THE UNIVERSITY *of* EDINBURGH
informatics

Cluster Resource Management

Luo Mai
University of Edinburgh

Why do we need resource management?

Diverse applications:
- Web service
- Data processing
- Machine learning
- …

Small Application 1

Mediuem Application 2

....

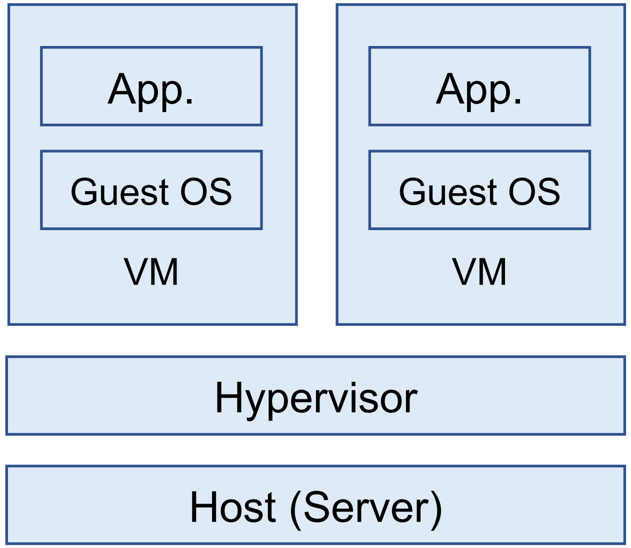
Large Application 3

"One size does not fit all"
- Virtualisation!

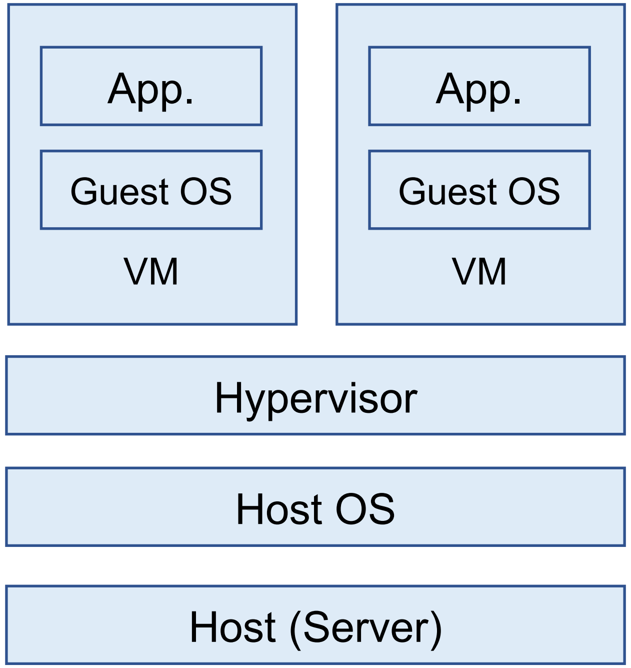
Servers in clusters:
- Homogeneous server configuration
- A server can have affluent resources (e.g., 256 CPUs, 8 GPUs, 1TB memory, 8TB SSD)

Virtual Machines

A **Virtual Machine (VM)** is an emulation of a physical computer. A **hypervisor** is a type of computer software, firmware or hardware that creates and runs VMs.



Type 1 - Native



Type 2 - Hosted

Type 1 hypervisors (Native)

- KVM, VMWare vSphere, …

Type 2 hypervisors (Hosted)

- Virtualbox, VMWare Fusion

Type 1 vs. Type 2

- Cost / Scale
- Portability

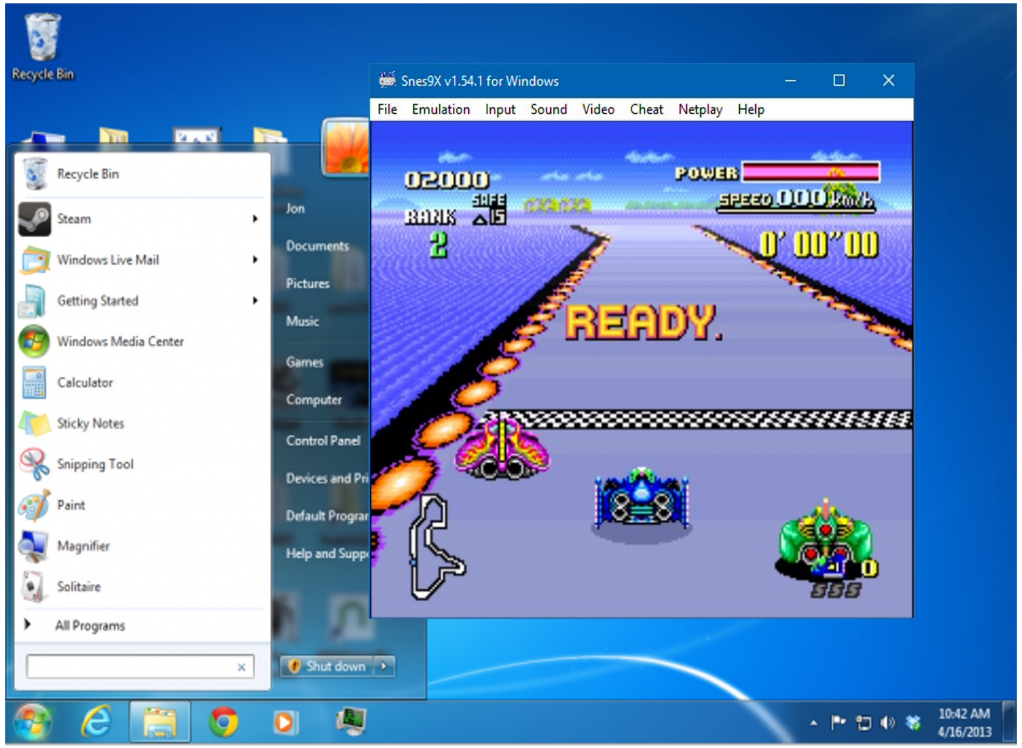
Types of Virtualisation

Different types of virtualisation shift the focus on different properties such as **execution speed**, **flexibility** and **security**.

- Software Emulation (e.g., QEMU)
- Hardware Virtualisation (e.g., KVM)
- Paravirtualisation (e.g., Xen [1] - Optional reading)

[1] Xen and the Art of Virtualization, SOSP 2003

Software Emulation

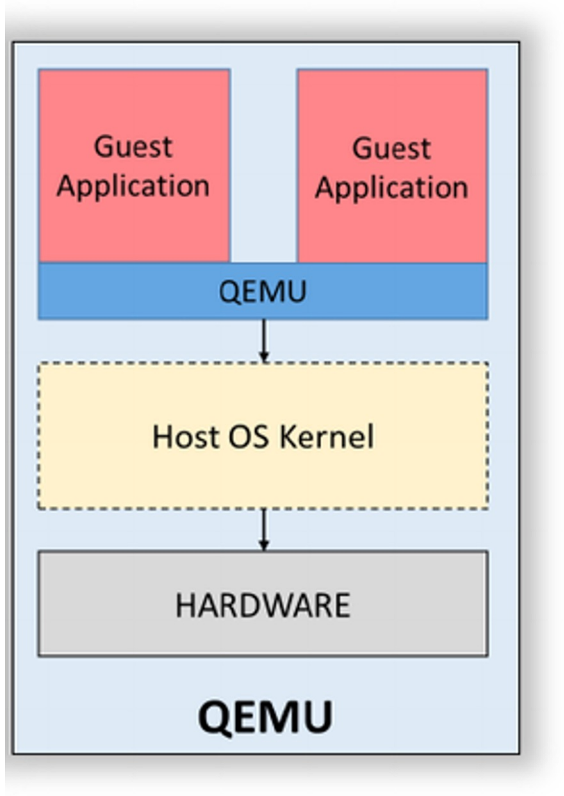


Execution of each guest instruction is emulated in software

- **highly flexible**, e.g. cross-architecture simulation
- **slow to run** - high overhead

speedup can be achieved using binary translation

Software Emulation

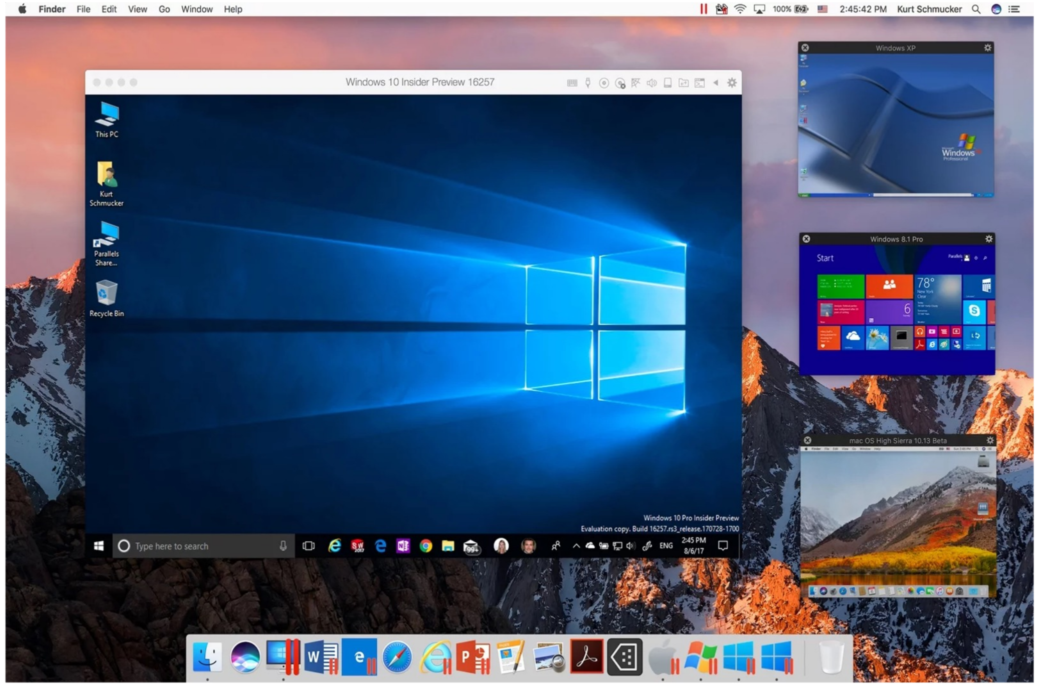


Qemu is a hosted hypervisor

- emulates hardware
- uses binary translation to speed up execution
- allows cross-architecture virtualisation supporting many architecture models

THE UNIVERSITY of EDINBURGH
informatics

Full System or Hardware Virtualisation



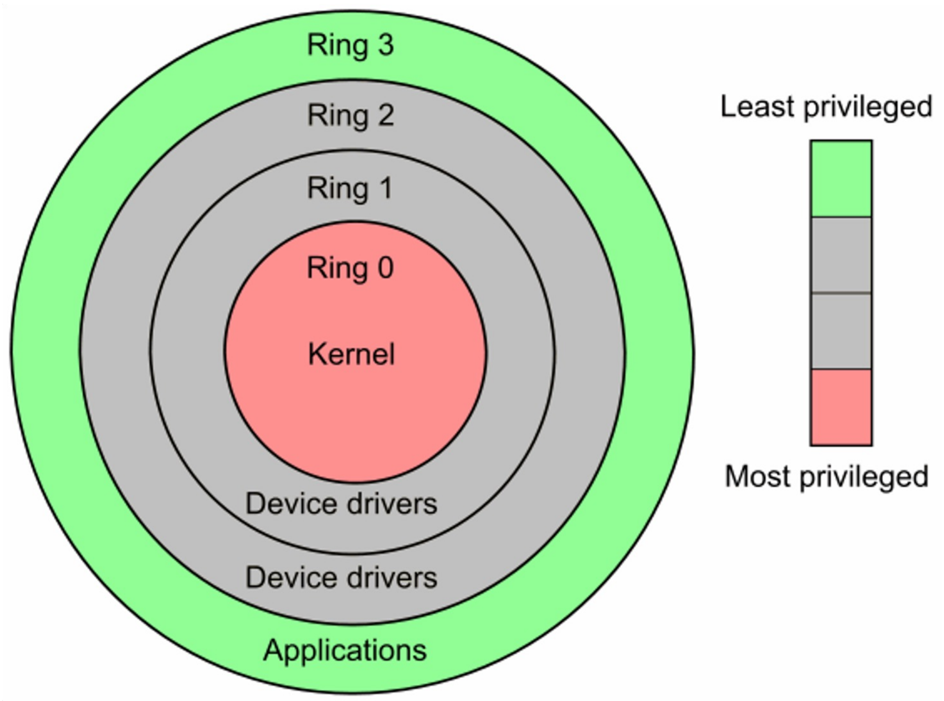
Guest OS instructions can be executed natively on the host CPU

- **Near native speed**
- **Less flexible** - same architecture simulation

How can you make sure a guest OS does not disturb the underlying host when directly running on Hardware?

THE UNIVERSITY of EDINBURGH
informatics

Kernel Mode vs. User Mode in Guest OS



OS Privilege Levels

Kernel Mode

- Code has unrestricted access to hardware
- Reserved for the lowest level, most trusted functions of OS

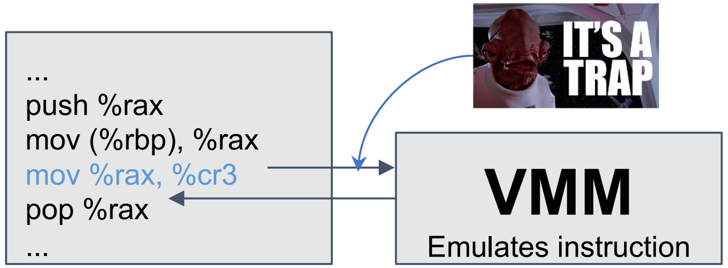
User Mode

- Code has no ability to directly access hardware or reference memory

How to handle privileged instructions?

Trap-and-Emulate

- The guest operating system runs “de-privileged”, all non-privileged instructions execute natively on the host.
- All privileged instructions trap to the Virtual Machine Manager (VMM) which implements the ”Hypervisor”
- VMM emulates these privileged operations.
- Guest resumes execution after emulation.

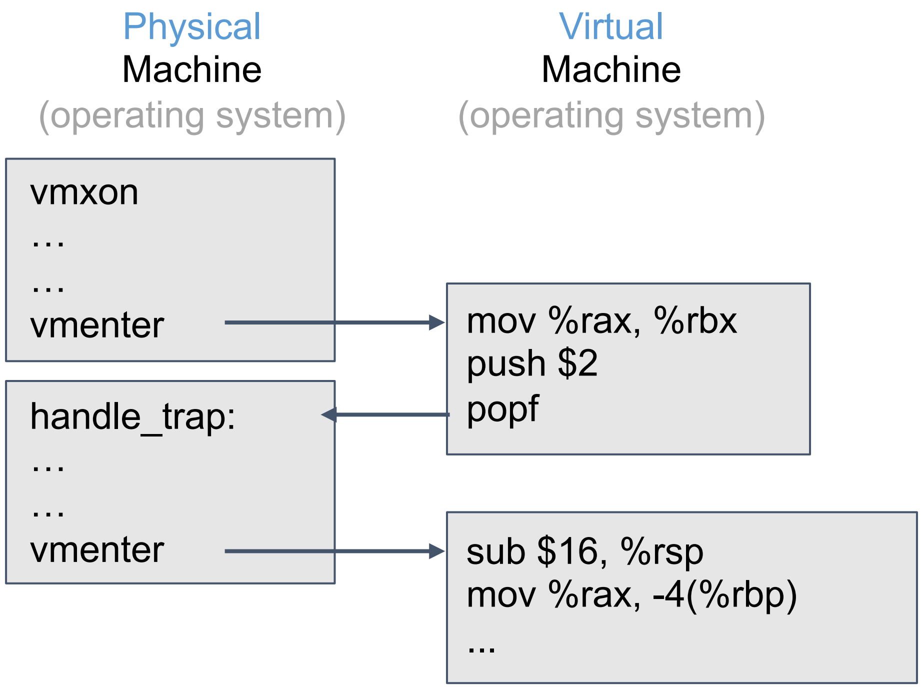


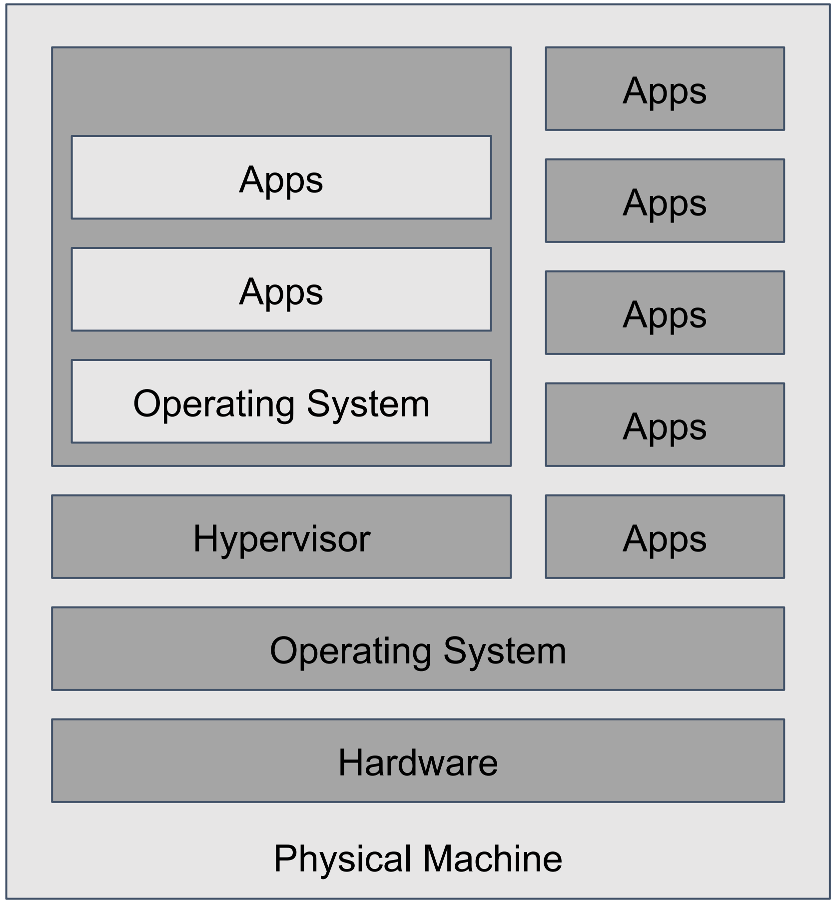
Problem: Not all privileged x86 instructions trap properly!

Virtualising x86

- Originally x86 was not "classically" virtualisable.
  - Some privileged instructions did not "trap", and so could not be emulated correctly.
- Interpretation is too slow
- Code Patching leaves traces of virtualisation
- Binary Translation is better but still incurs overhead.
- Since 2005, x86 processors now support virtualisation in hardware.
  - Intel-VT
  - AMD-V
- This enables trap-and-emulate style virtualisation.
- Unmodified operating systems can run natively on host machines.

Virtualising x86 on modern hardware





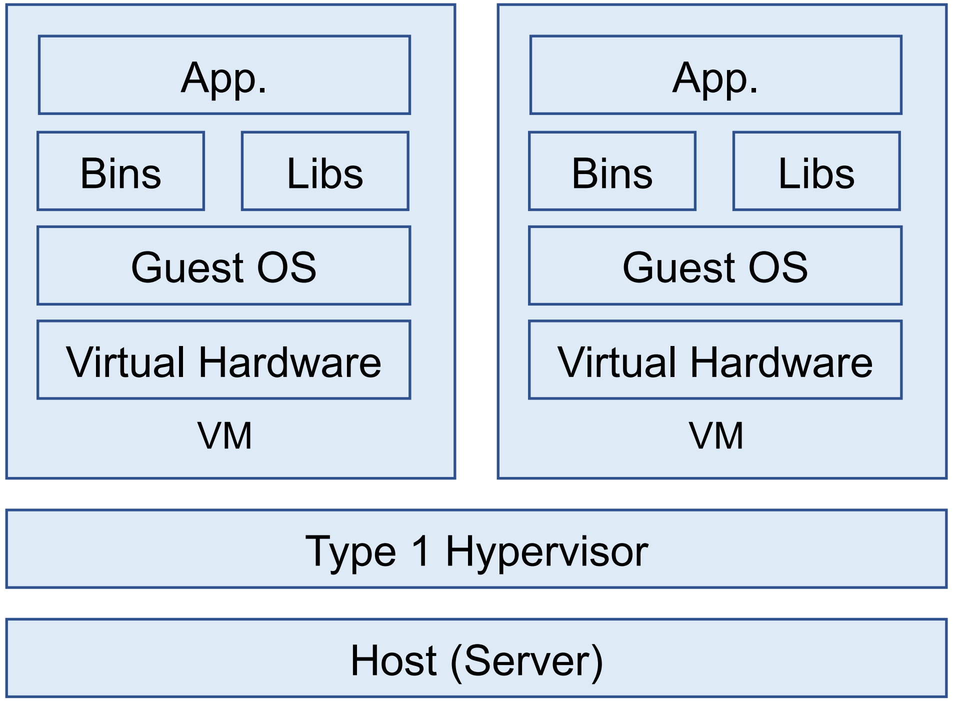
Benefits and Pricing Models of Cloud VMs

Benefits

- Cost savings
- Flexibility + Speed
- Lower downtime
- Security

Pricing models

- Pay-as-you-go
- Spot/Transient instances
- Reserved instances

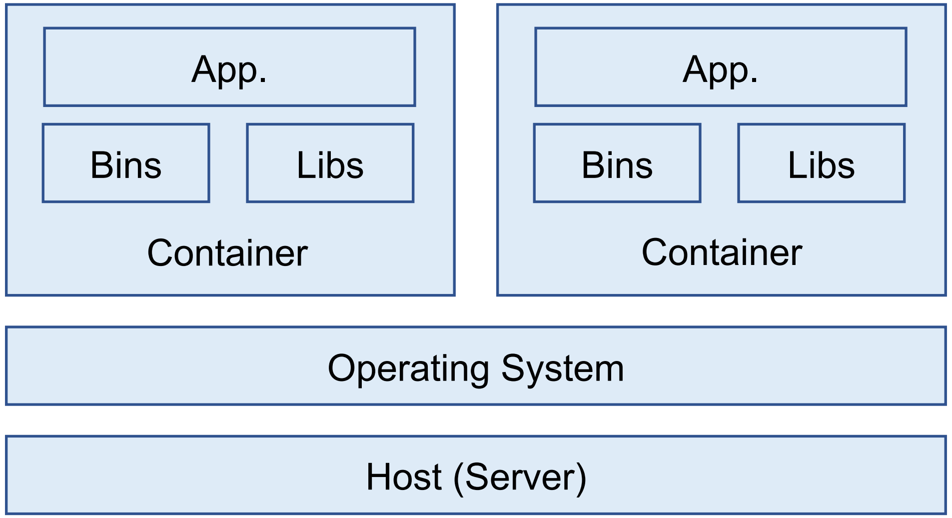


Questions?

THE UNIVERSITY of EDINBURGH informatics

Containers – OS-Level Virtualisation

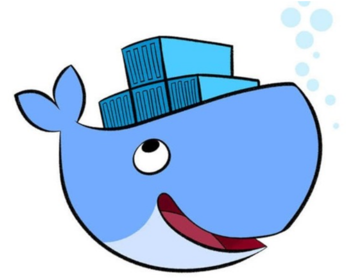
Containers are a lighter-weight, more agile way of handling virtualization — since they don't use a hypervisor, you can enjoy faster resource provisioning and speedier availability of new applications.



How can containerisation be achieved?

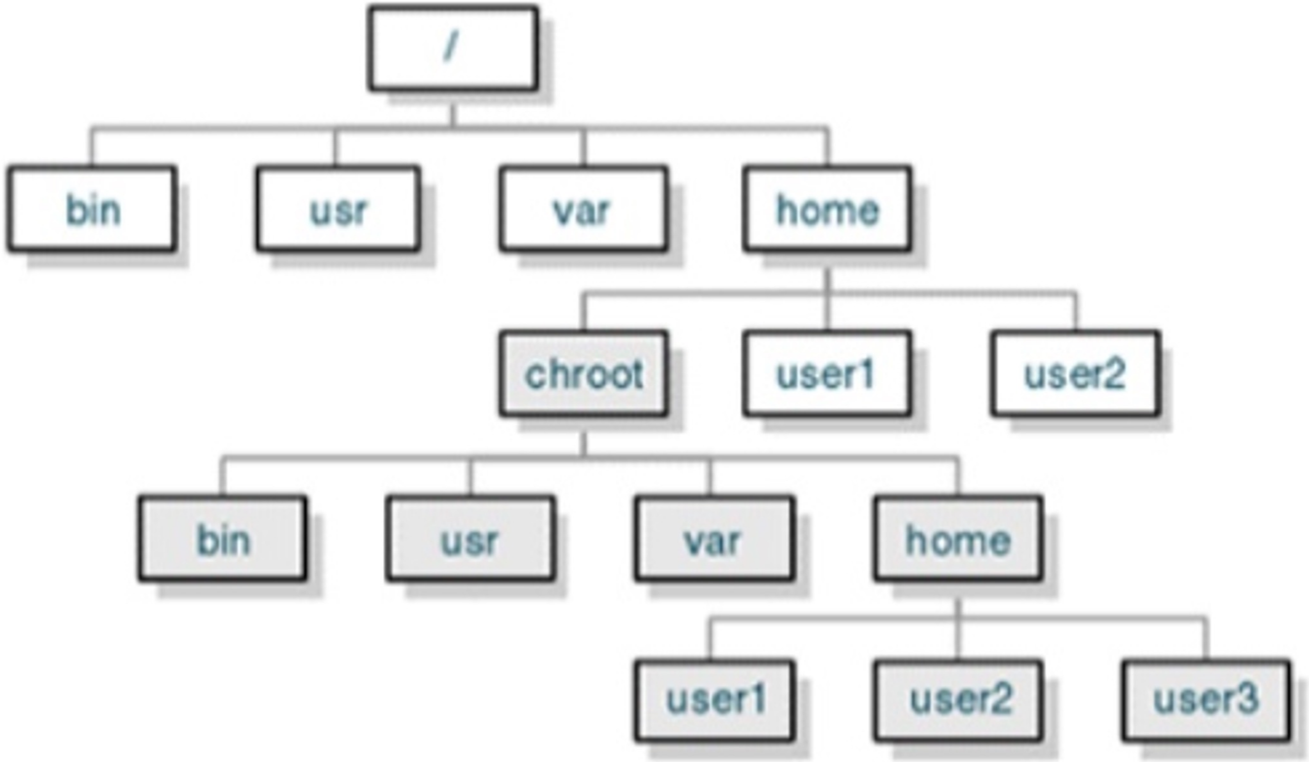
- provide user space abstraction for each container
  - isolated view at the system for container content

- provide a container management system to manage container instances and standardised access to contents



Chroot

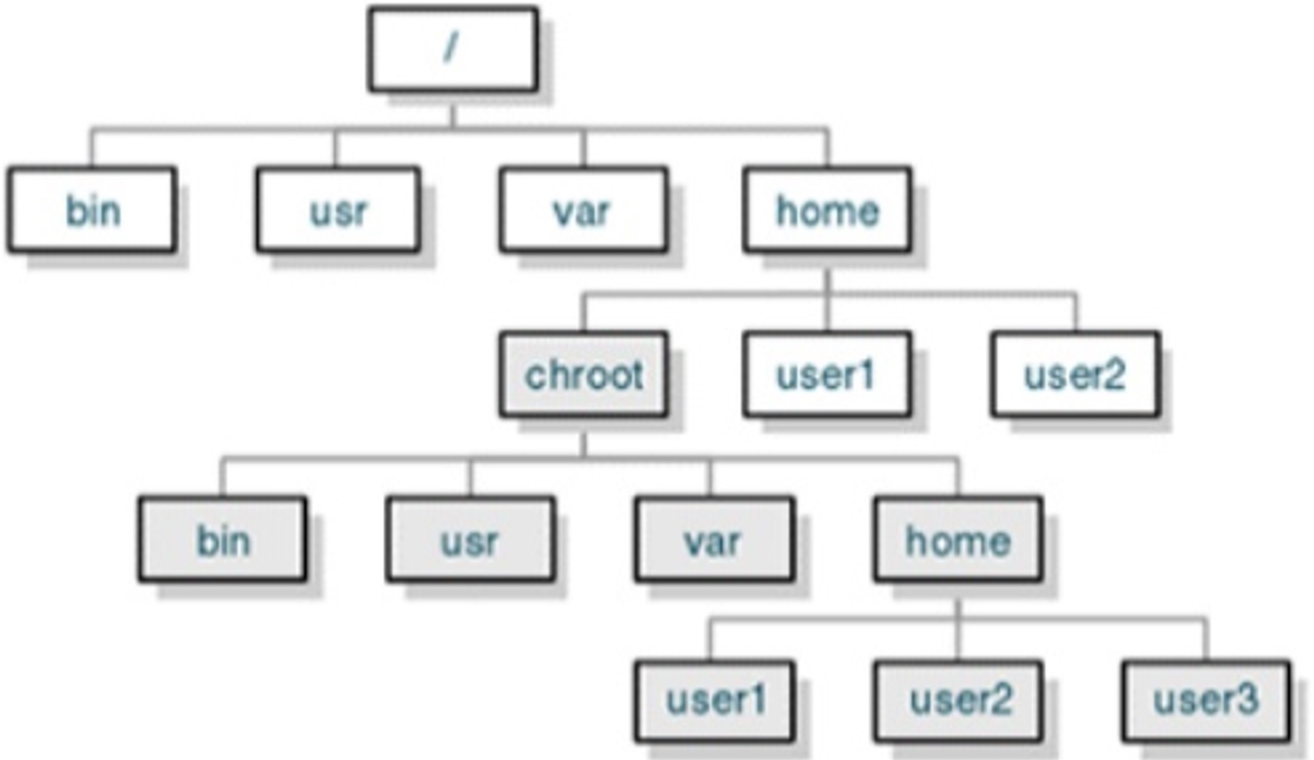
- Linux processes have a root directory
- chroot changes the root directory for a new process and its children
- such a jailed process cannot access files outside its root directory structure



Chroot

Limitations

- you can break out of a chroot jail with root privileges
- no resource limits
  - memory, cpu
- no isolation
  - network, devices, processes



Not a secure sandbox!

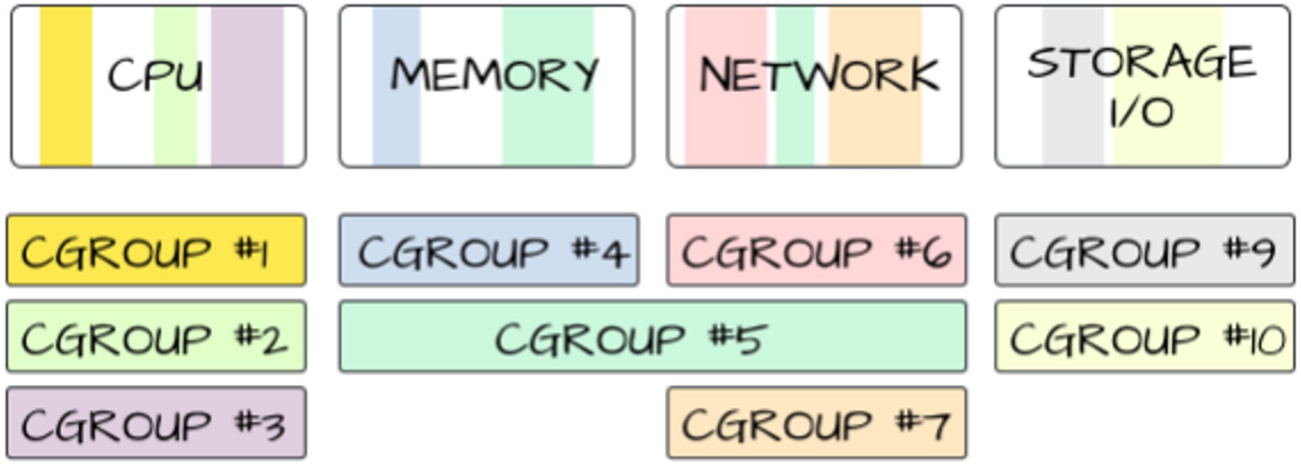
How to limit resources and achieve isolation?

- **control groups** - limit what you can use
  - resource control
- **namespaces** - limit what you can see
  - isolated view at system

Resource control through cgroups

Linux kernel feature to limit account and isolate resource usage for groups of processes

- cpu
- memory
- disk I/O
- devices
- network
- etc. ...

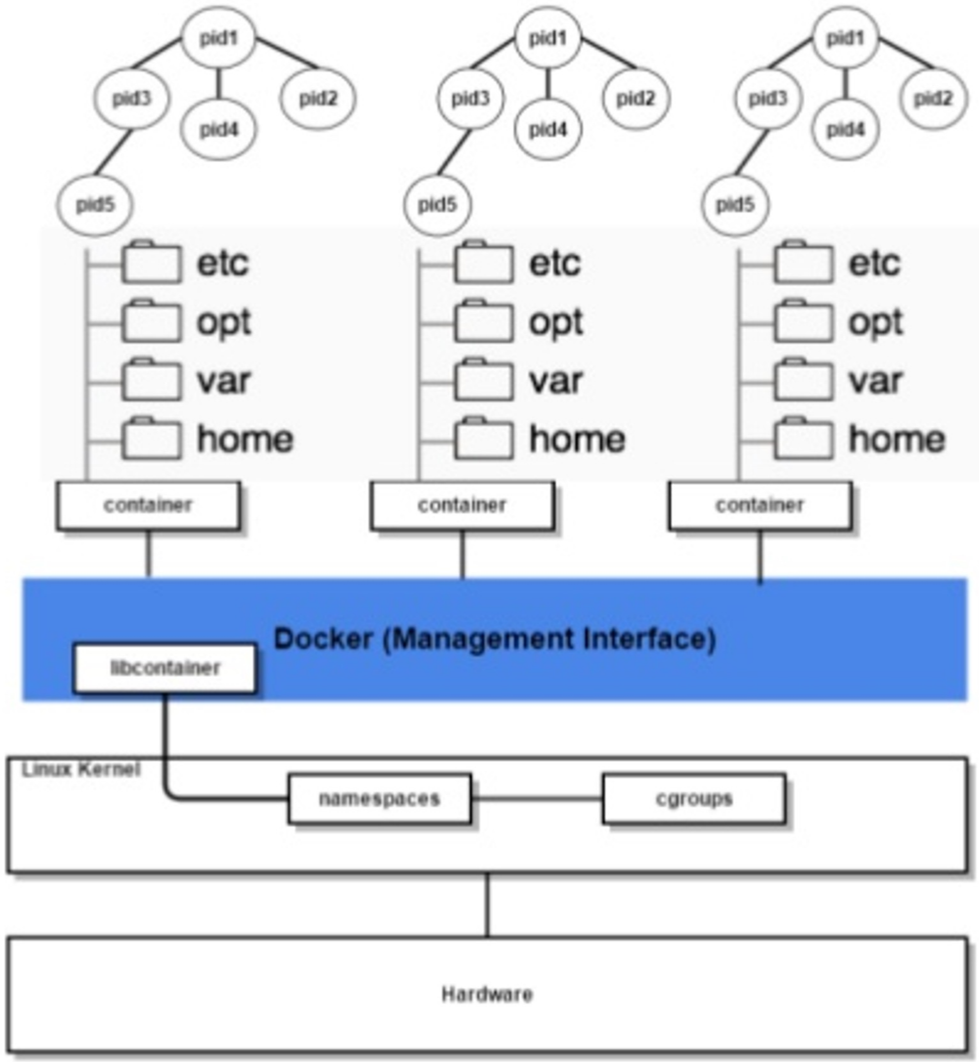


Isolation through namespaces

Namespaces provide containers with their own view of the underlying Linux system.

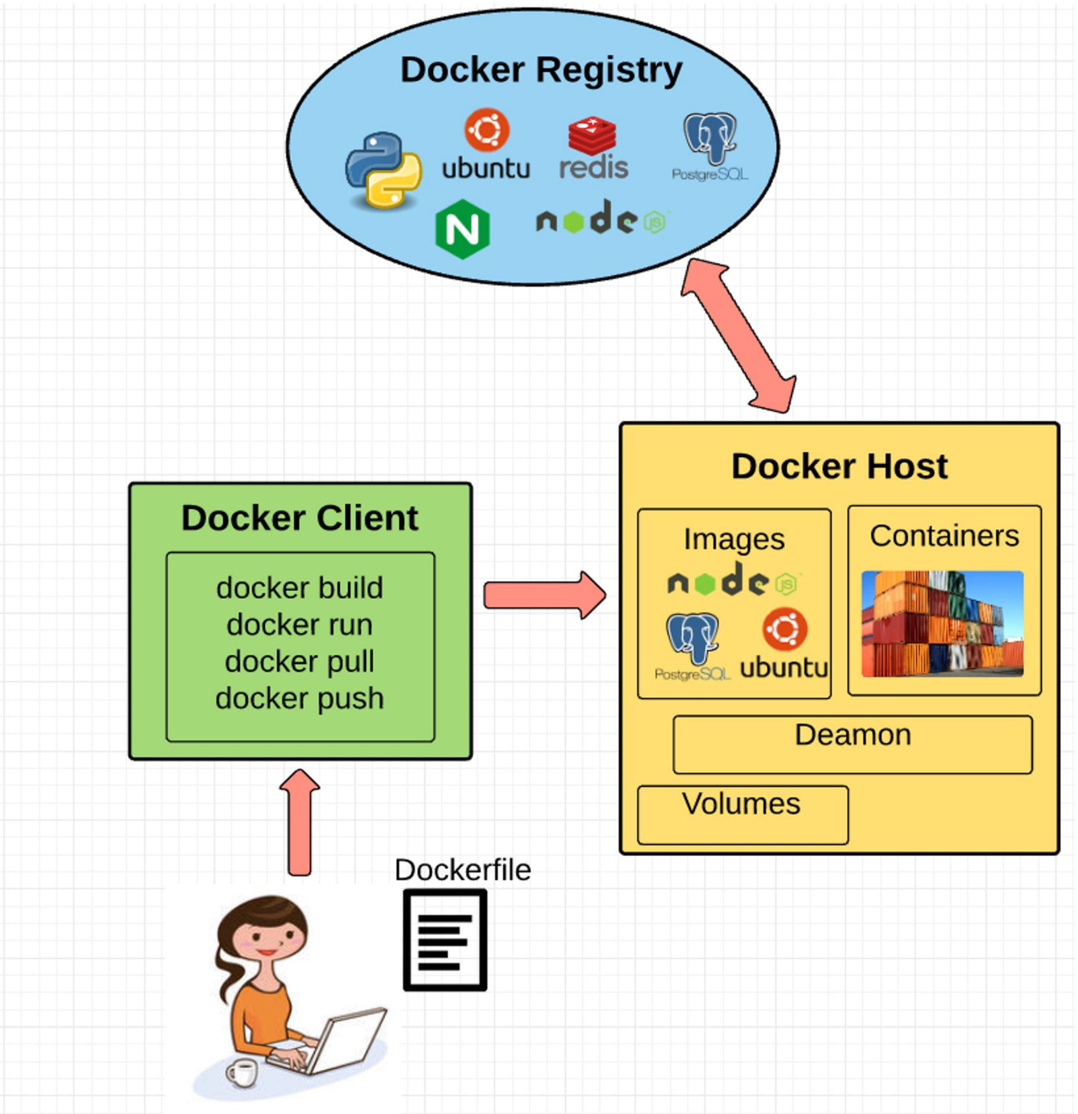
- **NET**: IP addresses, IP routing tables, port numbers
- **PID**: process IDs
- **MNT**: system mounts
- **UTS**: host name
- **IPC**: inter process communication resources
- **USER**: user ids

Docker Container under the hood



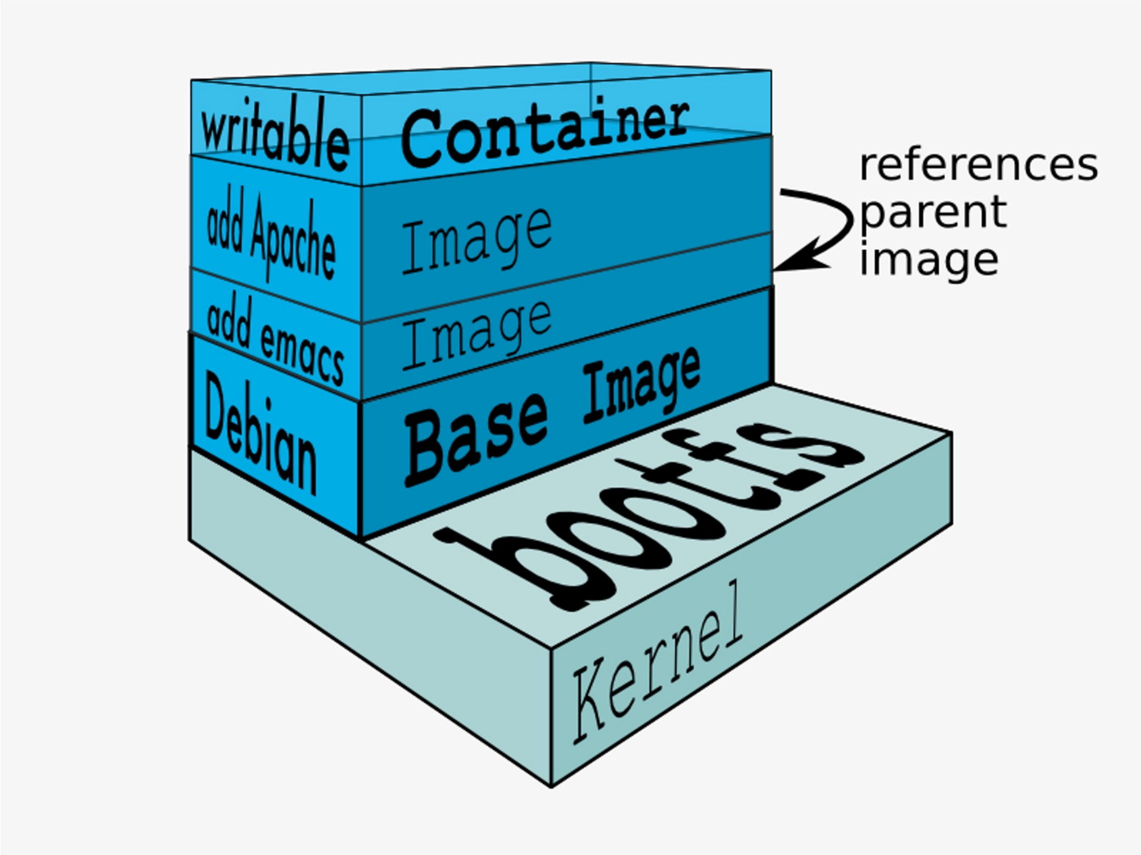
The Docker Eco-System

- **Docker Engine**: docker runtimer containing: Deamon, Client and API for remote access
- **Dockerfile**: contains instructions to build a docker image
- **Image:** layered read-only file system described by dockerfile
- **Volumes:** shared "data" part of a container
- **Container:** wraps application code and dependencies as described in image
- **Docker Registry:** server side app to share and distribute images



Docker Images

- Docker file describes docker image
- start with a base image
- layer dependencies on top
- Union file system

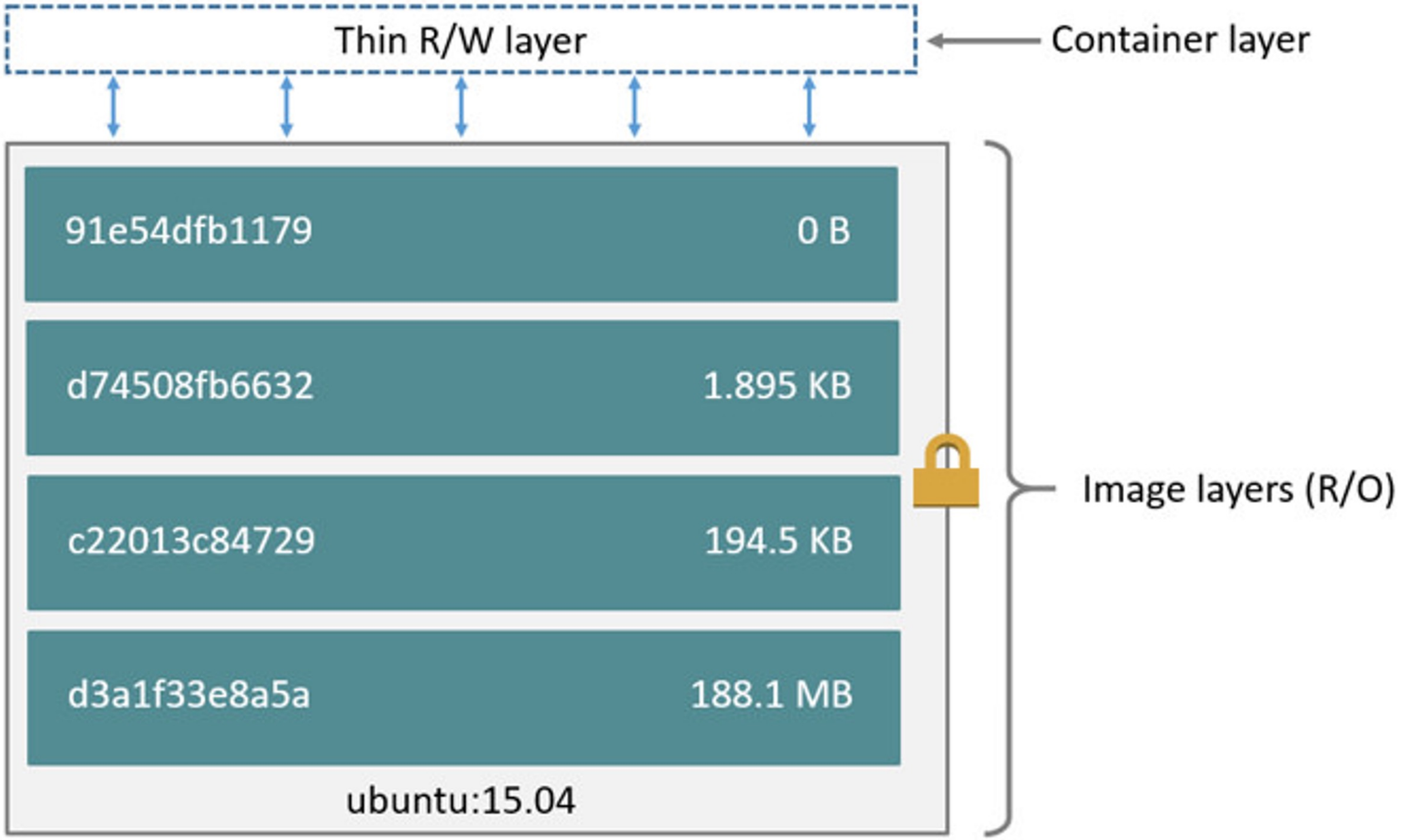


THE UNIVERSITY of EDINBURGH
informatics

Docker Images

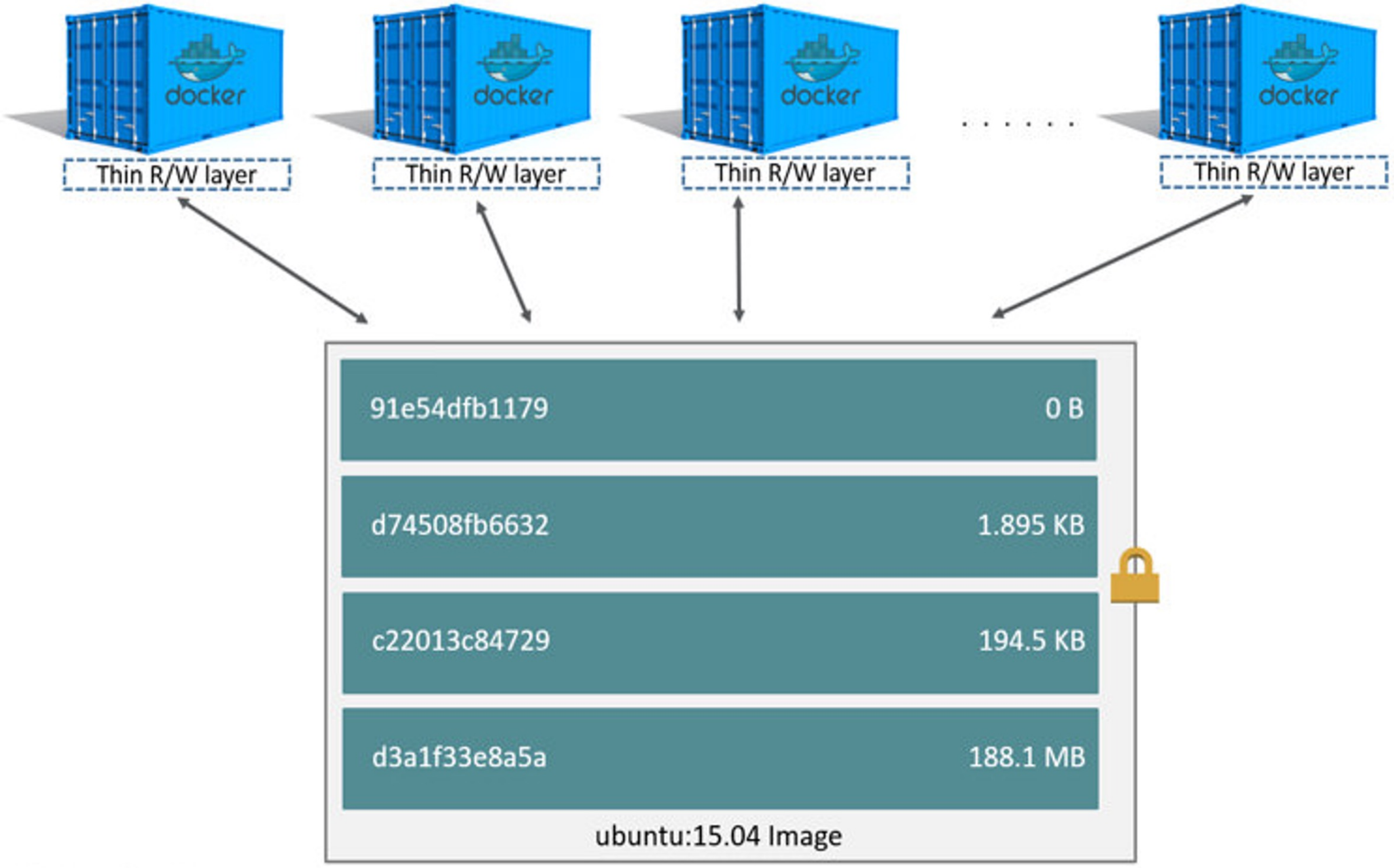
Docker File

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Container
(based on ubuntu:15.04 image)

Docker Images



VMs vs. Containers

VM

- Heavy-weight in terms of layers of system software (GBs in size)
- Take minutes to start
- Allows multiple OS to execute concurrently
- Provide a high level of isolation
  - Fault
  - Resource

Containers

- Light-weight (MBs in size)
- Take seconds to start
- Share a common OS and kernel
- Don't offer the same level of isolation
  - A kernel crash caused by one guest will affect everyone else
- The interface offered is at the level of system calls and ABI – much more prone to security problems than the API exposed by the Hypervisor

How to choose virtualisation?

	Without Container	With Conatiner
Without VM	Bare-mantal	Contrainer
With VM	VM	Container in VM

Considerations

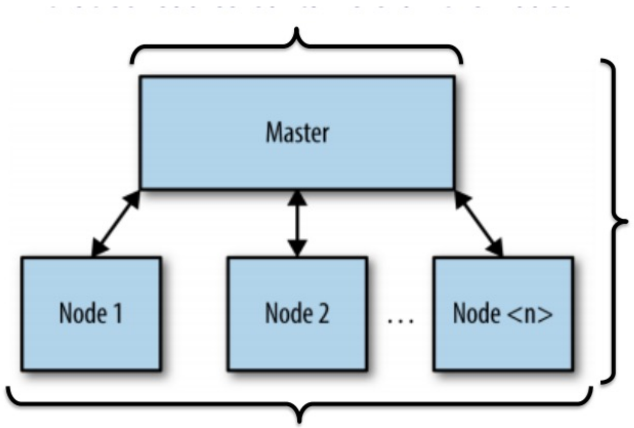
- Virtualisation costs
  - Some hardware does not support virtualisation
  - Virtualisation is not free
- Scale
  - What is the best way to communicate? NVLink, shared memory, socket, REST
- Security
  - What if your container or VM is compromised?
- Isolation
  - Multi-user vs. Single-user with multi-job
- Flexibility
  - Multi-OS vs. Single-OS

Questions?

Kubernetes - Container Orchestration

Kubernetes — also known as “k8s” — is a container orchestration platform for scheduling and automating the deployment, management, and scaling of containerized applications.

- Masters run special coordinating software that schedules containers on the nodes.

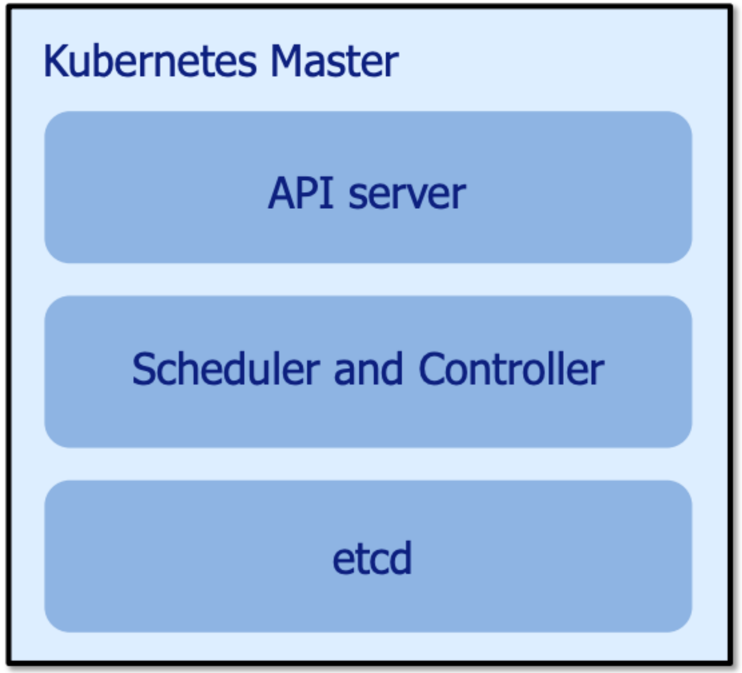


A bunch of machines sit networked together in many data centres.

The collection of masters and nodes is known as a cluster.

- Worker machines are called nodes.
- Each machine hosts 1+ Docker container.

The Master

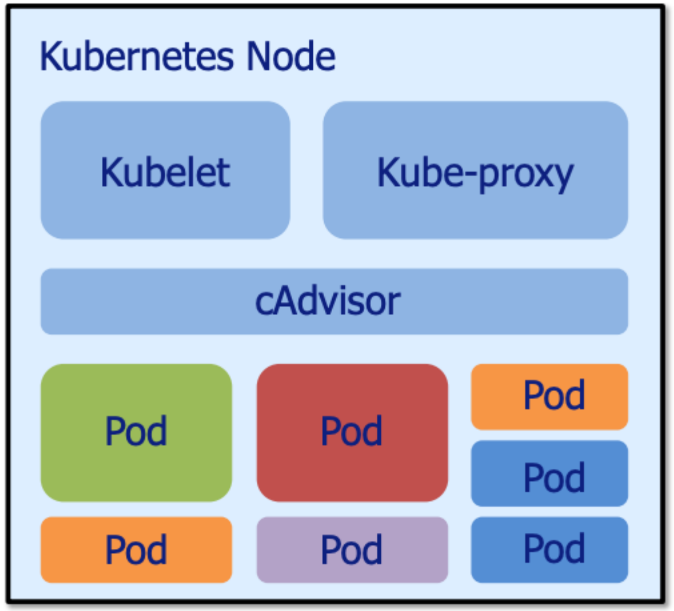


1. **API server**
    1. Nearly all components of the master and nodes accomplish their tasks by making API calls.
    2. These calls are handled by the API server running on the master.

2. **Scheduler and Controller Manager**
    1. Processes that schedule containers (i.e., pods) onto target nodes.
    2. Make sure that the correct number of these things are always running.

3. **etcd – distributed reliable key-value store**
    1. Responsible to keep and replicate the current configuration and run state of the cluster.
    2. Implemented as light-weight distributed KV store.

The nodes

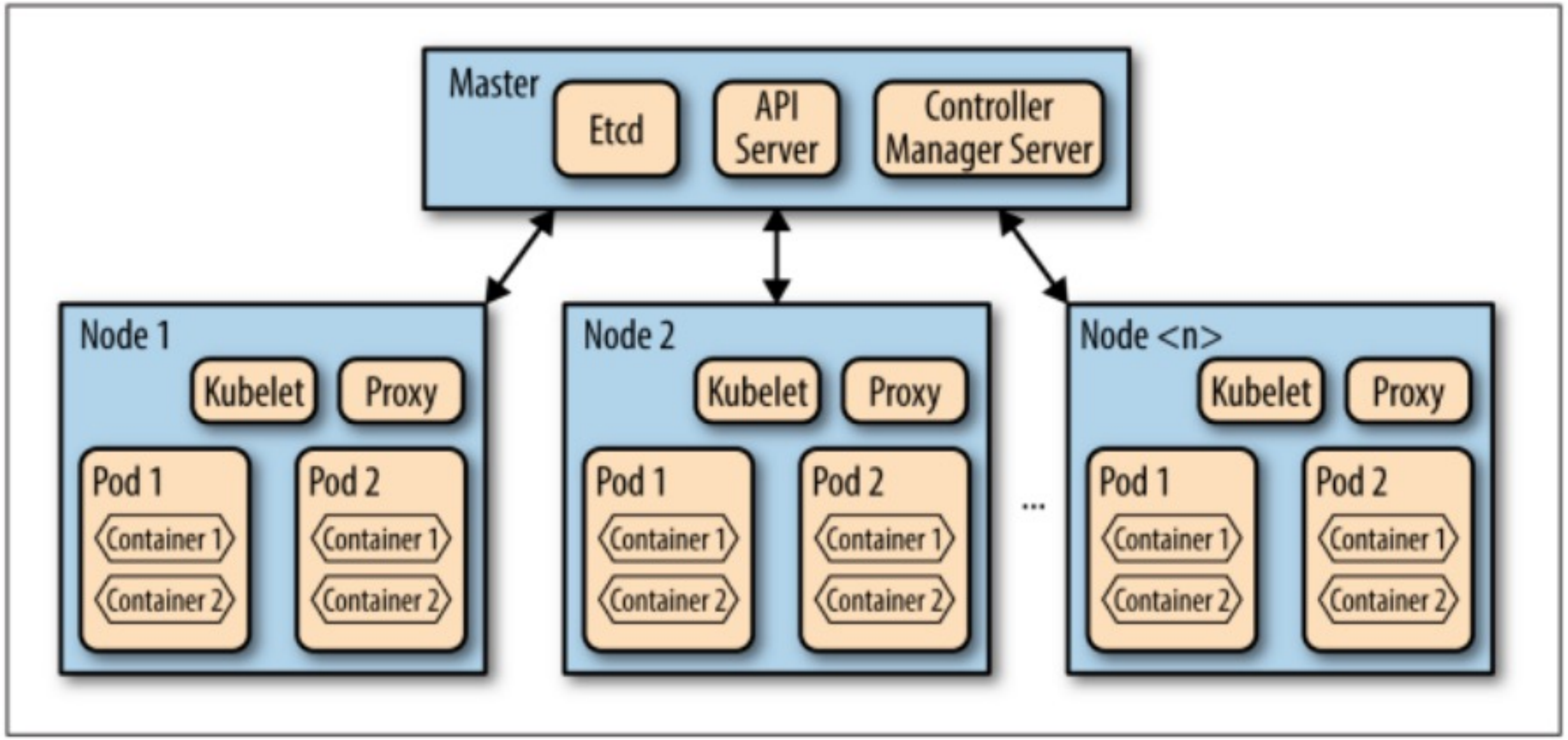


1. **Kubelet**
   1. special background process (daemon)
   2. execute commands from the master to create, destroy, and monitor containers on that host.

2. **Kube-proxy**
   1. simple network proxy to separate the IP address of target container from the name of the service it provides.

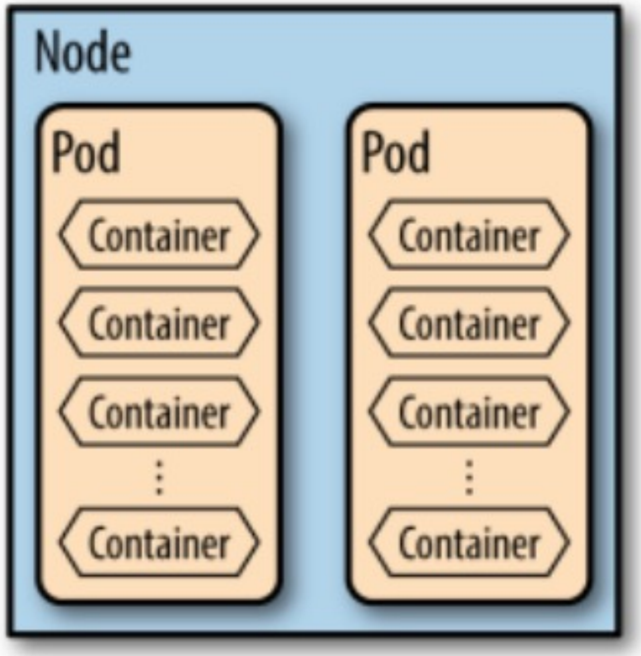
3. **cAdvisor (optional)**
   1. is a special daemon that collects, aggregates, processes, and exports information about the running containers.

Full picture of a Kubenetes cluster



src: Kubernetes – Scheduling the future at Cloud Scale, David K. Rensin

Pod



- A pod is a collection of containers that are bundled and scheduled together because they share a common resource – usually a file system or IP address.
- Pod serves as Kubernetes' core unit of management.
- Pods make up the difference between containerization and virtualization by making it possible to run multiple dependent processes together.
- At runtime, pods can be scaled by creating replica sets.

Why not just run multiple programs in a single container?

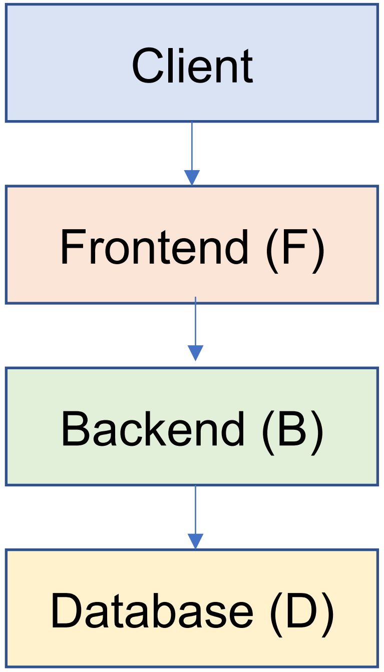
1. **Transparency**
   1. 1+ process in a container – you are responsible for monitoring and managing the resources each uses.
   2. By separating logical units of work into separate containers – Kubernetes can manage for you
   3. Makes things easier to debug and fix.

2. **Deployment and Maintenance**
   1. Individual containers can be rebuilt and redeployed by you whenever you make a software change.
   2. This decoupling of deployment dependencies will make your development and testing faster.
   3. It also makes it super easy to rollback in case there is a problem.

3. **Efficiency**
   1. The infrastructure takes on more responsibility, so the containers can be lighter-weight.

Cluster Management Example with k8s

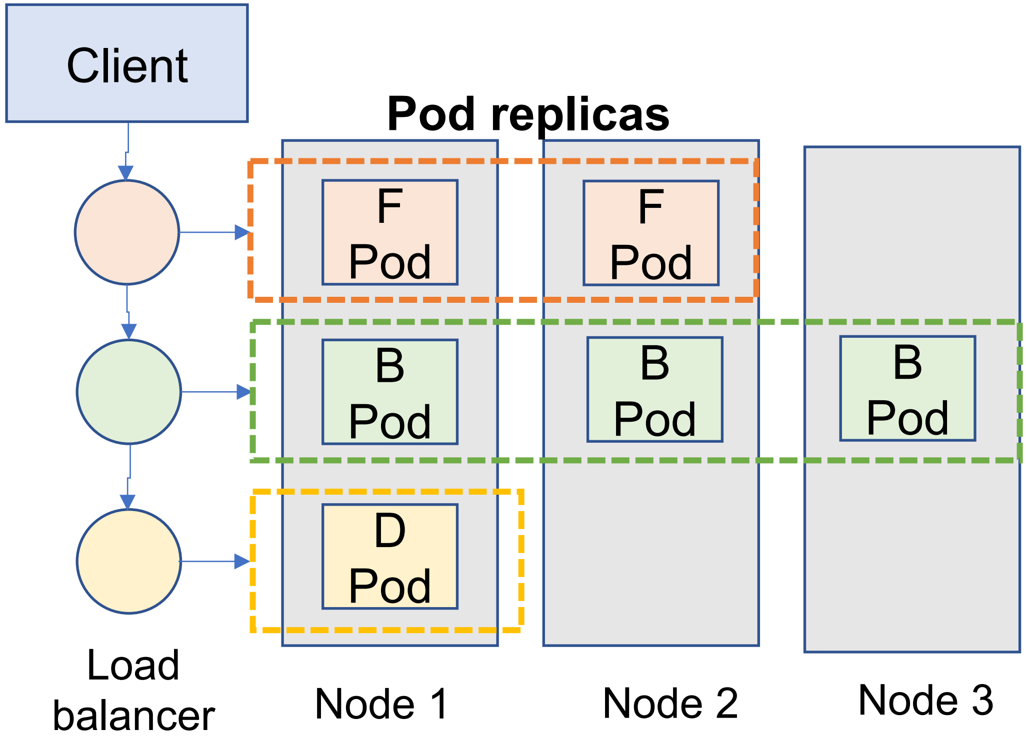
Web Service (Logical View)



K8s functionalities:

- Deployment
- Rollouts
- Service discovery
- Storage & networking
- Load balancing & scaling
- Failure recovery

Web Service (Physical View)



Questions?