University of BRISTOL

Containers

Dr Dan Schien – COMSM0010

Lecture 7

bristol.ac.uk

Admin

- AWS – watch your cost
- Terminate when done
- Estimate cost before launching services

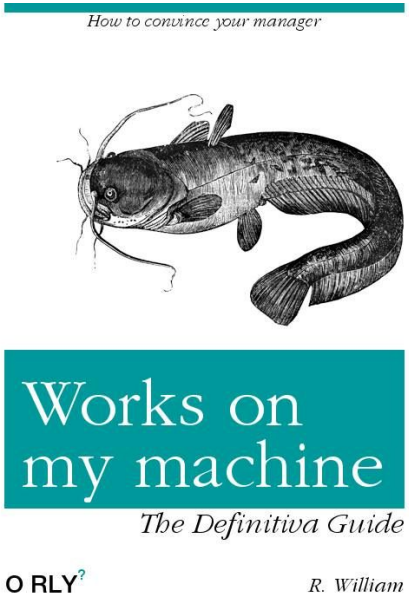
Goals

- Understand main concepts underlying Containers
- Be familiar with creating and running a container

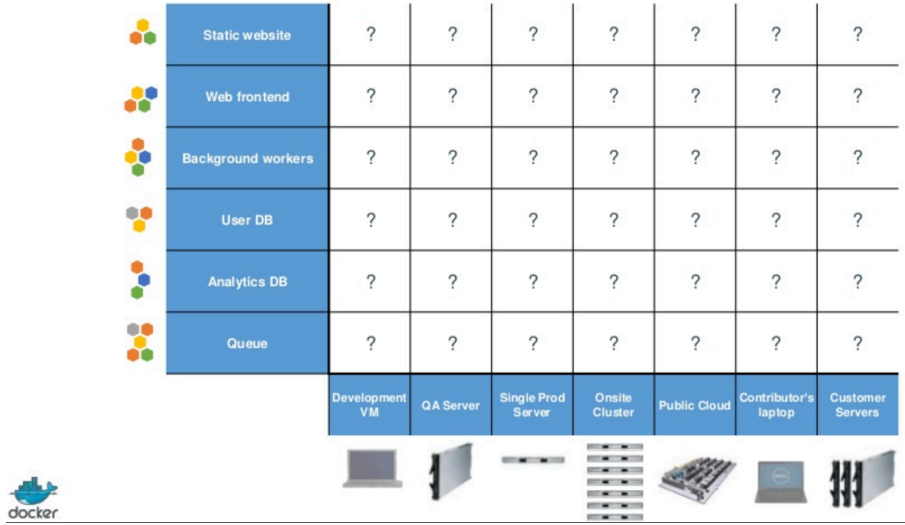
Backflash - Virtualisation

- Abstraction of compute resources (CPU, mem, IO)
- Multitenancy
- Consolidation
- Hypervisor manages guest access to host hardware

It works on my machine

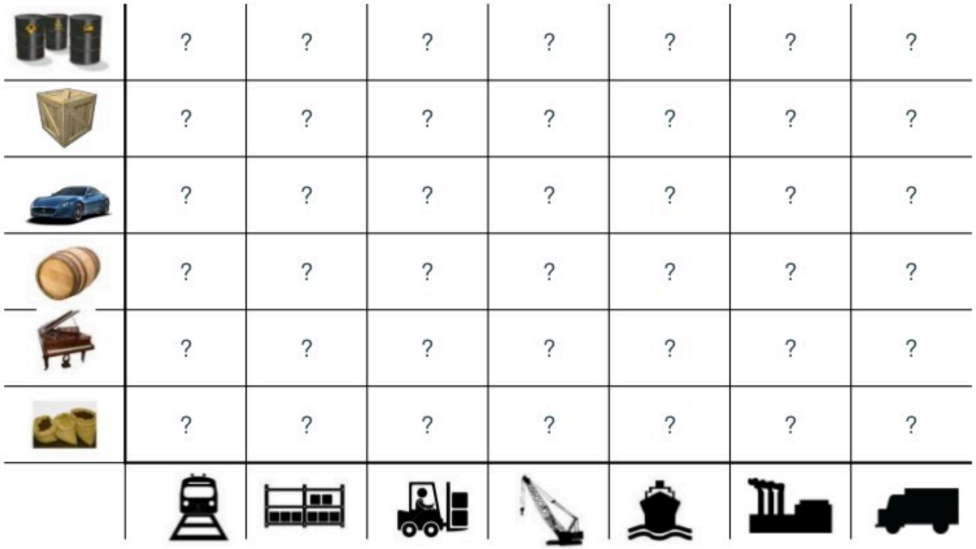


The Matrix From Hell



https://www.slideshare.net/dotCloud/docker-intro-november

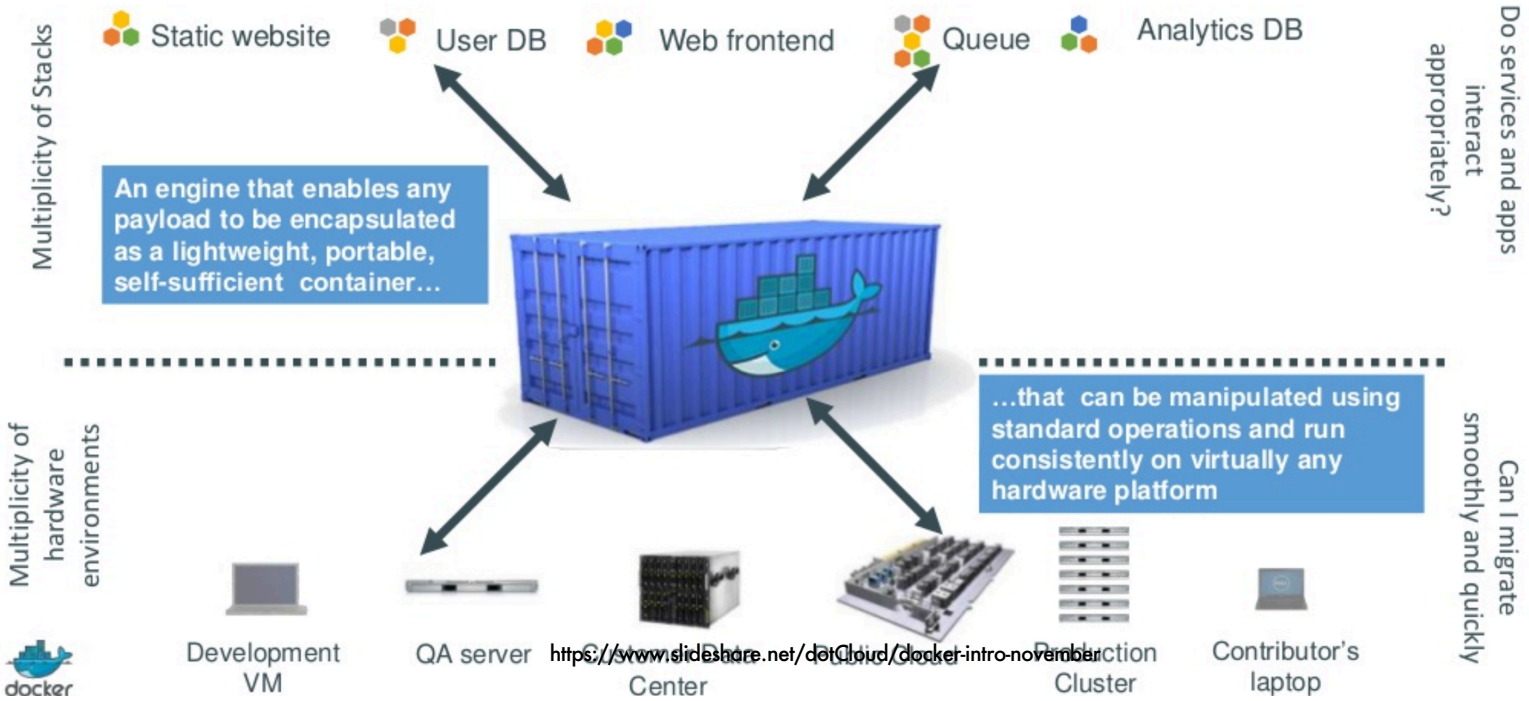
Also a matrix from hell



https://www.slideshare.net/dotCloud/docker-intro-november

Solution: Intermodal Shipping Container

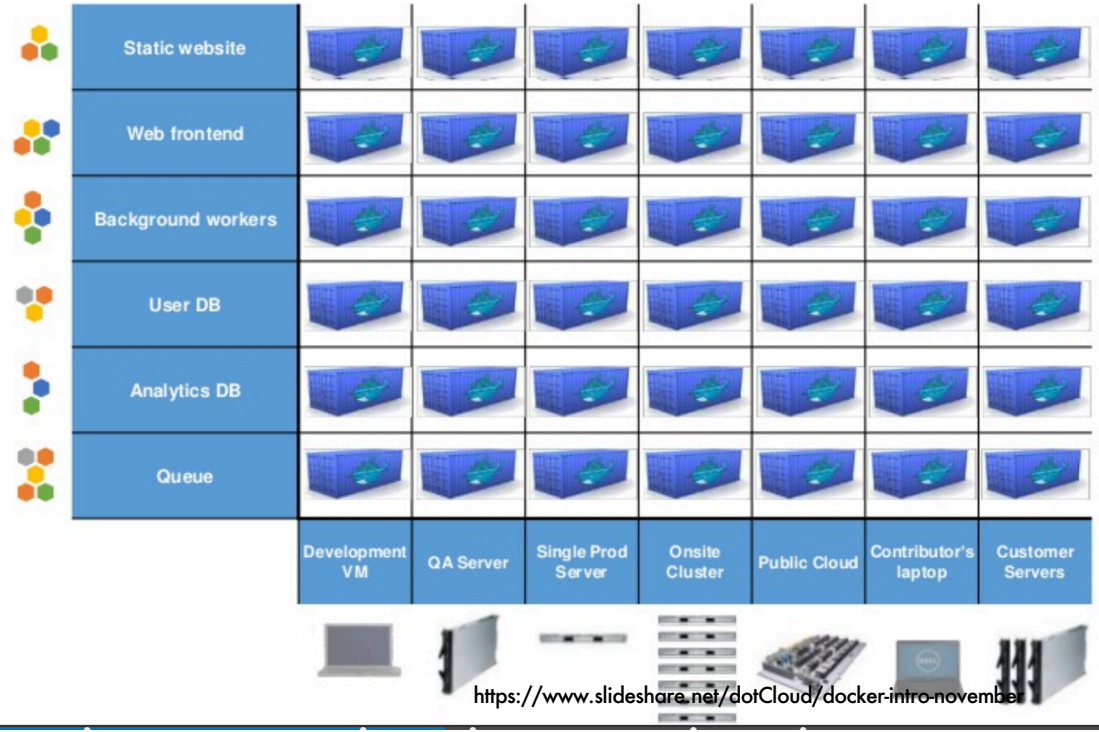
Docker is a shipping container system for code



https://www.slideshare.net/dotCloud/docker-intro-november

bristol.ac.uk

Docker eliminates the matrix from Hell



https://www.slideshare.net/dotCloud/docker-intro-november

What do containers bring?

- Reproducability
- Portability
- Flexibility
- Isolation

- In
  - Development
  - Testing
  - Production

Some Details

Main Components of Container System

- Linux containers – Lxc
  - Cgroups and namespaces (/proc/PID/cgroup /proc/PID/ns)
- Container runtimes
  - Executables that read the container runtime specification, configure the kernel (ns and cgroup),
  - launch the initial process
  - Eg. runc, Kata, gVisor, Nabla, etc.
- Container images
  - Applications
- Container storage
  - Linux storage systems used to store container images on copyon-write (COW) filesystems
  - E.g. OverlayFS, devicemapper, vfs, btrfs, aufs, zfs, etc
- Container registries
  - Web servers used to store container images
  - E.g. Quay.io, Docker.io, Artifactory, Google Container Registry, Amazon Elastic Container Registry, etc.
- Container engines
  - Container tools used to pull images from container registries and assemble them on the host before creating the runtime specification and launching the container runtime
  - E.g. Docker, Podman, CRI-O, Containerd, Rkt, Garden
- Container image builders
  - Tools used to create container images
  - E.g. Buildah, Docker Build, img, Kaniko, orca-build, etc.

Docker System Components

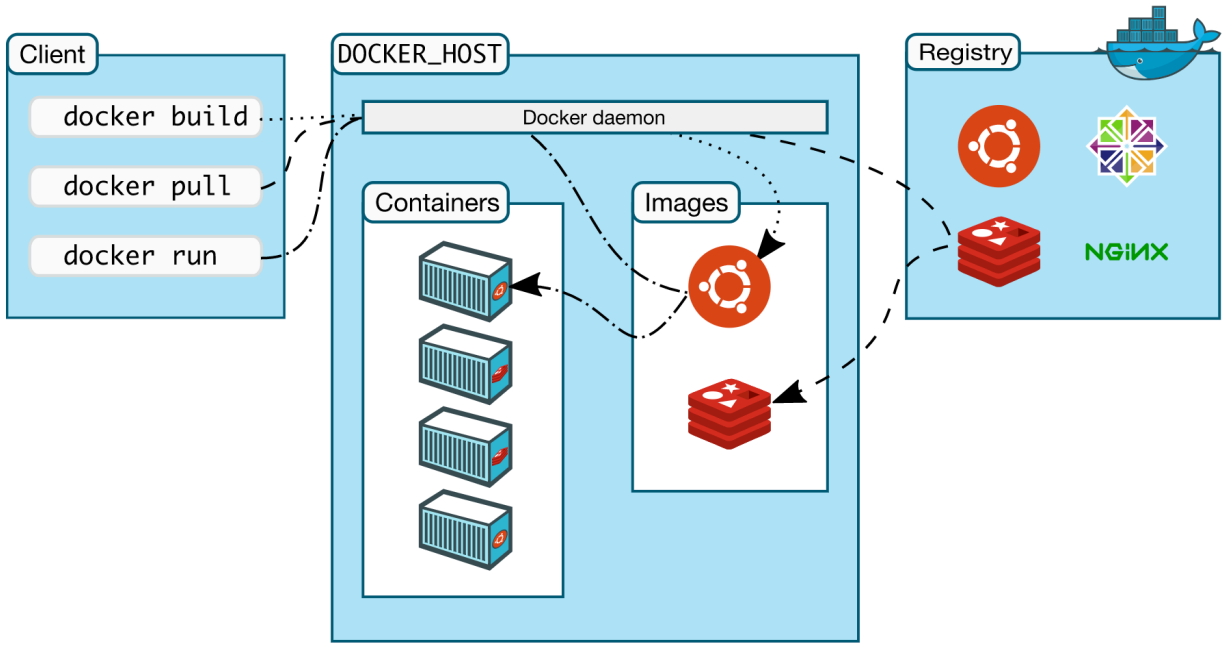
- A container image format spec
- Tools for building container images (Dockerfile/docker build)
- Tools to manage container images (docker images, docker rm , etc.)
- Tools to manage instances of containers (docker ps, docker rm , etc.)
- A system to share container images (docker push/pull)
- Tools to run containers (docker run)
  - container runtimes are responsible for setting up namespaces and cgroups for containers

Open Container Initiative (OCI)

- open governance structure
- creating open industry standards around container formats and runtime
- the Runtime Specification (runtime-spec) and the Image Specification (image-spec)

Docker Architecture

https://docs.docker.con



Docker Objects

- Images
  - read-only template with instructions for creating a Docker container
  - Layered
  - Dockerfile
- Container
  - runnable instance of an image
  - create, start, stop, move, or delete
  - defined by its image as well as any configuration options you provide to it when you create or start it
  - Ephemeral - any changes not stored to a mounted persistent volume will be lost

Linux Containers

- Namespaces
  - Isolation between user processes
  - pid, net, ipc, mnt, utc
- cgroups
  - allocate resources among user-defined groups of tasks (processes)
  - monitor the cgroups you configure, deny cgroups access to certain resources
  - Blkio, cpu, memory, net_cls, net_prio
  - Cgroups are organized hierarchically, like processes, and child cgroups inherit some of the attributes of their parents
- UnionFS
  - Layerd
  - can allow access to merged view of contents of several layers of a volume

Namespaces

- Namespaces let you virtualize system resources, like the file system or networking, for each container
- each kind of namespace applies to a specific resource.
- each namespace creates barriers between processes

- **pid namespace**: Responsible for isolating the process (PID: Process ID).
- **net namespace**: It manages network interfaces (NET: Networking).
- **ipc namespace**: It manages access to IPC resources (IPC: InterProcess Communication).
- **mnt namespace**: Responsible for managing the filesystem mount points (MNT: Mount).
- **uts namespace**: Isolates kernel and version identifiers and hostname (UTS: Unix Timesharing System).

Control Groups

- Cgroups provide a way to limit the amount of resources like CPU and memory that each container can use.
- Each time you create a new container named e.g. “jose”, a cgroup of the same name appears, e.g. in “/cgroup/jose

Container Images

- standard TAR file
- **base image**
  - **Rootfs (container root filesystem):** root (/) of the OS with /usr, /var, /home,
  - **JSON file (container configuration):**
    - **command** or **entrypoint** to run on starts (**PID 1 in ns)**
    - **environment variables** to set for the container
    - the container's **working directory**
- **Layered with differences**

Copy on Write File Systems

- Eg. OverlayFS, btrfs,
- Data is not overwritten but held separately
- Better recovery
- Built-in transactions
- Snapshots - virtual copy, diff to snapshot all subsequently written blocks
- Multiple containers can share same base images

Performance

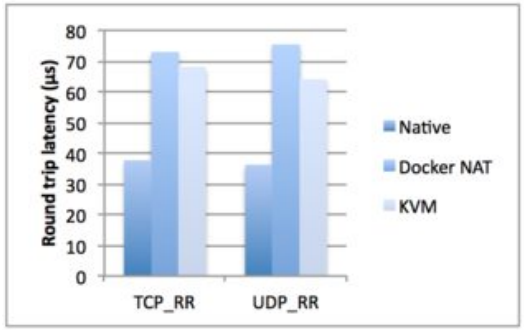


Fig. 3. Network round-trip latency (μs).

An Updated Performance Comparison of Virtual Machines and Linux Containers

Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio
IBM Research, Austin, TX
{wmf, apferrei, rajamony, rubioj}@us.ibm.com

***Abstract*—Cloud computing makes extensive use of virtual machines (VMs) because they permit workloads to be isolated from one another and for the resource usage to be somewhat controlled. However, the extra levels of abstraction involved in virtualization reduce workload performance, which is passed on to customers as worse price/performance. Newer advances in container-based virtualization simplifies the deployment of applications while continuing to permit control of the resources allocated to different applications.**

In this paper, we explore the performance of traditional virtual machine deployments, and contrast them with the use of Linux containers. We use a suite of workloads that stress CPU, memory, storage, and networking resources. We use KVM as a representative hypervisor and Docker as a container manager. Our results show that containers result in equal or better performance than VMs in almost all cases. Both VMs and containers require tuning to support I/O-intensive applications. We also discuss the implications of our performance results for future cloud architectures.

Within the last two years, Docker [45] has emerged as a standard runtime, image format, and build system for Linux containers.

This paper looks at two different ways of achieving resource control today, viz., containers and virtual machines and compares the performance of a set of workloads in both environments to that of natively executing the workload on hardware. In addition to a set of benchmarks that stress different aspects such as compute, memory bandwidth, memory latency, network bandwidth, and I/O bandwidth, we also explore the performance of two real applications, viz., Redis and MySQL on the different environments.

Our goal is to isolate and understand the overhead introduced by virtual machines (specifically KVM) and containers (specifically Docker) relative to non-virtualized Linux. We expect other hypervisors such as Xen, VMware ESX, and Microsoft Hyper-V to provide similar performance to KVM

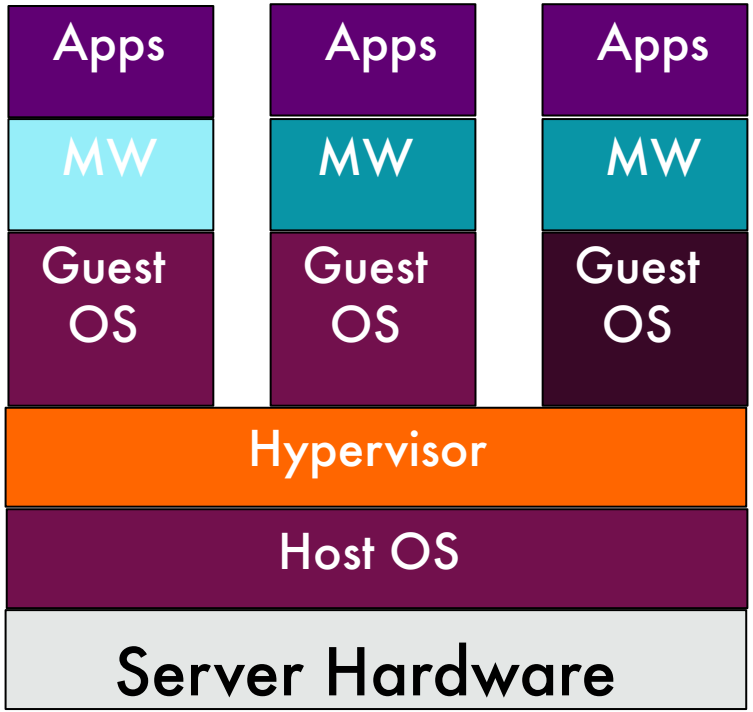
https://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf

- Network isolation in containers takes a hit,
- Disk IO is almost native,
- Aggregate performance is much better than VM

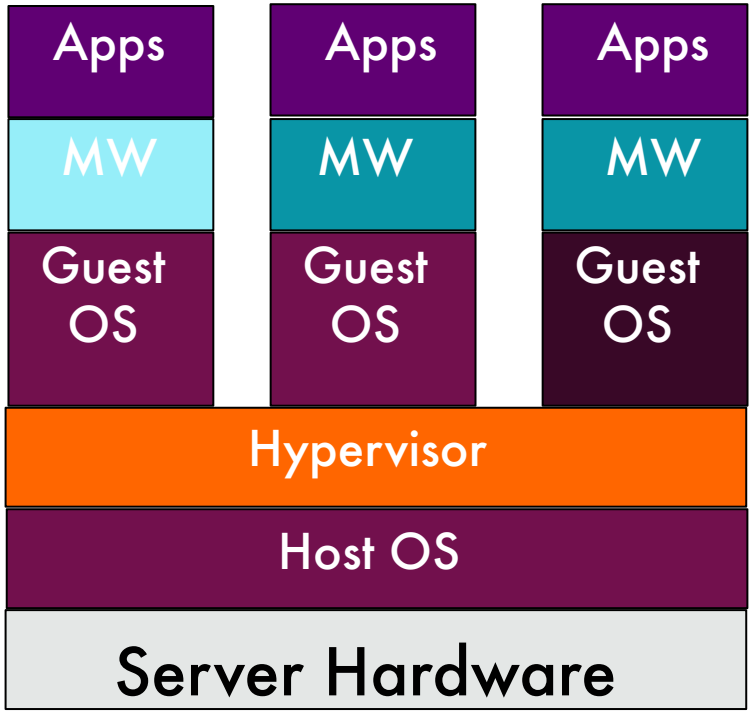
Virtualisation compared to Containerisation

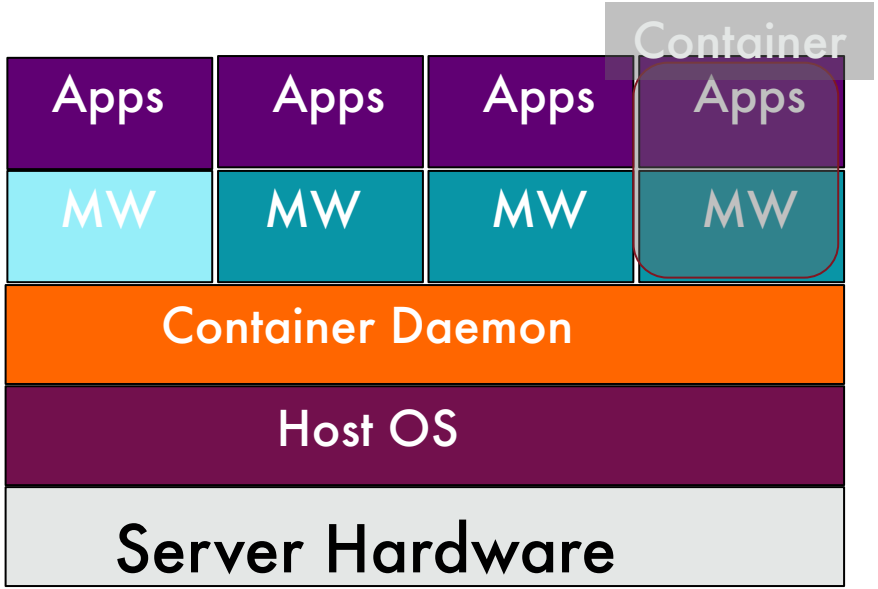
- Similarities
  - Both emulate compute infrastructure
  - Both encapsulate the tenant
- Differences
  - VMs provide "hardware level virtualisation" vs containers 'operating system level virtualisation"
  - VMs need 10s of seconds for provisioning vs containers need milliseconds provisioning
  - Virtualisation performance is slower than containers in most dimensions except for networking
  - VM tenants are fully isolated vs containers provide process level isolation to tenants

Containerization: Docker

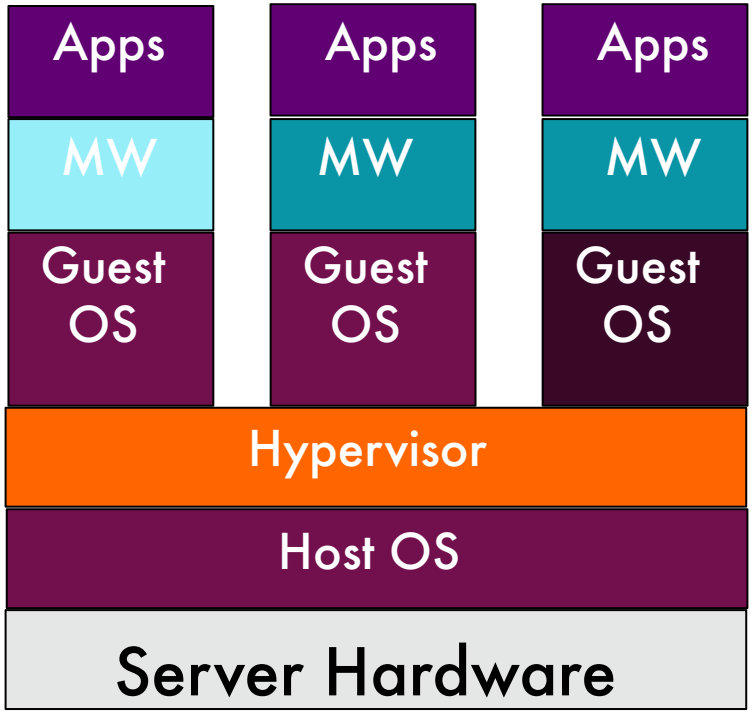


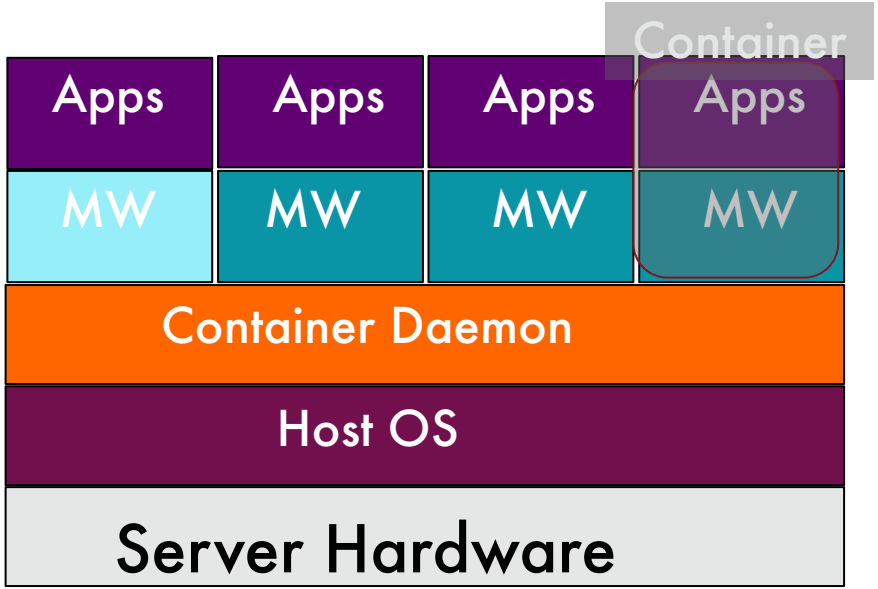
Containerization: Docker





Containerization: Docker

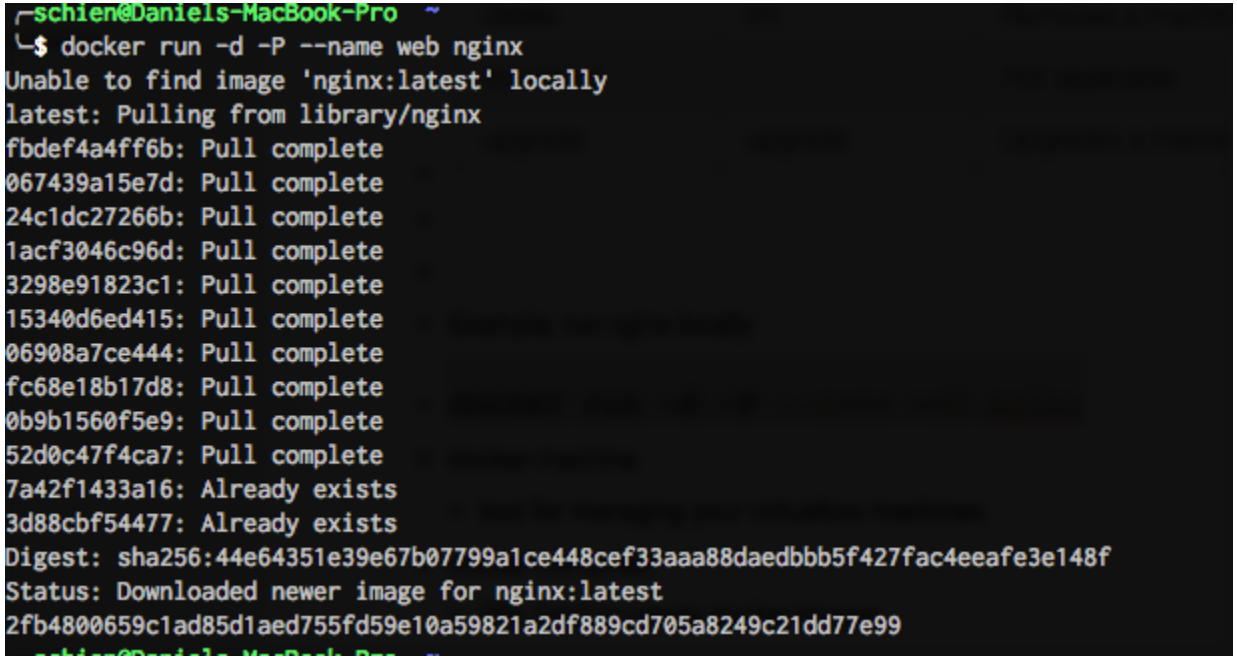




Containers vs Images

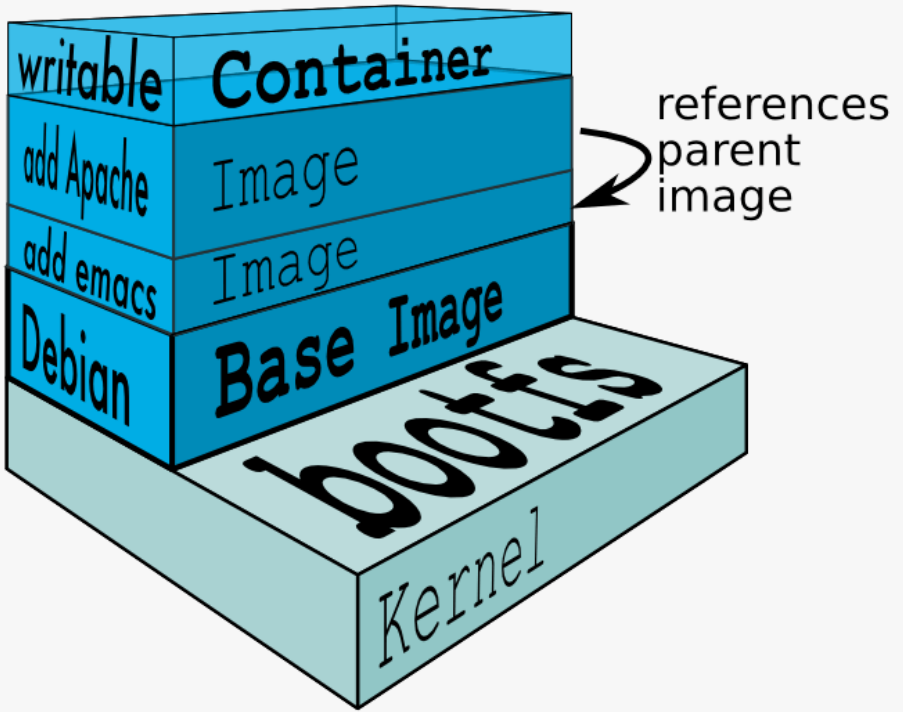
- Containers are *running images*
- Deploy public images on docker hub
- You create a local repository of these images on your dev machine
- All cloud operators provide container service for your images
- Can deploy private image repository (“container registry”)

Images are composed from layers



Like ‘diff’ only the changes from one layer to the next are stored in an image

Anatomy of Image



Building Images

- Each instruction (line) defines an image with a unique checksum
- Images are layered
- When composing new images existing images are re-used
- Compiler inspects each instruction and checks if the image exists
- For ADD and COPY instructions the file timestamps are compared with those in the cached images

Dockerfiles

- Define containers as a sequence of instructions
- Build ar

```
FROM python:3.5.1

RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y libmemcached-dev

ENV PYTHONUNBUFFERED 1

ADD requirements/ /app/
WORKDIR /app/
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

EXPOSE 8000
EXPOSE 8888
```

bristol.a

Example: run nginx on OSX

- Default example on the docker website
  - https://docs.docker.com/installation/mac/
- Nginx is a HTTP server (like Apache)
- `docker run -d -P --name web nginx`

- *1. pull image 'nginx' from docker hub*
- *2. start container from image, name 'web'*
- *3. forward all ports (-P)*
- *4. run as daemon (-d)*

Further Reading

- https://opensource.com/article/18/8/sysadmins-guide-containers
- https://www.ianlewis.org/en/container-runtimes-part-1-introduction-container-r
- https://web.archive.org/web/20150310145336/http://blog.dotcloud.com/under-the-hood-linux-kernels-on-dotcloud-part
- https://www.youtube.com/watch?v=cJp86kGOAQg

Review

- VMs provide access to infrastructure in the Cloud, on demand, self-serve, metered
- Hypervisor isolates guests from hardware
- Different Virtualisation approaches
- Xen, KVM