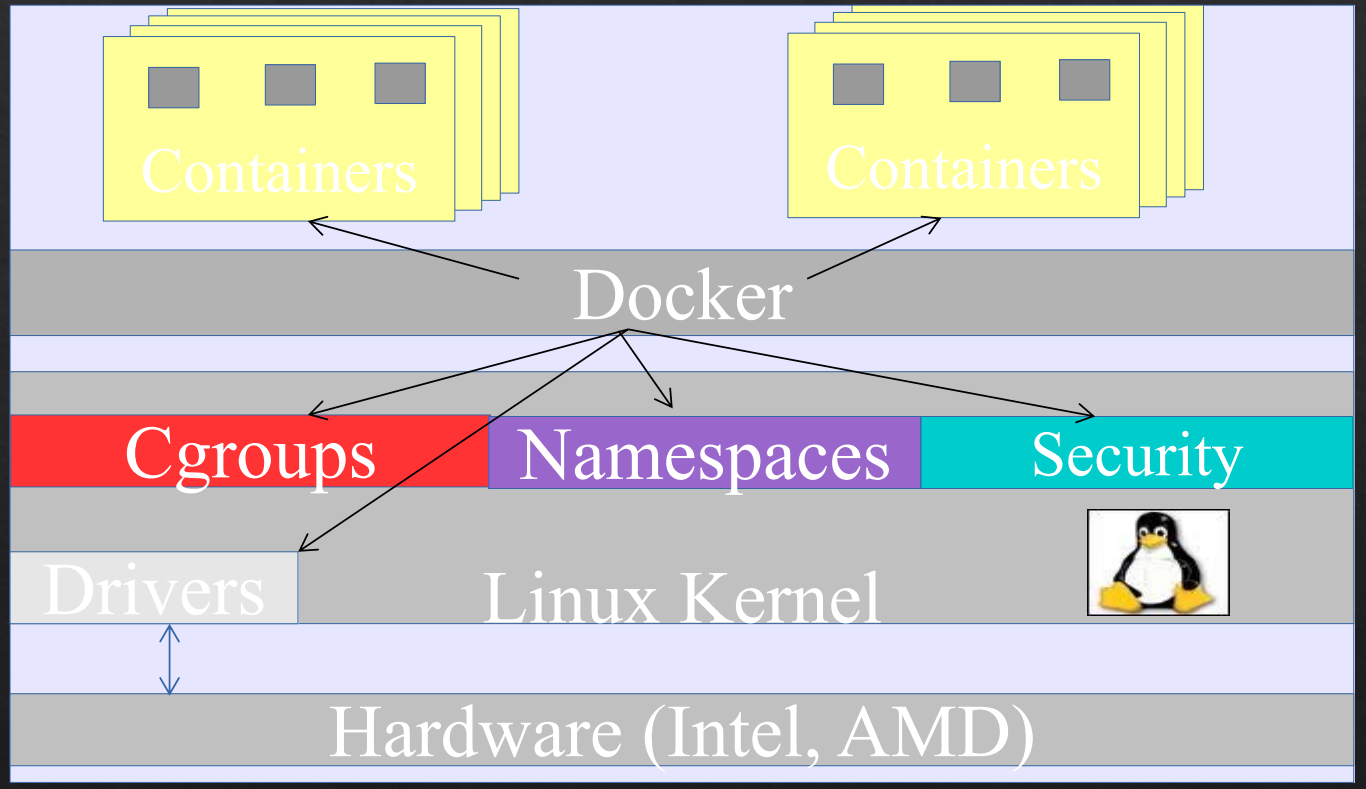
Linux Container Architecture



Linux Container Architecture

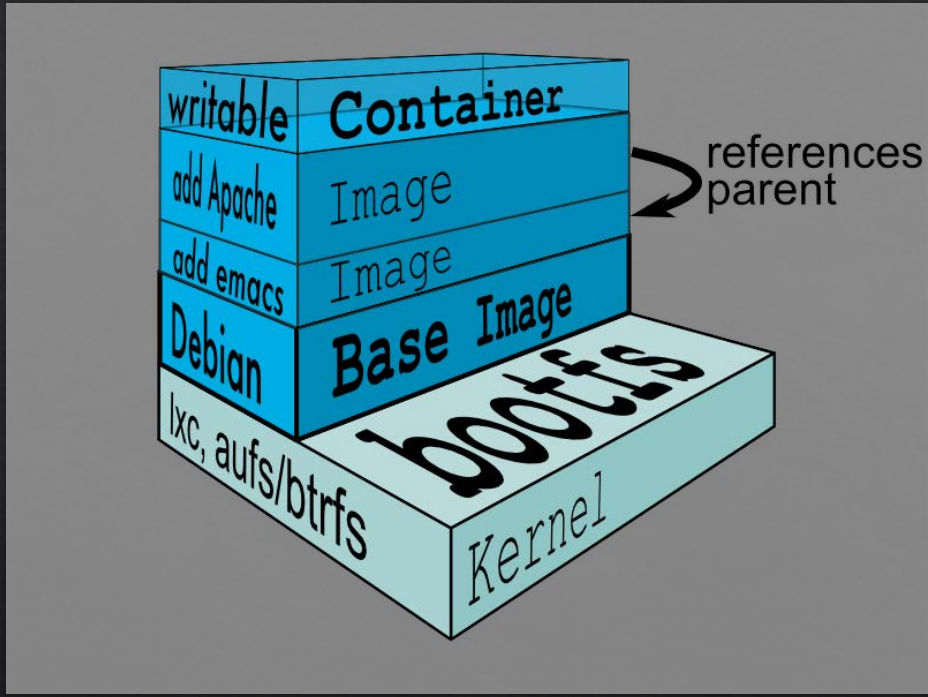


Linux Container Architecture



AUFS (AnotherUnionFS)

- Union file systems
  - Allows several file-systems to be mounted at one time, appearing to be one file-system.
- Docker uses it to create a layered file system.



Pretty picture from http://docs.docker.io/en/latest/terms/layer/

Docker Networking

- Docker uses a standard network bridge called docker0
- docker0 gets assigned an unused IP range
- Containers get bonded to docker0
- docker0's IP is the gateway for containers

Docker Daemon

- Software layer that allow for easy creation and management of Docker containers (glorified LXC instances).

Example Docker Work Flow (CentOS + SSH)

docker pull centos

docker run -t centos yum -y install openssh-server

docker commit <image id> centos/ssh

docker run -d -p 2222:22 -t centos/22 /usr/sbin/sshd

Host/Container Communication

- Ports can be exposed and mapped to the host with “docker -p”
  - docker run -p 80:80 -t <image> <cmd>

Container to Container Communication

- Handled with container linking
  - docker run -d -p 6666 --name parent <image>
  - docker run --link parent:child -i -t <image>
- Linked container will have environment variables like:
  - CHILD_PORT_6666_TCP_ADDR=172.17.0.2
  - CHILD_PORT=tcp://172.17.0.2

Docker Registry

- Software that runs the docker index at https://hub.docker.io/
- Base images for things like CentOS, Ubuntu, Fedora, etc are pulled from the public registry.
- Local registries can be created and submitted to.

Docker Files

- Allow for easy recreation of an application anywhere Docker can be ran. Example:

```
# use the ubuntu base image provided by dotCloud

FROM ubuntu


# make sure the package repository is up to date

RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list

RUN apt-get update


# install memcached

RUN apt-get install -y memcached
```

Uses for Docker

- Testing on multiple versions of multiple distros without the pitfalls of standard virtualiztion.
- Running newer or older versions of applications not in your host OSes repos.
- Creating an applications that can be easily rebuilt and reused on any distro, anywhere.

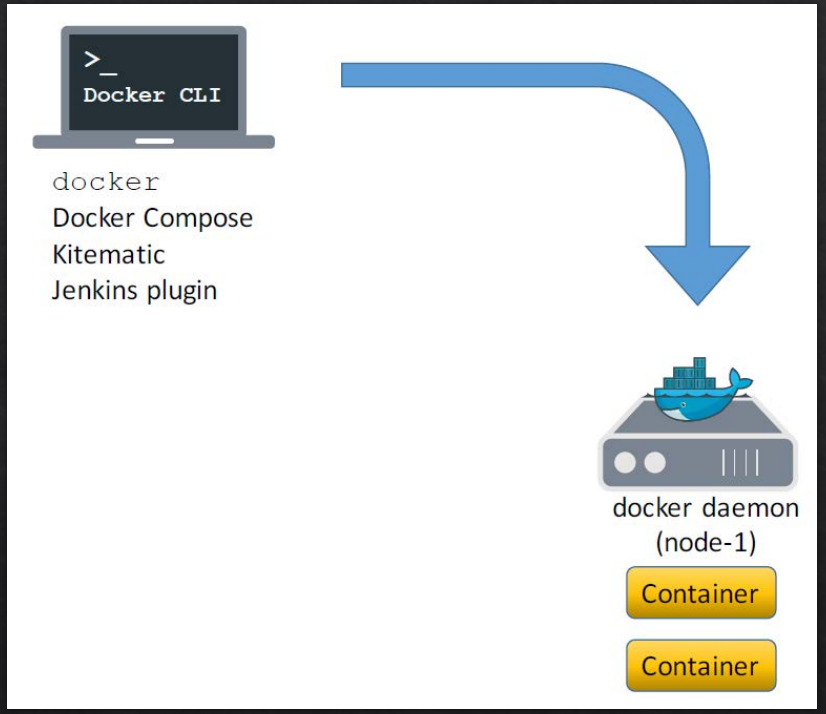
Docker Swarm

Running docker on a cloud

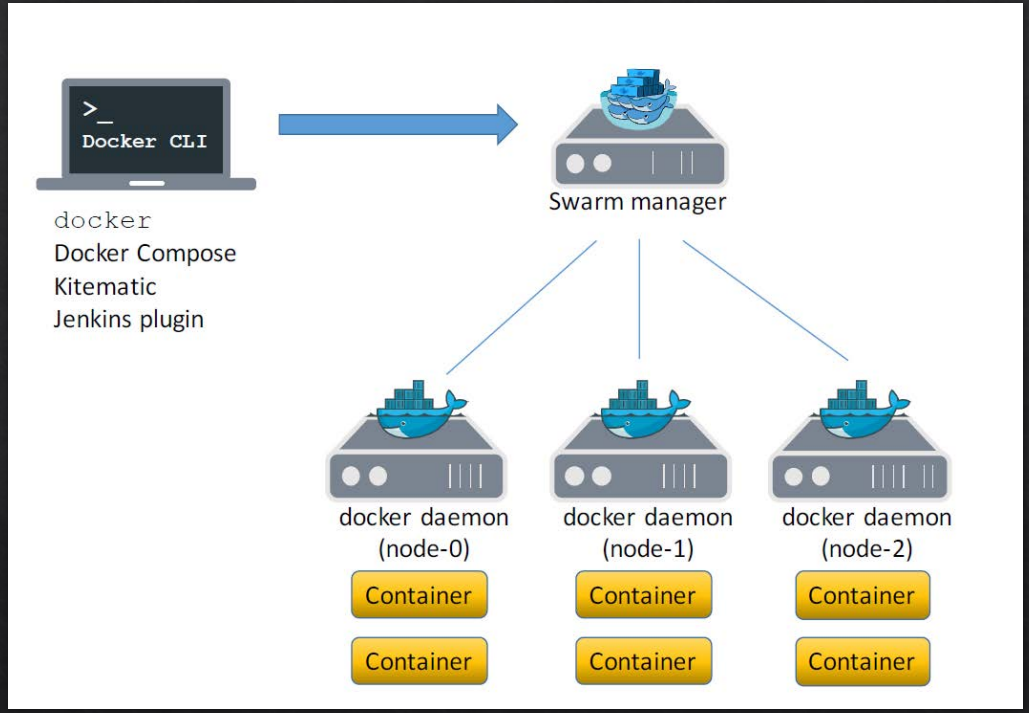
What is Swarm?

- Swarm turns multiple Docker hosts into a single, virtual Docker host.

Classical docker usage



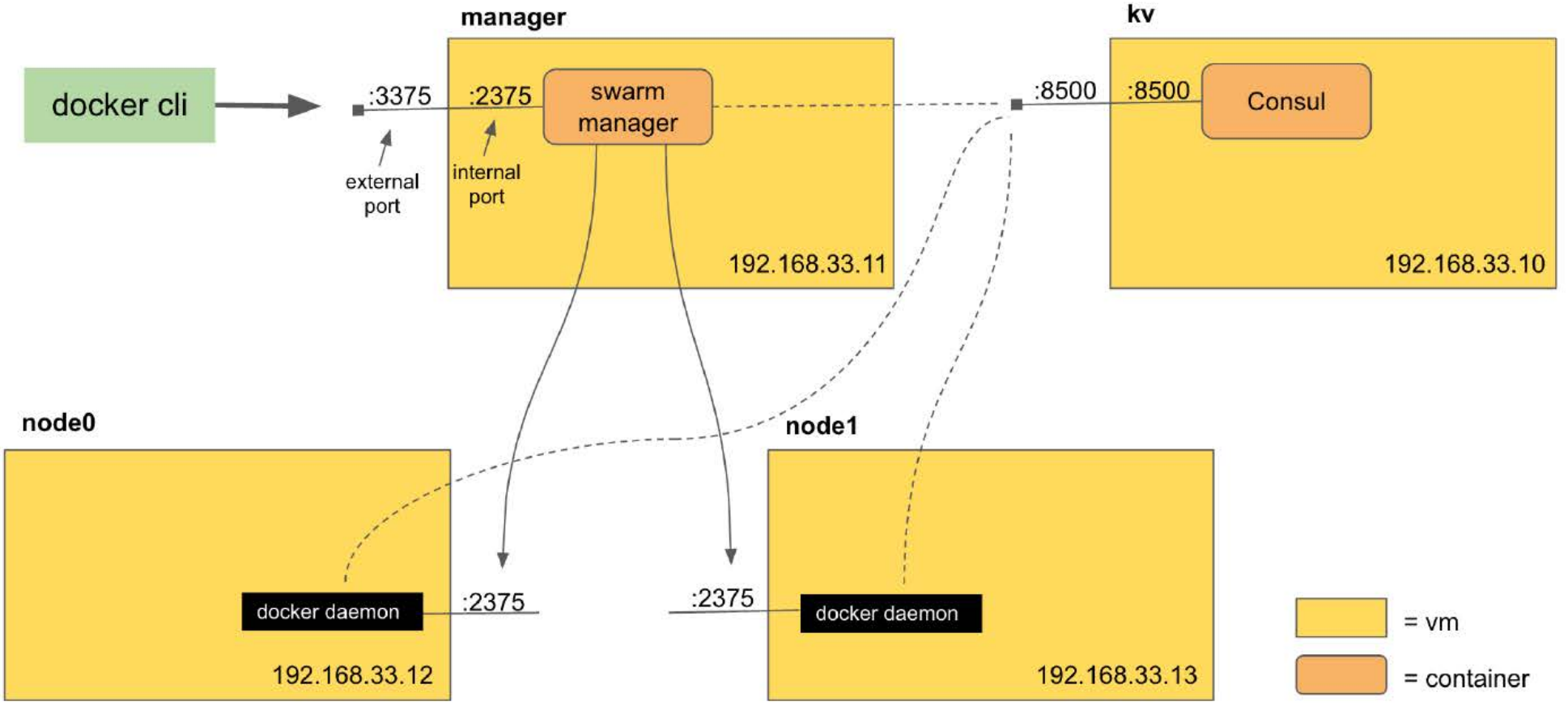
… with Swarm


>_
Docker CLI
docker
Docker Compose
Kitematic
Jenkins plugin
Swarm manager
docker daemon
(node-0)
docker daemon
(node-1)
docker daemon
(node-2)
Container
Container
Container
Container
Container
Container

Swarm features

- Scheduling and rescheduling on failure
- Labels, affinity and constraints to guide the placement of containers

Swarm Cluster



3 steps to run your swarm cluster

- 1/ Create a KV (for node discovery and lib networkIP allocation)
- 2/ Run your Swarm manager (on a single node)
- 3/ restart your Docker daemon with swarm parameters (on all your worker nodes)

Docker Compose

Multi-container apps are a hassle.

- Build images from Dockerfiles
- Pull images from the Hub or a private registry
- Configure and create containers
- Start and stop containers
- Stream their logs

Multi-container apps are a hassle.

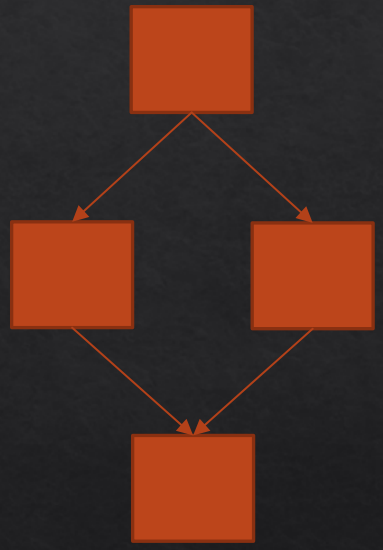
$ docker pull redis:latest

$ docker build -t web .

$ docker run -d --name=db redis:latest redis-server --appendonly yes

$ docker run -d --name=web --link db:db –p 5000:5000 -v `pwd`:/code web python app.py

Multi-container apps are a hassle.



$ docker pull ...

$ docker pull ...

$ docker build ...

$ docker build ...

$ docker run ...

$ docker run ...

$ docker run ...

$ docker run ...

Docker Compose:
Get an app running in one command.

Native Docker API

- The same API used by **other Docker commands**.
- The same API used by **other Docker tools**.
- The same API **exposed by Docker Swarm**.

- **Multi-container apps on multi-host clusters.**

Built into the Docker client

- New command: docker up
- Enhanced commands: docker build, pull, run, start, stop, kill, rm…

A small toy example of docker compose file

Create a file called *.yml in your project directory :

```yaml
version: '2'
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: "redis:alpine"
```

This Compose file defines two services, web and redis. The web service:

- Uses an image that's built from the Dockerfile in the current directory.
- Forwards the exposed port 5000 on the container to port 5000 on the host machine.
- Mounts the project directory on the host to /code inside the container, allowing you to modify the code without having to rebuild the image.

The redis service uses a public Redis image pulled from the Docker Hub registry.

Summary

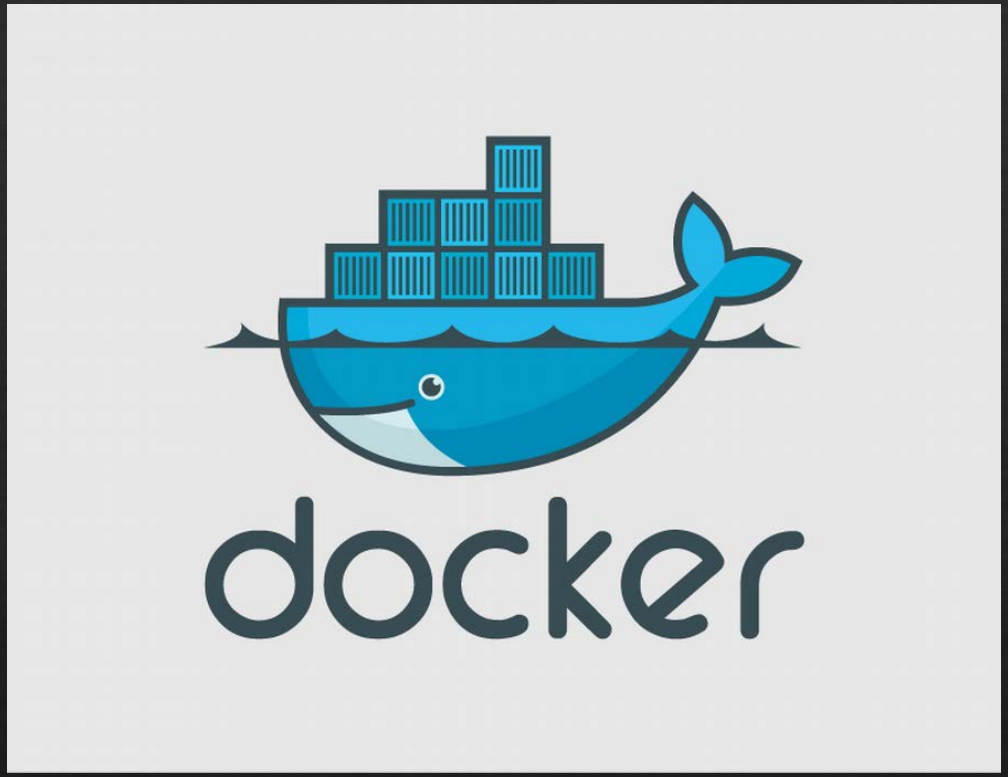
Advantages:

- Portable configuration
- Reuse images other people have built
- Lightweight, fast
- Easy to scale up
- ”Build once run everywhere”

Disadvantages:

- Fast moving

Thank You



https://www.docker.io/