Building a Microservices Platform with Kubernetes

Matthew Mark Miller
@DataMiller

Cloud Native:

Microservices running inside
Containers on top of
Platforms on any infrastructure

Microservice

A software component of a system that is independently releasable and independently scalable from other parts of the system.

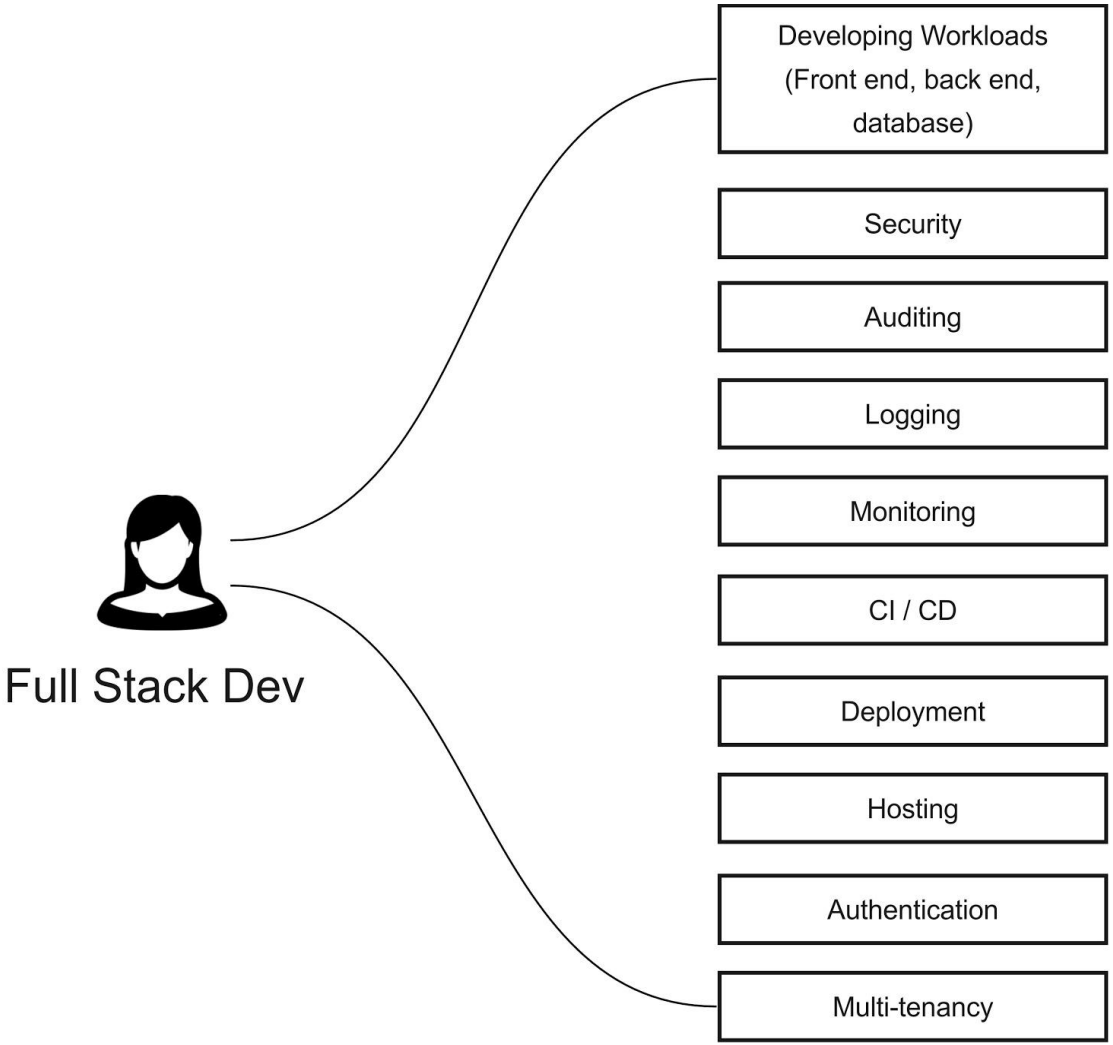
Container

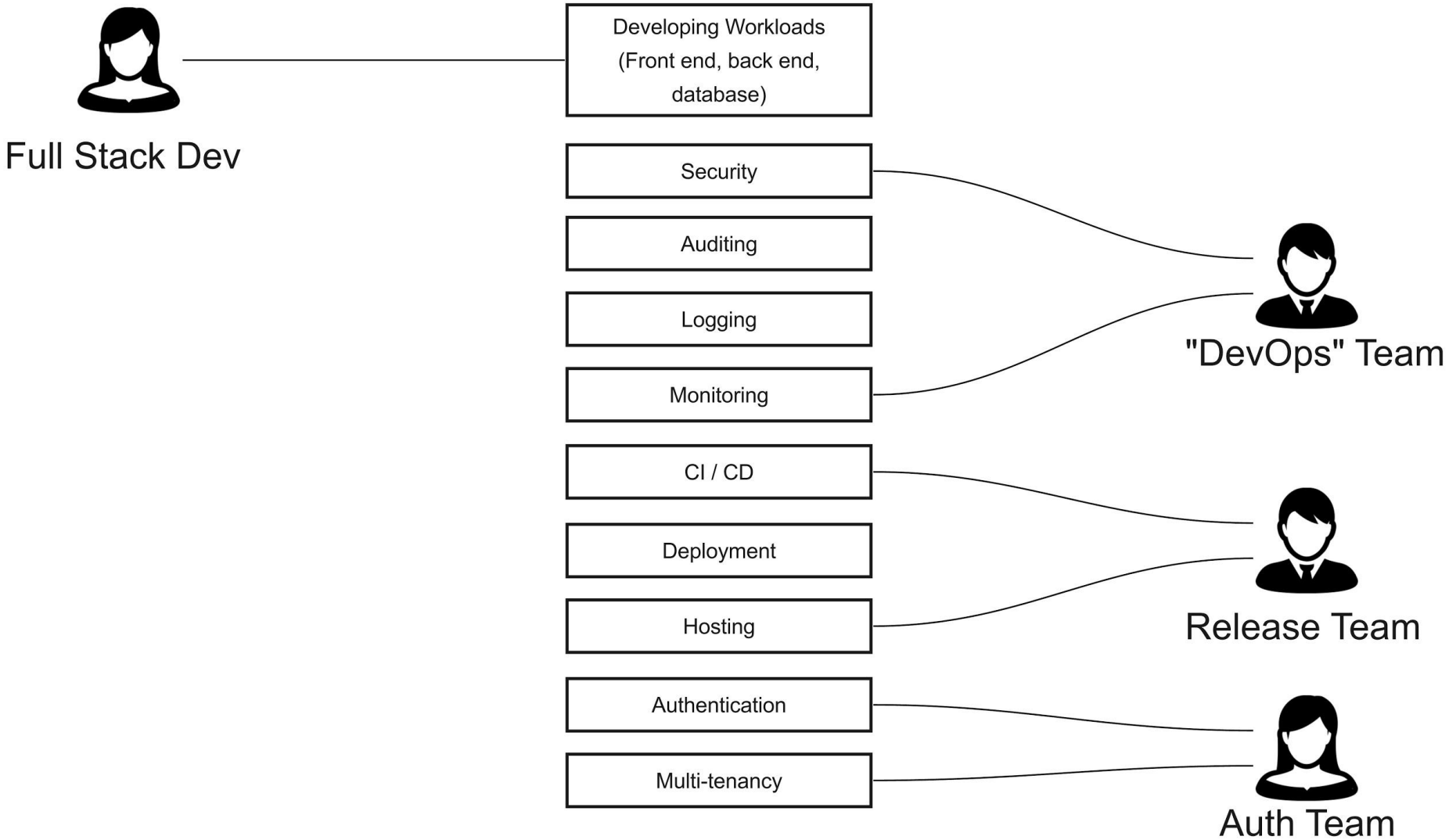
A software process whose access has been reduced to the point that it thinks it is the only thing running.

Platform

The parts of your service that you don't build yourself.

But wait...aren’t we supposed to be Full Stack?!


Full Stack Dev
Developing Workloads (Front end, back end, database)
Security
Auditing
Logging
Monitoring
CI / CD
Deployment
Hosting
Authentication
Multi-tenancy

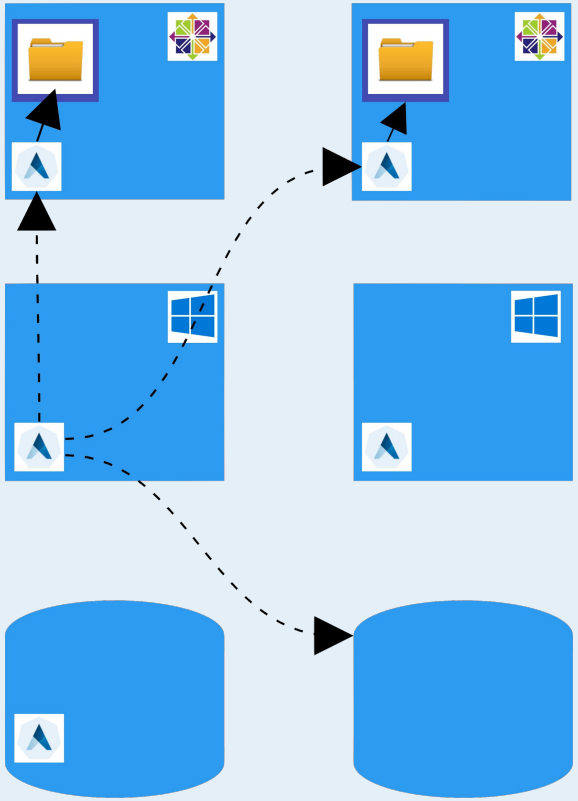

Full Stack Dev
Developing Workloads (Front end, back end, database)
Security
Auditing
Logging
Monitoring
CI / CD
Deployment
Hosting
Authentication
Multi-tenancy
"DevOps" Team
Release Team
Auth Team

A platform’s responsibility is to make implicit the link between a service and the resource it consumes.

APPRENDA

workload.war

APPRENDA



Clouds operate because of workload orchestration

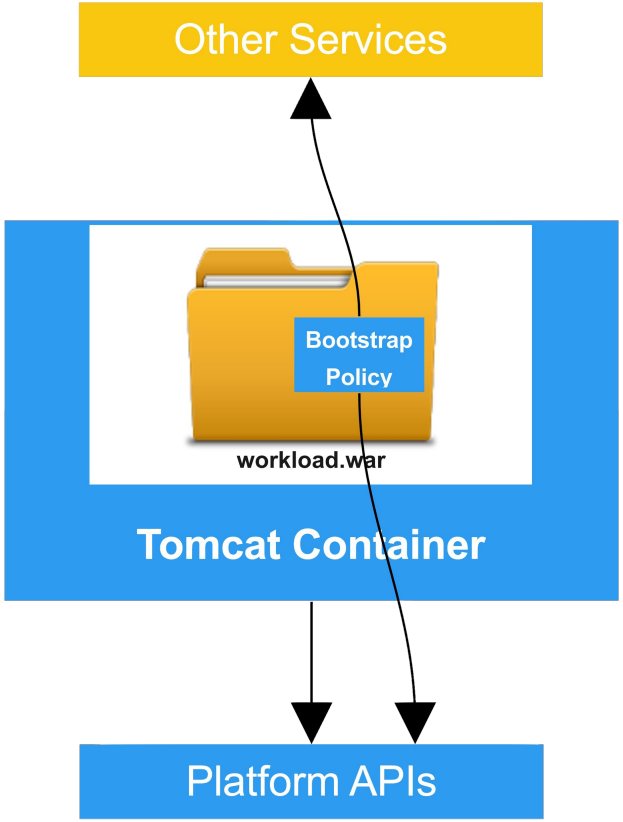
Don’t roll your own orchestration.

Integrating workloads requires tinkering at runtime

Token swapping

Modifications to the host container's configuration

Swapping in binaries

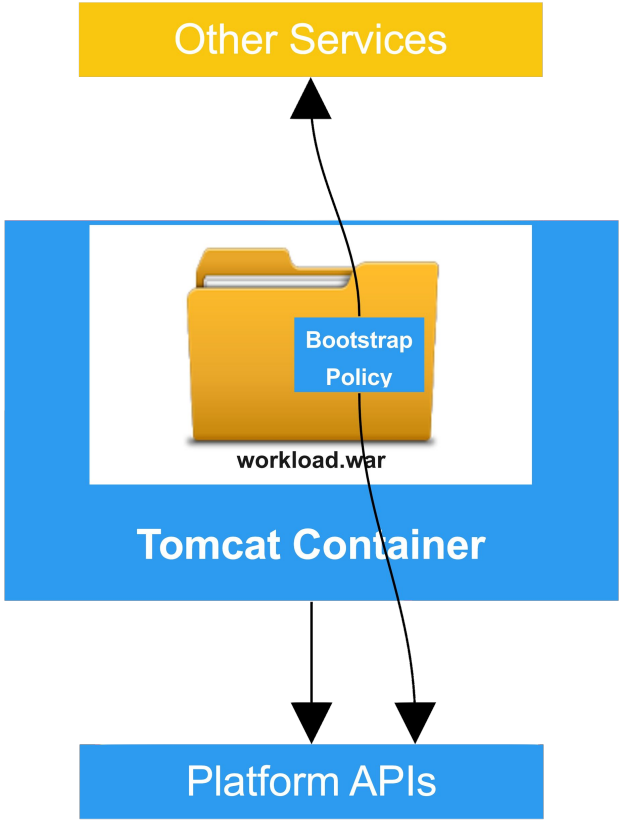


Integrating this way isn’t easy

Takes time & testing to get it right

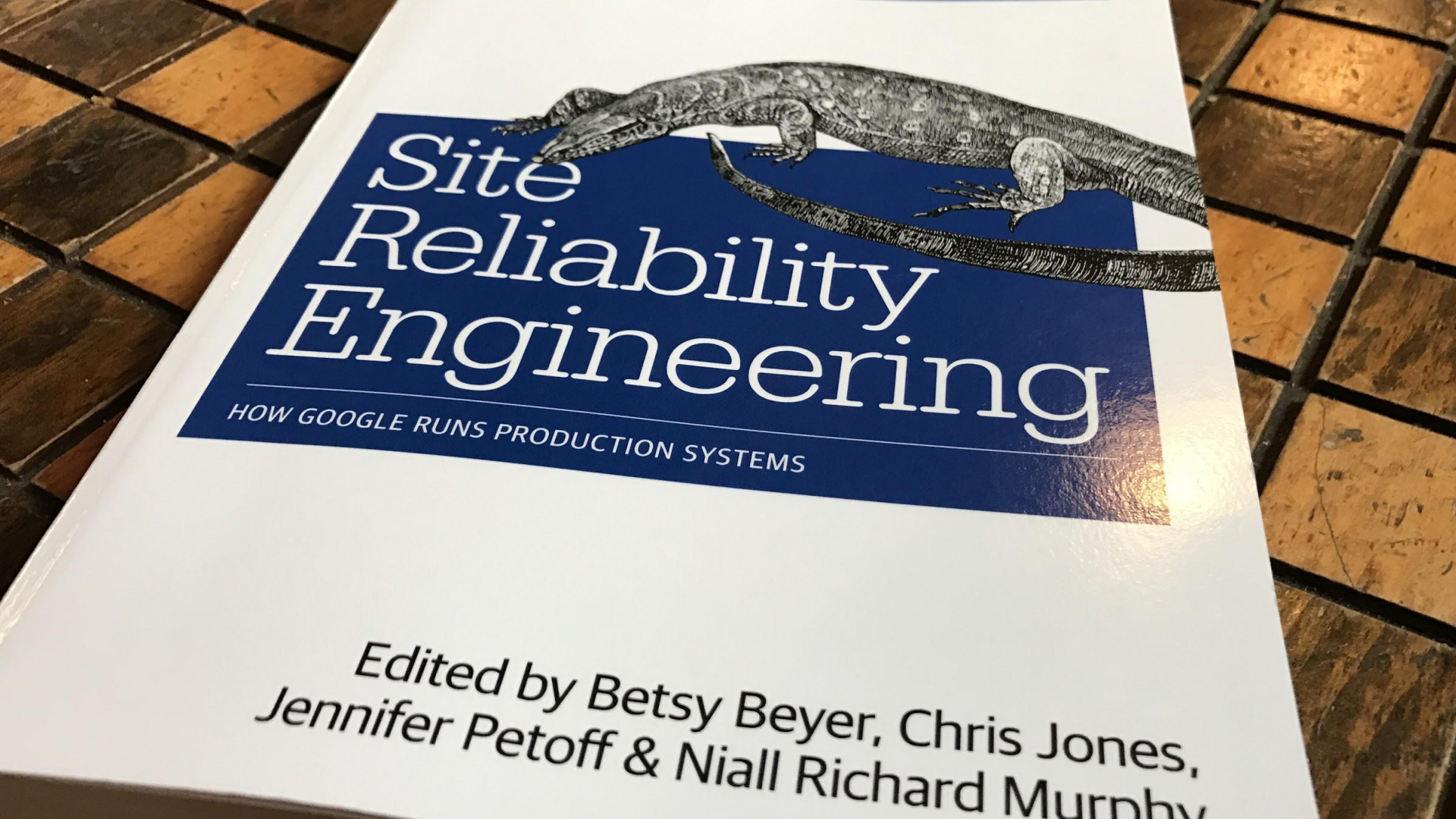
What you built and tested isn’t necessarily what runs in production.

Leads to providers offering fewer, more highly opinionated stacks



A big question for platform engineers:

How can we spend more time building useful services and less time maintaining the platform?


Site Reliability Engineering
HOW GOOGLE RUNS PRODUCTION SYSTEMS
Edited by Betsy Beyer, Chris Jones,
Jennifer Petoff & Niall Richard Murphy

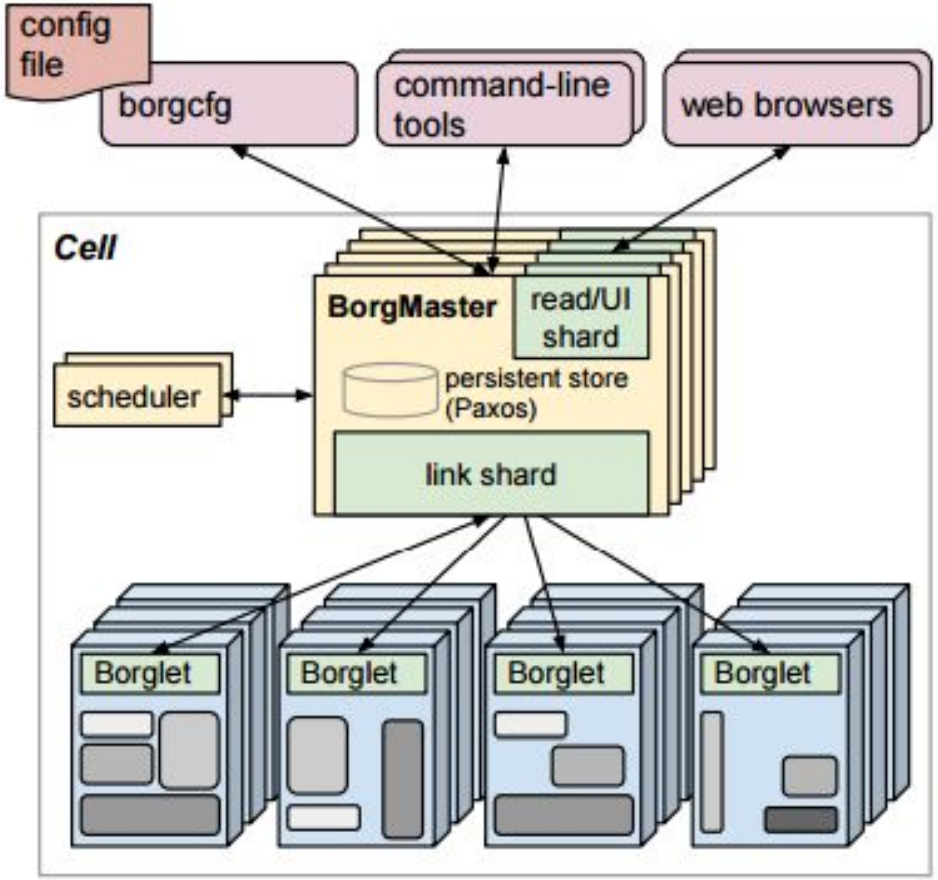
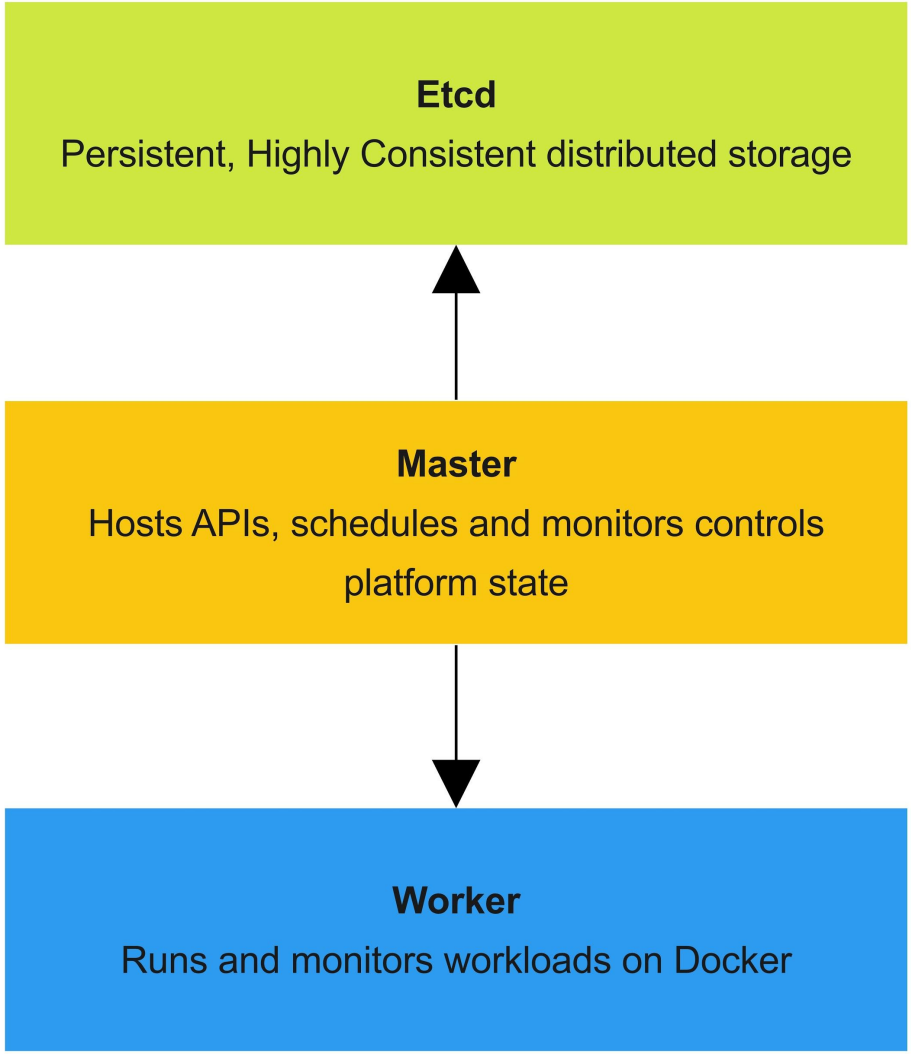


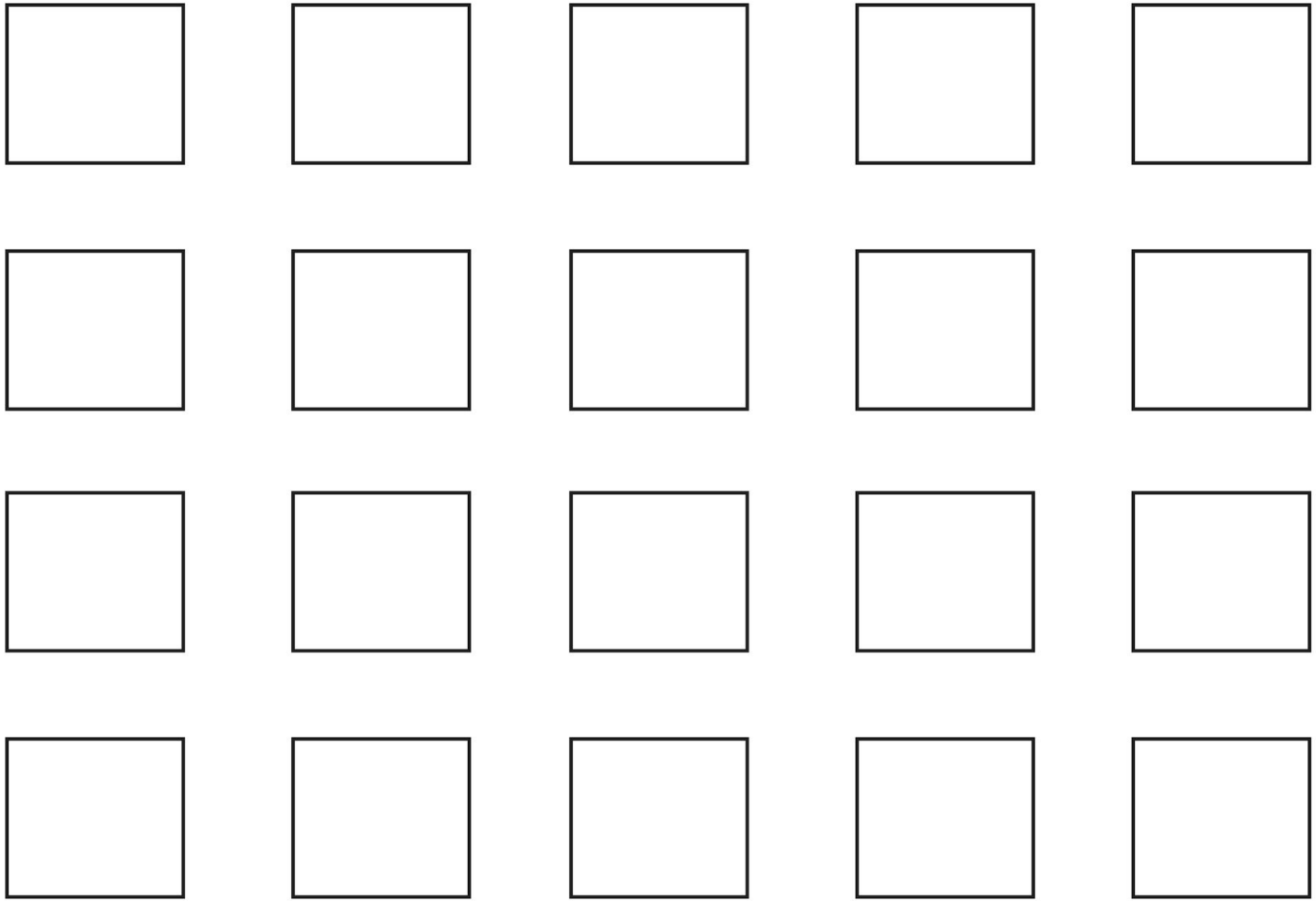
Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

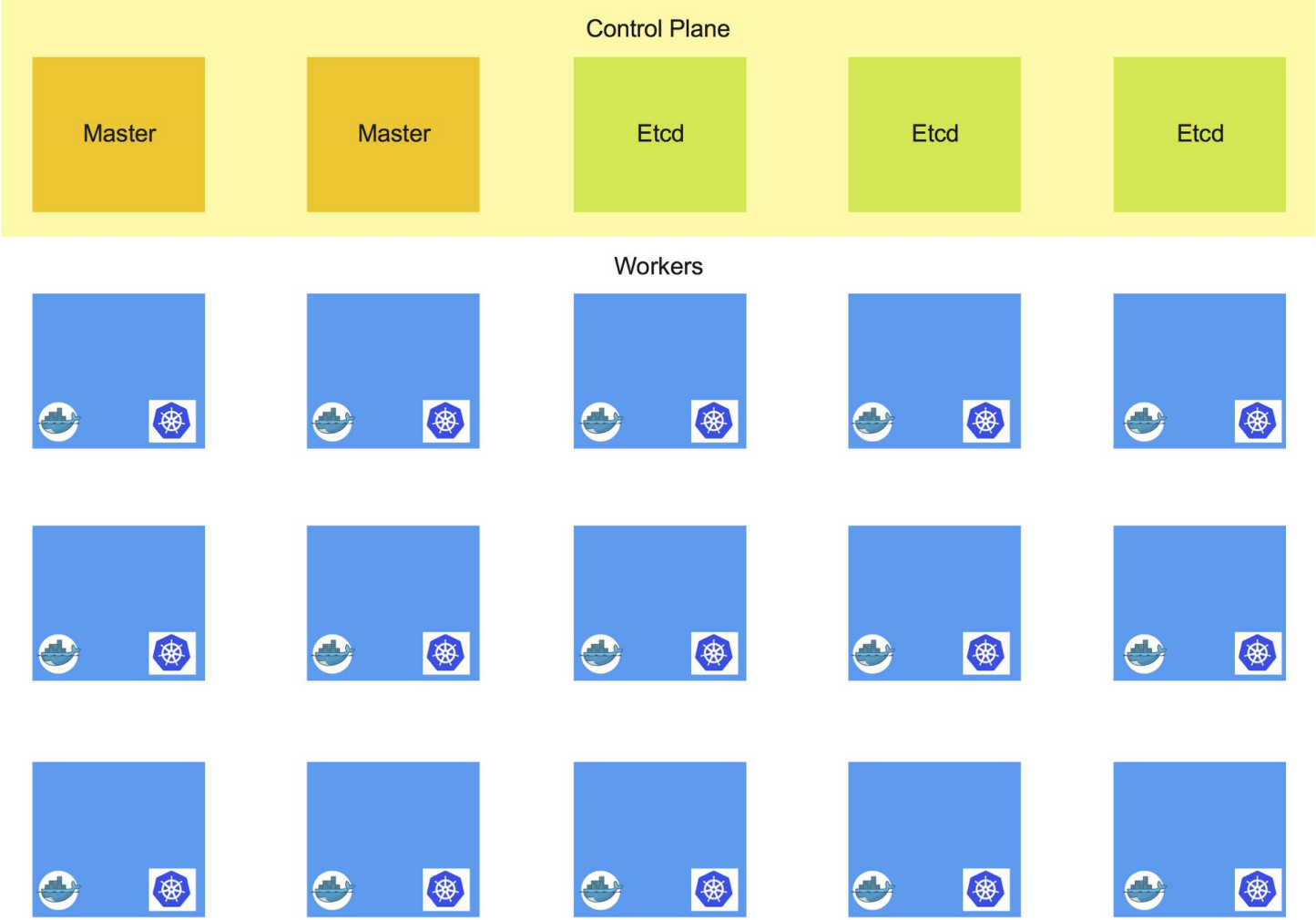
Kubernetes

Borg meets Docker; Resistance is futile

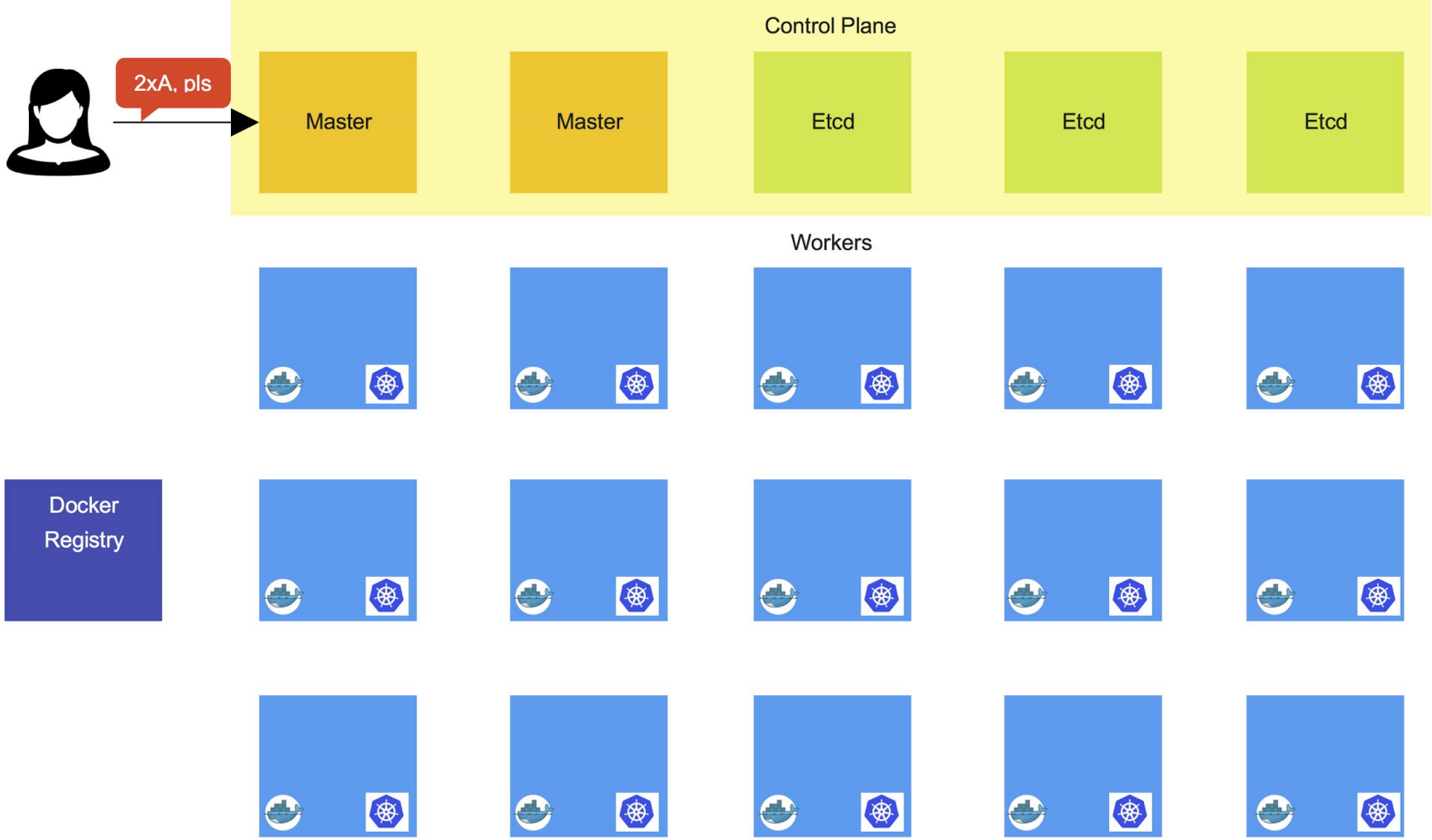

Etcd
Persistent, Highly Consistent distributed storage
Master
Hosts APIs, schedules and monitors controls platform state
Worker
Runs and monitors workloads on Docker

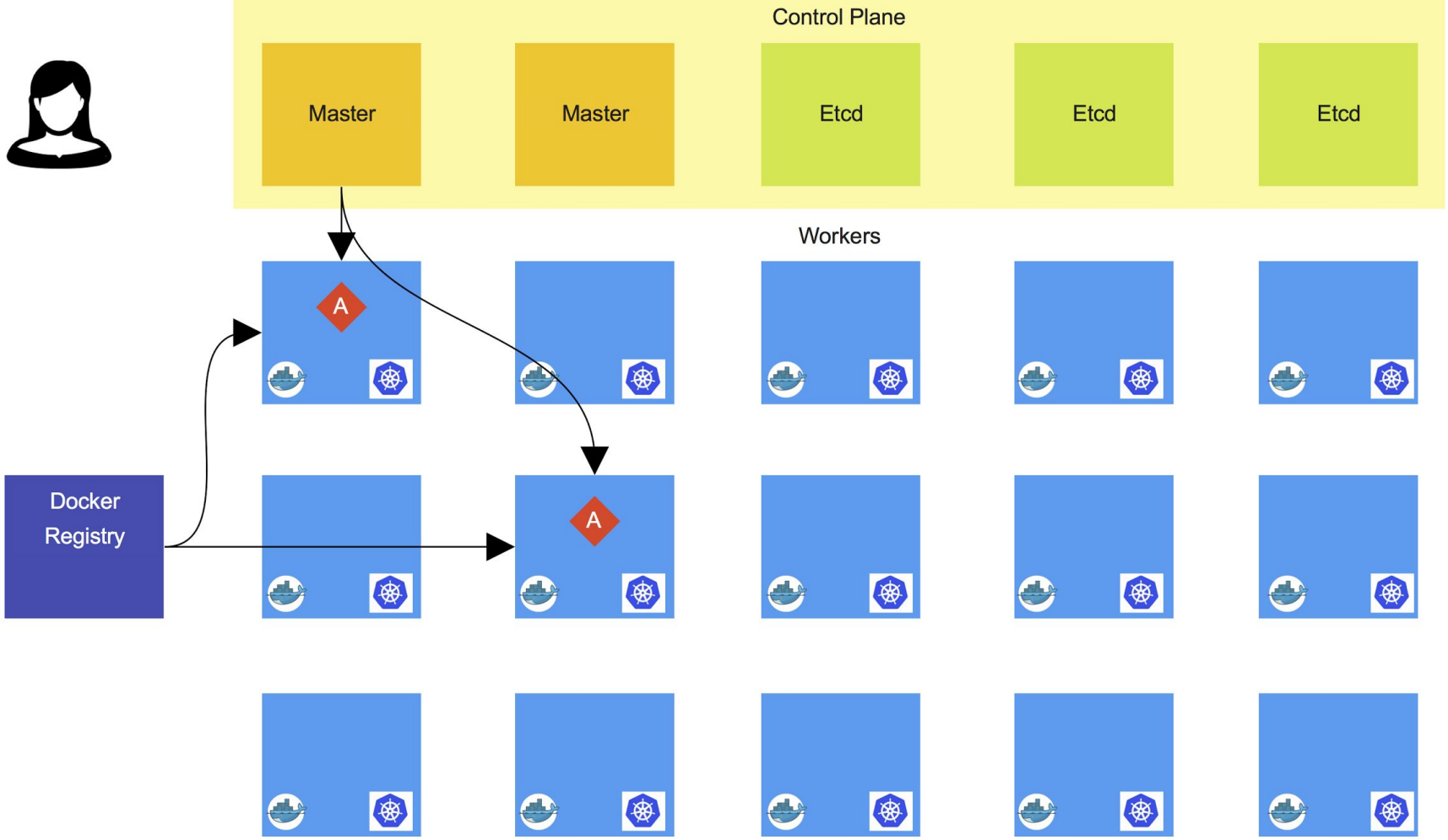
Provisioned machines (called Nodes)




Control Plane
Master
Master
Etcd
Etcd
Etcd
Workers


Control Plane
Master
Master
Etcd
Etcd
Etcd
Workers
A
Docker
Registry


Control Plane
2xA, pls
Master
Master
Etcd
Etcd
Etcd
Workers
Docker Registry


Control Plane
Master
Master
Etcd
Etcd
Etcd
Workers
A
A
Docker
Registry

Kubernetes is popular, open and growing

- Most active project on GitHub - out of over 3.6 million
- Over 7K LinkedIn Professionals
- Hedges against vendor lock-in with largest container management ecosystem

GITHUB		
36,000+ COMMITS	160+ RELEASES	900+ CONTRIBUTORS
Top 100 FORKED GITHUB PROJECT	Top 2 STARRED GO PROJECT	Top 0.01% STARRED GITHUB PROJECT

To those of us building platforms, Kubernetes offers

Reliable cluster & workload management

A stack agnostic hosting abstraction (Docker)

Battle-tested fundamental abstractions that give rise to powerful deployment patterns

Kubernetes Fundamentals

Controllers

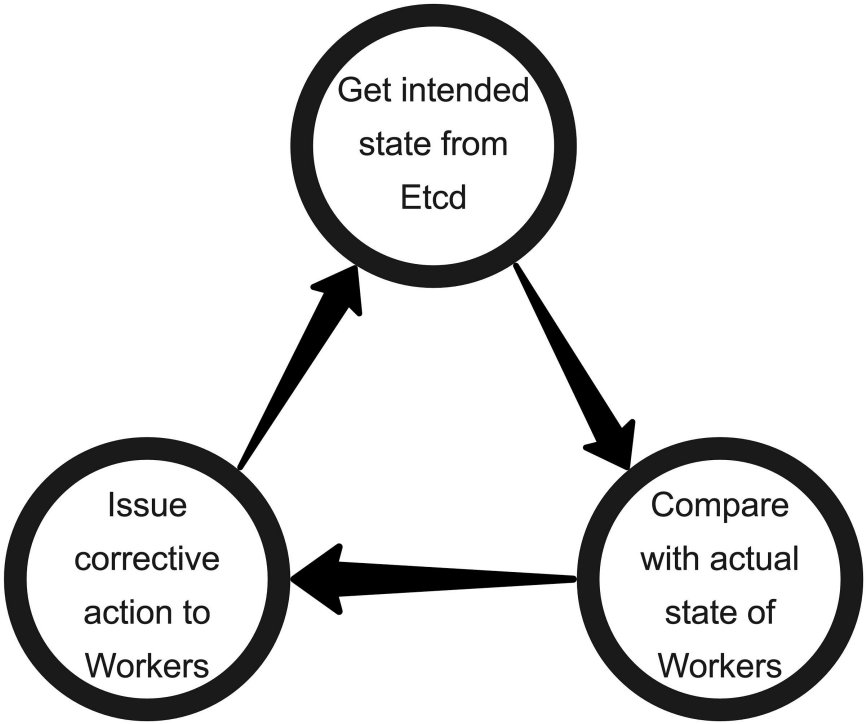
Loops that maintain state

Run continuously on Master

Each Kubernetes object gets its own Controller

Controllers are pluggable & lightweight

Rely on **declarative** manifests to determine intent



The Pod

Many containers, working together as a single unit

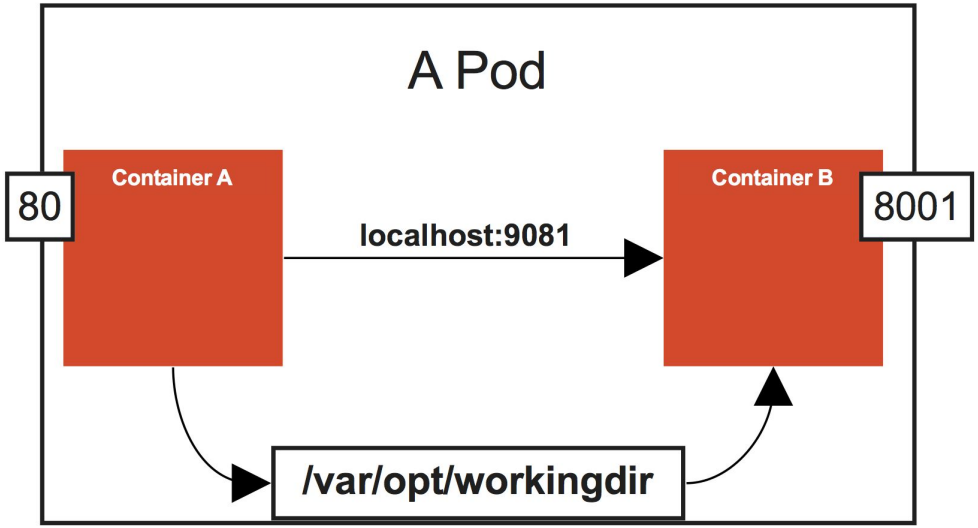
Shared IP & localhost

Shared filesystem

Scale together

Separate hardware limits

Can be tagged with a **label**, providing scheduling advice



Services

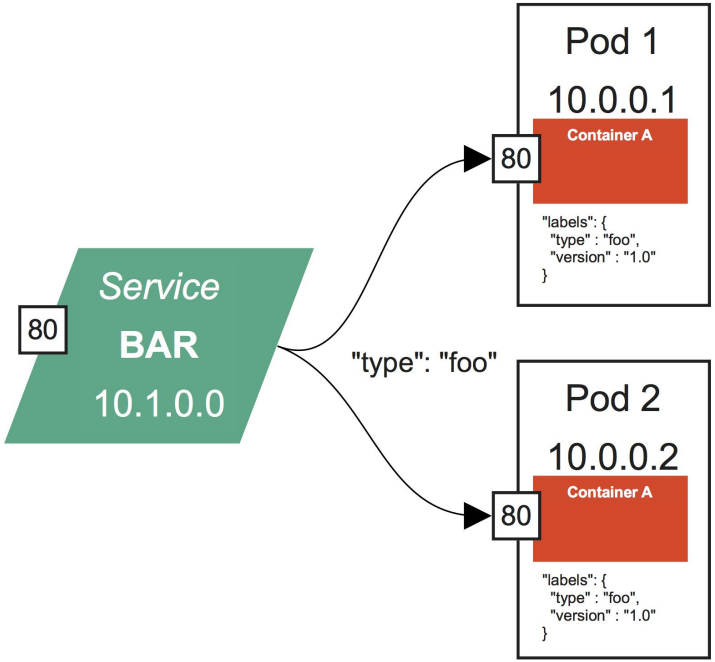
Permanent, logical addresses for internal services

Expose a name, port and stable IP for a group of pods

Load balance between individual pods

Provided to pods via DNS or environment variable

Constructed using a **selector** onto pod **labels** (sort of like a database query)



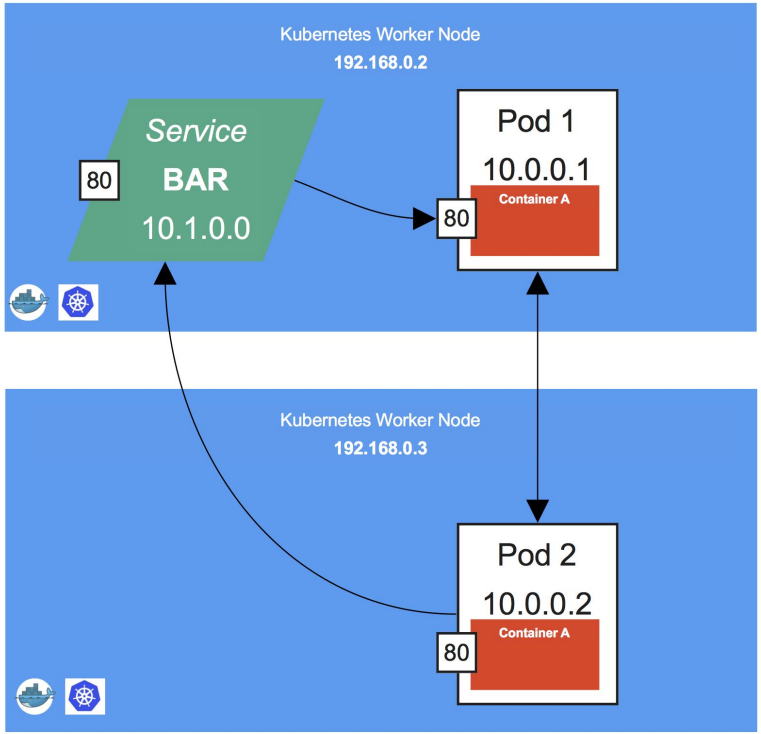
Networking

Rules for all Kubernetes installations

Each Pod gets its own unique IP address (which is the same outside and in)

All Pods must be able to communicate with each other **without NAT**

All Pods must be able to communicate with and participate in Services



Ingress

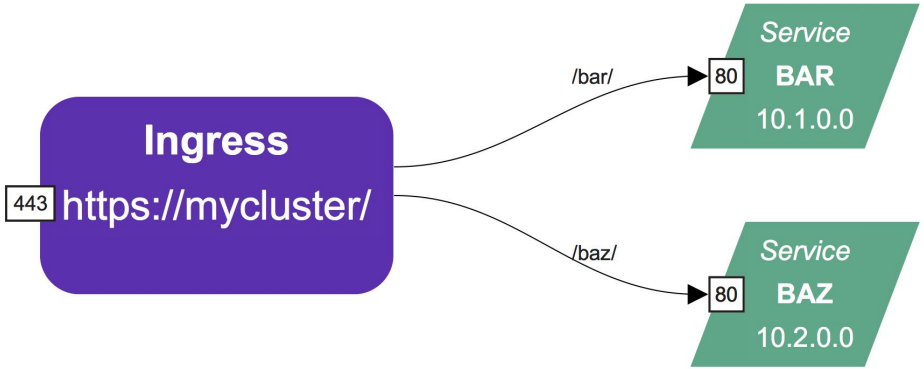
Simplifies Layer 7 access to Kubernetes services

Works with load balancers, including cloud load balancers & nginx

Presents a single root URL mapping to multiple services

Publicly expose private networks

Terminates TLS/SSL

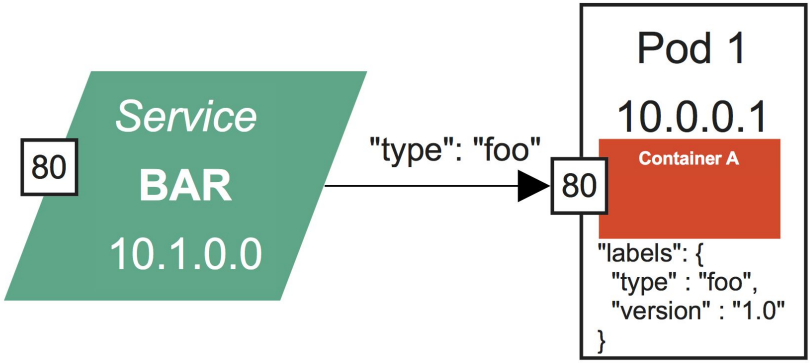


Using the fundamentals to build a platform

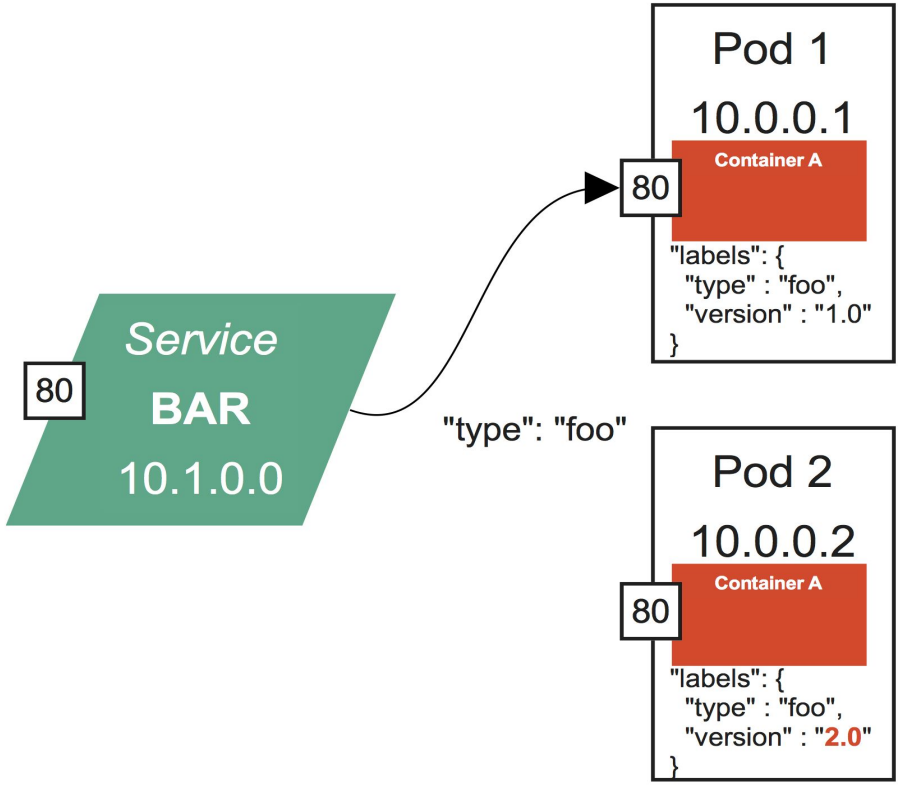
Pod patterns from Burns & Oppenheimer, USENIX 2016

How can my platform provide availability during workload releases?

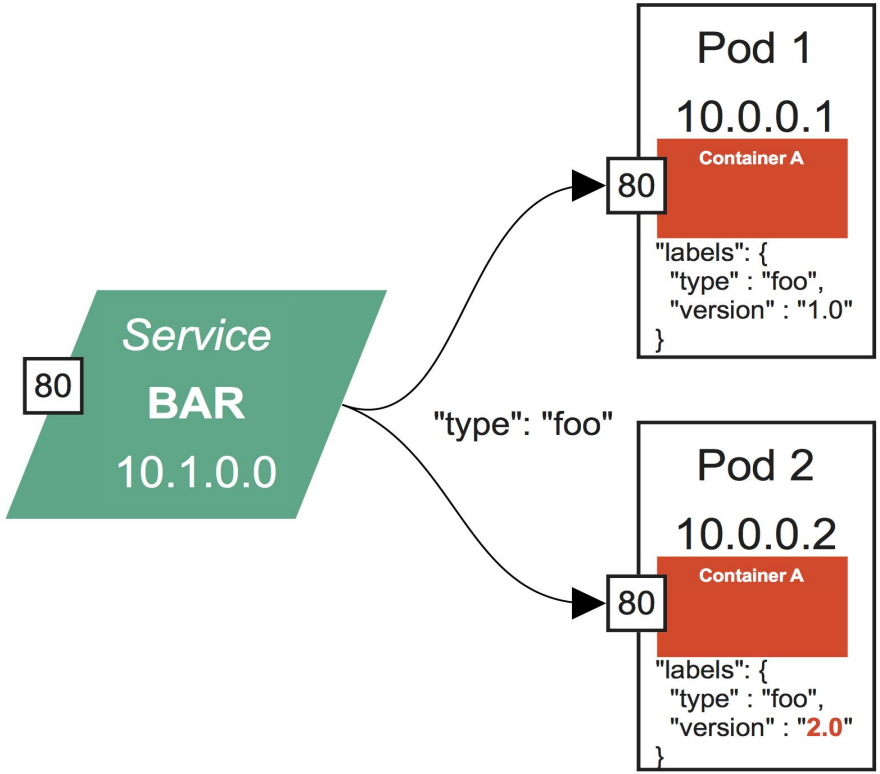
Rolling Deployments



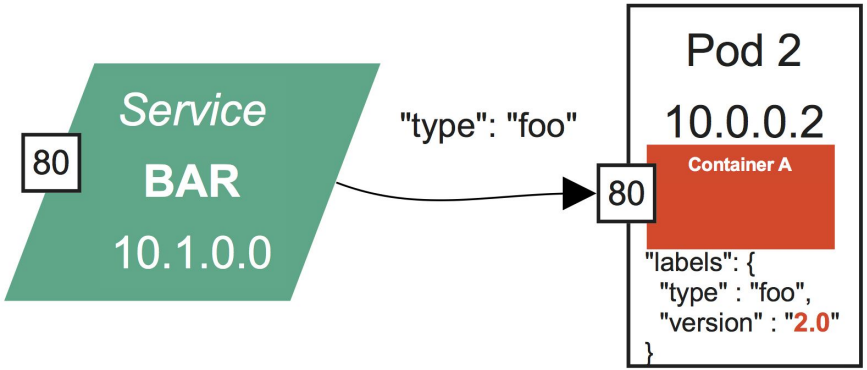
Rolling Deployments



Rolling Deployments

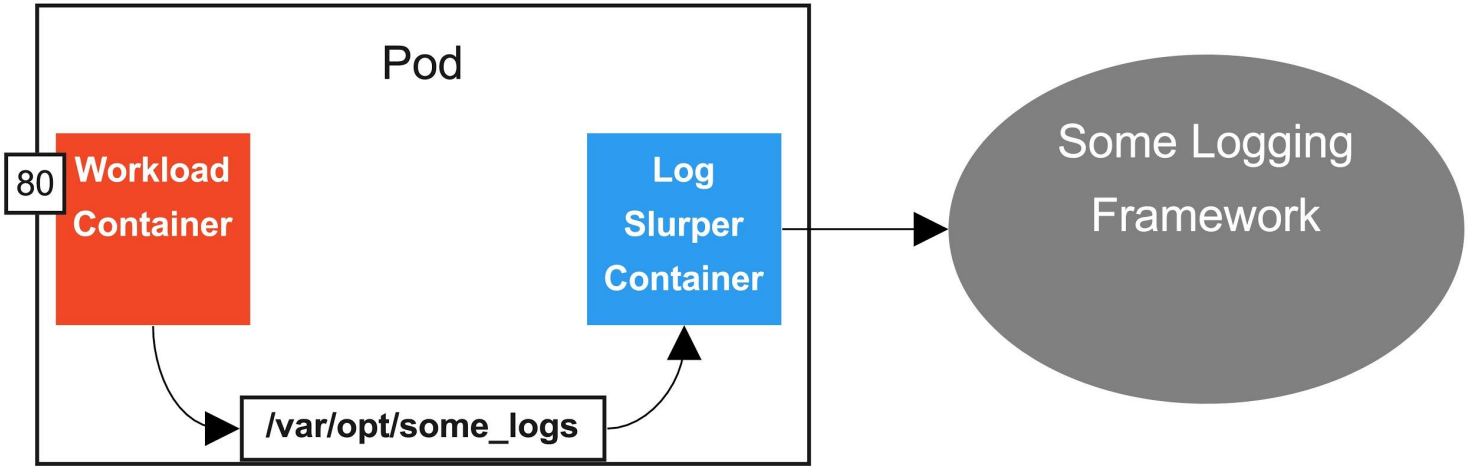


Rolling Deployments



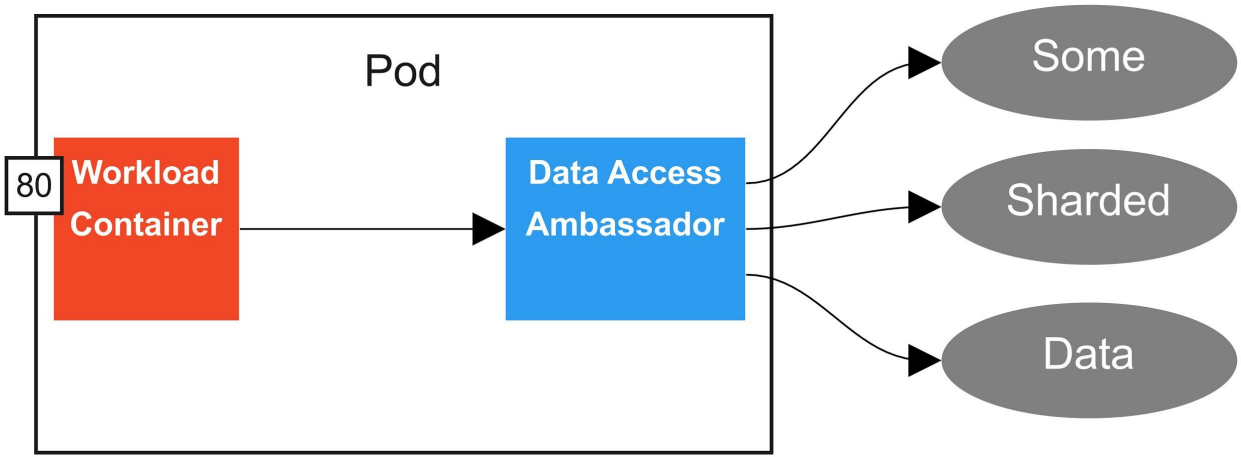
How can my platform non-destructively add functionality to a workload?

Sidecars


Pod
80
Workload Container
Log Slurper Container
/var/opt/some_logs
Some Logging Framework

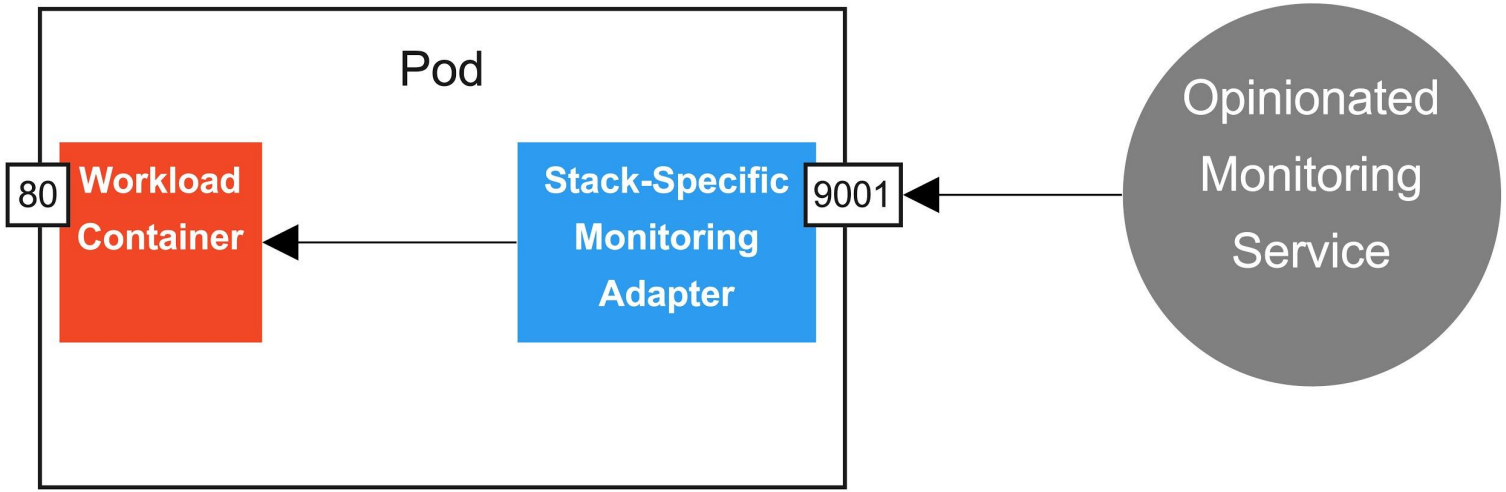
How can my platform insulate workloads from complexity and state of services?

Ambassador


Pod
80
Workload Container
Data Access Ambassador
Some
Sharded
Data

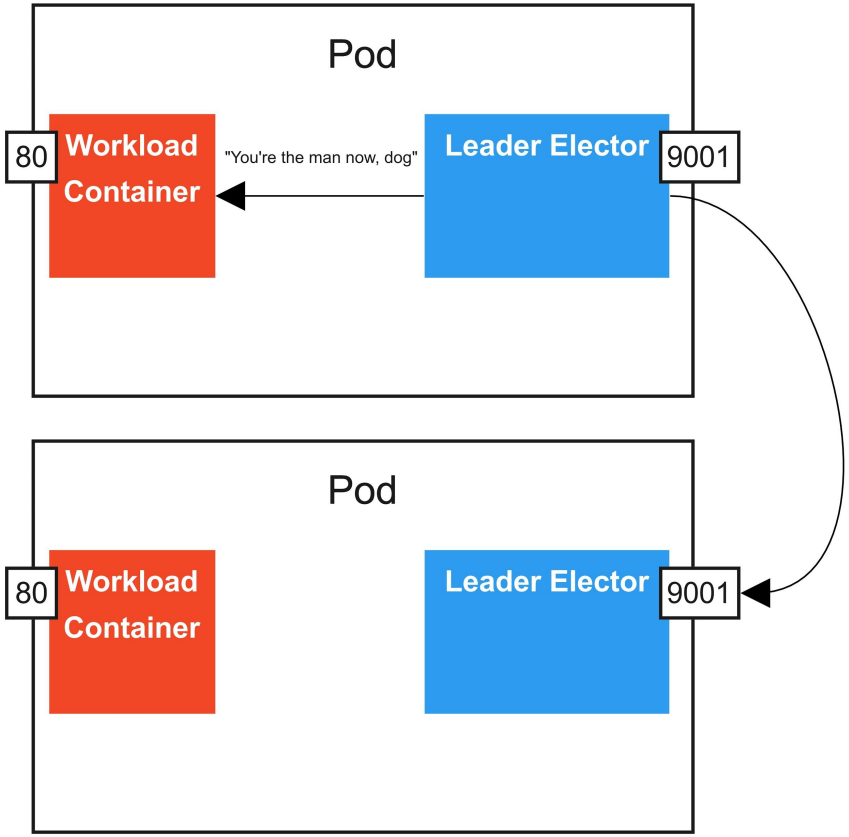
How can my platform communicate with a workload when I want a different protocol than it was built with?

Adapter


Pod
80
Workload Container
Stack-Specific Monitoring Adapter
9001
Opinionated Monitoring Service

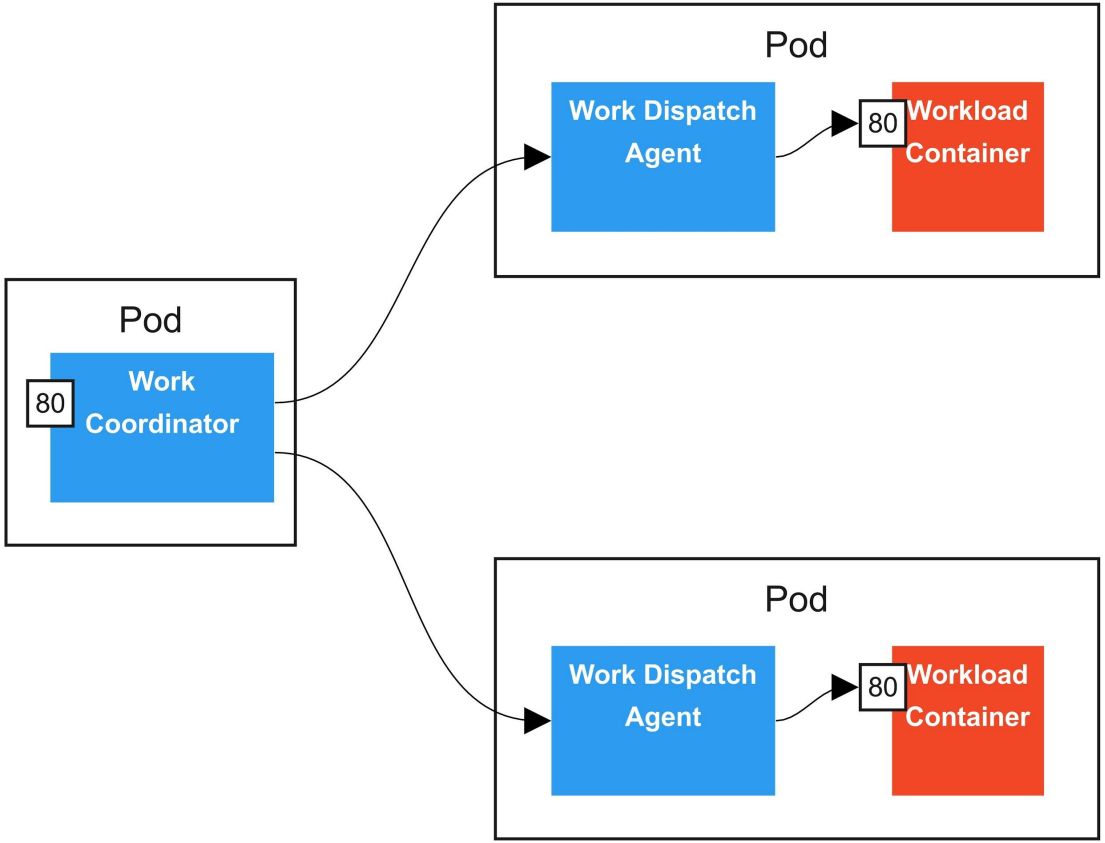
How can my platform provide “singleton” behaviors in a scaled-out service?

Leader Elector



How can my platform provide “work queue” behavior without altering a workload?

Work Queue


Pod
Work Dispatch Agent
80
Workload Container
Pod
80
Work Coordinator
Pod
Work Dispatch Agent
80
Workload Container

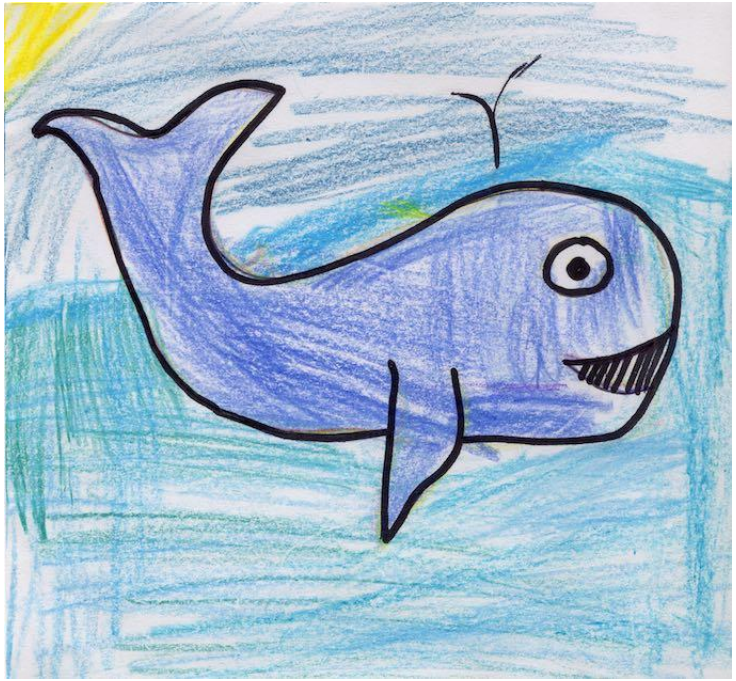
Kubernetes Tweet Bait

“Could this be POSIX of distributed systems?!”

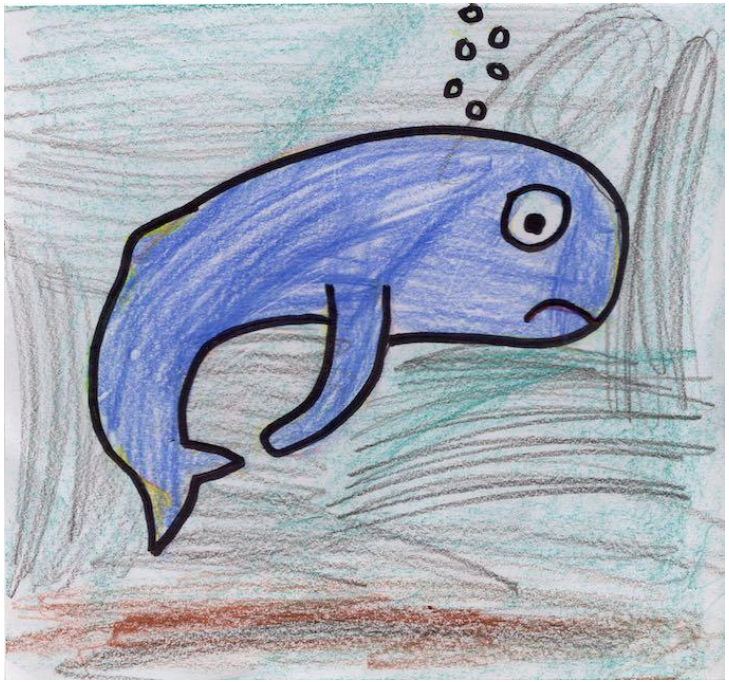
How does it all come together?

Scalewhale: A troubled service

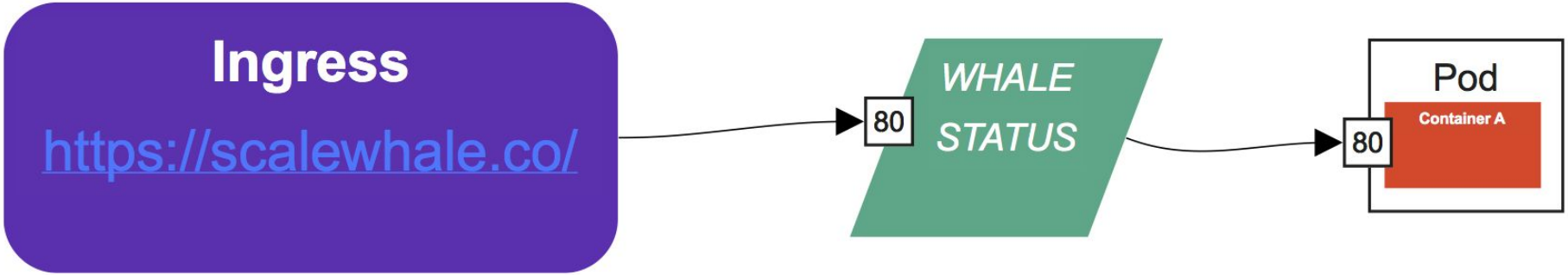
The output we want…



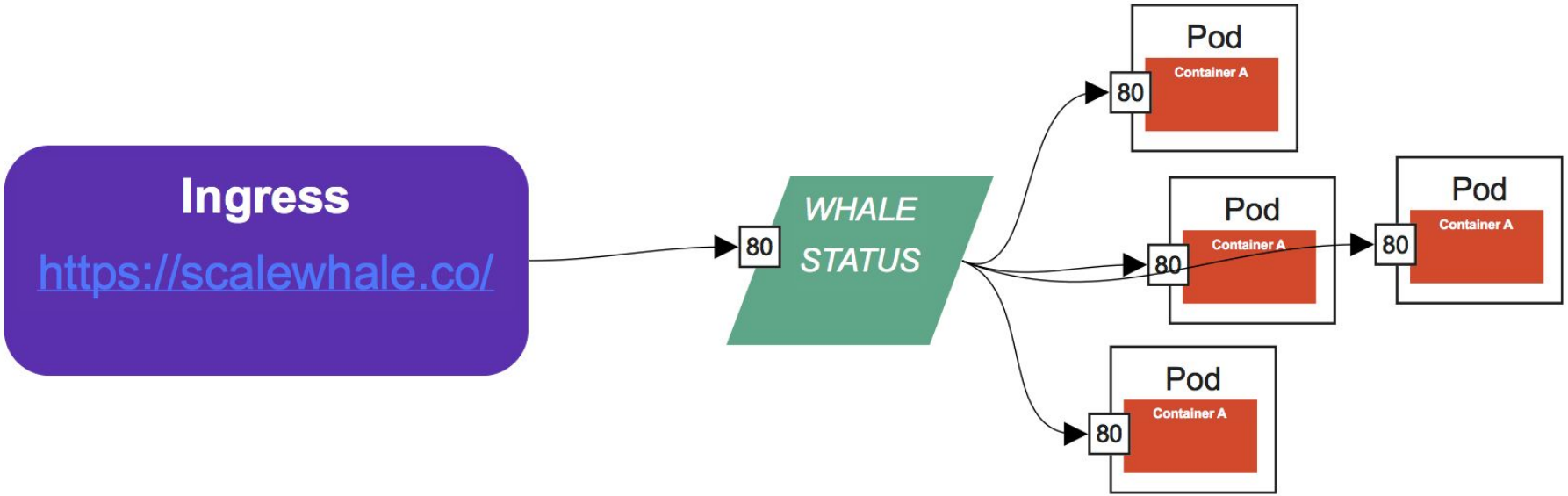
… but we get overloaded



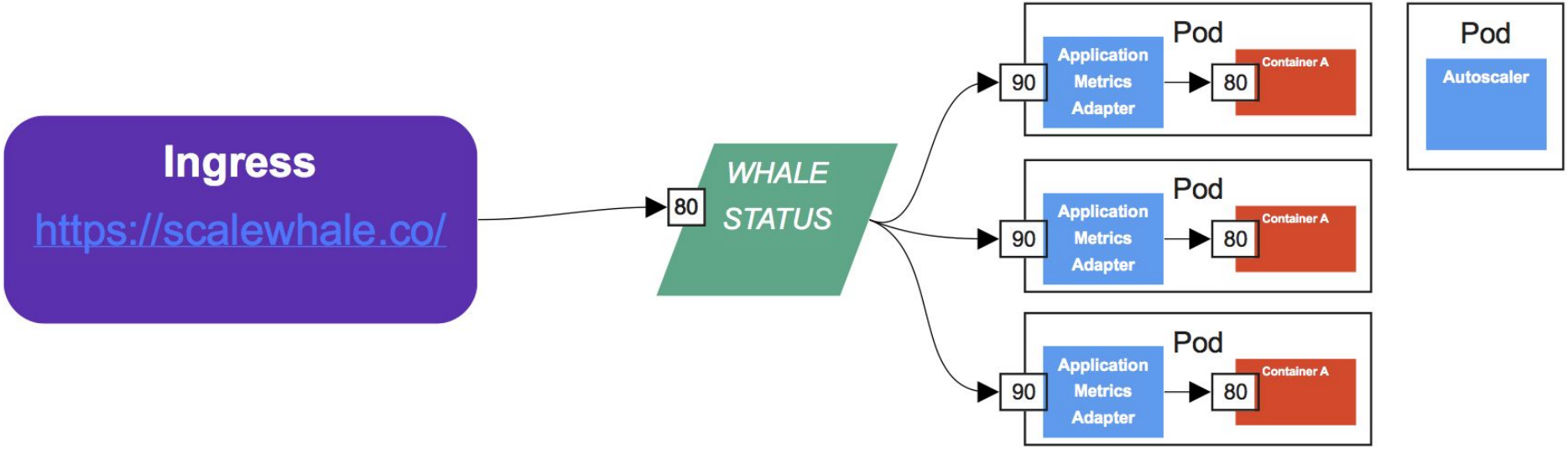
Initial rollout



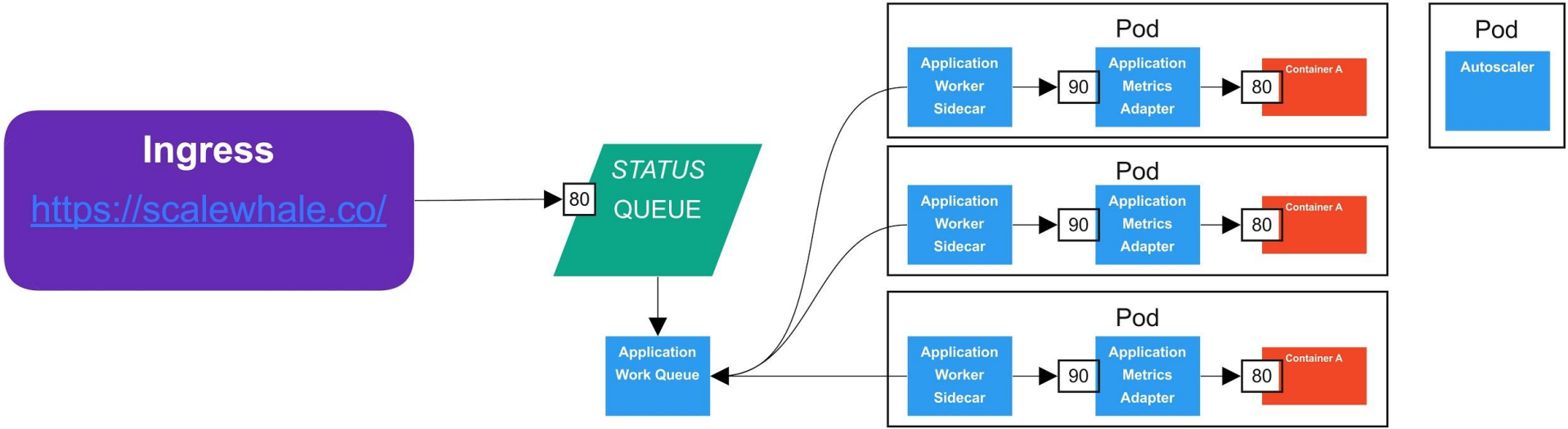
Brute force scale-out



Metric-driven Autoscale



Swap in a work queue!



Questions

Get hip to the heptagon

A platform is a real developer advantage but must avoid reinvention and being overly proscriptive.

Kubernetes was built to bring independence from hardware choices.

Kubernetes also brings separation of concerns to dev teams.

It’s built from simple rules and objects that improve the usefulness and portability of containers.

Slides available at
https://is.gd/k8splatform

Bibliography

“Design Patterns for Container-base Distributed Systems” -- Burns, Oppenheimer USENIX 2016

“Site Reliability Engineering” -- Beyer, Jones, Petoff, Murphy. O’Reilly 2016

“From Google to the World: The Kubernetes Origin Story” -- McLuckie, 2016