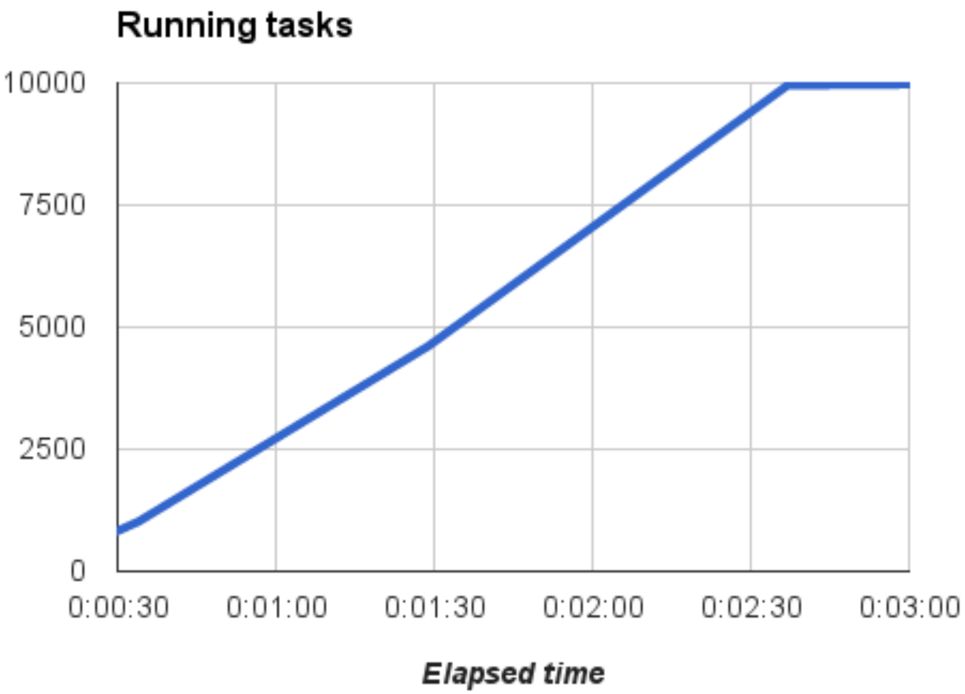
OK, so I have all these Containers
What now?

Image by Connie Zhou

Developer View

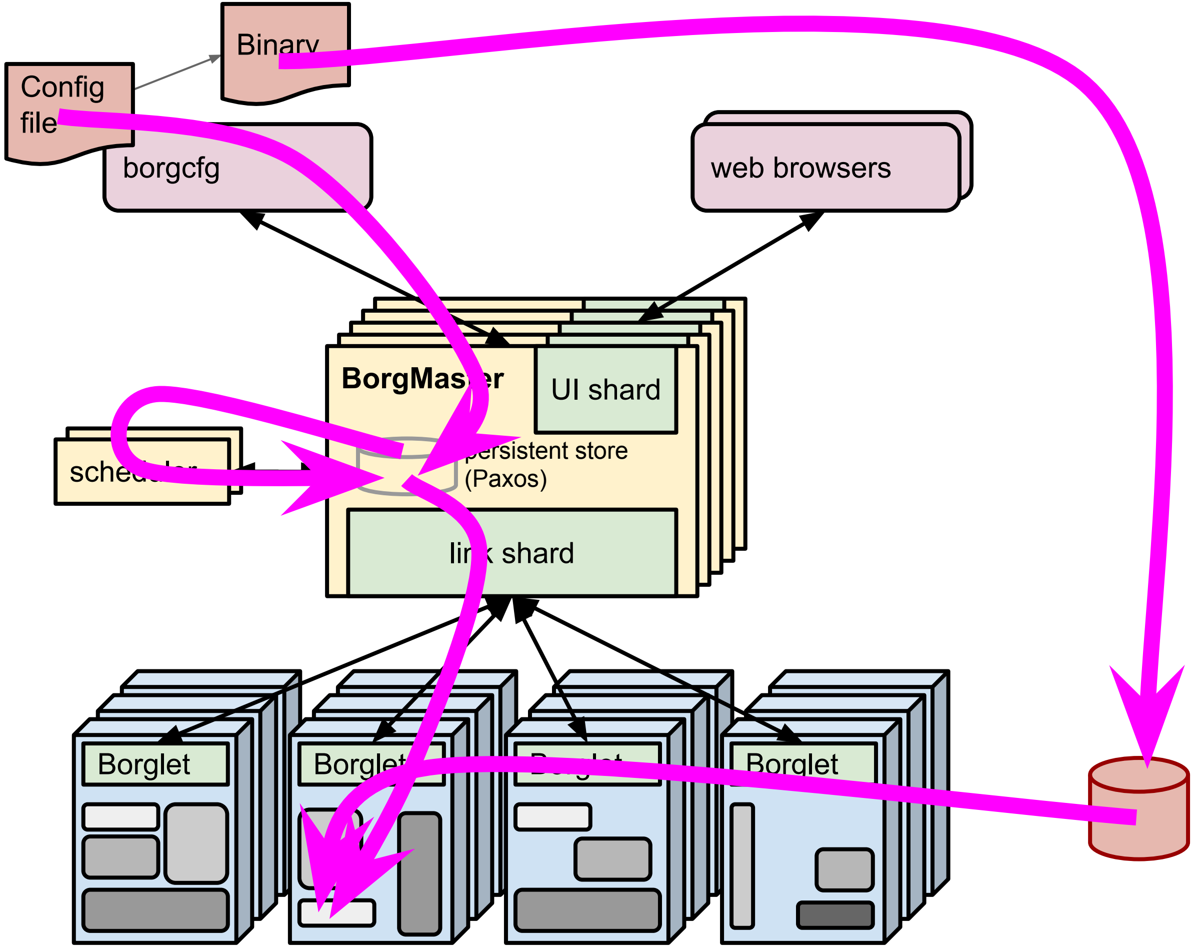
```
job hello_world = {
  runtime = { cell = 'ic' }              // Cell (cluster) to run in
  binary = '.../hello_world_webserver'   // Program to run
  args = { port = '%port%' }             // Command line parameters
  requirements = {     // Resource requirements
    ram = 100M
    disk = 100M
    cpu = 0.1
  }
  replicas = 10000     // Number of tasks
}
```

Developer View



Developer View

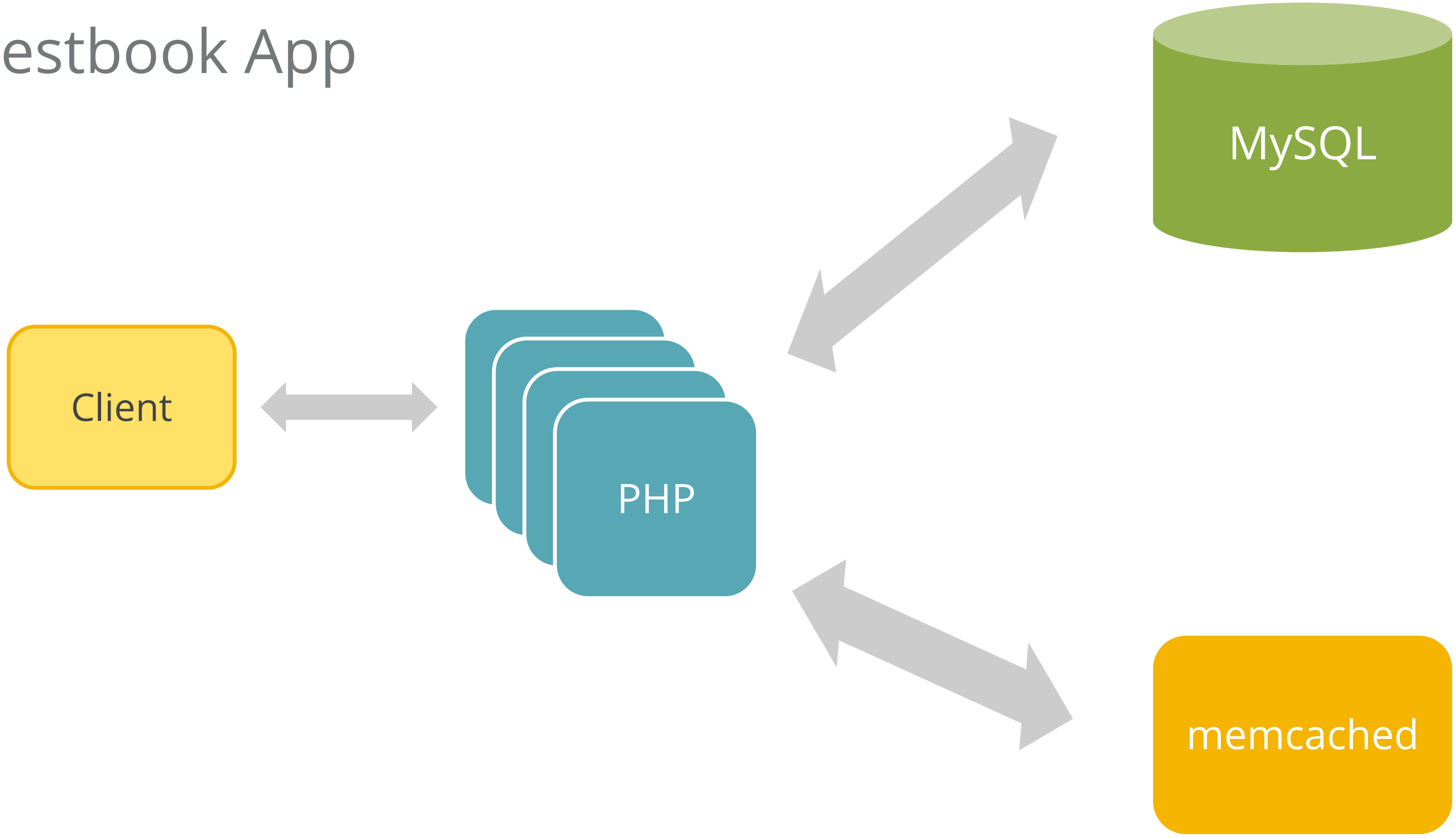
What just happened?



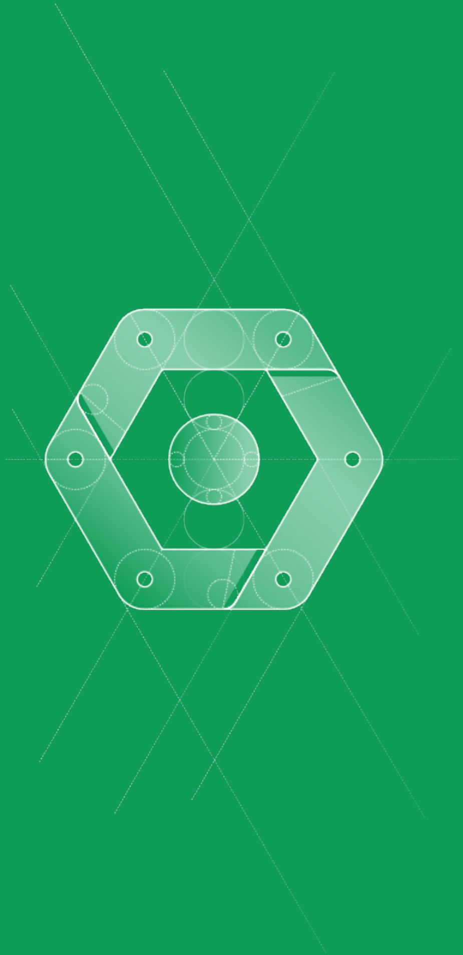
#kubernetes @tekgrrl

Image by Connie Zhou

Guestbook App



Containers



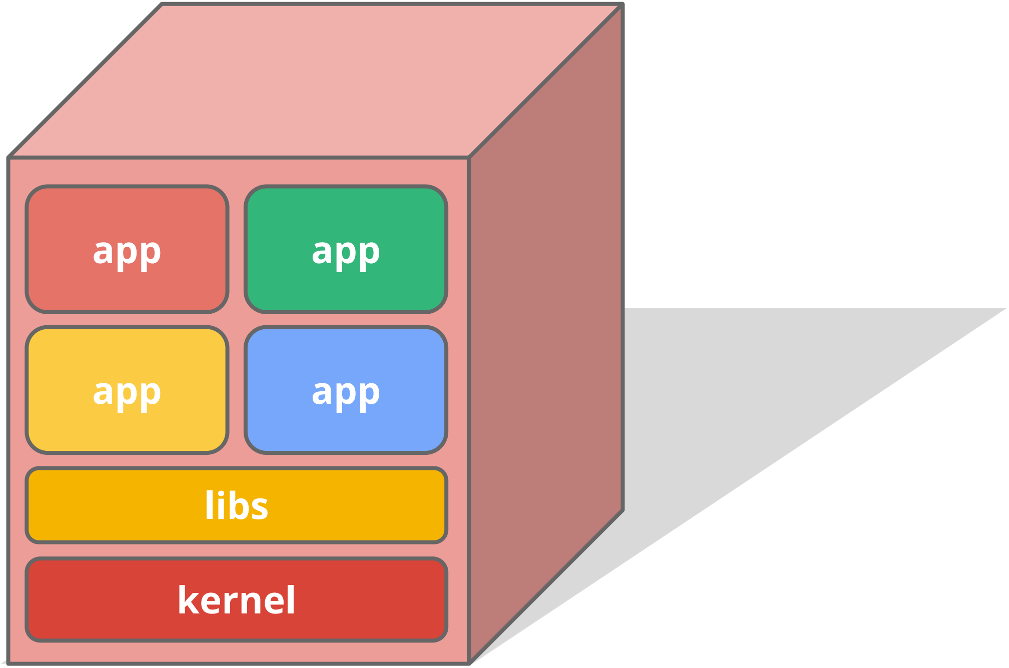
Old Way: Shared Machines

No isolation

No namespacing

Common libs

Highly coupled apps and OS



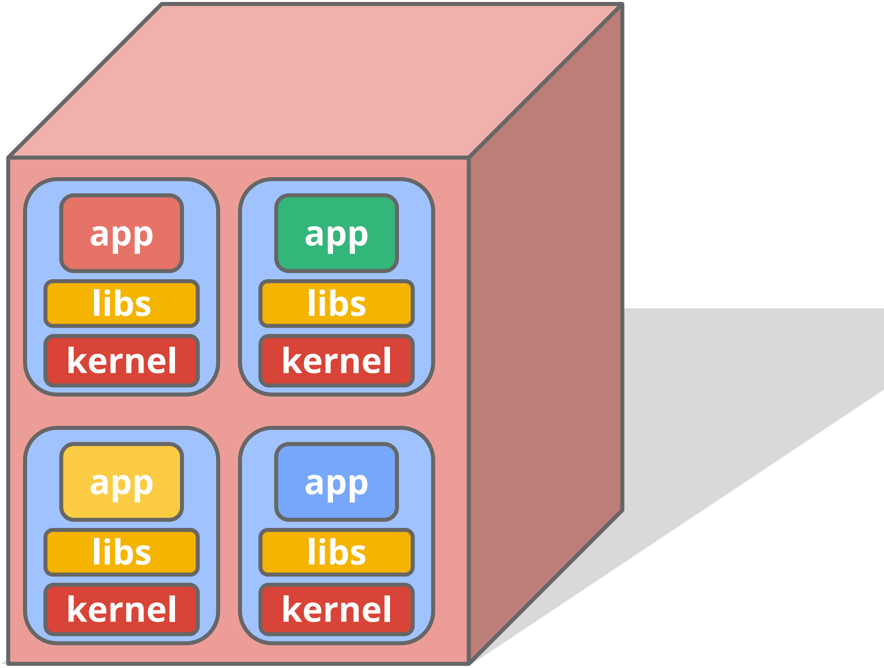
Old Way: Virtual Machines

Some isolation

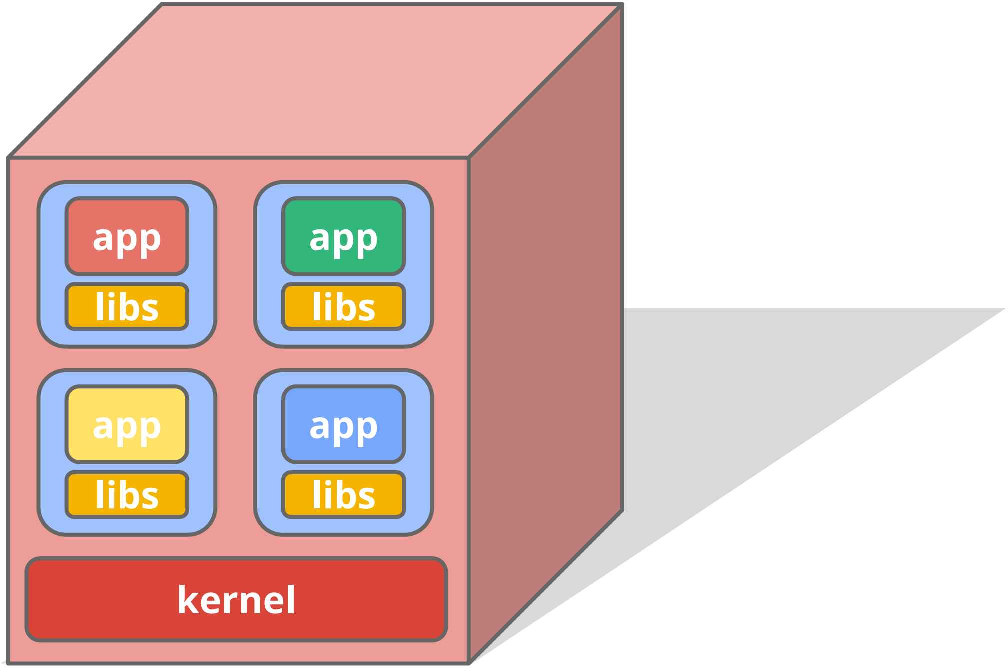
Inefficient

Still highly coupled to the guest OS

Hard to manage



New Way: Containers



Container Images

- An image is a stack of Read-Only file system layers.
- Usual process:
  - build
  - push to repository
  - pull to execution host
  - start container from image

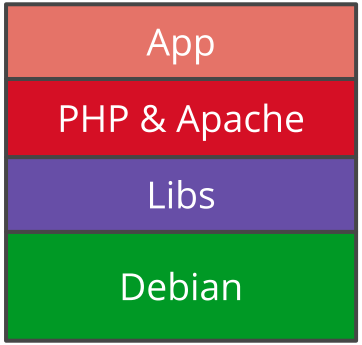
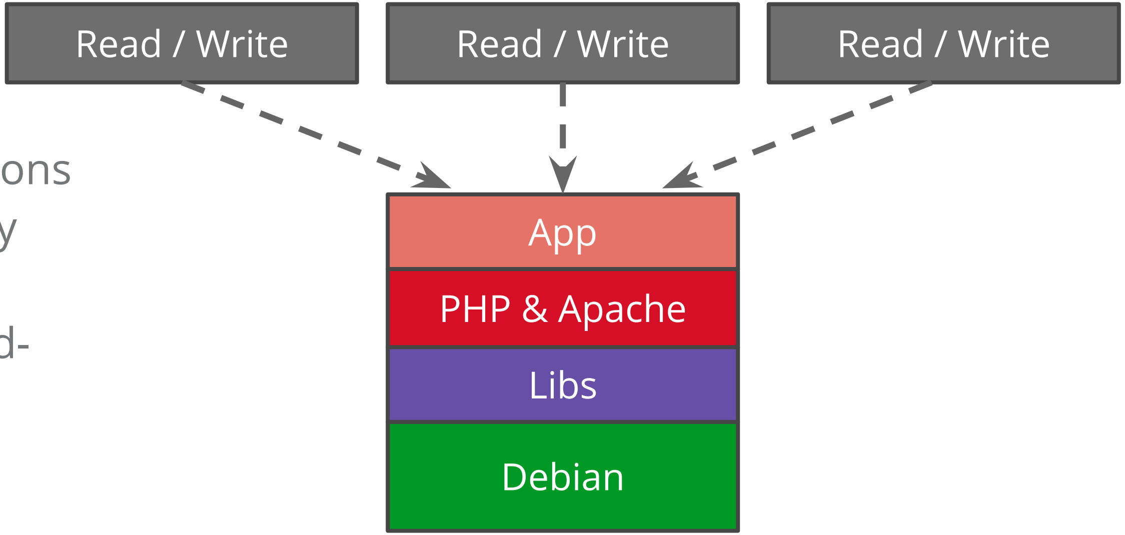


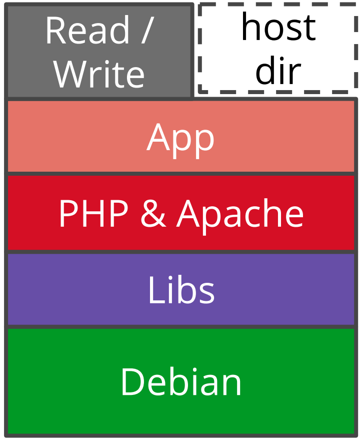
Image Layers

- A container is a process
  - started with kernel restrictions
  - a stack of shared Read-Only file system layers
  - plus a process specific Read-Write layer
- Every new container gets a new Read-Write later. All containers from the same image start from **exactly the same state!**

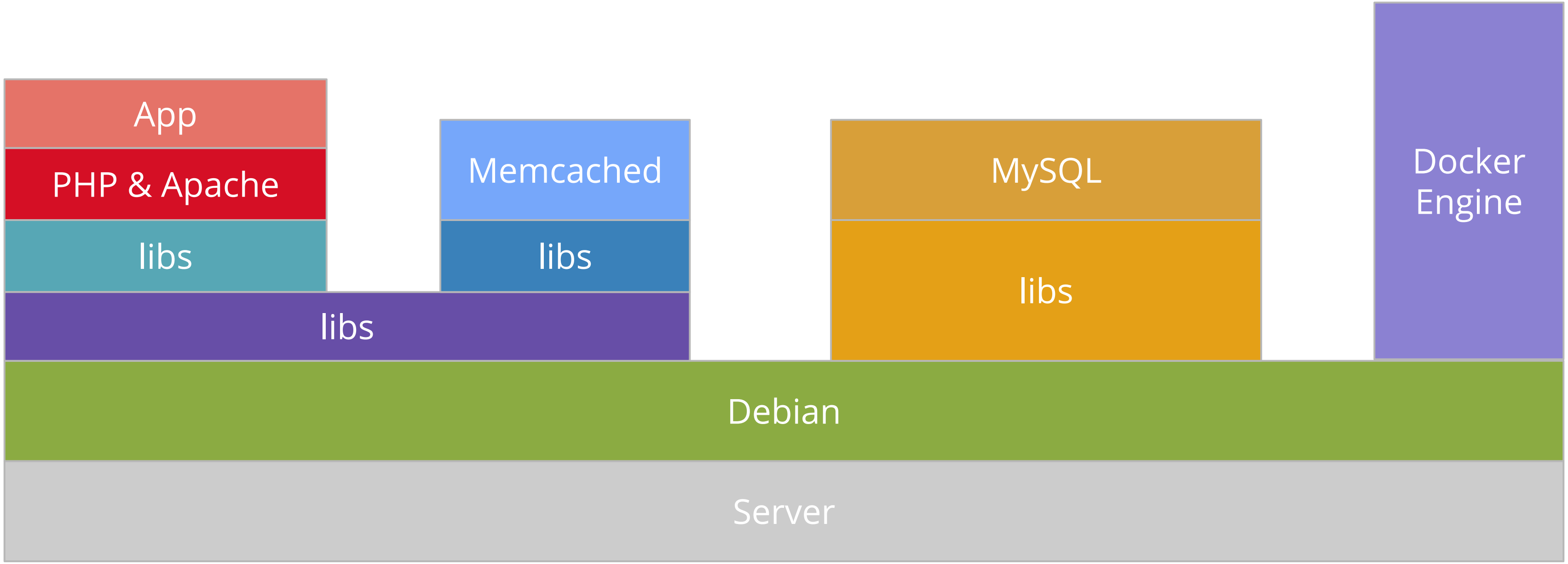


Mounting Host Directories

- It's possible to mount host directories into a container's filesystem.
- These are mutable and do outlive the container.
- They're **only** available on that host.



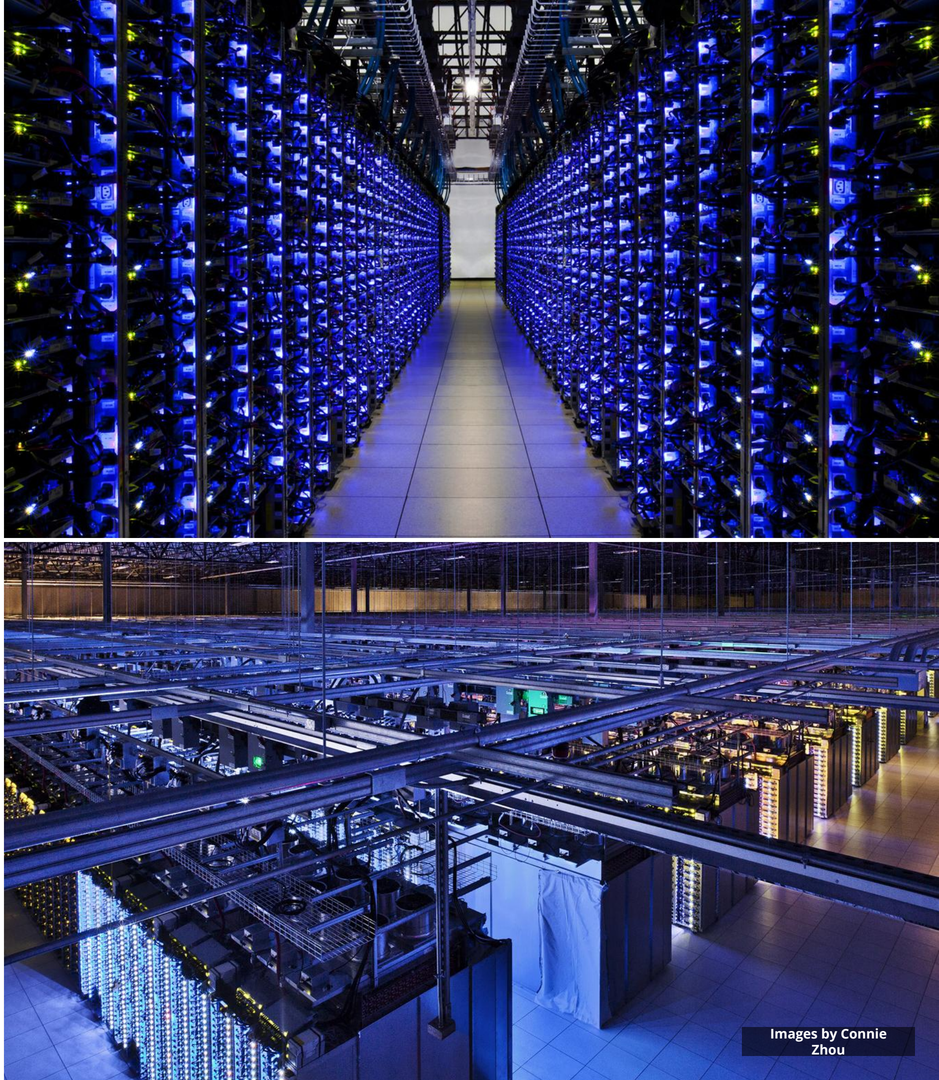
Docker Example



Why containers?

- Performance
- Repeatability
- Quality of service
- Accounting
- Portability

A **fundamentally different** way of managing **applications**



Images by Connie Zhou

containers are awesome
let's use lots of them!

Demo

Kubernetes

Kubernetes

Greek for *"Helmsman"*; also the root of the word *"Governor"*

- Orchestrator for Docker containers
- Supports multi-cloud environments
- Inspired and informed by Google's experiences and internal systems
- **Open source**, written in **Go**

Manage applications, not machines

Concepts Intro

Container

Pod

Service

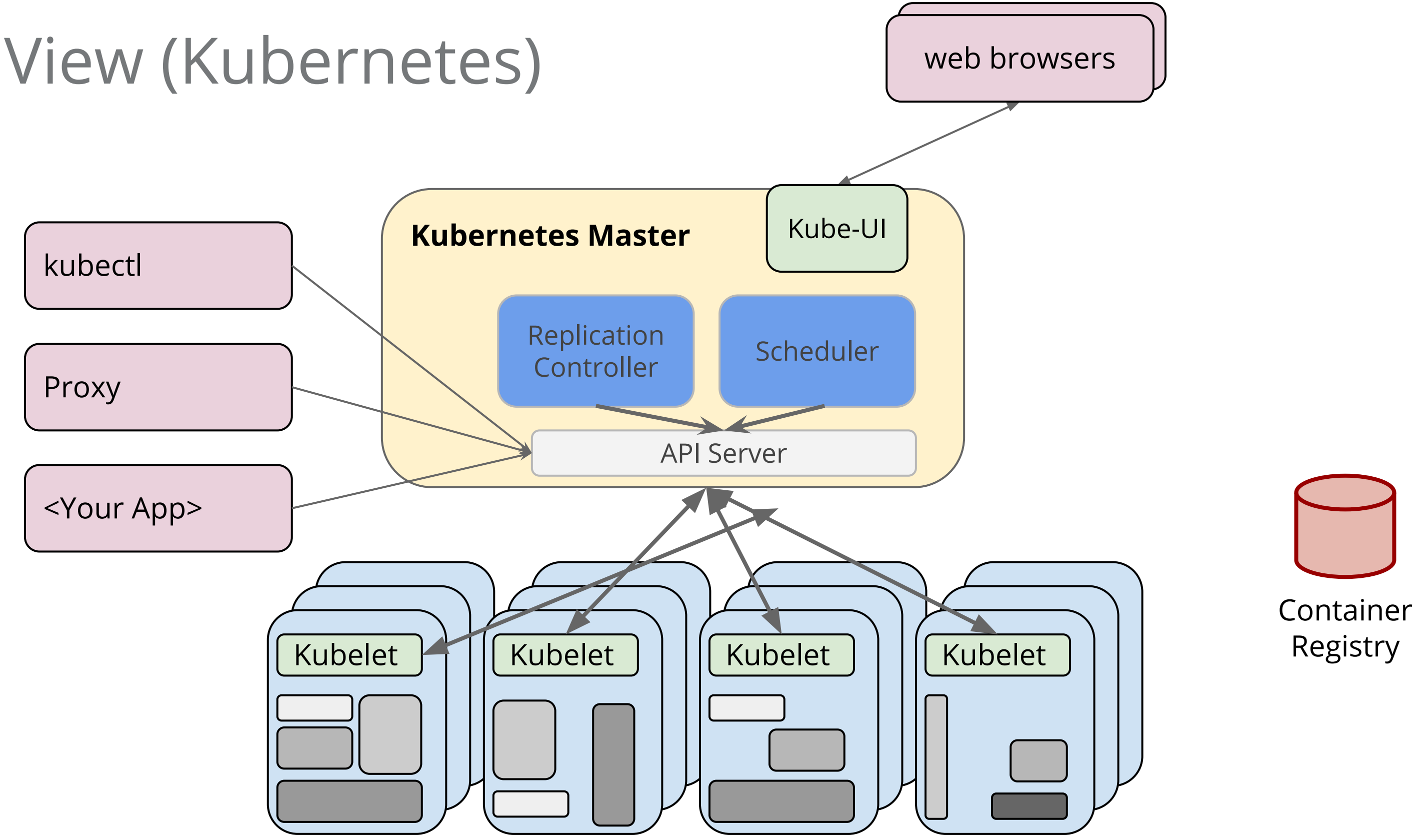
Volume

Label

Replication Controller

Node

Developer View (Kubernetes)



@tekgrrl #kubernetes #gotoldn

Cluster Options

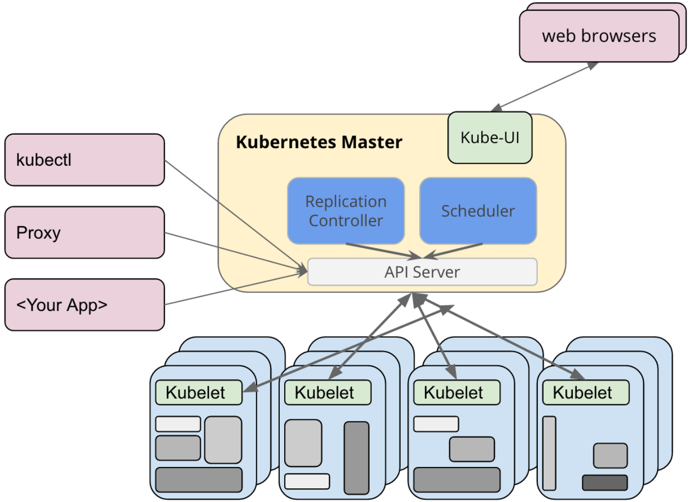
From **Laptop** to high-availability **multi-node cluster**

Hosted or **self managed**

On-Premise or **Cloud**

Bare Metal or **Virtual Machines**

Many options, See Matrix for details



Kubernetes Cluster Matrix: http://bit.ly/1MmhpMW

So what do we run on the nodes?

Containers?

Demo

Pods

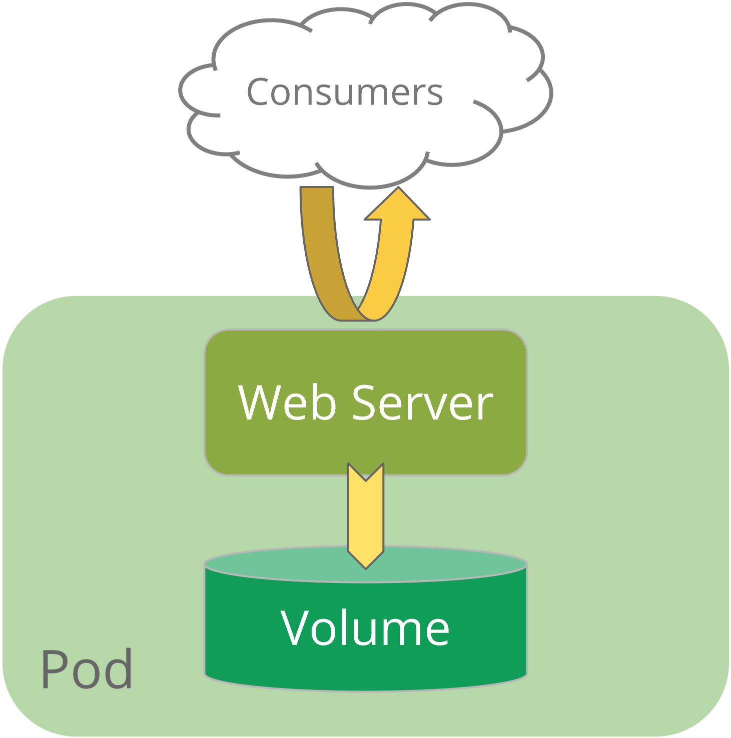
The atom of scheduling for containers

Application specific “logical host”

Ephemeral

- can die and be replaced

Single container pods can be created directly from a container image



Pods

Can be used to group containers & shared volumes

Containers are **tightly** coupled

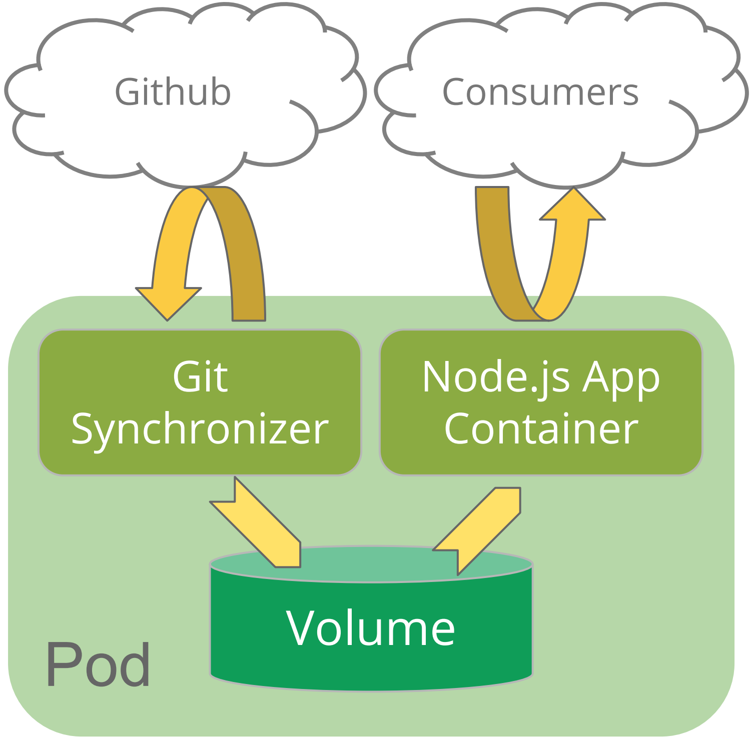
Shared namespace

- **Shared network IP and port namespace**

Ephemeral

- Containers in pods live and die together

Think in terms of services that you usually run on the same machine



Volume

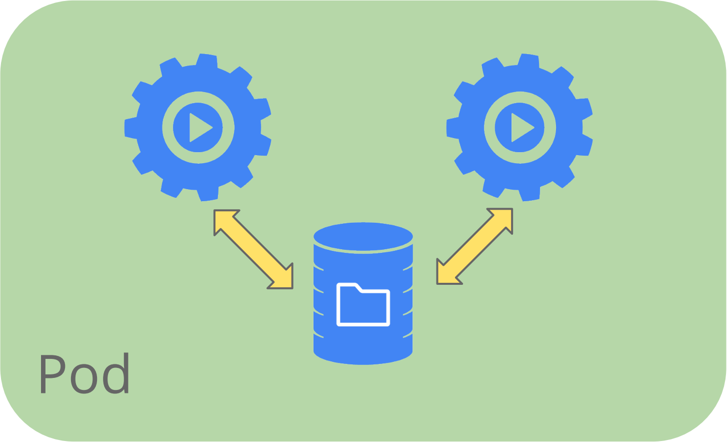
Bound to the Pod that encloses it

Look like Directories to Containers

What and where they are determined by Volume Type

Many Volume options

- **EmptyDir**
    - Lives with the pod



Volume

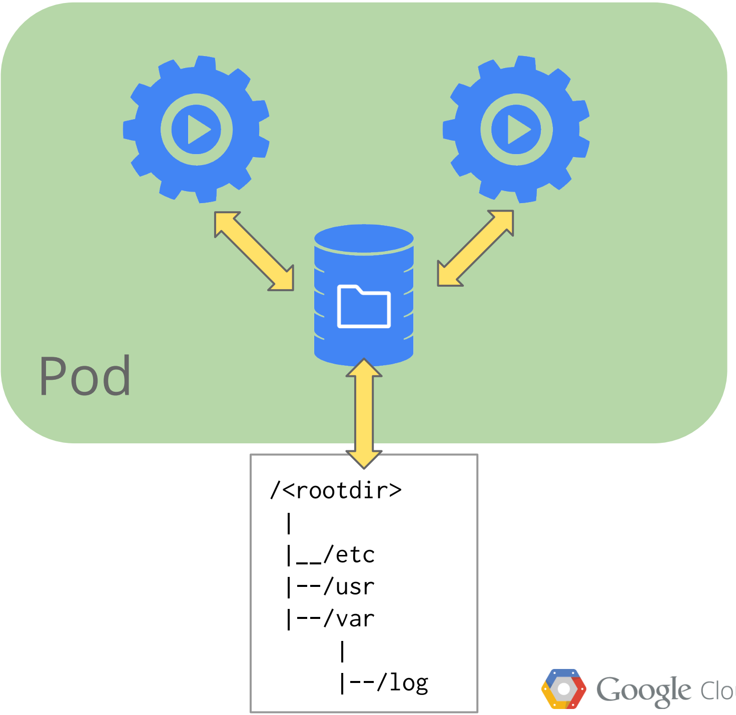
Bound to the Pod that encloses it

Look like Directories to Containers

What and where they are determined by Volume Type

Many Volume options

- EmptyDir
- **HostPath**
    - Maps to directory on host
    - Use with caution



```
/<rootdir>
 |
 |__/etc
 |--/usr
 |--/var
     |
     |--/log
```

Volume

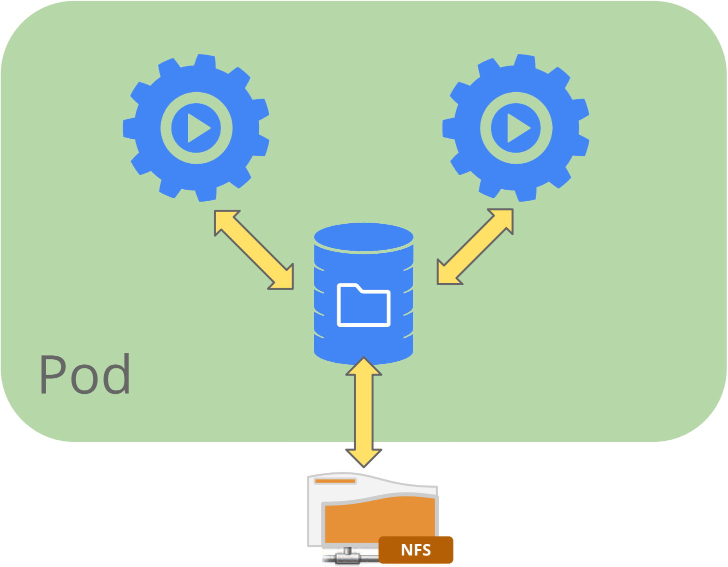
Bound to the Pod that encloses it

Look like Directories to Containers

What and where they are determined by Volume Type

Many Volume options

- EmptyDir
- HostPath
- **nfs (and similar services)**



Volume

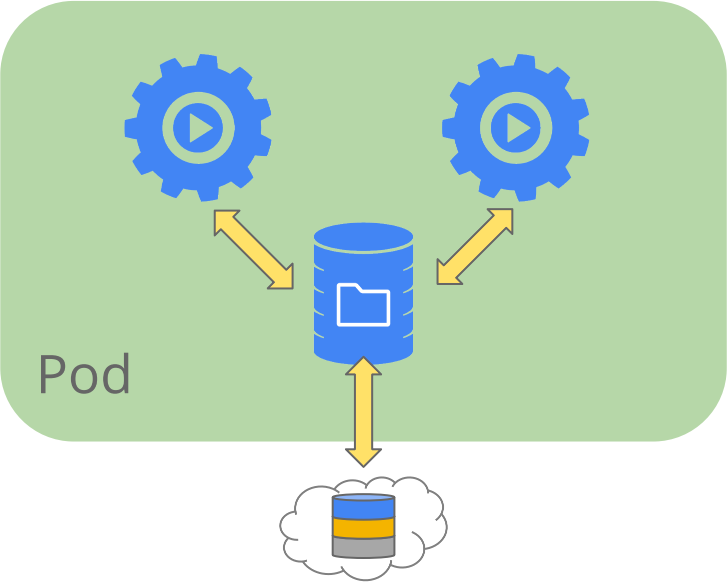
Bound to the Pod that encloses it

Look like Directories to Containers

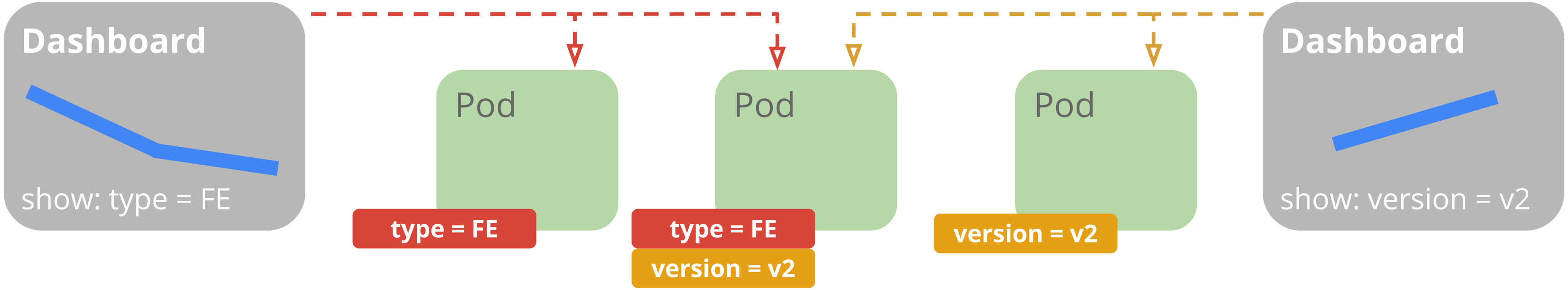
What and where they are determined by Volume Type

Many Volume options

- EmptyDir
- HostPath
- nfs (and similar services)
- **Cloud Provider Block Storage**



Labels ← These are important

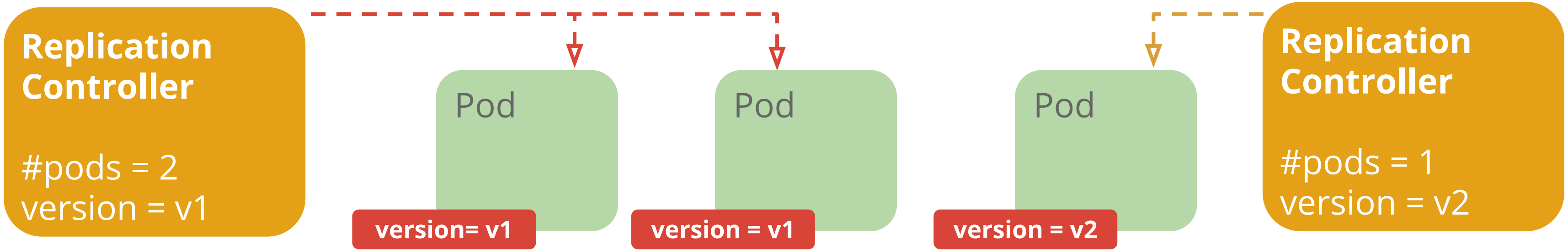


Behavior		Benefits
Metadata with semantic meaning	➔	Allow for intent of many users (e.g. dashboards)
Membership identifier	➔	Build higher level systems …
The only Grouping Mechanism	➔	Queryable by Selectors

Developer View (Replication Controller)

```
selector:
  name: frontend
...
spec:
  containers:
  - name: php-guestbook
    image: php-guestbook:europython
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
    ports:
    - containerPort: 80
      protocol: TCP
replicas: 10000
```

Replication Controllers



Behavior

- Keeps Pods running
- Gives direct control of Pod #s
- Grouped by Label Selector

Benefits

➔ Recreates Pods, maintains desired state
➔ Fine-grained control for scaling
➔ Standard grouping semantics

Replication Controllers

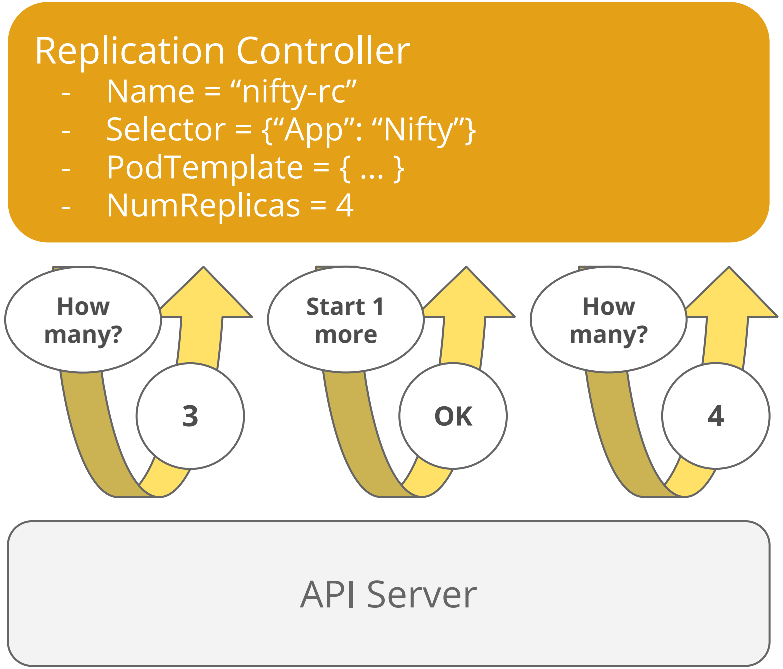
Canonical example of control loops

Have one job: ensure N copies of a pod

- if too few, start new ones
- if too many, kill some
- group == selector

Replicated pods are fungible

- No implied order or identity



Replication Controller

- Name = “nifty-rc”
- Selector = {“App”: “Nifty”}
- PodTemplate = { ... }
- NumReplicas = 4

Container Liveness

Process Level: Kubelet checks with Docker that Container is running

App Level: User defined health checks:

- HTTP Health checks (Kubelet calls a Web Hook)
- Container Exec (Kubelet runs command in container)
- TCP Socket (Kubelet attempts to open a socket to the container)

Services

A logical grouping of pods that perform the same function

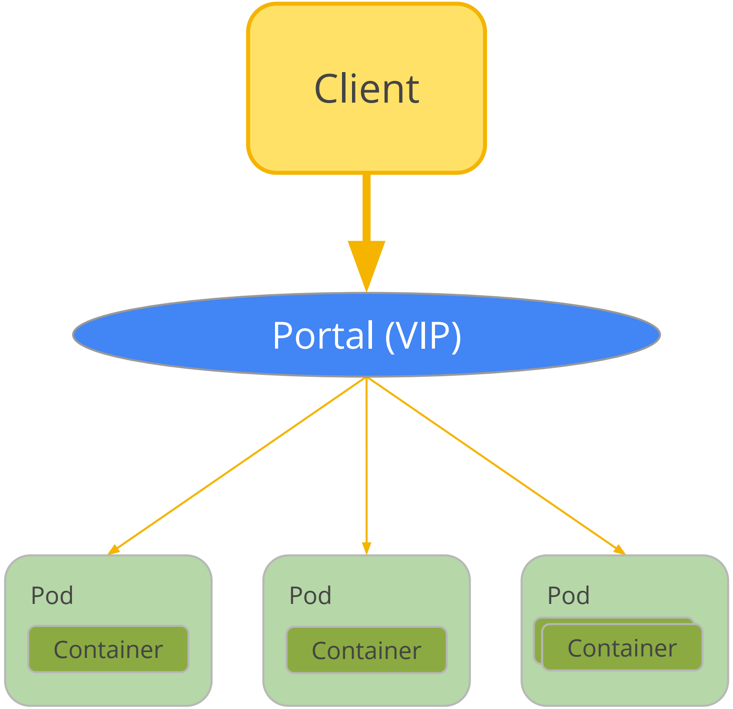
- group == selector

Choice of pod is random but supports session affinity (ClientIP)

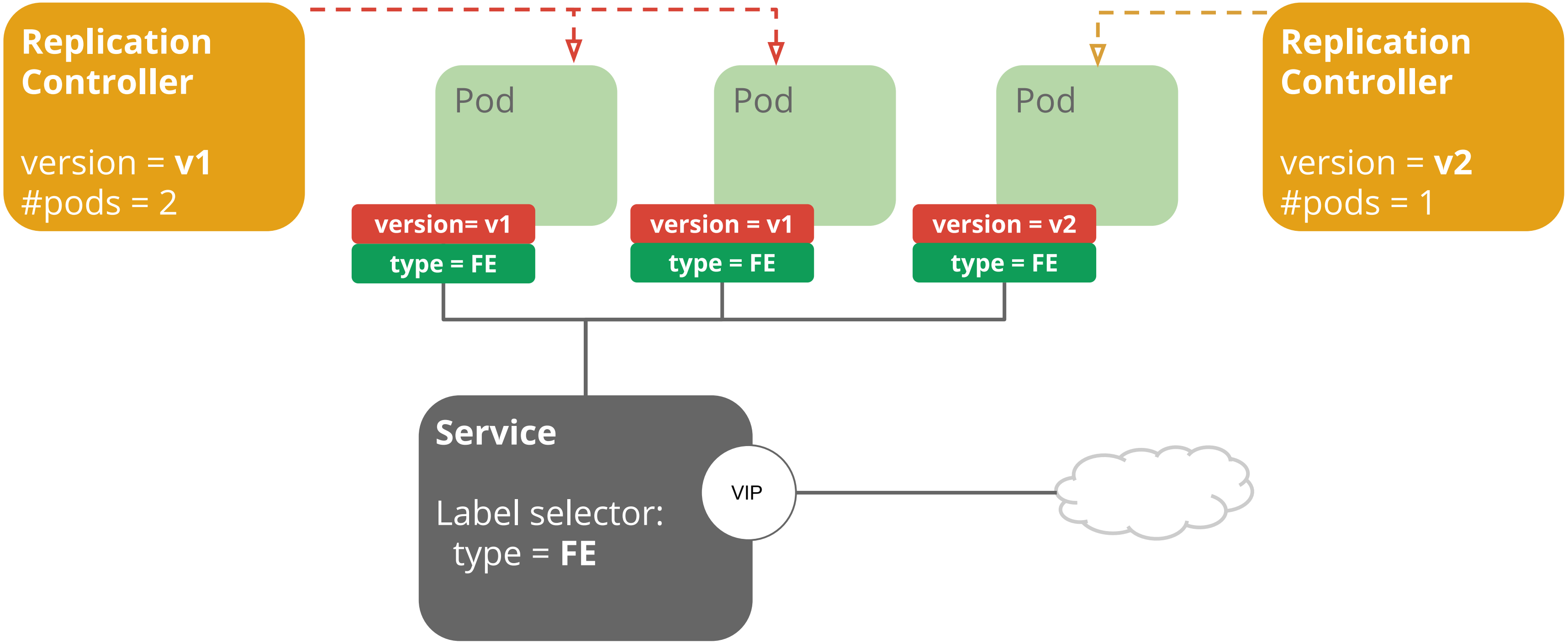
Gets a **stable** virtual IP and port

- also a DNS name

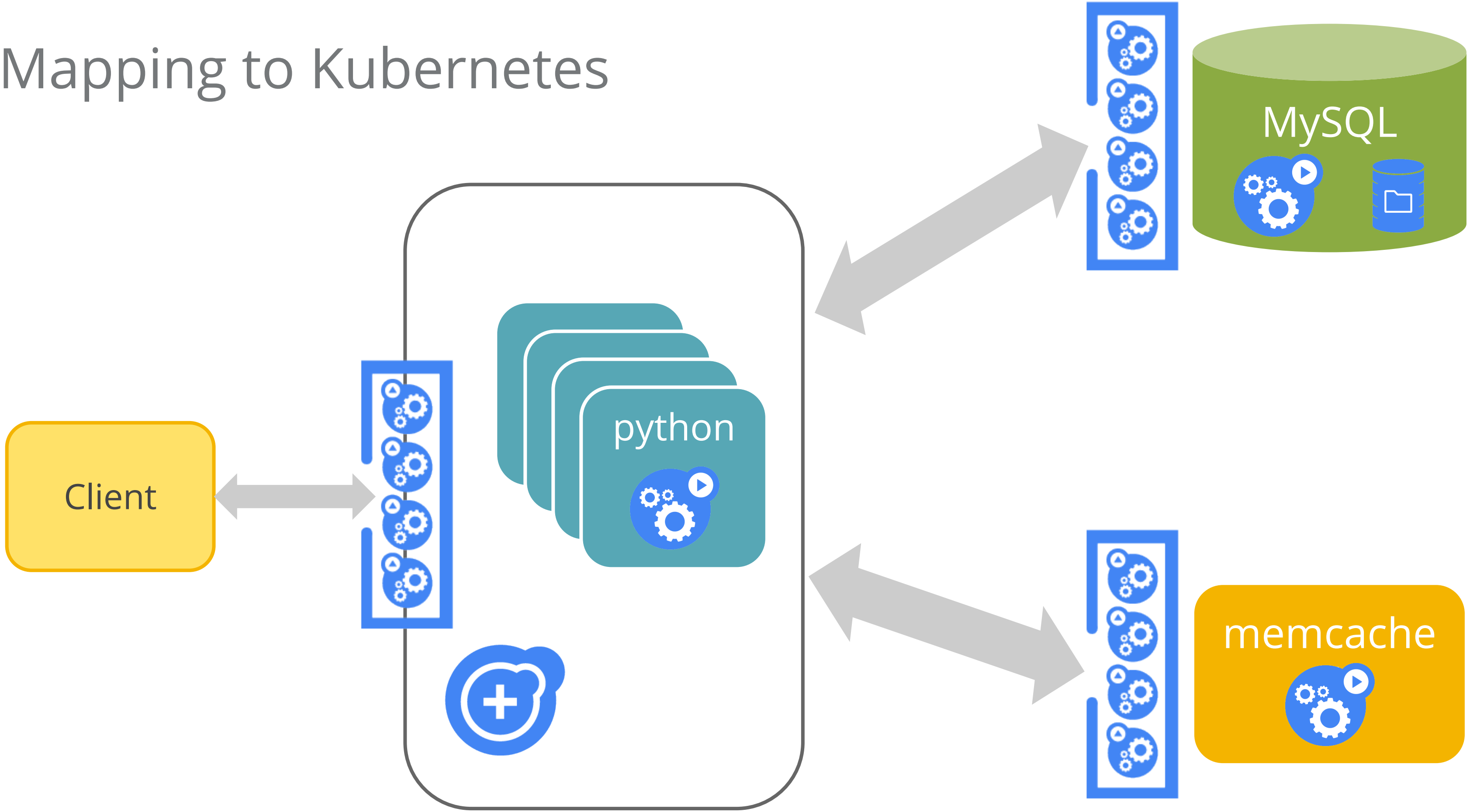
Hide complexity - ideal for non-native apps



Canary Example



Mapping to Kubernetes



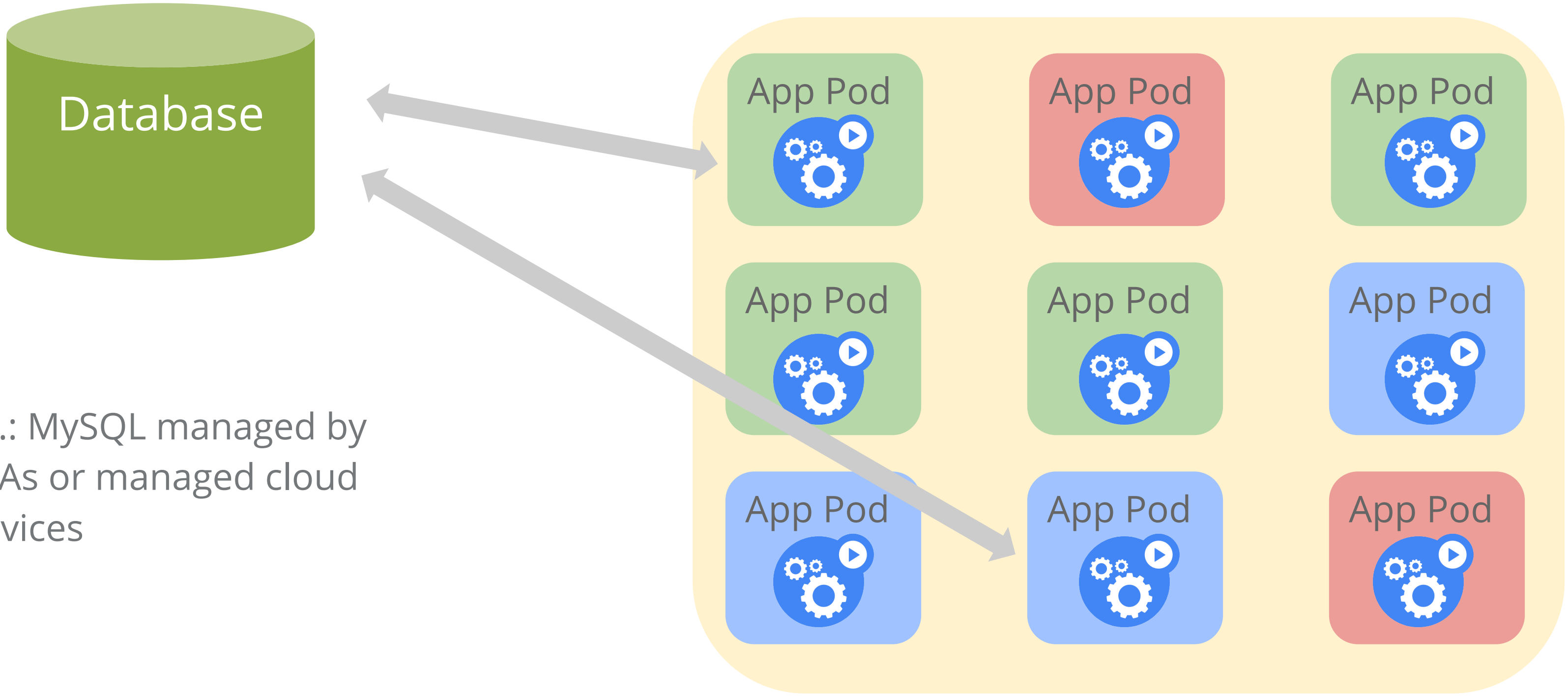
#kubernetes @tekgrrl

I still have questions about state!

In a cluster of ephemeral containers

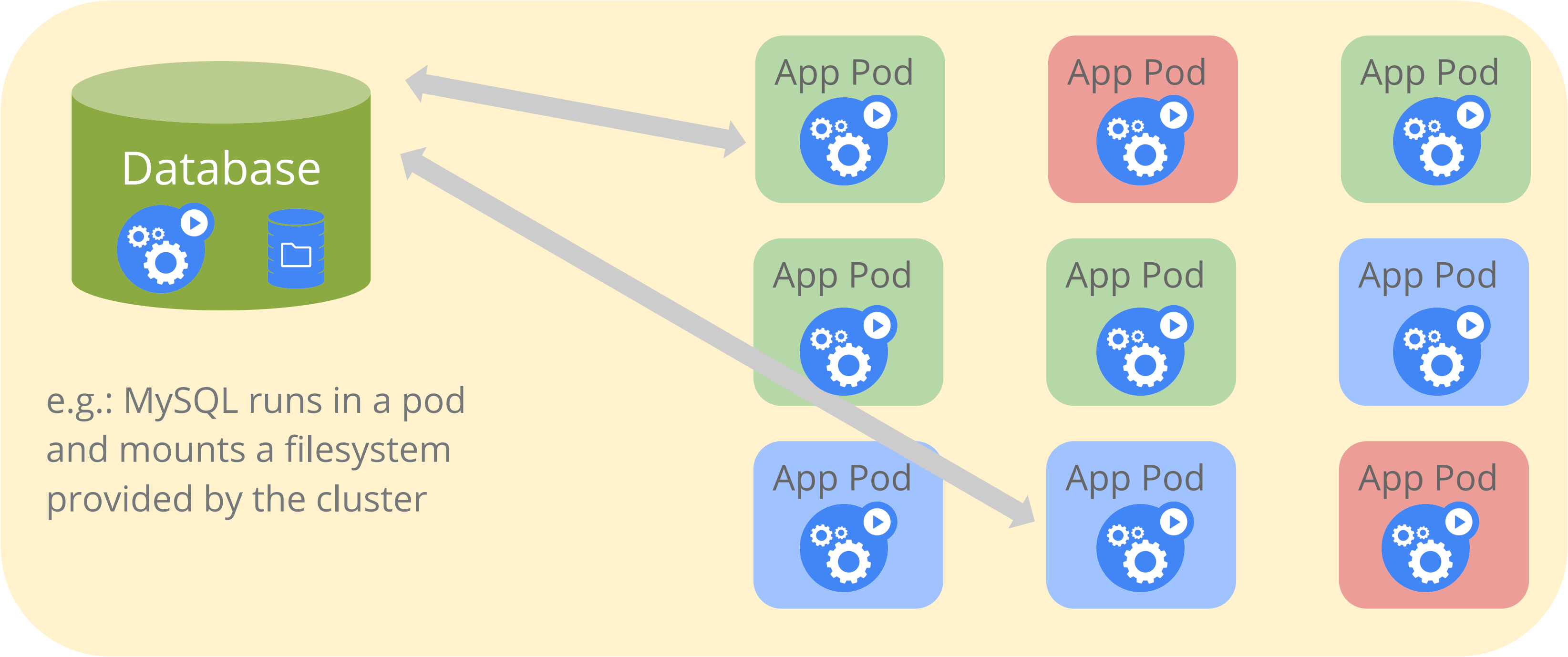
Application state must exist outside of the container

Outside the Cluster



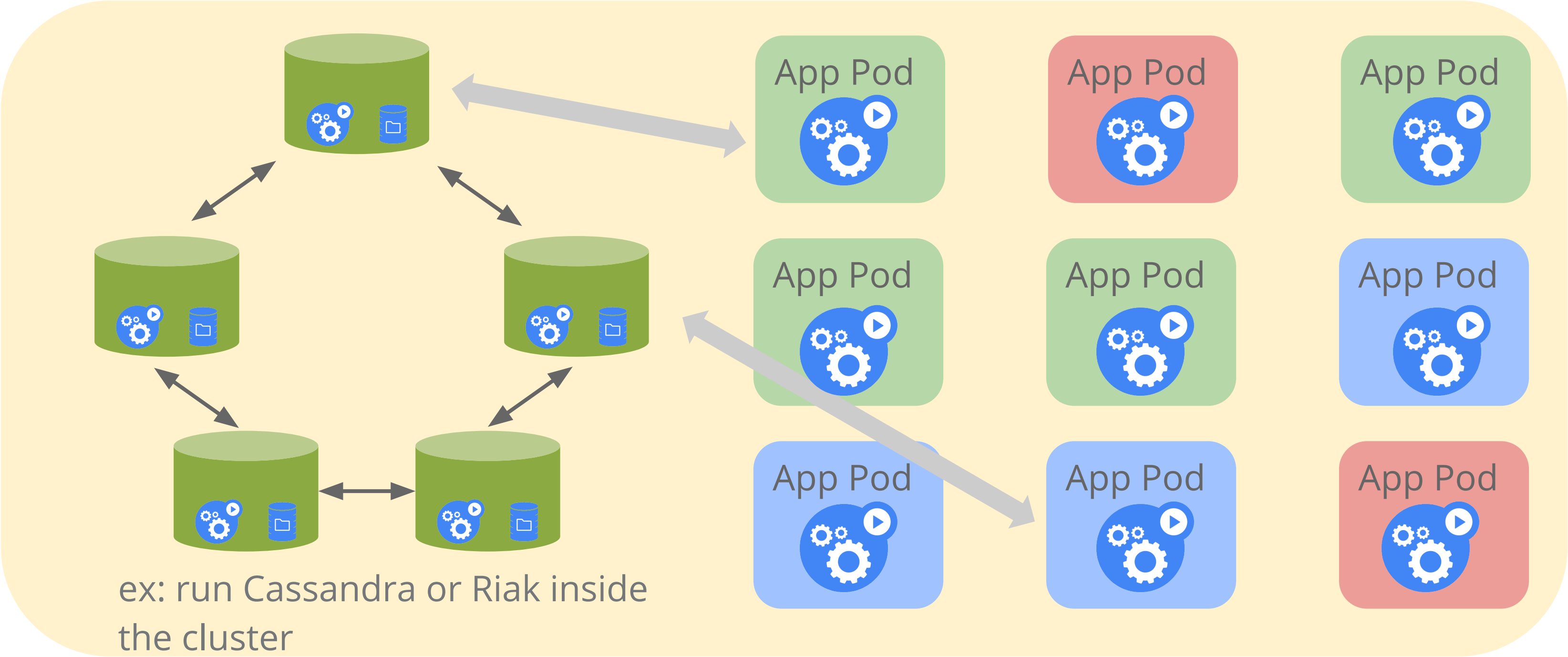
e.g.: MySQL managed by DBAs or managed cloud services

Adapt to run in the Cluster



e.g.: MySQL runs in a pod and mounts a filesystem provided by the cluster

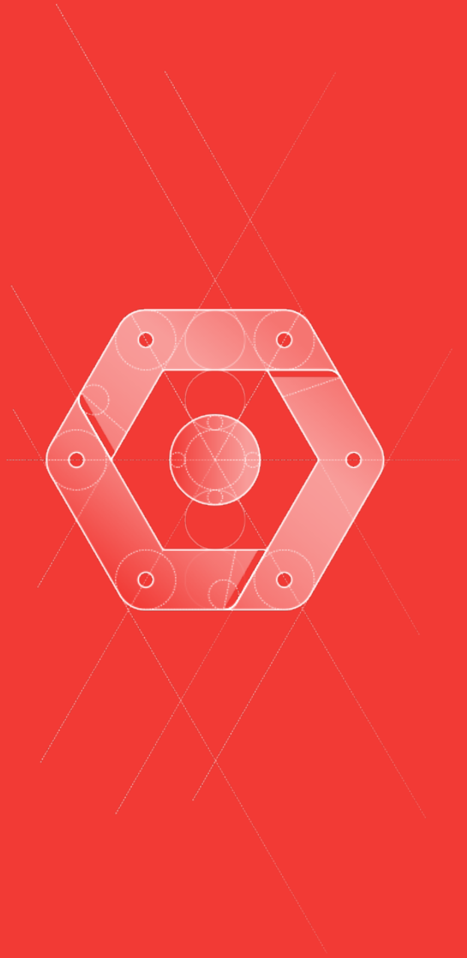
Cluster Native



ex: run Cassandra or Riak inside the cluster

Demo

Container Engine



Google Container Engine (Beta)

Managed Kubernetes (Kubernetes v1)

Manages Kubernetes master uptime

Manages Updates

Cluster Resize via Managed Instance Groups

Centralised Logging

Google Cloud VPN support

Kubernetes Status

Kubernetes 1.0 as of mid July

- Formerly announced at OSCON this week

Open sourced in June, 2014

- won the BlackDuck “rookie of the year” award

Google launched **Google Container Engine** (GKE)

- hosted Kubernetes
- https://cloud.google.com/container-engine/

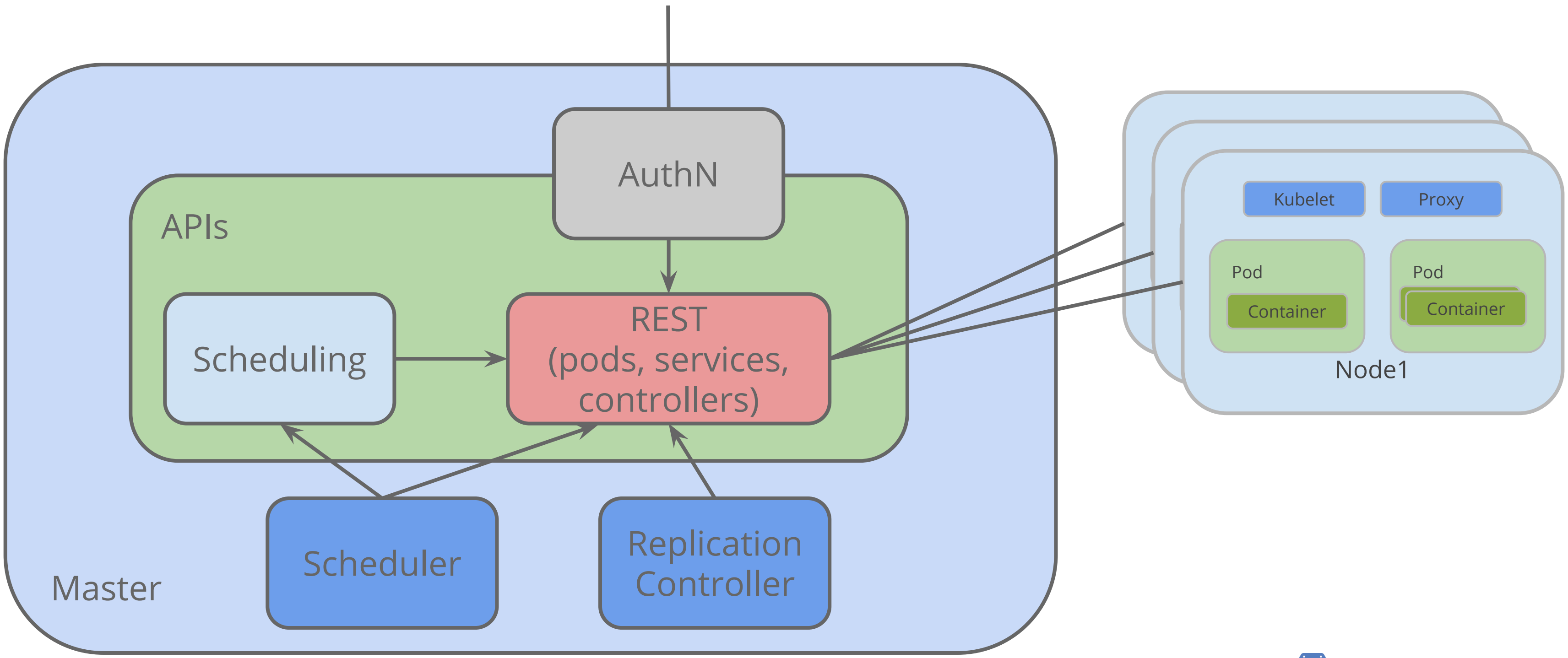
Roadmap:

- https://github.com/GoogleCloudPlatform/kubernetes/milestones

Demo - Visualization

Visualizing Kubernetes

```
$ kubectl proxy --www=k8s-visualizer/
```



#kubernetes @tekgrrl

Open Container Initiative

why argue about the width of train tracks, when you can worry about laying track and building the best possible engines?

Kubernetes is **Open Source**

We want your help!

http://kubernetes.io

https://github.com/GoogleCloudPlatform/kubernetes

irc.freenode.net *#google-containers*

@kubernetesio

Tweet questions to:
@tekgrrl

Questions