Google

Kubernetes
The Path to Cloud Native

Eric Brewer
VP, Infrastructure

@eric_brewer

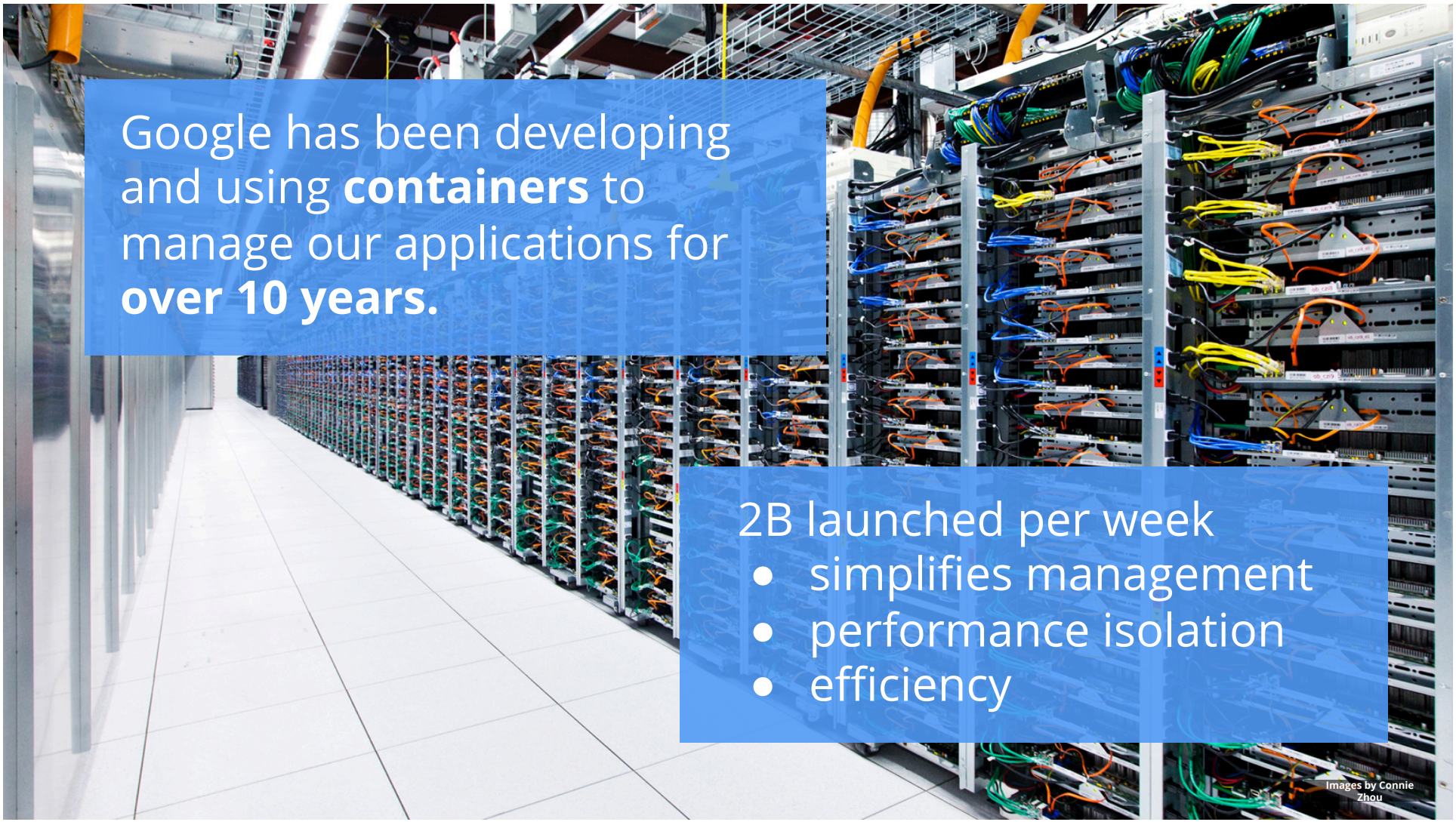
August 28, 2015
ACM SOCC

Google

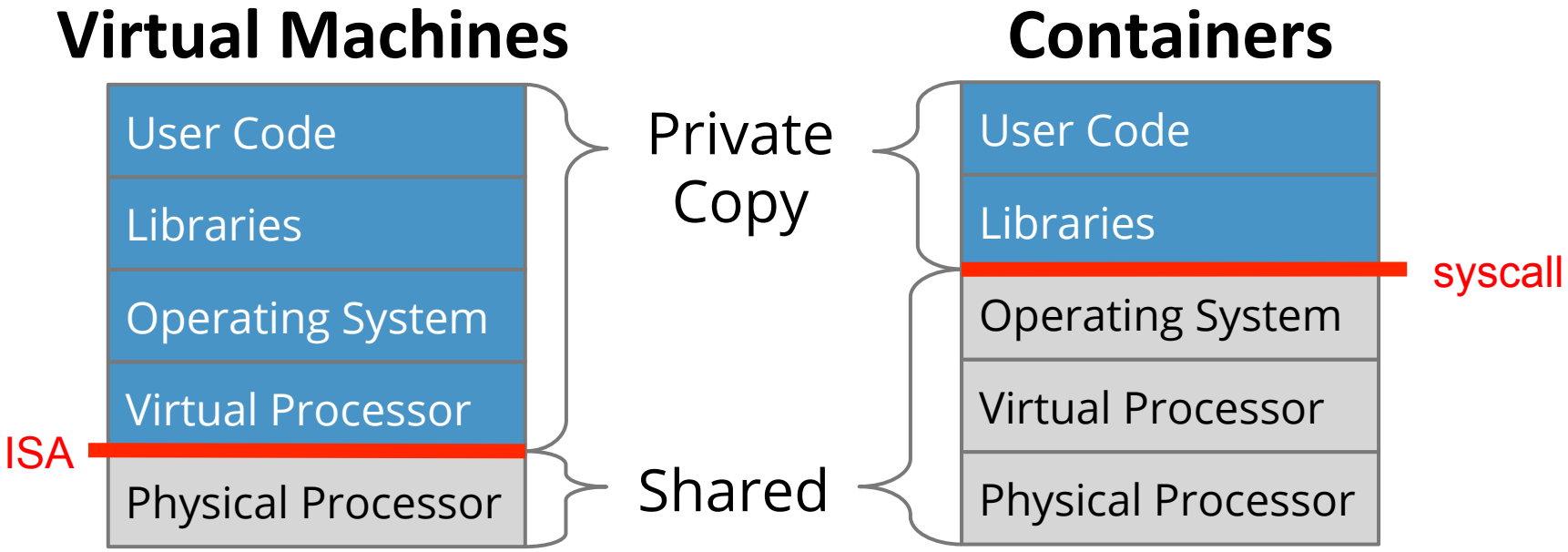
“Cloud Native” Applications

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.**
2B launched per week
- simplifies management
- performance isolation
- efficiency
Images by Connie Zhou

VMs vs. Containers



Containers: less overhead, enable more “magic”

Merging Two Kinds of Containers

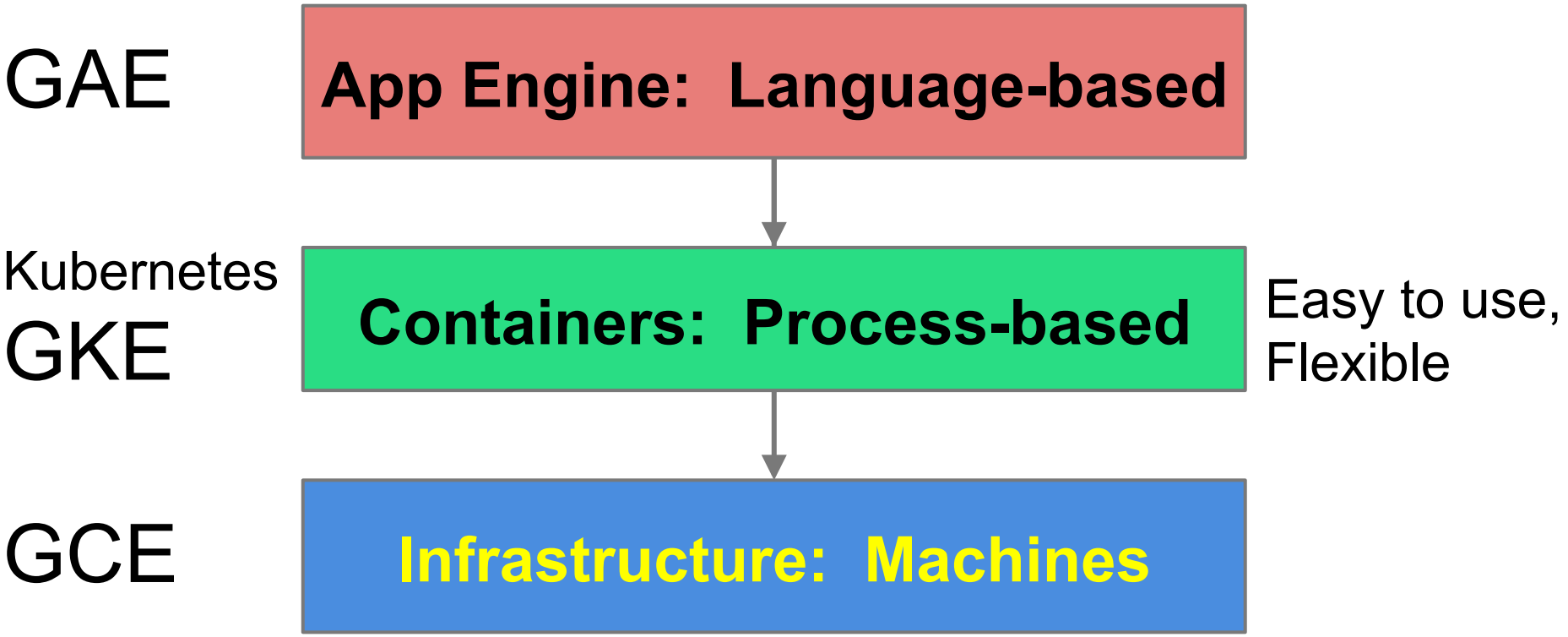
Docker

- 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



Google

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

Google

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

Google

Service-Oriented Architecture?

This is similar, but a also new

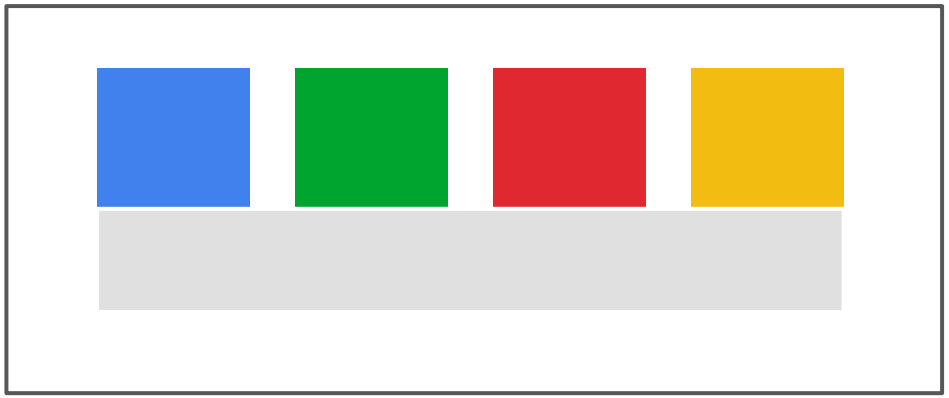
Practical difference:

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

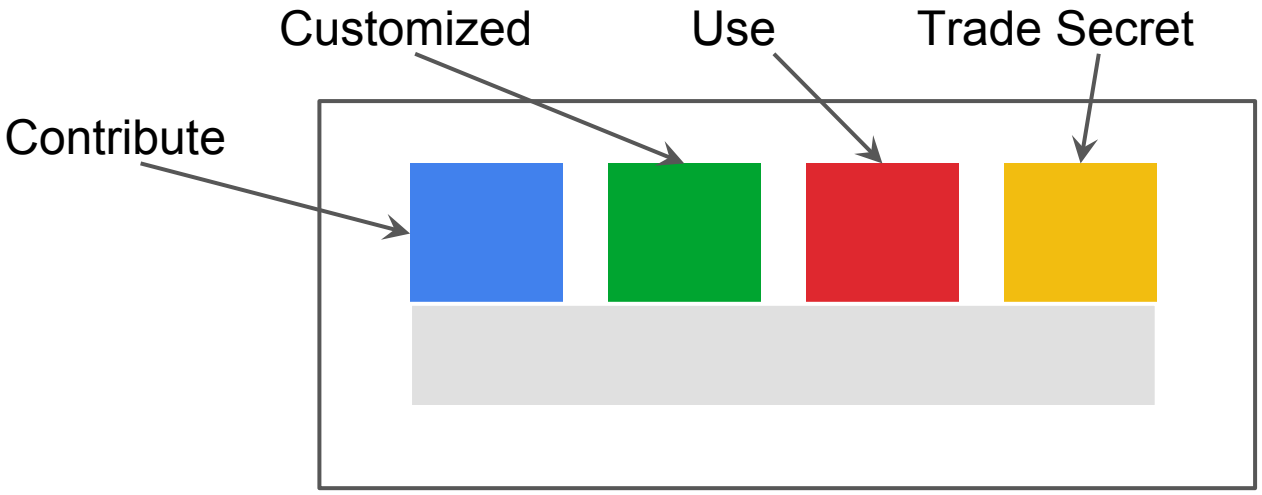
Much better *structure*

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

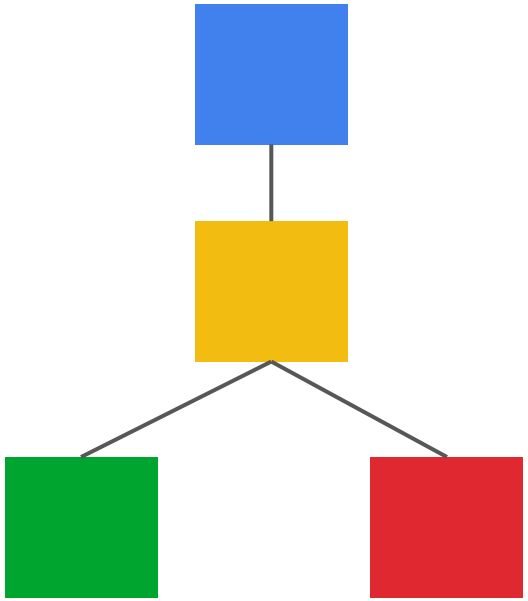
A Quick Look @ Your Code



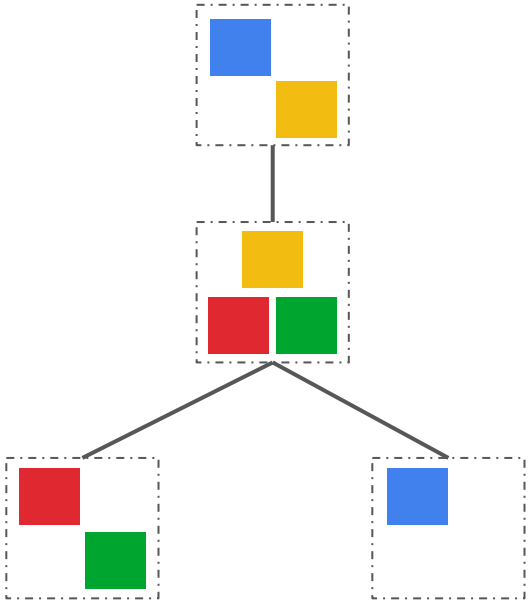
& Your Code Community



SOA… wrong granularity



Kubernetes: sub-structure



Don’t think of a container as the boundary of your application

“A container is more like a class in an object-oriented language.”

--- Google’s Brendan Burns

Requirements...

Sharing among containers

- Share namespaces (esp. PID, Network and IPC)
- Share filesystems
- (Often) Share a resource hierarchy

Requirements...

Atomic co-scheduling of containers

- Composition requires co-location

Requirements...

Parameterization of containers

- Configurable at runtime
- Documented and discoverable

Google

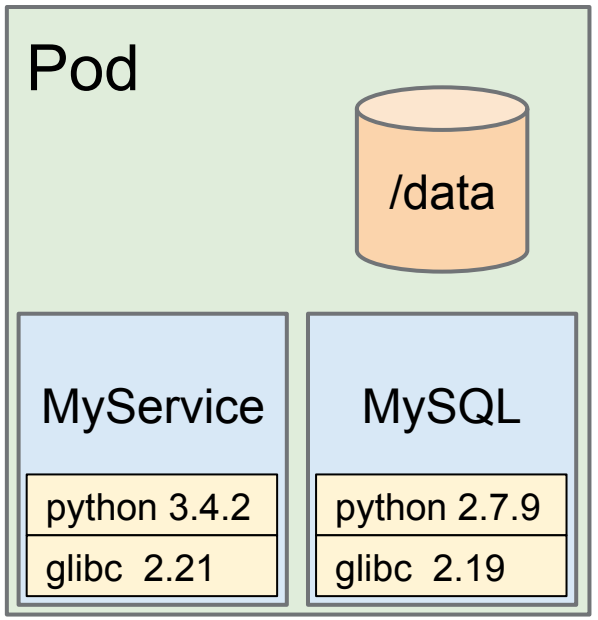
Substructure

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

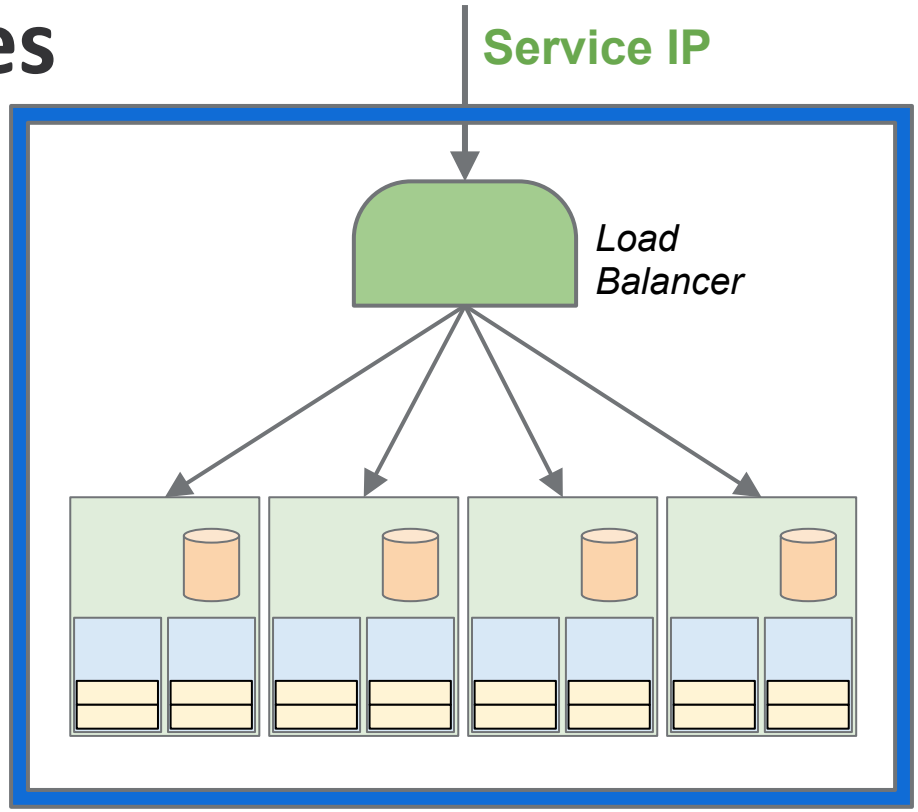


Google

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

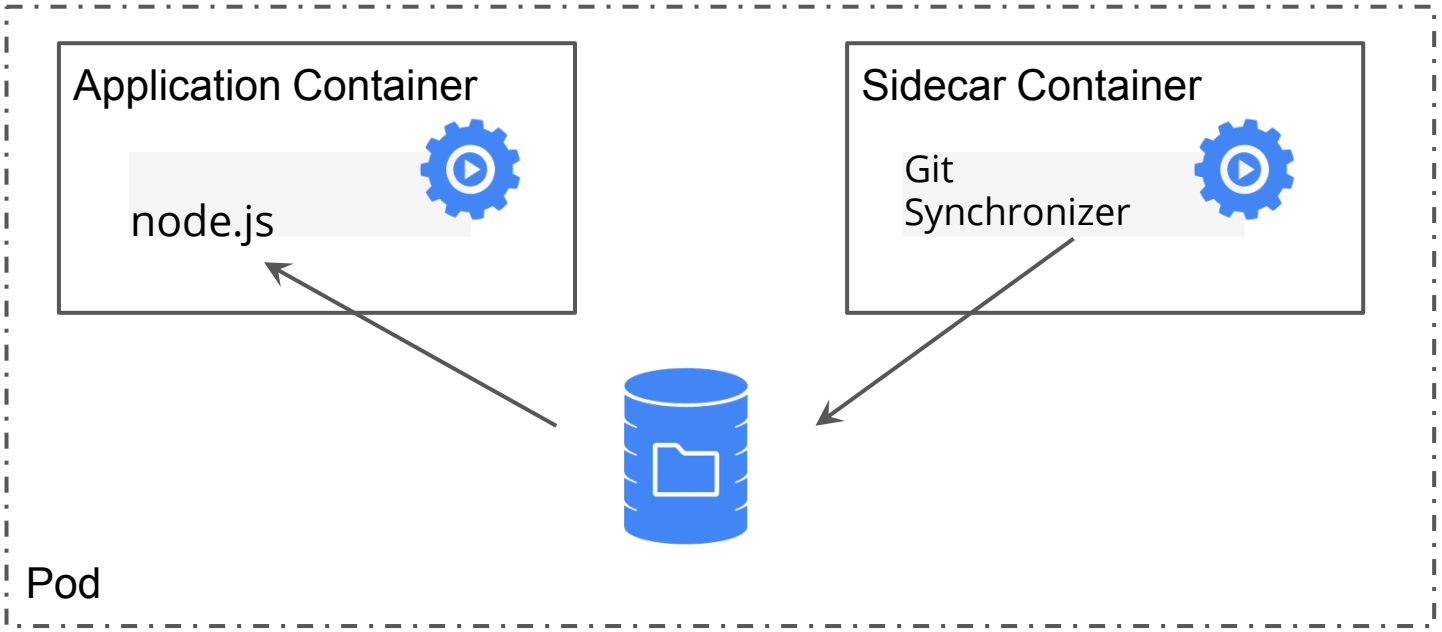


Some Patterns...

Examples of how you use substructure

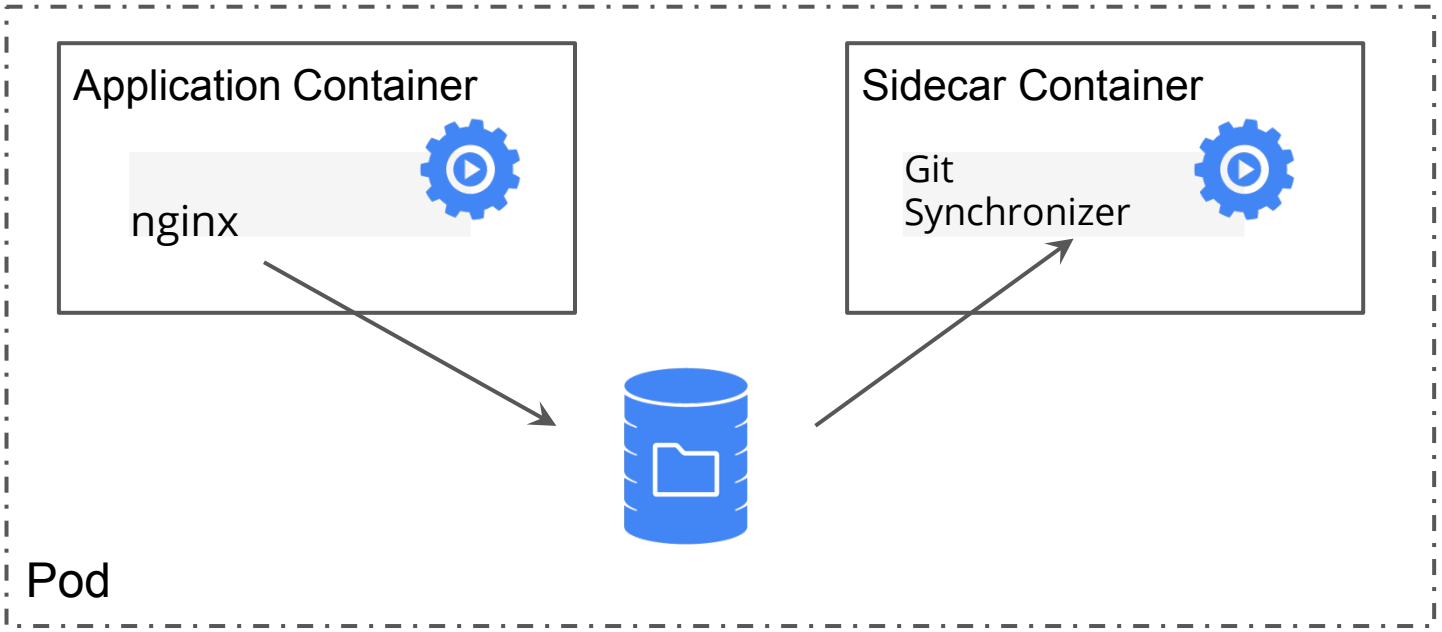
Sidecars

Sidecars extend and enhance



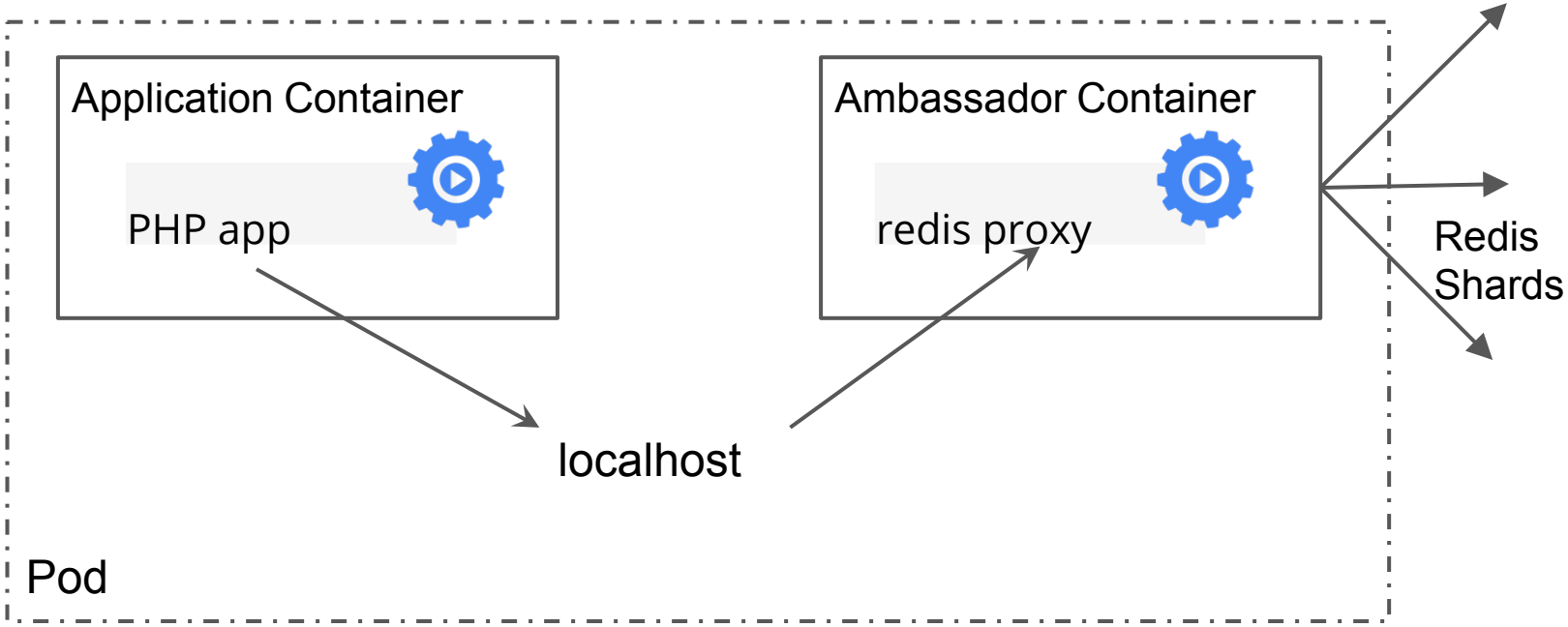
Sidecar Pattern

Sidecars extend and enhance



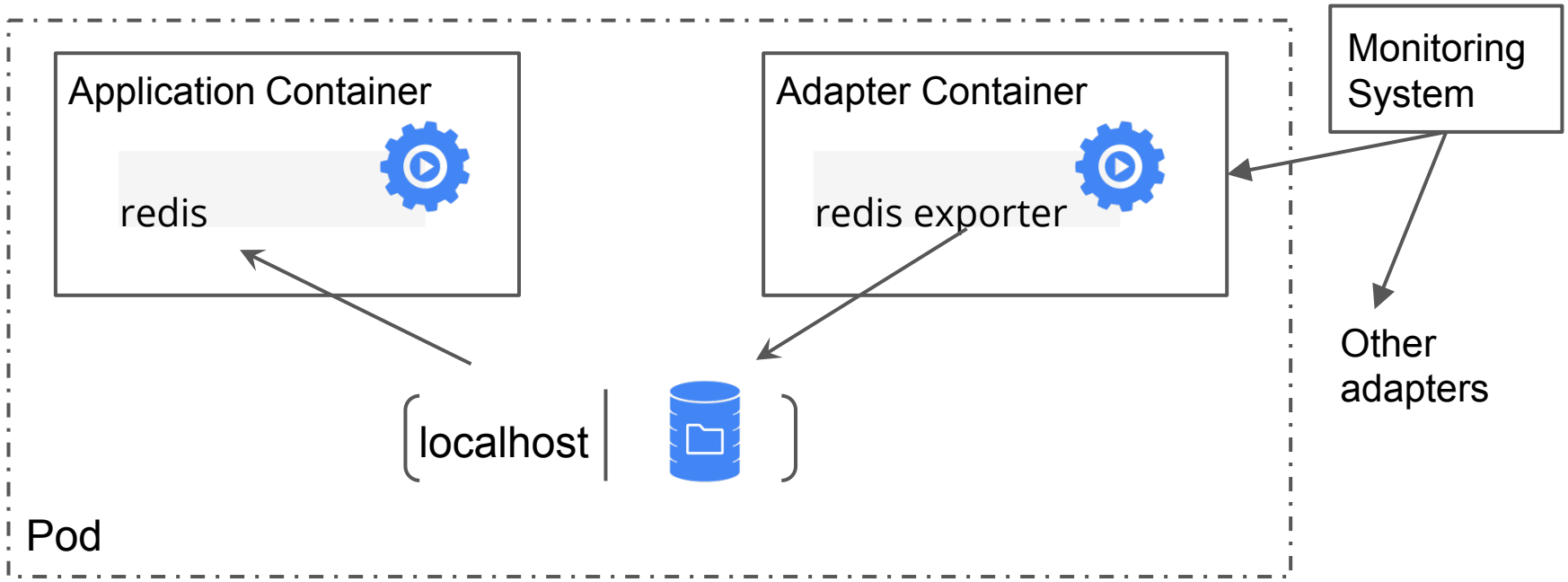
Ambassador Pattern

Ambassadors represent and present



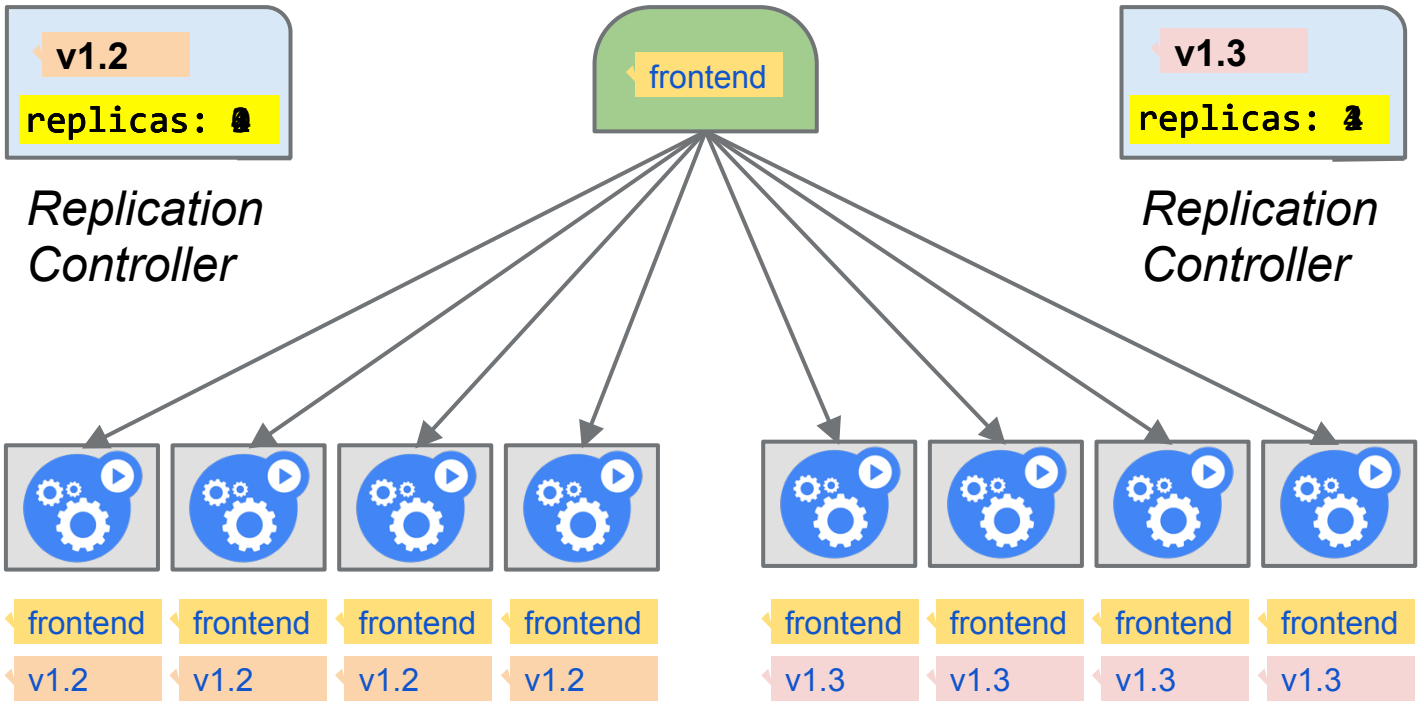
Adapter Pattern

Adapters normalize and abstract



Google

Example: Rolling Upgrade with Labels



Google

Summary

A new path for Cloud Native applications:

- Collection of independent (micro) services
- Each service evolves on its own
  - Scale as needed
  - Update as needed
  - Mix versions as needed
- Pods provide critical structure
  - Template for service members
  - Group containers and volumes
  - Dedicated IP and thus port space
- Containers are the new “classes”

Google

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