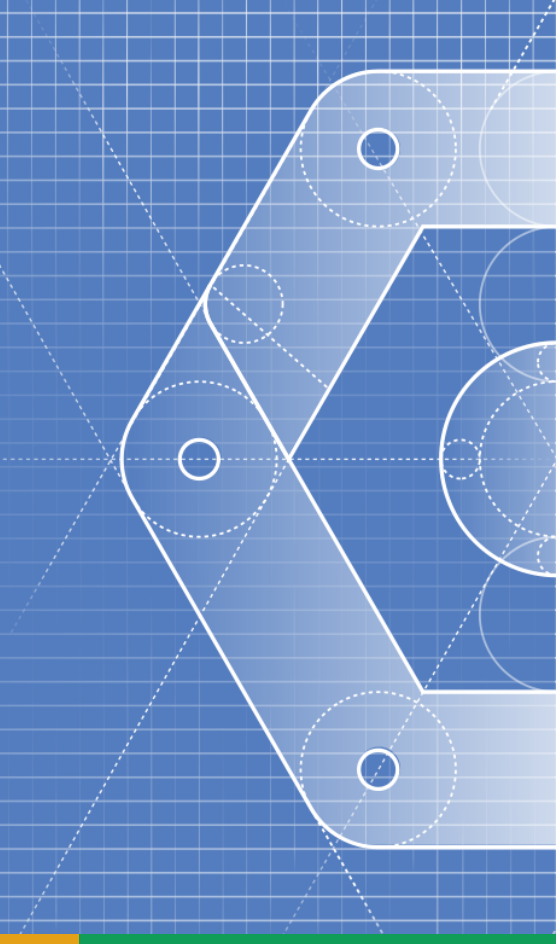
Kubernetes

with Java-based Microservices

How To w/ @saturnism



Ray Tsang

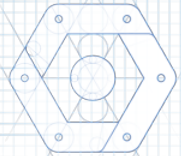
Developer Advocate

@saturnism | +RayTsang

Ray Tsang

Developer
Architect
Traveler
Photographer

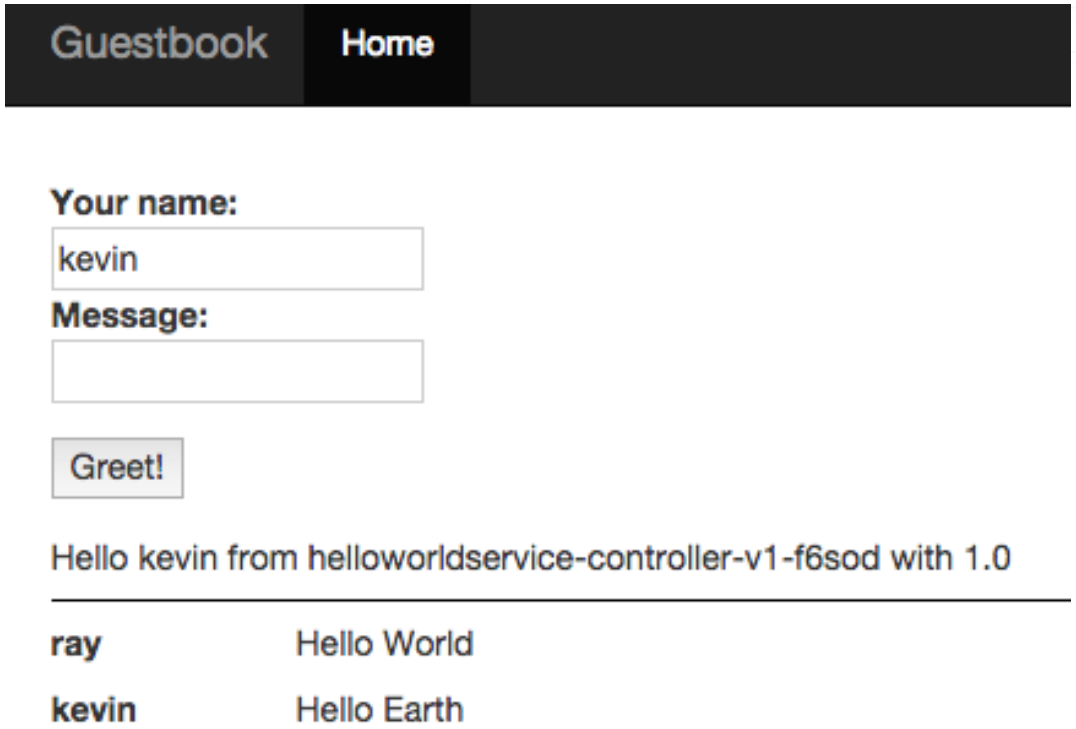
flickr.com/saturnism

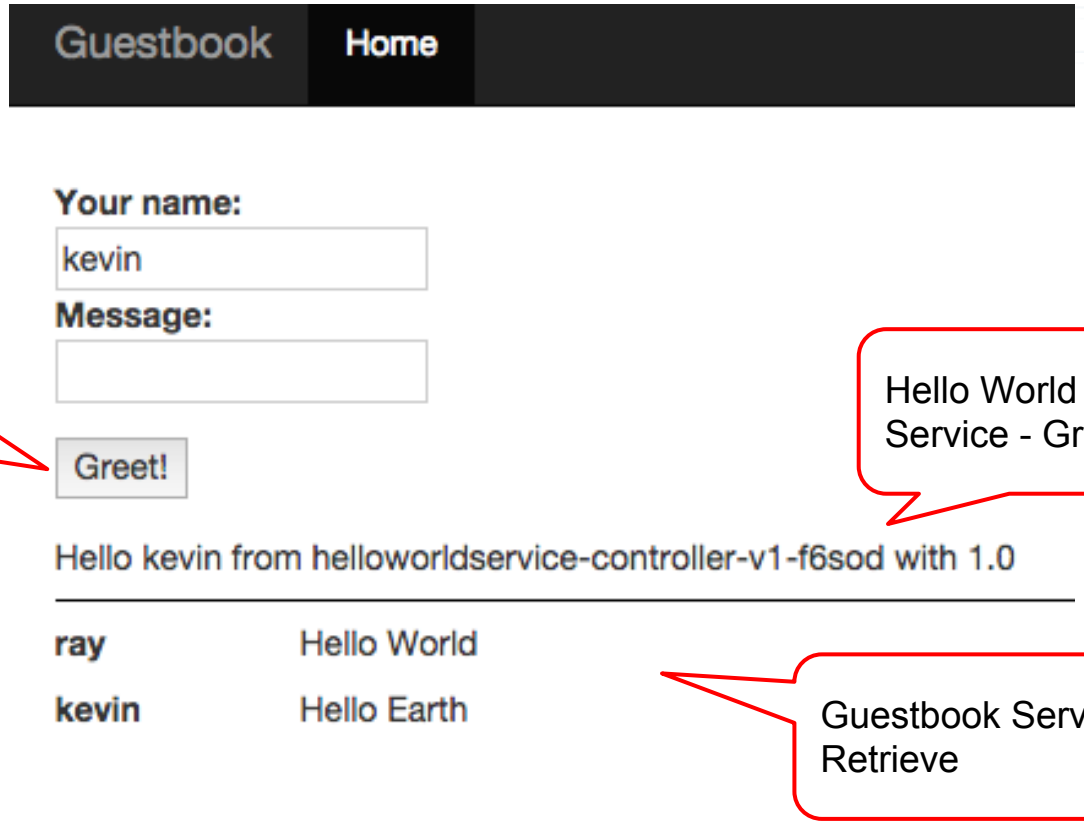


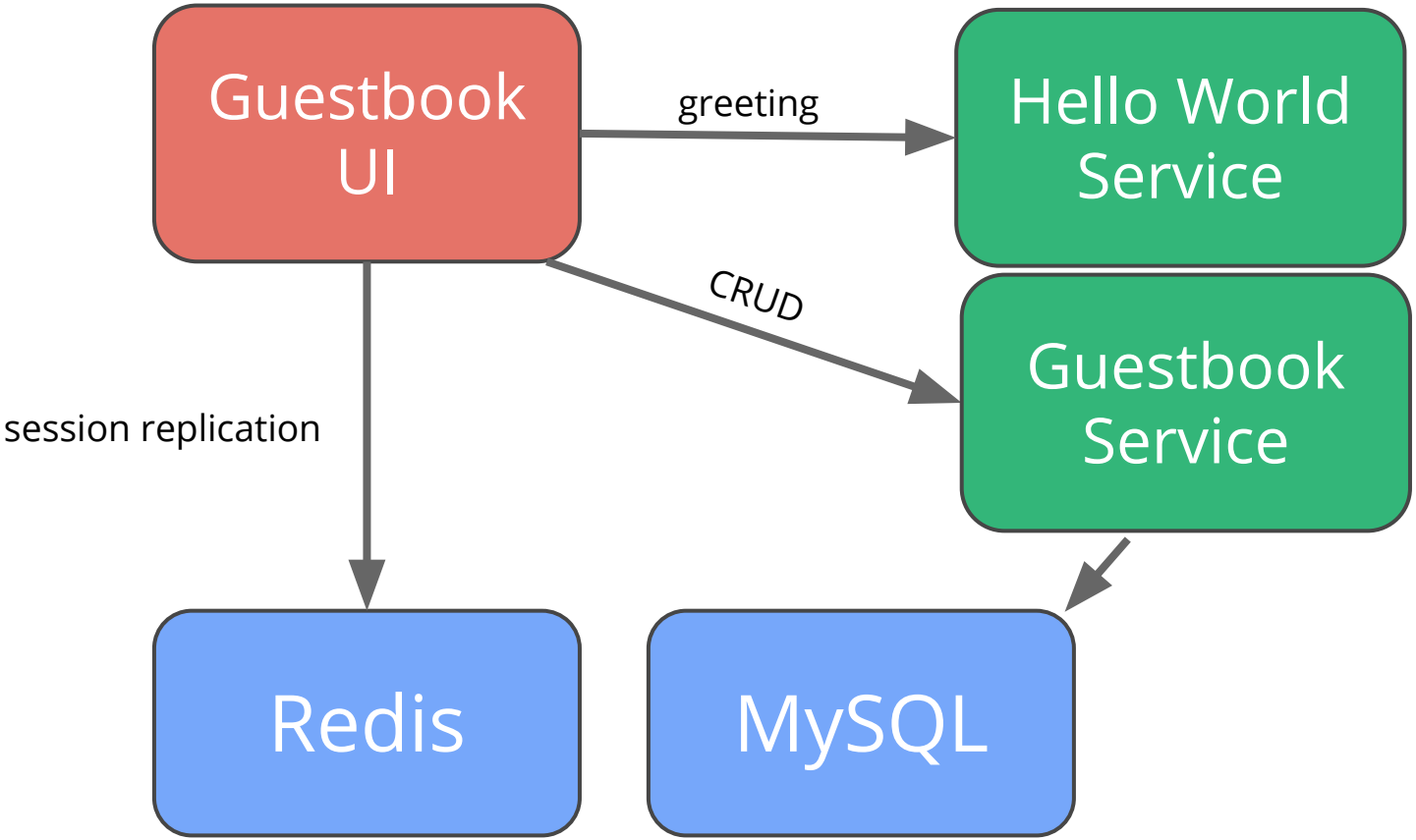
Microservices?

You probably heard a lot already!

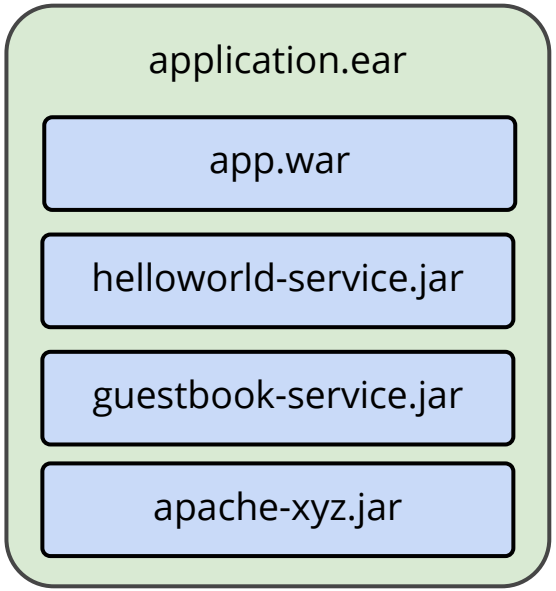
No theories here - just a how to

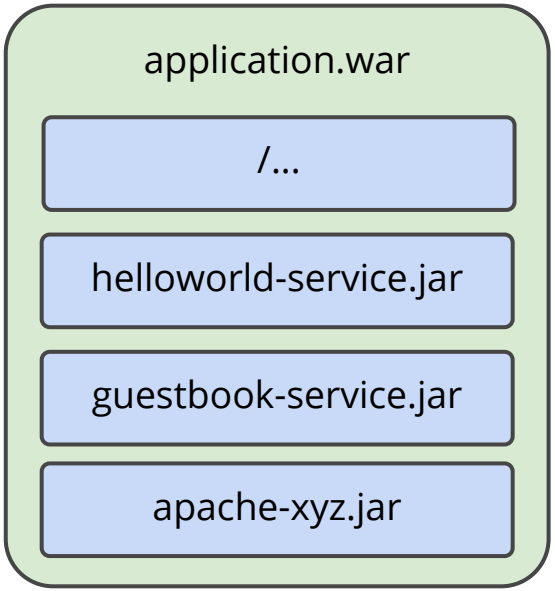

Guestbook
Home
Your name:
kevin
Message:
Greet!
Hello kevin from helloworldservice-controller-v1-f6sod with 1.0
ray
Hello World
kevin
Hello Earth


Guestbook
Home
Your name:
kevin
Message:
Greet!
Hello kevin from helloworldservice-controller-v1-f6sod with 1.0
ray
Hello World
kevin
Hello Earth
Guestbook Service - Create
Hello World Service - Greet
Guestbook Service - Retrieve


Guestbook UI
greeting
Hello World Service
CRUD
Guestbook Service
session replication
Redis
MySQL

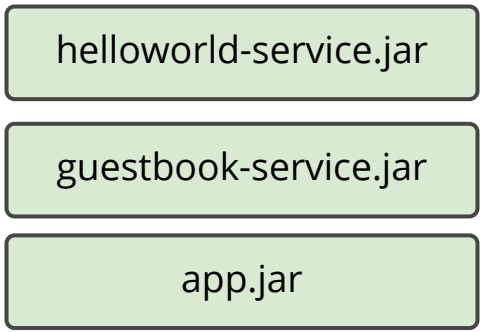
Package & Deployment

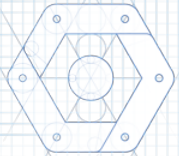




Microservices Way?

Package & Deployment





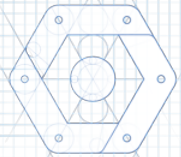
Deployment? Just run it!

java -jar helloworld-service.jar

java -jar guestbook-service.jar

java -jar app.jar

Let’s see some code

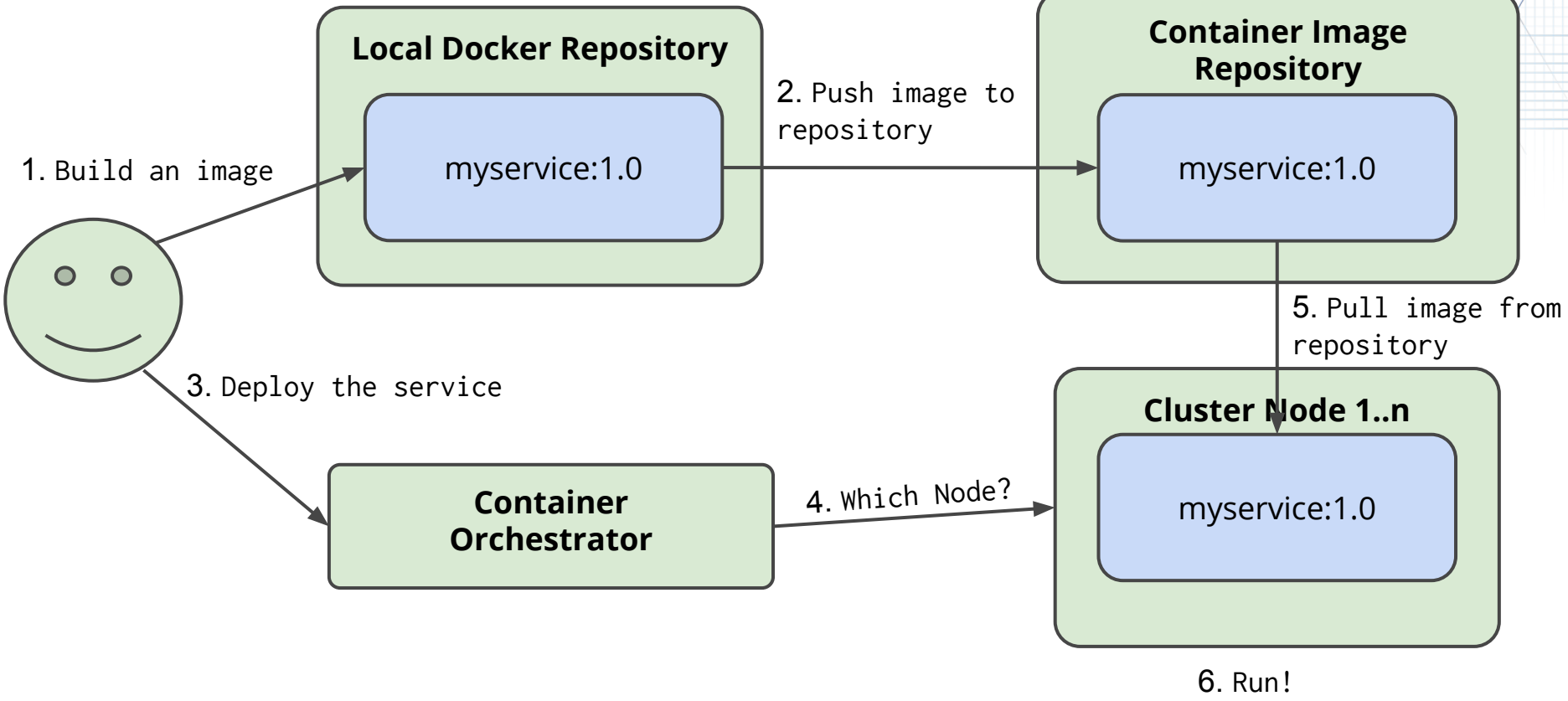


So many services

Deploy, Manage, Ports, Discovery, Isolation... How?

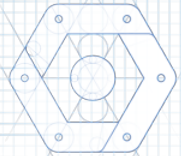
Containers & Orchestration

To the Rescue!


Local Docker Repository
myservice:1.0
Container Image Repository
myservice:1.0
1. Build an image
2. Push image to repository
3. Deploy the service
5. Pull image from repository
Container Orchestrator
4. Which Node?
Cluster Node 1..n
myservice:1.0
6. Run!

Containerize Option #1

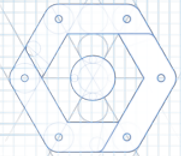
`Dockerfile`



Containerize Option #2

spotify/docker-maven-plugin

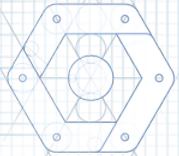
```
mvn docker:build
```



Containerize Option #3

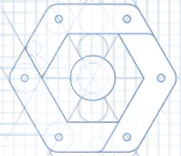
Docker Hub / GitHub

saturnism/spring-boot



Let’s run the container!

```
docker run -ti -p 8080:8080 helloworld-service
```



MySQL

```
docker run -d --name mysql -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=yourpassword -e MYSQL_DATABASE=app mysql
```

Redis

```
docker run -d --name redis redis
```

Guestbook Service

```
docker run -ti --name guestbookservice --link mysql:mysql
saturnism/guestbook-service
```

Hello World Service

```
docker run -ti --name helloworldservice \
saturnism/spring-boot-helloworld-service:1.0
```

Guestbook UI

```
docker run -ti --rm --link redis:redis \
--link helloworldservice:helloworldservice \
--link guestbookservice:guestbookservice \
-p 8080:8080 saturnism/spring-boot-helloworld-ui
```

Docker Compose

```
docker-compose up
```

Everything at Google runs in containers

Shipping Containers At Clyde, by Steve Gibson

Everything at Google runs in containers

Launch over **2 billion** containers **per week**.

Shipping Containers At Clyde, by Steve Gibson

Enter Kubernetes

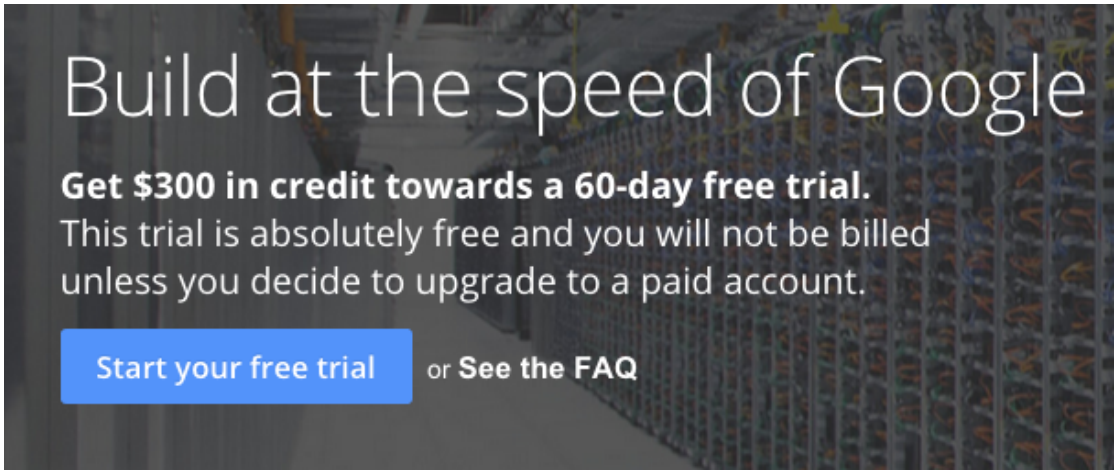
Greek for *“Helmsman”*; also the root of the word *“Governor”*

- Container **orchestrator**
- Runs containers
- Supports **multiple cloud** and **bare-metal** environments
- Inspired and informed by Google’s experiences and internal systems
- **Open source**, written in **Go**

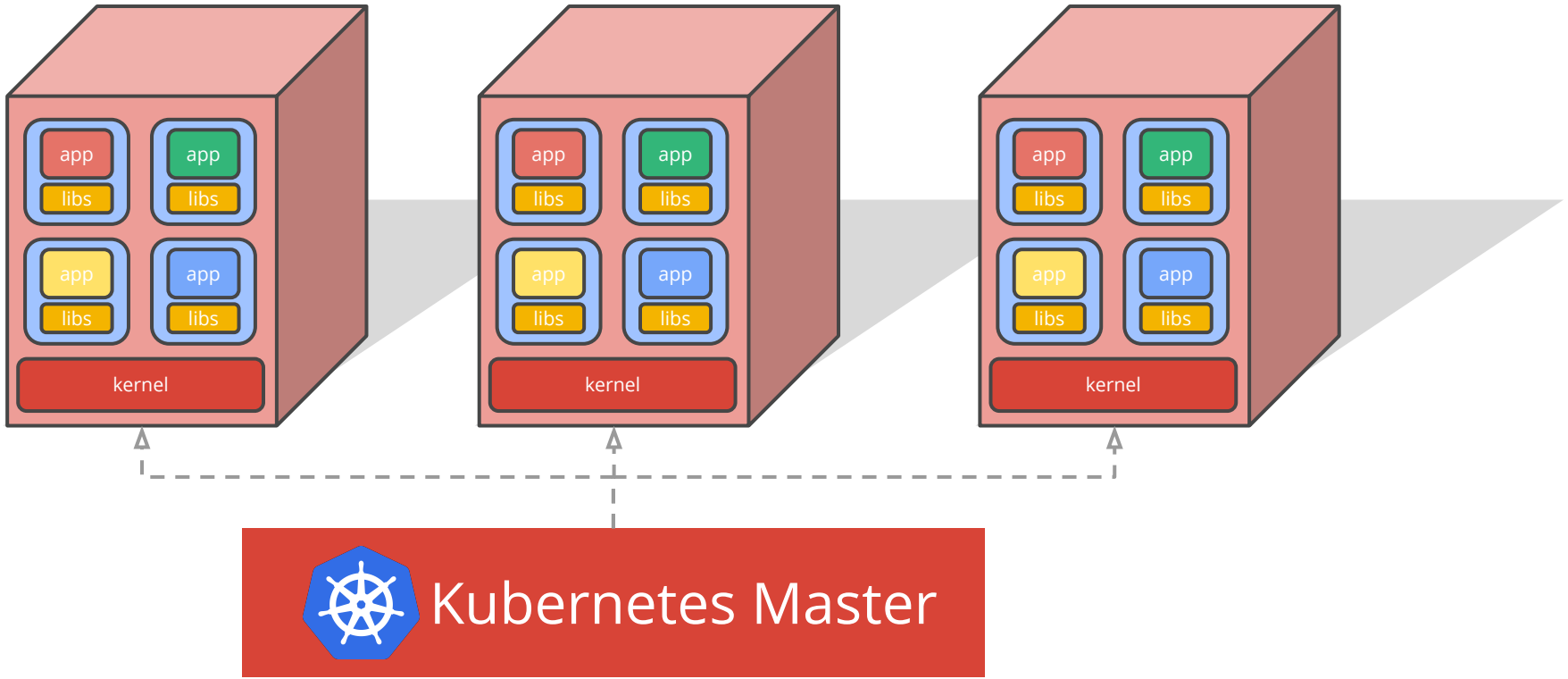
Manage applications, not machines

Try out Google Container Engine

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



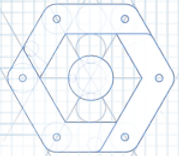
Kubernetes Cluster



Distributed process scheduler

cluster of machines as one!

Open Source Community



Project

Version 1.0

Hosted on GitHub

410+ contributors

14,800+ commits

9,200+ GitHub stars

Partners

CoreOS

HP

IBM

Mesosphere

Microsoft

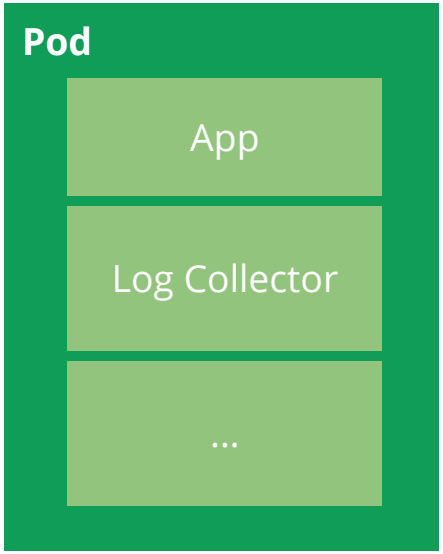
Pivotal

Red Hat

SaltStack

VMWare

Pods



Group of containers

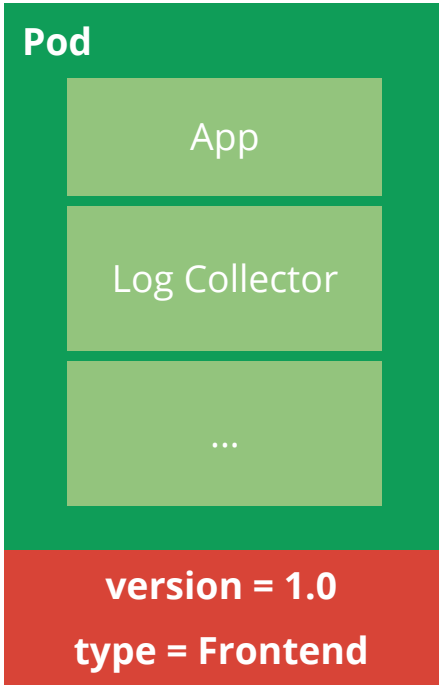
Live and die together

Shared network interface

Shared volumes

Unique Routable IP

Labels

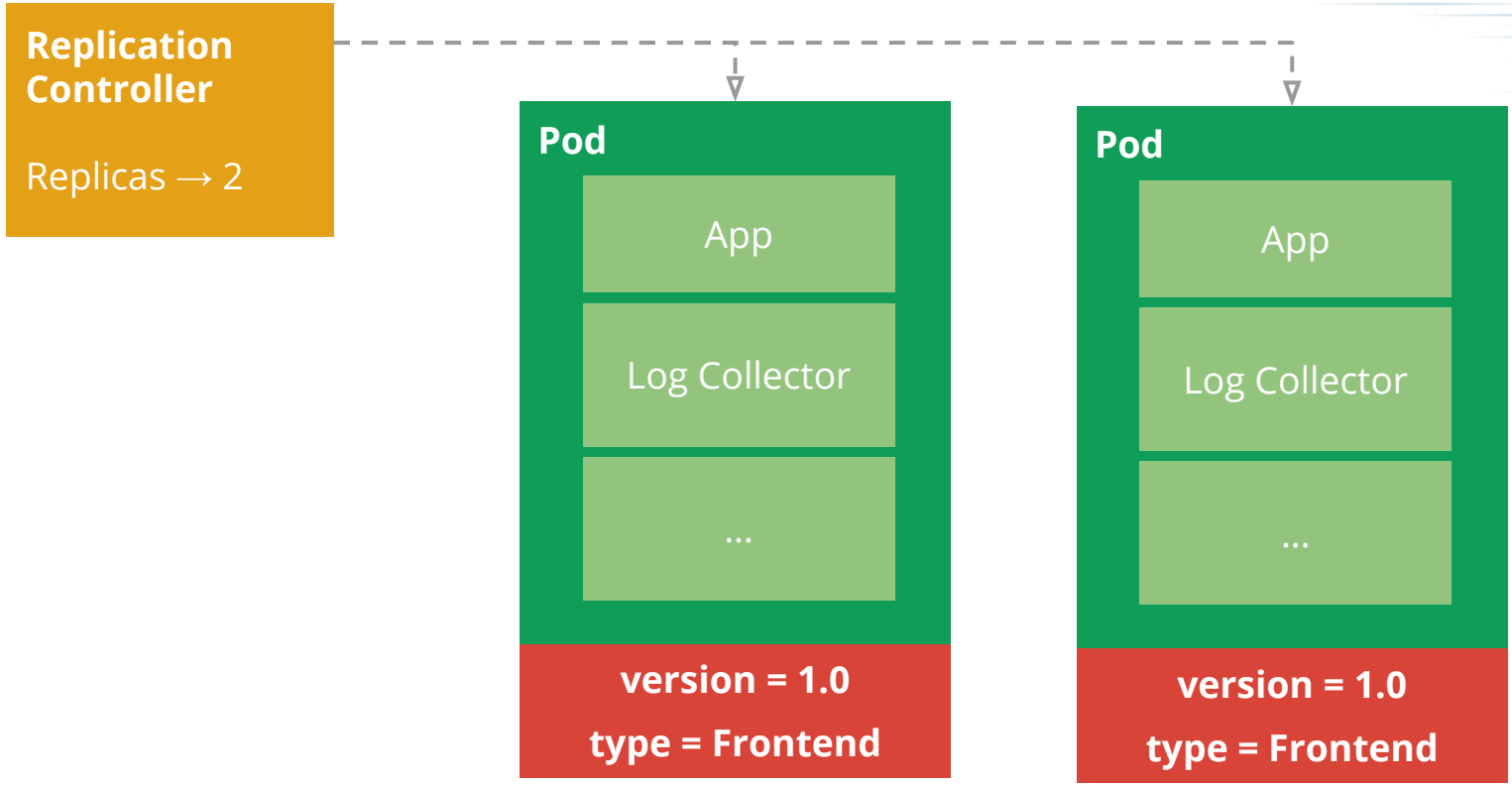


Label anything

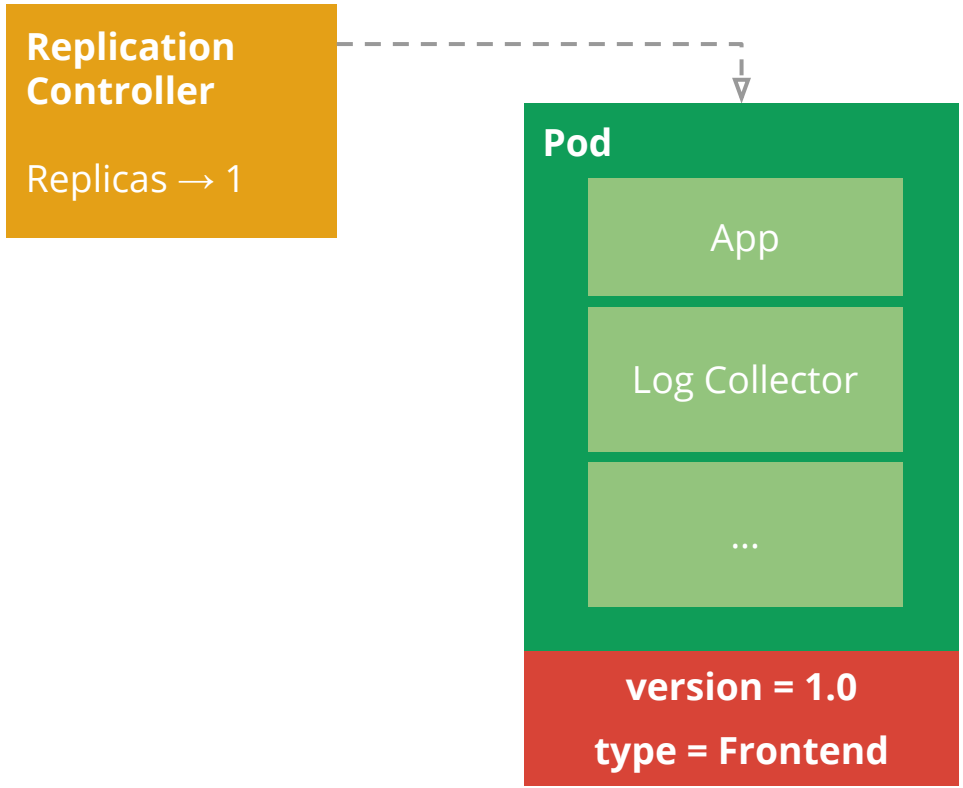
Name-value pair

Make your own

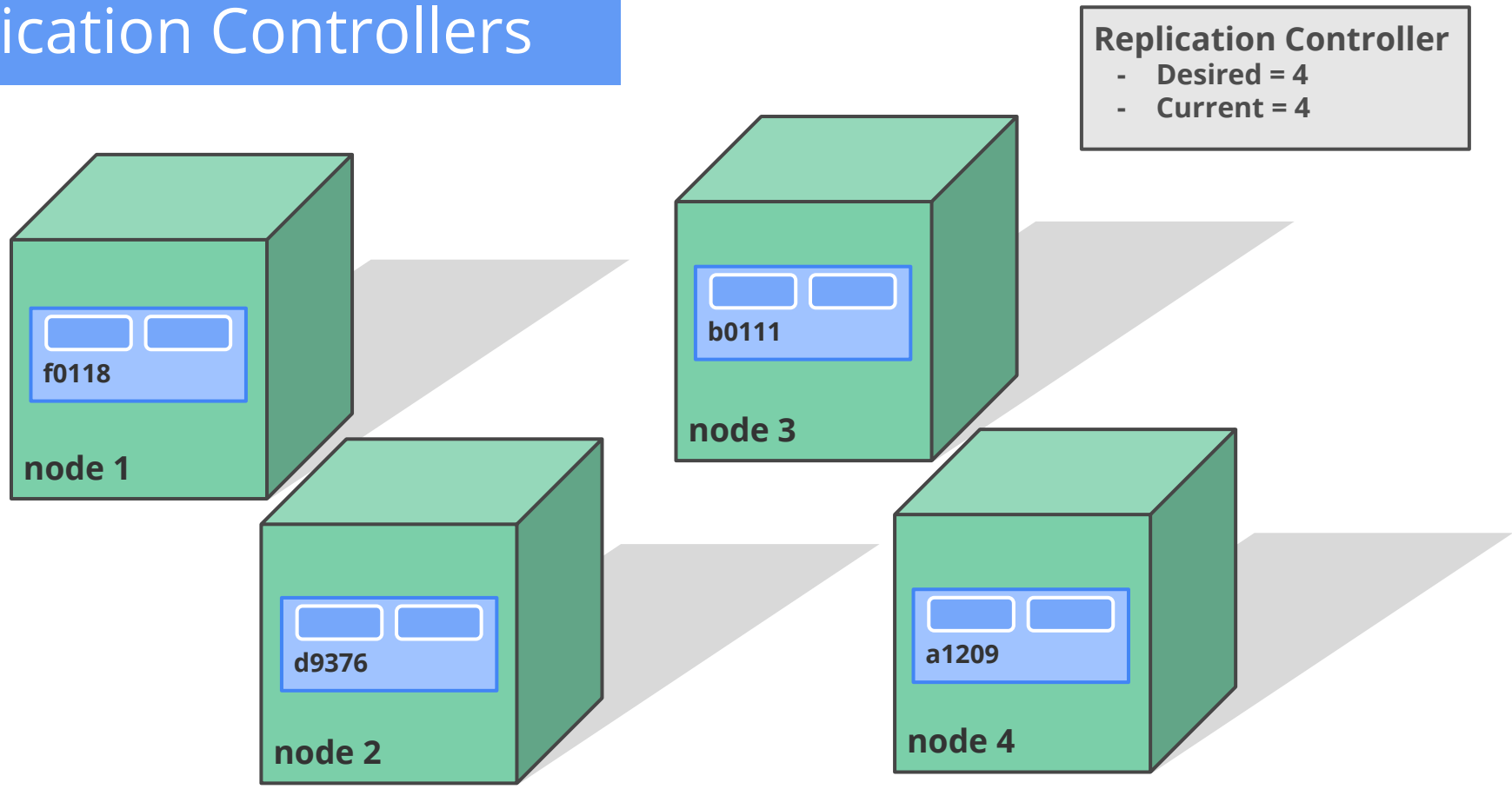
Replication Controllers



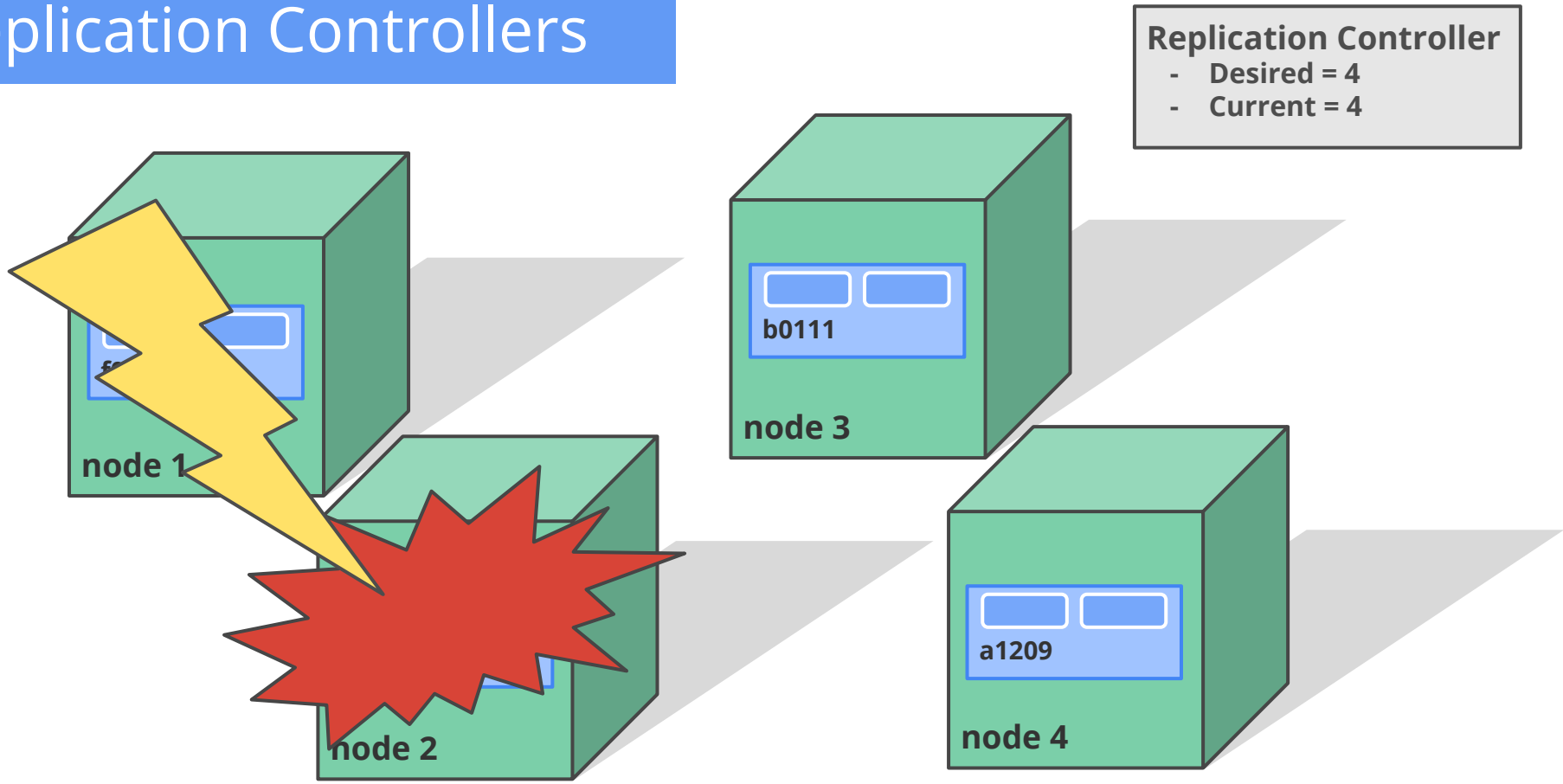
Replication Controllers



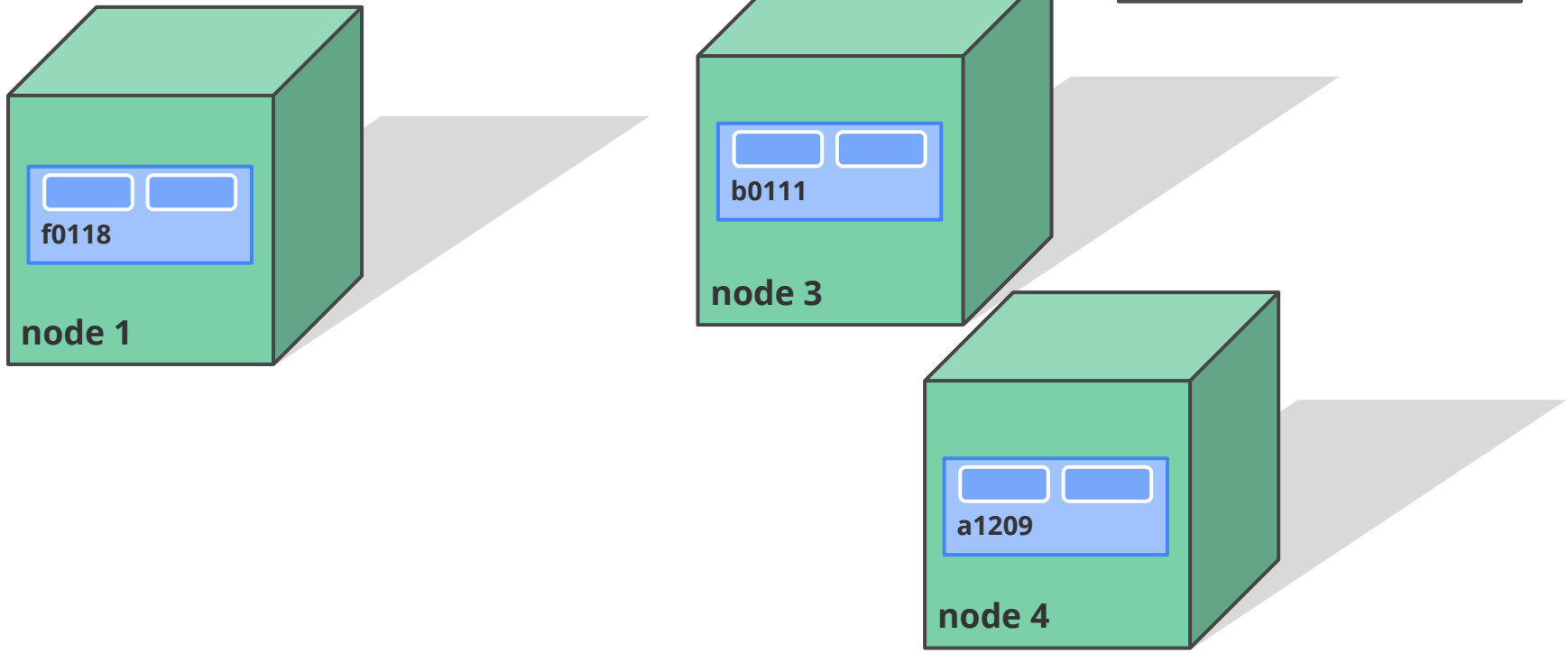
Replication Controllers



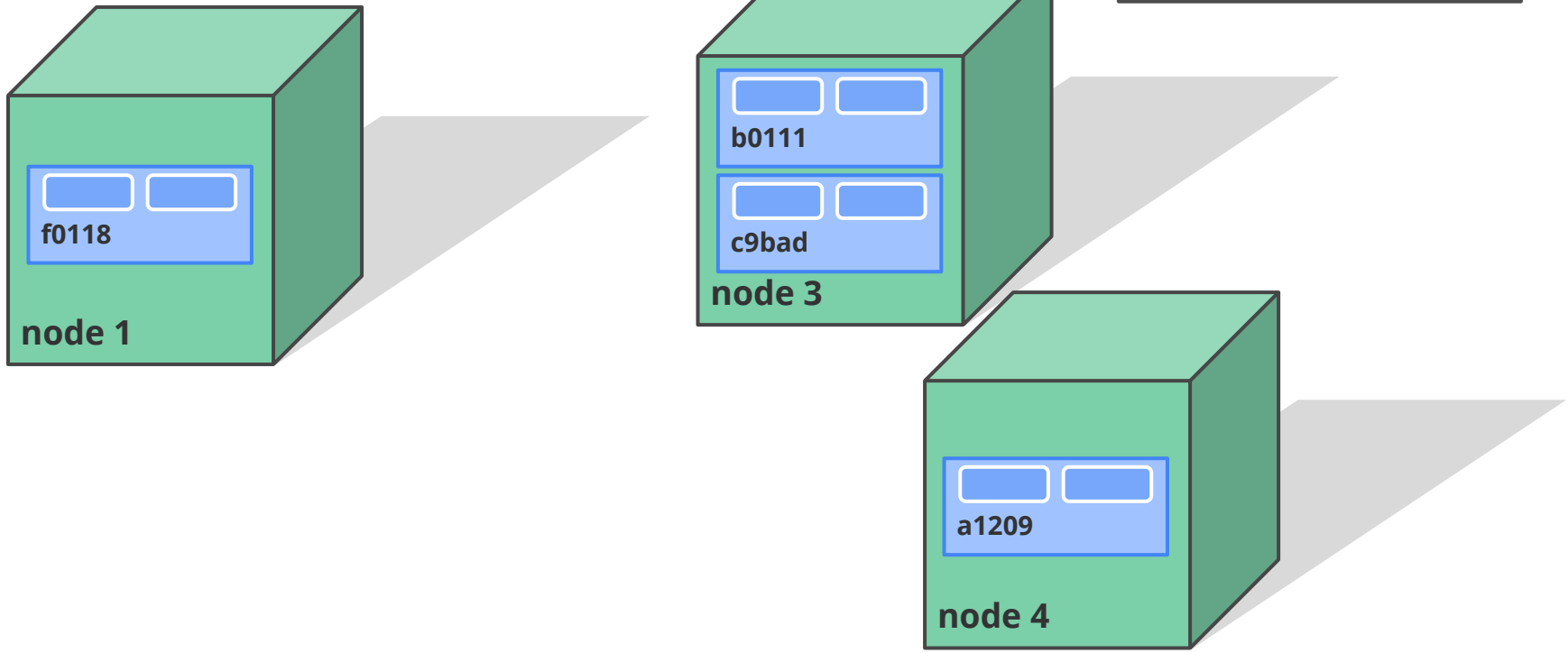
Replication Controllers



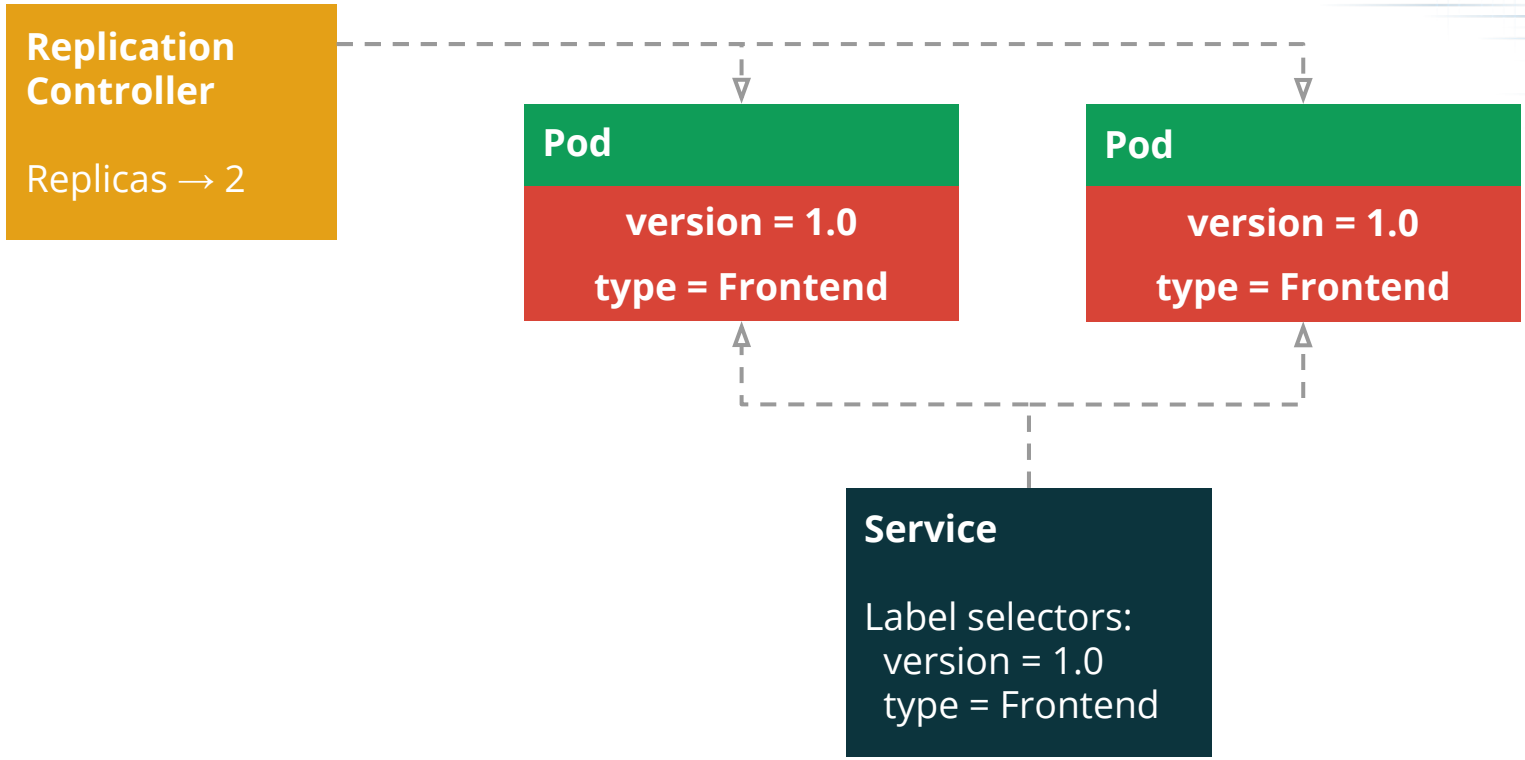
Replication Controllers



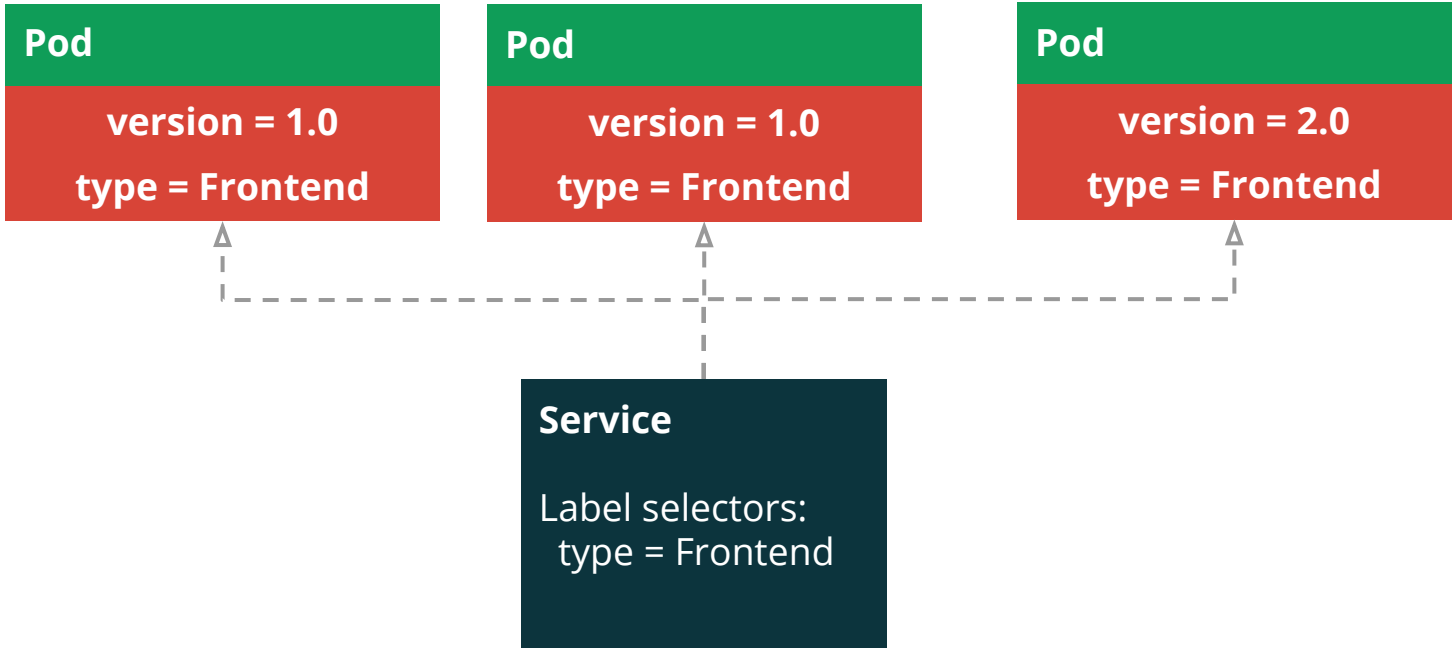
Replication Controllers



Services


Replication Controller
Replicas → 2
Pod
version = 1.0
type = Frontend
Pod
version = 1.0
type = Frontend
Service
Label selectors:
version = 1.0
type = Frontend

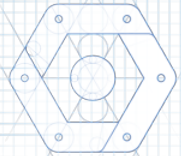
Services


Pod
version = 1.0
type = Frontend
Pod
version = 1.0
type = Frontend
Pod
version = 2.0
type = Frontend
Service
Label selectors:
type = Frontend

Service discovery

Read service IP addresses via environmental variables

```
root@b25bcd04ad3d:/app/src# export | grep REDIS_PORT
declare -x REDIS_PORT="tcp://172.17.0.244:6379"
declare -x REDIS_PORT_6379_TCP="tcp://172.17.0.244:6379"
declare -x REDIS_PORT_6379_TCP_ADDR="172.17.0.244"
declare -x REDIS_PORT_6379_TCP_PORT="6379"
declare -x REDIS_PORT_6379_TCP_PROTO="tcp"
```



Service discovery

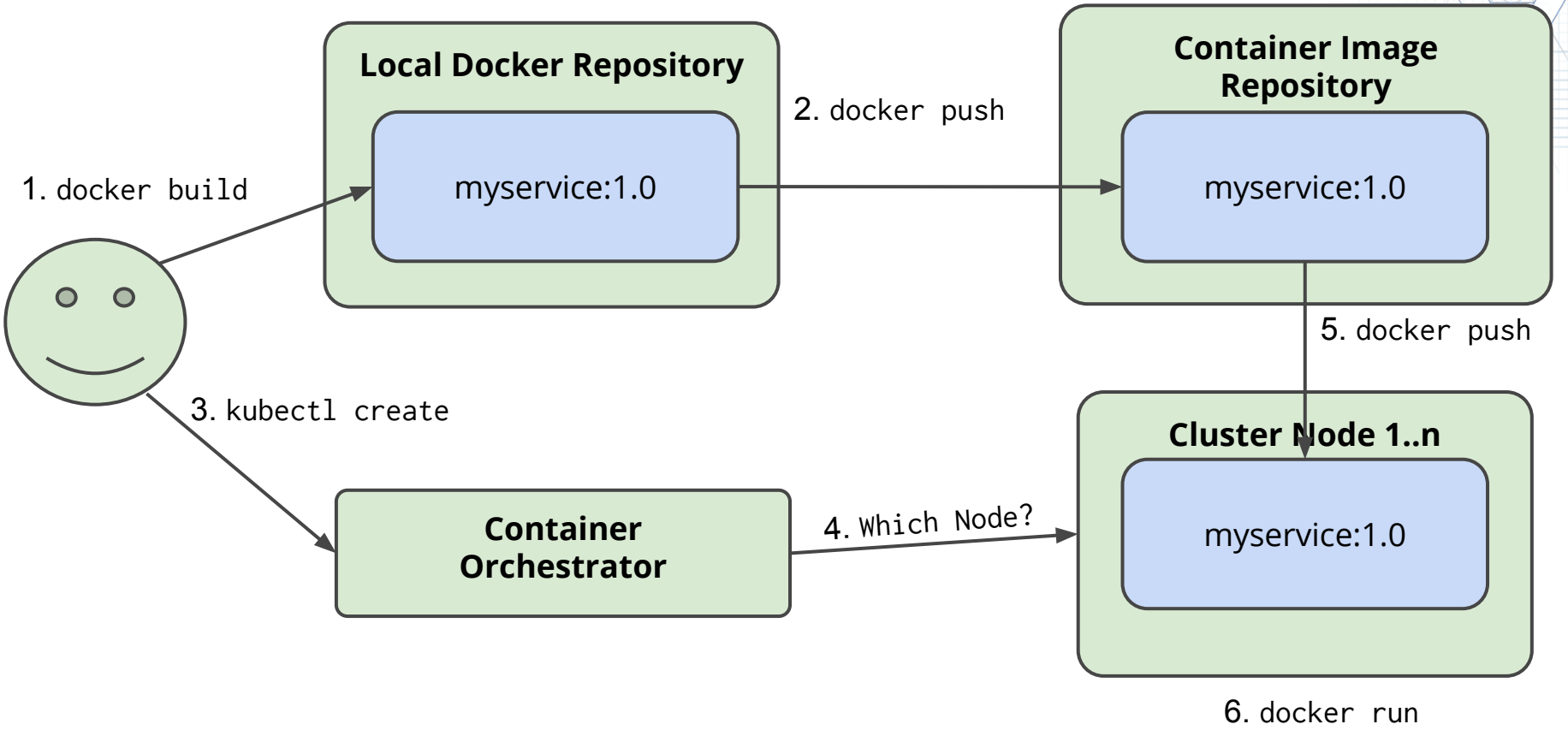
Kubernetes API

or...

DNS Lookups!

`ping redis`

Let’s Deploy


Local Docker Repository
myservice:1.0
1. docker build
2. docker push
Container Image Repository
myservice:1.0
5. docker push
3. kubectl create
Container Orchestrator
4. Which Node?
Cluster Node 1..n
myservice:1.0
6. docker run

Docker Repository

DockerHub or ... Private Repository (gcr.io)

Tag image

```
docker tag spring-boot-demo gcr.io/wise-coyote-827/spring-boot-demo
```

Push image

```
gcloud docker push gcr.io/wise-coyote-827/spring-boot-demo
```

Deploy Redis Pod

```
kubectl create -f redis-master-pod.json
```

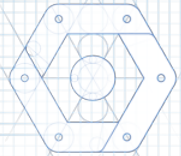
Deploy Redis Service

```
kubectl create -f redis-master-service.json
```

Deploy a fleet of microservice

```
kubectl create -f spring-boot-demo-controller.json
```

DevOps with Kubernetes



Versioning container image

```
docker tag spring-boot-demo spring-boot-demo:1.0
```

Staging vs. production

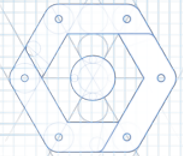
Use labels - deploy in the same infrastructure

Canary

Use service, and replication controllers to canary new versions

Rollback

Super simple with versioned containers



Rolling upgrade

Similar to canary, but slowly let the new version take over

http://kubernetes.io/**gettingstarted**/

Vagrant

```
export KUBERNETES_PROVIDER=vagrant

curl -sS https://get.k8s.io | bash
```

Google Container Engine - Beta

http://cloud.google.com/container-engine

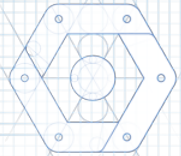
Tips and Tricks

Test locally!

Set environmental variables

Use Docker Machine

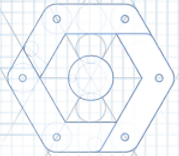
In the cloud - faster network to download images



SecureRandom - slow =(

For development and testing

```
-Djava.security.egd=file:/dev/urandom
```

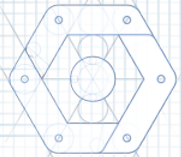


Don’t Log to Container Filesystem!

Log to a volume... docker -v /tmp/log:/log

Or

Send it elsewhere!

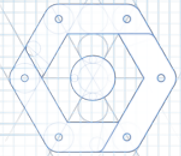


Clean up disk spaces

Every image, layer, and, even containers litters

```
docker rm $(docker ps -a -q)
```

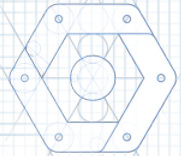
```
docker rmi $(docker images -q --filter dangling=true)
```



Combine RUN commands

apt-get update && apt-get install xyz && apt-get clean

Saves you space.



Session affinity

Scalable systems don’t need it...

But if absolutely needed, base on ClientIP

Use Spring Profile

One container - run on Docker Machine, Kubernetes, and App Engine!

Thanks!

http://bit.ly/1QLg5E1

http://kubernetes.io

Images by Connie Zhou