Kubernetes &
Google Container Engine Overview

DevOps Meetup Singapore

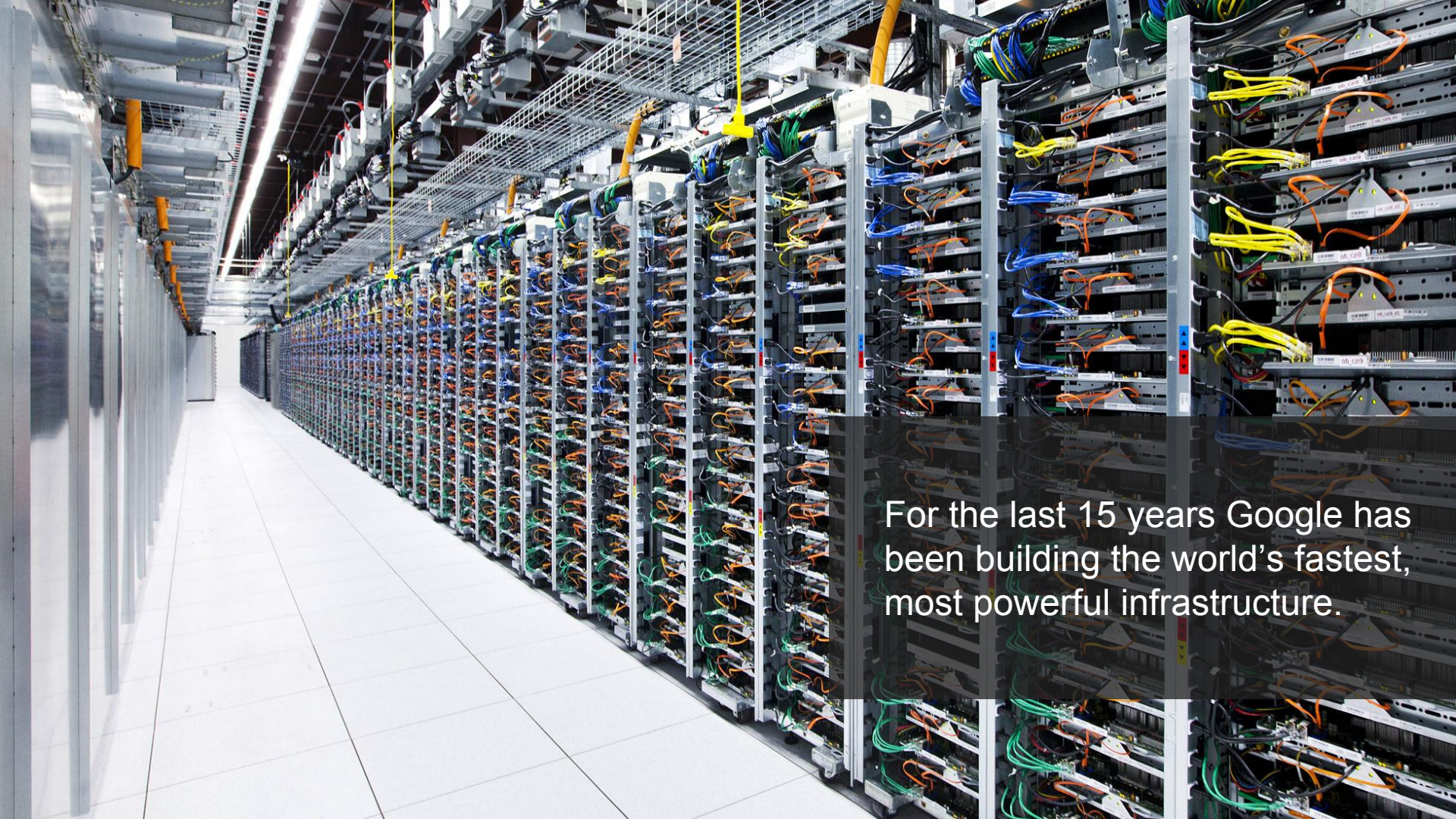
Google Cloud Platform

Ian Lewis

Developer Advocate - Google Cloud Platform

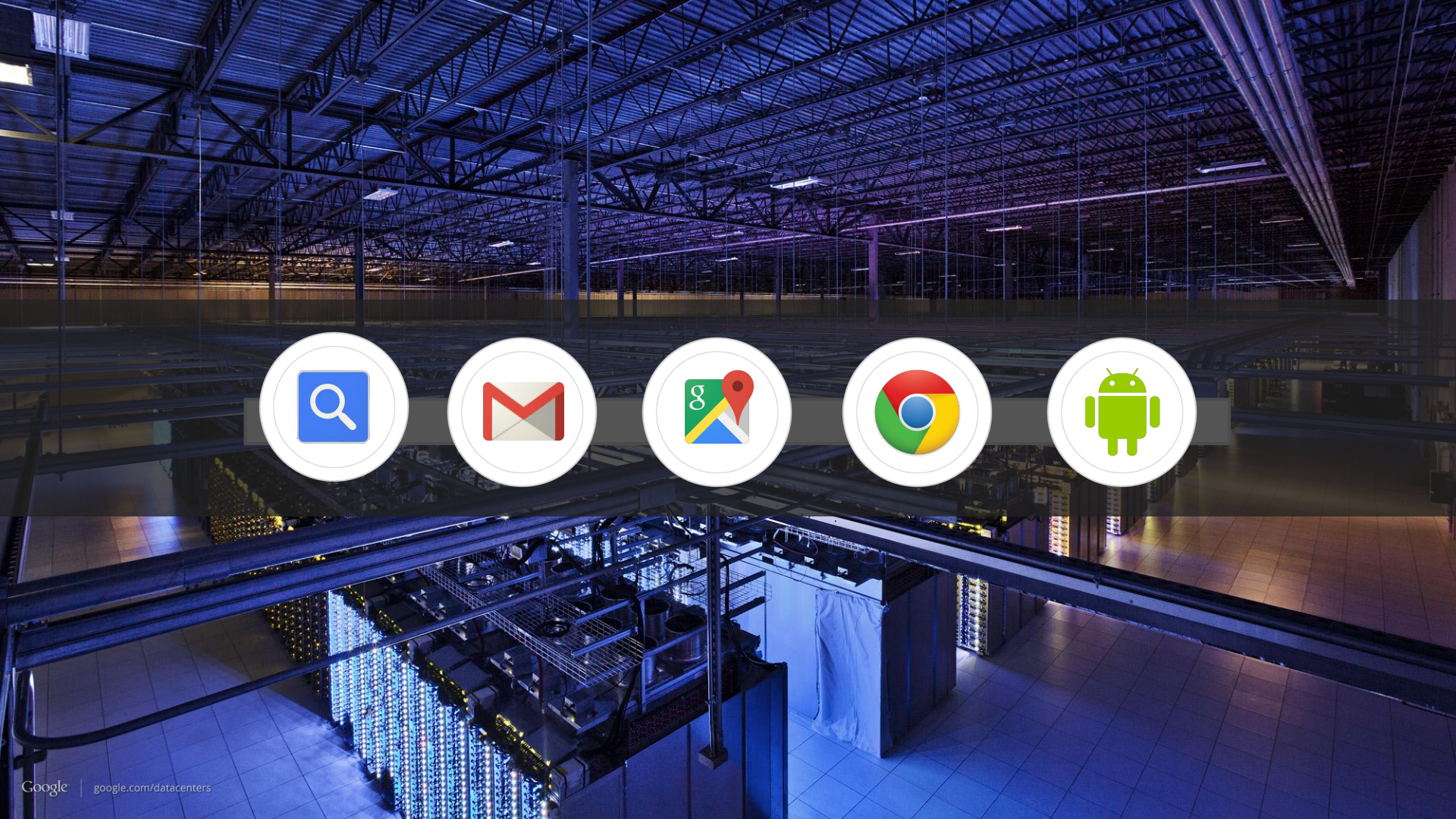
Tokyo, Japan

@IanMLewis


For the last 15 years Google has been building the world’s fastest, most powerful infrastructure.

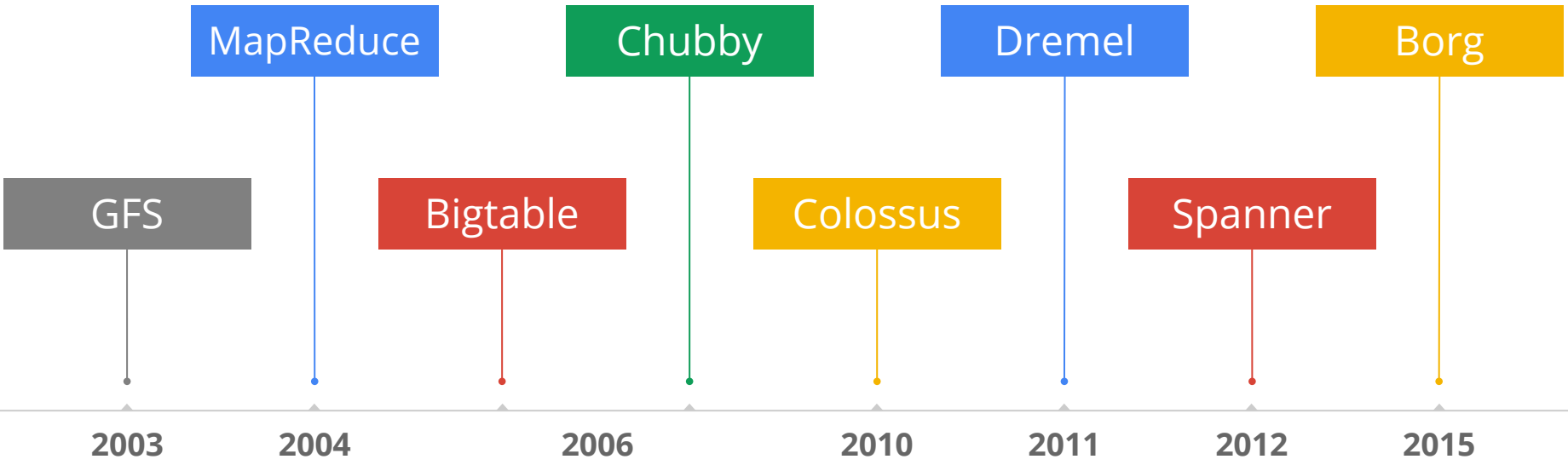
Google’s World Spanning Backbone Network

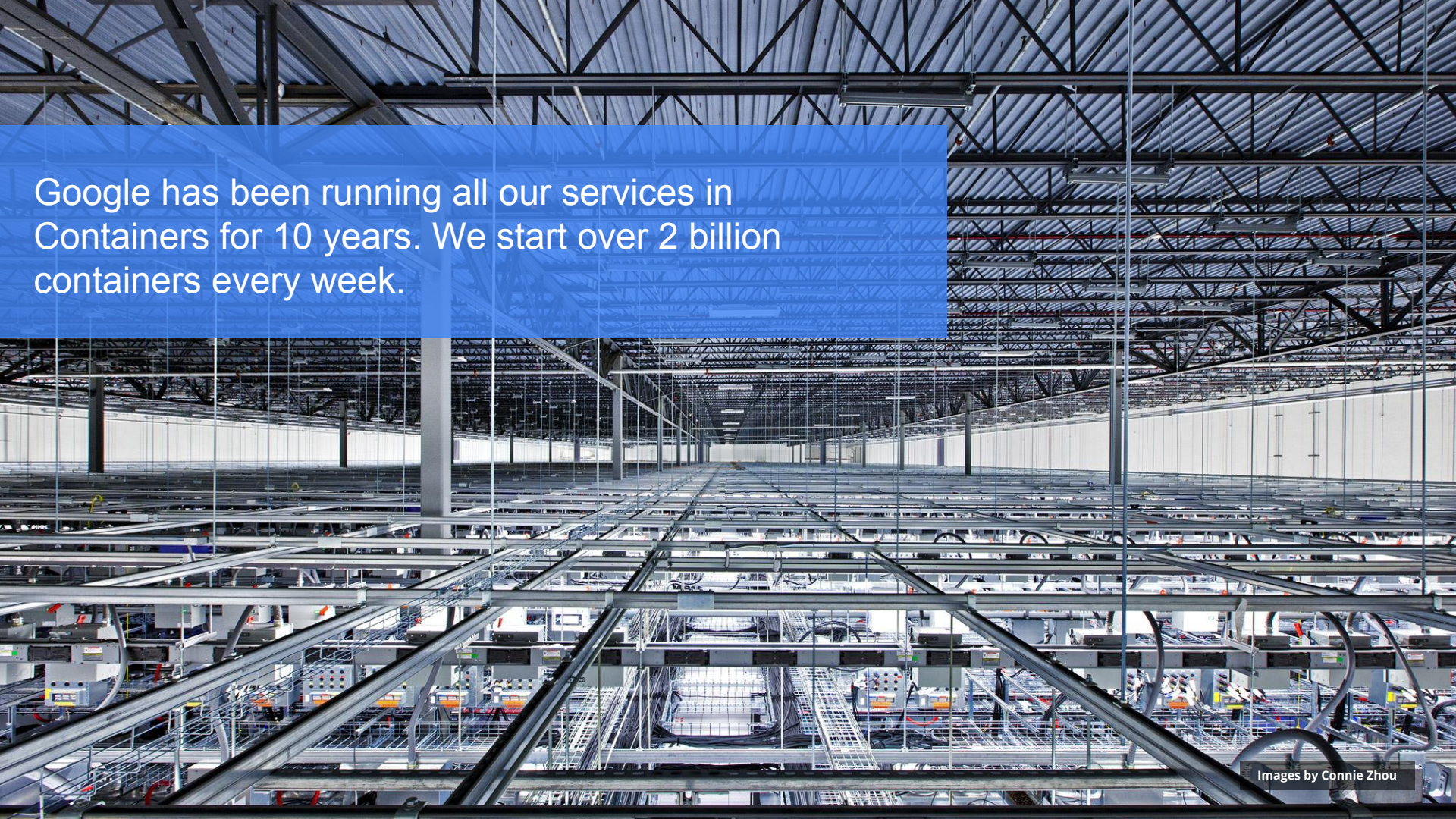
Google Cloud Platform


Google | google.com/datacenters


Google-Grade Networking
33 Countries
70 Edge Locations
The most of any Cloud Provider

Google Cloud Platform


GFS
MapReduce
Bigtable
Chubby
Colossus
Dremel
Spanner
Borg
2003
2004
2006
2010
2011
2012
2015


Google has been running all our services in Containers for 10 years. We start over 2 billion containers every week.
Images by Connie Zhou

Large-scale cluster management at Google with Borg

Abhishek Verma† Luis Pedrosa‡ Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes

Google Inc.

http://research.google.com/pubs/pub43438.html

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture

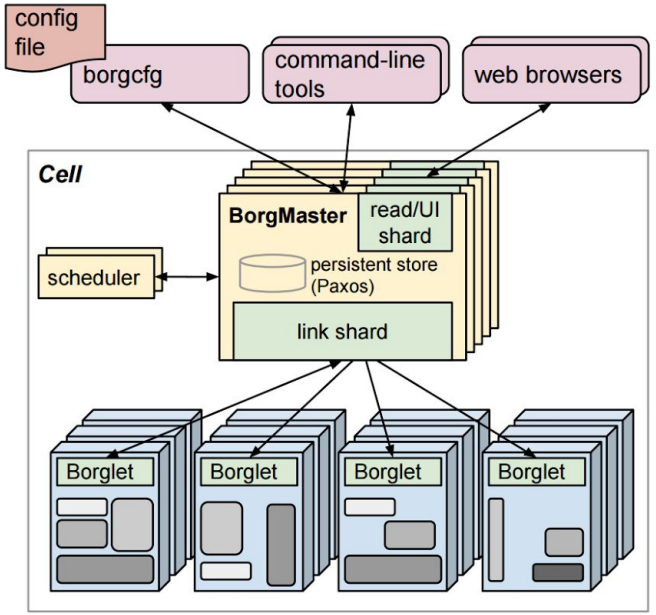



Image by Connie
Zhou

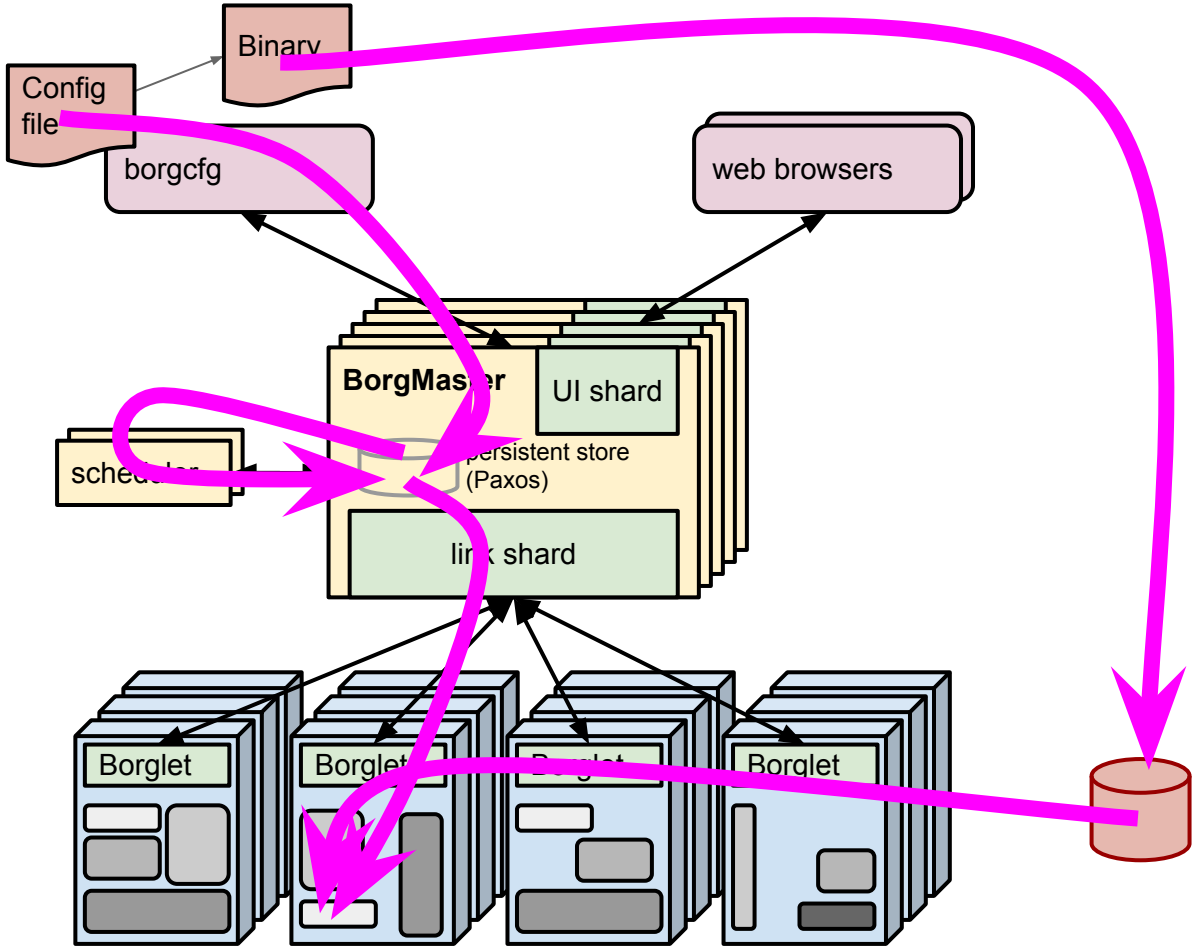
Developer View

```
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?



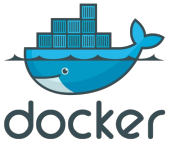

Hello world!
Image by Connie Zhou

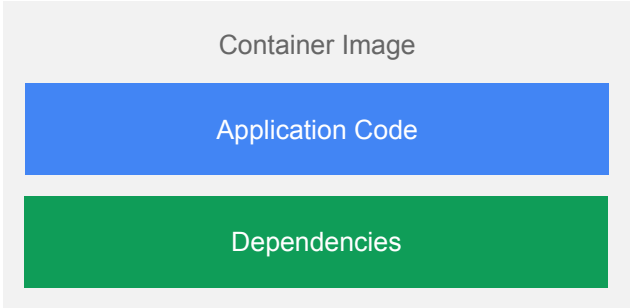
What are Containers?

Containers encapsulate application code and all dependencies.

Applications can be depend less on the infrastructure where it runs.

- In traditional IT environments, applications needed specific infrastructure. Dependencies needed to be installed beforehand.
- Containers incorporate applications and their dependencies so deployment to development, test, and production can be made easier.
- Don’t need to be dependent on on-premise, private or public cloud environments.





Why Containers?

Fast

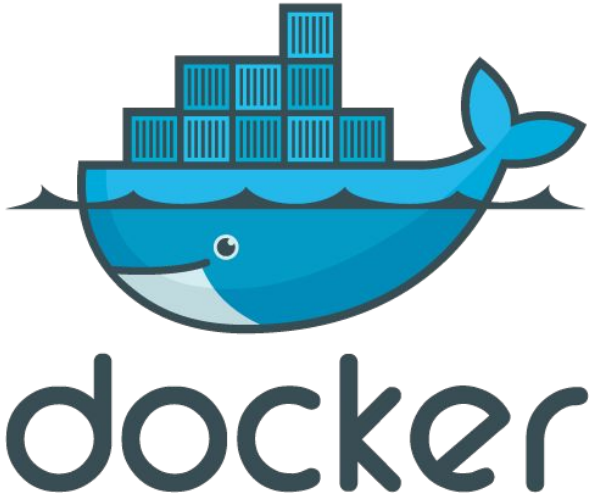
Simple and Fast compared to VMs. Can be started in just a few milliseconds.

Portable

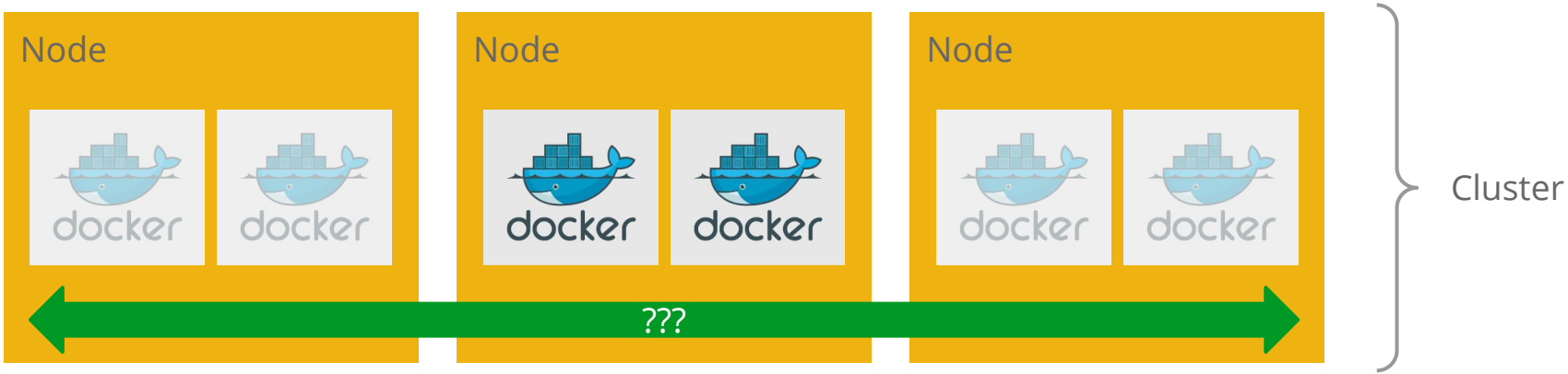
Can be run in a many environments.

Efficiency

Low overhead. Resources use by containers can be limited.


docker

Container Management



- How to deploy to multiple nodes?
- How to deal with node failures?
- How to deal with container failures?
- How do you update your applications?

Kubernetes

κυβερνήτης: *Greek for “pilot” or “helmsman of a ship”*
the open source cluster manager from Google

CNCF(Cloud Native Computing Foundation)

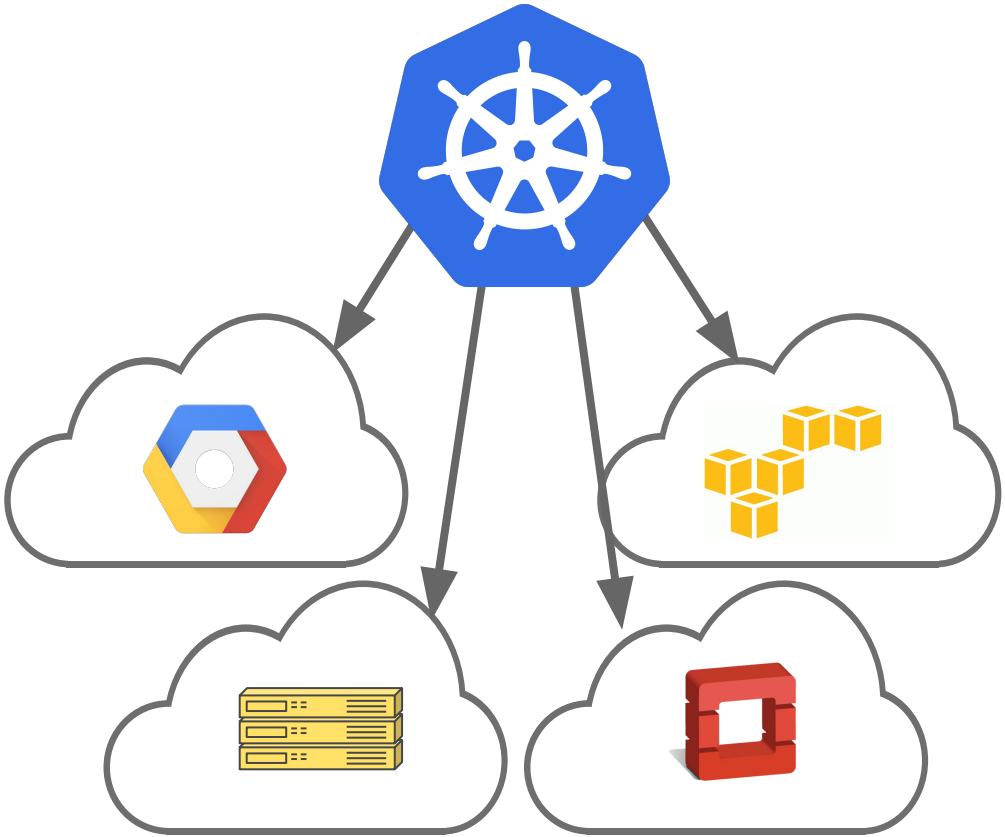
Workload portability

Goal: Avoid vendor lock-in

Runs in many environments, including “bare metal” and “your laptop”

The API and the implementation are 100% open

The whole system is modular and replaceable



Workload portability

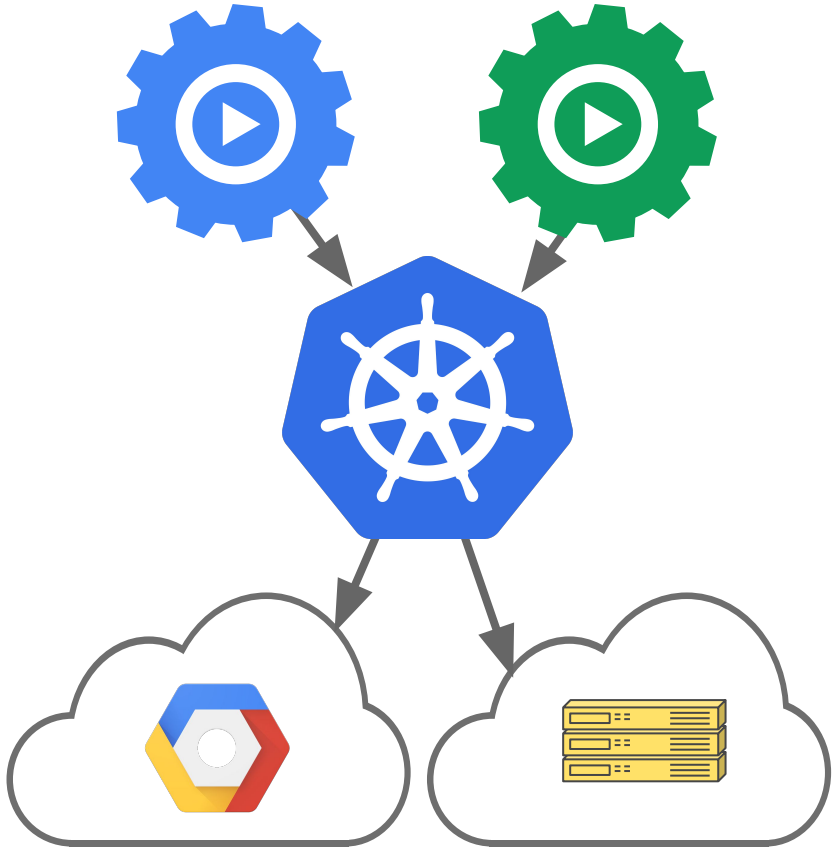
Goal: Write once, run anywhere*

Don’t force apps to know about concepts that are cloud-provider-specific

Examples of this:

- Network model
- Ingress
- Service load-balancers
- PersistentVolumes

** approximately*



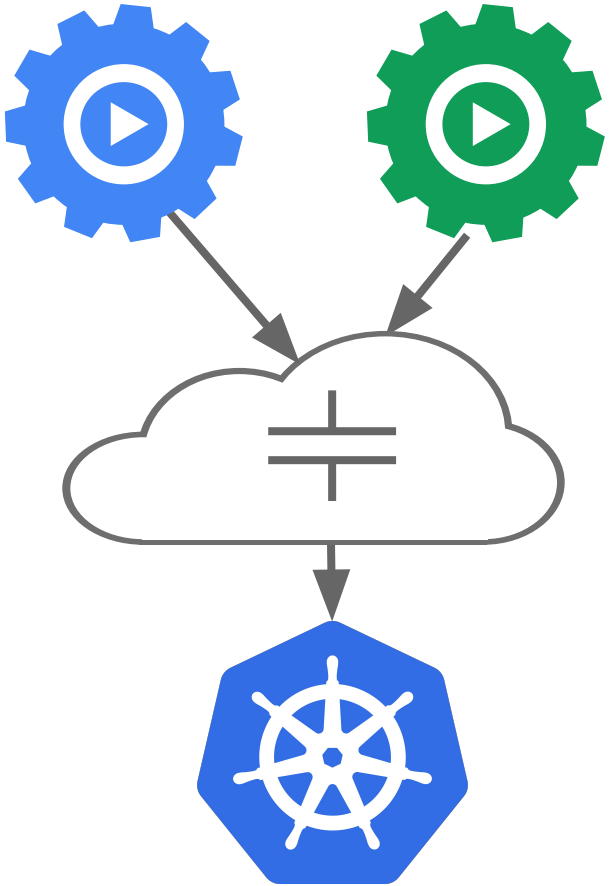
Workload portability

Goal: Avoid coupling

Don’t force apps to know about concepts that are Kubernetes-specific

Examples of this:

- Namespaces
- Services / DNS
- Downward API
- Secrets / ConfigMaps



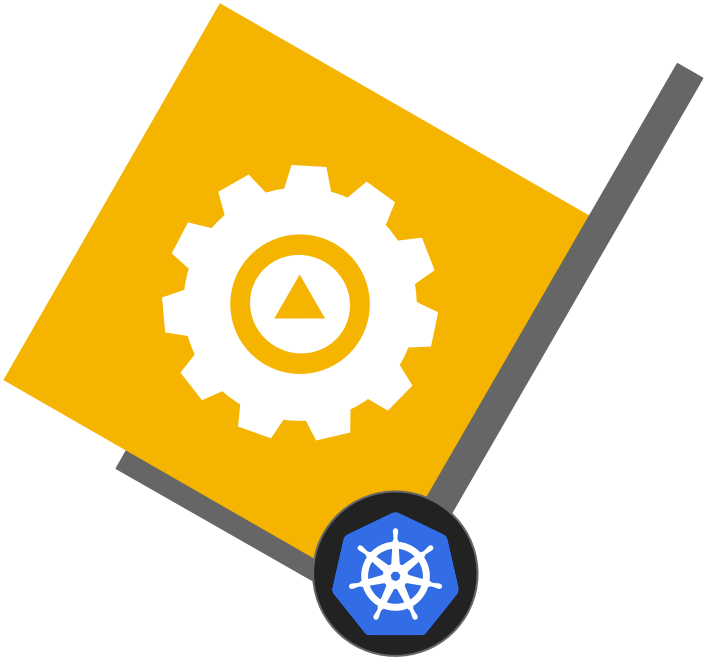
Workload portability

Result: Portability

Build your apps on-prem, lift-and-shift into cloud when you are ready

Don’t get stuck with a platform that doesn’t work for you

Put your app on wheels and move it whenever and wherever you need



Container Engine

Pods

Small group of containers & volumes

Tightly coupled

The atom of scheduling & placement

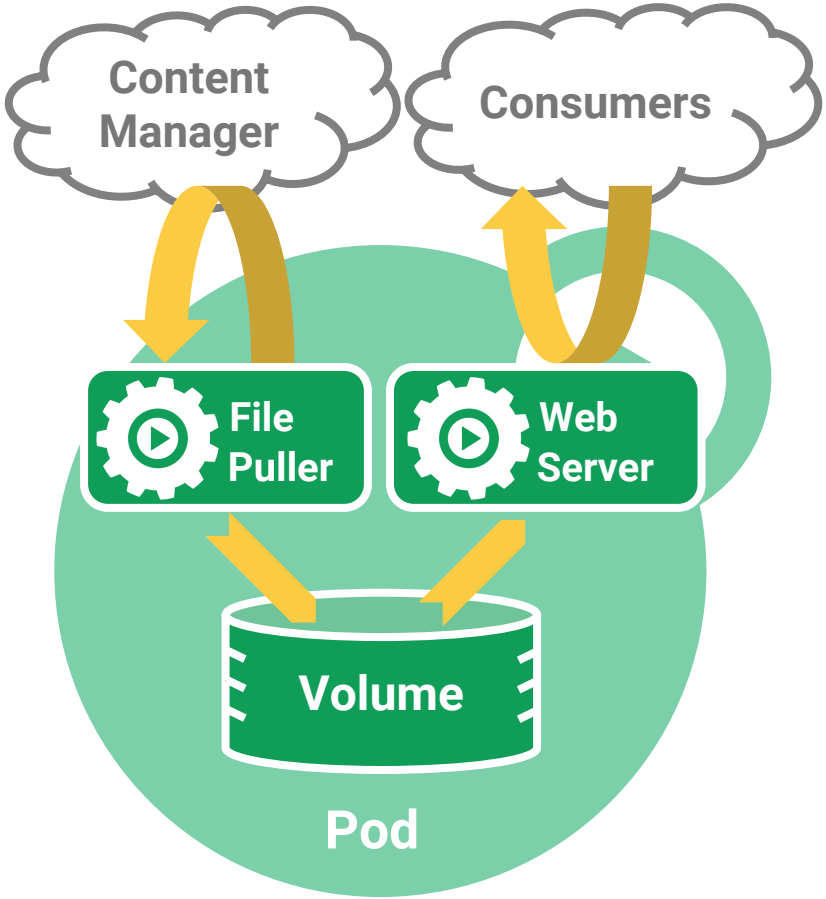
Shared namespace

- share IP address & localhost
- share IPC, etc.

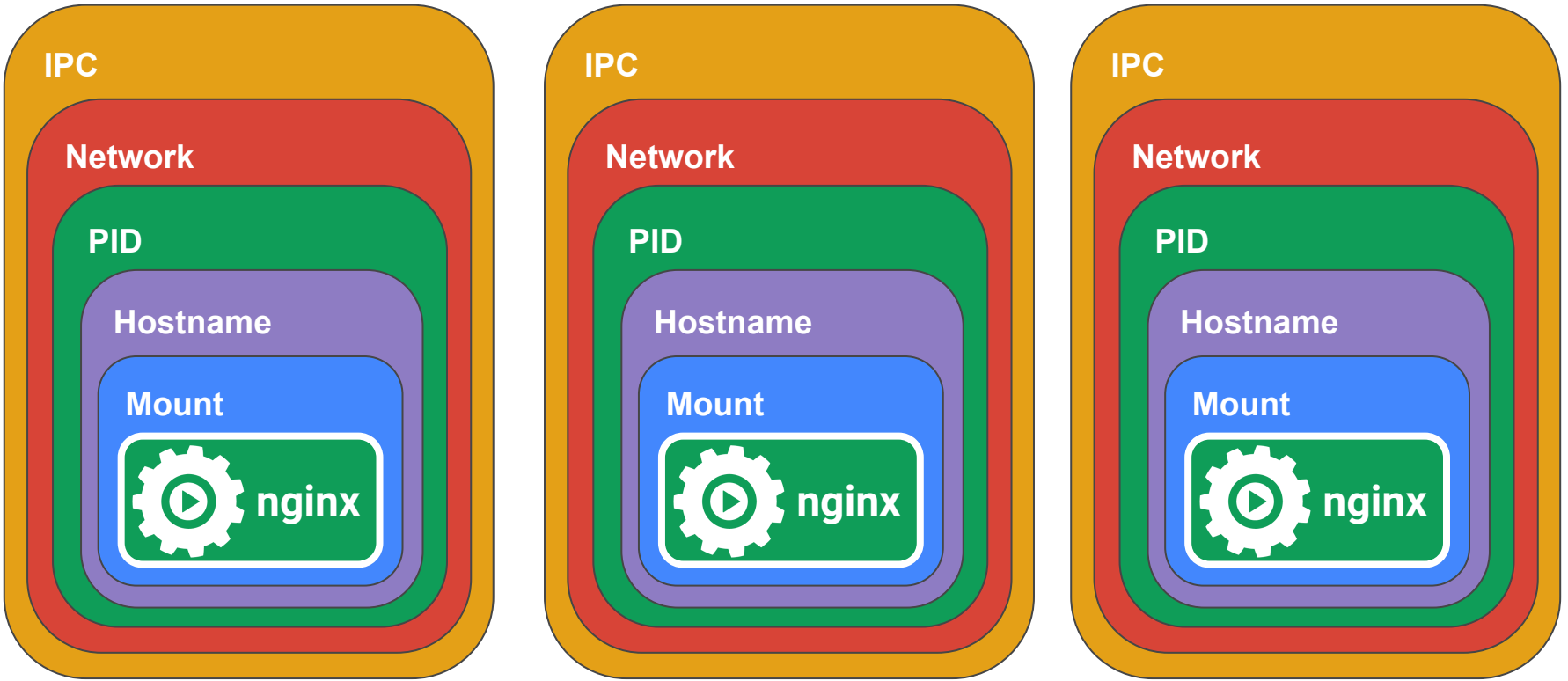
Managed lifecycle

- bound to a node, restart in place
- can die, cannot be reborn with same ID

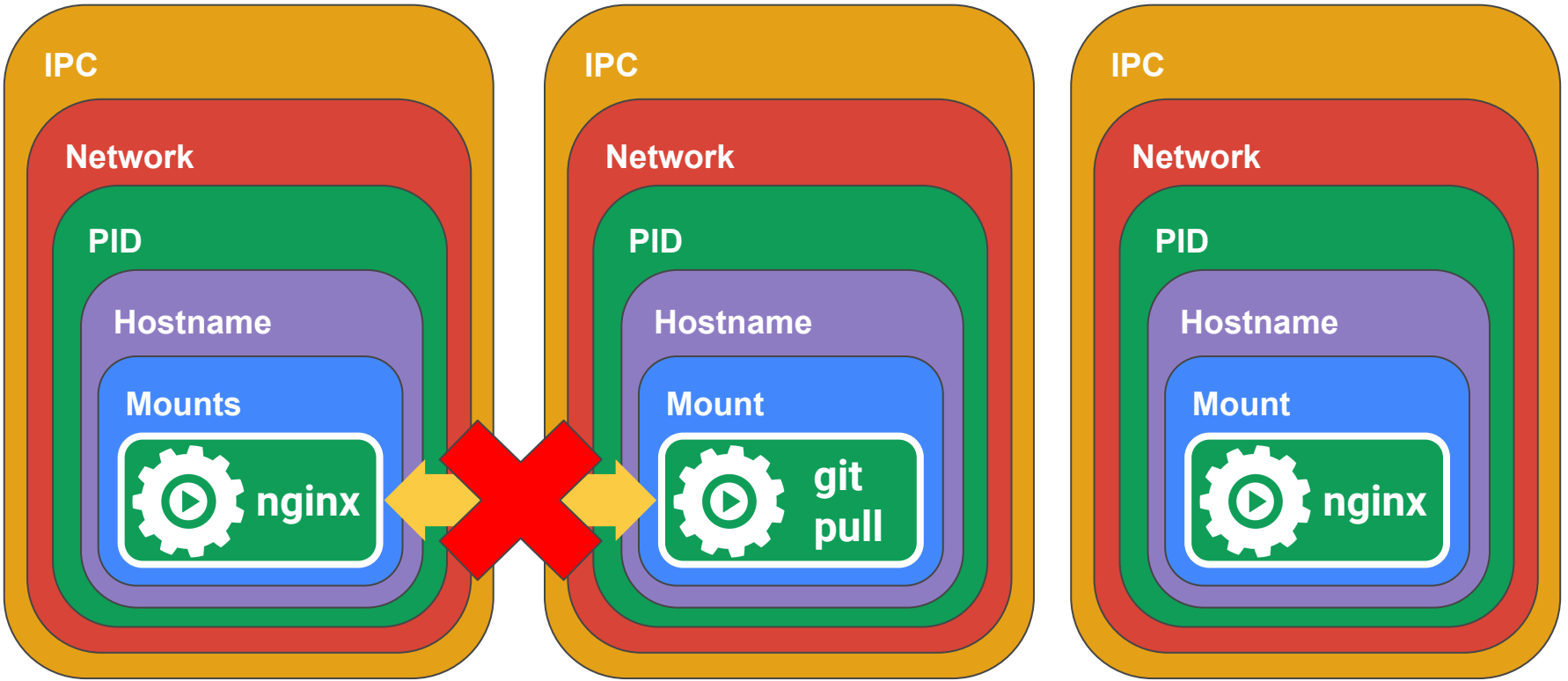
Example: data puller & web server



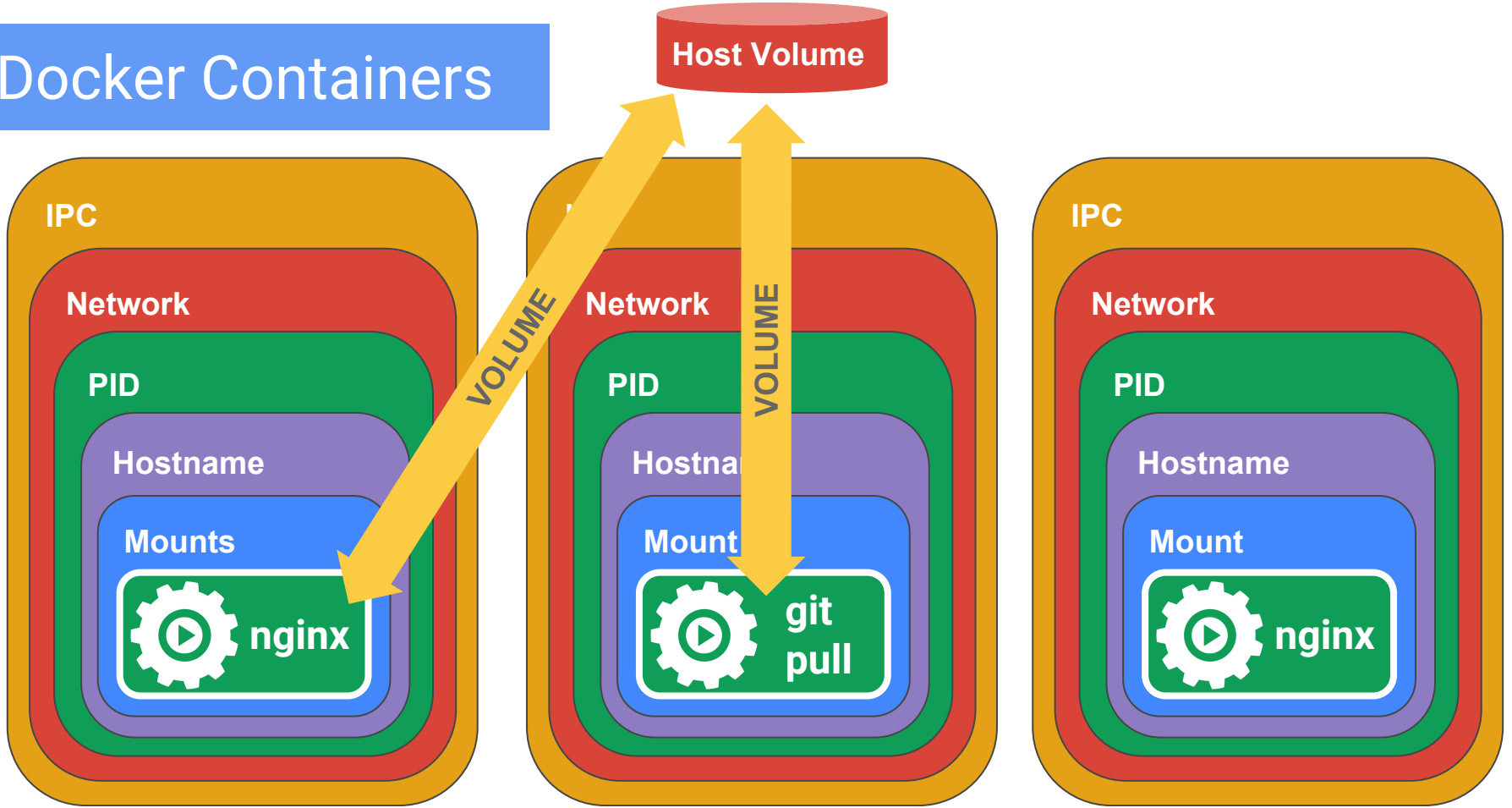
Docker Containers



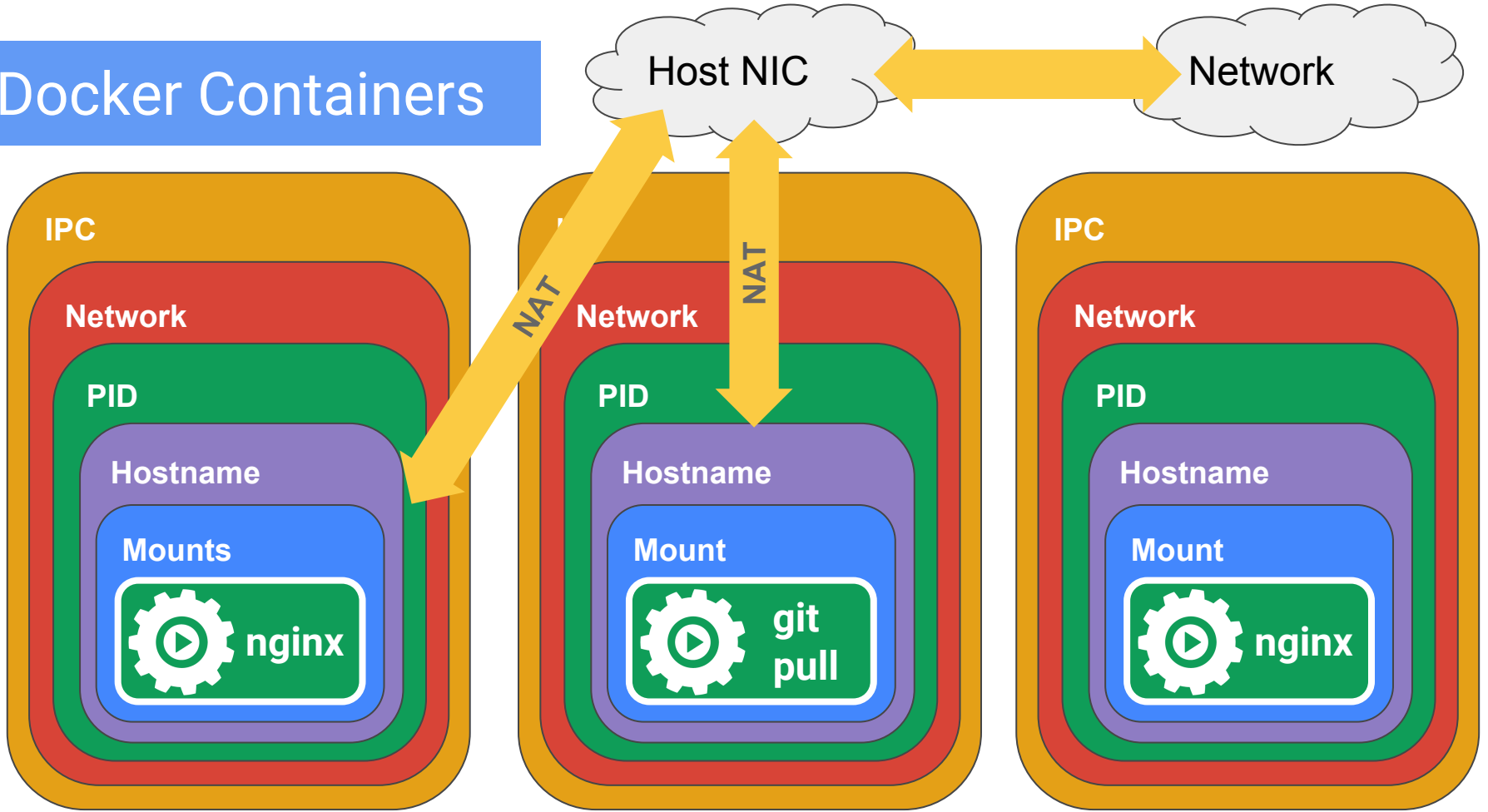
Docker Containers



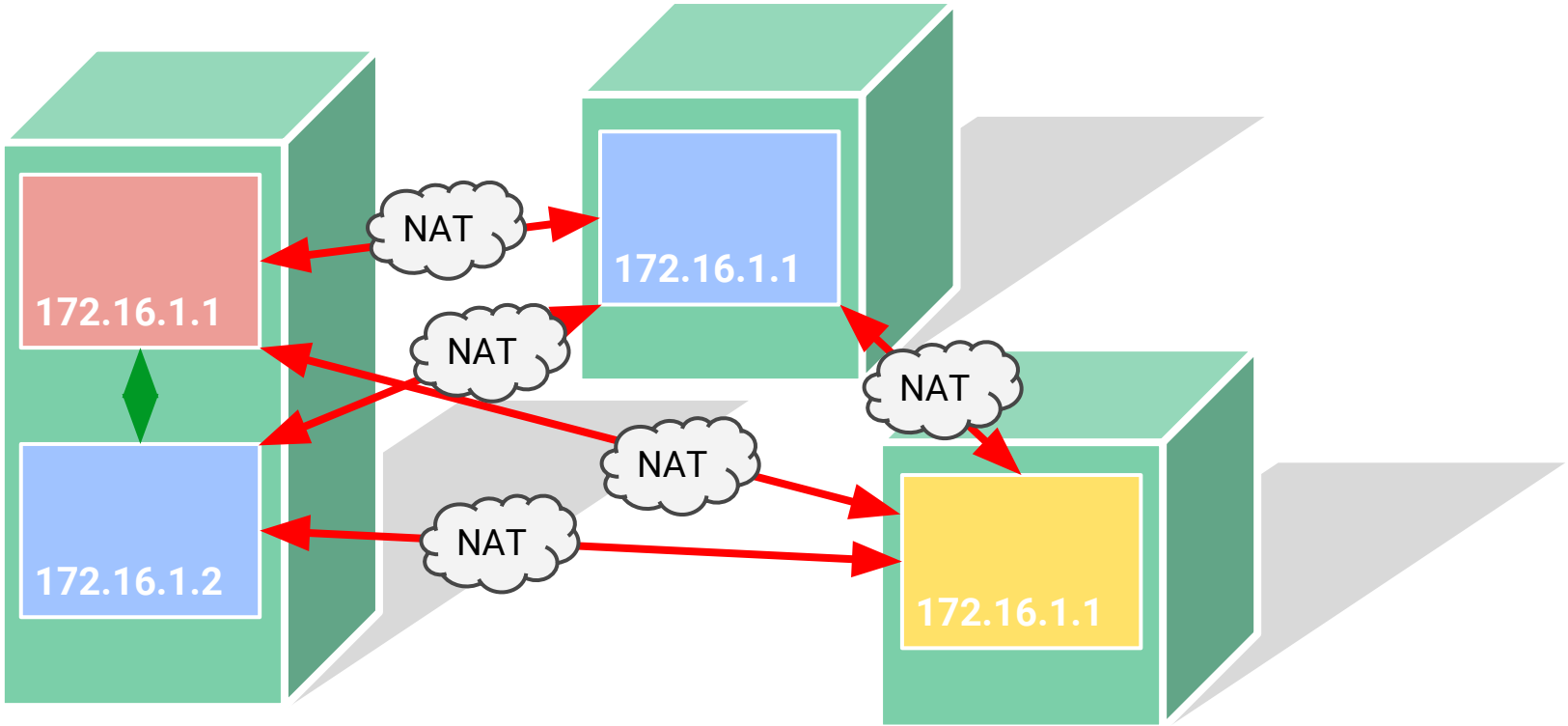
Docker Containers



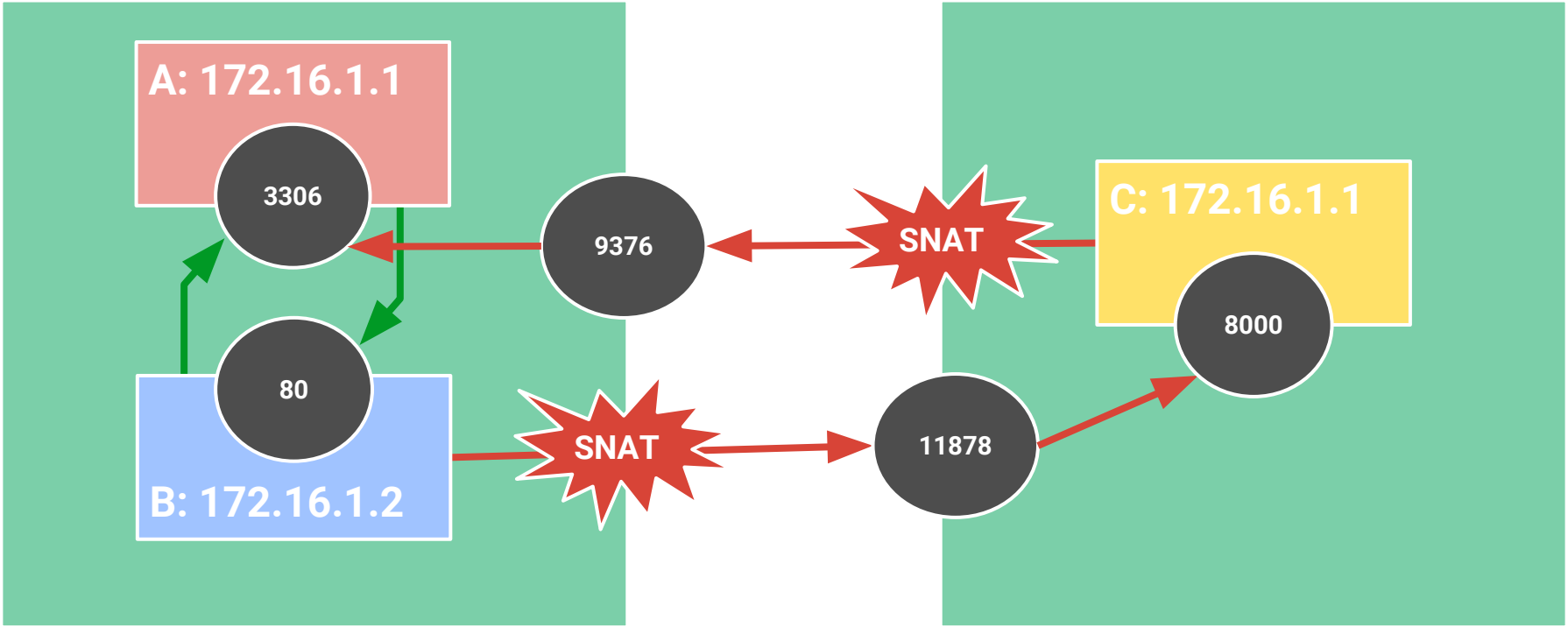
Docker Containers



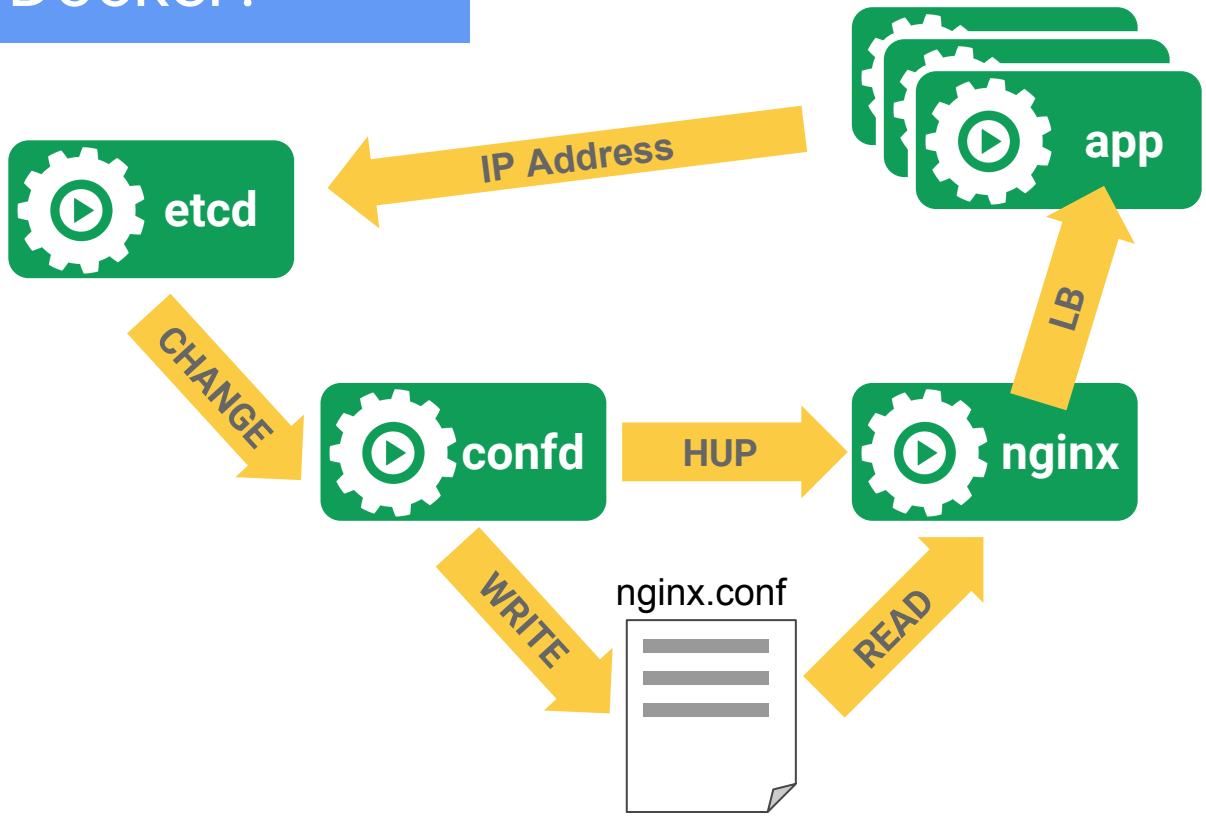
Docker networking



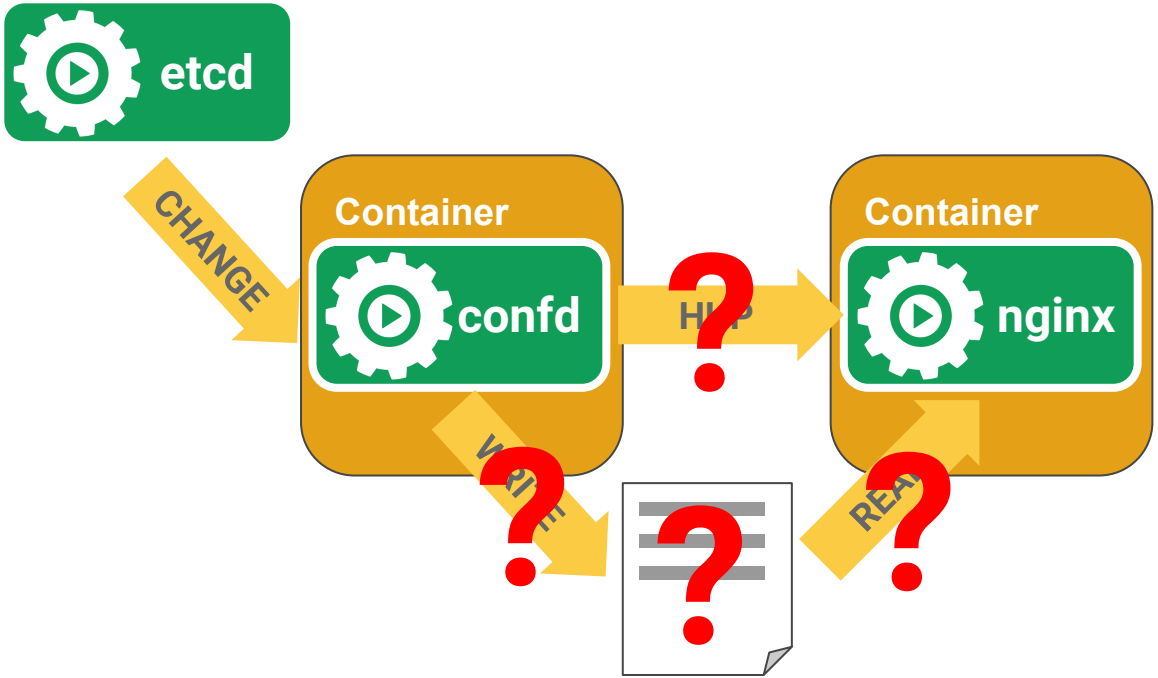
Port mapping



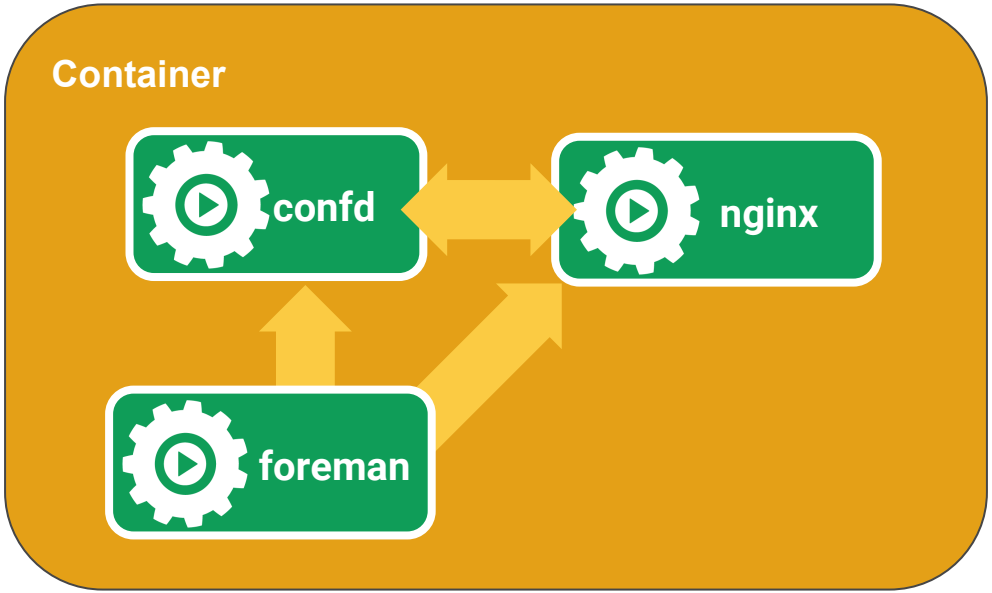
Pods & Docker?


app
IP Address
etcd
LB
CHANGE
confd
HUP
nginx
nginx.conf
WRITE
READ

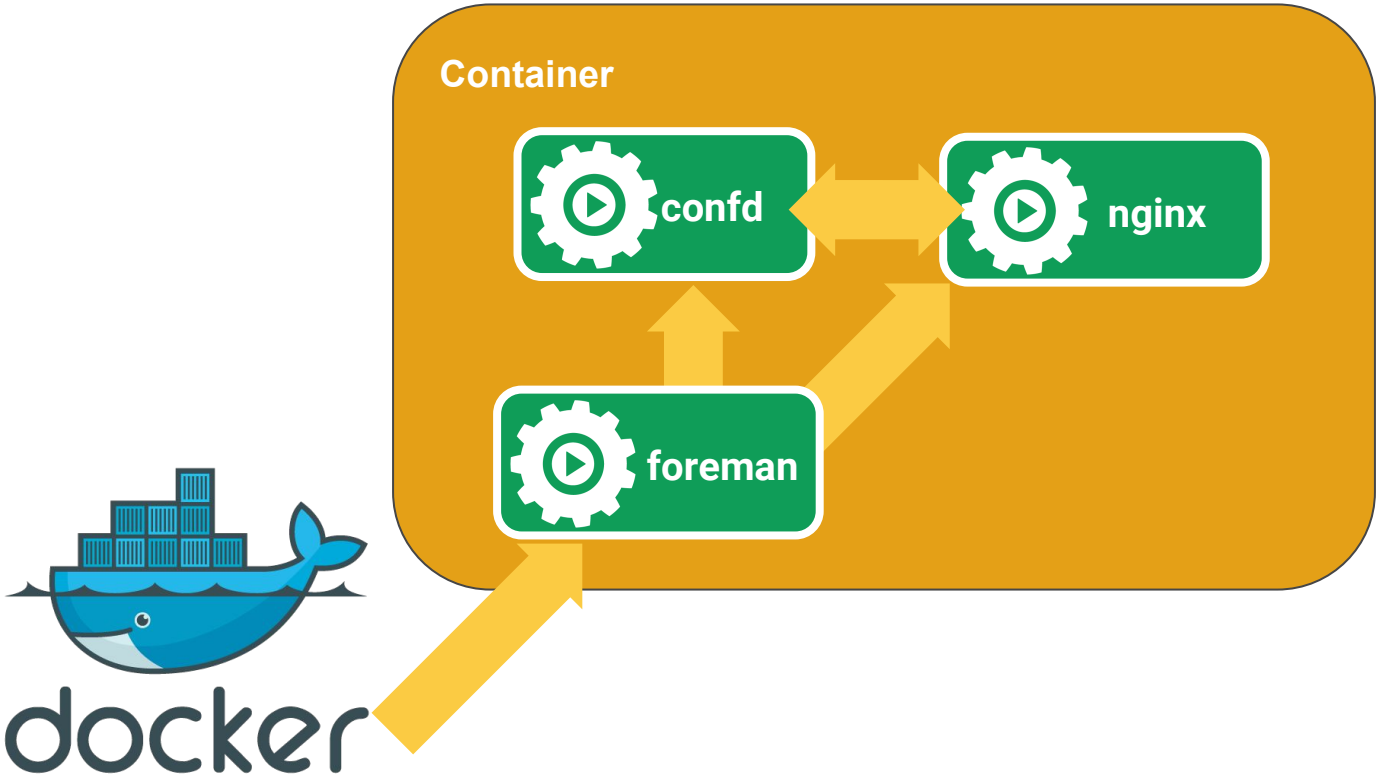
Pods & Docker?



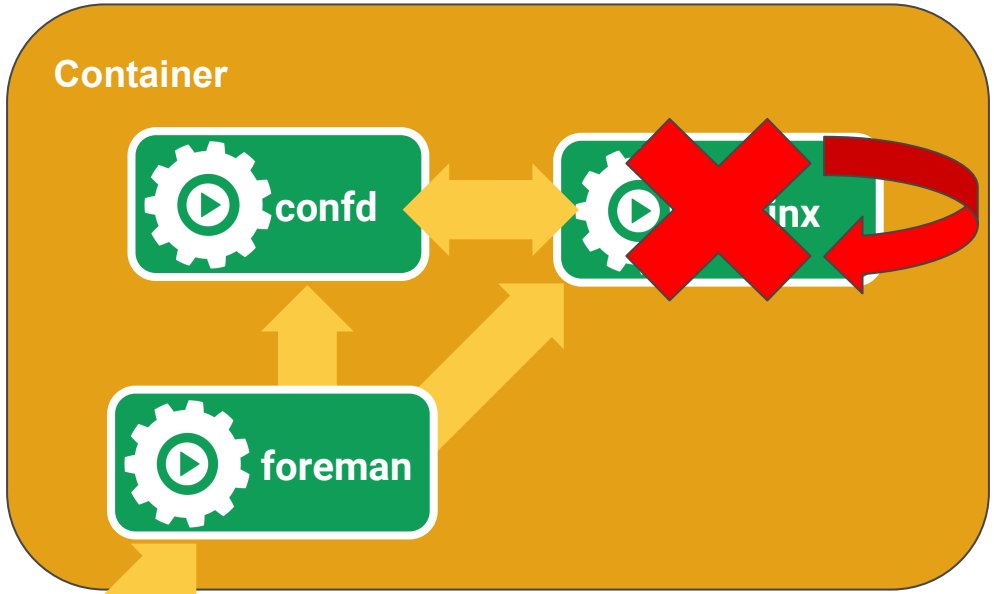
Pods & Docker?



Pods & Docker?

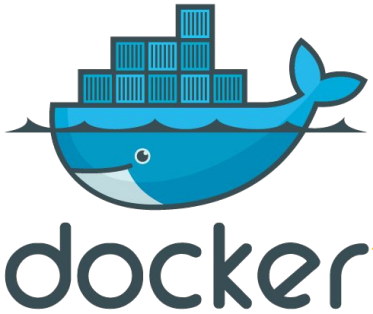


Pods & Docker?



Crash-Restart Loop

Everything's A-OK!!

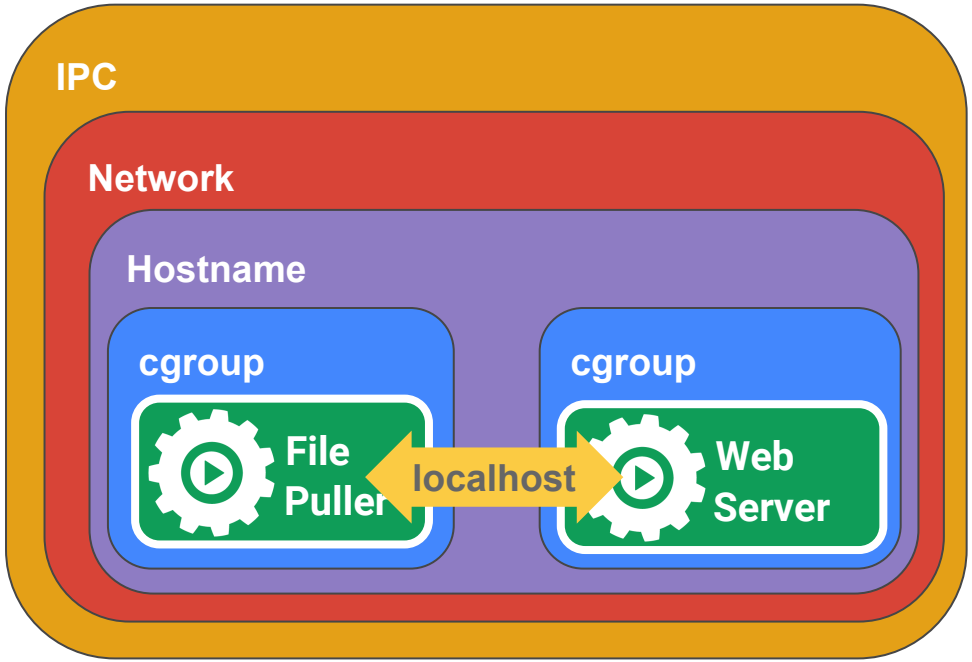


Pods

docker …

--net=container:id

--ipc=container:id



Pods (TODO)

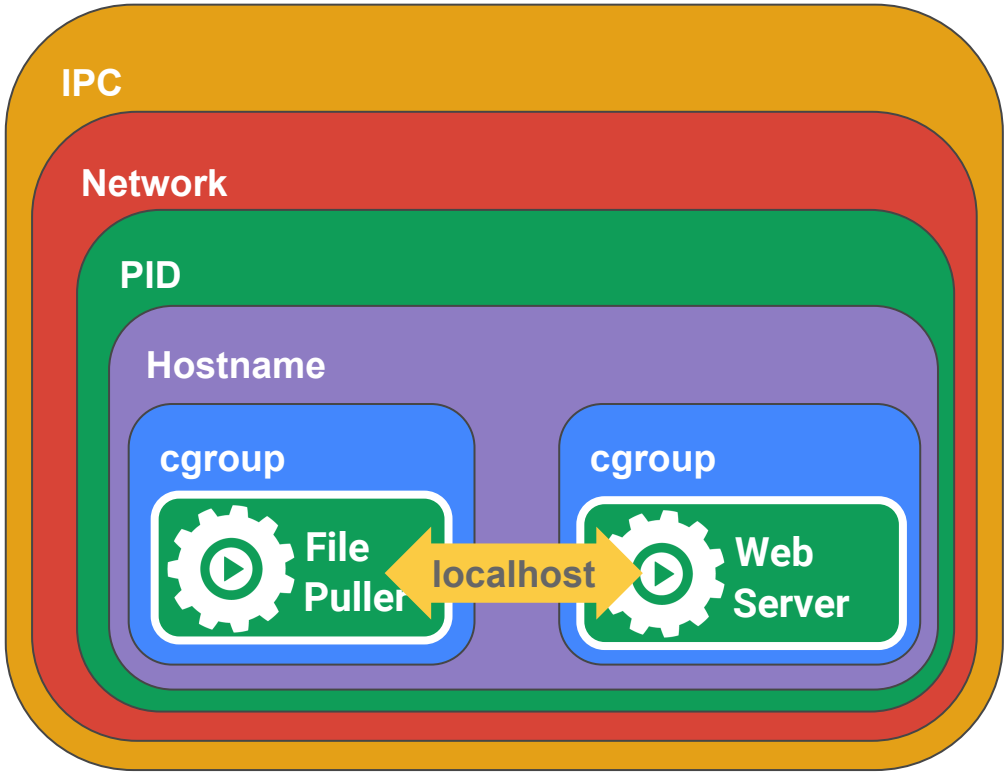
docker …

--net=container:id

--ipc=container:id

--pid=container:id

https://github.com/docker/docker/issues/10163



Kubernetes networking

IPs are **cluster-scoped**

- vs docker default private IP

Pods can reach each other directly

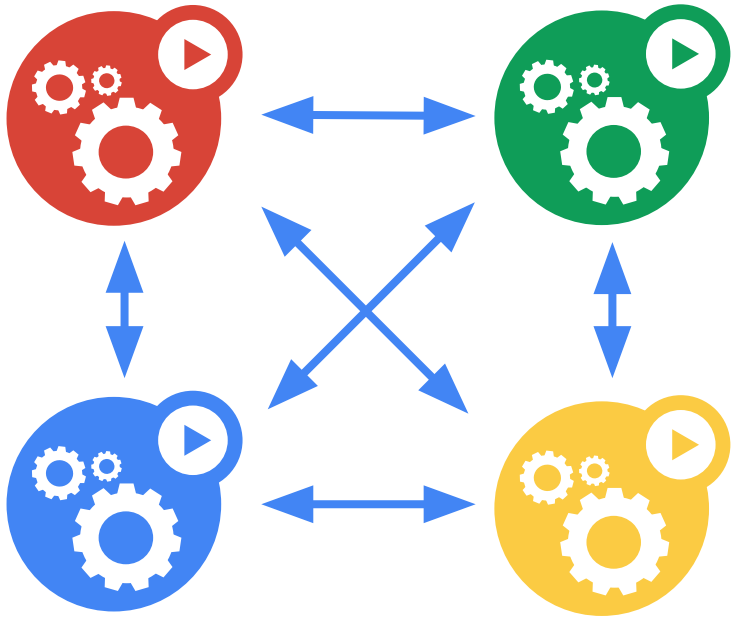
- even across nodes

No brokering of port numbers

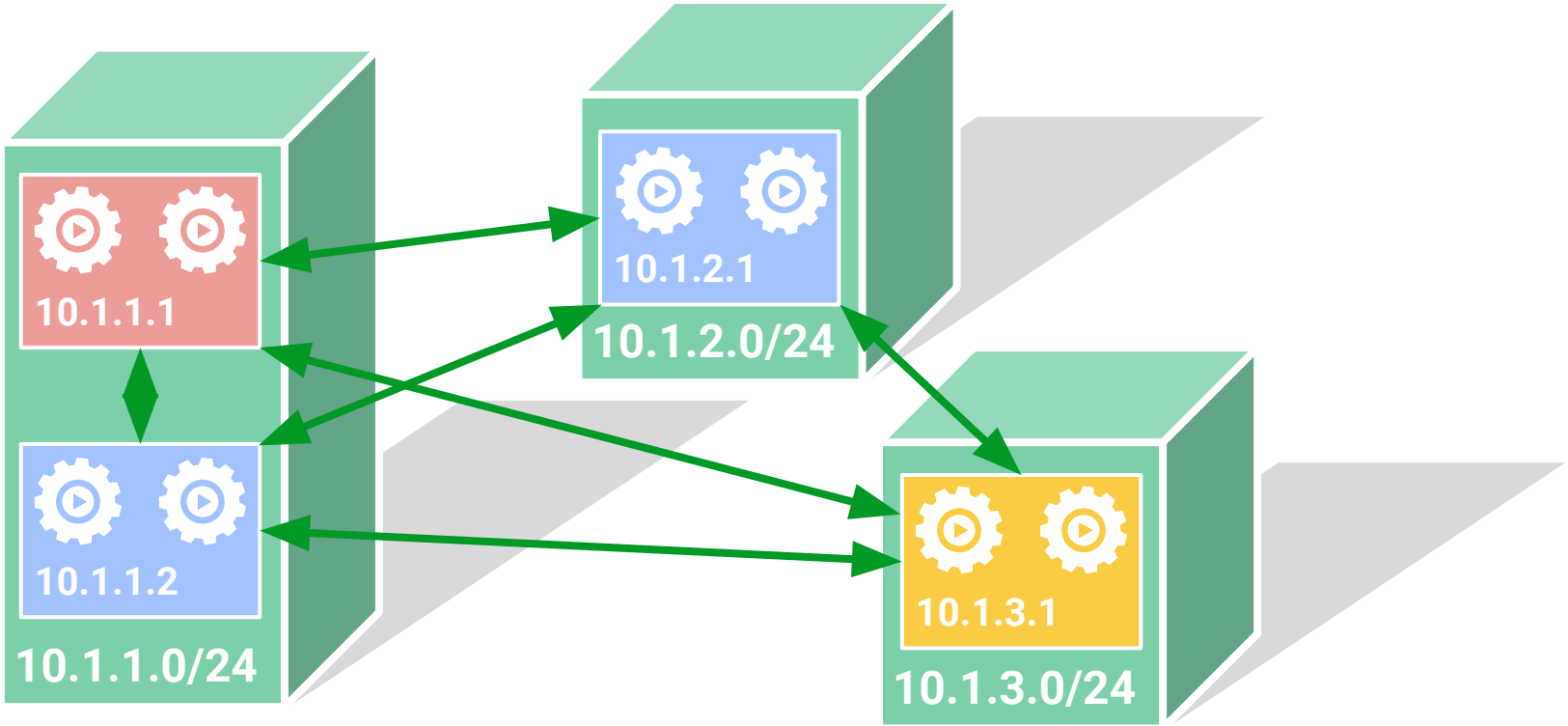
- too complex, why bother?

This is a fundamental requirement

- can be L3 routed
- can be underlayed (cloud)
- can be overlayed (SDN)



Kubernetes networking



ConfigMaps

Goal: manage app configuration

- ...without making overly-brittle container images

12-factor says config comes from the environment

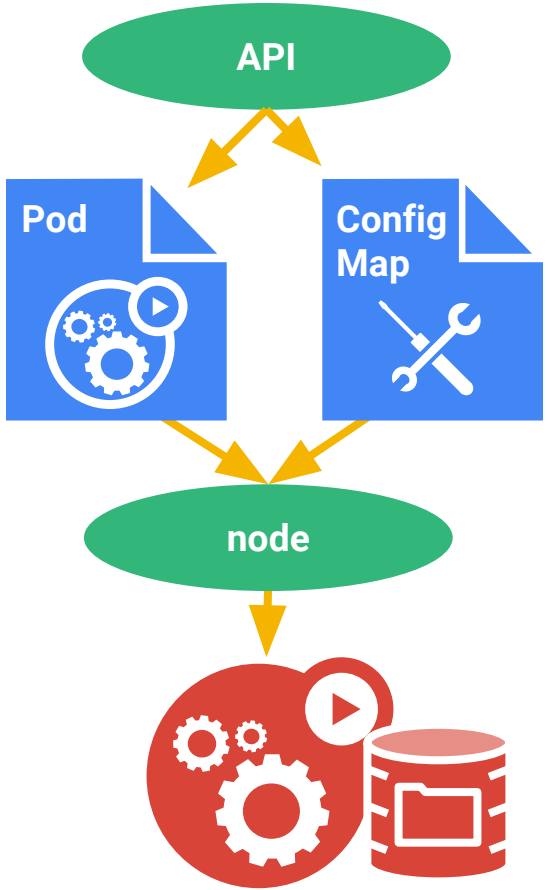
- Kubernetes is the environment

Manage config via the Kubernetes API

Inject config as a virtual volume into your Pods

- late-binding, live-updated (atomic)
- also available as env vars

Status: GA in Kubernetes v1.2



Secrets

Goal: grant a pod access to a secured *something*

- don't put secrets in the container image!

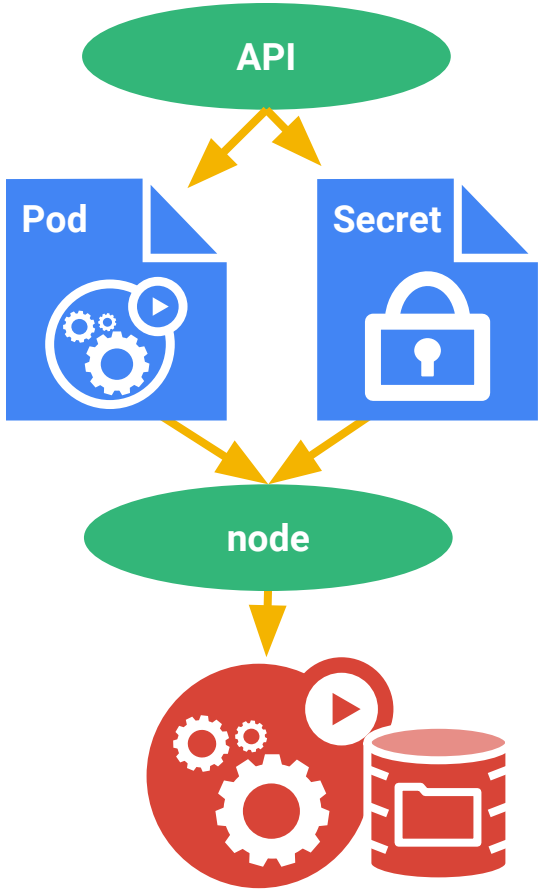
12-factor says config comes from the environment

- Kubernetes is the environment

Manage secrets via the Kubernetes API

Inject secrets as virtual volumes into your Pods

- late-binding, tmpfs - never touches disk
- also available as env vars



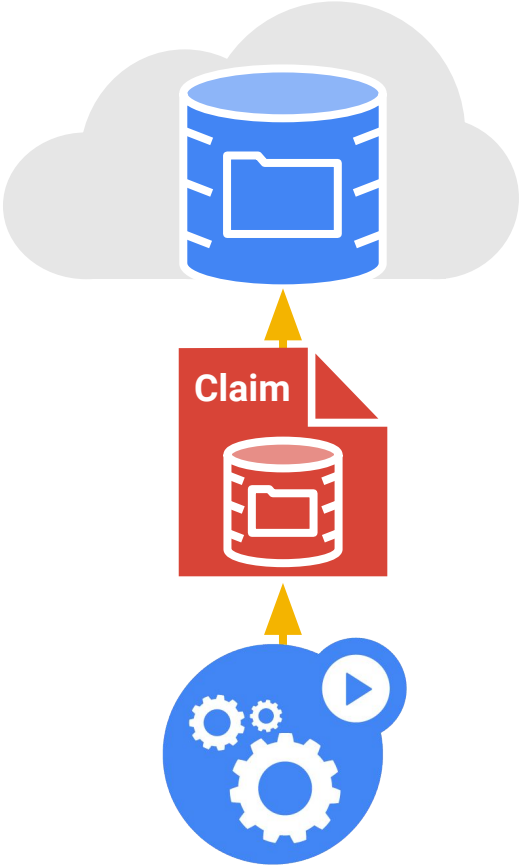
PersistentVolumes

A higher-level storage abstraction
- insulation from any one cloud environment

Admin provisions them, users claim them
- **NEW: auto-provisioning (alpha in v1.2)**

Independent lifetime from consumers
- lives until user is done with it
- can be handed-off between pods

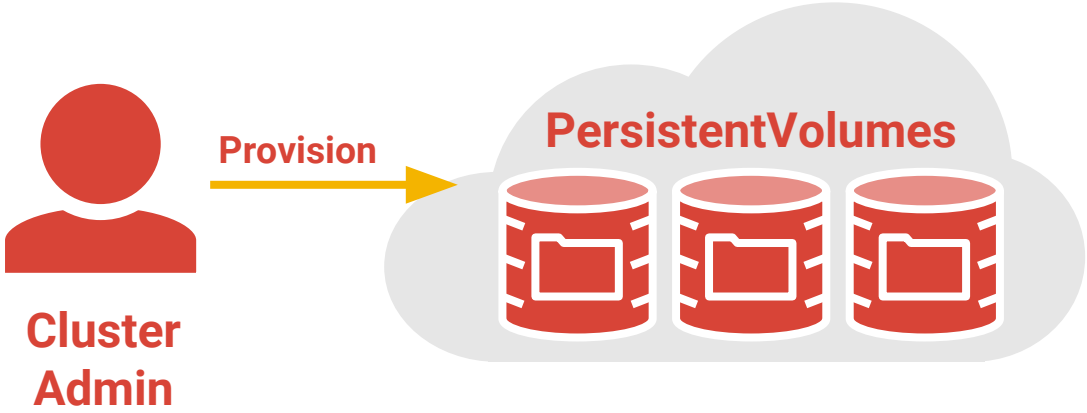
Dynamically “scheduled” and managed, like nodes and pods



PersistentVolumes

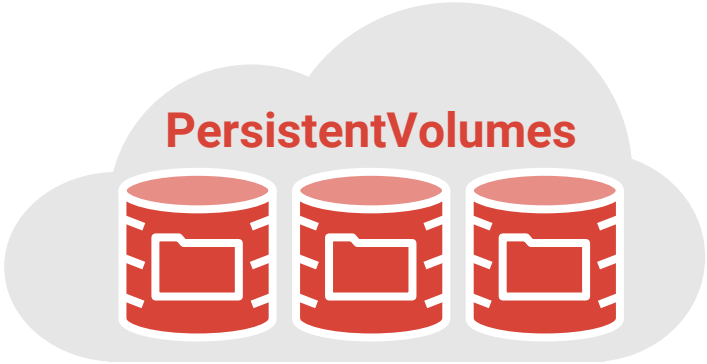
Cluster Admin

PersistentVolumes



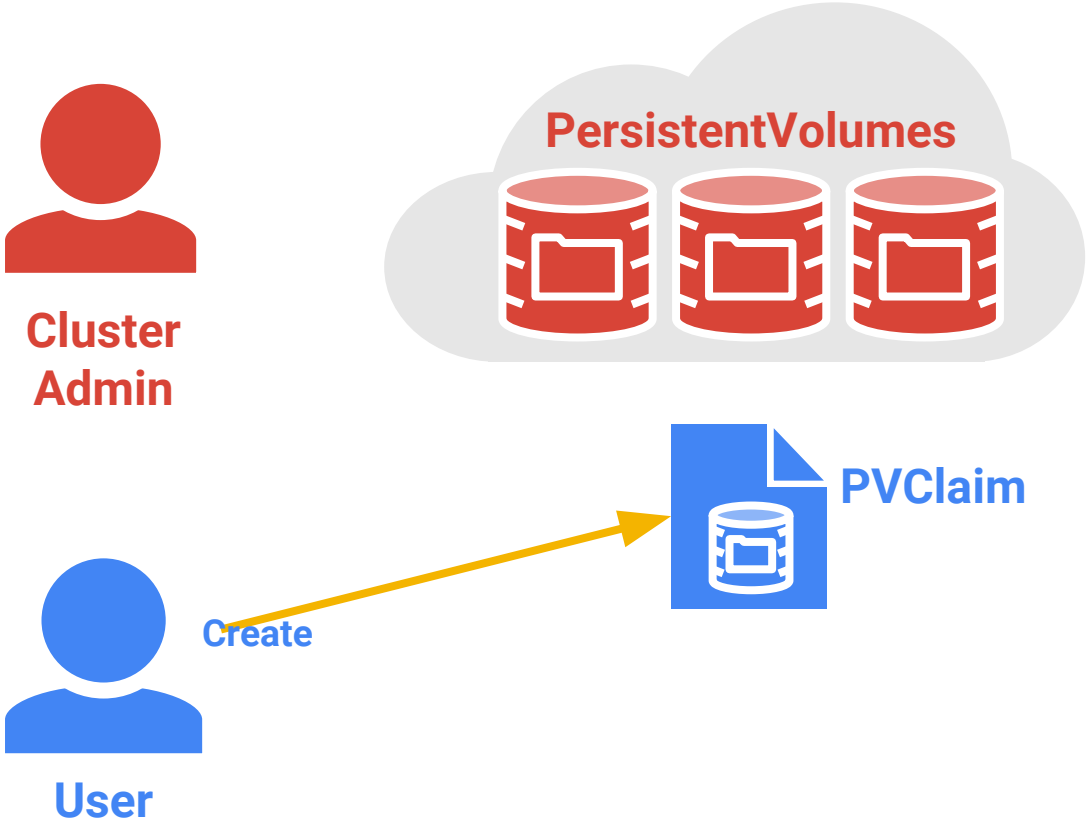
PersistentVolumes

Cluster Admin

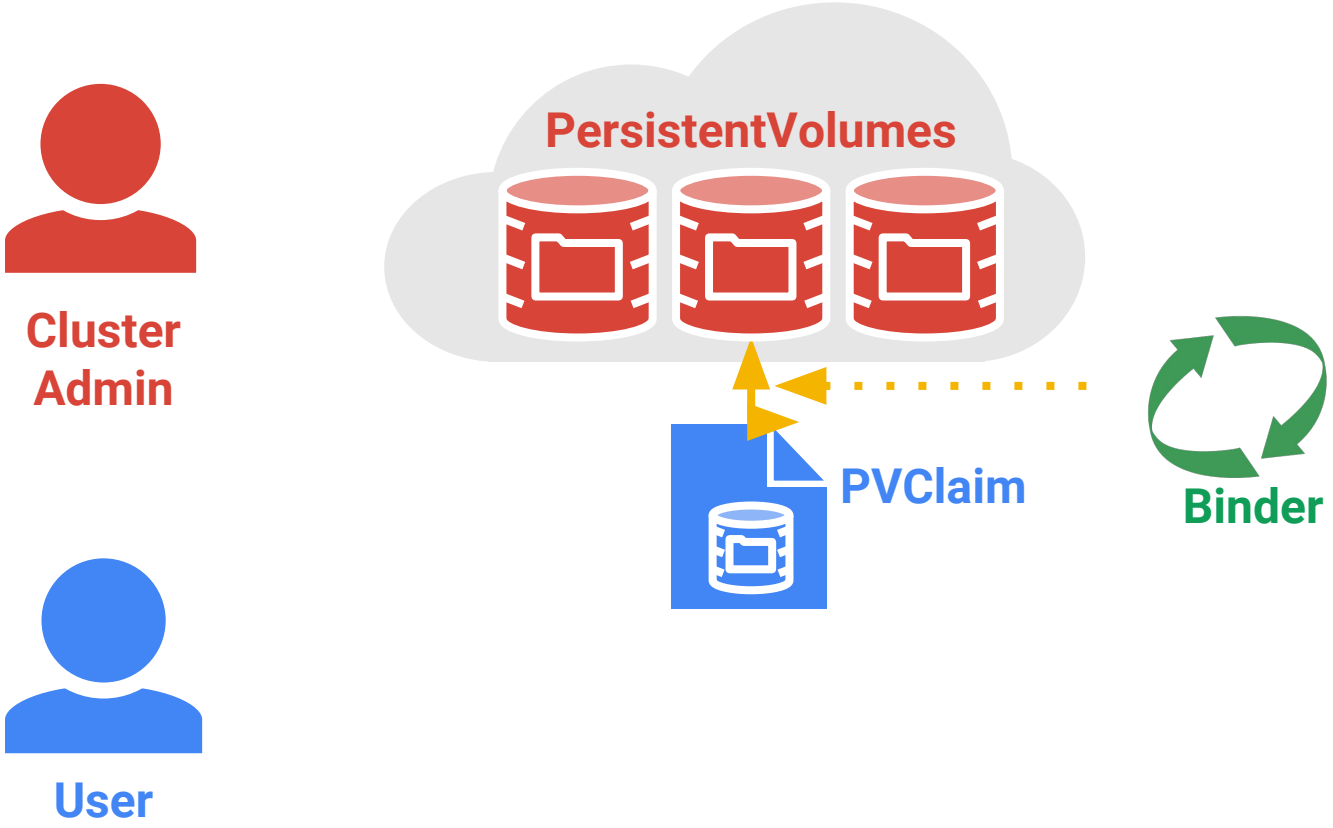


User

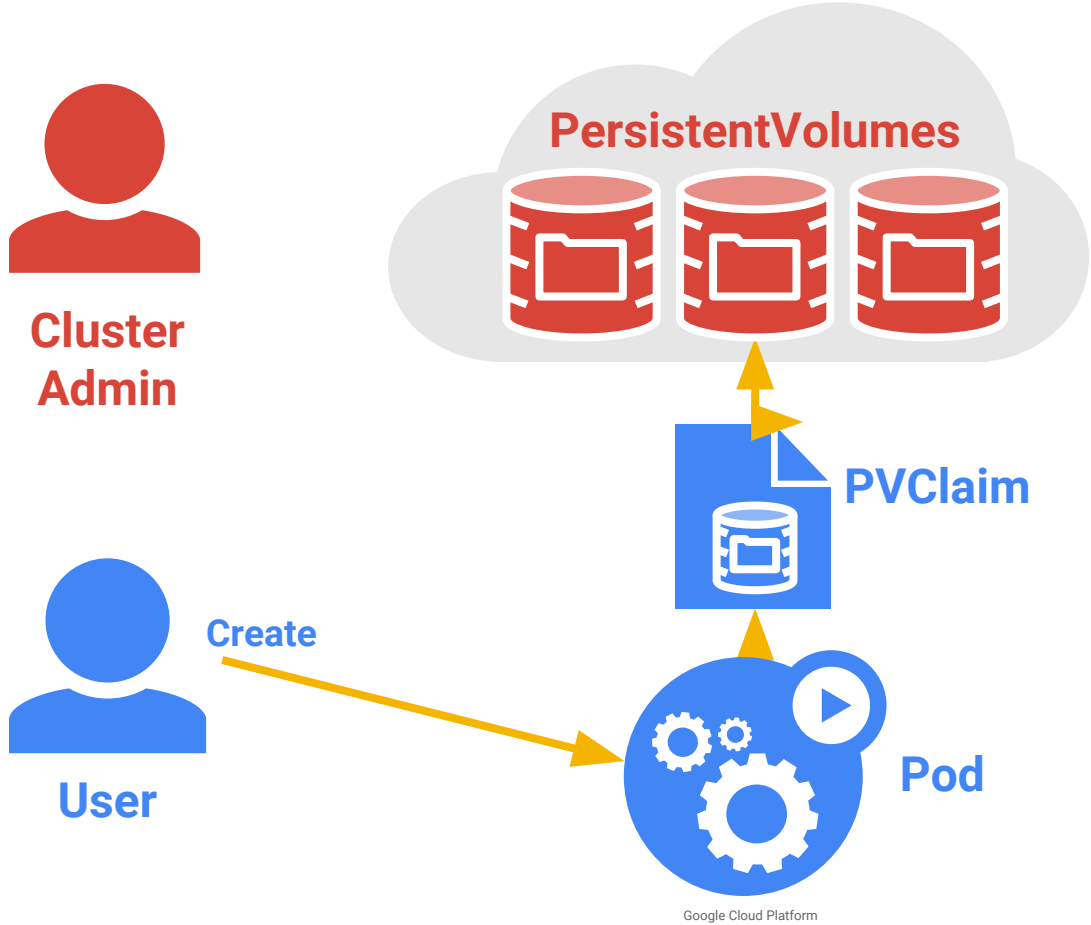
PersistentVolumes



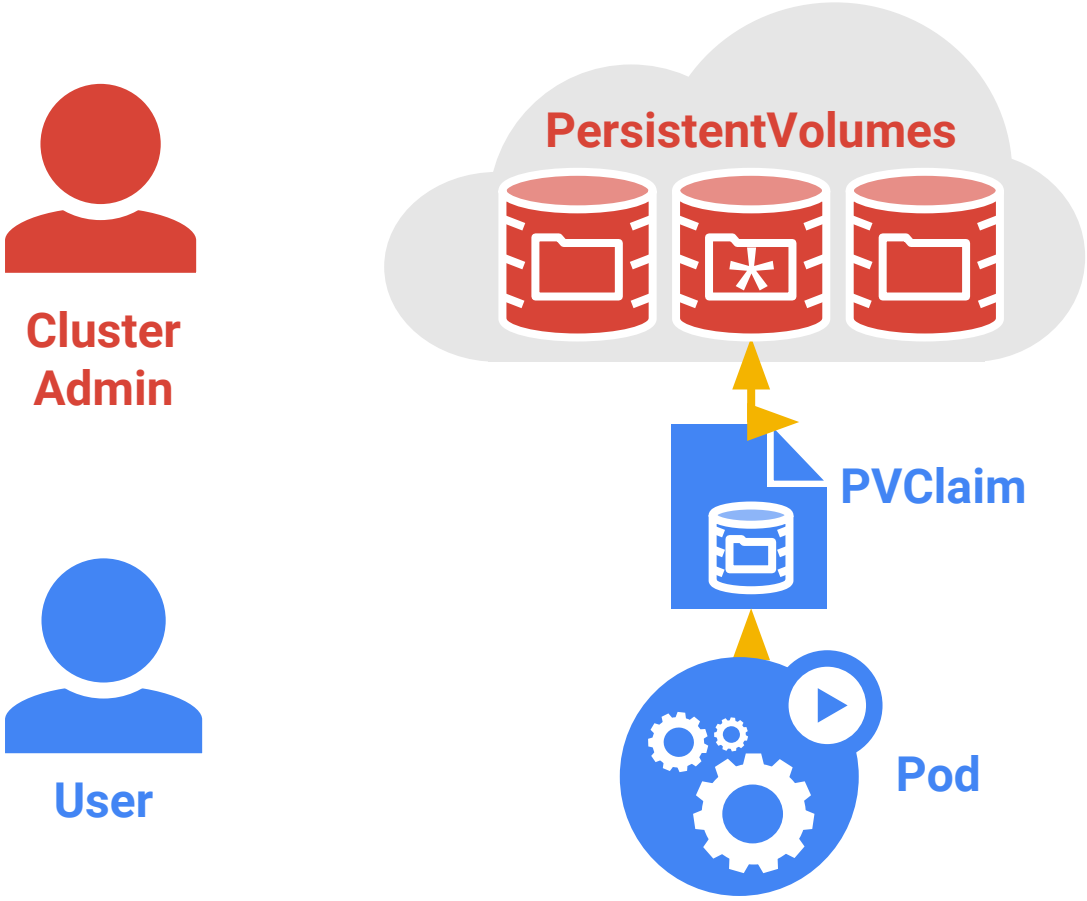
PersistentVolumes



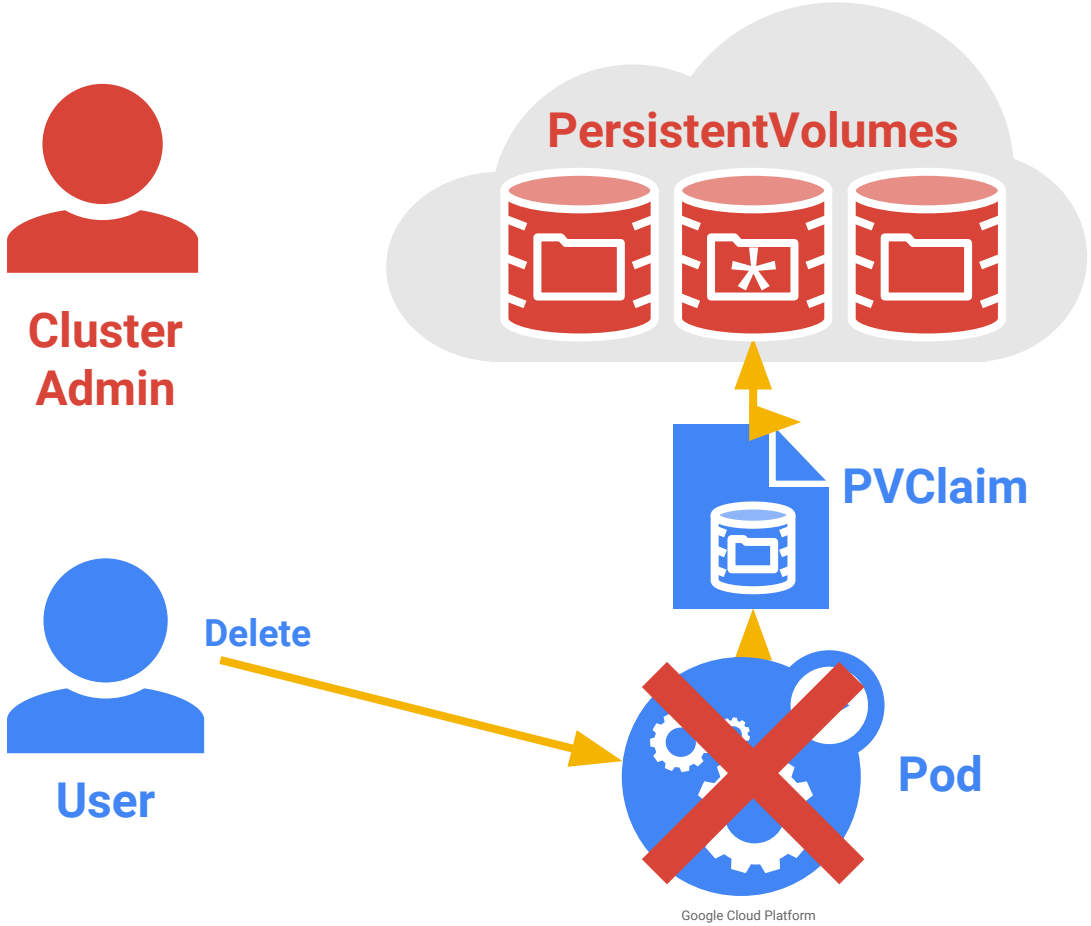
PersistentVolumes



PersistentVolumes



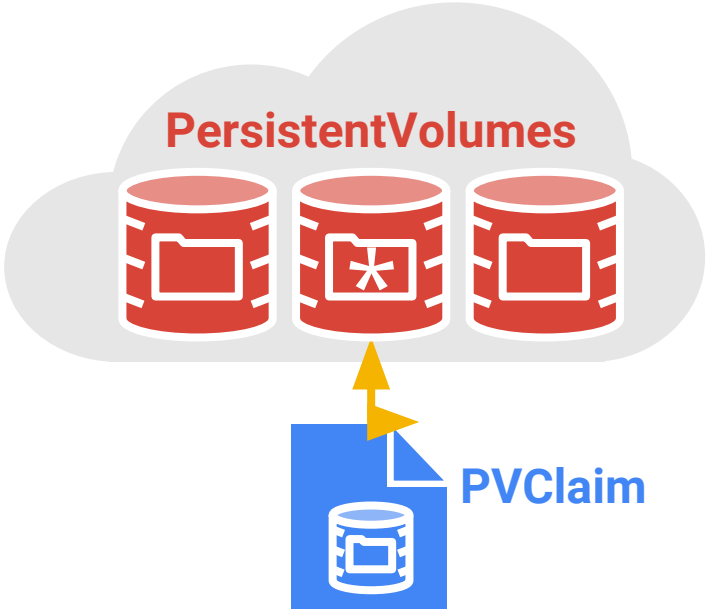
PersistentVolumes



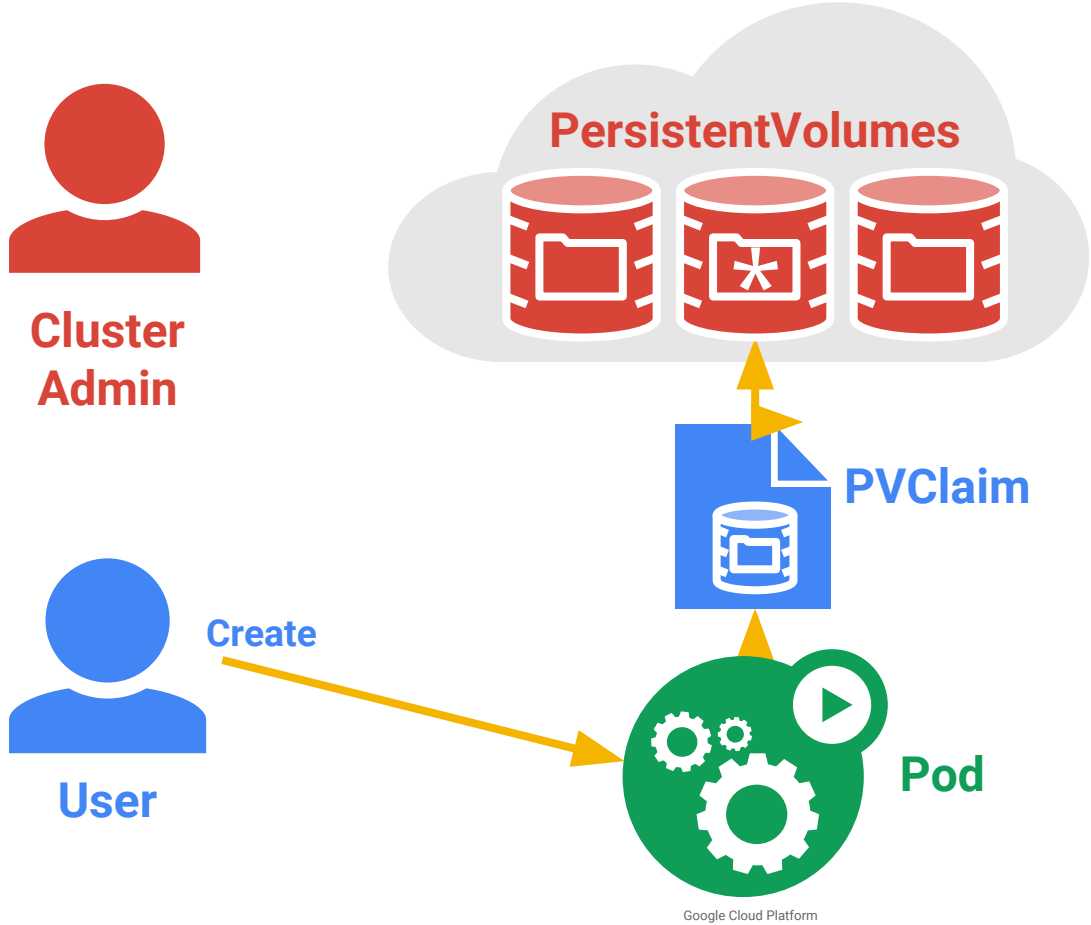
PersistentVolumes

Cluster Admin

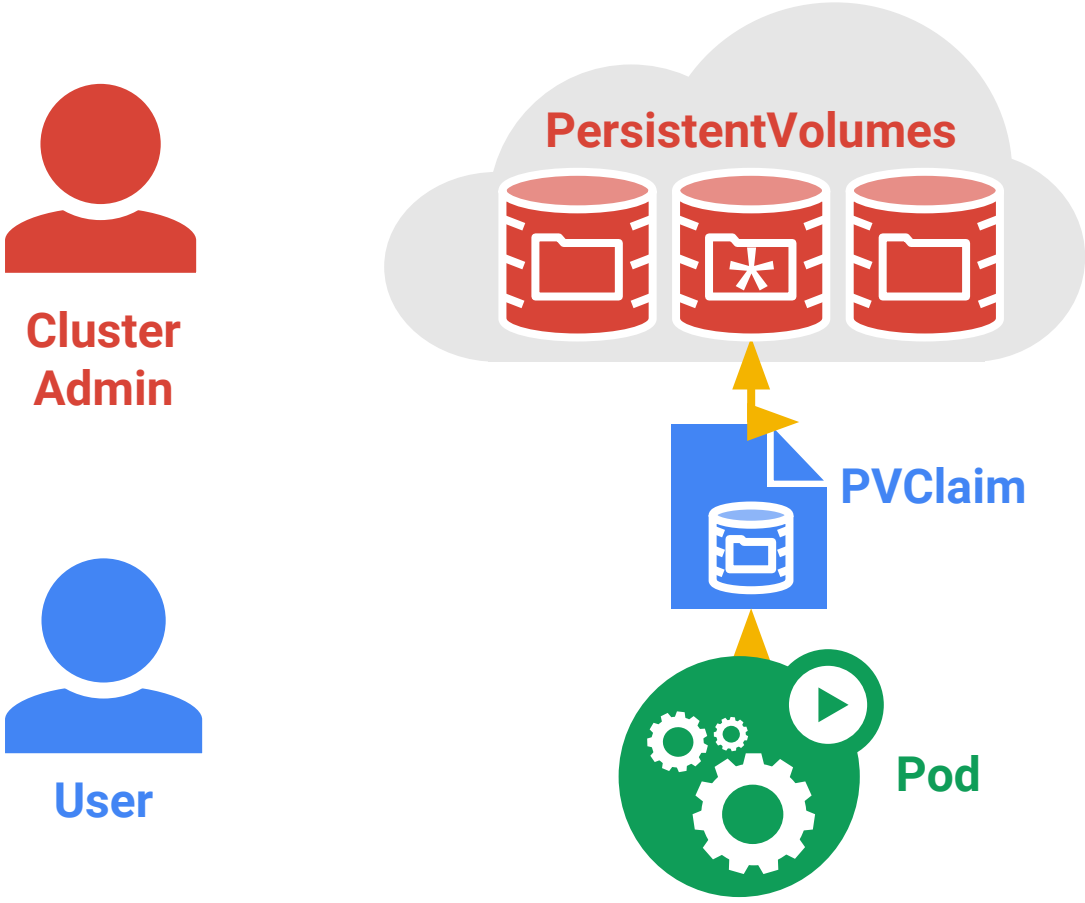
User



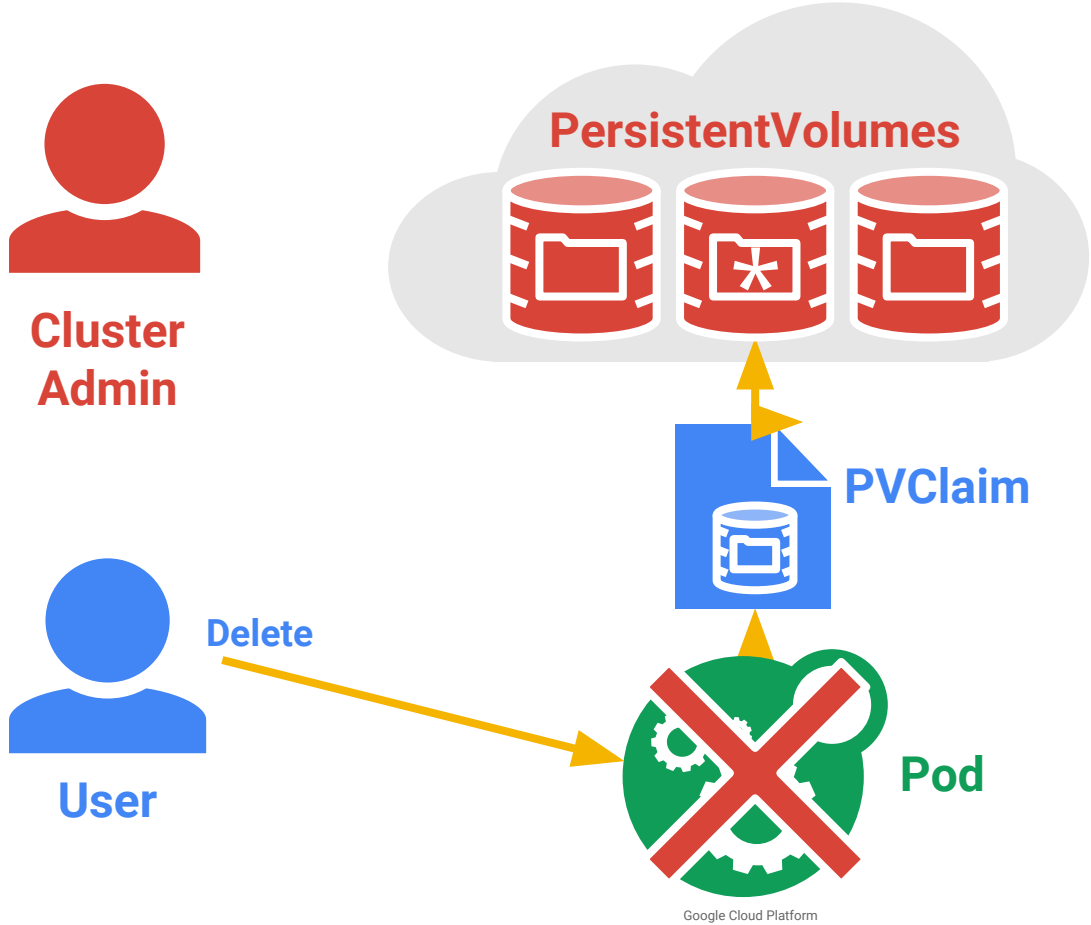
PersistentVolumes



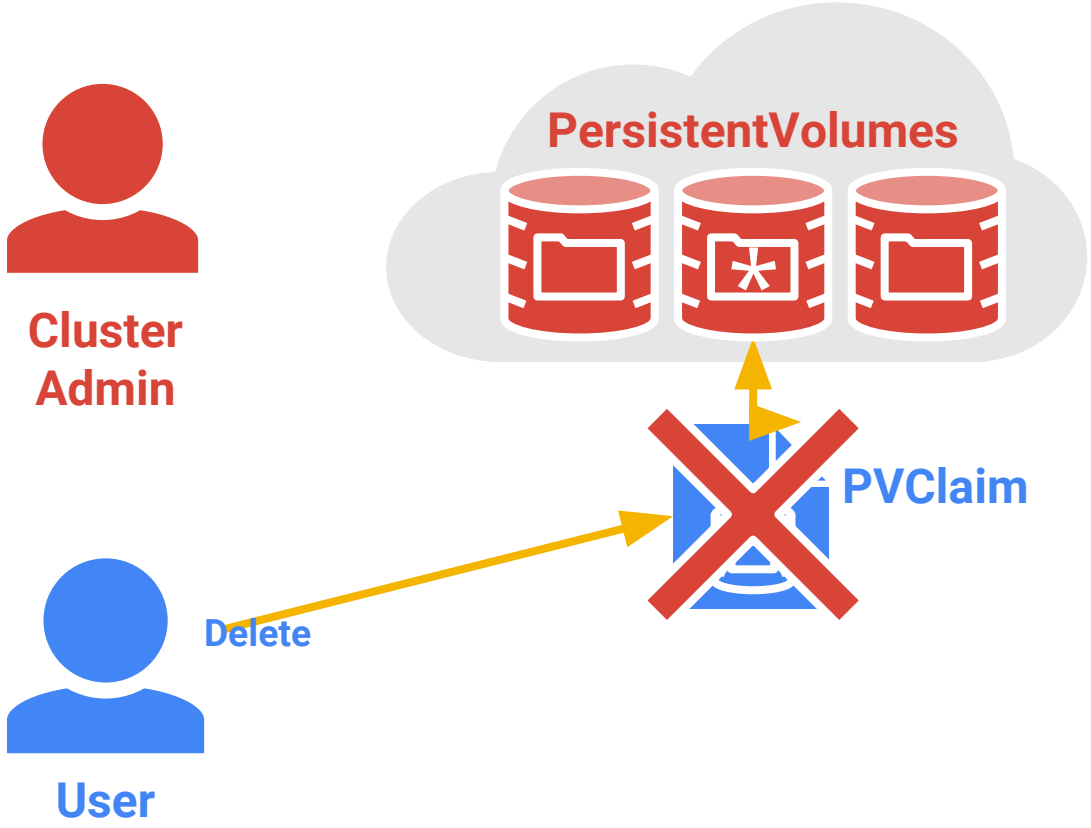
PersistentVolumes



PersistentVolumes

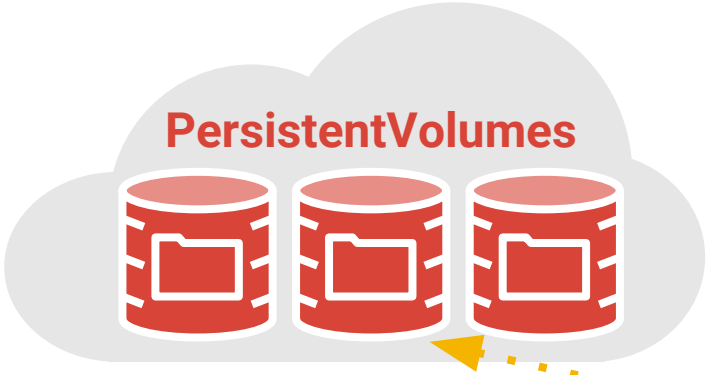


PersistentVolumes



PersistentVolumes

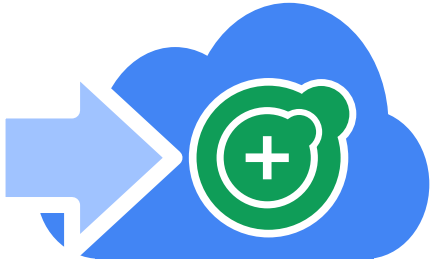
Cluster Admin



Recycler

User

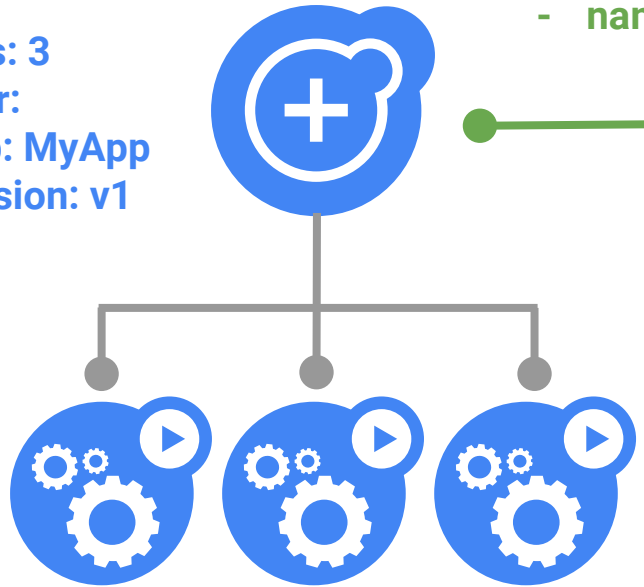
Deployments



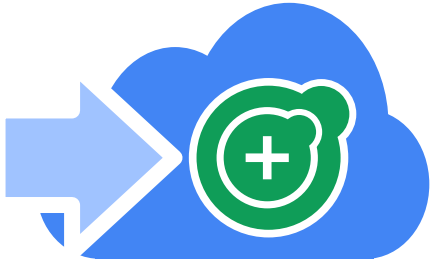
kubectl create ...

Deployment
- name: MyApp

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v1



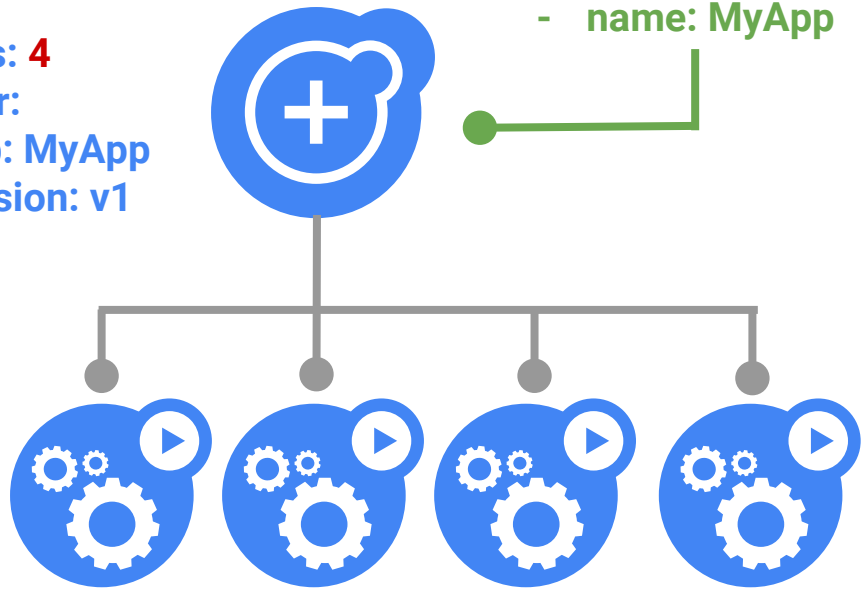
Deployments



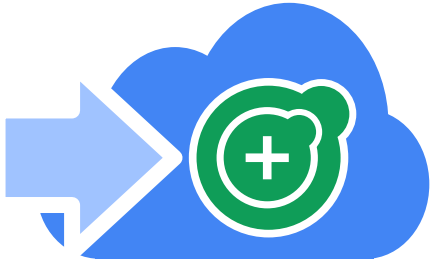
kubectl create ...

Deployment
- name: MyApp

ReplicaSet
- replicas: 4
- selector:
  - app: MyApp
  - version: v1



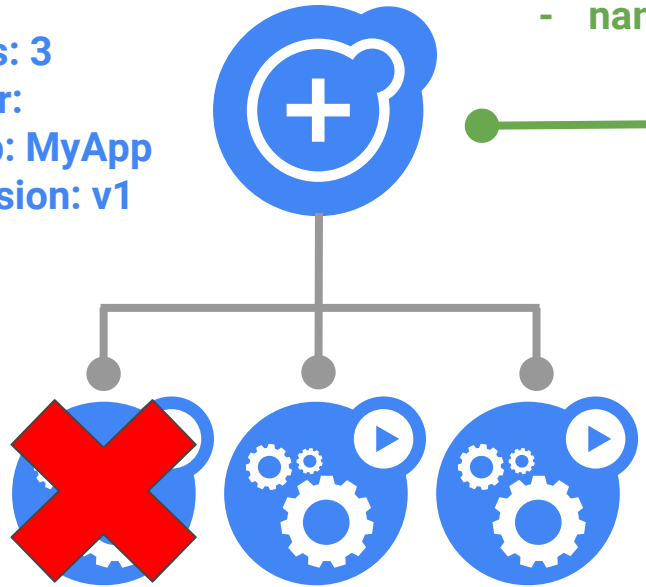
Deployments



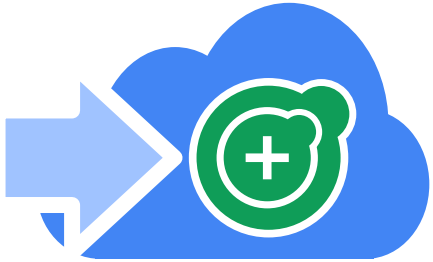
kubectl create ...

Deployment
- name: MyApp

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v1



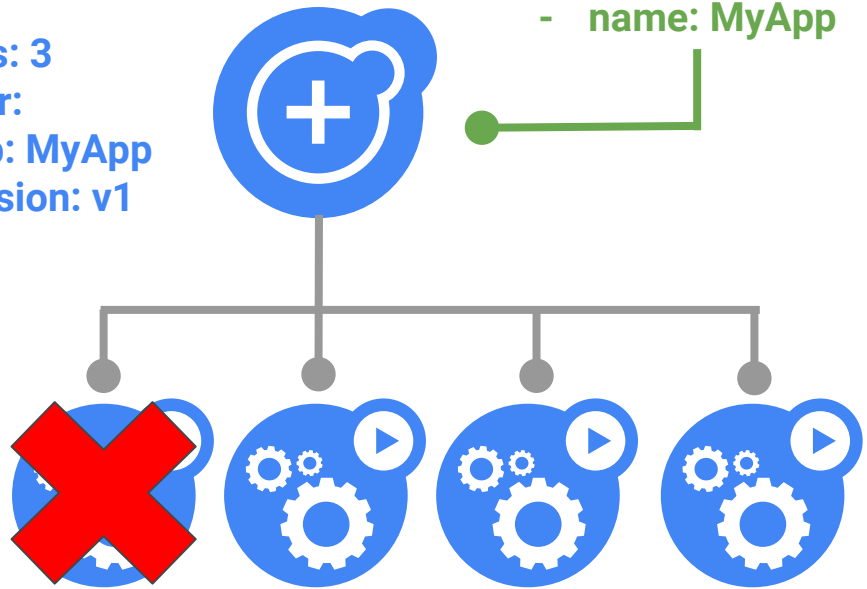
Deployments



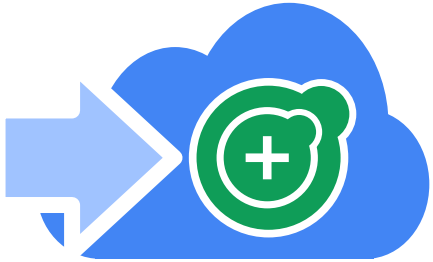
kubectl create ...

Deployment
- name: MyApp

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v1



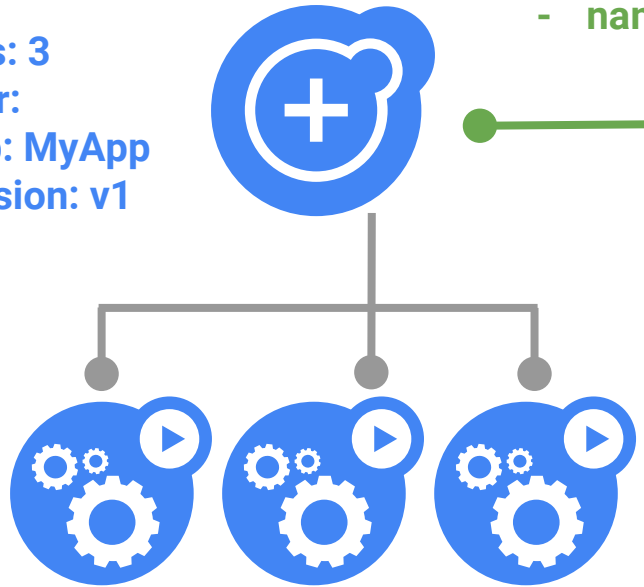
Rolling Updates



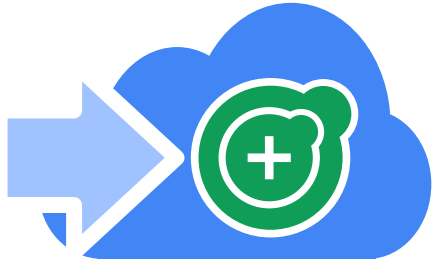
kubectl apply ...

Deployment
- name: MyApp

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v1



Rolling Updates

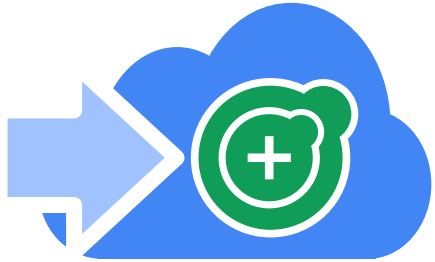


Deployment
- name: MyApp

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- replicas: 0
- selector:
  - app: MyApp
  - version: v2

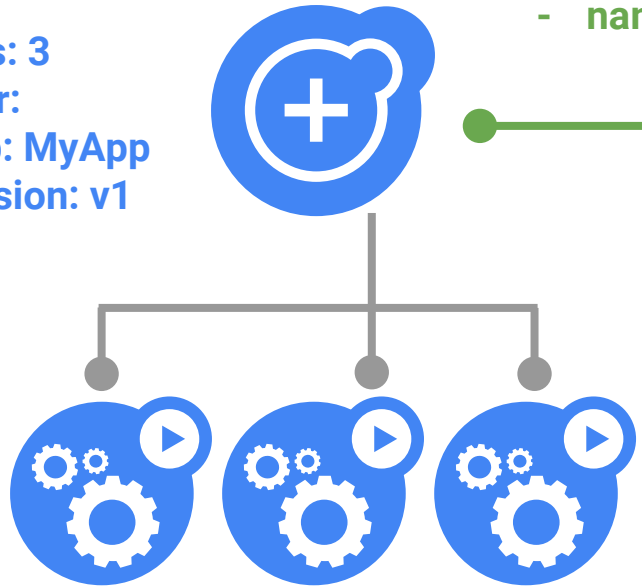
Rolling Updates

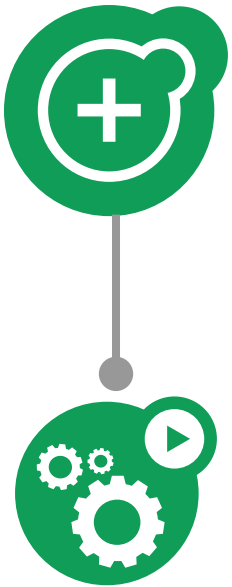


Deployment
- name: MyApp

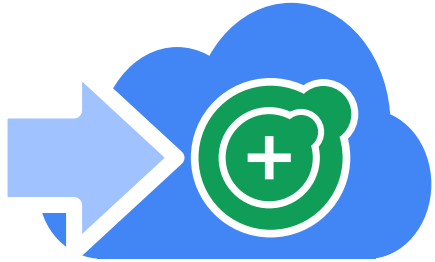
ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- replicas: 1
- selector:
  - app: MyApp
  - version: v2





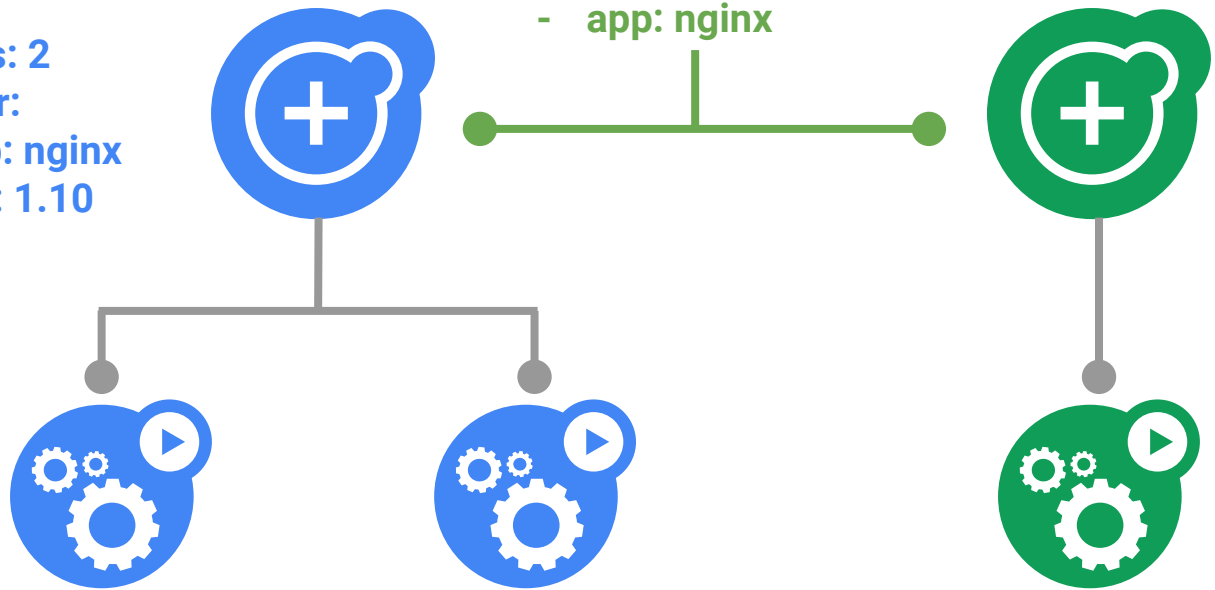
Rolling Updates



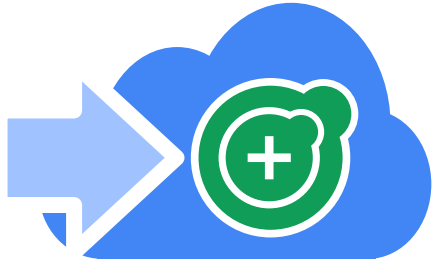
Deployment
- app: nginx

ReplicaSet
- replicas: 2
- selector:
  - app: nginx
  - ver: 1.10

ReplicaSet
- replicas: 1
- selector:
  - app: nginx
  - ver: 1.11



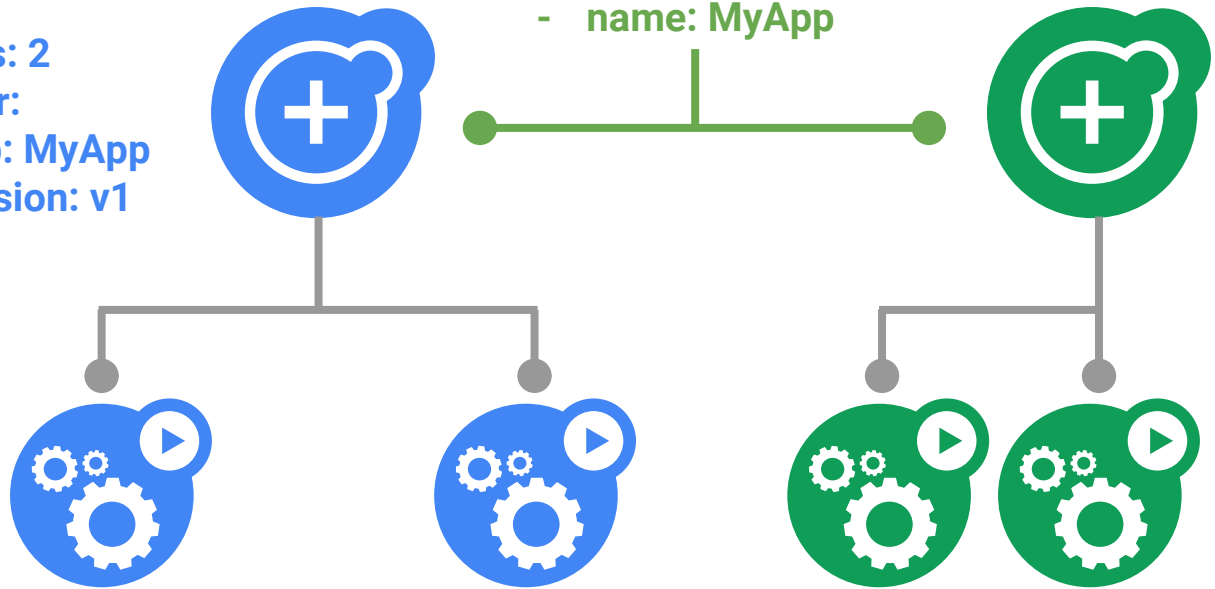
Rolling Updates



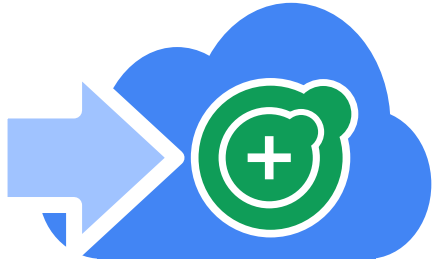
Deployment
- name: MyApp

ReplicaSet
- replicas: 2
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- replicas: 2
- selector:
  - app: MyApp
  - version: v2



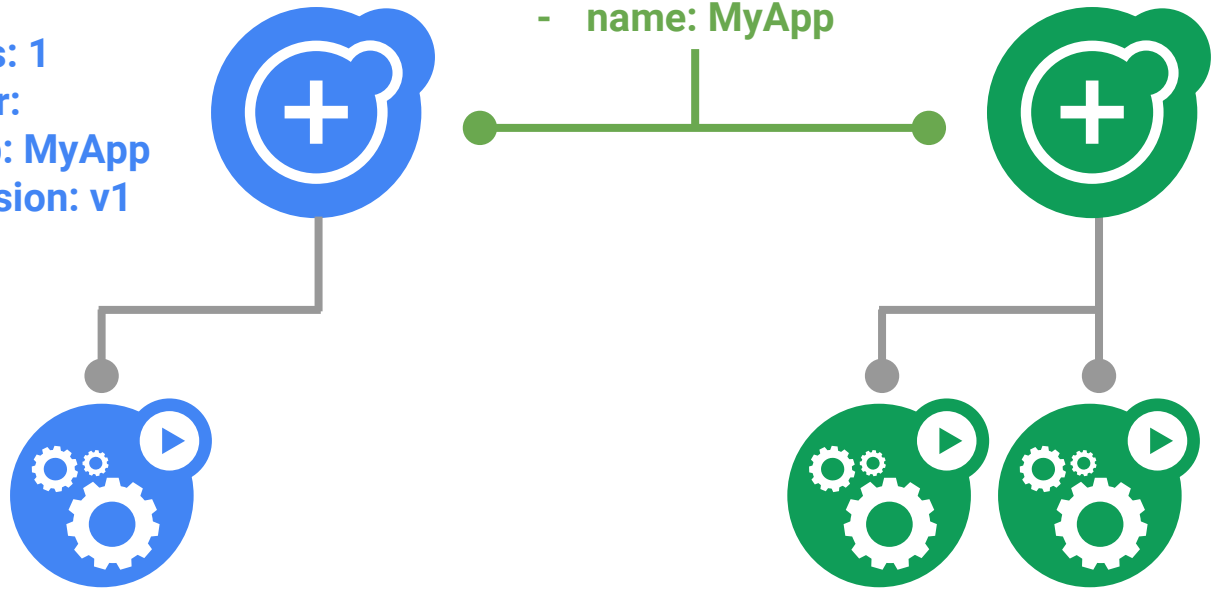
Rolling Updates



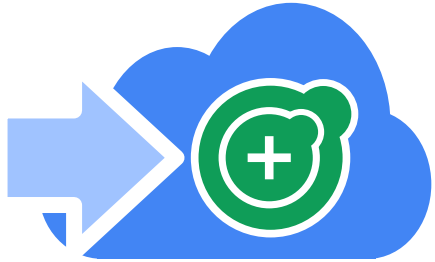
Deployment
- name: MyApp

ReplicaSet
- replicas: 1
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- replicas: 2
- selector:
  - app: MyApp
  - version: v2



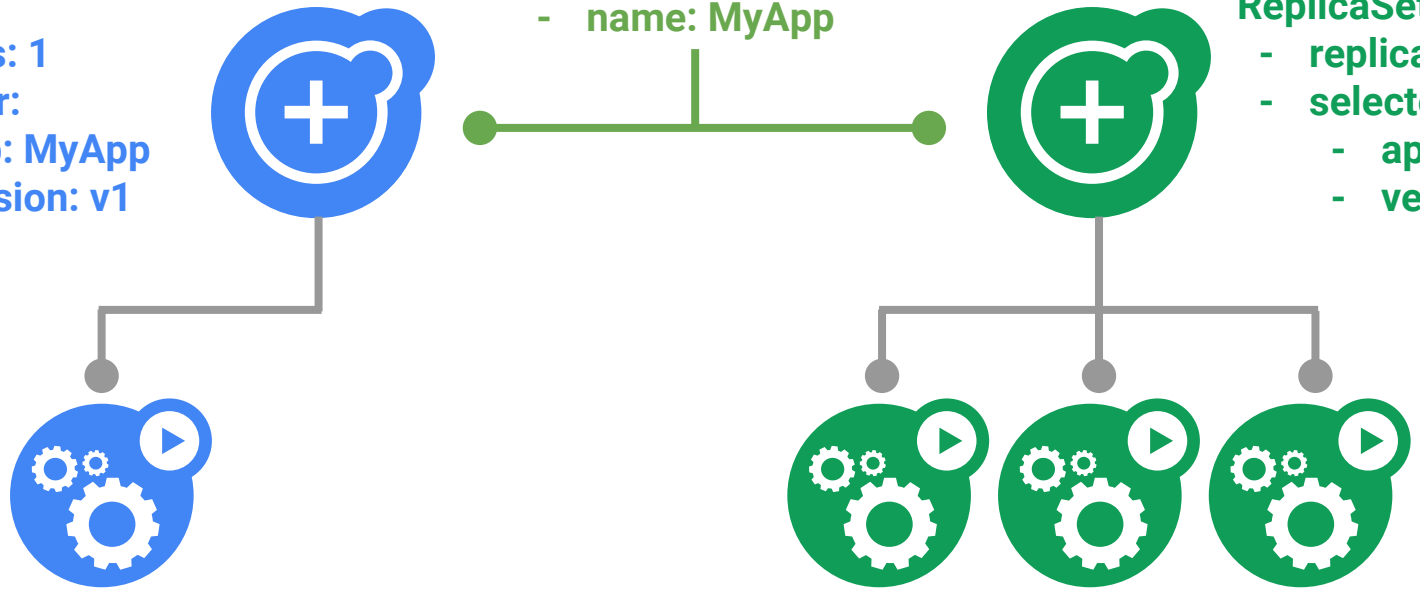
Rolling Updates



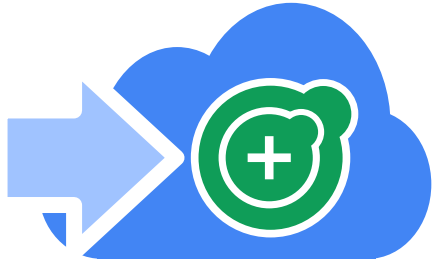
Deployment
- name: MyApp

ReplicaSet
- replicas: 1
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v2



Rolling Updates

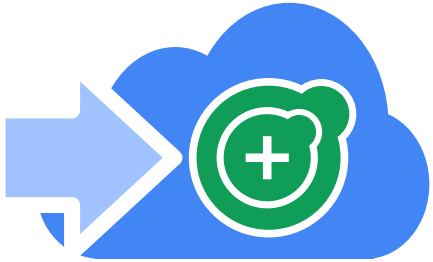


Deployment
- name: MyApp

ReplicaSet
- replicas: 0
- selector:
  - app: MyApp
  - version: v1

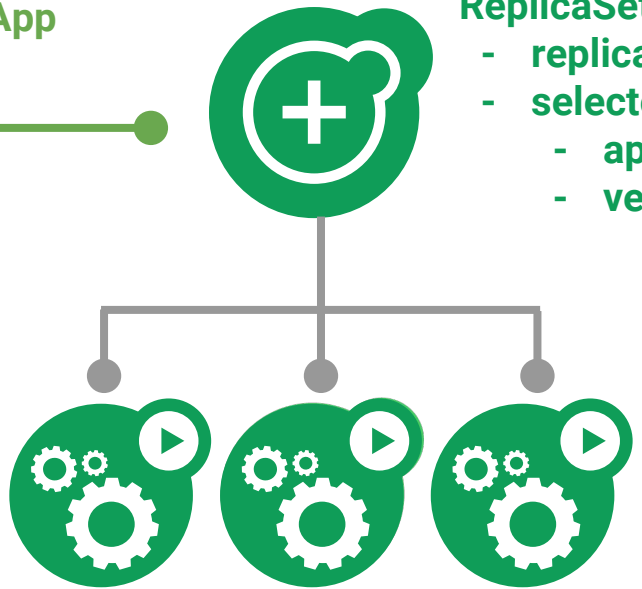
ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v2

Rolling Updates



Deployment
- name: MyApp

ReplicaSet
- replicas: 3
- selector:
  - app: MyApp
  - version: v2



Services

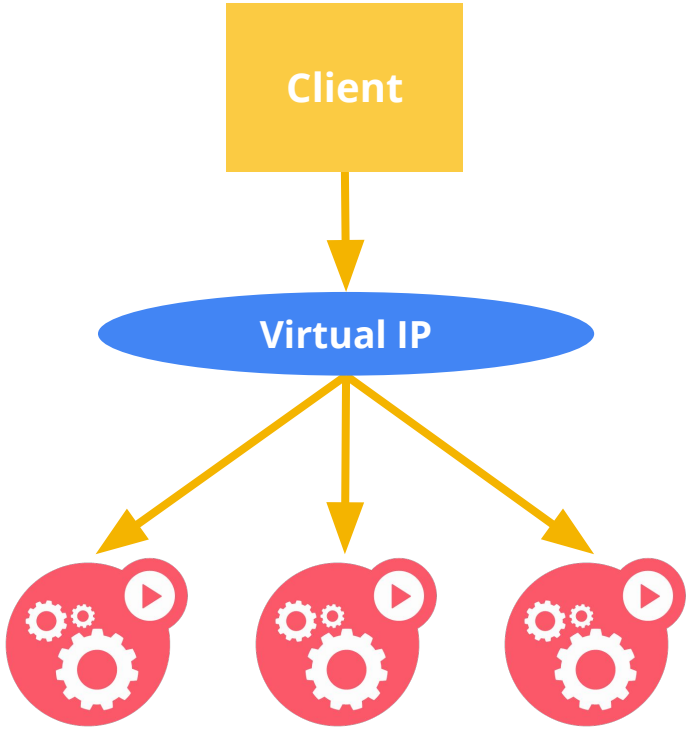
A group of pods that **work together**
- grouped by a selector

Defines access policy
- “load balanced” or “headless”

Gets a stable **virtual IP** and port
- sometimes called the service *portal*
- also a DNS name

VIP is managed by *kube-proxy*
- watches all services
- updates iptables when backends change

Hides complexity - ideal for non-native apps



Labels

Arbitrary metadata

Attached to any API object

Generally represent **identity**

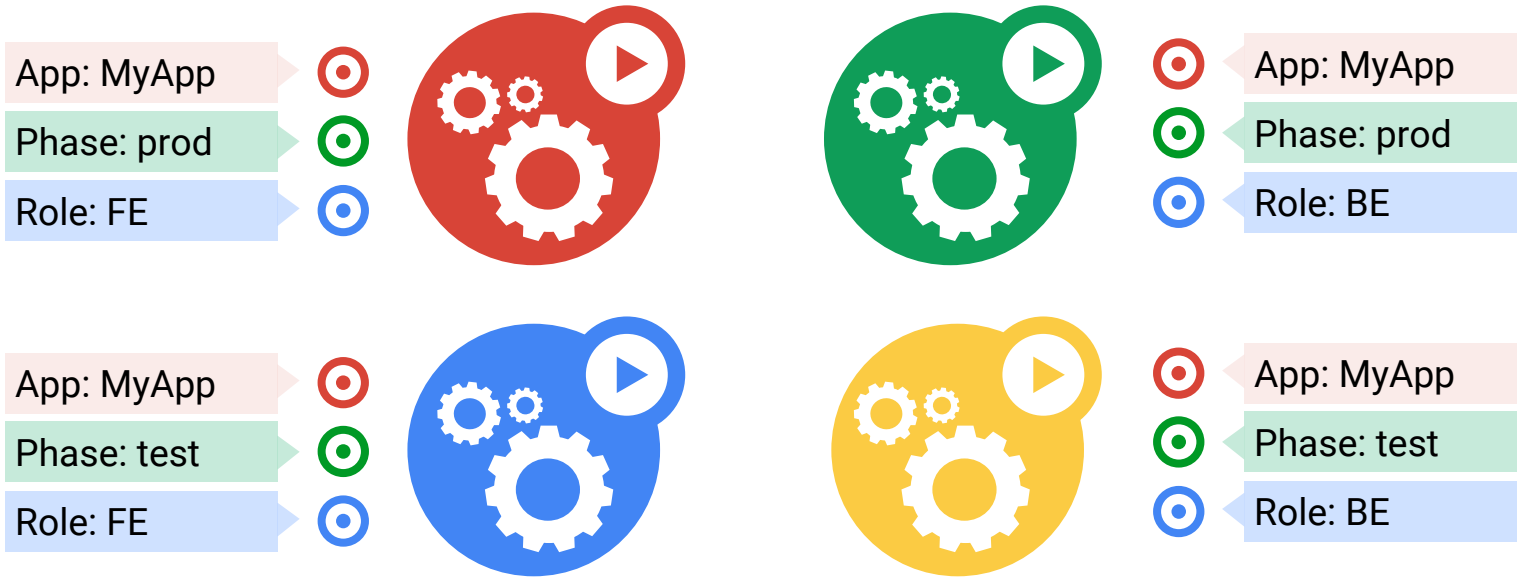
Queryable by **selectors**

- think SQL '*select ... where ...*'

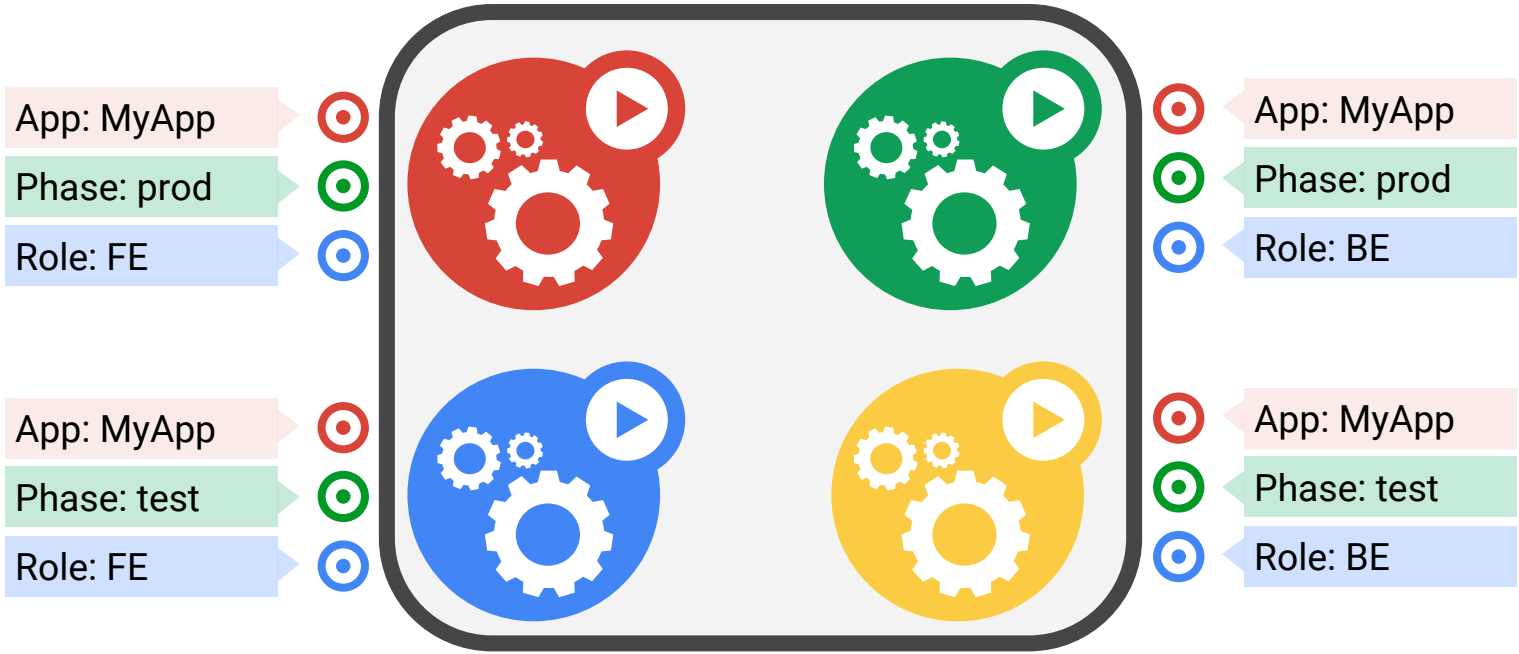
The **only** grouping mechanism

- pods under a ReplicationController
- pods in a Service
- capabilities of a node (constraints)

Selectors

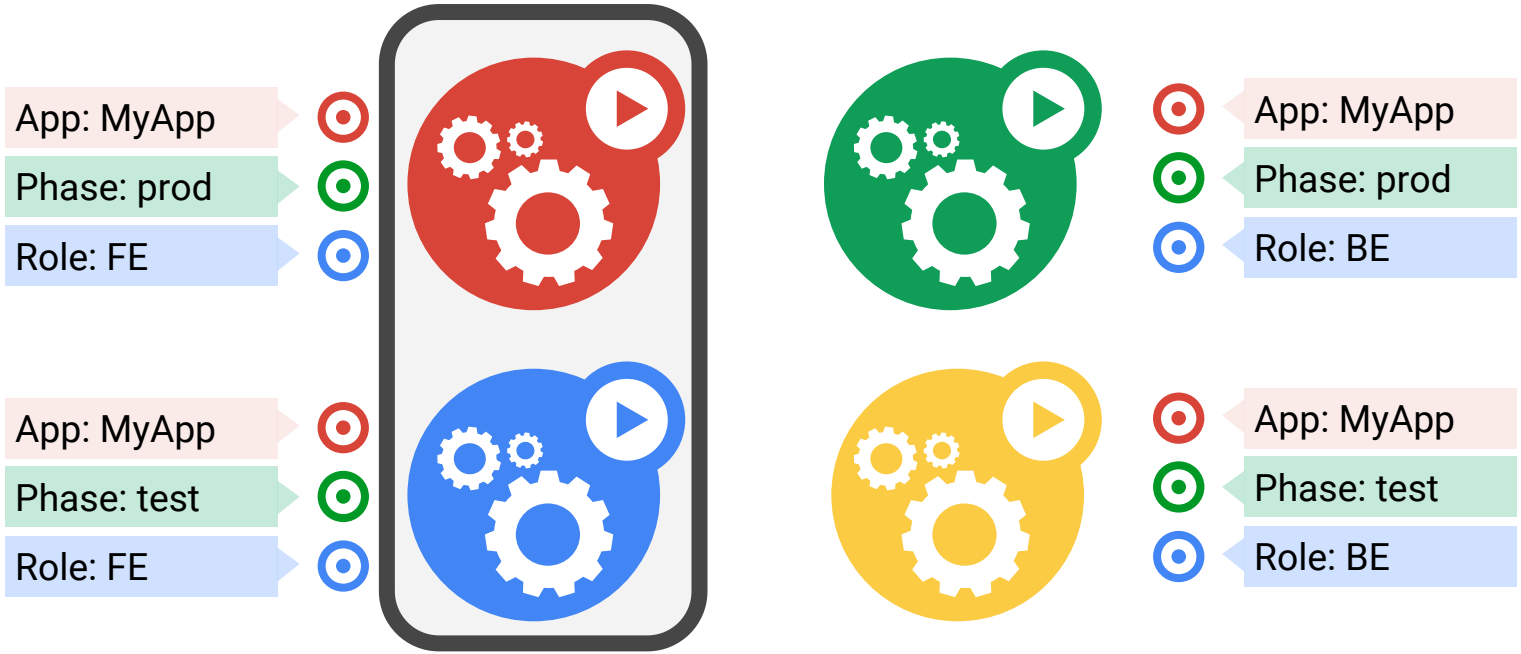

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE

Selectors



App = MyApp

Selectors

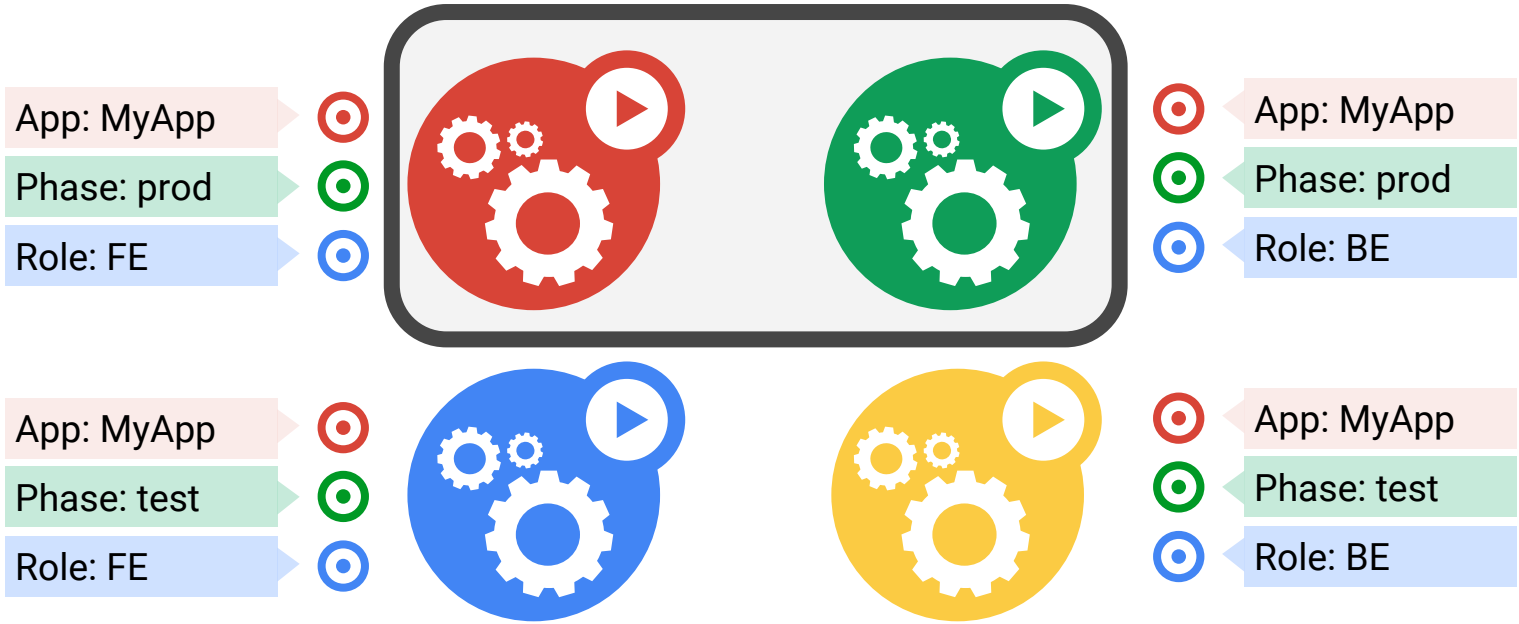


App = MyApp, Role = FE

Selectors

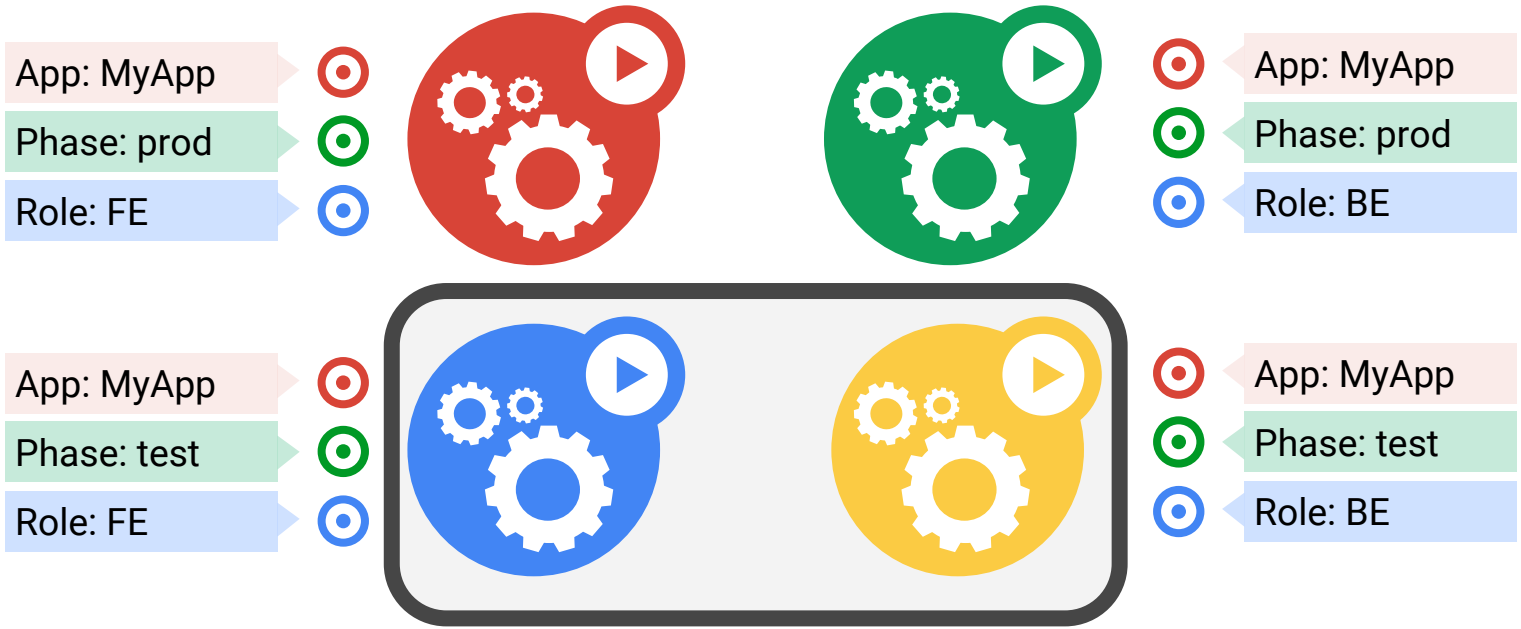
App = MyApp, Role = BE

Selectors



App = MyApp, Phase = prod

Selectors


App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE
App = MyApp, Phase = test

Jobs

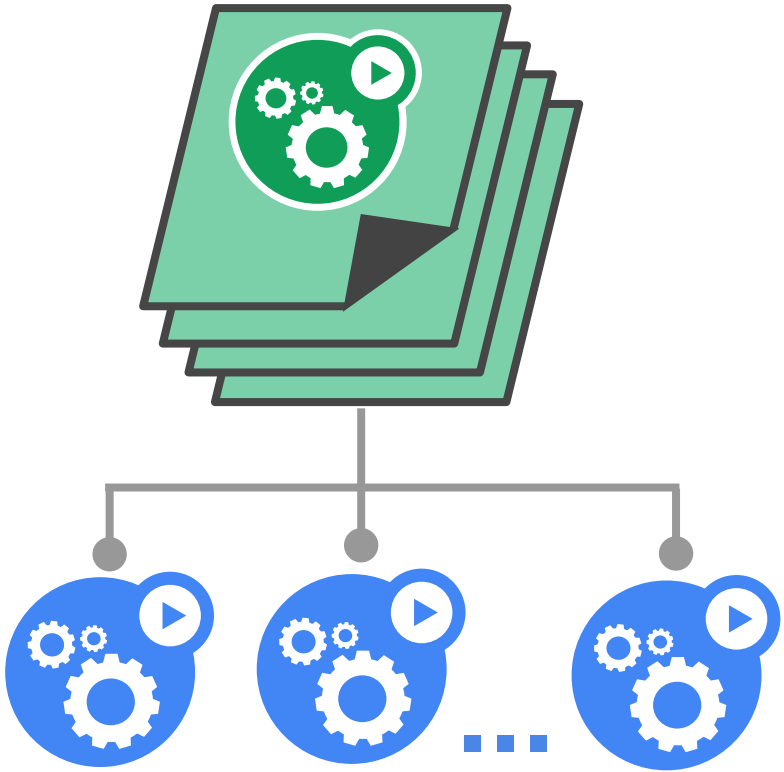
Run-to-completion, as opposed to run-forever

- Express parallelism vs. required completions
- Workflow: restart on failure
- Build/test: don't restart on failure

Aggregates success/failure counts

Built for batch and big-data work

Status: GA in Kubernetes v1.2



Namespaces

Problem: I have too much stuff!

- name collisions in the API
- poor isolation between users
- don't want to expose things like Secrets

Solution: Slice up the cluster

- create new Namespaces as needed
  - per-user, per-app, per-department, etc.
- part of the API - NOT private machines
- most API objects are namespaced
  - part of the REST URL path
- Namespaces are just another API object
- One-step cleanup - delete the Namespace
- Obvious hook for policy enforcement (e.g. quota)

slack.kubernetes.io

Thank You