Kubernetes:
The Path to Cloud Native

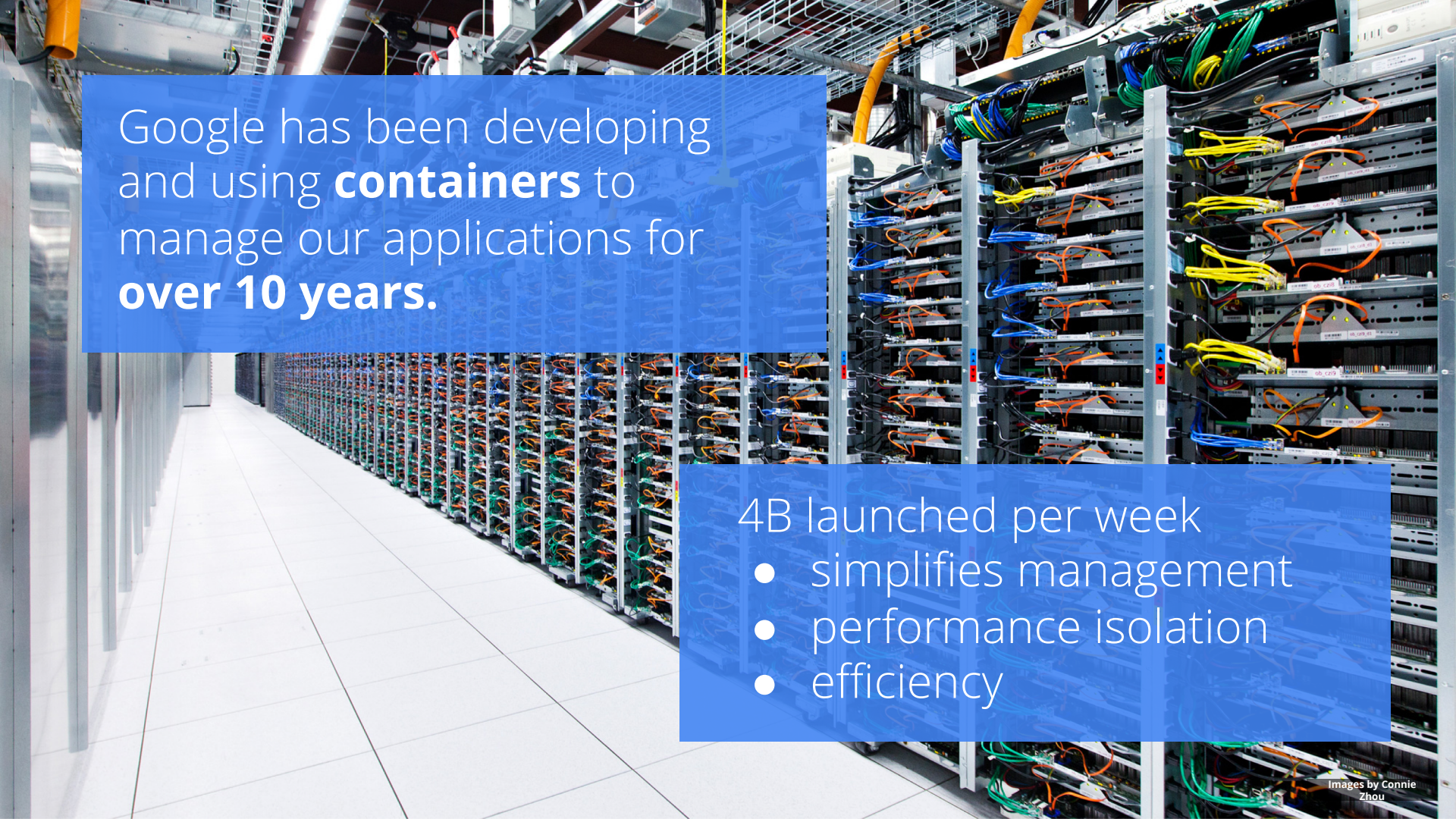
Eric Brewer
VP, Infrastructure

November 2019

“Cloud Native” Applications

Middle of a great transition

- unlimited “ethereal” resources in the Cloud
- an environment of *services* not machines
- thinking in APIs and co-designed services
- high availability offered and expected



Google has been developing and using **containers** to manage our applications for **over 10 years.**

4B launched per week

- simplifies management
- performance isolation
- efficiency

Images by Connie Zhou

Merging Two Kinds of Containers

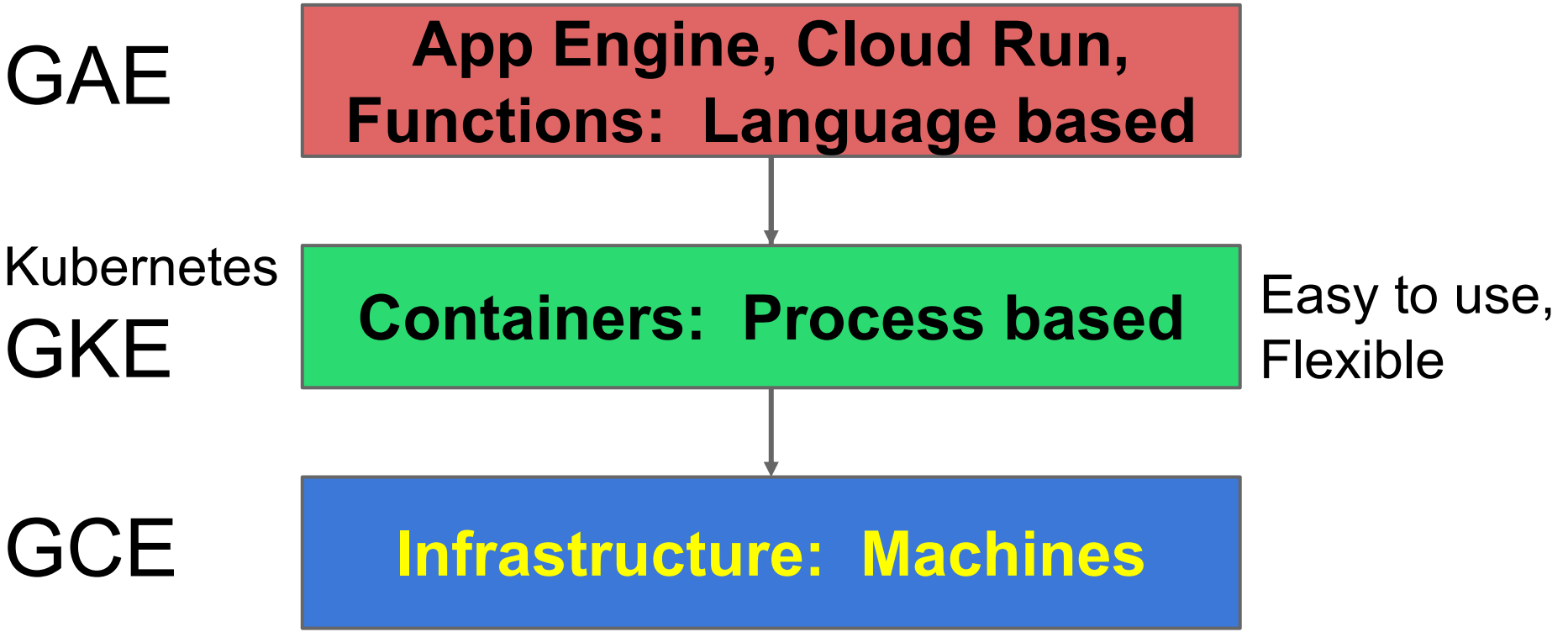
Docker

- It's about ***packaging***
- Control:
  - packages
  - versions
  - (some config)
- Layered file system
- => Prod matches testing

Linux Containers

- It's about ***isolation***
  … *performance isolation*
- not *security* isolation
  … use VMs for that
- Manage CPUs, memory, bandwidth, …
- Nested groups

Google Platform Layering



Kubernetes: Higher level of Abstraction

Think About

- Composition of services
- Load-balancing
- Names of services
- State management
- Monitoring and Logging
- Upgrading

Don't Worry About

- OS details
- Packages — no conflicts
- Machine sizes (much)
- Mixing languages
- Port conflicts

Evolution is the Real Value

Apps Structured as Independent Microservices

- Encapsulated state with APIs (like “objects”)
- Mixture of languages
- Mixture of *teams*

Services are *Abstract*

- A “Service” is just a long-lived abstract name
- Varied implementations over time (versions)
- Kubernetes routes to the right implementation

Service-Oriented Architecture?

This is similar, but a also new

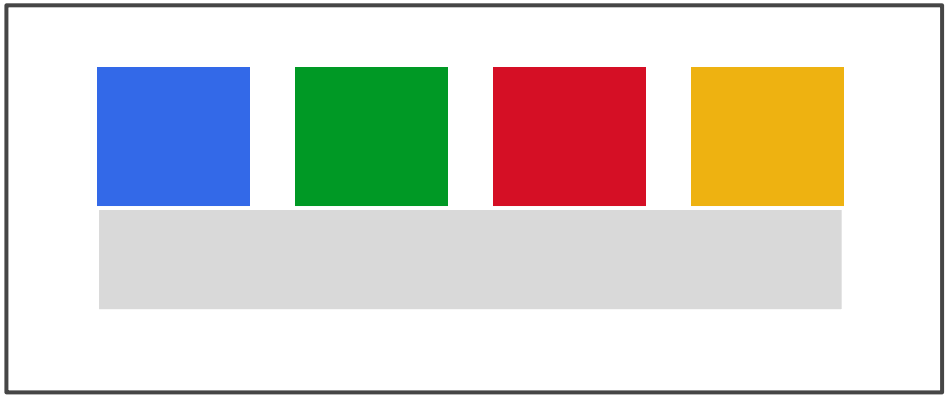
Practical difference:

- *Simple* network RPCs now common
- JSON/http for REST (or gRPC for sessions)

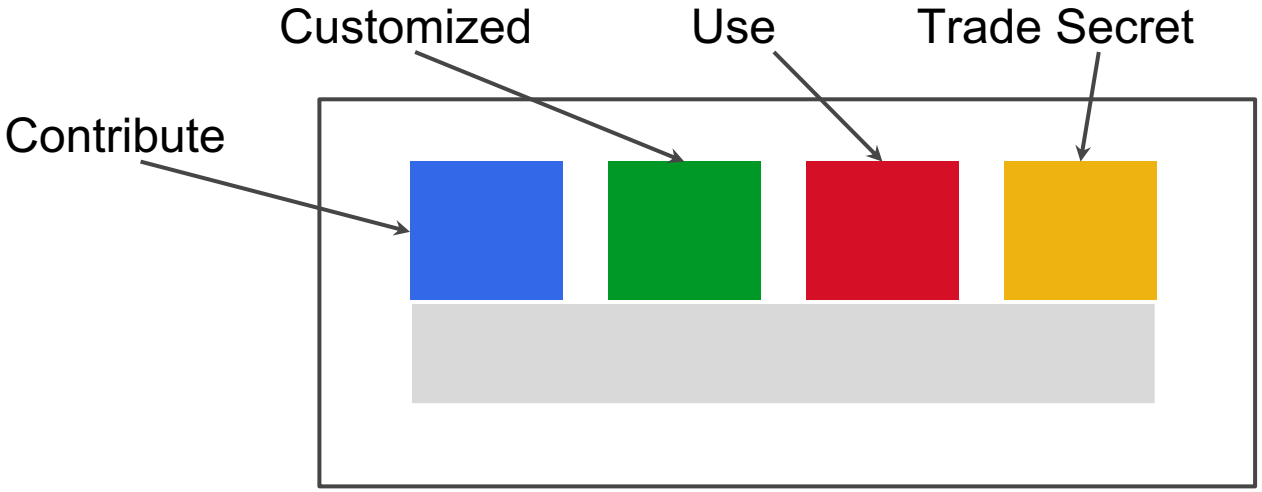
Much better *structure*

- Micro ⇒ smaller services and more of them
- New in Kubernetes: modular sub-services

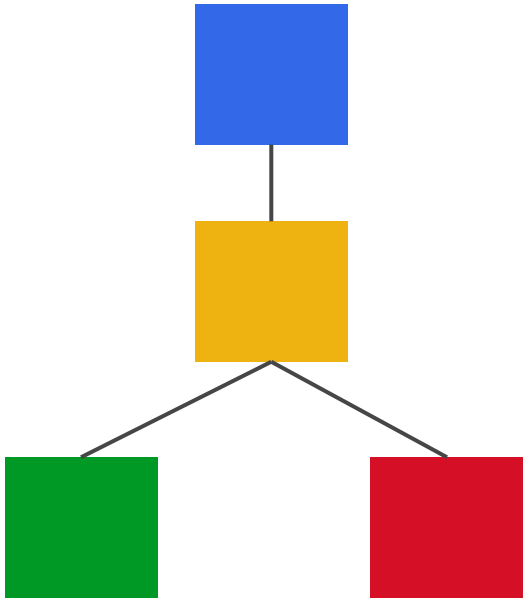
A Quick Look @ Your Code



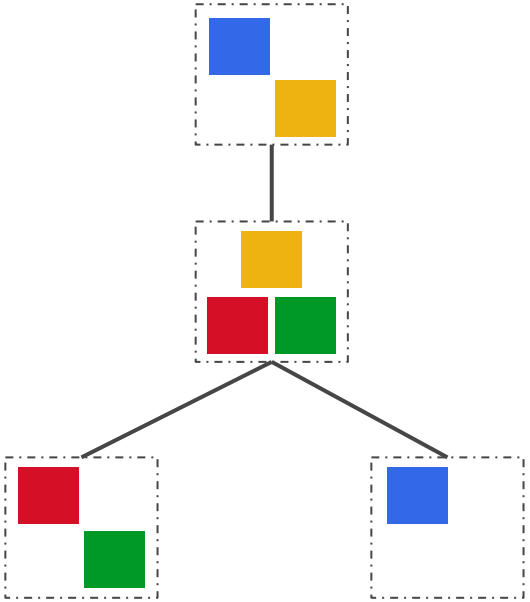
& Your Code Community



SOA… wrong granularity



Kubernetes: sub-structure



Container is ***not*** the
application boundary...

... more like a big persistent OO class..
Or a reusable building block

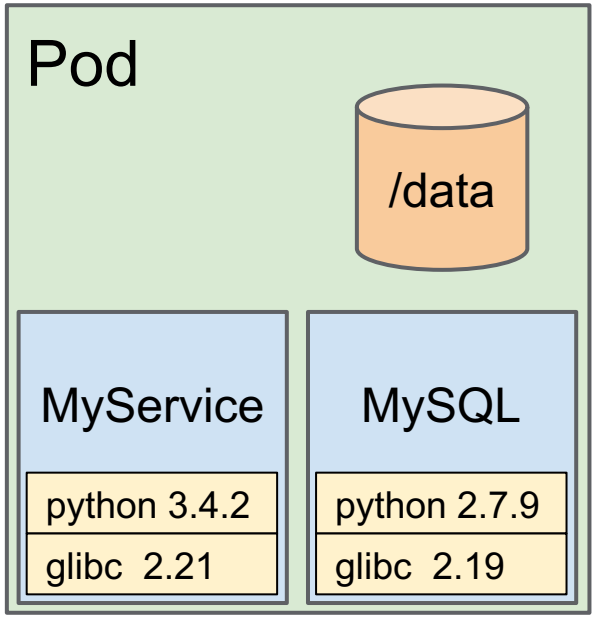
Substructure

Containers:

- Handle *package* dependencies
- Different versions, same machine
- No “DLL hell”

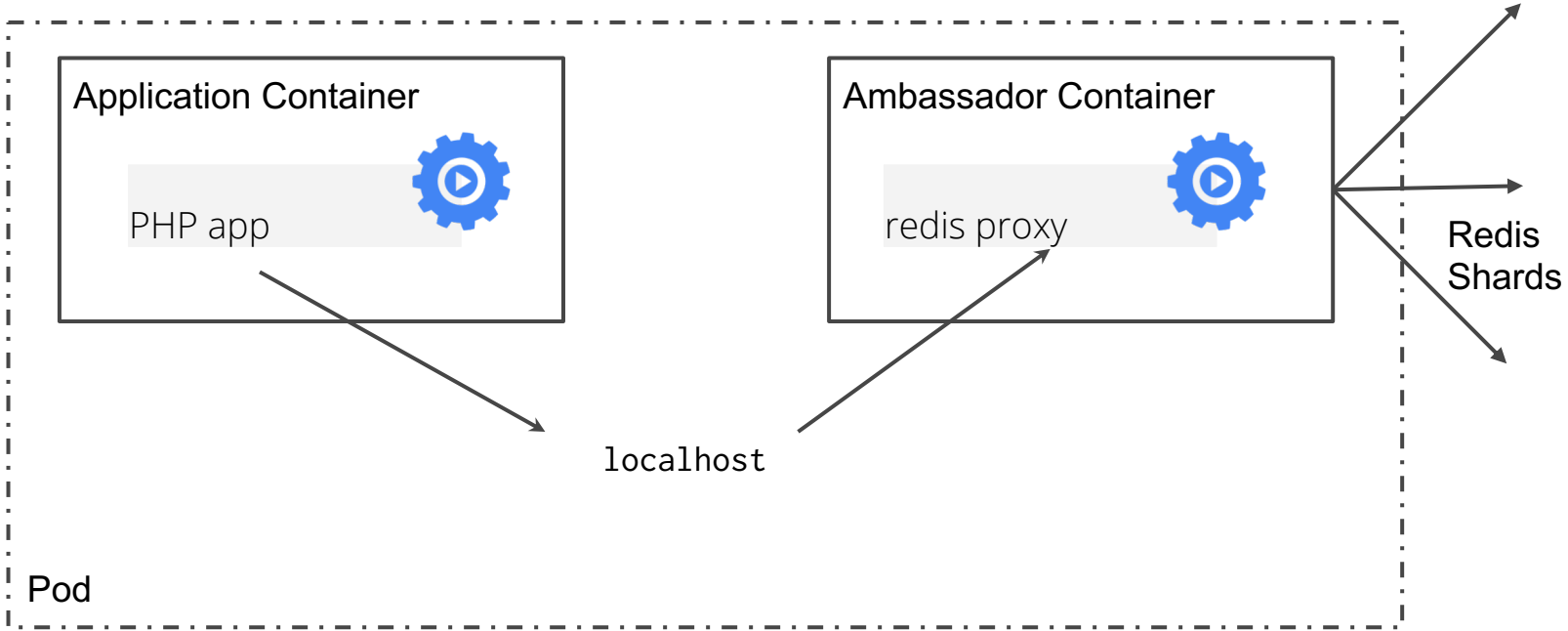
Pods:

- *Co-locate* containers
- Shared volumes
- IP address, independent port space
- Unit of deployment, migration



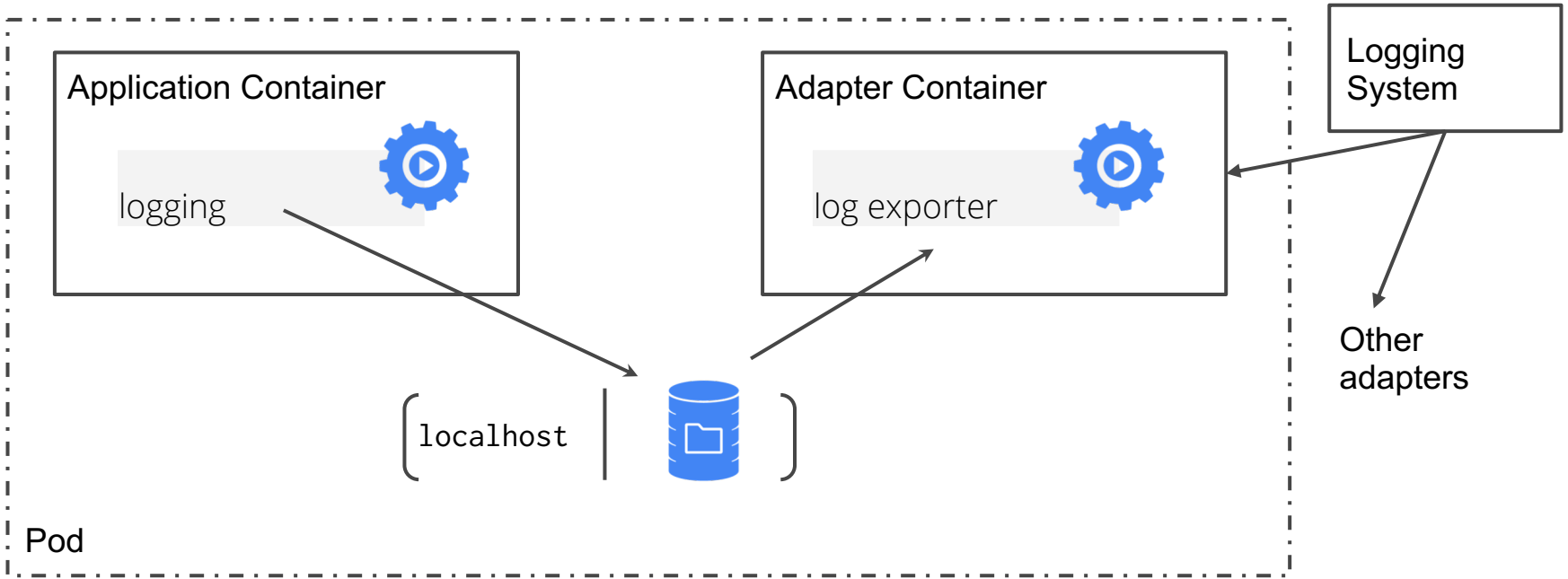
Ambassador Pattern

Ambassadors represent and present



Sidecar Pattern

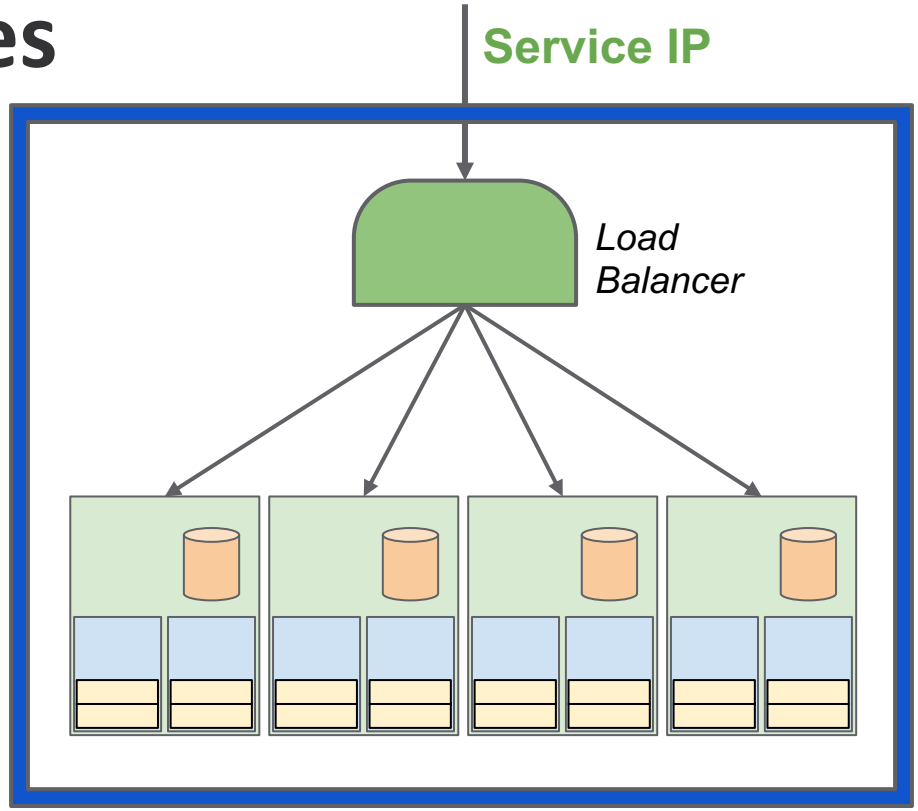
Use a logging container from another team



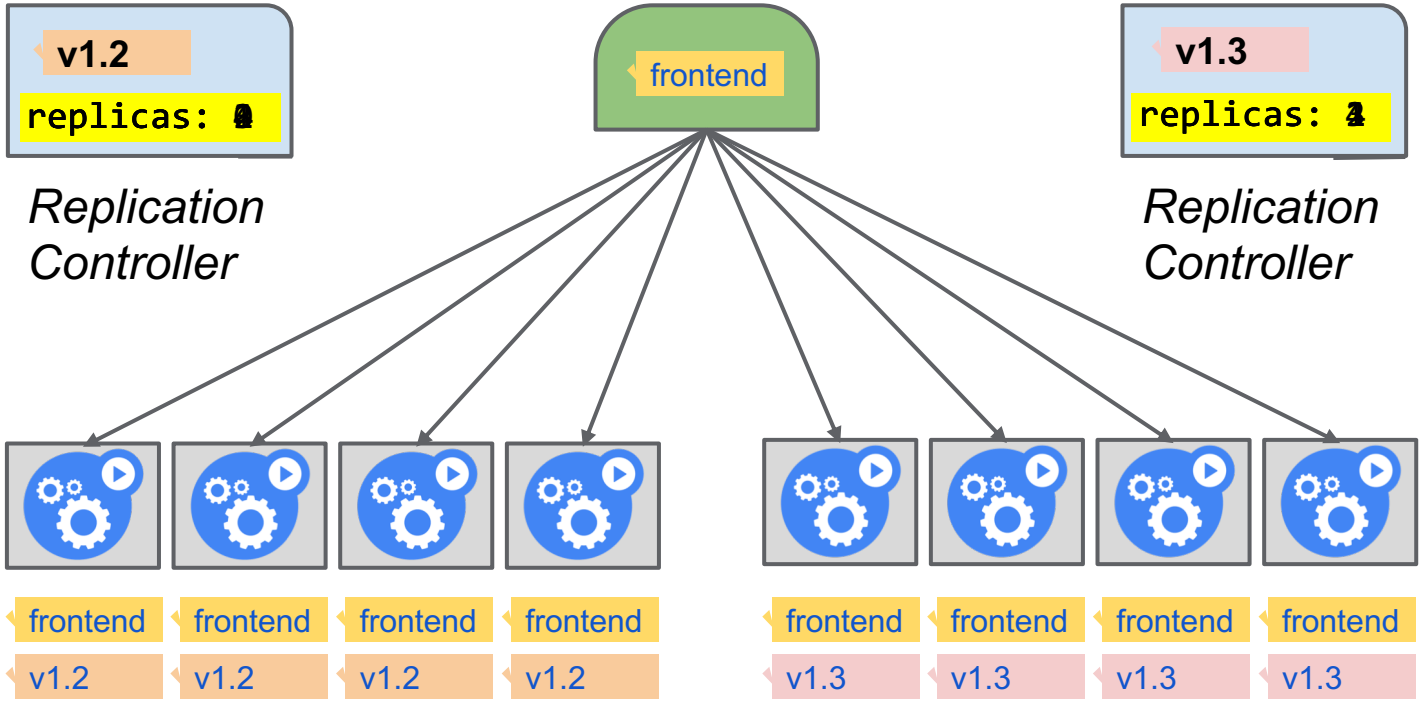
Dependencies: Services

Service:

- Replicated pods
  - Source pod is a template
- Auto-restart member pods
- Abstract name (DNS)
- IP address for the service
  - in addition to the members
- Load balancing among replicas



Example: Rolling Upgrade with Labels



What about state? (“stateful services”)

Easy option today:

Services can be stateful, but they use storage services underneath

(BigTable, Spanner, MySQL, … SAP)

Can we containerize the storage services?

Yes — all of Google’s storage systems are built this way

Current state: pods with durable disks as building blocks

- Enables MySQL, Cassandra, etc.

Service ***Mesh***: Applications are collections of services

Not uncommon to have 100s of small services

- Some with state, some without…

Goals:

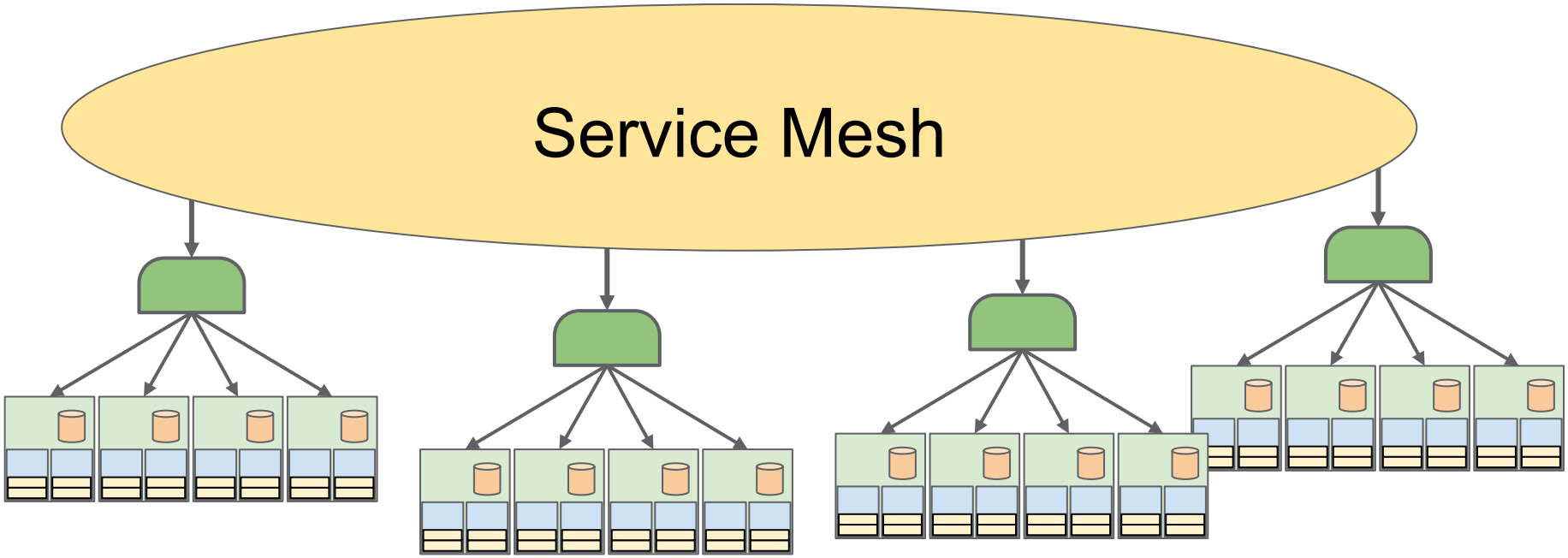
- Deploy service updates independently
  - Need versioning
- Uniform and easy support for services
- **Decouple operations from development**
  - E.g. changing logging without redeploying services

Istio: insert a services control layer using L7 proxy

Simple k8s: services have a load balancer
Istio: services have an extensible L7 proxy

- Advanced load balancing
- Telemetry: uniform data collection about services
  - E.g. latency distribution
- Security: handle auth and access control
- Quota: limit usage by some callers
- Uniform policies

Most important: **change policies without changing application code**


Service Mesh

Open Source

Key decision: core pieces are open source

- Kubernetes
- Tensorflow for ML
- Istio service mesh (launched with IBM and Lyft)
- gRPC (fast streaming RPC)
- Bazel (large-scale build system)

Kubernetes and Tensorflow are both top 0.01% projects on github

Roughly rebuilding Google in open source...

What about “serverless”

Literally means you don’t have to manage servers

- But also usage-based billing (vs resource based, eg. VMs)
- Often means “functions”

Function model by itself is not great

- Hard to manage state
- Expensive
- Hard to debug above a certain size

Many “functions as a service” (FaaS) being built on Kubernetes

Kubernetes is really a platform for platforms

Hybrid Cloud and Multi-Cloud

Strong demand to mix on prem and Cloud(s)

Open Source makes this vastly easier

Two models, both are used together:

- Partition services — run different things in different places
  - Secure bidirectional traffic with direct peering
- Consistent Environment
  - Run services in either place without code changes
  - May involve some storage replication for latency/cost

Summary

"Cloud" should run at a higher level of abstraction
… but still be able to run all the things

Containers -> pods -> services -> service mesh
Google roughly built this way internally, with some rougher edges

Open Source is a key part of driving adoption
Freedom to change environments

BACKUP

How?

Implemented by a number of (unrelated) Linux APIs:

- **cgroups:** Restrict resources a process can consume
  - CPU, memory, disk IO, ...
- **namespaces:** Change a process's view of the system
  - Network interfaces, PIDs, users, mounts, ...
- **capabilities:** Limits what a user can do
  - mount, kill, chown, ...
- **chroots:** Determines what parts of the filesystem a user can see