Lecture 18:
Containers & Virtualization

John Cunniff

github/wabscale

Some slides derived from: G. Sandoval, Tanenbaum/Bo,
Jérôme Petazzoni, and Brendan Dolan-Gavitt
Thanks !!

whoami

- Graduated from NYU 2 years ago
- Was president of the OSIRIS Lab
- Senior Engineer at Vola Dynamics
- Created & maintaining Anubis LMS

VOLA DYNAMICS

Intuitive. Fast. Robust.

Industry-leading options analytics.

What is Vola Dynamics?

- “Swiss army knife” for options analytics
- 100x faster, robust & accurate option pricer & volatility fitter

VOLA DYNAMICS

Intuitive. Fast. Robust.

Industry-leading options analytics.

Jobs Jobs Jobs

Vola Dynamics is also hiring! (SWEs & Options Quants)

- Send your resumes to **resumes@voladynamics.com**

Applied Containers & Orchestration

I may be teaching a class next semester!

Applied Containers & Orchestration

Put in your course reviews that you want a modern containers course!

Virtualization in VMs

- Virtualization in VMs
- Containers
  - Namespacing
  - Cgroups
- Where containers run
- Cloud / k8s / Anubis

Virtualization in VMs

Until today we've been talking about operating systems running on **physical machines**: a collection of

1. **real hardware** resources,
2. that the operating system has **exclusive** access
3. through **hardware interfaces** (instruction set architectures, device I/O ports, etc.

Virtualization in VMs

Operating systems can also run inside **virtual machines (VMs)**.

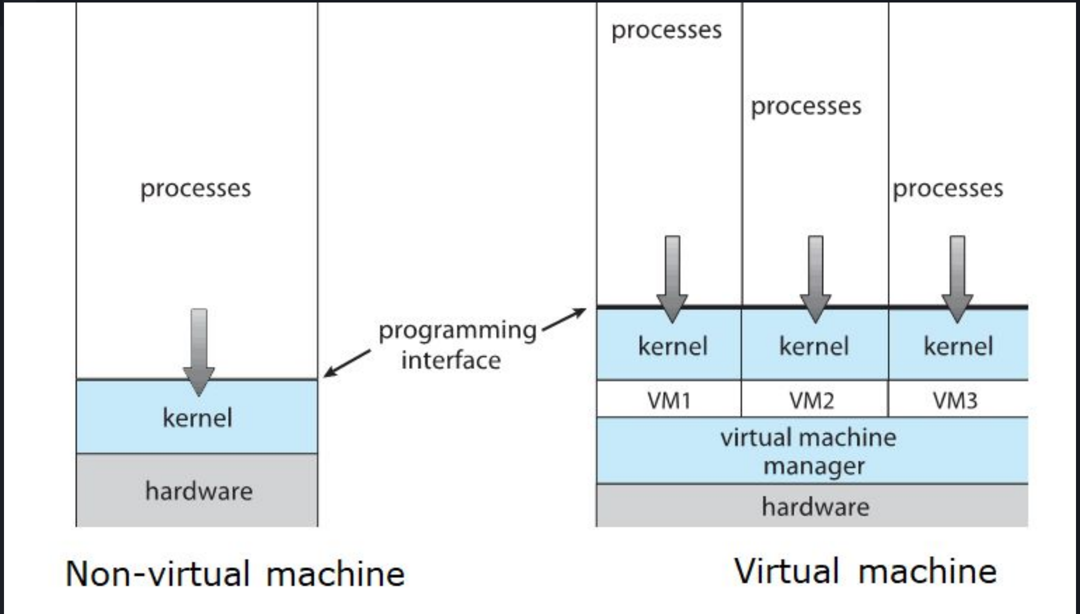
- We refer to an operating system running inside a virtual machine as a ***guest OS.***
- Virtual machines differ from physical machines in important ways.
- They ***do not* provide** the guest OS with **exclusive** access to the underlying physical machine.
- Equivalently, they ***do not* provide** the guest OS with **privileged (**or fully-privileged) access to the physical machine.

Virtualization in VMs

Virtualization adds a **new layer** to the software stack: the **hypervisor** (*aka Virtual Machine Monitor or* ***VMM***). The VMM mediates shared access to hardware by the different OSes and is:

- a piece of software running on an operating system (the **host OS**)
- that can allow another operating system (the **guest OS**) to be run as an application
- alongside other applications.

No-VM vs VM



Silberschatz, Galving and Gagne. 2018

Why Virtualize?

- **Flexibility**: virtual machines are easier to instantiate and tear down, can be ***migrated*** between physical hosts
- **Stability**: by splitting services across different virtual machines, if one crashes it will not affect the others
- **Security**: virtual machines are isolated from one another, so (e.g.) the web server can't access data on the email server

Some Terminology for VMs

- The running hypervisor is known as the **host**
- The virtual machines running under the hypervisor are **guests**
- If there is a special guest VM that is used for managing the rest, it is usually called **domain 0** or the **control domain**

Three approaches to Virtualization

- **Full virtualization**. Should be able to run an **unmodified** guest OS. **Example**: VirtualBox, VMWare.

- **Paravirtualization**. Includes **small changes** to the **guest operating system** to improve interaction with the virtual machine monitor. **Example**: Xen, Amazon EC2.

- **Container virtualization**. **Namespace** and other isolation techniques performed *by* ***the operating system*** to isolate sets of applications from each other. **Example**: Docker

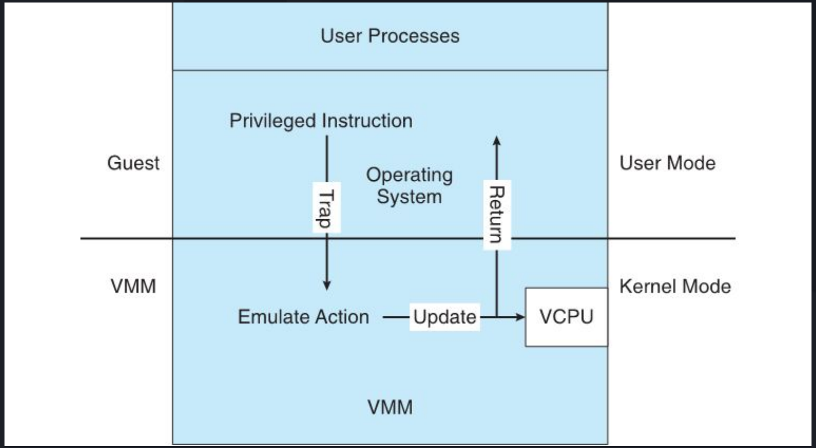
Requirements for Virtualization

- On most CPUs, there are **sensitive** instructions –those that **behave differently in kernel vs user mode**
  - Performing I/O, changing MMU mappings, etc.
- There are also **privileged** instructions – those that cause a **trap** into kernel mode
- Popek and Goldberg showed that an architecture is virtualizable only if the sensitive instructions are a *subset* of the privileged instructions

x86 Hardware Virtualization

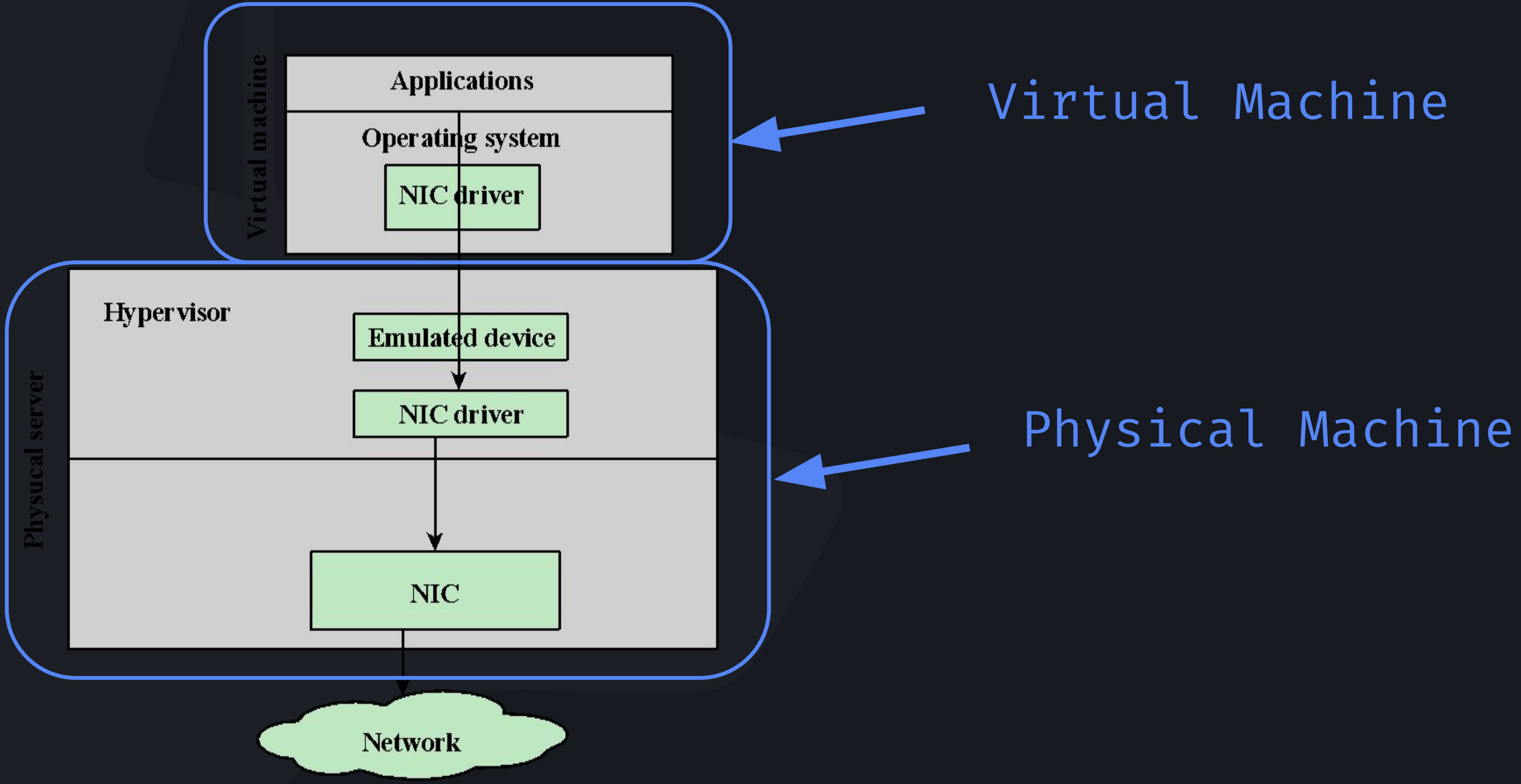
- In 2005 Intel fixed these issues by introducing VT-x
  - Around the same time AMD also fixed them... with an incompatible set of virtualization extensions 😑
- VT-x introduced two new processor modes: ***root mode*** and ***non-root mode***
- When the processor is running in non-root mode, **sensitive instructions** cause a **vmexit** – aka, a **trap to the hypervisor**

Virtualization Implementation



1. Priv. Instruction
2. Trap
3. Emulate Action
4. Update VCPU
5. Return

Silberschatz, Galving and Gagne. 2018


Virtual machine
Applications
Operating system
NIC driver
Physucal server
Hypervisor
Emulated device
NIC driver
NIC
Network
Virtual Machine
Physical Machine

I/O Virtualization

- Two final issues exist with virtualized I/O: DMA and interrupts
- For DMA, the problem is that the DMA hardware will be programmed with physical addresses, which must be **remapped** by the hypervisor
  - (~2009) Intel added an **IOMMU** that allows device memory accesses to be remapped without hypervisor intervention
- Interrupts from devices must also be **remapped** – the interrupt number seen by the guest virtual machine may not be the same as the interrupt number seen by the host

Review: Virtualization

How did we create a virtual machine (VM)?

- Start with a physical machine
- Create software (hypervisor) responsible for isolating the guest OS inside the VM
- VM resources (memory, disk, networking, etc.) are provided by the physical machine but visibility outside of the VM is limited

Review: Virtualization

What were the implications?

- VM and physical machine *share same instruction set*, so must the host and guest
- Guest OS can provide a *different application binary interface* (ABI) inside the VM
- Lots of challenges in getting this to work because guest OS expects to have privileged hardware access

Containers

- Virtualization in VMs
- Containers
  - Namespacing
  - Cgroups
- Where containers run
- Cloud / k8s / Anubis

OS Virtualization ⇒ Containers

How do we create a virtual operating system (container)?

- Start with a real operating system.
- Create software responsible for isolating guest software inside the container
- (That software seems to lack a canonical name—and today it's actually a bunch of different tools.)
  - runc, rkt, lxc, and docker to name a few
- Container resources (processes, files, network sockets, etc.) are provided by the real operating system but **visibility outside the container is limited**

Containers

What are they exactly

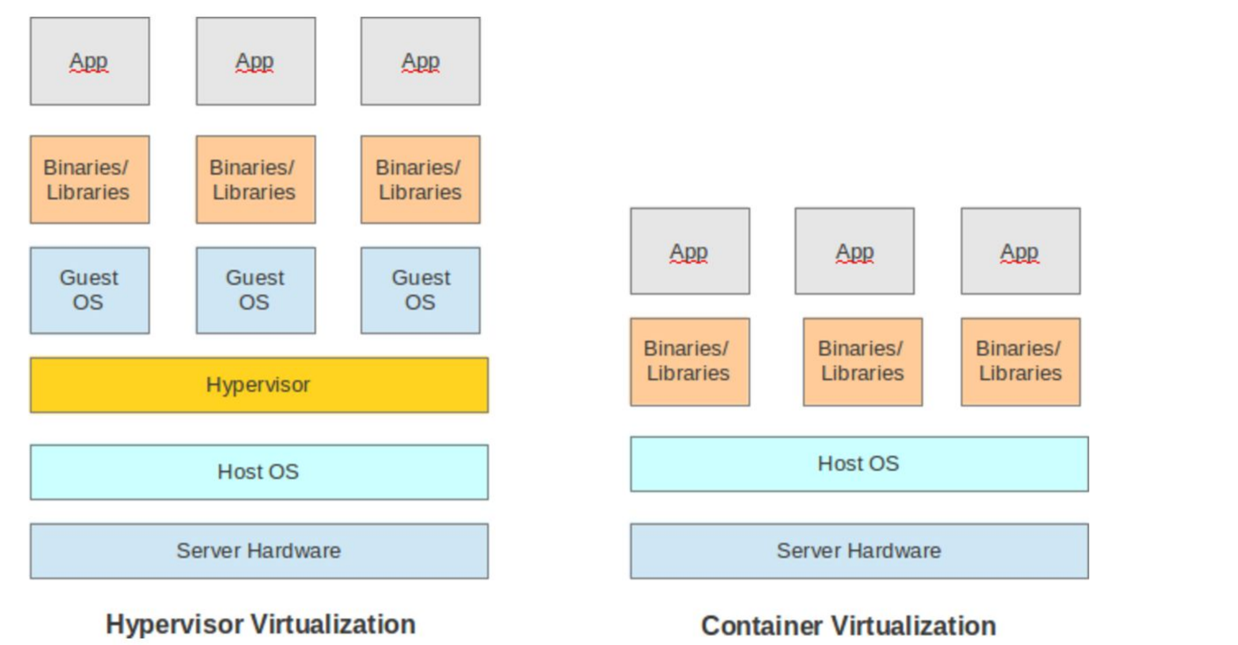
- Sort of like chroot on steroids
- They are implemented through user level Container Engines / Runtime, **not by the kernel itself**
- You probably already know Docker
    - Docker itself uses containerd/runc for the actual containers
- There is also lxc, rkt (pronounced rocket), and runc for example

Containers

What are the implications?

- Container and real OS *share* ***same kernel***
- So applications inside and outside the kernel must ***share the same ABI*** (Application Binary Interface)
- Challenges is getting this to work are due to *shared OS namespaces*

VM vs Container



Containers vs VMs (T/F)

You can run a Windows container on GNU/Linux.

- False. Container shares the kernel with the host.

You can run a Debian container from Glorious Arch Linux.

- True. As long as the container uses the same kernel

Containers vs VMs (T/F)

Running ps inside the container will show all processes running on the machine.

- False. Container process namespaces is isolated from the host.

Why use containers at all?

Shares many (but not all) of the benefits of hardware virtualization with **much lower overhead**.

- Can **package** a program / service into something that will run exactly the same on most any machine.
- Can **adjust / limit** hardware container resources to system needs.
- Can **split a system** up into microservices, then use a CNI (container networking interface) to let them connect to each other.

Why use containers at all?

Isolation

- Container should not leak information inside and outside the container
- Can isolate all of the configuration and software dependencies a particular application needs to run

Containers

Container system call path:
Application inside the container **makes a system call**

- **Trap** to the host OS
- It is then up to the **kernel to consider** resource namespacing

Containers

On GNU/Linux you are always in a container!

- **Linux starts in a container** with **no limits** that can see everything
- So if you think you're getting a performance benefit by not using containers you're wrong!

Containers

- Virtualization in VMs
- Containers
  - Namespacing
  - Cgroups
- Where containers run
- Cloud / k8s / Anubis

Namespacing

- When you run a container, your container runtime creates a set of namespaces for that container
- Provide a layer of isolation. Limits what you can see/affect/use
- Implemented within the kernel
- Multiple types of resource **namespaces**

Types of Namespaces

- pid
- net
- mnt
- uts
- ipc
- user

Namespacing

`ls -l /proc/self/ns` to see what namespaces you are in

```
┌─jc@athena ~/jcs  ‹master›
└─➤ ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 net -> 'net:[4026531992]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 time -> 'time:[4026531834]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 user -> 'user:[4026531837]'
lrwxrwxrwx 1 jc jc 0 Apr 10 18:08 uts -> 'uts:[4026531838]'
```

Number is what pid namespace the current process is in

PID Namespacing

- Processes within a PID namespace only see processes in the **same PID namespace**
- Each PID namespace has its **own numbering** starting at 1
- If PID 1 exits, the whole namespace is killed

PID Namespacing

- Those namespaces can be **nested**
- A process ends up having different PIDs **depending on namespace**

PID Namespacing

What happens when you run ps in a container?

Only the ps process visible

PIDs start at 1

```
┌─jc@athena ~
└─➤ docker run -it alpine ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 ps aux
┌─jc@athena ~
└─➤ █
```

Net Namespacing

Net namespace in practice

- **Typical use-case:** use virtual ethernet (veth) pairs (two virtual interfaces acting as a cross-over cable)
  - eth0 in container network namespace
    paired with vethXXX in host network namespace
    all the vethXXX are bridged together
- But also: the magic of --net host shared localhost (and more!)

Net Namespacing

Let's think about what this lets us do

- Create a virtual interface with its own network
- Use net namespace in multiple containers
- **You then have multiple docker containers that are connected to each other on a virtual network**

This is where things get really powerful

Containers

- Virtualization in VMs
- Containers
  - Namespacing
  - Cgroups
- Where containers run
- Cloud / k8s / Anubis

Cgroups

- ## Control Group

- Implemented within the kernel
- limits what resources you are allowed to use
- cpu and memory cgroups very common with containers
- It is up to your container engine to handle the cgroup

CPU Cgroups

- CPU cgroup Keeps track of user/system
- CPU time Keeps track of usage per CPU Allows to set weights
- Because of variations in things like core clock speed, and instruction time execution, there is no 100% precise way to limit CPU

CPU Cgroups

Try systemd-cgtop to see cgroup usage!

Control Group	Tasks	%CPU	Memory	Input/s	Output/s
/	1689	5.0	6.0G	0B	254.7K
user.slice	1122	4.4	37.8G	0B	127.3K
user.slice/user-1000.slice	1122	4.4	37.8G	-	-
user.slice/user-1000.slice/session-9.scope	821	3.2	5.5G	-	-
user.slice/user-1000.slice/session-8.scope	268	1.1	31.0G	-	-
system.slice	102	0.3	1.0G	-	-
system.slice/tailscaled.service	21	0.2	137.9M	-	-
user.slice/user-1000.slice/user@1000.service	32	0.0	89.6M	-	-
system.slice/systemd-oomd.service	1	0.0	1.6M	-	-
system.slice/containerd.service	21	0.0	88.7M	-	-
dev-hugepages.mount	-	-	56.0K	-	-
dev-mqueue.mount	-	-	80.0K	-	-
init.scope	1	-	7.2M	-	-
sys-fs-fuse-connections.mount	-	-	8.0K	-	-
sys-kernel-config.mount	-	-	24.0K	-	-
sys-kernel-debug.mount	-	-	4.0K	-	-
sys-kernel-tracing.mount	-	-	4.0K	-	-
system.slice/boot-efi.mount	-	-	36.0K	-	-
system.slice/dbus.service	1	-	1.8M	-	-
system.slice/docker.service	39	-	639.7M	-	-
system.slice/home.mount	-	-	84.0K	-	-
system.slice/polkit.service	3	-	4.9M	-	-

There's so much more!

- Some stuff we're not covering but is very cool
  - Linuxkit
  - Storage drivers
  - Overlay networks
  - Copy-on-Write!
  - Container registries
  - selinux + capabilities
  - Rootless docker
  - Build-kit
  - Breaking security

Docker on Linux

Docker on Mac

Docker on Windows

Containers

- Virtualization in VMs
- Containers
  - Namespacing
  - Cgroups
  - Where containers run
- Cloud / k8s / Anubis

Where containers run

- LXC (or linux containers) were initially released in 2008
- Since then there have been many more engines / container runtimes that have come about
- **containerd**, **rkt**, **podman**, etc…

Where containers run

You may be asking how docker works on MacOS and Windows since it’s kernel is not GNU/Linux

- **It doesn’t**
- Docker for MacOS runs a **linux virtual machine** that then runs docker
- The networking and volumes do not always work as expected

Where containers run

- **They all revolve around Linux**
- There has since been windows containers added (*but they are awful*)

Containers

- Virtualization in VMs
- Containers
  - Namespacing
  - Cgroups
- Where containers run
- Cloud / k8s / Anubis

Where containers run

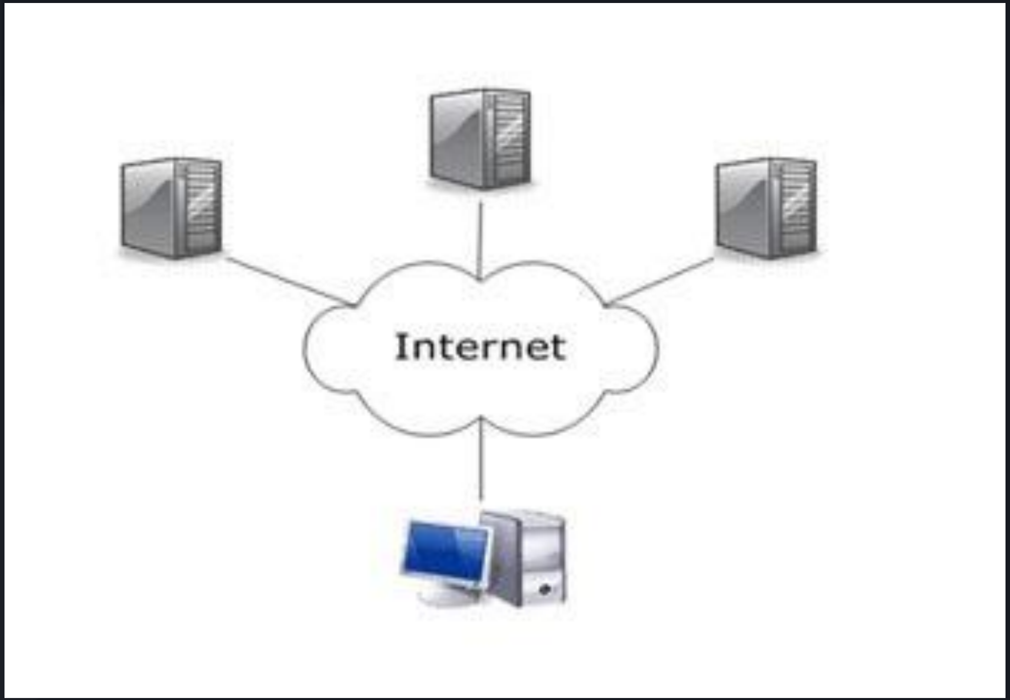
Some of you may have only used docker on MacOS or Windows and hate it

- The things you hate are all from docker-desktop not from docker itself!
- **Docker runs like a dream on GNU/Linux**
- On GNU/Linux it has barely any overhead

The Cloud

- With the resurgence of virtualization, it has become popular to talk about the **cloud**
- This **somewhat nebulous concept** traces back to old network diagrams
- Usually used a **buzzword**

The Cloud



The Cloud

- In modern usage, the cloud refers to a **large number of physical servers** that rent out virtual machines for various services

- Clients get access to a **full virtual machine**

- Billing usually works according to how and what resources you use

The Cloud

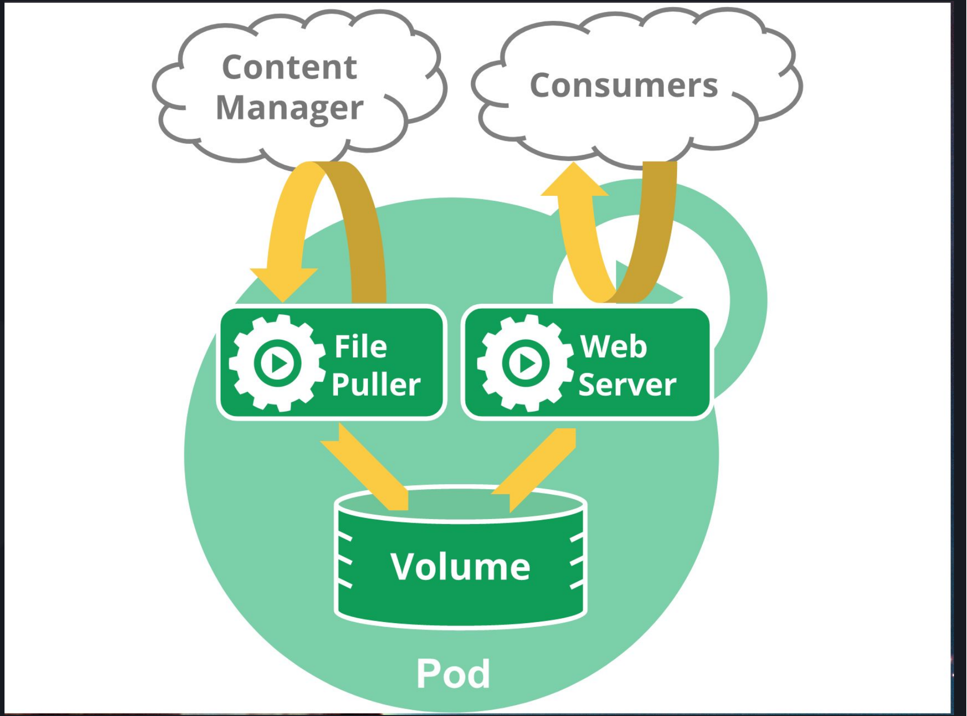
- Importantly, creating and destroying virtual servers can be accomplished **without human interaction**
- This ability to **flexibly acquire computing resources** can allow services to scale in response to changes on demand

Kubernetes

- Anubis runs on a container orchestration tool called **Kubernetes** or k8s (the 8 is for the number of letters in between k and s)
- Kube allows for things like CNI (container networking interfaces) and CSI (container storage interface) to be extended to many, many machines connected on a network
- This lets us design and easily implement large systems that rely on *many many* individual containers communicating at once

Kubernetes Pod

- A single unit of work in kubernetes
- Pods contain **containers**, **volumes**, **config-maps**, **secrets**, …
- Designed to be easy to **share resources** between containers in a pod

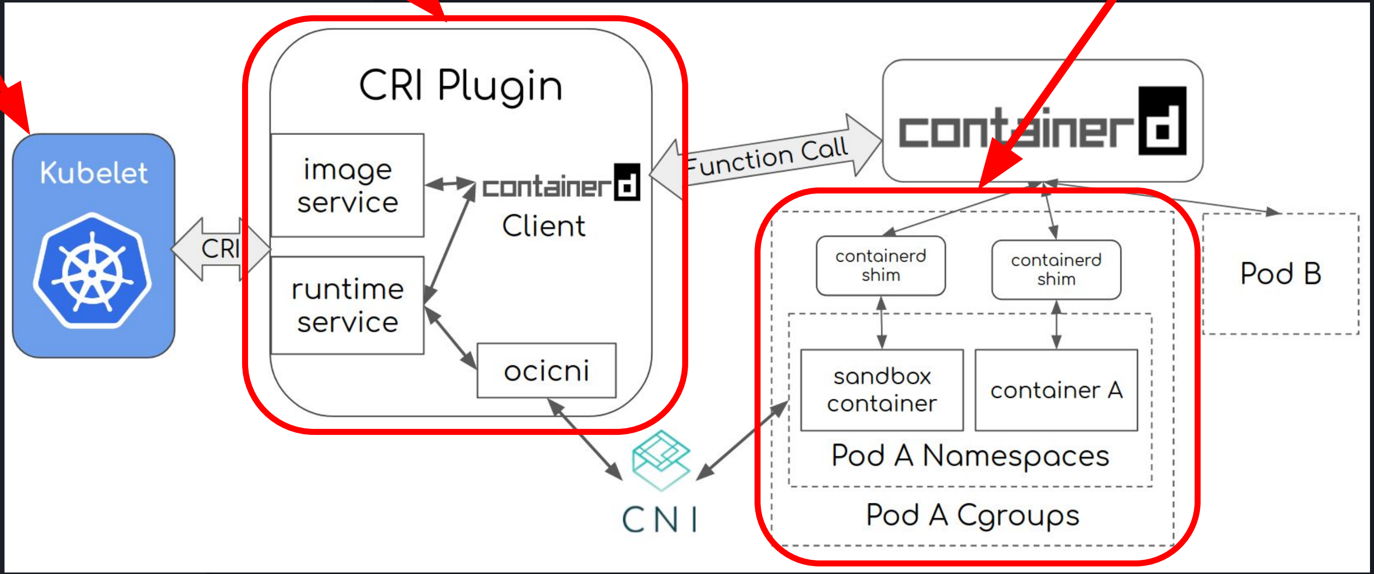


Kubernetes Pod Containerd

Primary K8S Node Agent

Container Resource Interface Plugin

Containerd Creates Pod Containers



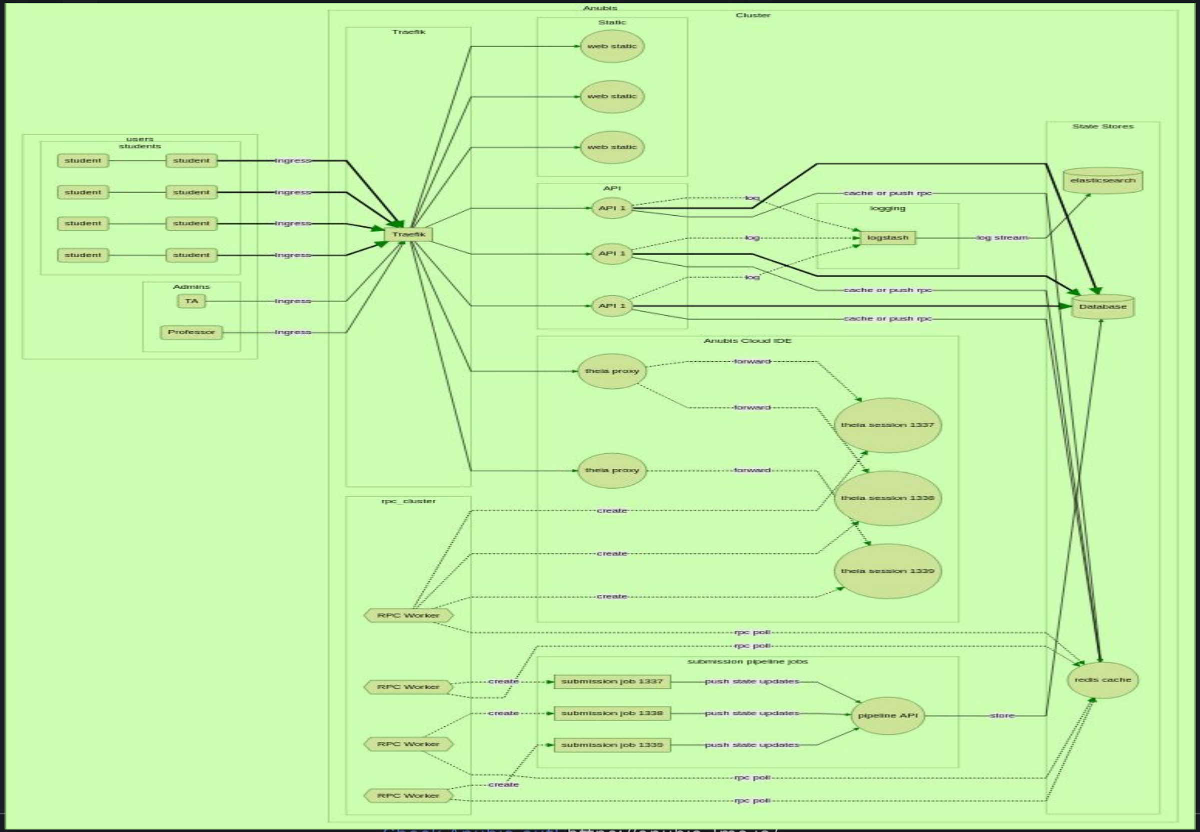
Anubis

- Anubis is a large system split up into **microservices**
  - Example: the web static (html and js) is separate from the python api
- There can be **many containers** within those microservices

Anubis

- There may be **thousands of containers running at any one time**
- 2023-02-07 there were **~366 IDEs open at once**
    - ~3000 individual containers for **JUST IDEs**

Anubis



Anubis
Cluster
Traefik
Static
web static
web static
web static
users
students
student
Admins
TA
Professor
ingress
API
API 1
log
logging
logstash
log stream
cache or push rpc
State Stores
elasticsearch
Database
Anubis Cloud IDE
theia proxy
forward
theia session 1337
theia session 1338
theia session 1339
rpc_cluster
RPC Worker
create
rpc poll
submission pipeline jobs
submission job 1337
submission job 1338
submission job 1339
push state updates
pipeline API
store
redis cache

Anubis IDEs

- Anubis Cloud IDEs are made up of **individual containers**
- Each student gets their own IDE pod (and therefore **separate environment/filesystem**)
- The IDEs have **CPU and Memory limits** handled by cgroups

Anubis IDEs

- Each Anubis Cloud IDE is itself made up of **3+ containers**
  - An “init container” that clones your repo
  - IDE server
  - Autosave “sidecar”
- The containers work together to make the Cloud IDEs possible

Anubis IDEs

Init Container
clones the git repo (has the
fixes any permission issues

Home Volume
/home/anubis
mounted over the nfs
mounted in each container

Theia IDE Server Container

Runs webserver you connect to
When you open a shell it
opens here

Shared
localhost

Sidecar Container

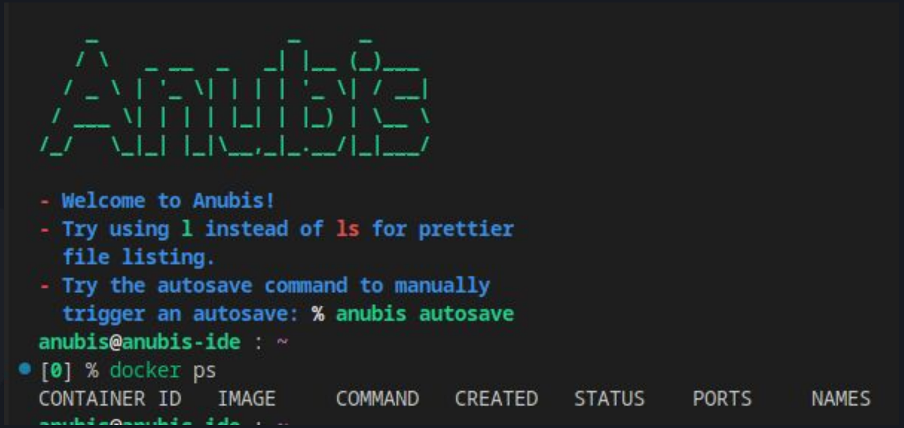
Handles autosave

Anubis IDEs

- The **containers work together** to make the Cloud IDEs possible
- It is all about **breaking up responsibilities**

Anubis IDEs

- Recently I have added a **dockerd** sidecar container
- This means **docker in IDEs**!



Future Readings

- Container Security by Liz Rice
  - github/lizrice
- Basically everything by Jess Frazelle
  - github/jessfraz
- Presentations by Jérôme Petazzoni
  - github/jpetazzo

Code to Read (All in Go)

- runC - OCI container spawning tool
  - github/opencontainers/runc
- containerd - container runtime
  - github/containerd/containerd
- kubernetes
  - github/kubernetes/kubernetes