@meteatamel

Introduction to Kubernetes

Mete Atamel
Developer Advocate for Google Cloud

Google Cloud Platform

Mete Atamel

Developer Advocate for Google Cloud Platform

@meteatamel

atamel@google.com

meteatamel.wordpress.com

Who are you?

Agenda

The Monolith
What is the Monolith and why is it bad?

Breaking The Monolith into Microservices
Why Microservices is the way to go?

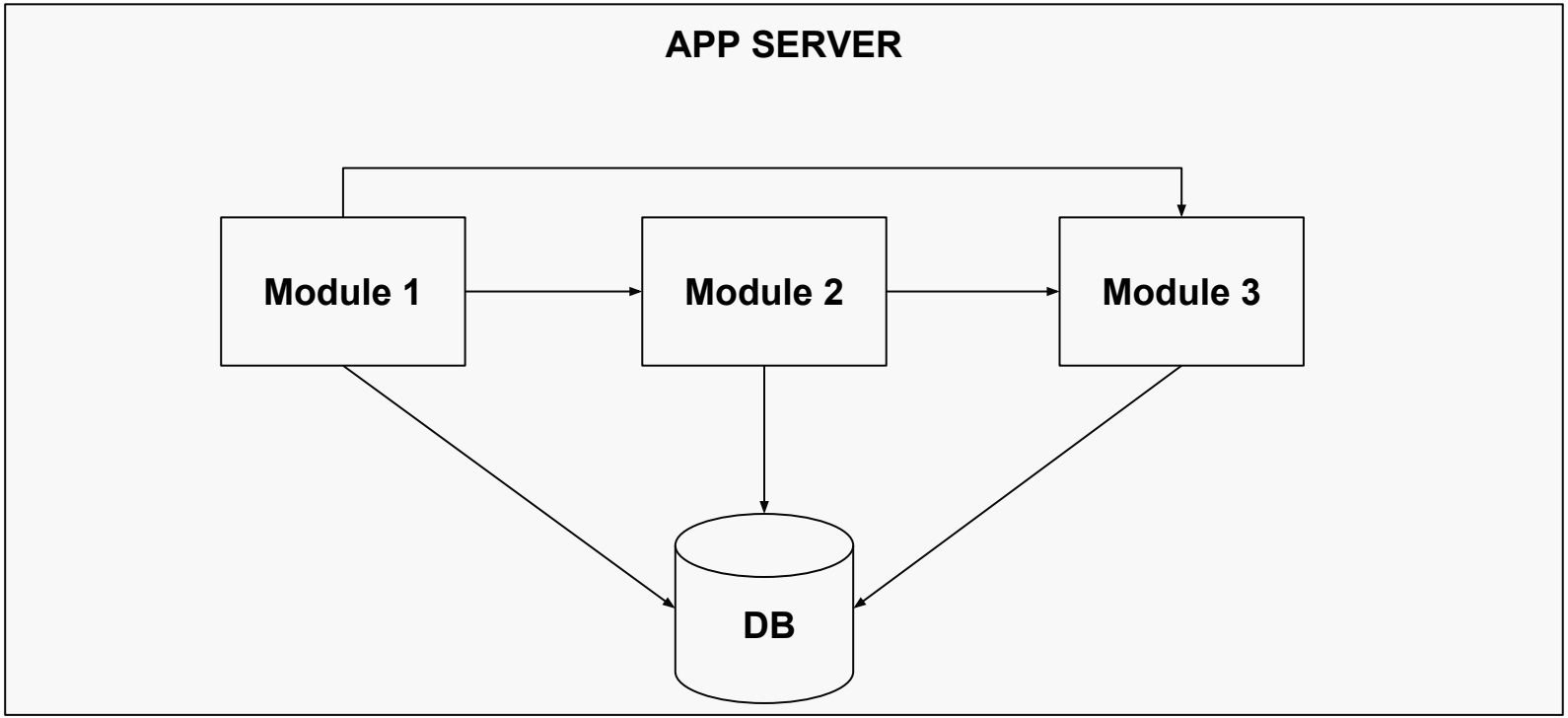
Problems with Microservices
Exchanging one set of problems with another?

Containers and Kubernetes
What are containers and Kubernetes, how do they help?

Kubernetes building blocks
Pods, services, replication controllers/set and more

The Monolith

What is the Monolith?



Problems with the Monolith

Unnecessary tight coupling among different modules

All at once, or **none at all** update policy

Hard to scale different parts independently

Ignores different development velocity of different teams completely

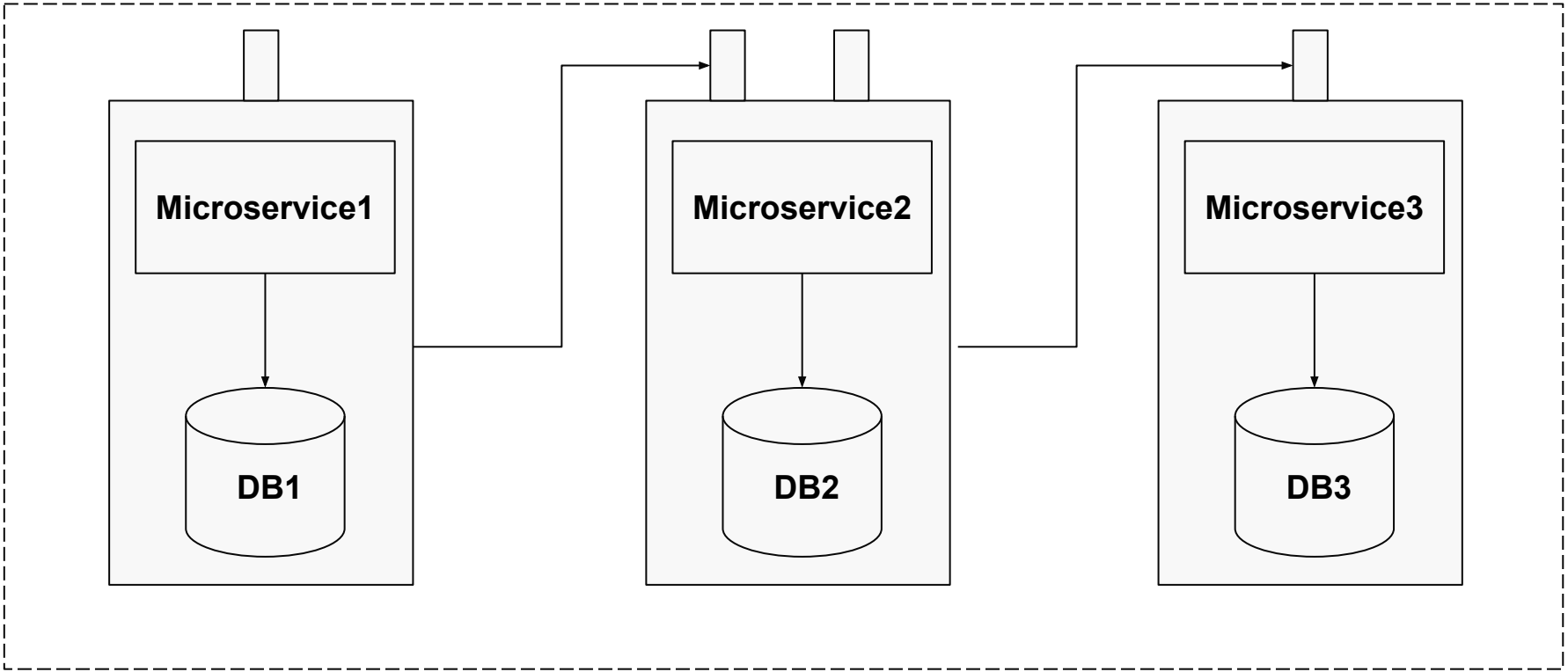
Hard to establish ownership of the whole system as it’s huge

Hard to debug and test in general, hard to run on a single development machine

@meteatamel

Breaking the Monolith into Microservices

The Monolith to Microservices



Problems with Microservices

Need to worry about multiple independent systems instead of one

Can be hard to debug and test across multiple services without proper logging

“But it works on my machine!” problem still applies

Common maintenance problems still apply: Redundancy, resilience, rolling upgrades, rolling downgrades

Containers and Kubernetes

Quick recap of Containers

Lightweight
Hermetically sealed
Isolated

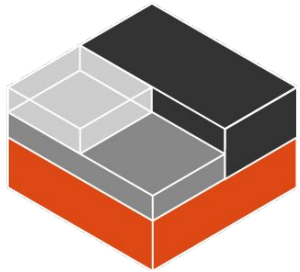
Easily deployable
Introspectable
Runnable

Linux processes

Improves overall developer experience
Fosters code and component reuse
Simplifies operations for cloud native applications

Docker

rkt



Everything at Google runs on containers

Gmail, Web Search, Maps, ...
MapReduce, batch, ...
GFS, Colossus, ...
Google's Cloud Platform: VMs run in containers!

We launch over **2 billion** containers per week

Containers are great but not enough

Containers help to create a lightweight and consistent environment for apps

But it does not solve common app management problems:

- Deploy your a new version of your app reliably
- Create resiliency
- Scale up and down
- Rollback a deployment
- Health checks
- Graceful shutdown
- Etc. etc. etc.

Kubernetes comes to rescue

http://kubernetes.io

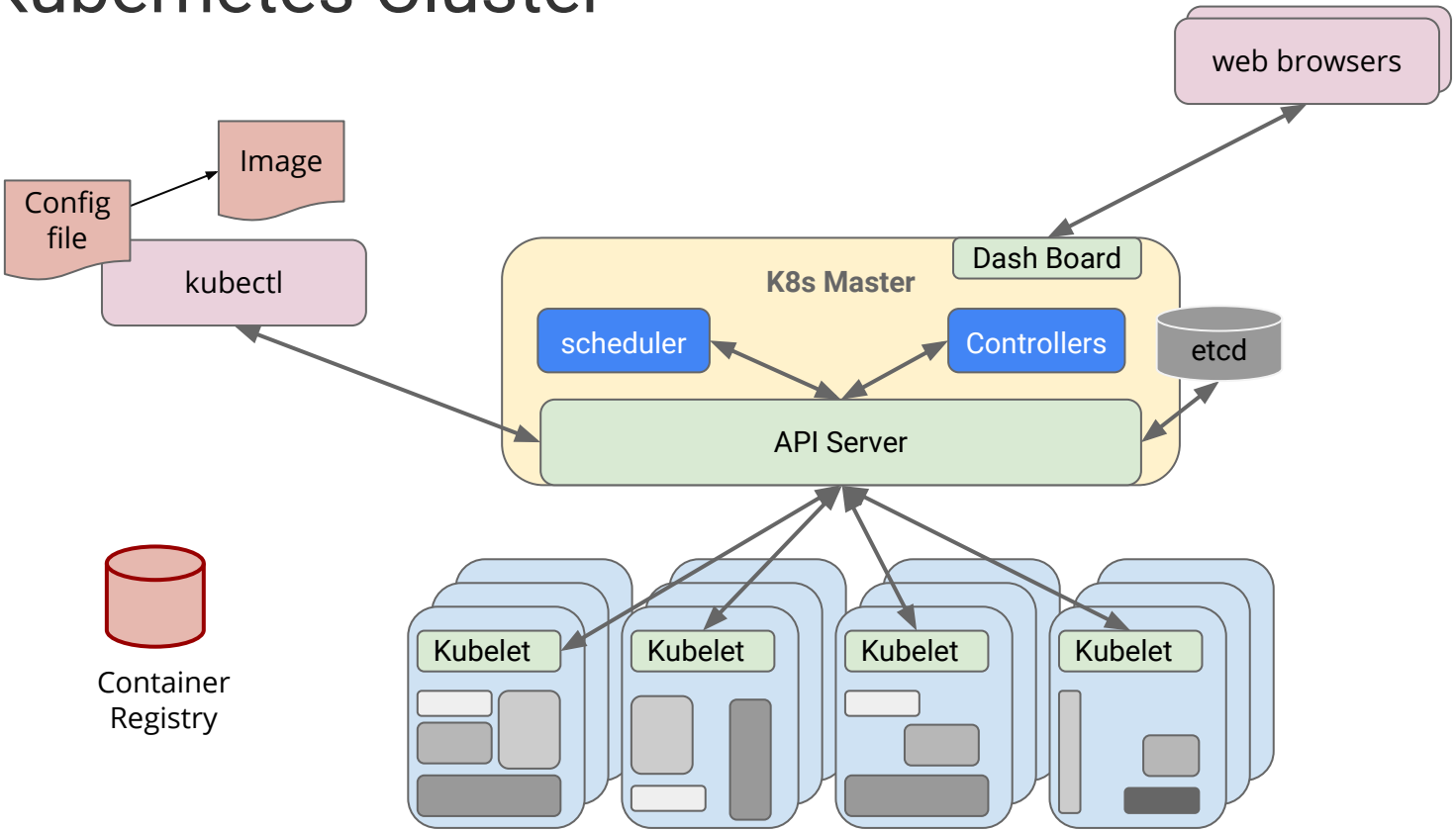
Open source container management platform

Based on years of experience running Borg at Google

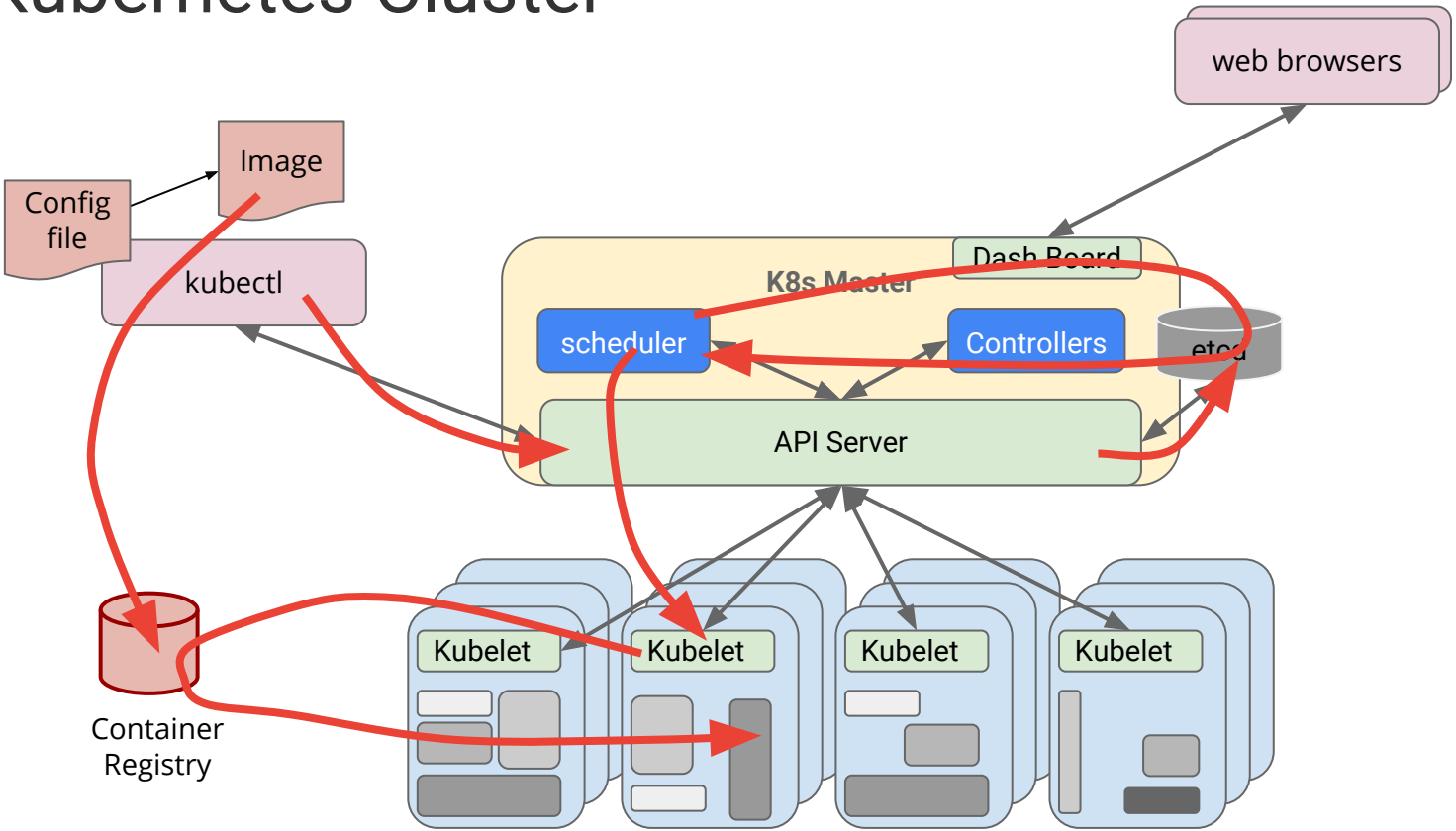
Runs everywhere: your laptop, on-prem, different cloud platforms

Helps with reliable deployment of apps, scaling, roll out and roll back of versions, autoscaling, health checks and more!

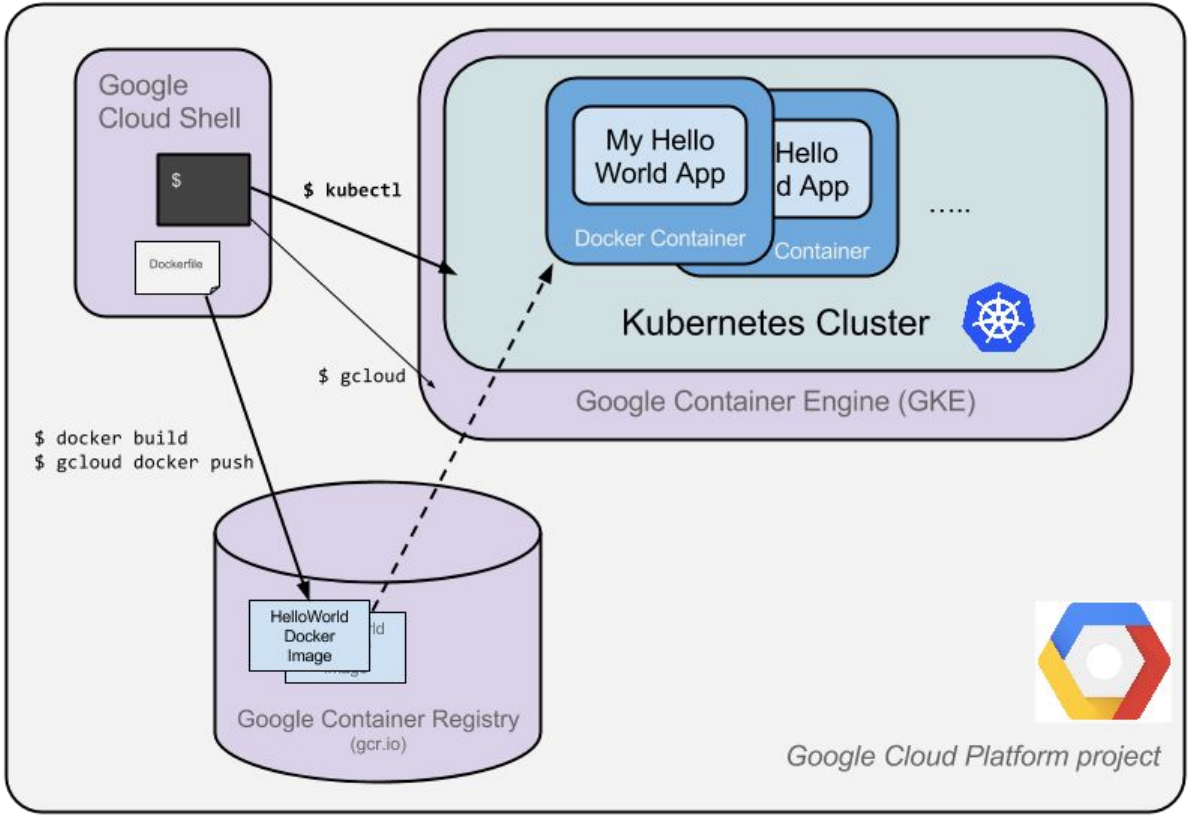
Kubernetes Cluster



Kubernetes Cluster



Kubernetes Cluster on GKE



Kubernetes Building Blocks

Pods

The atom of scheduling for containers

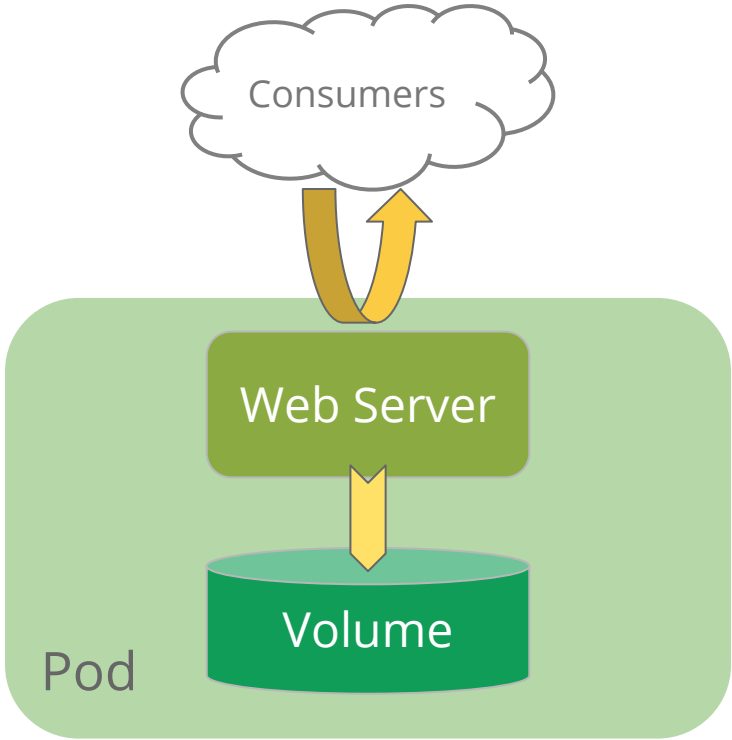
Represents an application specific **logical host**

Hosts **containers** and **volumes**

Each has its own routable (no NAT) IP address

Ephemeral

- Pods are functionally identical and therefore ephemeral and replaceable



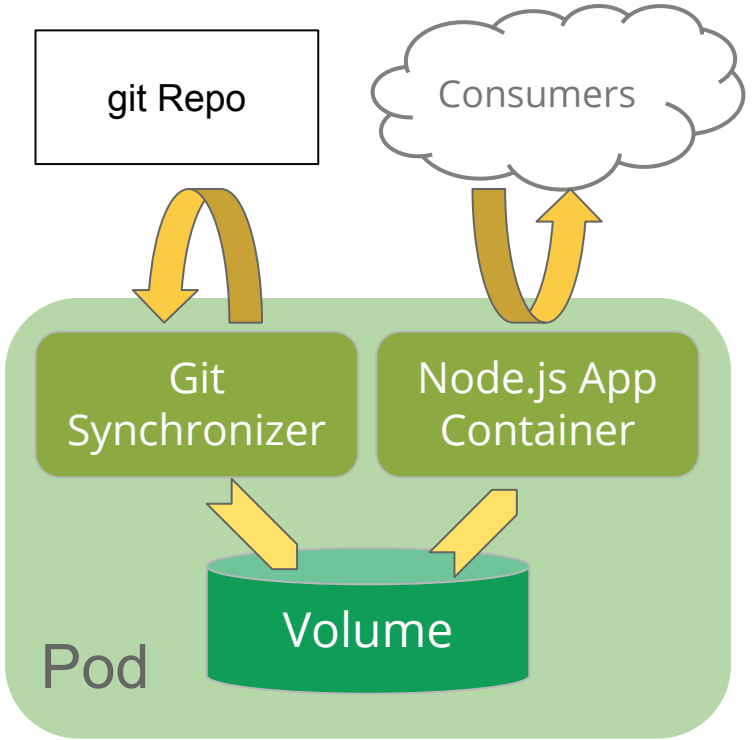
Pods

Can be used to group multiple containers & shared volumes

Containers within a pod are **tightly** coupled

Shared namespaces

- Containers in a pod share IP, port and IPC namespaces
- Containers in a pod talk to each other through localhost



Pods

Pods have IPs which are routable

Pods can reach each other without NAT

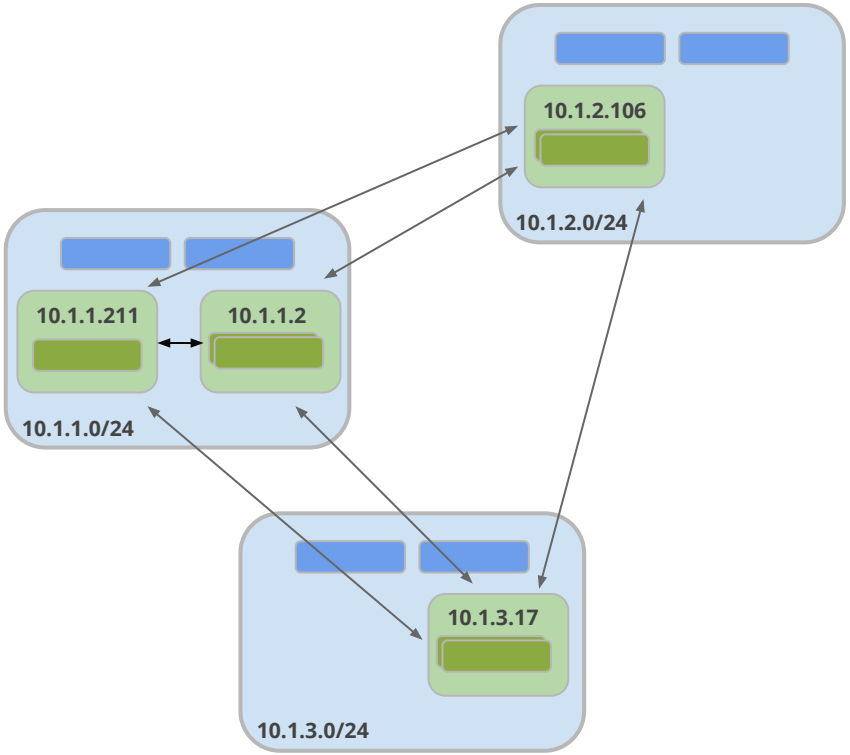
- Even across nodes

No Brokering of **Port Numbers**

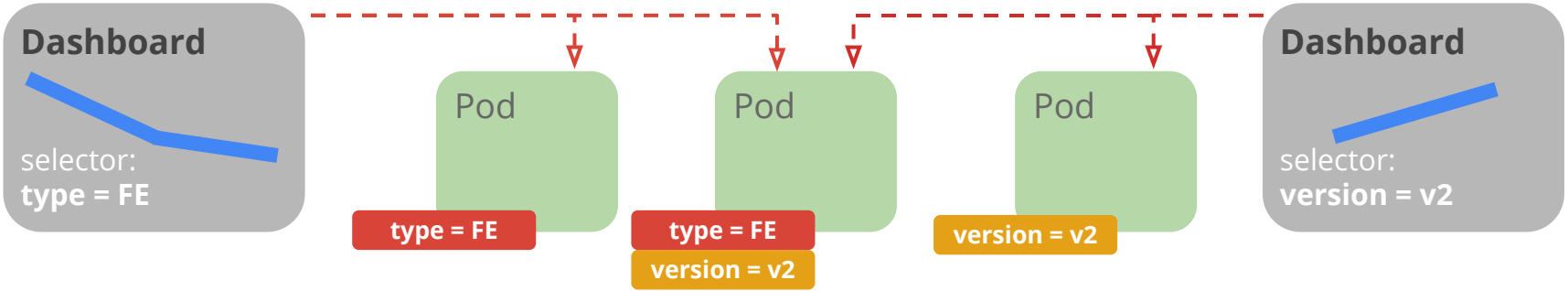
These are fundamental requirements

Many solutions

- GCE Advanced Routes, AWS Flannel, Weave, OpenVSwitch, Cloud Provider



Labels



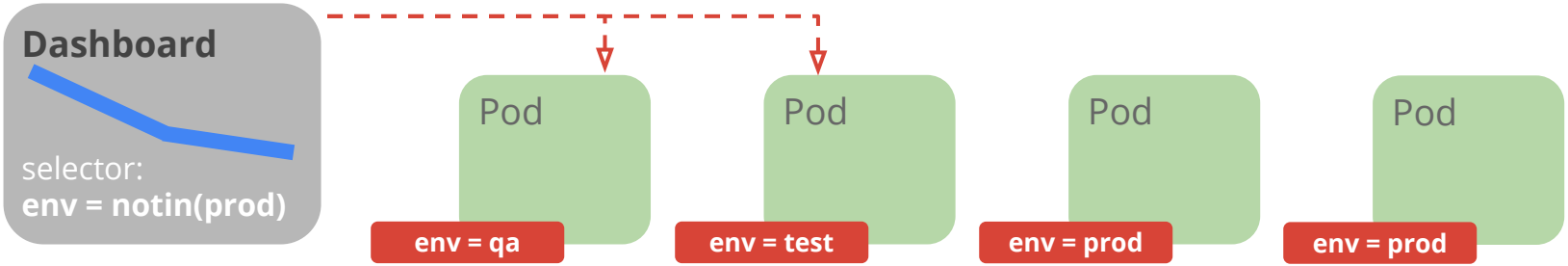
Behavior

- Metadata with semantic meaning
- Membership identifier
- The only Grouping Mechanism

Benefits

- ➔ Allow for intent of many users (e.g. dashboards)
- ➔ Build higher level systems …
- ➔ Queryable by Selectors

Label Expressions



Expressions

- env = prod
- tier != backend
- env = prod, tier !=backend
- env **in** (test,qa)
- release **notin** (stable,beta)
- tier
- **!**tier

Services

A logical grouping of pods that perform the same function (the Service's endpoints)

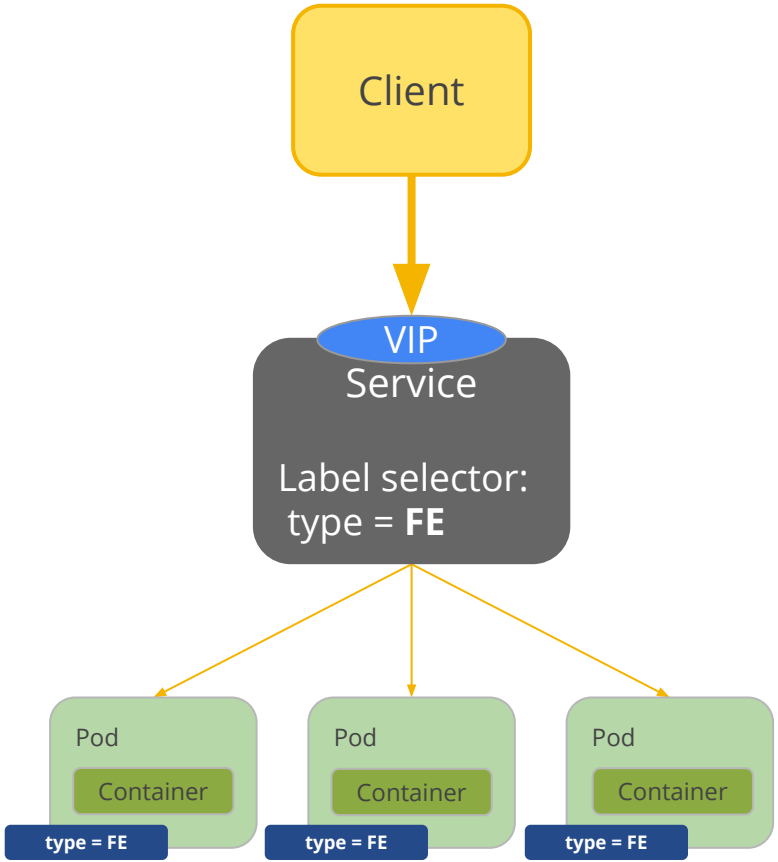
- grouped by label selector

Load balances incoming requests across constituent pods

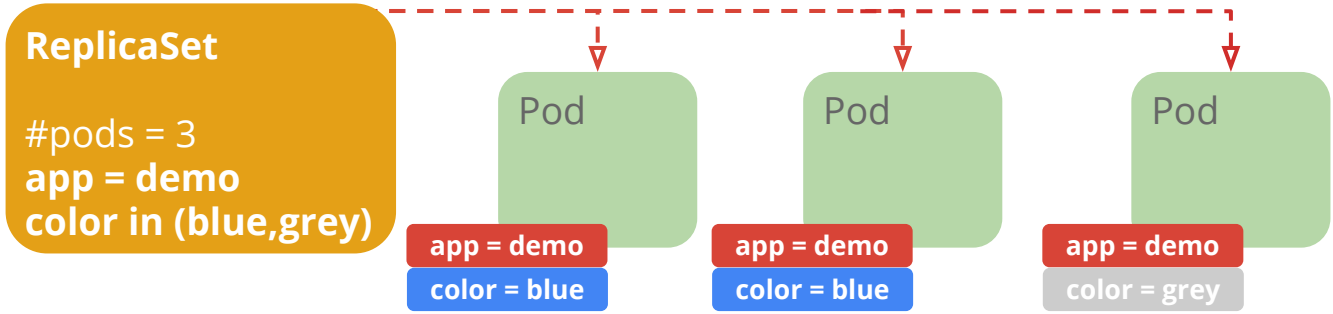
Choice of pod is random but supports session affinity (ClientIP)

Gets a **stable** virtual IP and port

- also a DNS name



Replication Controllers/Sets



Behavior

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

Benefits

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

Replication Controllers/Sets

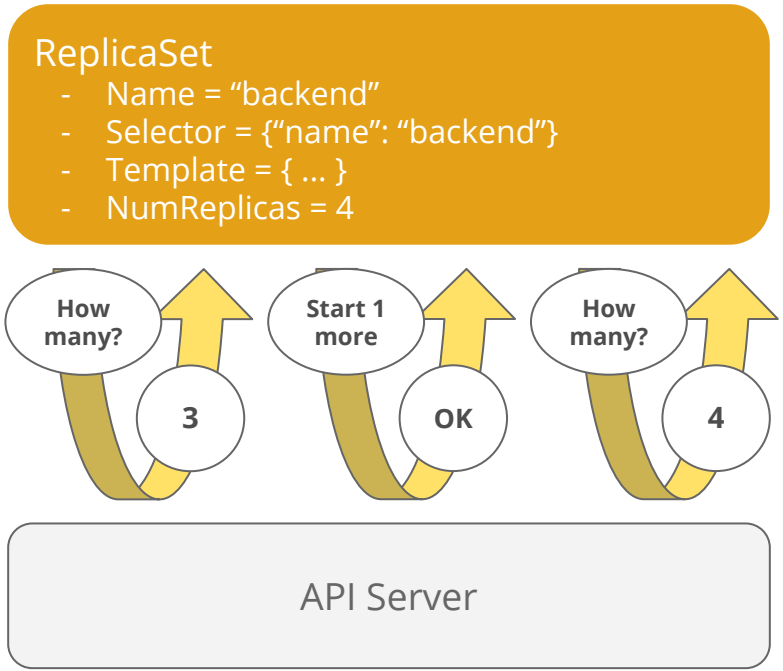
Canonical example of control loops

Have one job: ensure N copies of a pod

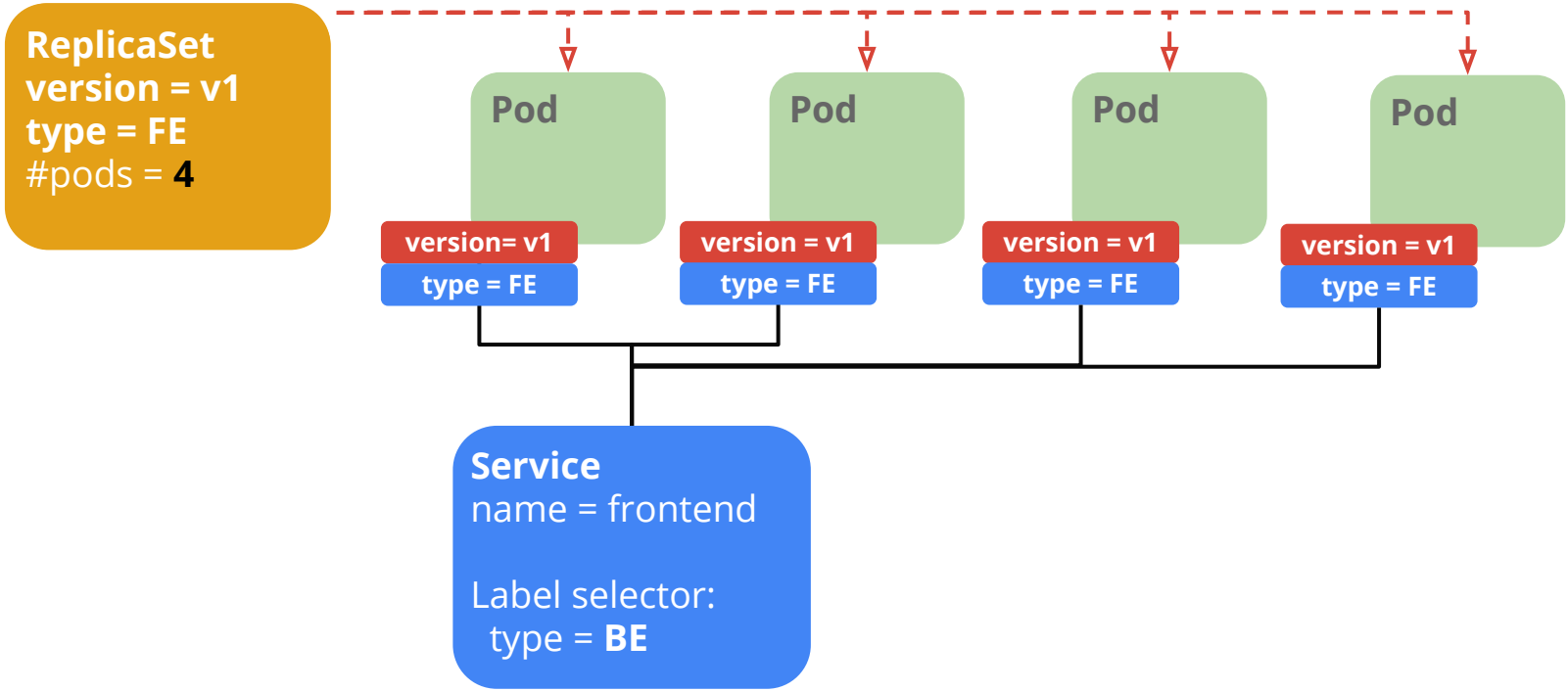
- if too few, start new ones
- if too many, kill some
- group == selector

Replicated pods are fungible

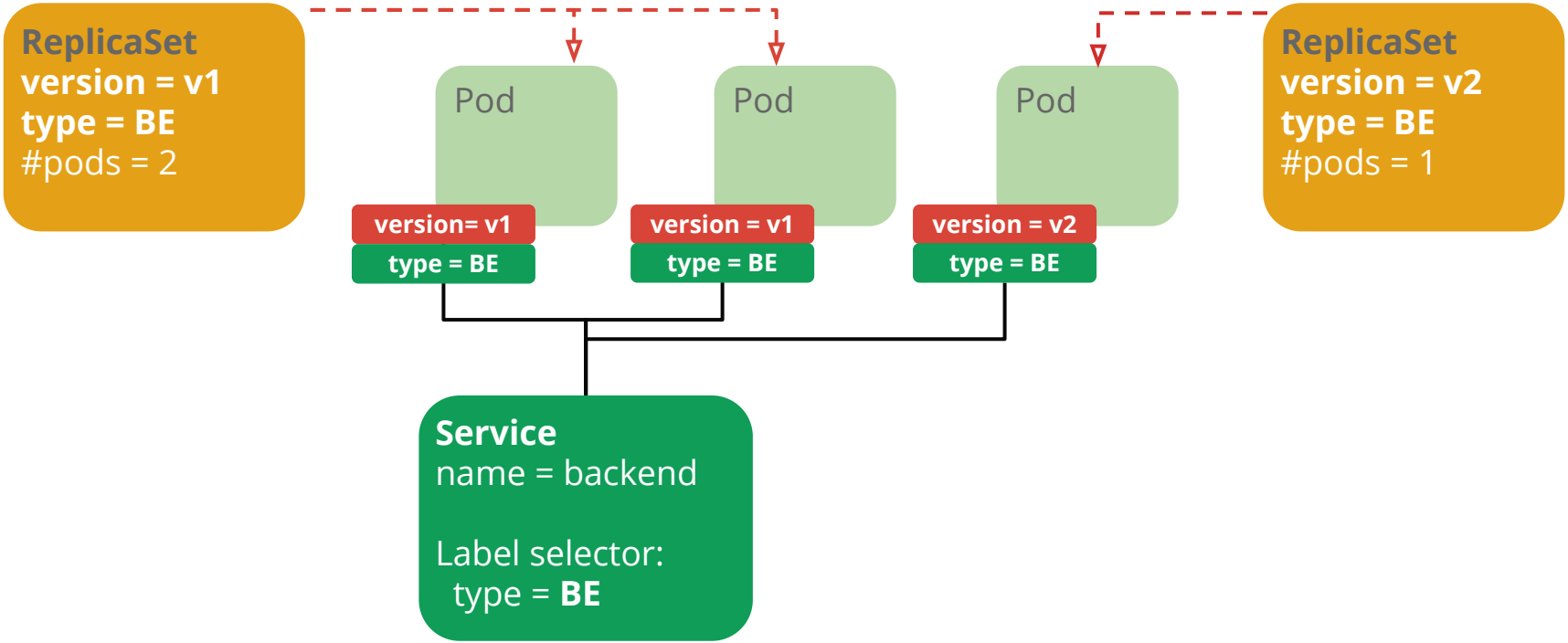
- No implied order or identity



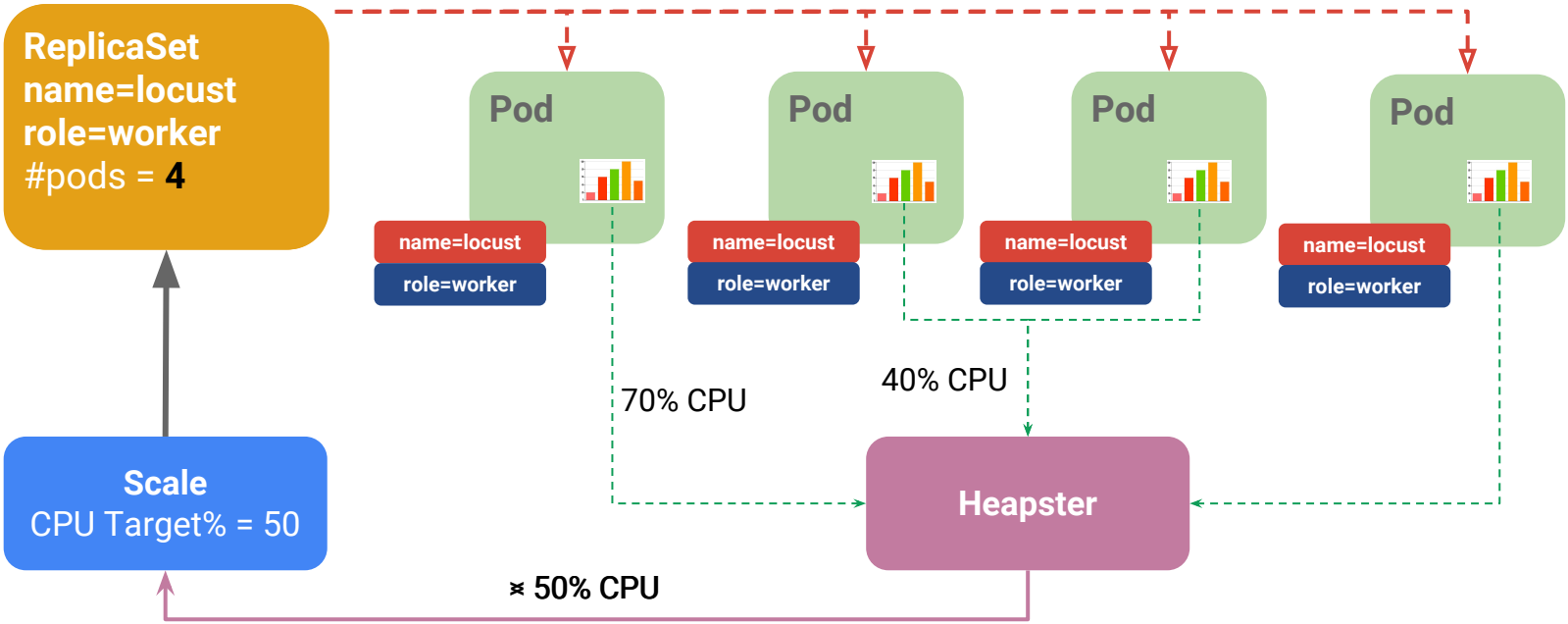
Scaling


ReplicaSet
version = v1
type = FE
#pods = 4
Pod
Pod
Pod
Pod
version= v1
type = FE
version = v1
type = FE
version = v1
type = FE
version = v1
type = FE
Service
name = frontend
Label selector:
type = BE

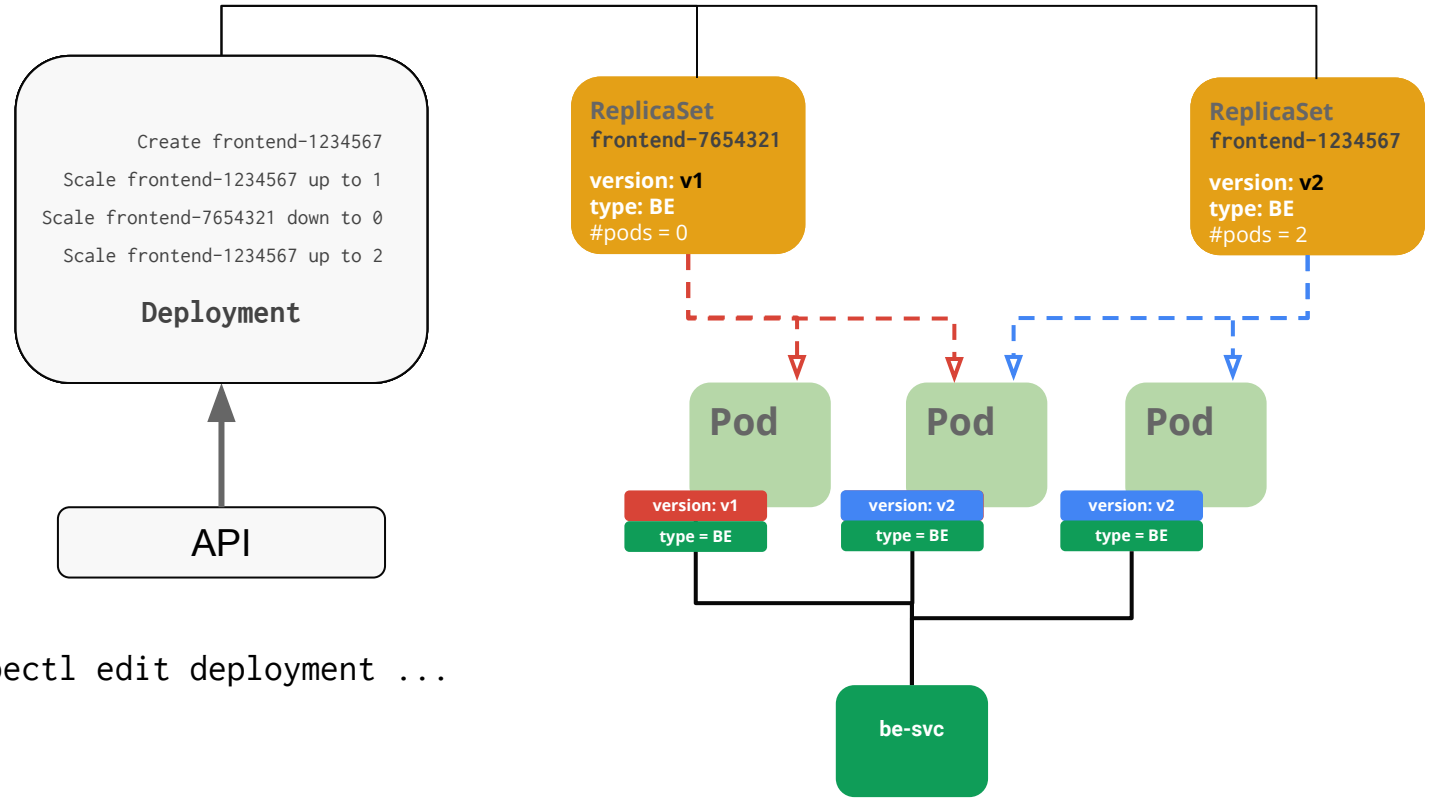
Canary


ReplicaSet
version = v1
type = BE
#pods = 2
Pod
Pod
Pod
version= v1
type = BE
version = v1
type = BE
version = v2
type = BE
ReplicaSet
version = v2
type = BE
#pods = 1
Service
name = backend
Label selector:
type = BE

Autoscaling



Rollout




```
kubectl edit deployment ...
```

There is much more!

Thank You

kubernetes.io
cloud.google.com/container-engine

Mete Atamel
@meteatamel
atamel@google.com
meteatamel.wordpress.com