Beyond containers:

The value of clusters and microservices

Craig McLuckie

Group Product Manager, Google Corporation

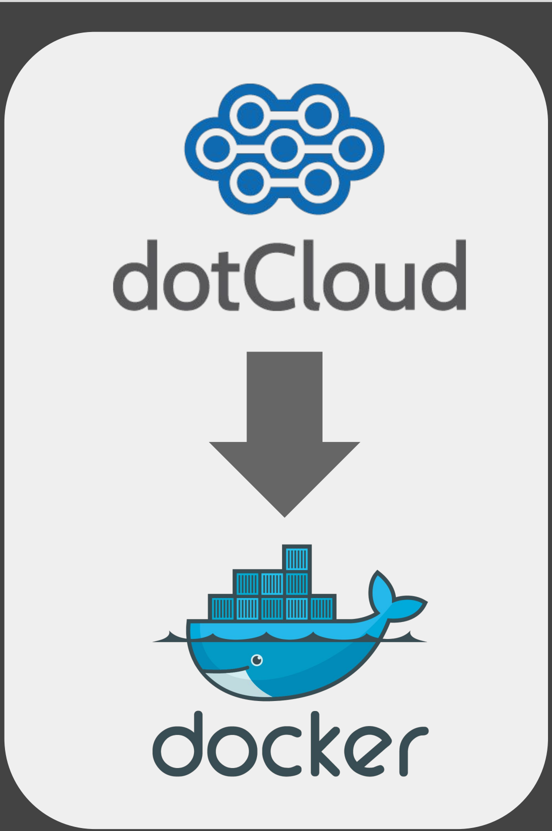
Founder, Kubernetes Project

Chairman, Cloud Native Computing Foundation

History of Containers

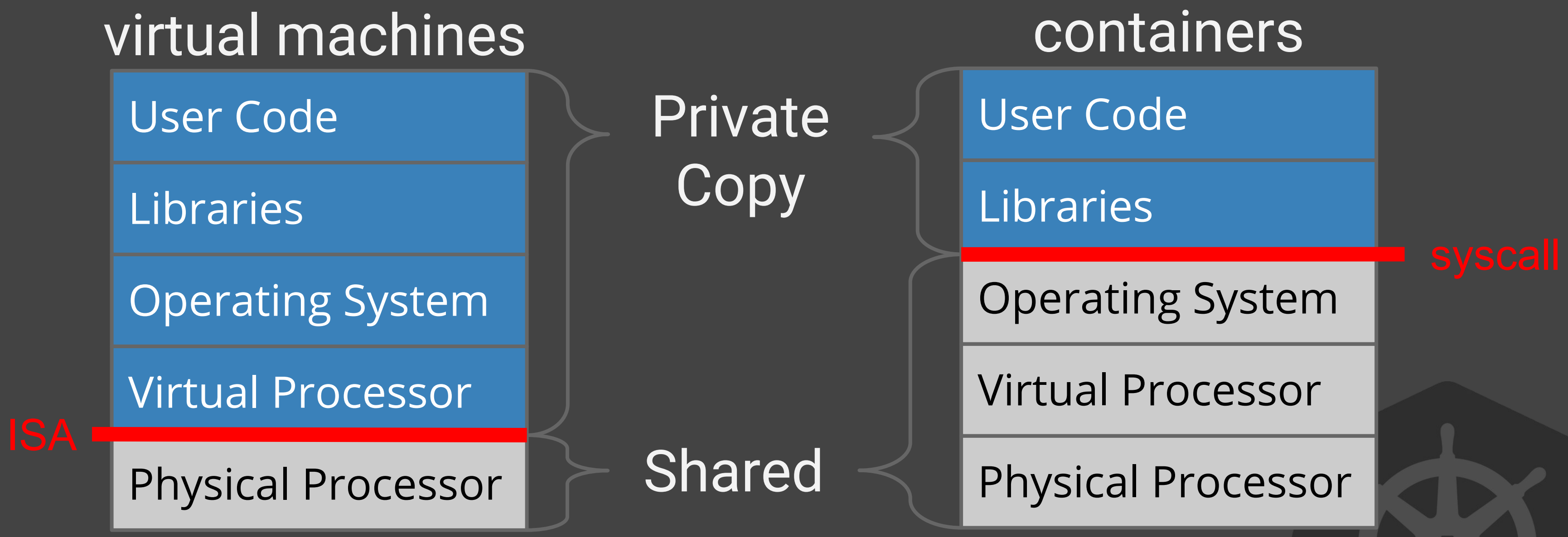
- Control Groups (cgroups)
  - Support efficient resource isolation for process groups
  - Contributed to Linux by Google (2006)
- Namespaces
  - Isolates global system resources
  - Supports OS level virtualization
- LXC
  - Lightweight userland interface to Linux kernel container features
  - Offers isolation and namespace support using cgroups
  - Barebones, but stable toolset
  - Early use was in PaaS platforms -- efficient multi-tenant scheduling (somewhat weak security)

Docker



- Docker wraps everything your code needs to run in a complete filesystem.
- Docker is magic in three ways:
  - Real portability: Syscall layer is same everywhere
  - High degree of reuse and extensibility: Stackable file system
  - Easy to use: Simple and accessible tooling

Container vs VM



containers: less overhead, enable more “magic”

So what remains?

- Development is a big part of the story, but what about operations?
  - Can we make better use of resources?
  - Can we reduce the cost of operations?
  - Can we build more flexible systems?
- Can we make it easier to build and run distributed systems?

Deploy to cluster not machine

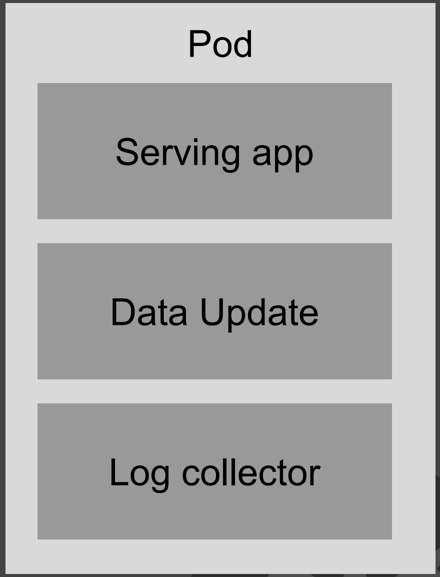
- A cluster is a collection of machines managed as a single unit
- In a cluster...
  - a microservice is an atom of software management and consumption
  - deployment is managed for you
  - your application cannot be affected by the health of a single machine

Kubernetes

- An Open Source technology started by Google
- Links together a number of virtual or physical machines to create a cluster
- Fills the operations gap for containers
  - Deploy container
  - Scale and manage health of containers
  - Connect containers to network
  - Attach storage
- Creates programmable ‘logical infrastructure’

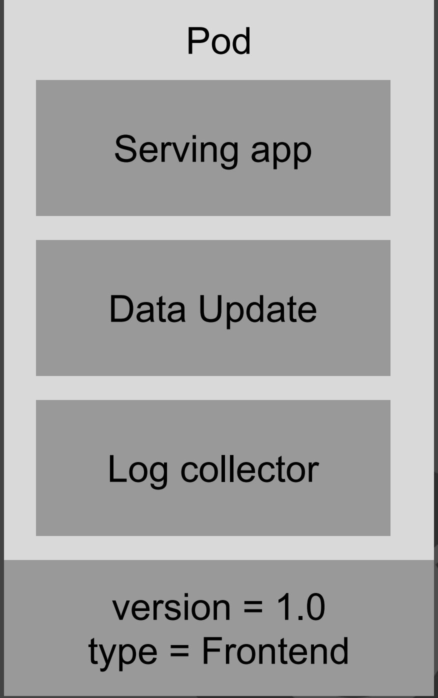
Kubernetes pod

- Unit of deployment
- One or more containers
- Shared fate
- Mapped dynamically to a host node
- Assigned a network interface
- Assigned volume(s)



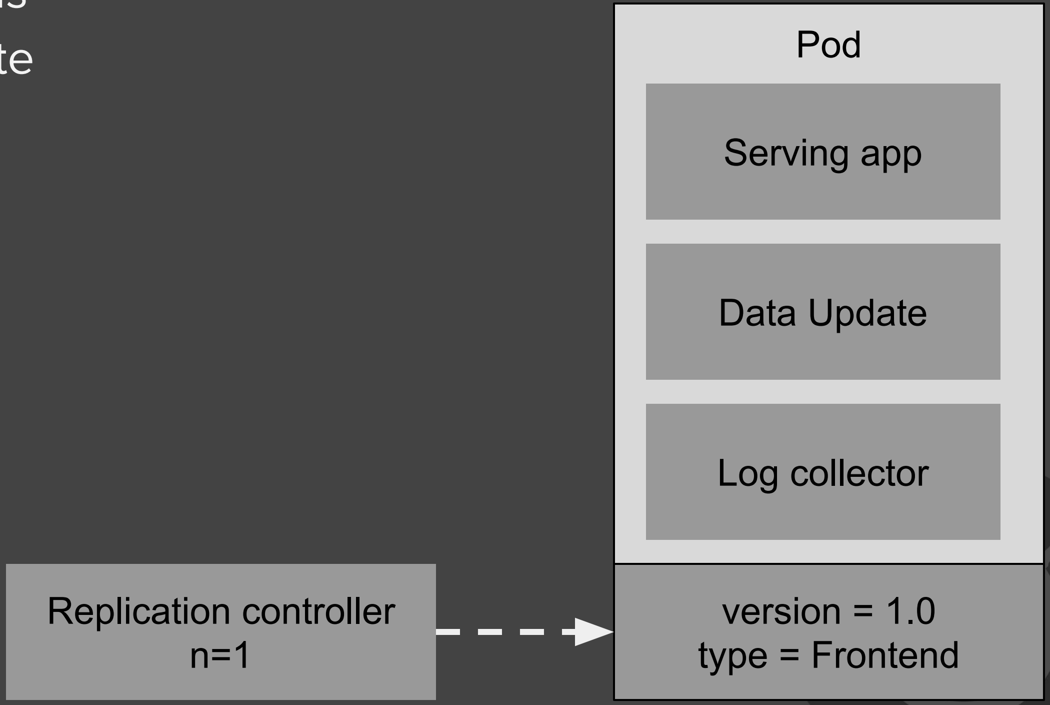
Kubernetes labels

- Label anything
- Name-value pair
- Create your own
- Membership determined via a label selector



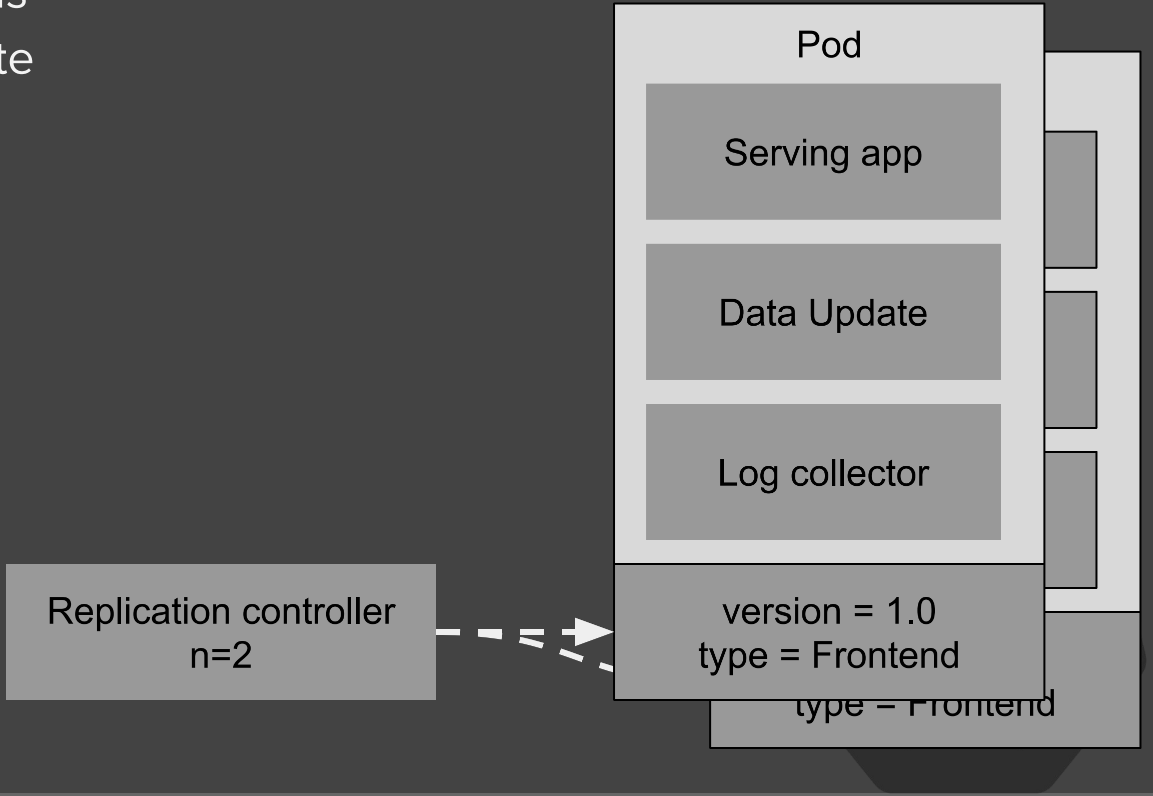
Replication controller

- Manage the lifecycle of pods
- Drive system to desired state
- If too few add pods
- If too many kill pods



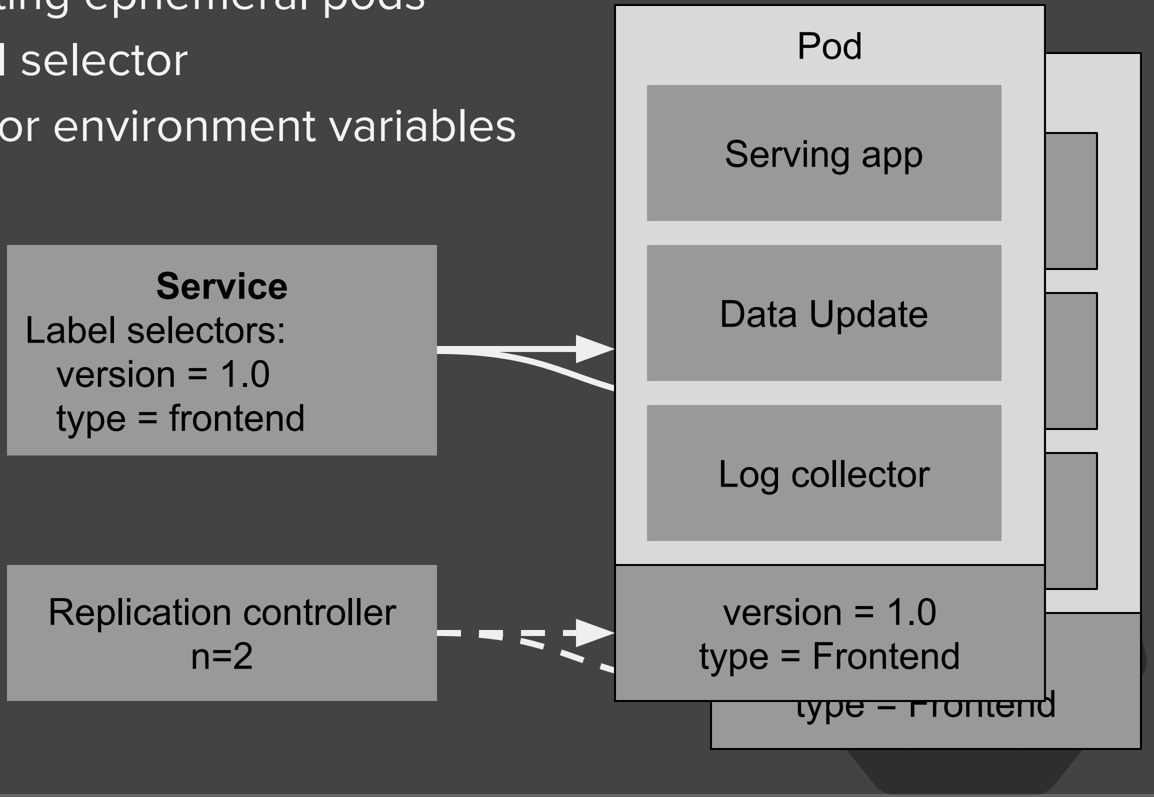
Replication controller

- Manage the lifecycle of pods
- Drive system to desired state
- If too few add pods
- If too many kill pods



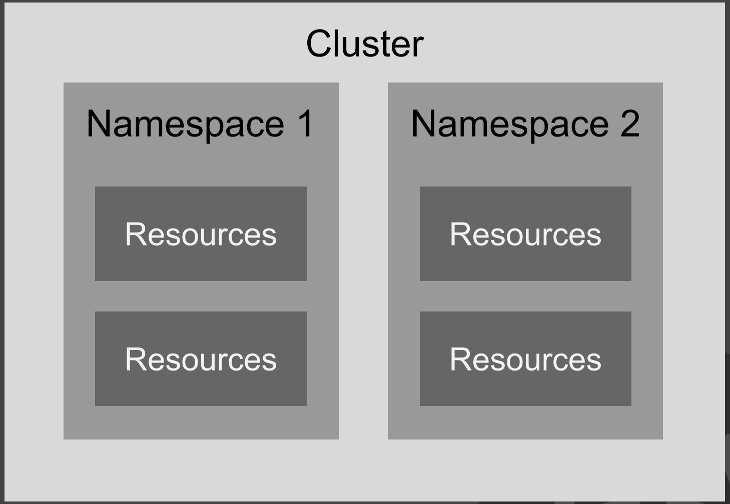
Kubernetes service

- Durable endpoint representing ephemeral pods
- Membership driven by label selector
- Discoverable through DNS or environment variables



Kubernetes namespaces

- Allow teams/services to be isolated
- Enable reuse of configuration
- Logical partition for quota and authZ



Kubernetes services and operations

- Label selector gives programmatic control over membership
- Service is a minimum atom of software consumption in a cluster
  - Minimally (hostname, port)
  - Optionally services can consume Endpoint API to understand mico-service membership
- Natural path to richer orchestration
  - Route based on label selector
  - Blue/Green
  - Canary
- Service offers scoping options
  - Cluster level common services
  - Namespace scope services
- ‘Mix in’ external services by creating a Service object without a selector
  - ‘Native’ consumption experience

A look at operations today

SysAdmins

Developers

- Code

Sys admin and release managers

- Install and configure

Granularity of operation**: Ticket**

+ More predictable

+ A first layer of specialization

+ Auditable, programmatic processes

- Slow

- Scales sub-linearly

- Human operator

A look at operations today

DevOps

Developers can code, right?

- Business logic in language A
- Deployment logic in language B

Granularity of operation: **Integration**

+ Hardware ops goes away
+ Imperative repeatable process
+ Faster deployments

- Doubling the necessary skillset
- Running imperative code in prod
- Effort scales ~linearly
- Loss of global control/visibility

A look at operations tomorrow

Cloud Native Ops

Systems rely common services and specialized teams

- Describe and package app
- Programmatic infrastructure
- Policy based on logical units
- Predictable common services

Granularity of operation: **API call**

+ Scales efficiently
+ Tools align with teams
+ Perfectly predictable and repeatable

- Requires cultural change
- Nascent: Still missing some tools
- Nascent: Missing stable interfaces

Common services

- Application services
  - Implementation detail (breaking down the monolith)
  - Shared artifacts, private instance
  - Multi-tenant
    - Private/Soft (you trust/control users)
    - Public/Hard (your users are hostile/idiots)
- Common distributed system services
  - Naming, discovery, quoruming, data sharding, etc generally just part of your cluster

Specialized roles emerge

- Infrastructure Operator
  - Generally your cloud provider
- Cluster and Services Operators
  - **Engineers** (SRE) run clustering technology, and support common cluster services
- Application operator
  - **Engineers** (SRE) run applications
- Application developer

This sounds like devops, and it could be. Difference is the tools are better, and there is a clear path to specialization. SREs automate themselves out of a job.

These are “hats” that people wear, not job titles.

An obvious tension

- Running things tend to keep running (mostly)
- Change breaks things
  - Binaries
  - Dependencies
  - Flags
- But the business is never going to sit still
  - (Software is eating the world)
- **Managing down the cost of change** is a critical competitive factor
  - Reduce toil
  - Increase control
  - Improve recoverability
  - Improve velocity

Architecting for change

- Adopting modern CI/CD practices solve a local problem
- Putting shared things behind a stable interface
- Track, tool and automate change
  - Declarative trumps imperative
  - Dead simple updates and rollback
- Remove degrees of freedom as early as possible in the deployment cycle
  - Static binary/image, fully resolved configuration, etc
  - Steer clear of imperative steps in production
  - Minimize deltas between dev, staging, prod
- Run multiple versions of the same thing in production
  - Learn small
  - Shift load carefully
  - Run experiments

Scaling your engineering team

- Remember Conway’s law
- Microservices allow a ‘crew’ to solve a specific problem
- ‘Throwing product over the wall’ → ‘running a service’
- Common services allow focus on just business logic

Driving up operational maturity

Specialization of operations can create truly differentiated operations professionals

1. Service level monitoring
   a. Latency
   b. Traffic
   c. Errors
   d. Saturation
2. Incident response playbooks
3. Blameless post-mortem / RCA
4. Qualification and staged releases
5. Experiments

Community and Cloud Native Computing Foundation

- Kubernetes is not just a Google project
- Kubernetes doesn’t live alone -- it is an important piece of the bigger story
- We need a safe place to collaborate and innovate
- That is why we donated Kubernetes to Cloud Native Computing Foundation

CLOUD NATIVE
COMPUTING FOUNDATION