CoreOS and the Container Revolution

The Journey to GIFEE

Jonathan Boulle

CoreOS: a short history

Way back in 2013...

- Status quo: set up a server and never touch it
- Internet full of servers running years-old software with dozens of vulnerabilities


ars TECHNICA
BIZ & IT
TECH
SCIENCE
POLICY
CARS
GAMING & CULTURE
FORUMS
RISK ASSESSMENT —
Ancient Linux servers: The blighted slum houses of the Internet [Updated]
Mass compromise infects old Web servers running Linux versions from 2007.

Enter CoreOS

San Francisco, California

CoreOS Mission:

Secure the Internet

Secure the internet?

- Vulnerabilities will always happen
- Only way to respond is to make software updates as ***automatic*** and ***seamless*** as possible

Secure the internet?

The browsers showed us the way!

Auto-updating browsers fixed security

We got HTML5 at the same time

Secure the internet?

- Start at the bottom of the stack: the OS
- **Container Linux** (formerly CoreOS Linux)

First problem...

- Updates are supposed to be automatic and seamless
- If the OS is always updating, what about applications running on it?

Patches to the OS and kernel are hard

APPLICATION

Dependency breakage

Uptime risk

SECURITY

Retest after updates

No automation

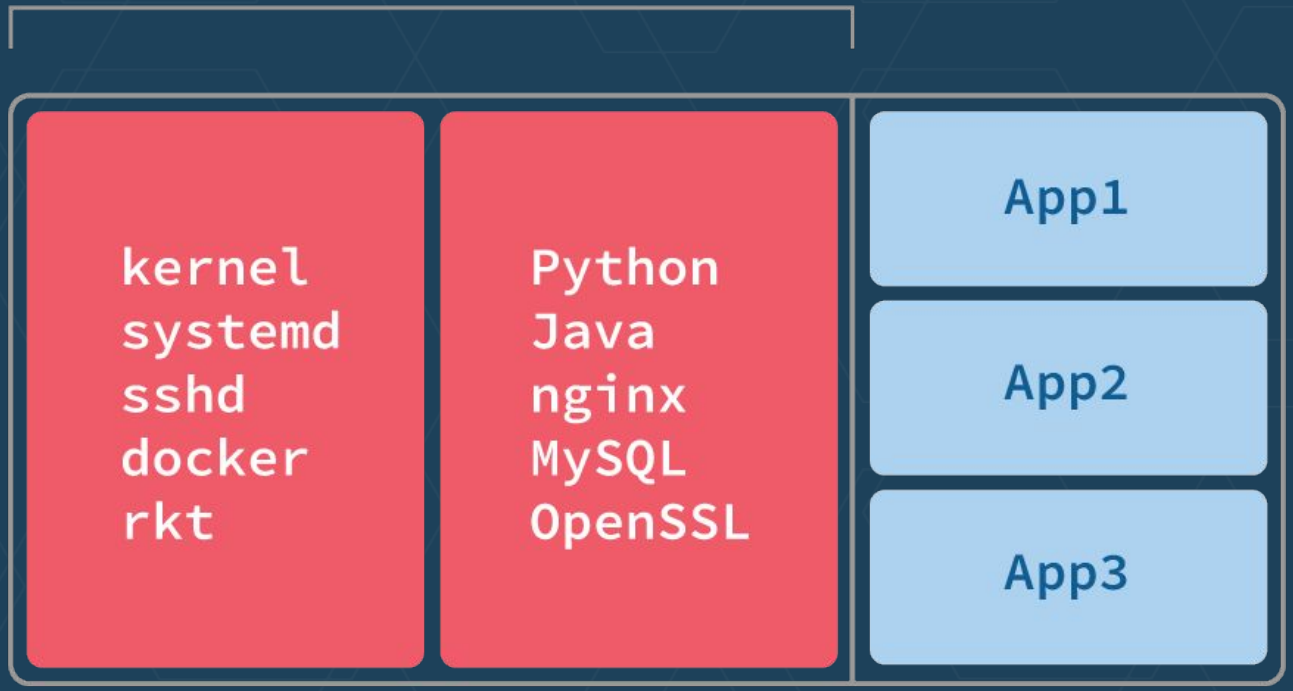
Automatic OS patches

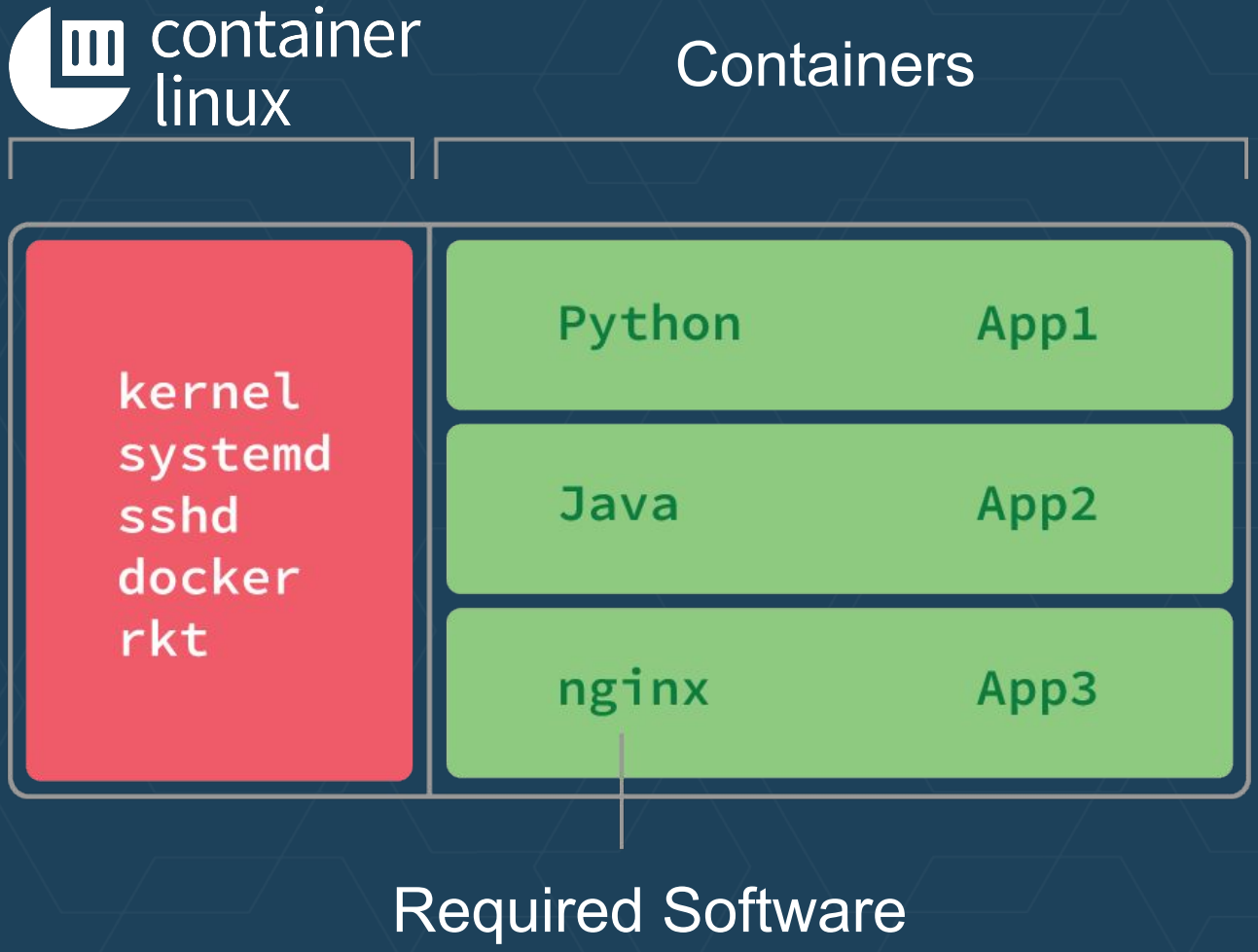
Classic use case for *containers* and *orchestration*

- *containers* decouple the application and OS update lifecycles (update at different cadences)
- *orchestration* decouples application and OS uptime (services can remain unaffected during OS downtime)

Abstract away app from the OS

Traditional Linux




container linux
Containers
kernel
systemd
sshd
docker
rkt
Python
App1
Java
App2
nginx
App3
Required Software

CoreOS

Base software managed by CoreOS

kernel systemd OpenSSH

Protect the app from the OS

- Change the OS libraries without affecting any applications

Protect apps from each other

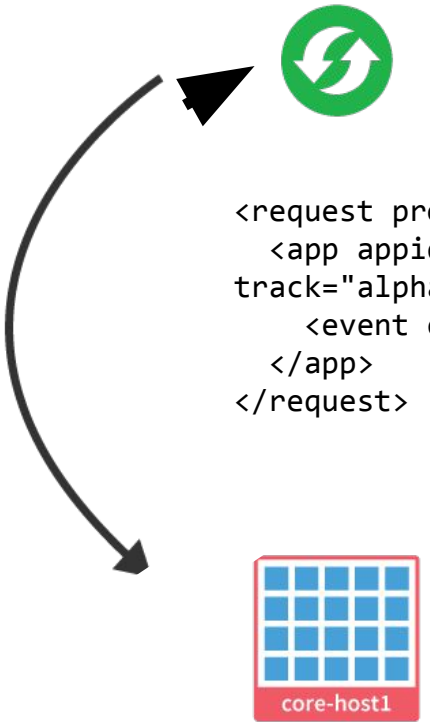
- Mixed versions of dependencies
  eg. python 3.4 & python 2.7
- Isolated network namespace
- Isolated file system namespace

How do Container Linux updates work?

- Omaha protocol (check-in/retrieval)
  - Simple XML-over-HTTP protocol developed by Google to facilitate polling and pulling updates from a server

Omaha protocol

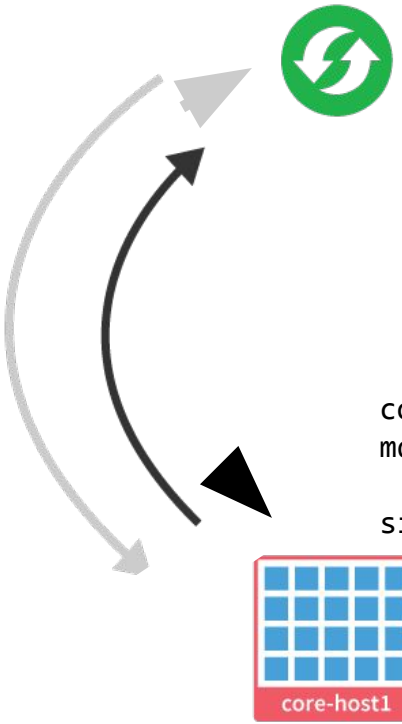
Omaha protocol



Client sends application id and current version to the update server

```
<request protocol="3.0" version="CoreOSUpdateEngine-0.1.0.0">
  <app appid="{e96281a6-d1af-4bde-9a0a-97b76e56dc57}" version="410.0.0"
track="alpha" from_track="alpha">
    <event eventtype="3"></event>
  </app>
</request>
```

Omaha protocol

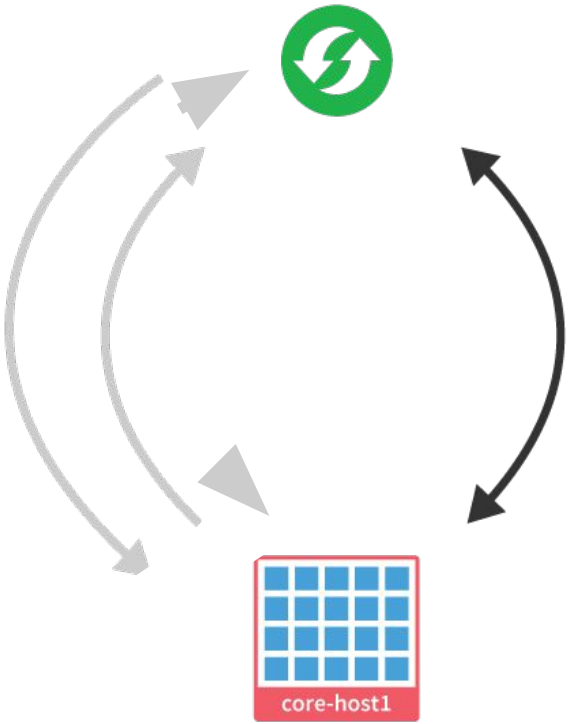


Client sends application id and current version to the update server

Update server responds with the URL of an update to be applied

```
  <url
codebase="https://commondatastorage.googleapis.com/update-storage.core-os.net/a
md64-usr/452.0.0/"></url>
  <package hash="D0lBAMD1Fwv8YqQuDYEAjXw6YZY=" name="update.gz"
size="103401155" required="false">
```

Omaha protocol

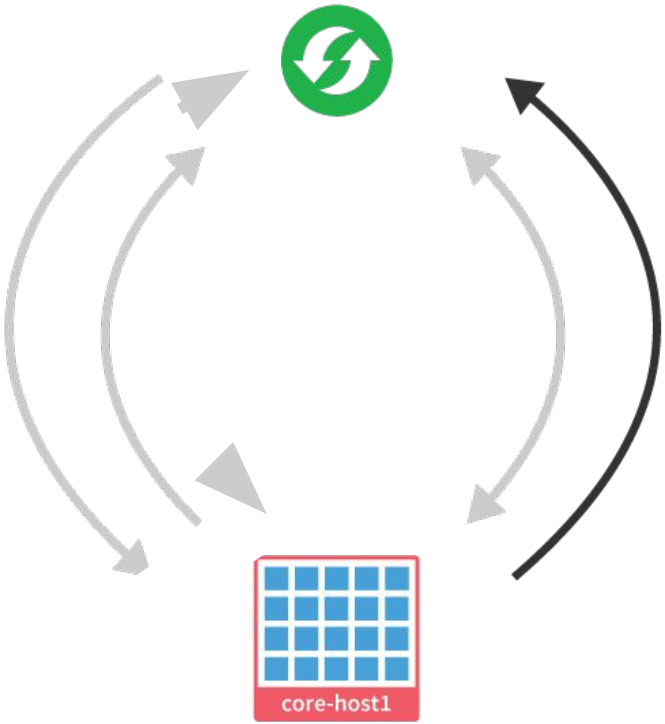


Client sends application id and current version to the update server

Update server responds with the URL of an update to be applied

Client downloads data, verifies hash & cryptographic signature, and applies the update

Omaha protocol



Client sends application id and current version to the update server

Update server responds with the URL of an update to be applied

Client downloads data, verifies hash & cryptographic signature, and applies the update

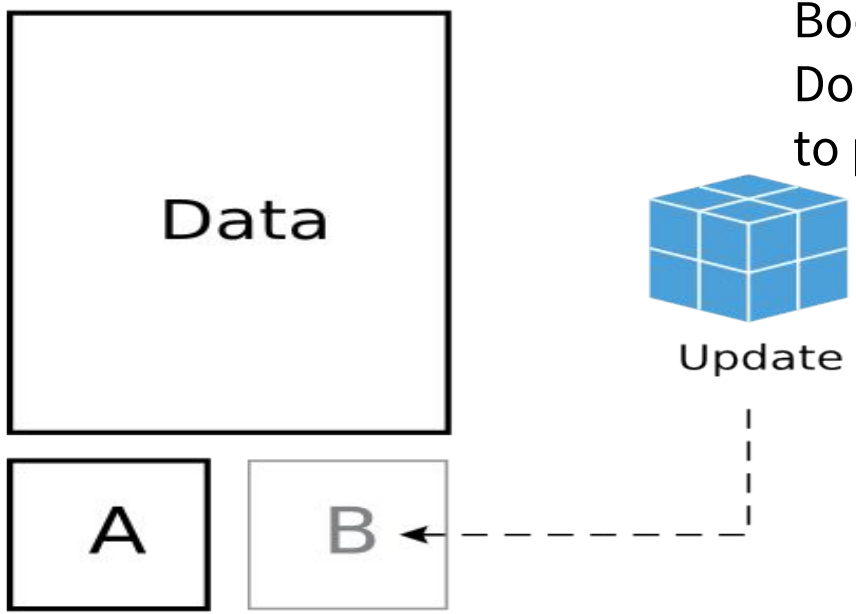
Updater exits with response code then reports the update to the update server

CoreOS

How do Container Linux updates work?

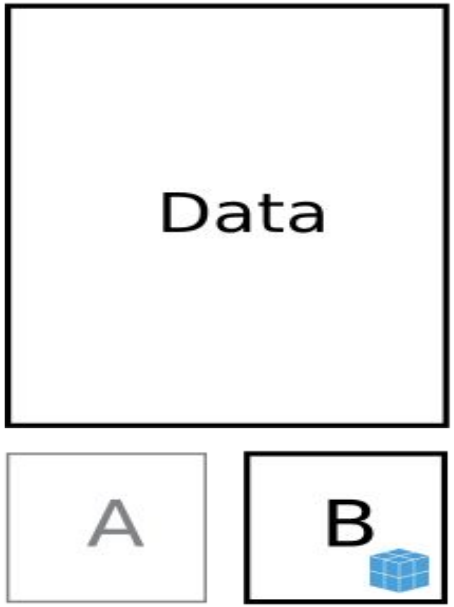
- Omaha protocol (check-in/retrieval)
  - Simple XML-over-HTTP protocol developed by Google to facilitate polling and pulling updates from a server
- Active/passive boot partitions
  - Atomically install update, reboot to test success

Active/passive root partitions



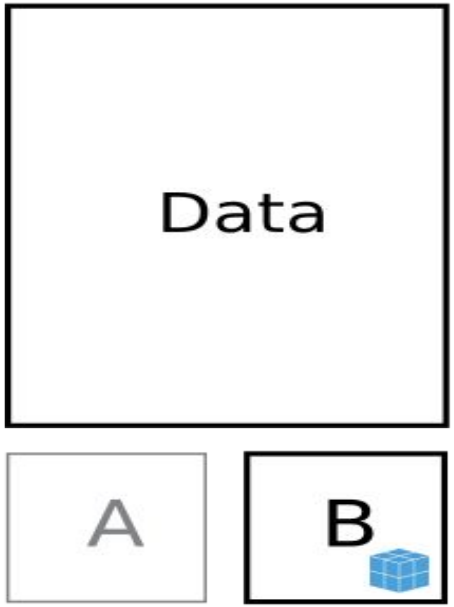
Booted off partition A.
Download update and commit to partition B. Change GPT.

Active/passive root partitions



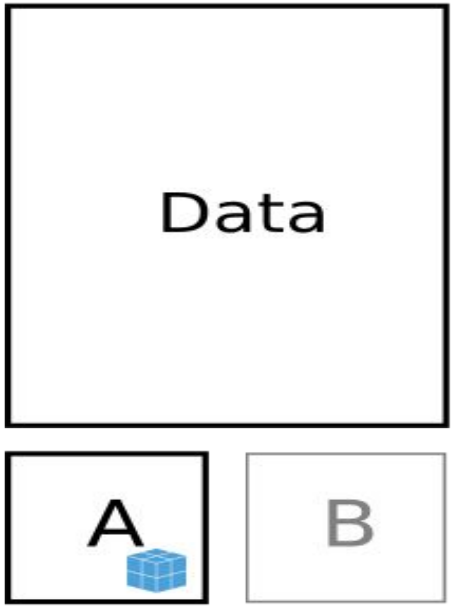
Reboot (into Partition B).
If tests succeed, continue normal operation and mark success in GPT.

Active/passive root partitions



But what if partition B fails update tests...

Active/passive root partitions

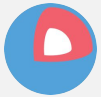


Change GPT to point to previous partition, reboot. Try update again later.

Active/passive root partitions

- core-01 ~ # cgpt show /dev/sda3
  start size contents
  264192 2097152 Label: "USR-A"
  Type: Alias for coreos-rootfs
  UUID: 7130C94A-213A-4E5A-8E26-6CCE9662
  Attr: priority=1 tries=0 successful=1

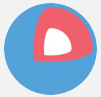
- core-01 ~ # cgpt show /dev/sda4
  start size contents
  2492416 2097152 Label: "USR-B"
  Type: Alias for coreos-rootfs
  UUID: E03DD35C-7C2D-4A47-B3FE-27F15780A
  Attr: priority=2 tries=1 successful=0



Active/passive root partitions

- core-01 ~ # cgpt show /dev/sda3
  start size contents
  264192 2097152 Label: "USR-A"
  Type: Alias for coreos-rootfs
  UUID: 7130C94A-213A-4E5A-8E26-6CCE9662
  Attr: **priority=1 tries=0 successful=1**

- core-01 ~ # cgpt show /dev/sda4
  start size contents
  2492416 2097152 Label: "USR-B"
  Type: Alias for coreos-rootfs
  UUID: E03DD35C-7C2D-4A47-B3FE-27F15780A
  Attr: **priority=2 tries=1 successful=0**



Patches to the OS and kernel are hard

APPLICATION

Dependency breakage

Uptime risk

SECURITY

Retest after updates

No automation

Patches to the OS and kernel are hard

APPLICATION

~~Dependency breakage~~

Uptime risk

SECURITY

~~Retest after updates~~

No automation

Patches to the OS and kernel are hard

APPLICATION

Uptime risk

SECURITY

No automation

Uptime risk

Uptime risk

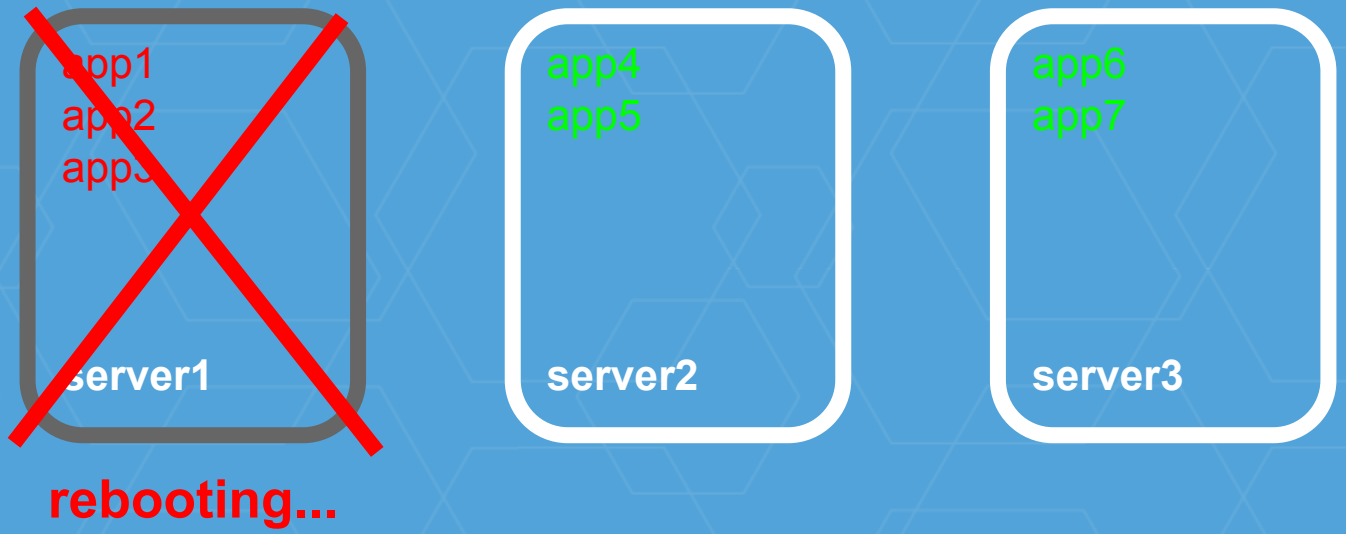
updating...

Uptime risk

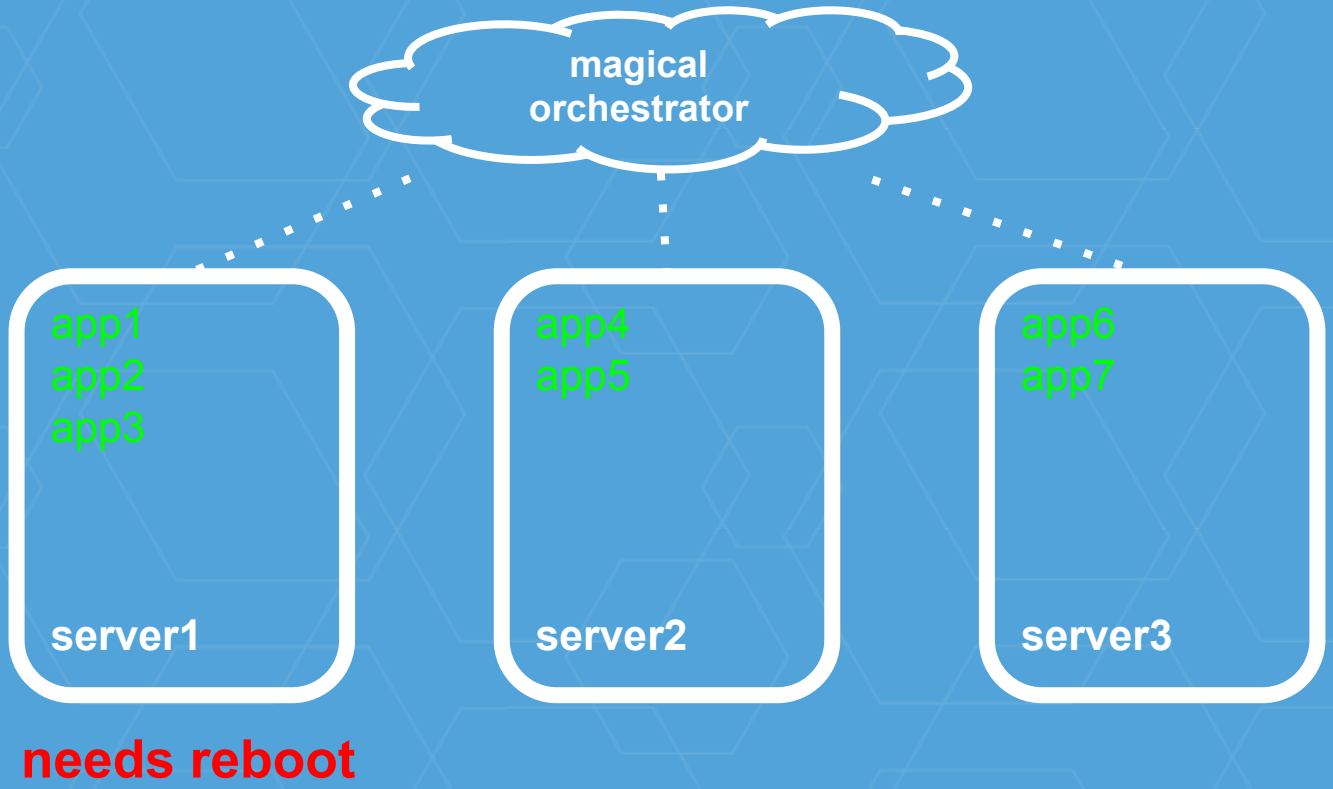

app1
app2
app3
server1
app4
app5
server2
app6
app7
server3
needs reboot

Without orchestration

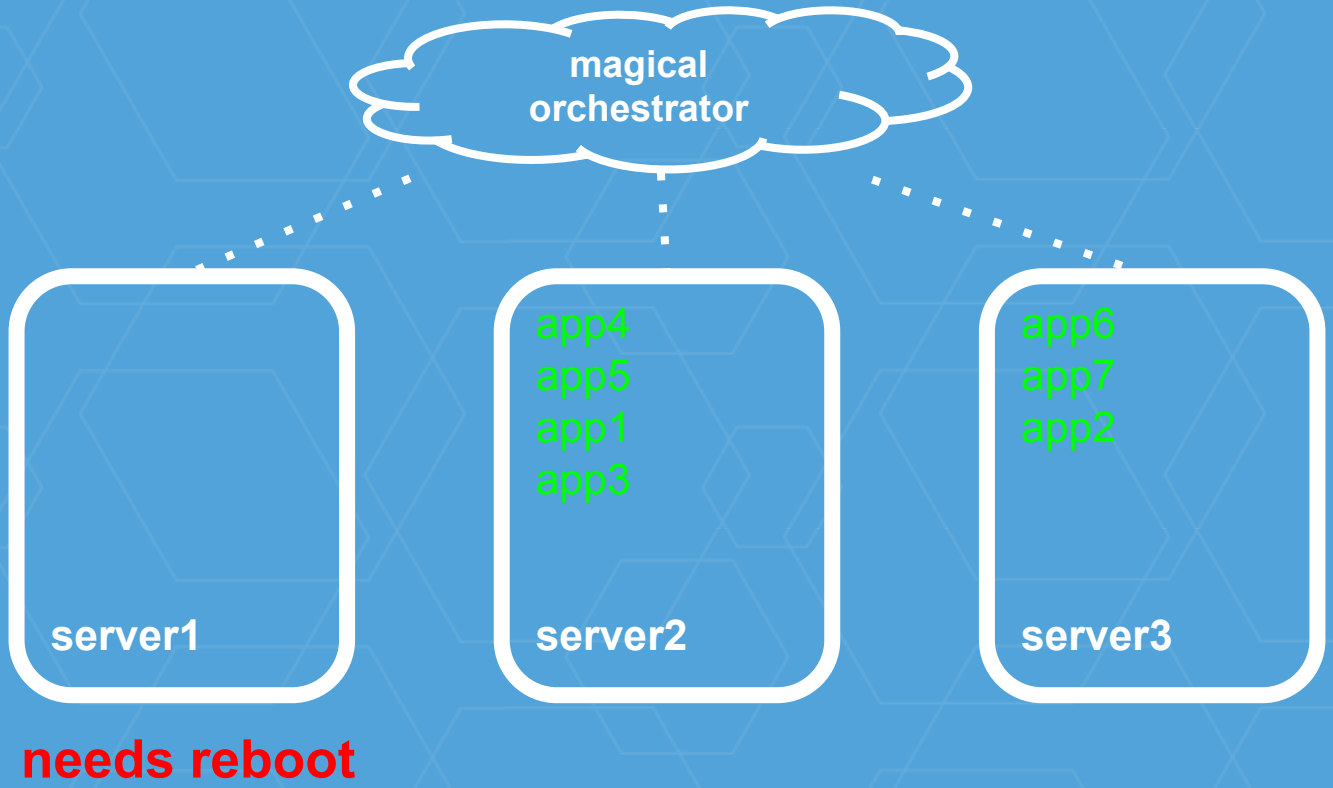
Without orchestration



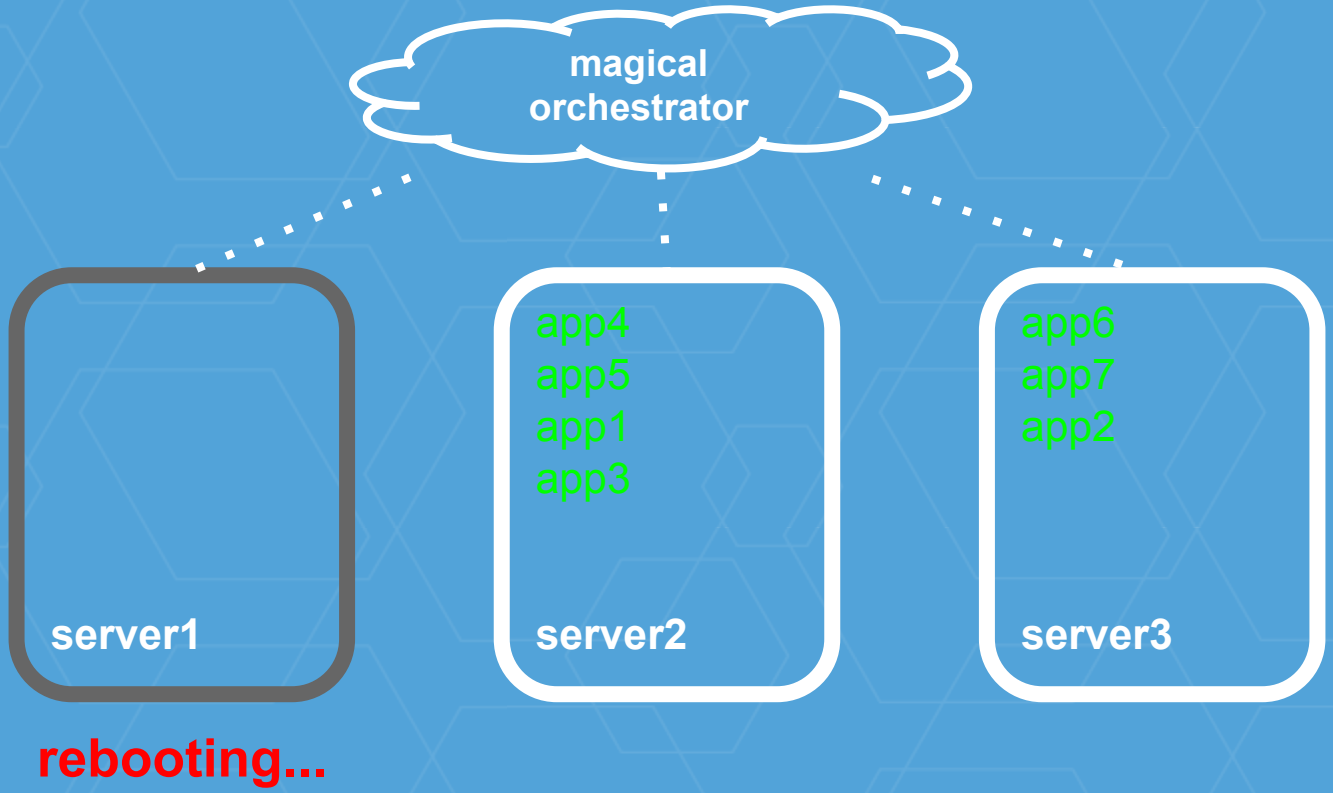
With orchestration



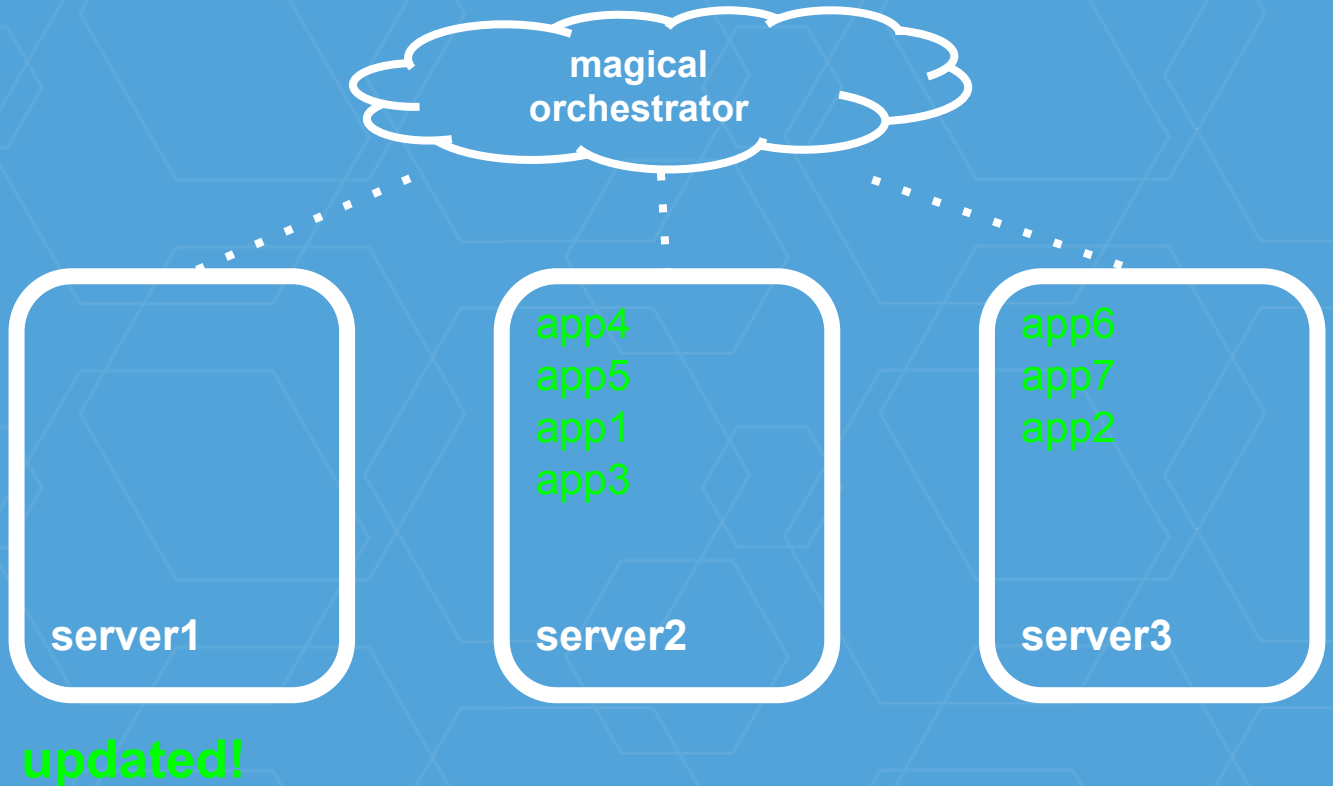
With orchestration



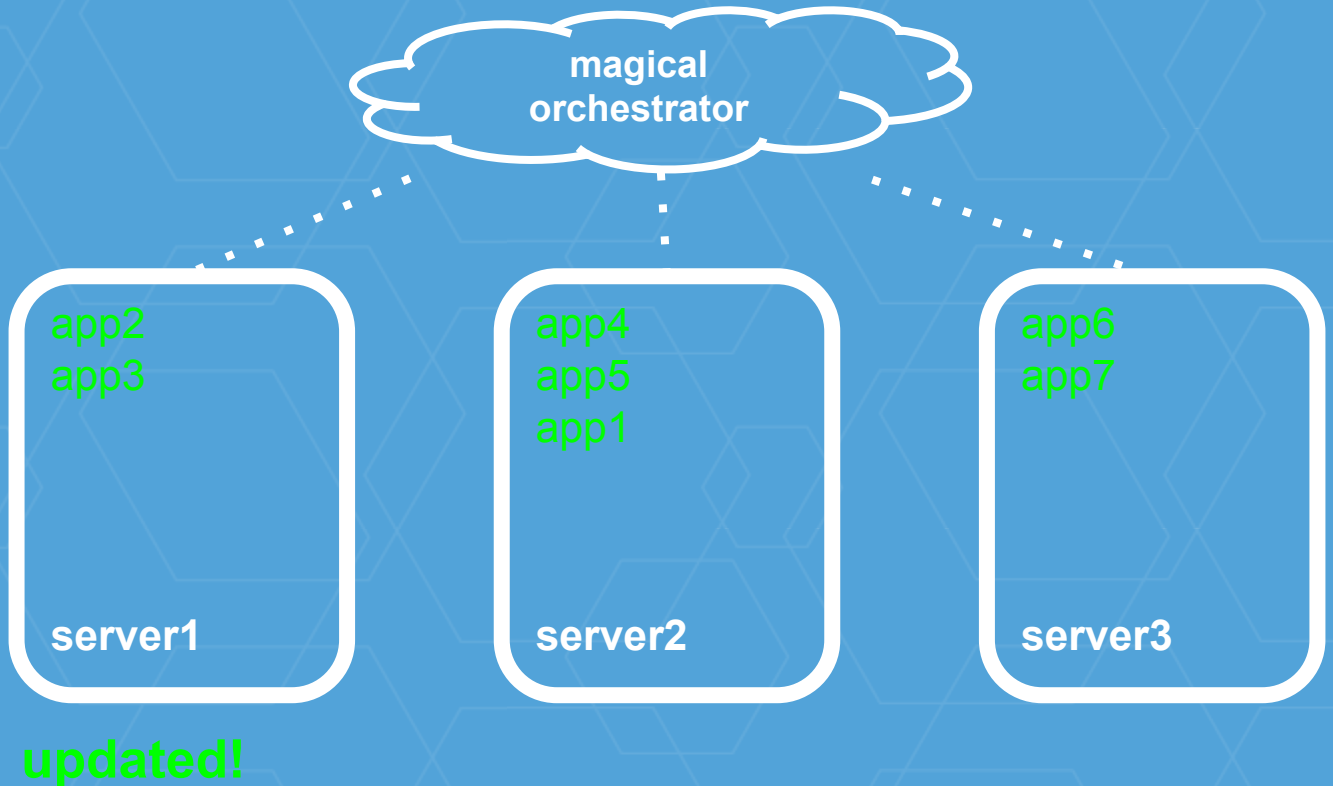
With orchestration



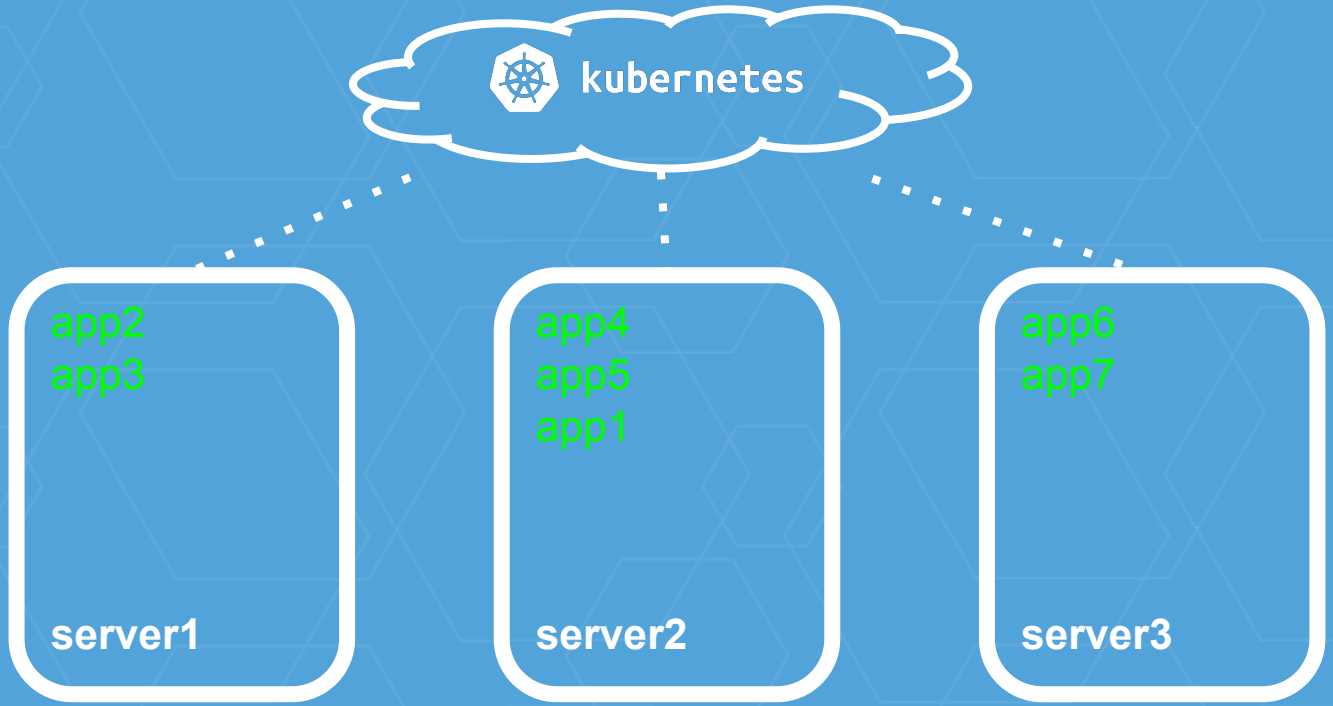
With orchestration



With orchestration

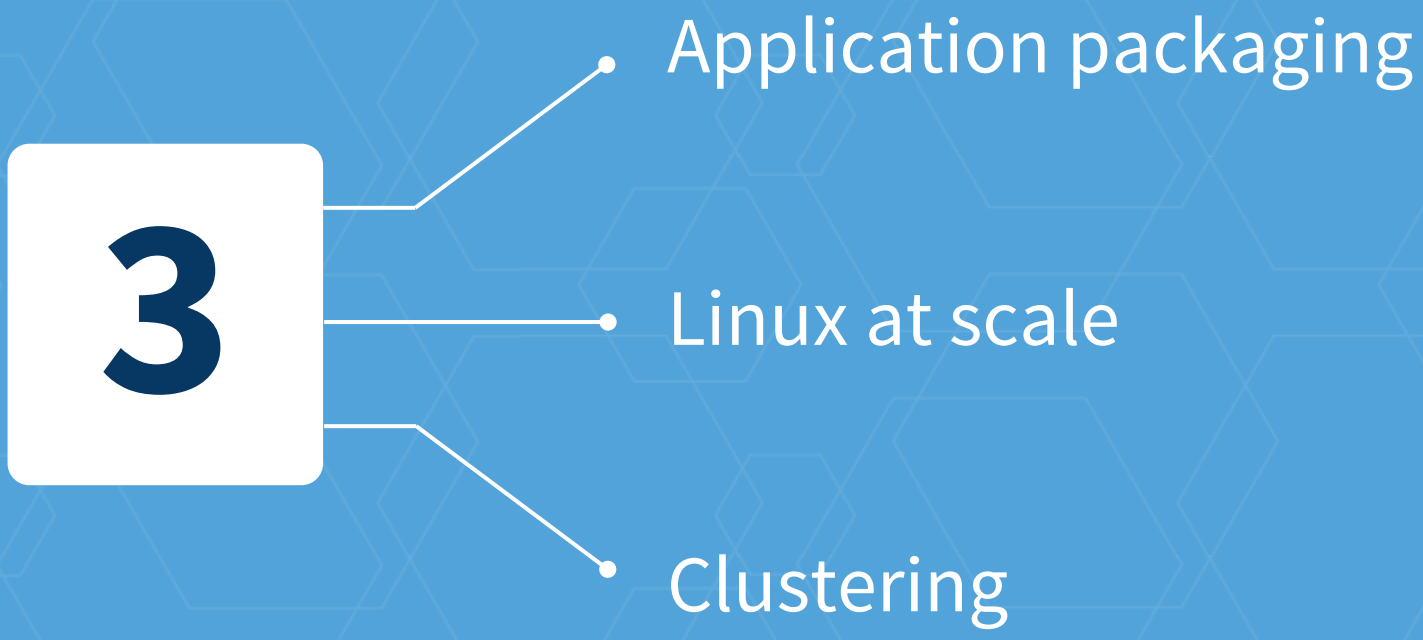


With orchestration



Let's talk about #GIFEE

Google Infrastructure For Everybody Else


3
Application packaging
Linux at scale
Clustering

Borg/Omega
Linux
Chubby

Google

Borg/Omega
Linux
Chubby

CoreOS

#GIFEE

Google

Borg/Omega
Linux
Chubby

Why build #GIFEE?

- Avoid single points of failure
- Regularly update software
- Consistent environment

CoreOS

Why build #GIFEE?

Regularly update software

1 Application Packaging

Containers

Run apps consistently
Separate from OS

What about the apps?

What about the apps?

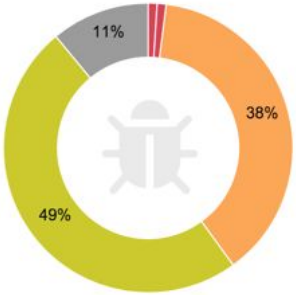
Vulnerabilities inside the containers?

- User code, we can't control
- But... we can help
- **Clair**: static vulnerability analyser

Clair container security auditing

- Search container metadata
- Identify vulnerabilities
- Explain update actions

4f3f3b6e0b74



Quay Security Scanner has detected **100** vulnerabilities.

- **1** Critical-level vulnerabilities.
- **1** High-level vulnerabilities.
- **38** Medium-level vulnerabilities.
- **49** Low-level vulnerabilities.
- **11** Negligible-level vulnerabilities.

Image Vulnerabilities

Filter Vulnerabilities...

Only display vulnerabilities with fixes

CVE	CVSS / SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN IMAGE
CVE-2014-9488	10	less	458-2	(None)	ADD file:9b5ba3935021955492697a...
CVE-2015-8391	9	pcre3	1:8.31-2ubuntu2.1	(None)	ADD file:9b5ba3935021955492697a...
CVE-2015-8380	7.5	pcre3	1:8.31-2ubuntu2.1	(None)	ADD file:9b5ba3935021955492697a...
CVE-2015-8472	7.5	libpng	1.2.50-1ubuntu2.14.04.1	1.2.50-1ubuntu2.14.04.2	ADD file:9b5ba3935021955492697a...
CVE-2015-8390	7.5	pcre3	1:8.31-2ubuntu2.1	(None)	ADD file:9b5ba3935021955492697a...

CoreOS

Clair container security auditing

After scanning millions of containers we found that over 80% still had Heartbleed

CoreOS

In-Progress
Universal Container Format

Packaged Downloaded Verified

CoreOS

Universal Container Format

Open Container Initiative: standardising what's inside the container

2 Linux at Scale

CoreOS

Container Linux

Run containers on any platform

3 Clustering

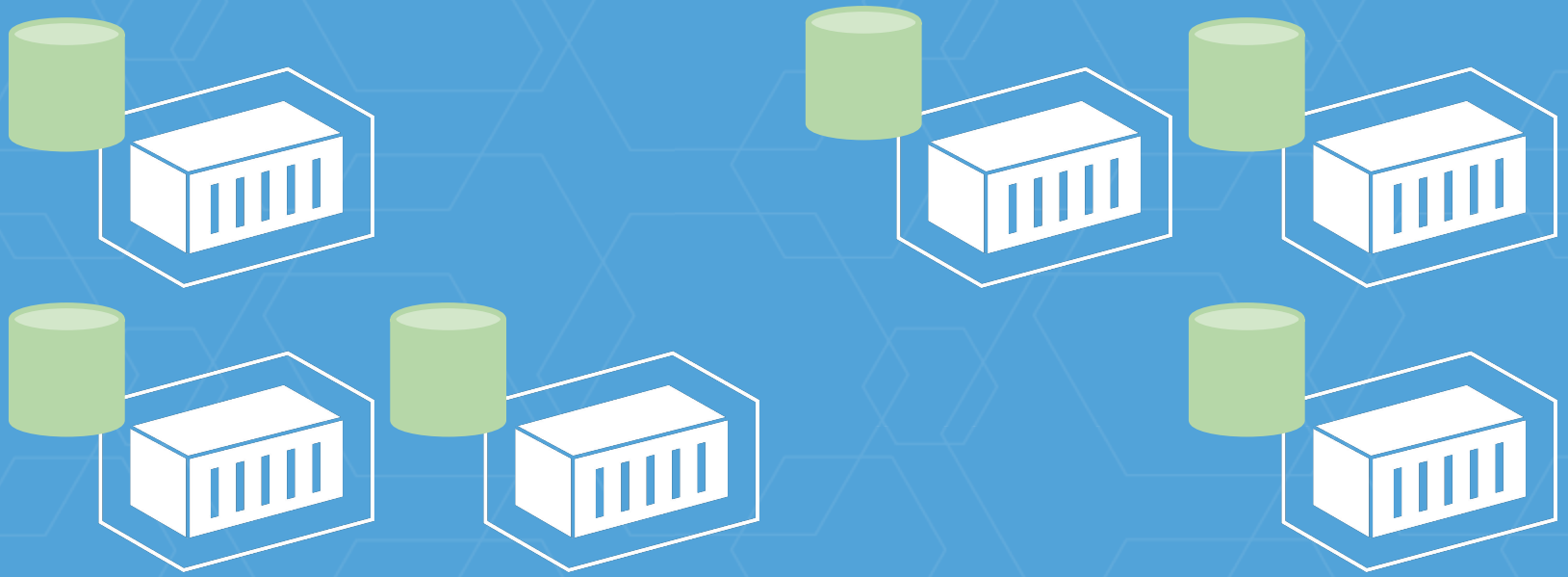
CoreOS

Operations Paradise

- Easy scale out
- Painless app upgrades
- Tolerant of machine failure

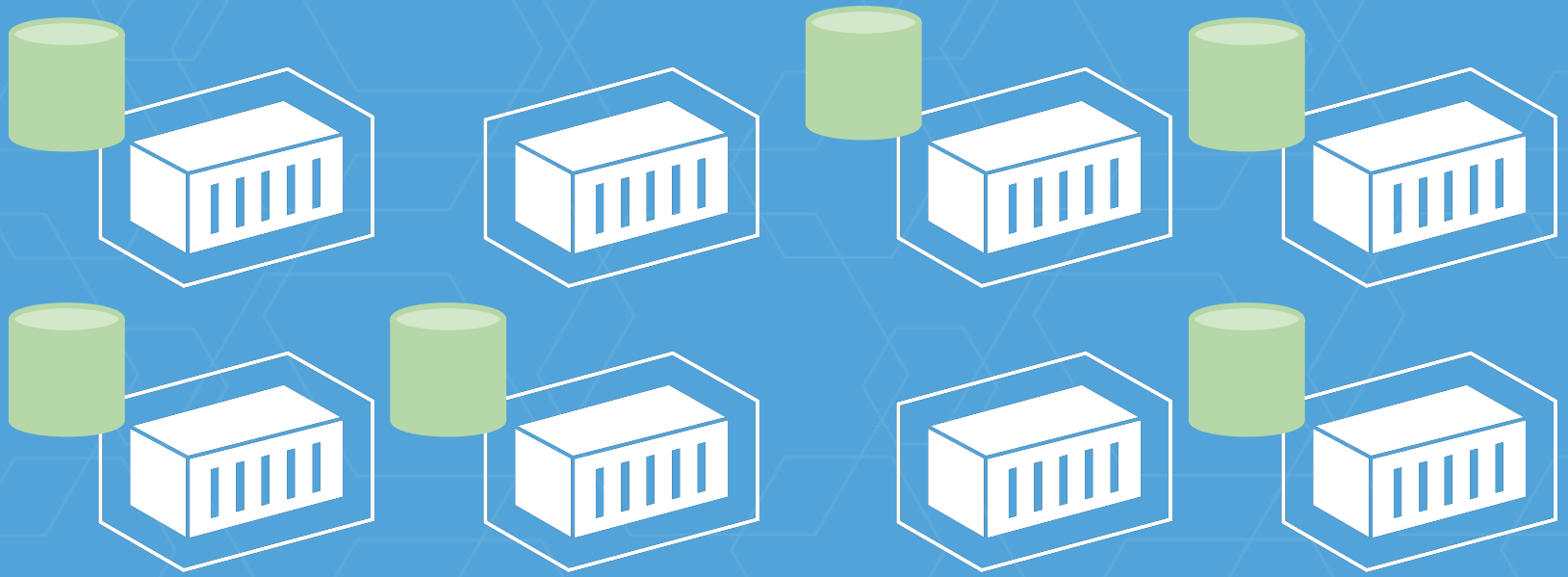
CoreOS

App Req/sec: **6,000**
App Healthy: **True**



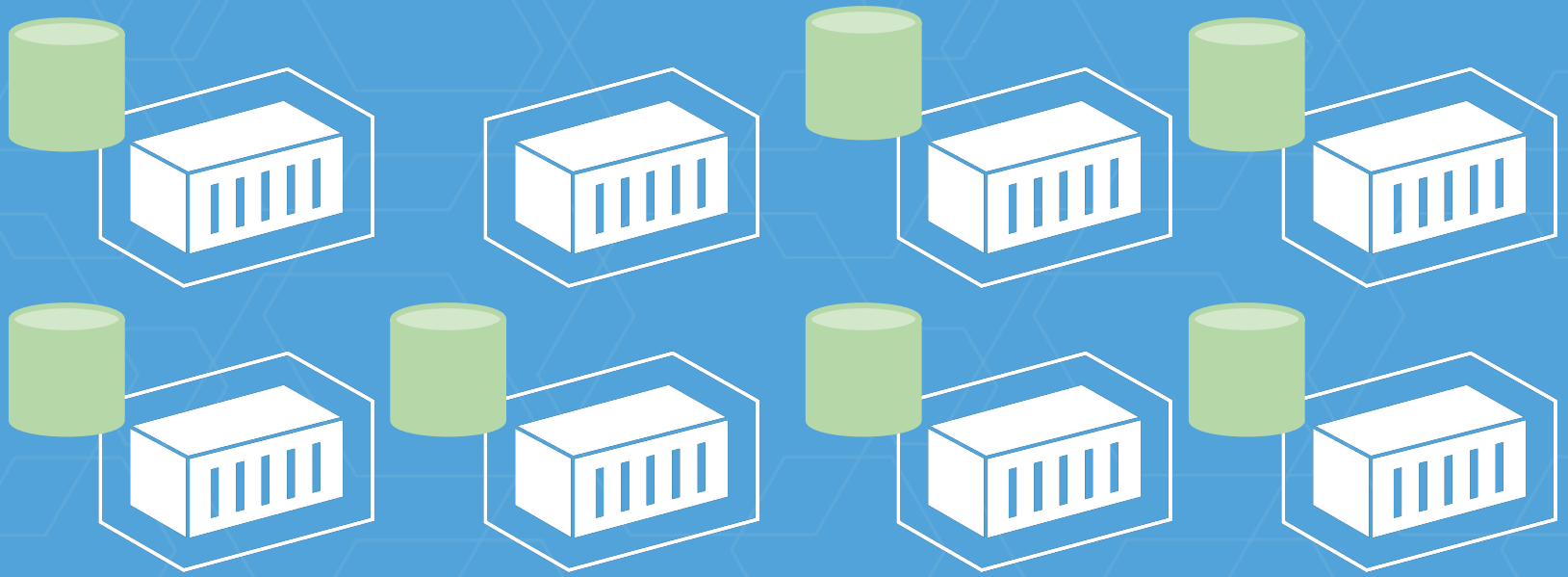
CoreOS

App Req/sec: **6,000**
App Healthy: **True**



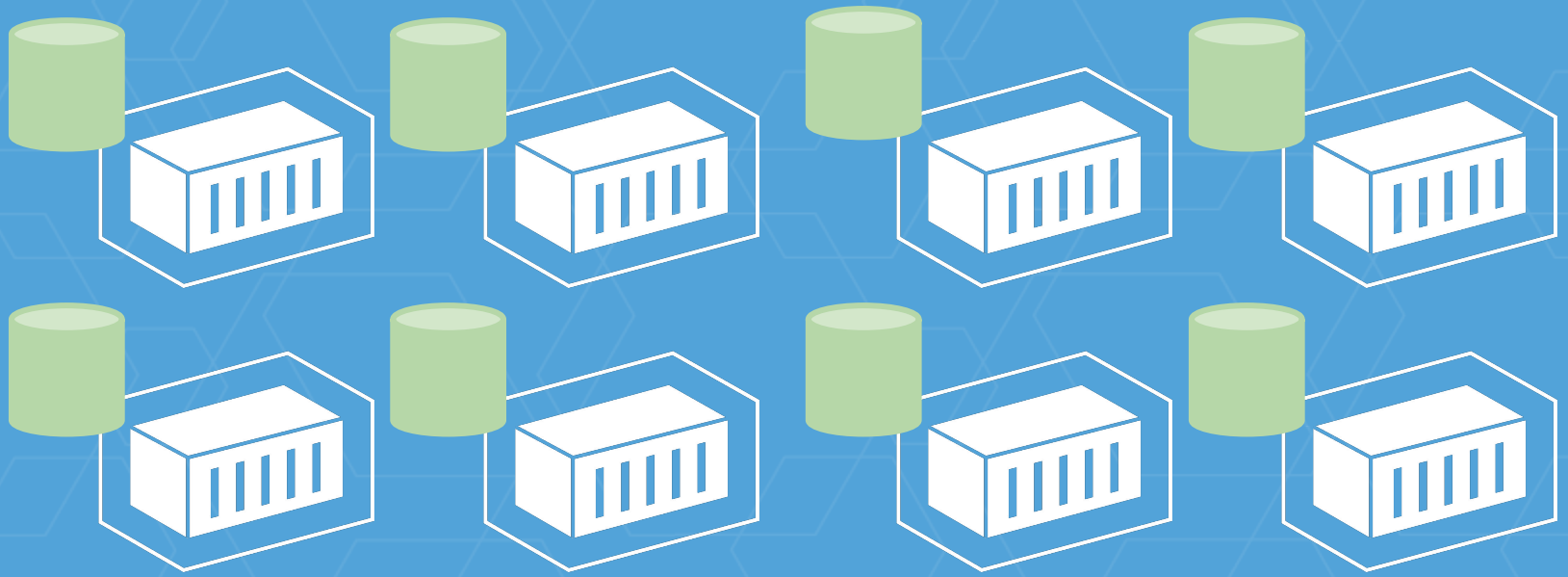
CoreOS

App Req/sec: **7,000**
App Healthy: **True**



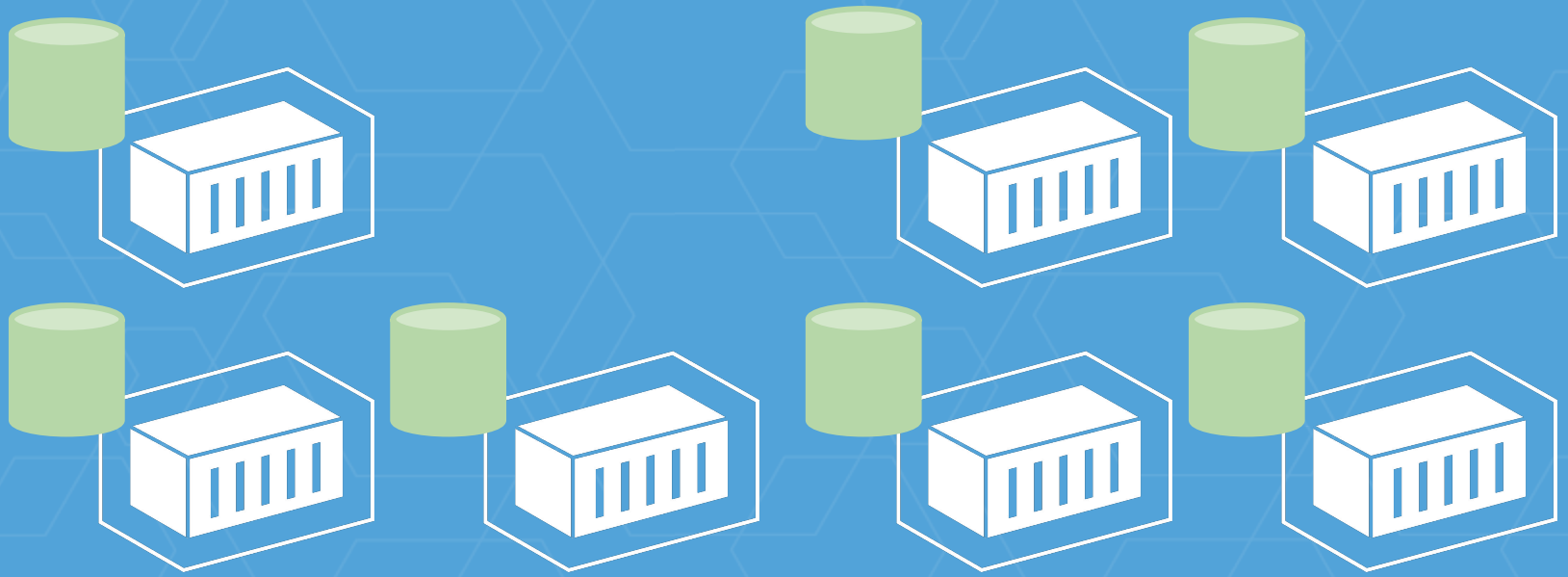
CoreOS

App Req/sec: **8,000**
App Healthy: **True**



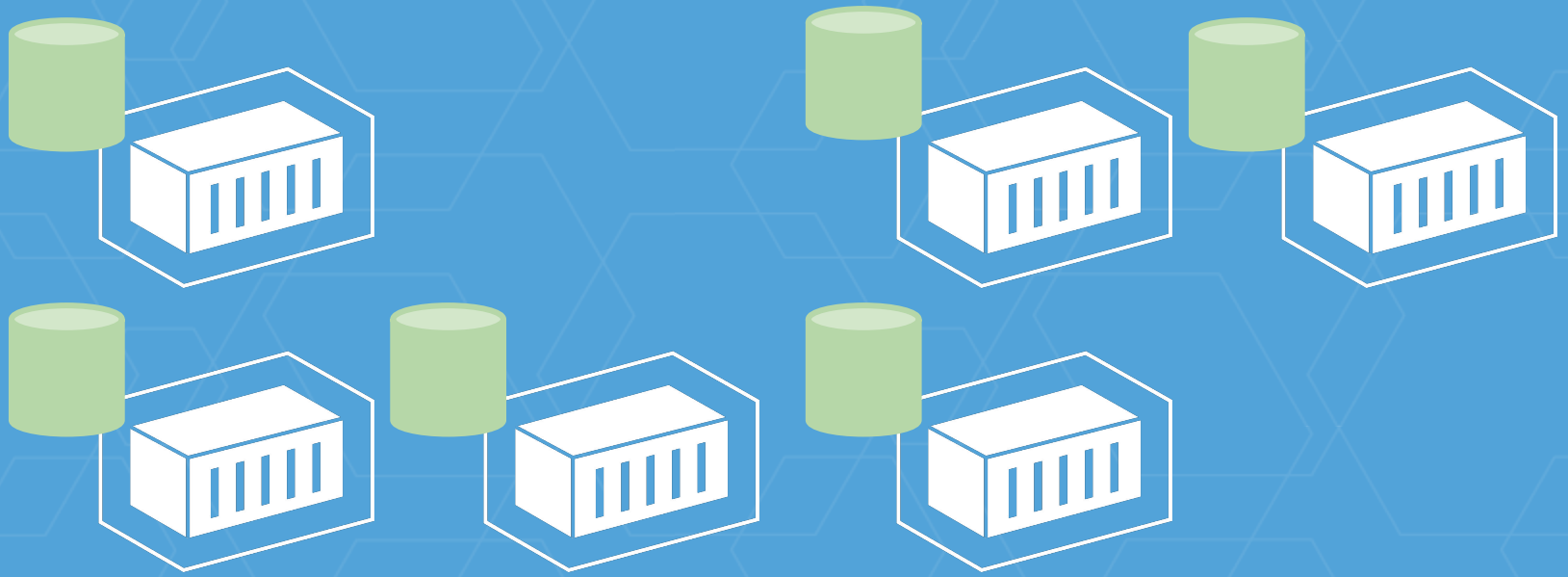
CoreOS

App Req/sec: **7,000**
App Healthy: **True**



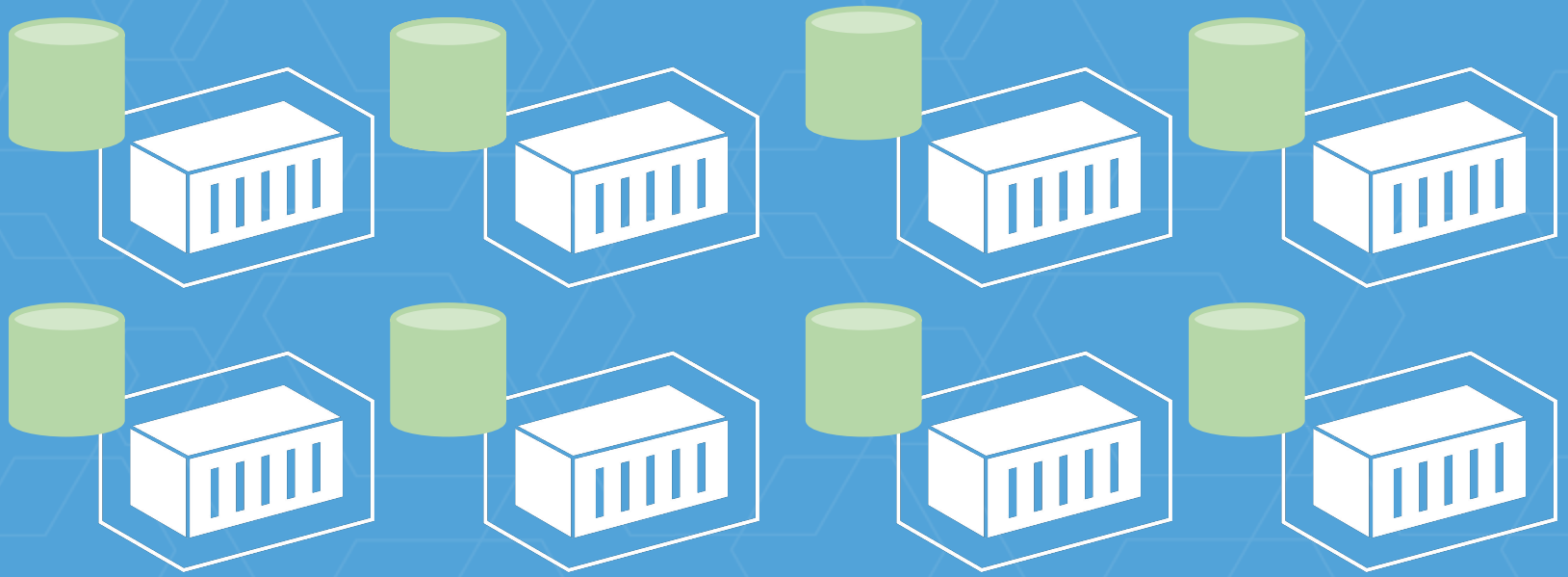
CoreOS

App Req/sec: **6,000**
App Healthy: **True**



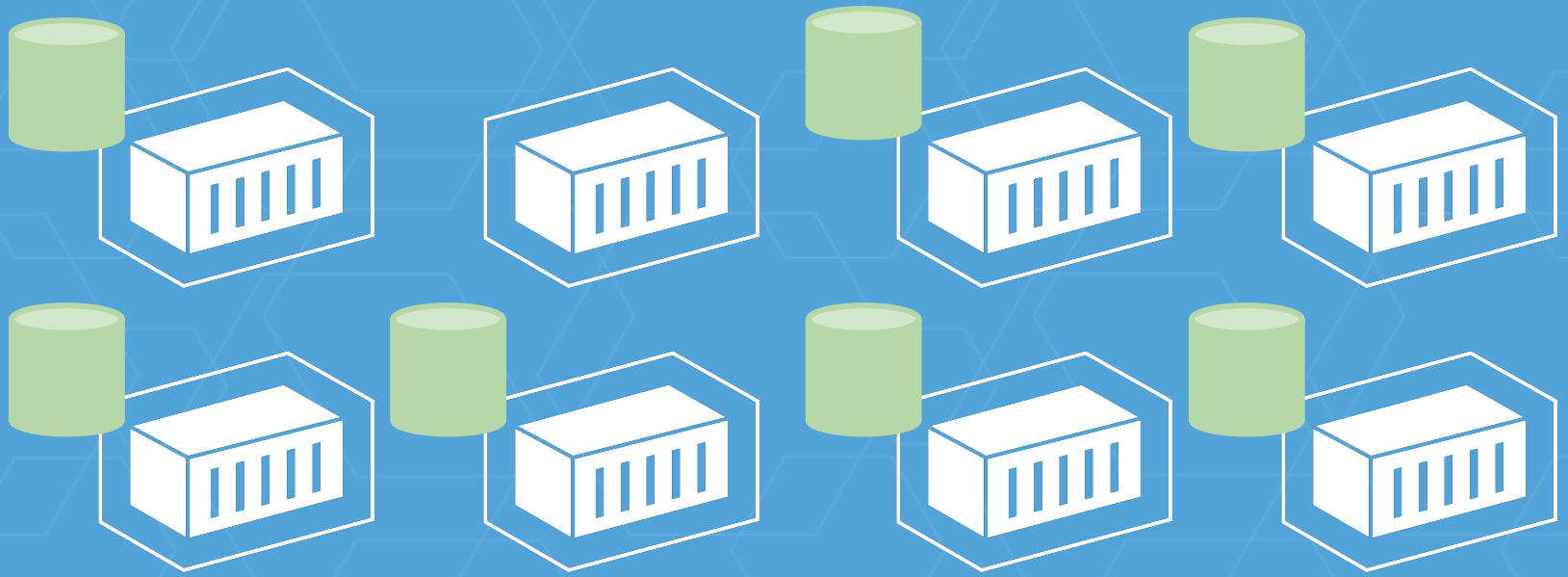
CoreOS

App Req/sec: **8,000**
App Healthy: **True**



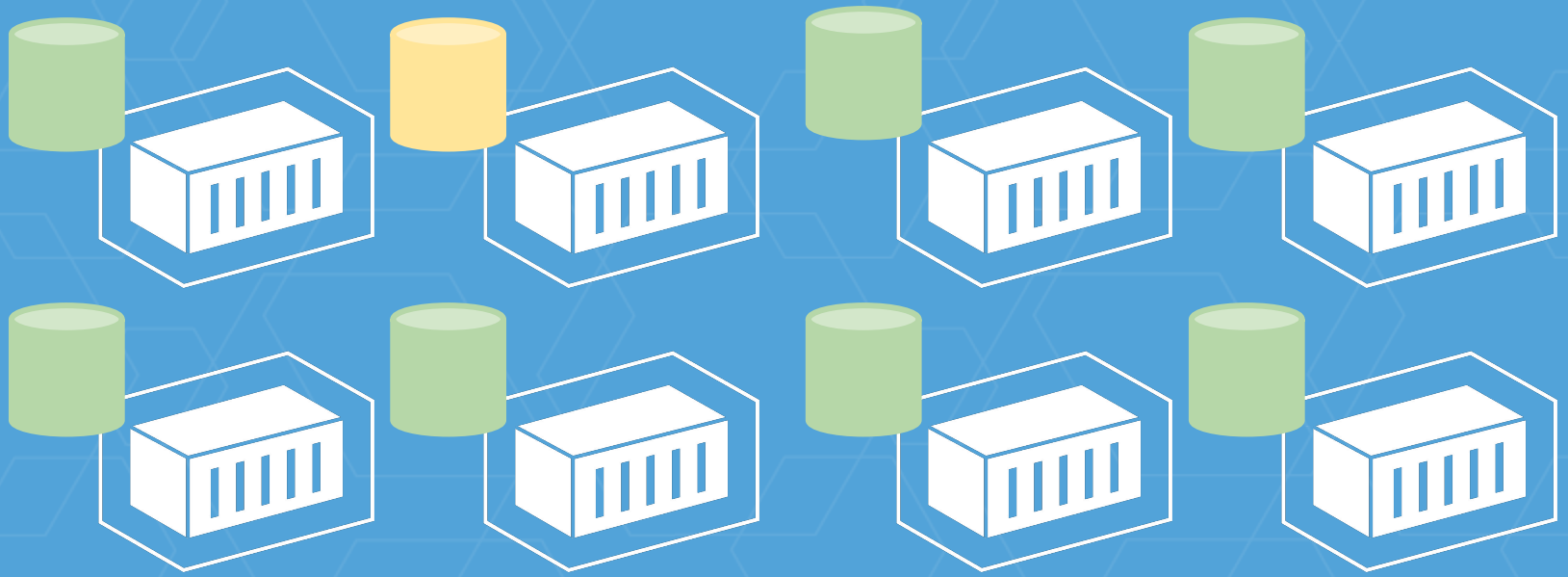
CoreOS

App Req/sec: **7,000**
App Healthy: **True**



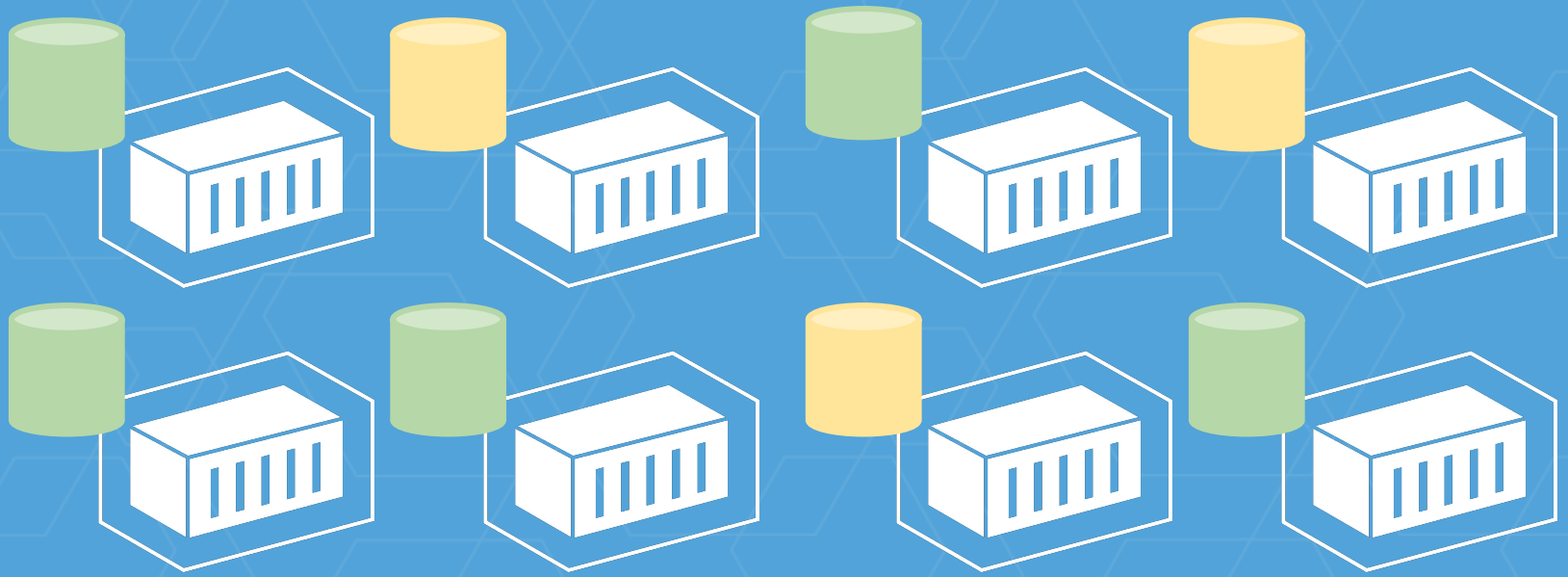
CoreOS

App Req/sec: **8,000**
App Healthy: **True**



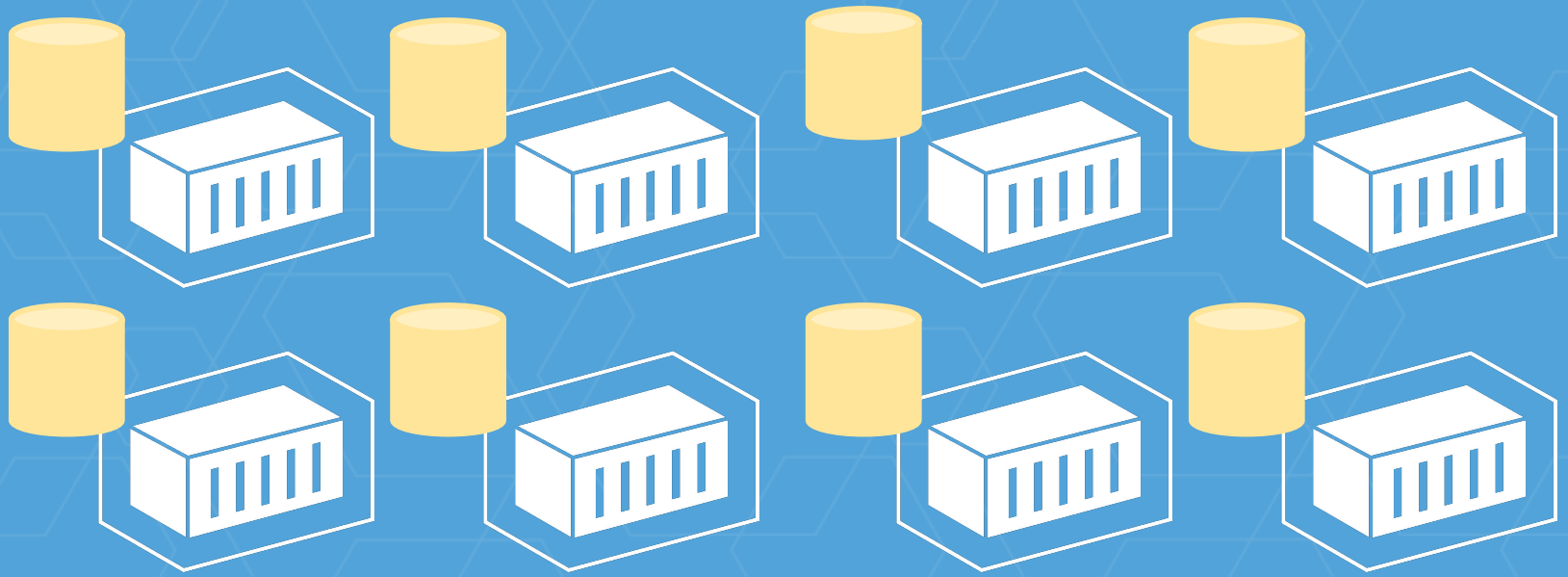
CoreOS

App Req/sec: **8,000**
App Healthy: **True**



CoreOS

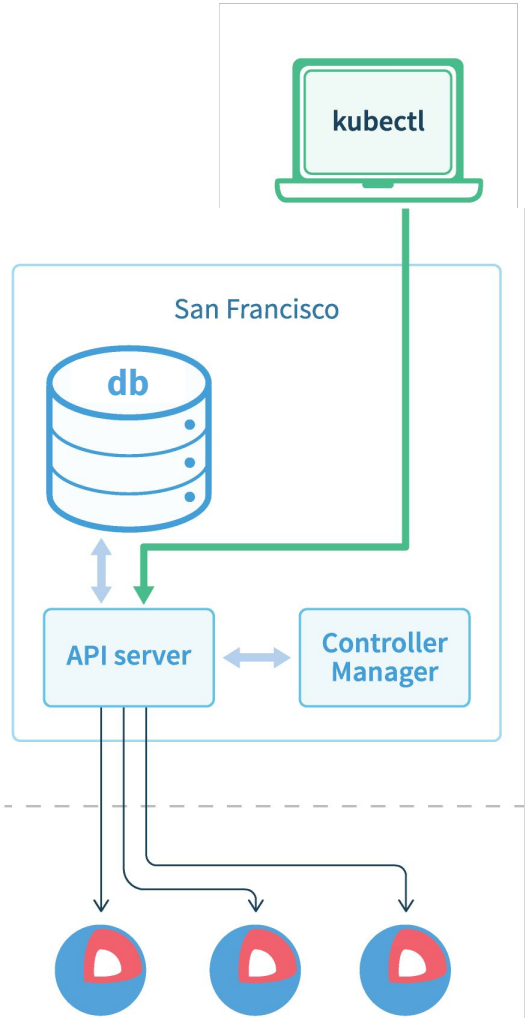
App Req/sec: **8,000**
App Healthy: **True**



CoreOS

kubernetes

API for easing scaling, painless upgrades, and failure tolerance



Simple cluster operations

Secure and Simple API

Friendly operational tools

CoreOS

Fault-tolerant database to support complex set of needs

- Leader election
- Cluster-wide Semaphores
- Dynamic configuration
- Service discovery

CoreOS

Hard Computer Science Problem

CoreOS

Hard Computer Science Problem

Chubby

CoreOS

Distributed reliable key-value store for the most critical data of a distributed system

Why build etcd?

- ✖ No existing “cloud native” solutions
- ✚ Simple HTTP + JSON APIs
- 🔄 Dynamic reconfiguration

Simple key/value

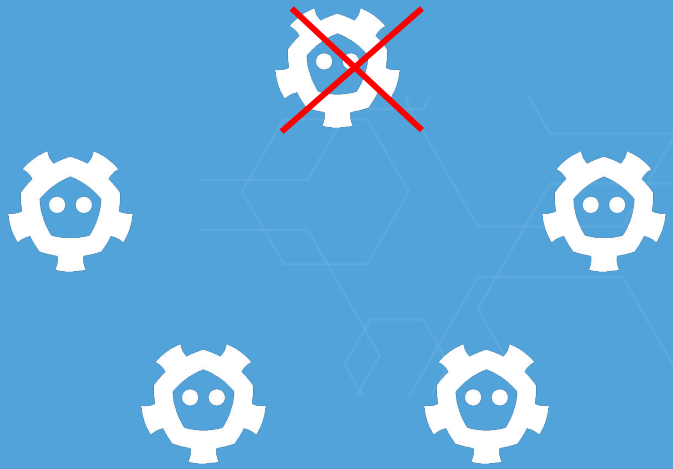
- “Distributed etc”
- Keys are versioned
- Changes can be watched

How does etcd work?

- Replicated log to model a state machine
- Raft: *"In Search of an Understandable Consensus Algorithm"* (Ongaro, 2014)
- Three key concepts: ***Leaders***, ***Elections***, ***Terms***
- etcd clusters elect a leader; all state changes performed by that leader

Available

Available



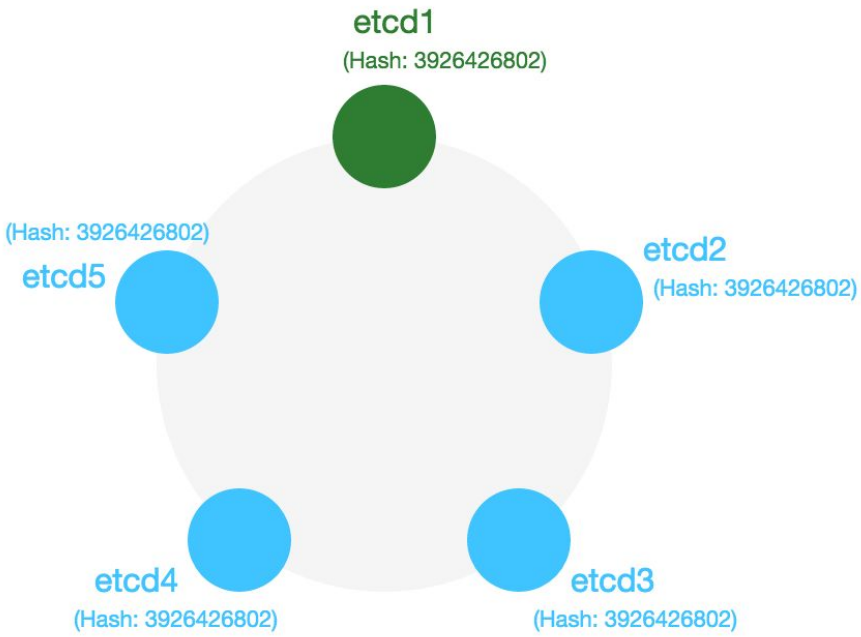
Available

Unavailable

Try it out!

play.etcd.io

Play etcd Home Code Issues



etcd1 etcd2 etcd3 etcd4 etcd5

PUT GET DELETE Submit

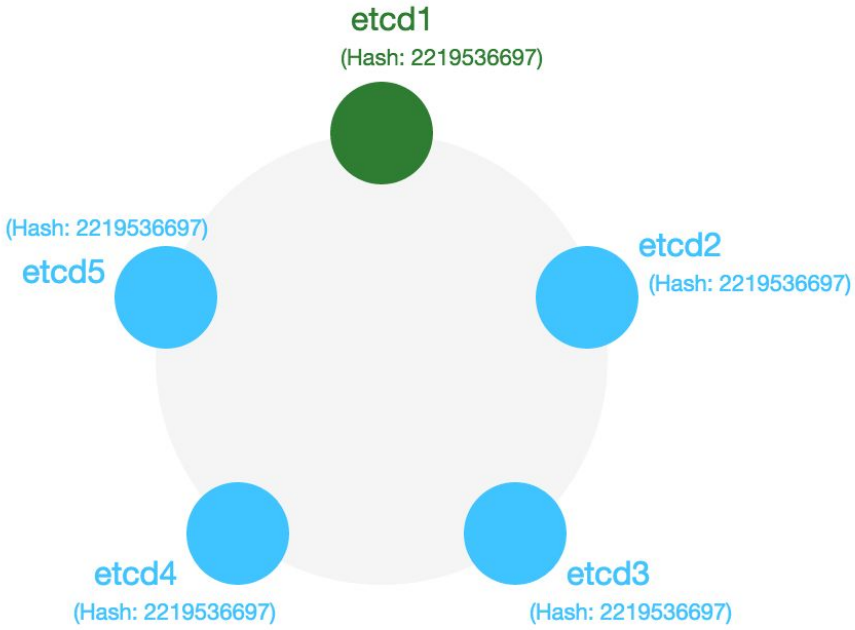
foo

BAZ

[PUT] success! "foo" : "BAZ" (at 2016-03-16 05:36:42 PST)

Log

Play etcd Home Code Issues



etcd1 etcd2 etcd3 etcd4 etcd5

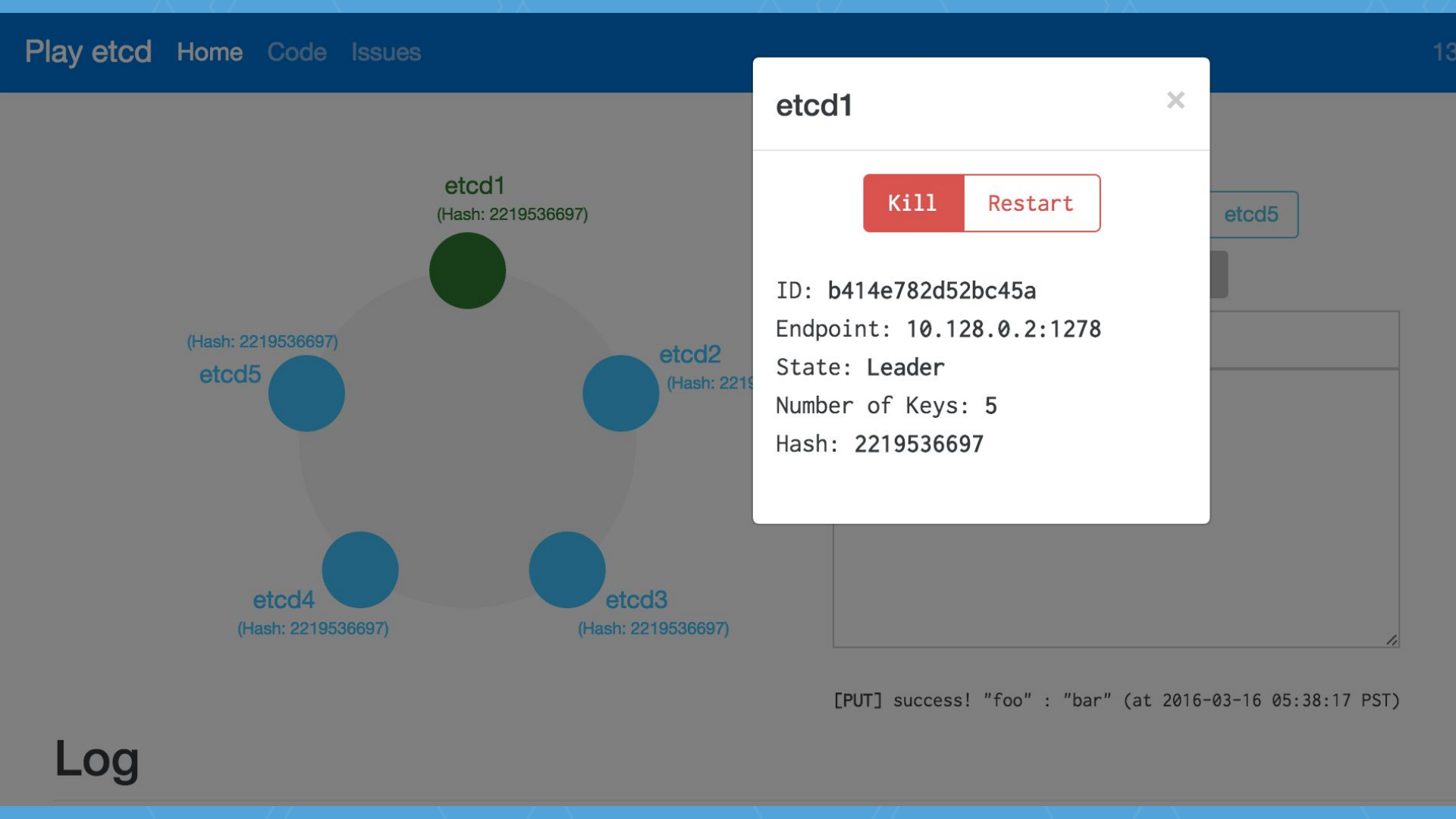
PUT GET DELETE Submit

foo

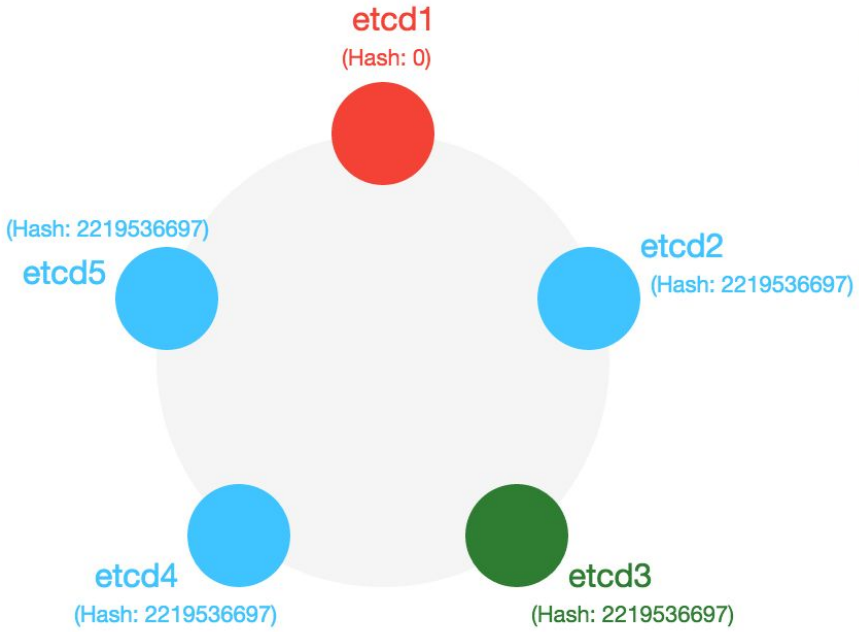
bar

[PUT] success! "foo" : "bar" (at 2016-03-16 05:38:17 PST)

Log


Play etcd
Home
Code
Issues
etcd1
(Hash: 2219536697)
etcd2
(Hash: 221
(Hash: 2219536697)
etcd5
etcd4
(Hash: 2219536697)
etcd3
(Hash: 2219536697)
etcd1
Kill
Restart
ID: b414e782d52bc45a
Endpoint: 10.128.0.2:1278
State: Leader
Number of Keys: 5
Hash: 2219536697
etcd5
[PUT] success! "foo" : "bar" (at 2016-03-16 05:38:17 PST)
Log

Play etcd Home Code Issues



etcd1 etcd2 etcd3 etcd4 etcd5

PUT GET DELETE Submit

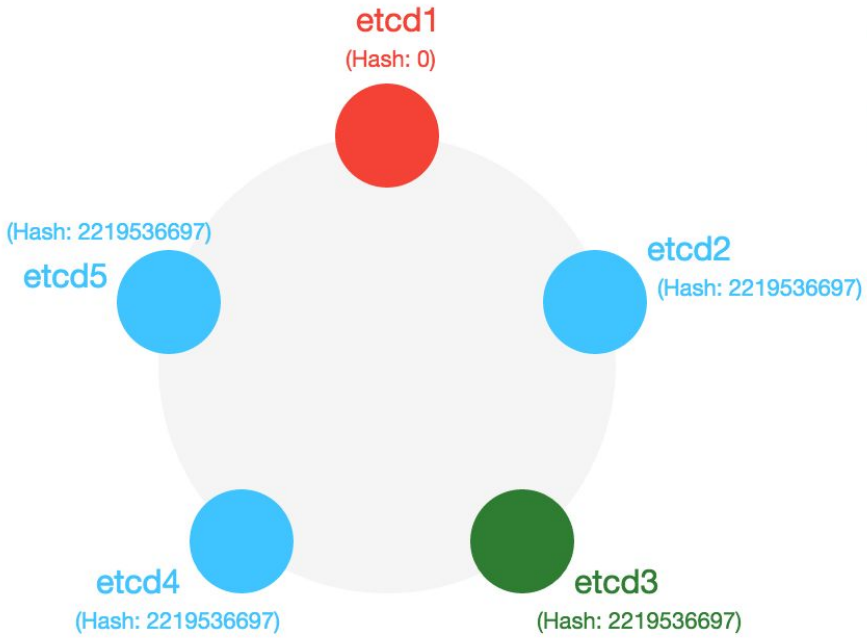
foo

bar

[PUT] success! "foo" : "bar" (at 2016-03-16 05:38:17 PST)

Log

Play etcd Home Code Issues



etcd1 etcd2 etcd3 etcd4 etcd5

PUT GET DELETE Submit

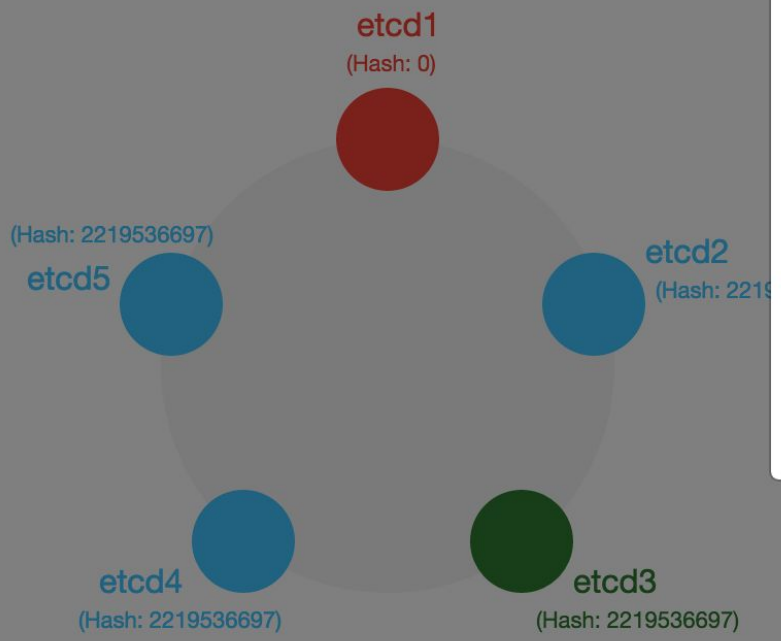
foo

BAZ

[PUT] success! "foo" : "bar" (at 2016-03-16 05:38:17 PST)

Log

Play etcd Home Code Issues



etcd1

Kill Restart

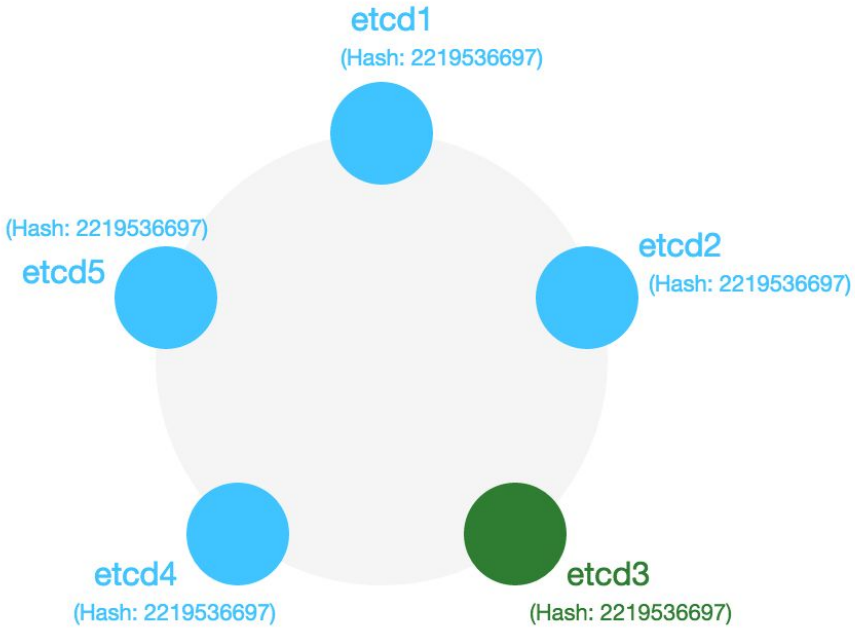
ID: unknown
Endpoint: 10.128.0.2:1278
State: unreachable
Number of Keys: 0
Hash: 0

etcd5

[PUT] error grpc: timed out trying to connect (key: "foo")

Log

Play etcd Home Code Issues



etcd1 | etcd2 | etcd3 | etcd4 | etcd5

PUT | GET | DELETE | Submit

foo

BAZ

[PUT] error grpc: timed out trying to connect (key: "foo")

Log

CoreOS

- Cluster-wide reboot lock - *locksmith*
- Service discovery - *vulcand, skydns*
- Cluster orchestration - *k8s, cloud foundry*

locksmith

- cluster wide reboot lock
  - "semaphore for reboots"
- Container Linux updates happen *automatically*
  - stop all the machines restarting at once...

Cluster Wide Reboot Lock

server1

server2

server3

needs reboot

Cluster Wide Reboot Lock

- Need to reboot? Decrement the semaphore key (*atomically*) with etcd.
- `manager.Reboot()` and wait...
- After reboot, increment the semaphore key in etcd (*atomically*).

Cluster Wide Reboot Lock

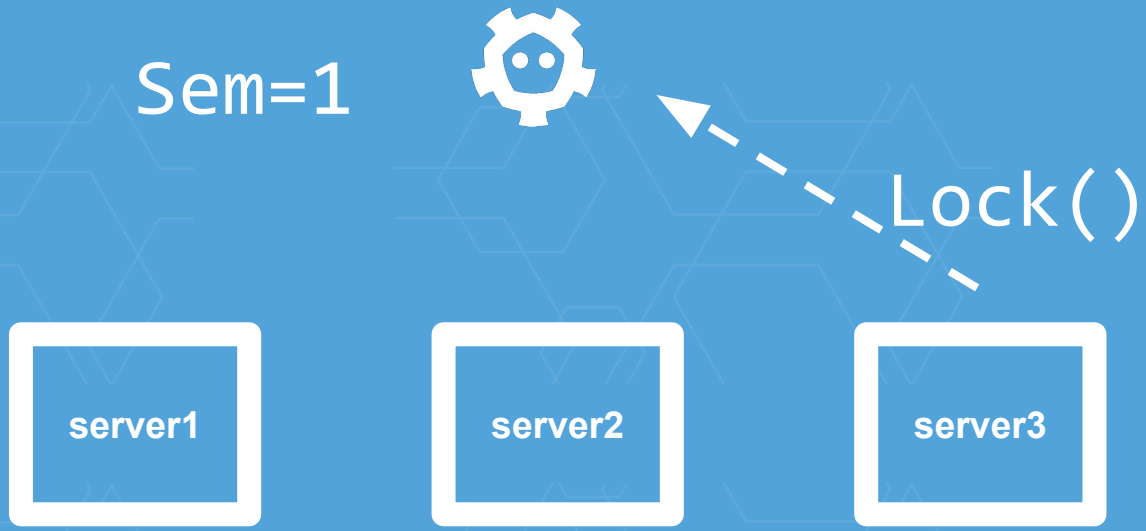
Sem=1

server1

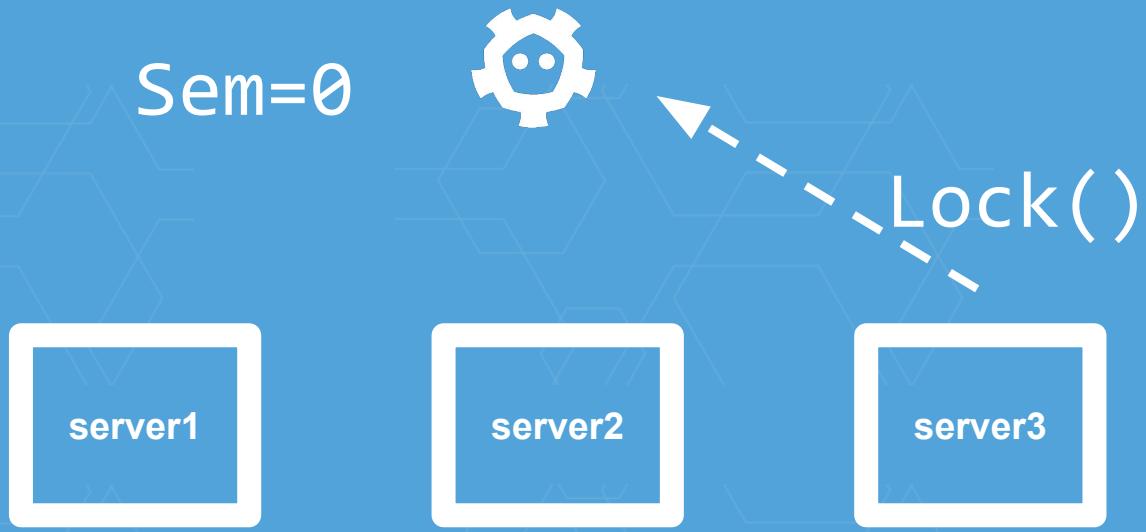
server2

server3

Cluster Wide Reboot Lock



Cluster Wide Reboot Lock



Cluster Wide Reboot Lock

Sem=0

server1

server2

server3

Reboot()

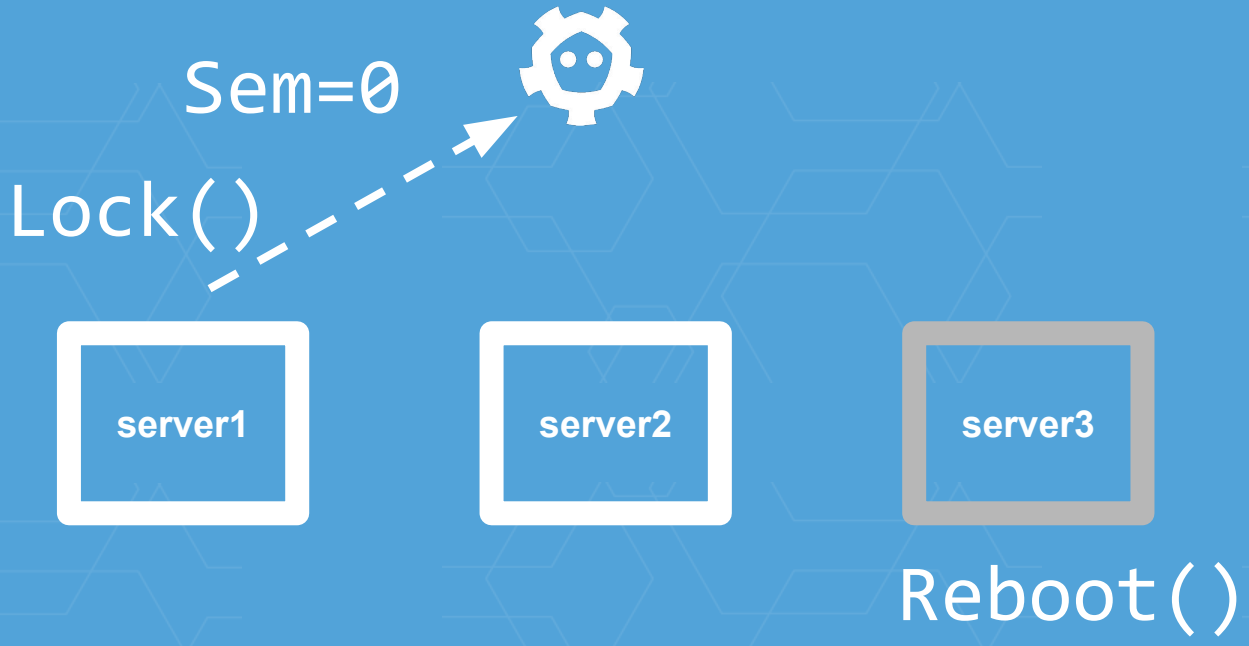
Cluster Wide Reboot Lock

Sem=0

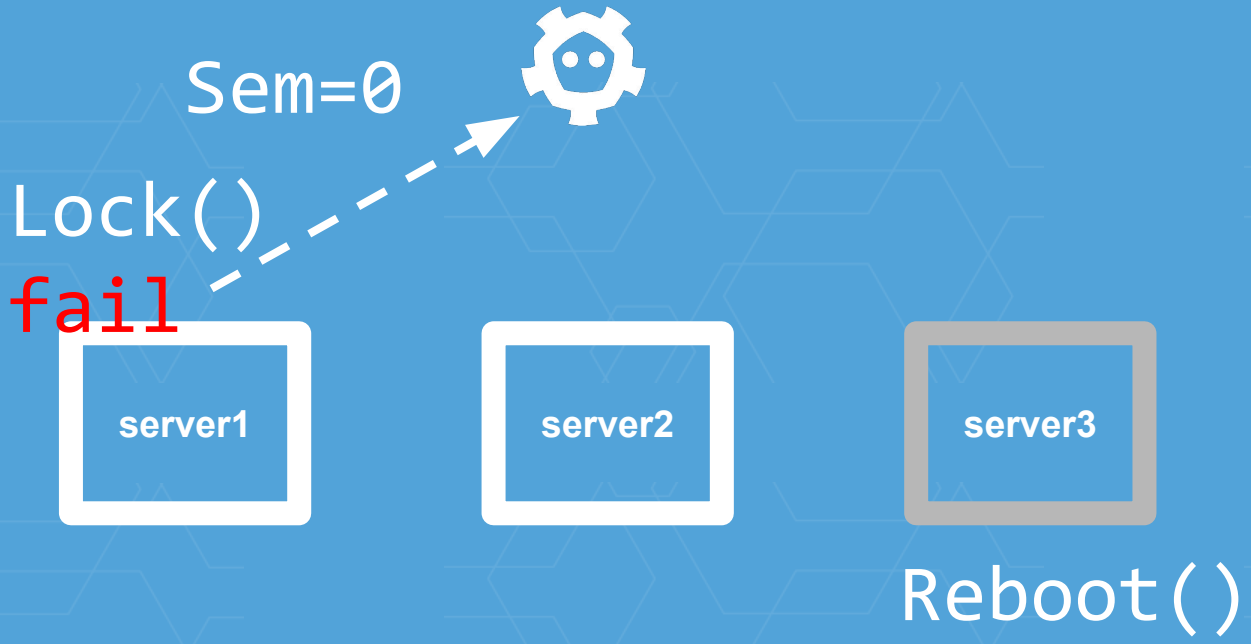
needs reboot

Reboot()

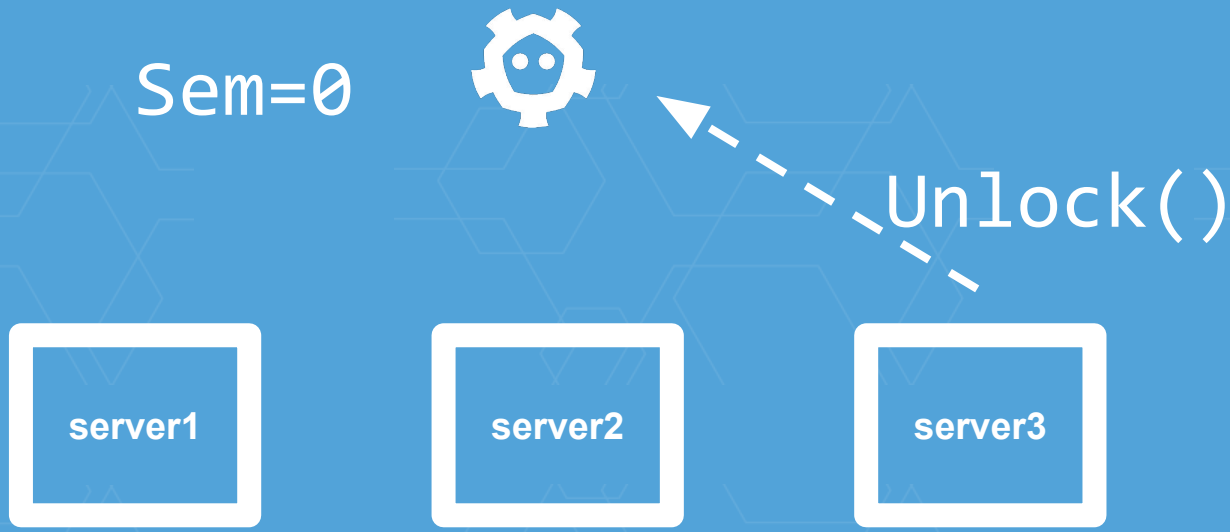
Cluster Wide Reboot Lock



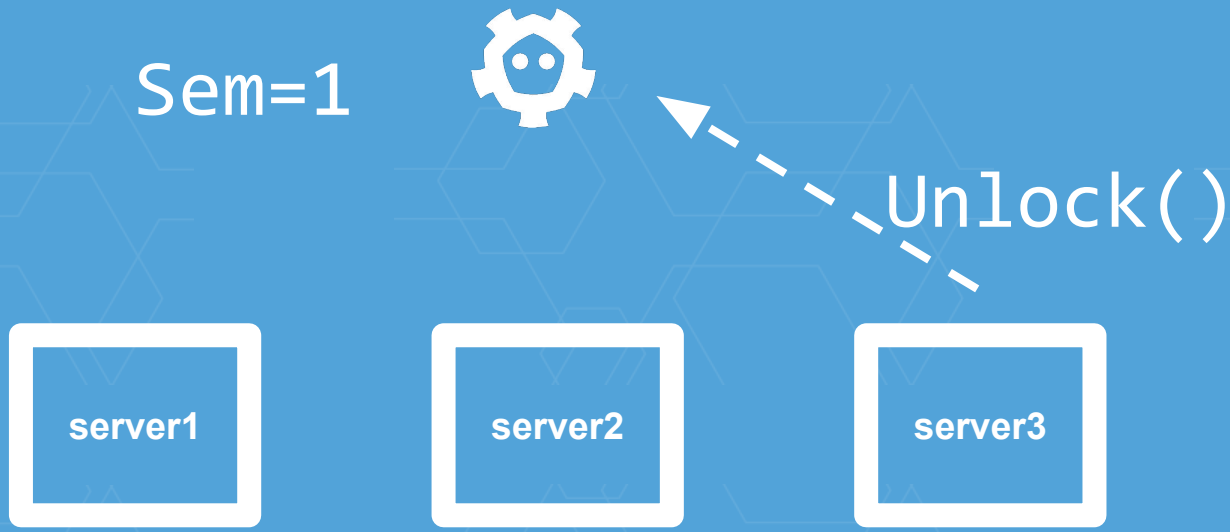
Cluster Wide Reboot Lock



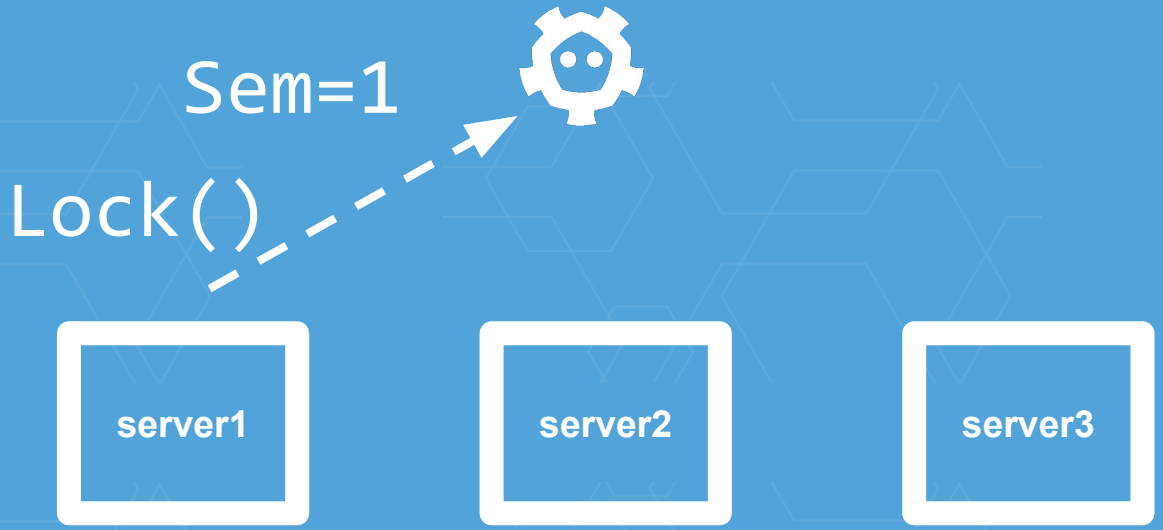
Cluster Wide Reboot Lock



Cluster Wide Reboot Lock



Cluster Wide Reboot Lock



CoreOS

- Cluster-wide reboot lock - *locksmith*
- Service discovery - *vulcand, skydns*
- Cluster orchestration - *k8s, cloud foundry*

CoreOS

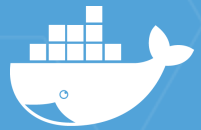
Industry Adoption

kubernetes

CLOUD FOUNDRY

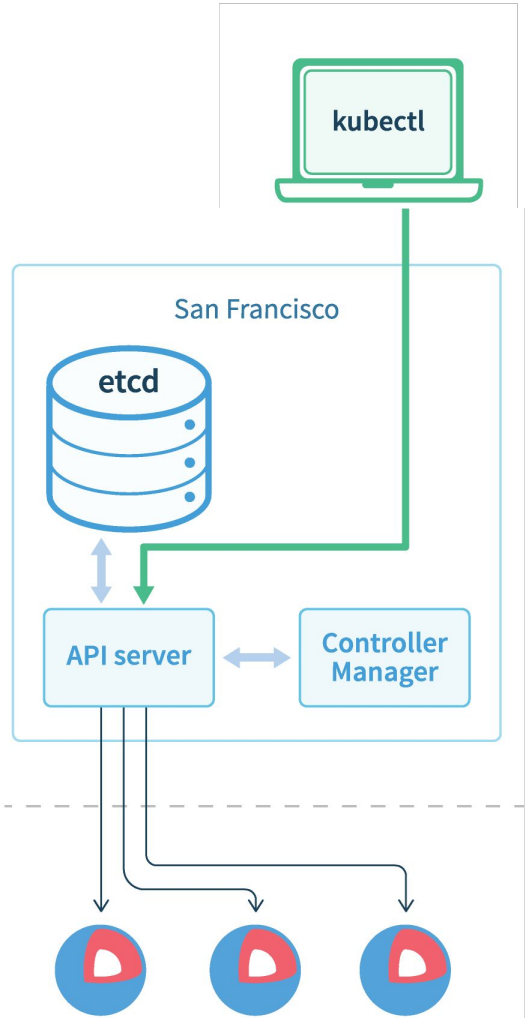
MESOS

DEIS



docker

500+ projects on Github



Simple cluster operations

Secure and Simple API

Friendly operational tools

CoreOS

#GIFEE

Google

Borg/Omega
Linux
Chubby

#GIFEE

Kubernetes
Container Linux
etcd

CoreOS

MISSION

Secure the Internet

CoreOS

STRATEGY

Separate Apps from OS

CoreOS

STRATEGY

Make Servers Consistent

CoreOS

STRATEGY

Make Servers Easy to Upgrade

CoreOS

STRATEGY

Simplify Application Management

CoreOS

STRATEGY

On Any Infrastructure

Kubernetes Everywhere

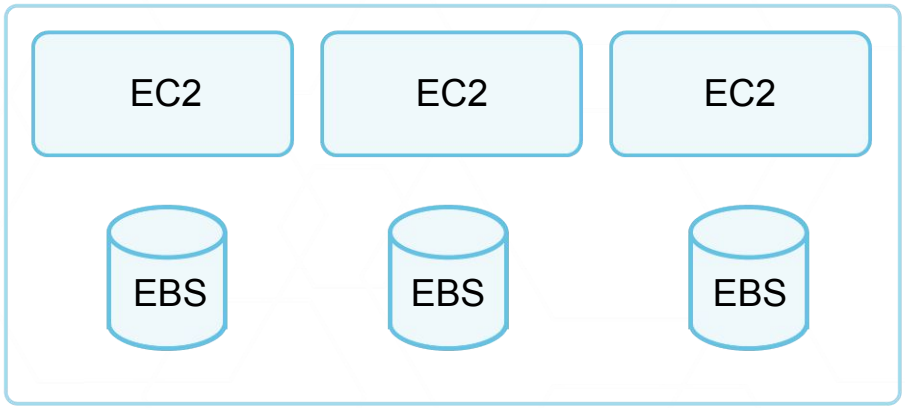
Consistent Infrastructure Everywhere

kubectl

k8s API

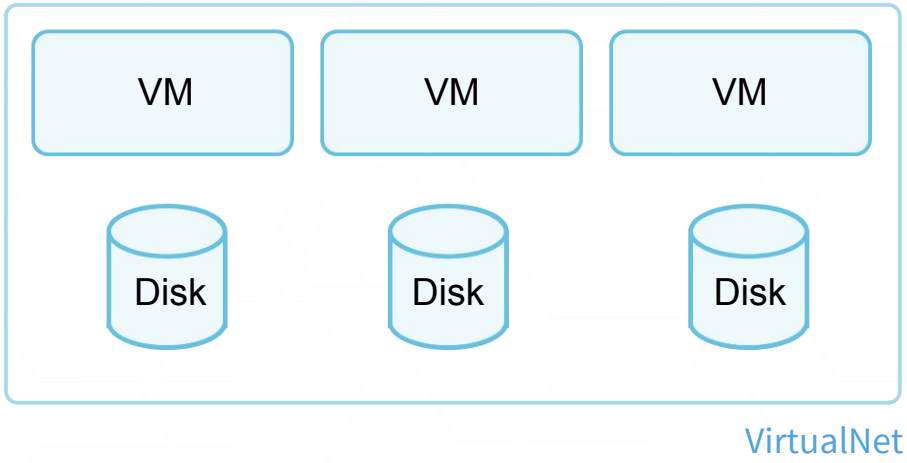
AWS APIs

kubernetes v1.5.2



kubectl

kubernetes v1.5.2



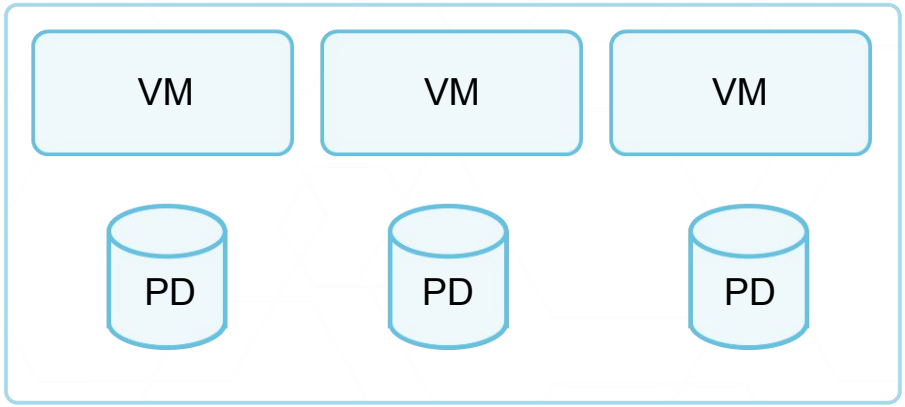
kubectl

k8s API

Google APIs

kubernetes

v1.5.2



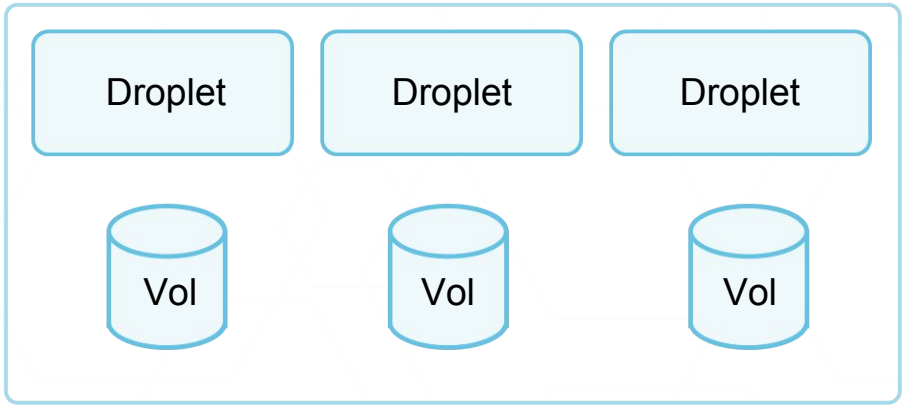
VirtualNet

kubectl

k8s API

Digitalocean APIs

kubernetes v1.5.2



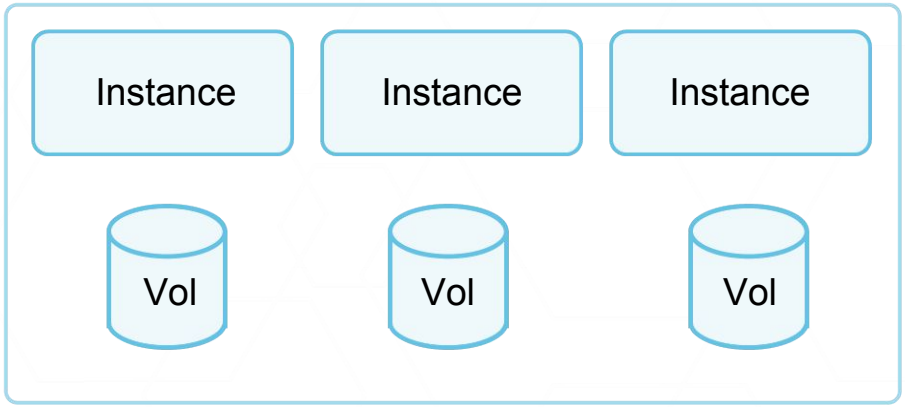
kubectl

k8s API

OpenStack APIs

kubernetes

v1.5.2



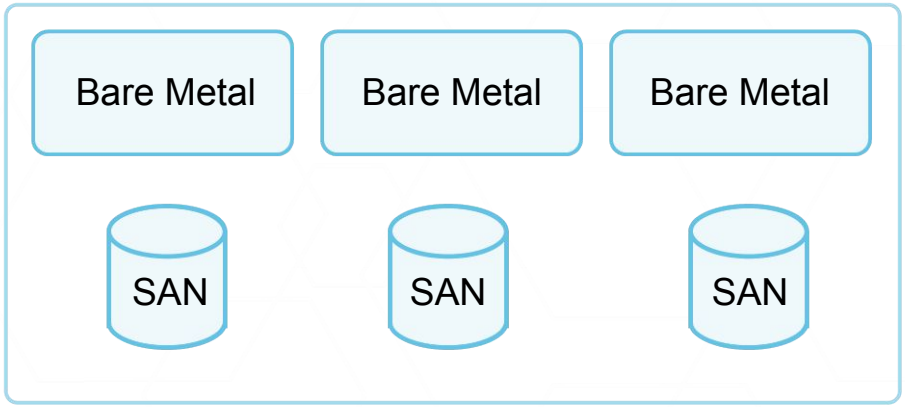
VirtualNet

kubectl

k8s API

kubernetes

v1.5.2



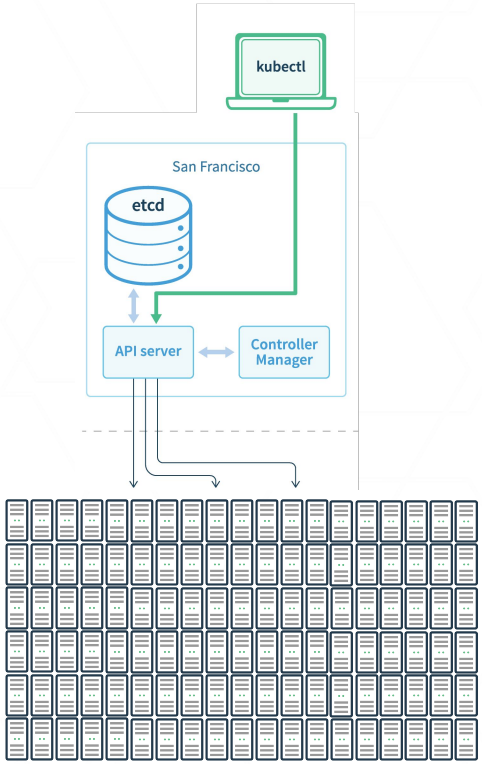
Ethernet

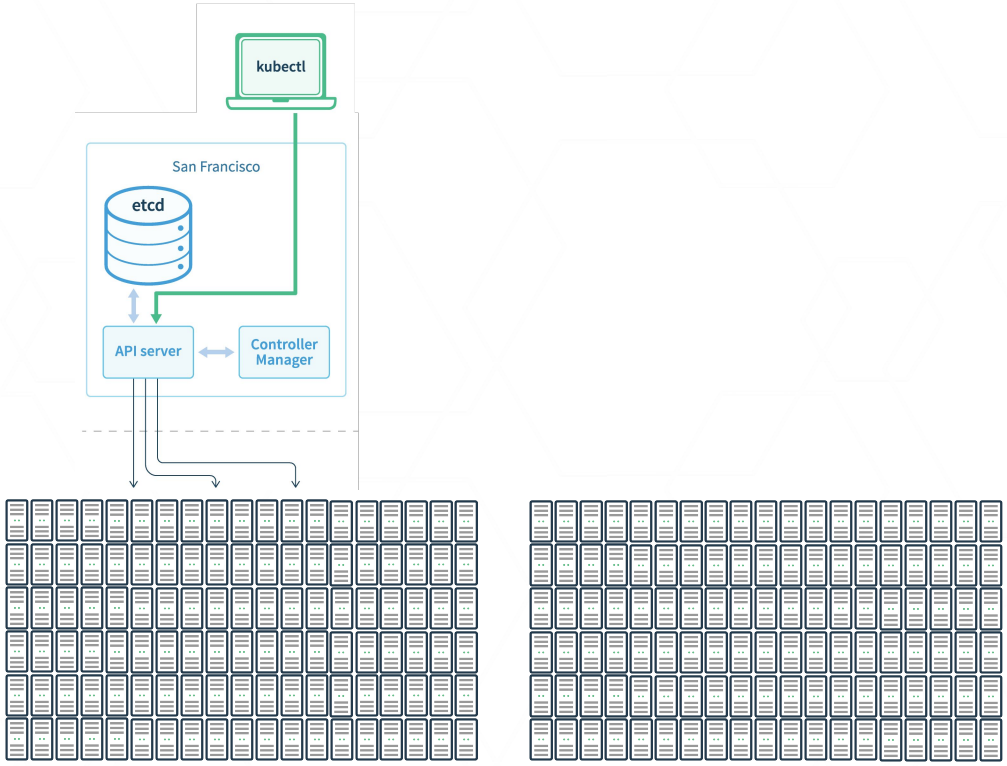
Consistency on all major components

- **Compute:** Physical, Virtual Machine, Cloud
- **Networking:** VXLAN, BGP, IPIP, forwarding, etc
- **Storage:** EBS, NFS, GlusterFS, Cinder, etc
- **Load Balancing:** ELB, nginx, Cloud LB, etc

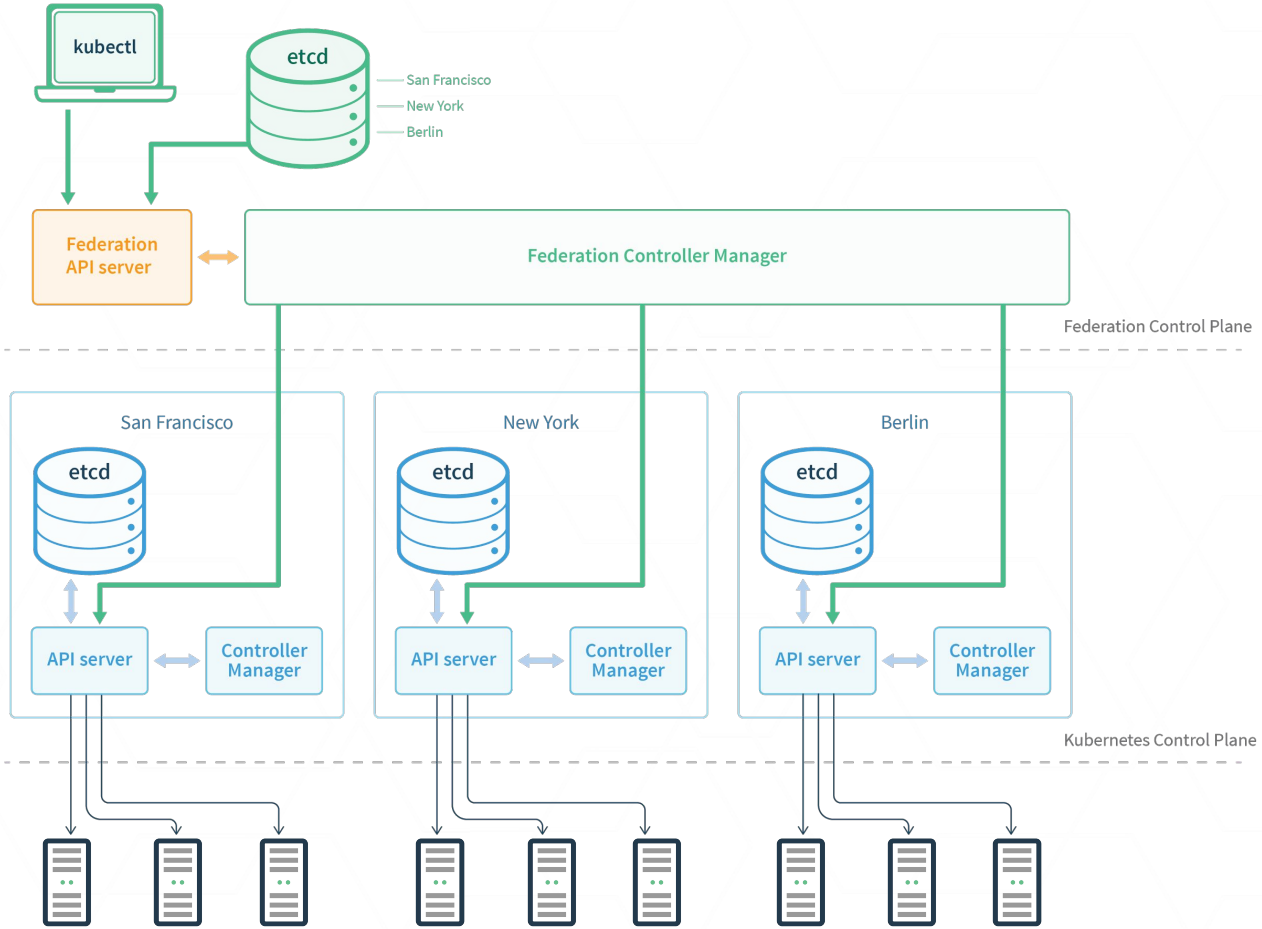
Federation

Handling Too Much Success

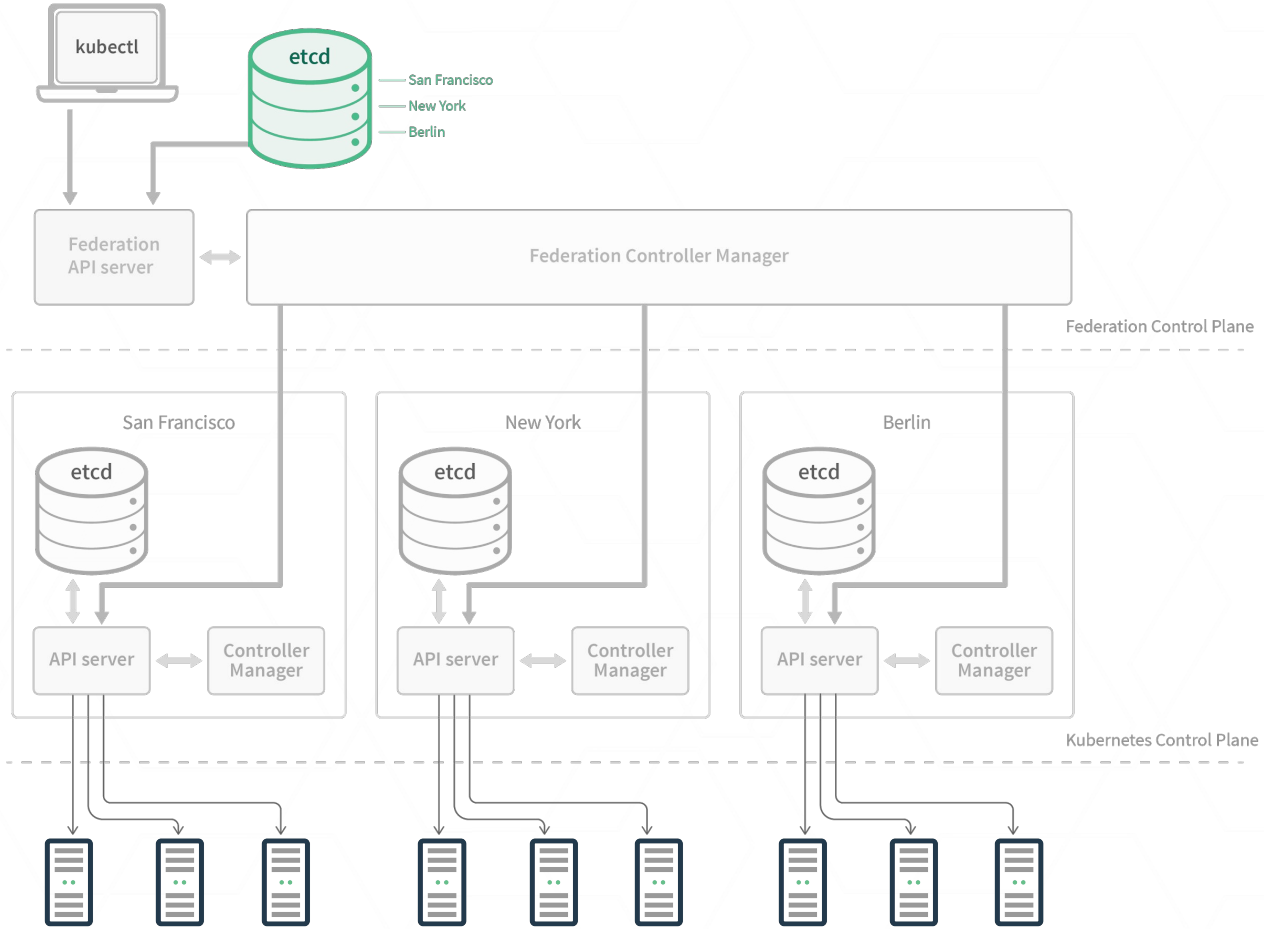

kubectl
San Francisco
etcd
API server
Controller Manager


kubectl
San Francisco
etcd
API server
Controller Manager

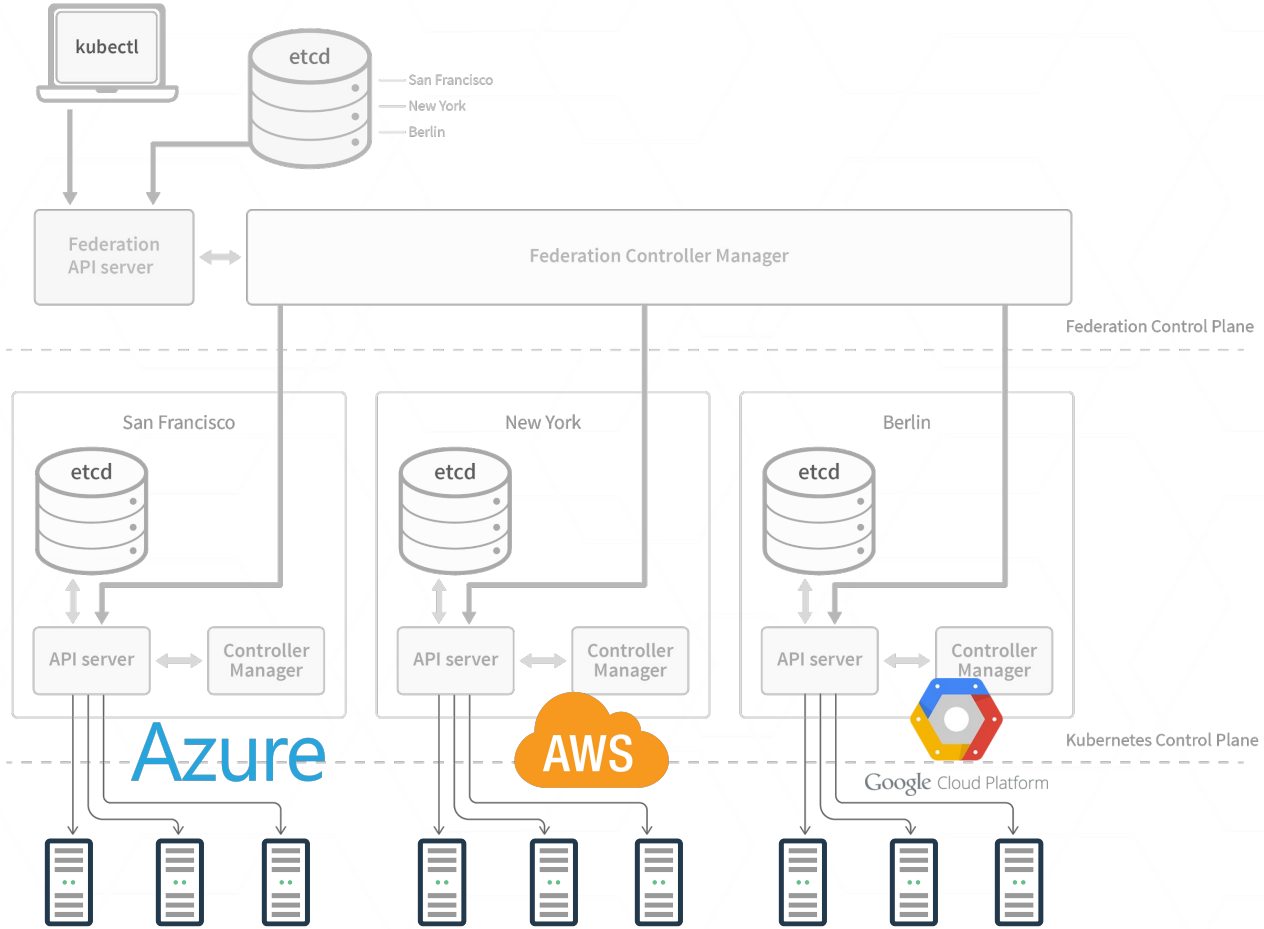

kubectl
San Francisco
etcd
API server
Controller Manager

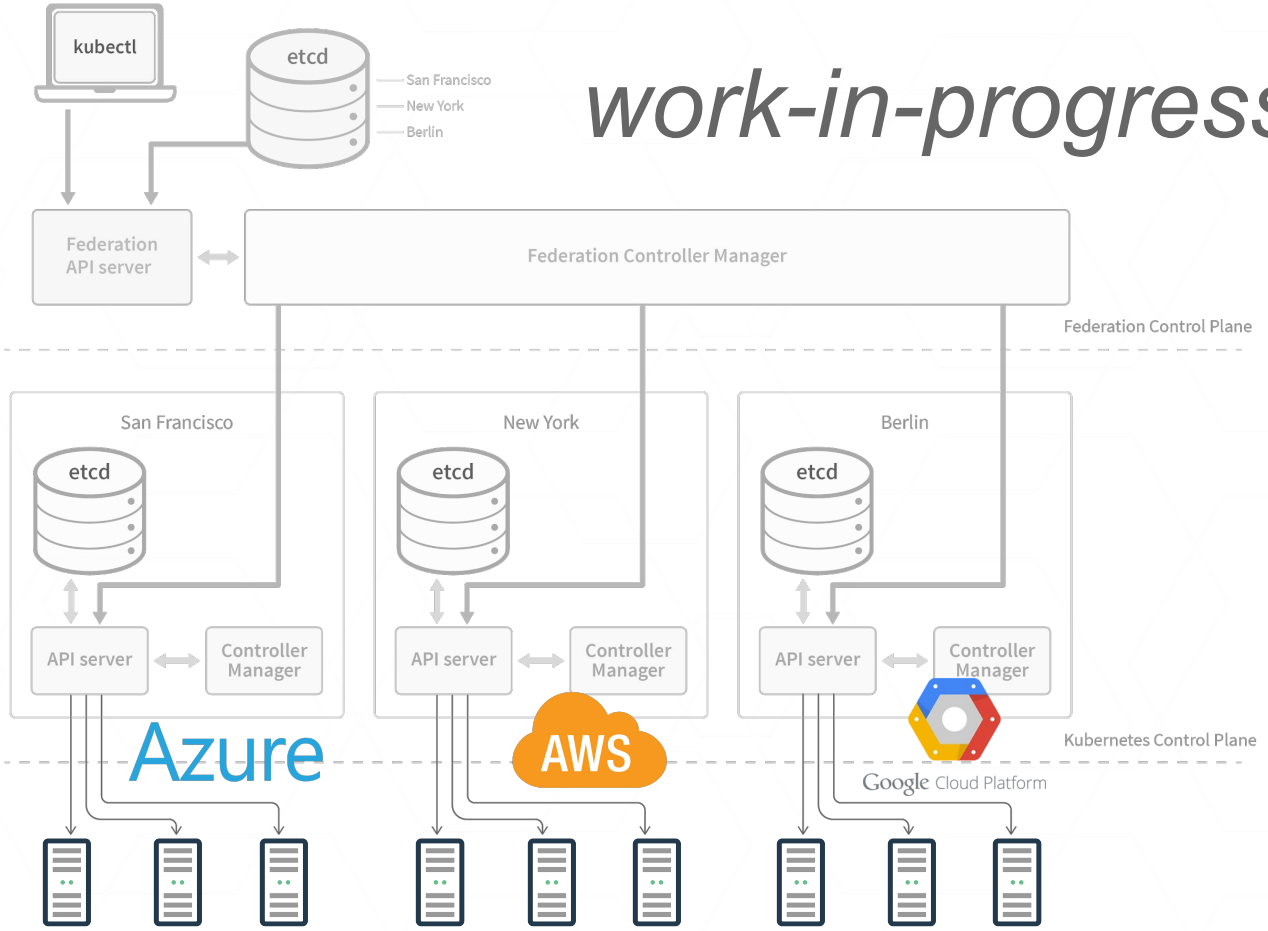

kubectl
etcd
San Francisco
New York
Berlin
Federation API server
Federation Controller Manager
Federation Control Plane
San Francisco
New York
Berlin
etcd
etcd
etcd
API server
Controller Manager
API server
Controller Manager
API server
Controller Manager
Kubernetes Control Plane

CoreOS


kubectl
etcd
San Francisco
New York
Berlin
Federation API server
Federation Controller Manager
Federation Control Plane
San Francisco
New York
Berlin
etcd
etcd
etcd
API server
Controller Manager
API server
Controller Manager
API server
Controller Manager
Kubernetes Control Plane

CoreOS


kubectl
etcd
San Francisco
New York
Berlin
Federation API server
Federation Controller Manager
Federation Control Plane
San Francisco
New York
Berlin
etcd
etcd
etcd
API server
Controller Manager
API server
Controller Manager
API server
Controller Manager
Azure
AWS
Google Cloud Platform
Kubernetes Control Plane


kubectl
etcd
San Francisco
New York
Berlin
work-in-progress
Federation API server
Federation Controller Manager
Federation Control Plane
San Francisco
New York
Berlin
etcd
etcd
etcd
API server
Controller Manager
API server
Controller Manager
API server
Controller Manager
Azure
AWS
Google Cloud Platform
Kubernetes Control Plane

CoreOS

Service Discovery

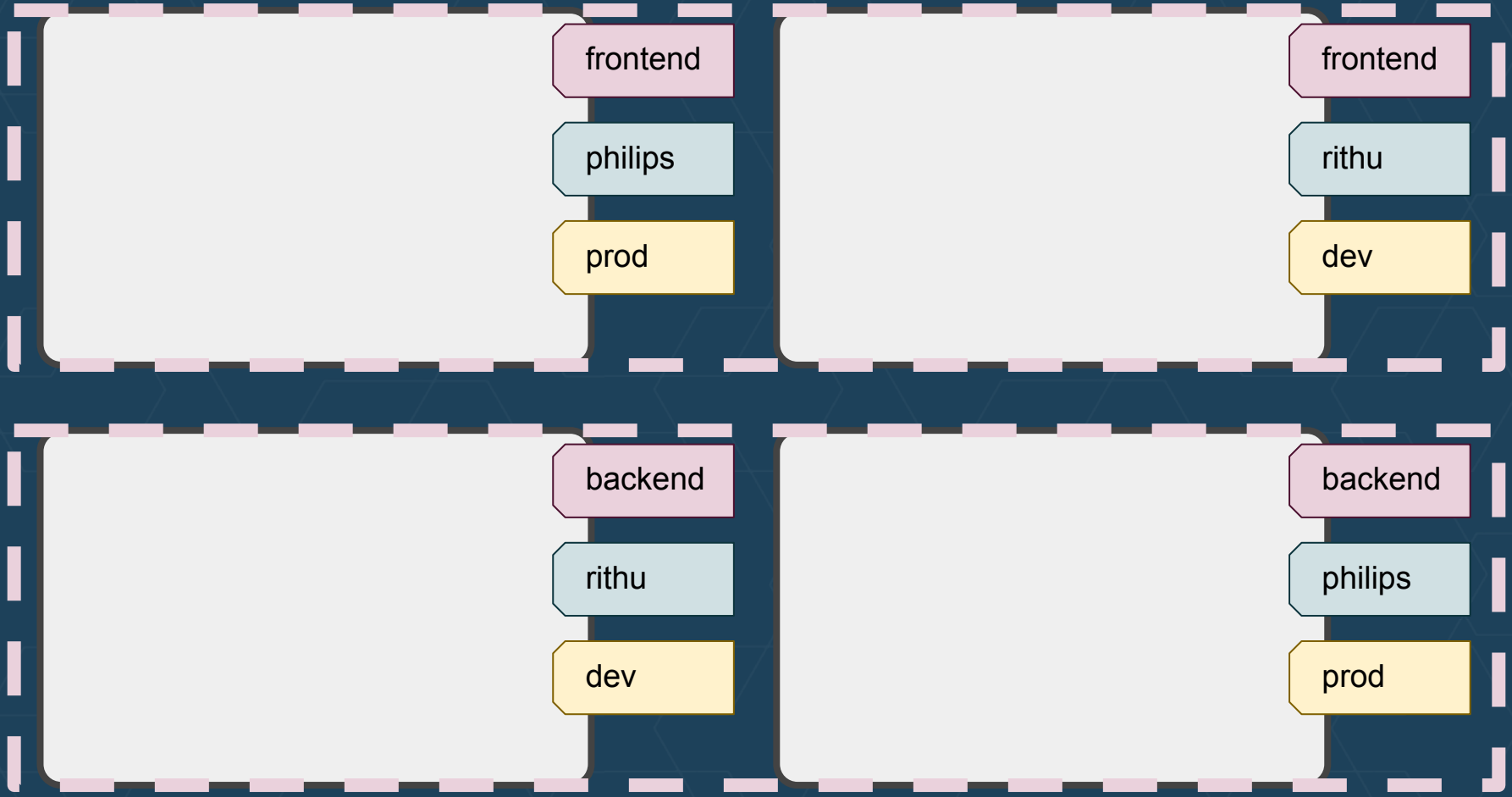
Labels, The Kubernetes Way

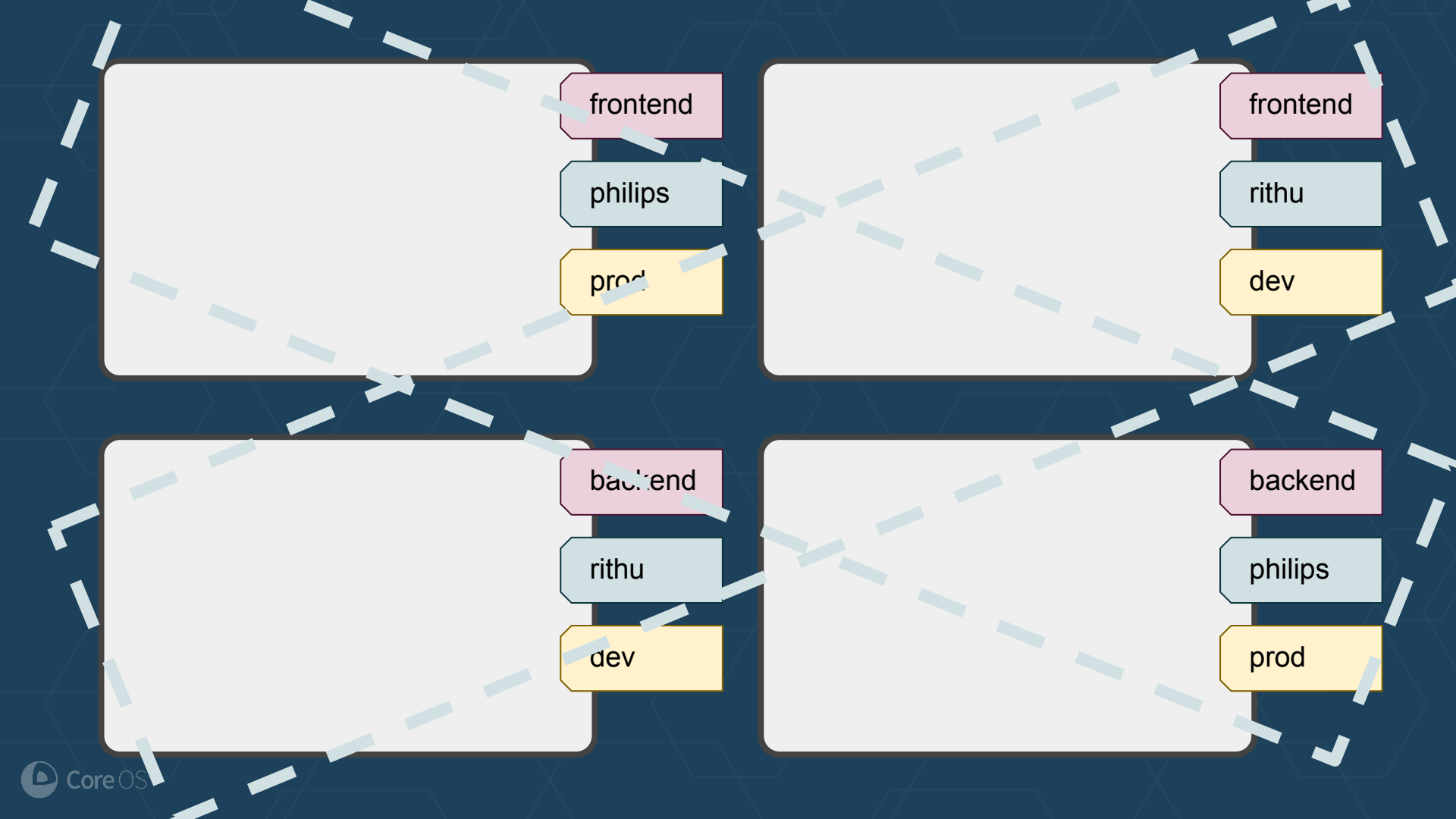

frontend
philips
prod

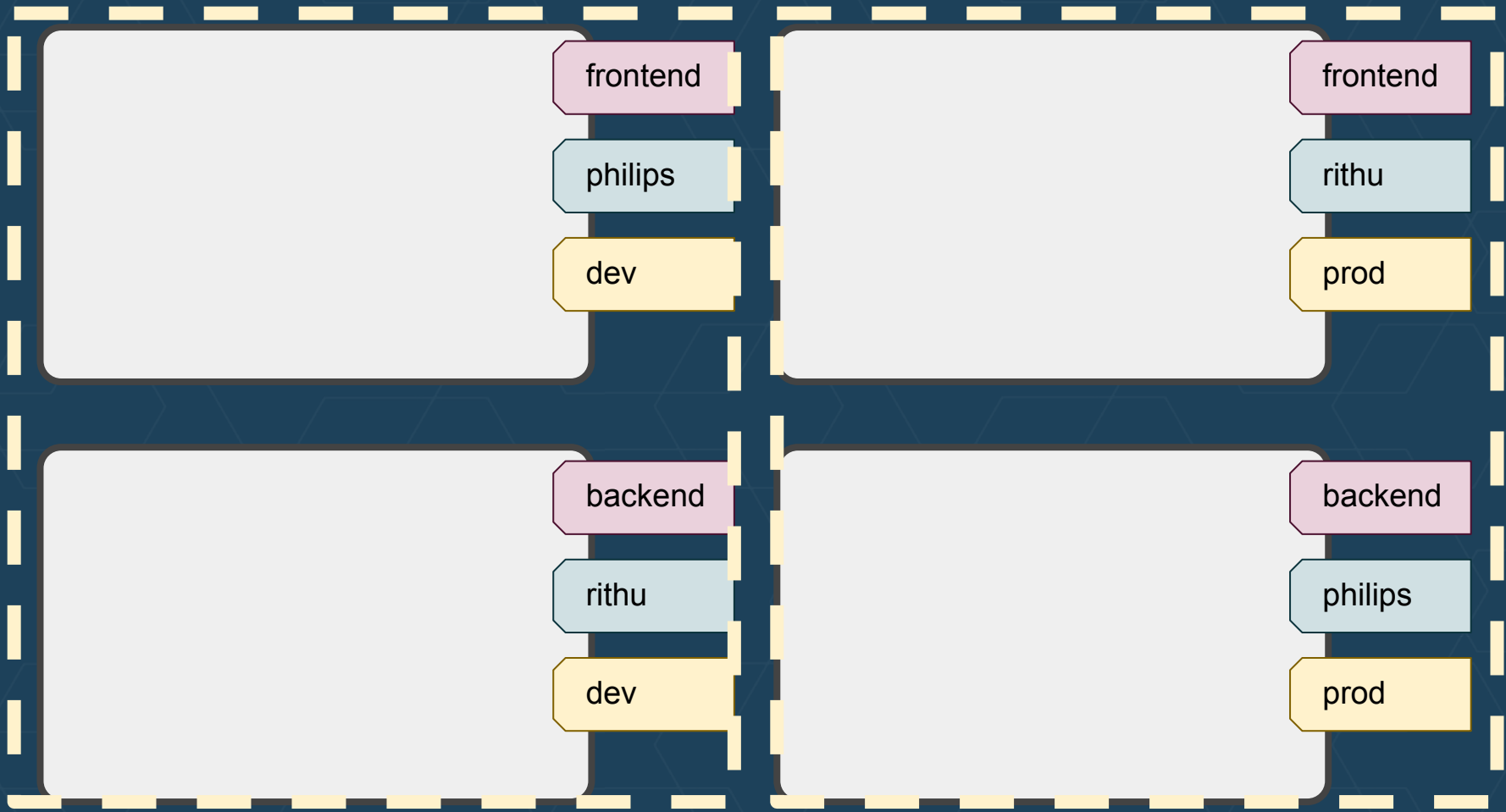

frontend
rithu
dev


backend
rithu
dev


backend
philips
prod


frontend
philips
prod
frontend
rithu
dev
backend
rithu
dev
backend
philips
prod

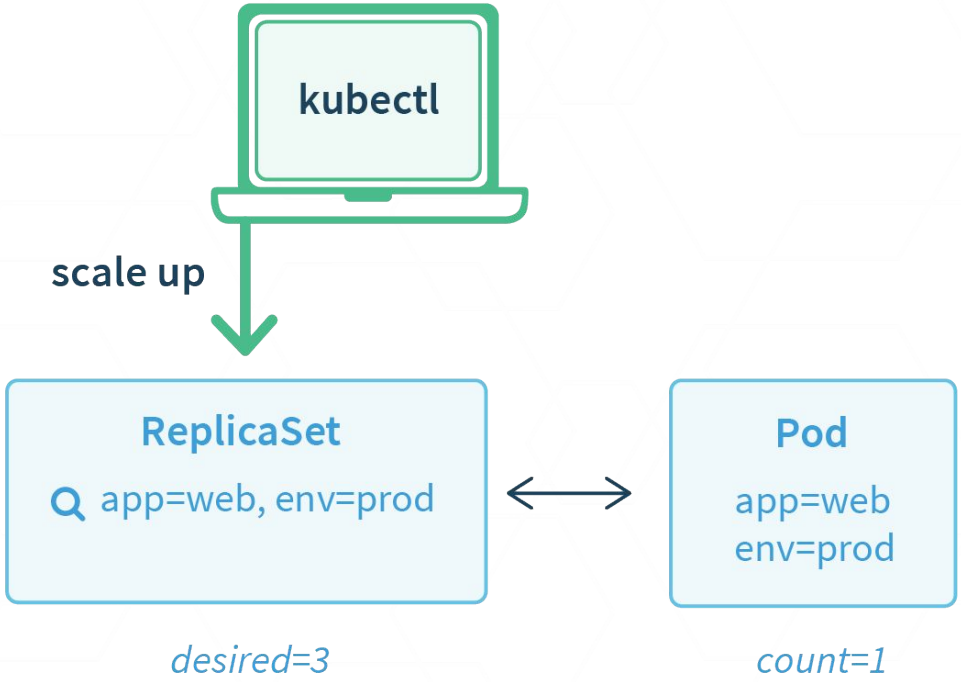

frontend
philips
prod
frontend
rithu
dev
backend
rithu
dev
backend
philips
prod


frontend
philips
dev
frontend
rithu
prod
backend
rithu
dev
backend
philips
prod

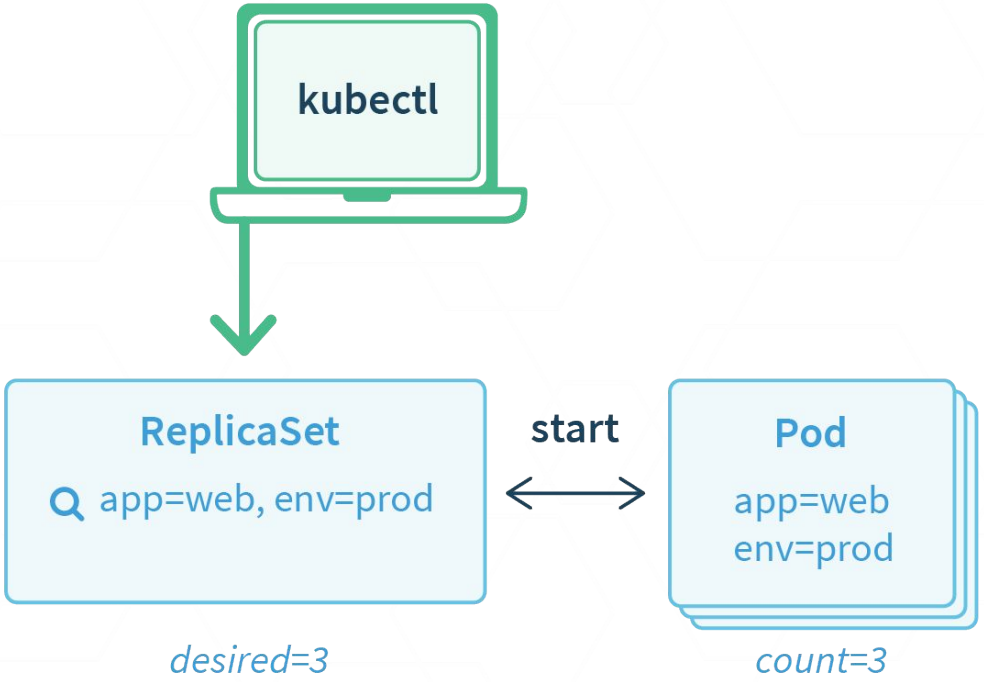
Scaling Applications

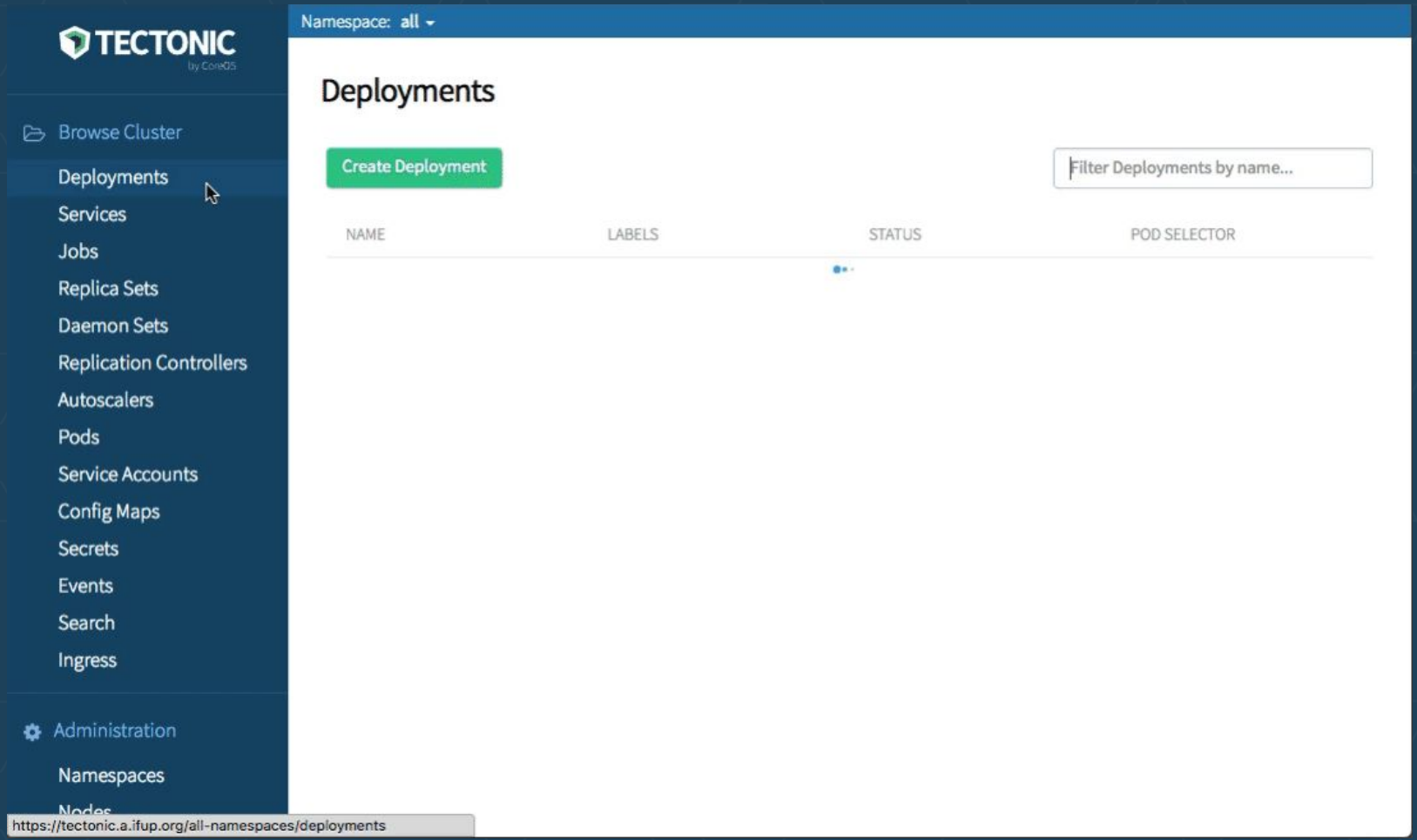
Reacting to Demand

ReplicaSet



ReplicaSet




TECTONIC
by CoreOS
Namespace: all
Browse Cluster
Deployments
Services
Jobs
Replica Sets
Daemon Sets
Replication Controllers
Autoscalers
Pods
Service Accounts
Config Maps
Secrets
Events
Search
Ingress
Administration
Namespaces
Nodes
Deployments
Create Deployment
Filter Deployments by name...
NAME
LABELS
STATUS
POD SELECTOR
https://tectonic.a.ifup.org/all-namespaces/deployments

Overview

Scaling Complex Apps

Creating a Database is Easy on Kubernetes

```
$ kubectl run db --image=quay.io/my/db
```

Managing a Distributed Database is Harder

- **Resize/Upgrade** - coordination for availability
- **Reconfigure** - tedious generation / templating
- **Backup** - requires coordination on instances
- **Healing** - observe and act for recovery

Introducing

Operators

The Dream

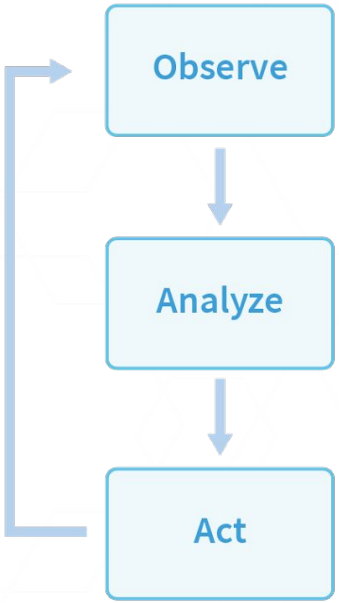
```
$ cat my-db-cluster.yaml
spec:
  clusterSize: 3
  readReplicas: 2
  version: v4.0.1
```

An Operator represents human operational knowledge in software, to reliably manage an application.

etcd Operator Resource

```
$ cat etcd-cluster.yaml
spec:
  clusterSize: 3
  version: v3.1.0
```

etcd Operator



Cluster “A” has 2 running pods:
- name: A-000, version 3.0.9
- name: A-001, version 3.1.0

Differences from desired config:
- should be version 3.1.0
- should have 3 members

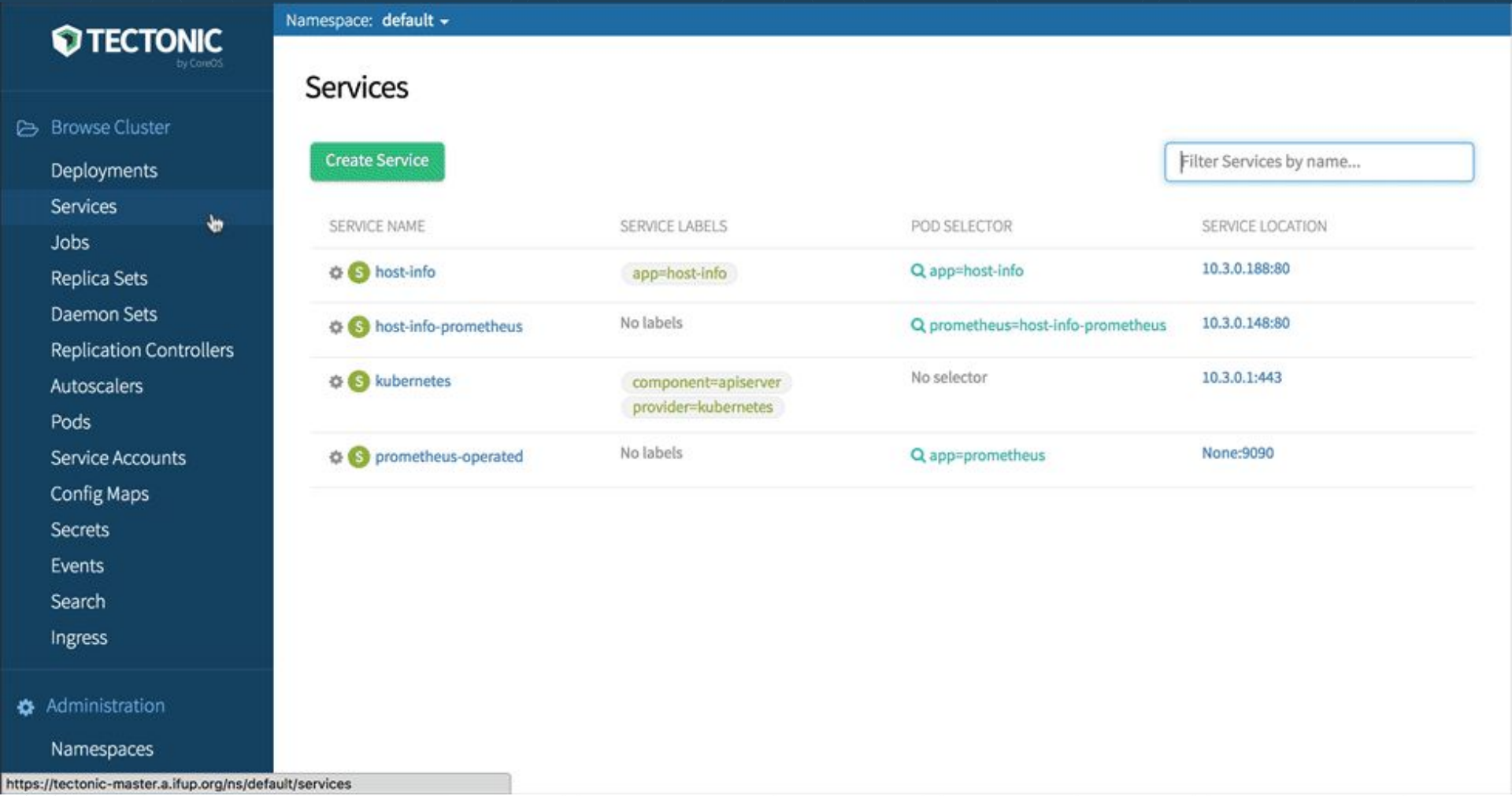
How to get to desired config:
- Recover 1 member
- Back up cluster
- Upgrade to 3.1.0

Monitoring

Foundation of Production

Prometheus Operator

- Operates Prometheus on k8s
- Handles common tasks:
  - Create/Destroy
  - Monitor Configuration
  - **Services Targets via Labels**
- Configured by resources


TECTONIC
by CoreOS
Namespace: default
Browse Cluster
Deployments
Services
Jobs
Replica Sets
Daemon Sets
Replication Controllers
Autoscalers
Pods
Service Accounts
Config Maps
Secrets
Events
Search
Ingress
Administration
Namespaces
Services
Create Service
Filter Services by name...
SERVICE NAME
SERVICE LABELS
POD SELECTOR
SERVICE LOCATION
host-info
app=host-info
app=host-info
10.3.0.188:80
host-info-prometheus
No labels
prometheus=host-info-prometheus
10.3.0.148:80
kubernetes
component=apiserver
provider=kubernetes
No selector
10.3.0.1:443
prometheus-operated
No labels
app=prometheus
None:9090
https://tectonic-master.a.ifup.org/ns/default/services

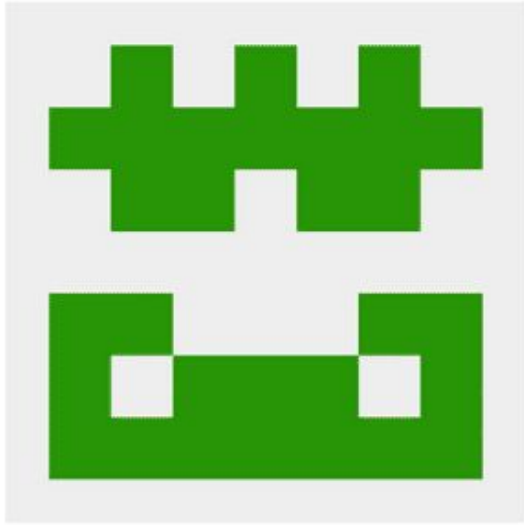
host.ifup.org

Prometheus Time Series Colle

CoreOS

host.ifup.org

You are visitor: 3919



host-info-2478745576-m6pff

Unknown Zone

Networking

lo

- 127.0.0.1/8
- ::1/128

eth0

CoreOS

Self Driving Infrastructure

Self-Driving Infrastructure

- *Container Linux* manages the OS
- *Operators* manage everything above it

Realising the dream: fully automated updates for the whole software stack!

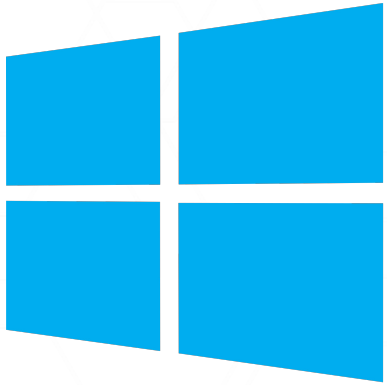
CoreOS Mission

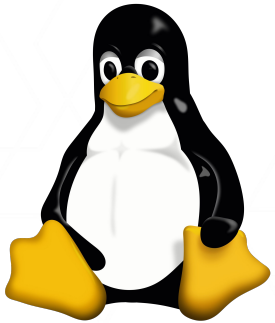
Secure the Internet

Free Stuff

Get Kubernetes For Yourself

Minikube - Kubernetes on your laptop





macOS

All-in-one Virtual Machine

Single file download

Latest Kubernetes

github.com/kubernetes/minikube

Tectonic Free Tier

Pure upstream Kubernetes

Production configuration

Bare metal and AWS installer

(more coming soon)

coreos.com/tectonic

Join Us

Build great code

github.com/coreos

Hack on Operating Systems

Build distributed systems

Geek out on internet identity protocols

Container Focused Operating System

Clustered Database

Identity & Federation

10+ other major projects

github.com/kubernetes

Most active project on GitHub

Dozens of special interest groups

Not just code! Docs, PM, and more

kubernetes

Work With Us

Community is taking off

kubernetes

What container and PaaS tools are used to manage OpenStack applications?

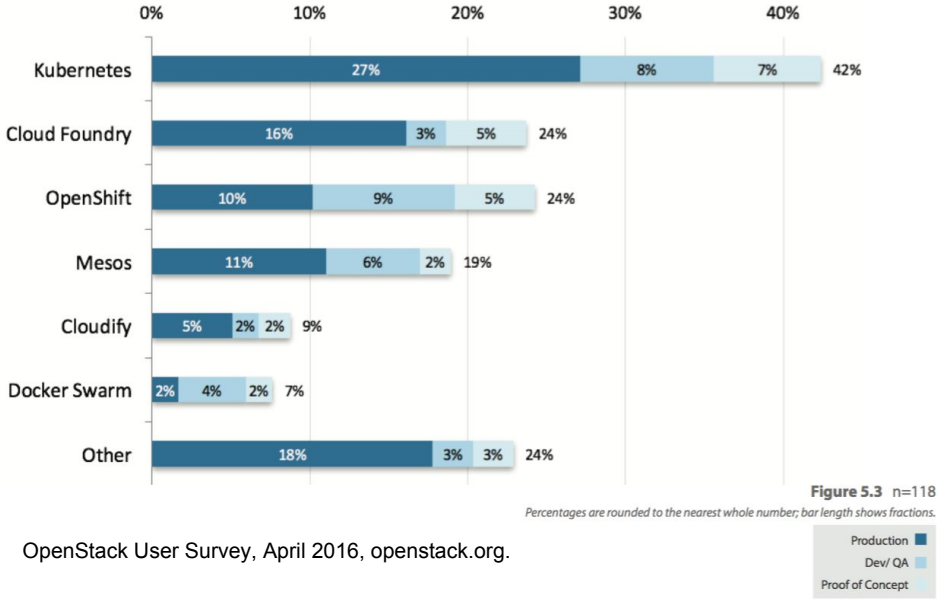
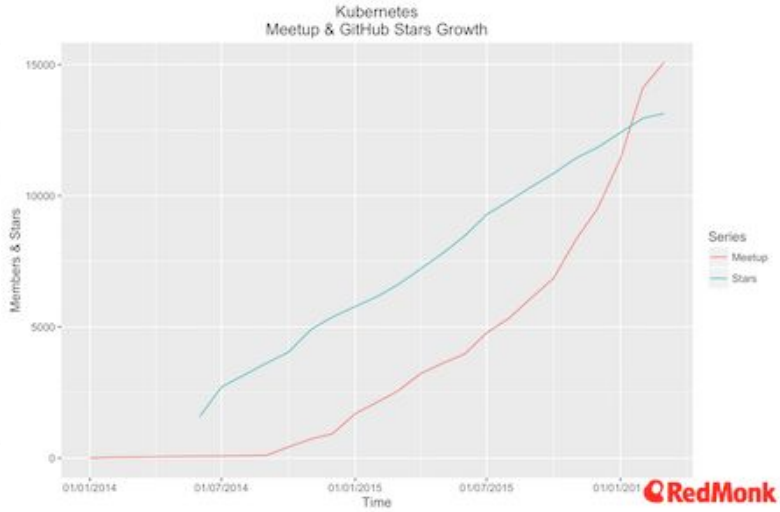


Figure 5.3 n=118

Percentages are rounded to the nearest whole number; bar length shows fractions.

OpenStack User Survey, April 2016, openstack.org.

Developer engagement



"The Further Evolution of Kubernetes," March 2016.

CoreOS

CoreOS is running the world's containers

We're hiring: careers@coreos.com

OPEN SOURCE	ENTERPRISE
90+ Projects on GitHub, 1,000+ Contributors	Support plans, training and more
container linux dex etcd	TECTONIC QUAY
coreos.com	sales@coreos.com

Thanks!

QUESTIONS?

jonathan.boulle@coreos.com

@baronboulle

linkedin.com/in/jonathanboulle

LONGER CHAT?

Let’s talk!

IRC

More events: coreos.com/community

We’re hiring: coreos.com/careers