Embracing Cloud Native at a Thriving, Established Company

Brian Akins | Principal Engineer

About Me

@bakins

Principal Engineer, MailChimp

Terrible public speaker

Kubernetes since 0.14

C, Go, Containers, etc

Raspberry Pi, sheet metal, homebrew beer

MailChimp®

1,000,000,000

MailChimp sends about one billion emails per day

We’re hiring!

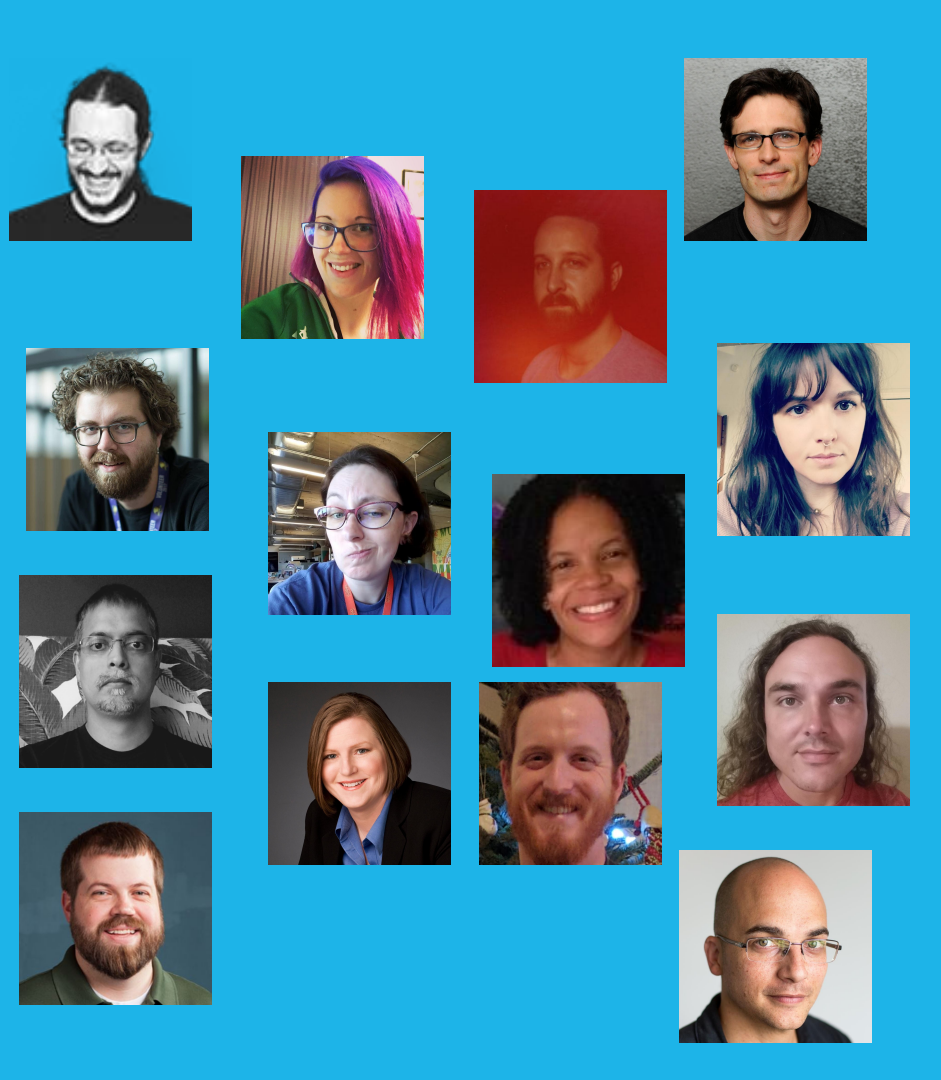
https://mailchimp.com/jobs/

Atlanta
Brooklyn
Oakland
Remote

Thanks to Team Pando*

Dustin	Kelly
Steph	Patryk
Ted	Kelly
Shanna	Adria
Kris	Jeffrey
Vicki	Robby
Joe	Eric

*Pando - clonal colony aspen tree in Utah

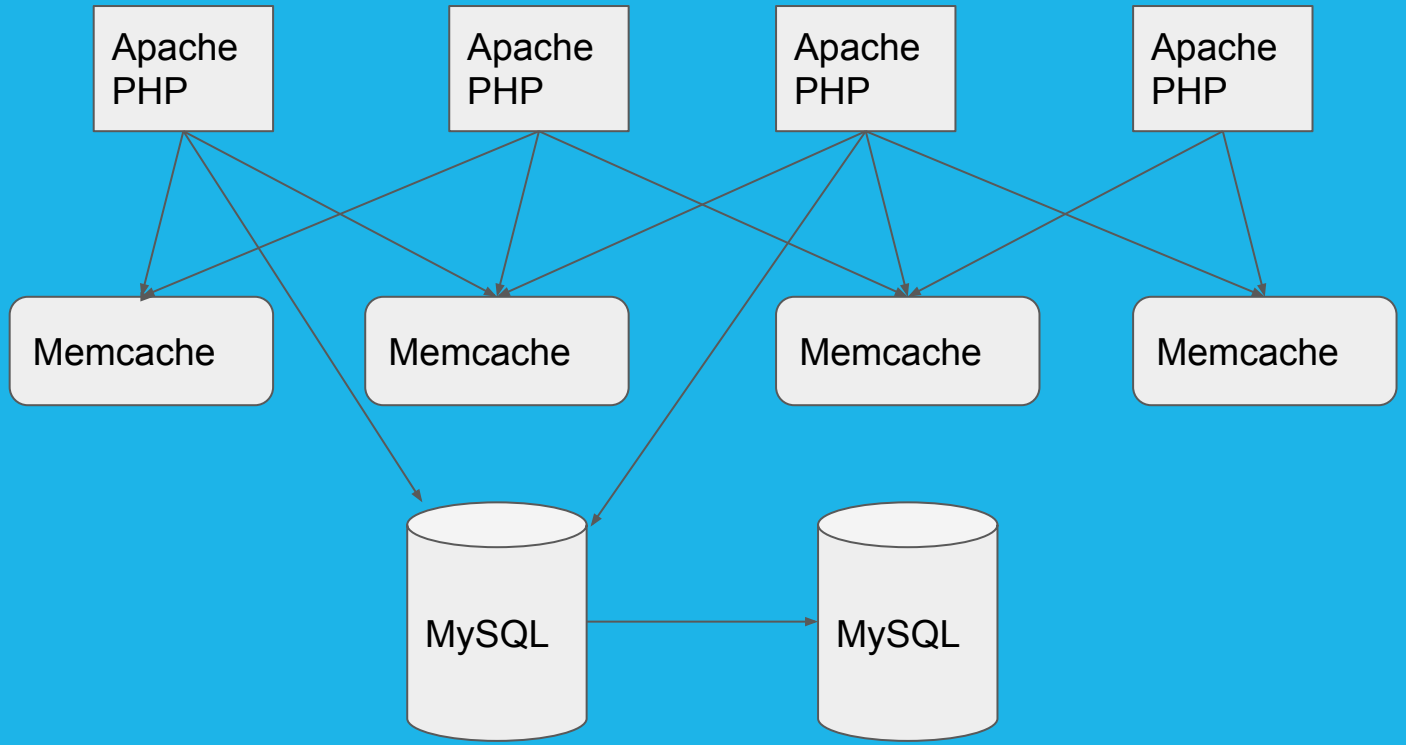


Warning!

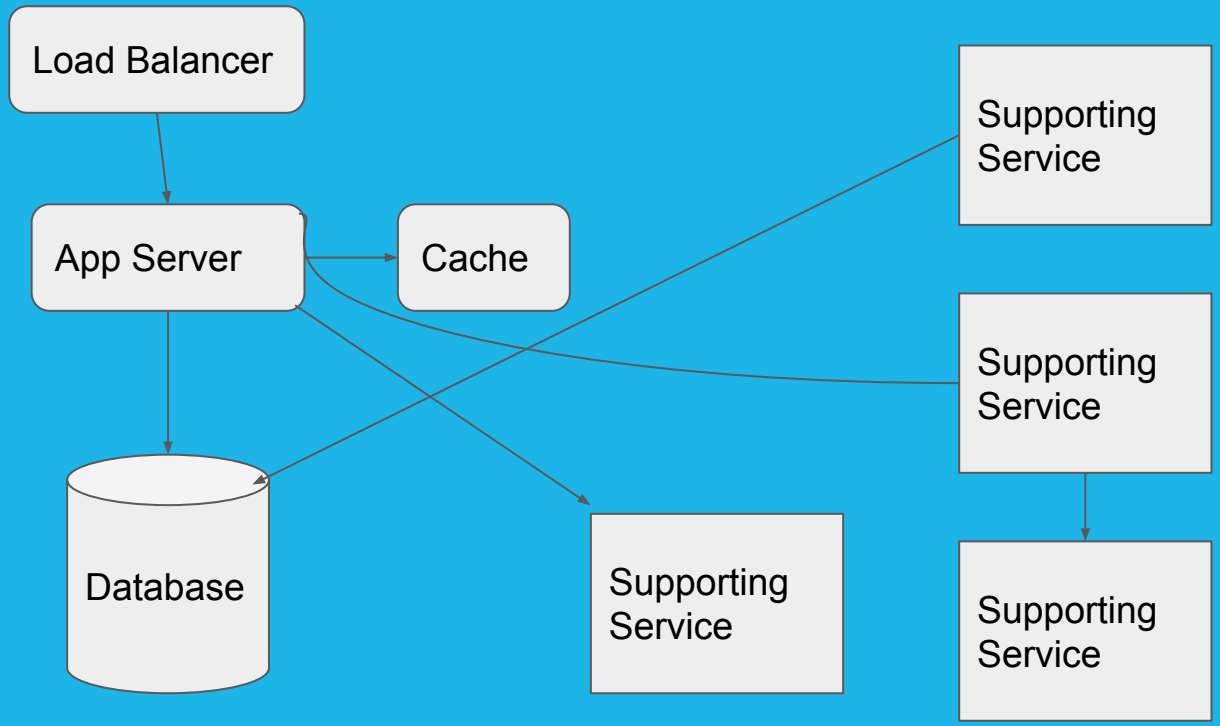
This is a summary of our journey. For clarity and brevity, I am omitting a ton of details and context. It may or may not be applicable to your situation. What worked for us may not work for you.

Current MailChimp Architecture

Typical LAMP Monolith



Maybe Not That Simple



Scaling Out

17 “shards”

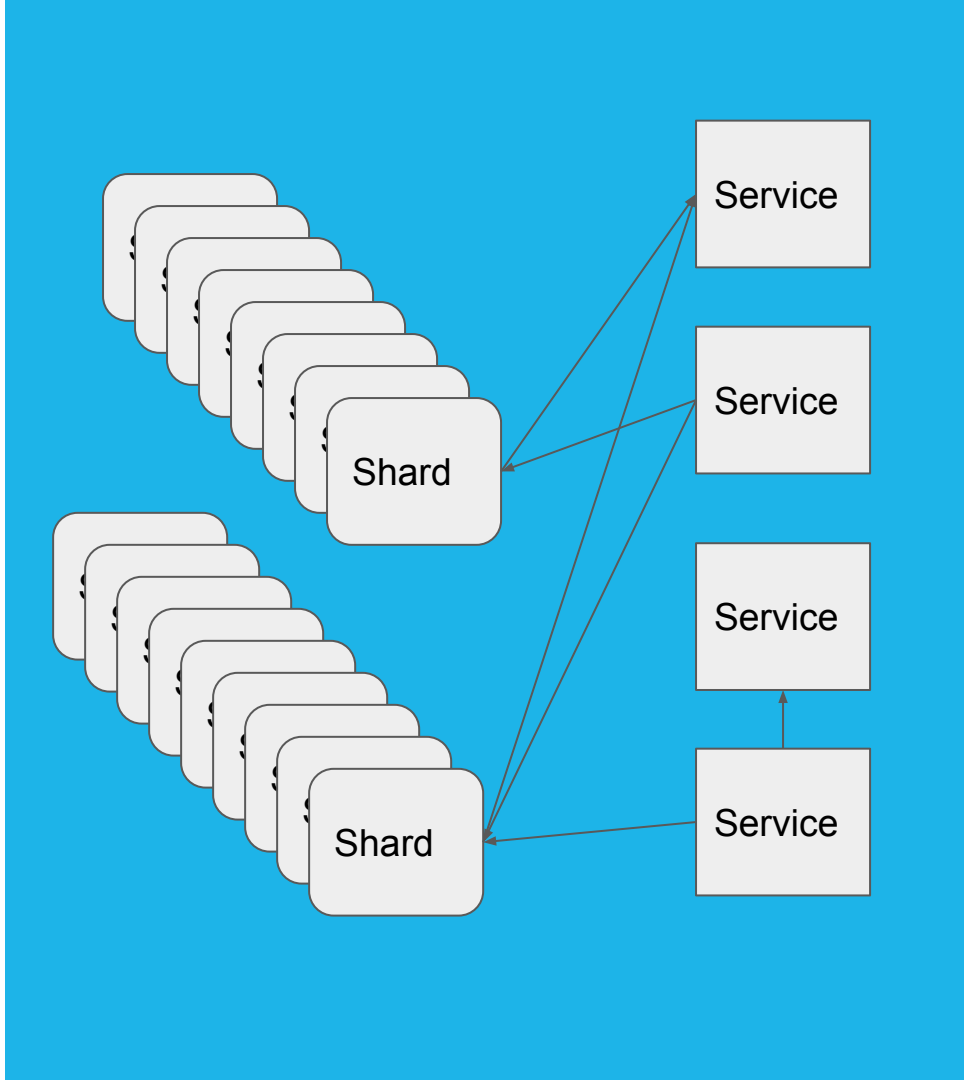
17 Instances of the Monolith

17 Load Balancer Pairs

Hardware in 3 datacenters

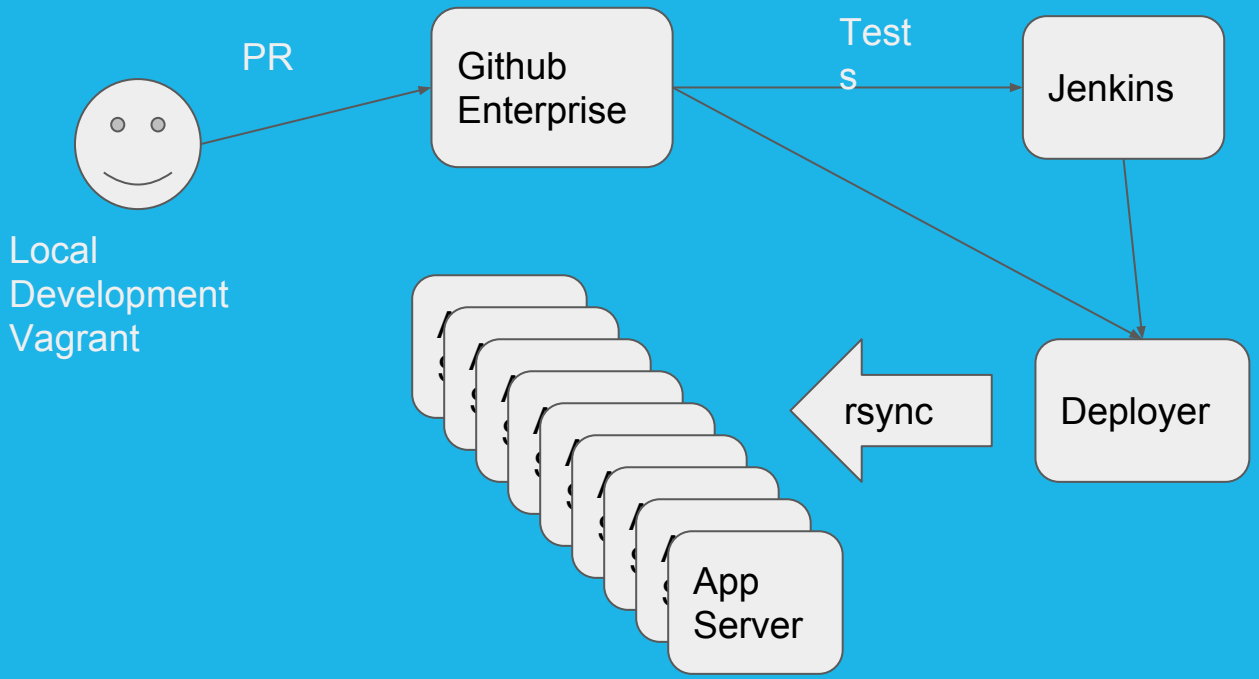
Mostly Baremetal - Few VMs

Managed with Puppet



Current MailChimp Workflow

How Code Gets To Production



Automatically Deploy on Merge To Master

Engineering Expectations

Local development gives instant feedback

- Edit PHP and see results in browser
- Can run tests locally

Master is always running in production

- All shards run same PHP code
- Master must always work
- Feature flags

Why Change?

Monolithic Service becoming unwieldy

- Experimentation becoming more difficult
- Many teams iterating on same code

Hardware Resource Scaling

- Ordering lead times
- Scaling on the fly

Atomic, Immutable, Self-Contained Deployments

DR/BCP

Standardize logging, metrics, etc

How to Get There

Plans vs. Reality

Phase One: On Premise Kubernetes

Deploying and operating Kubernetes on bare metal in multiple datacenters.

- Deployment
- Networking
- Authentication
- Integration
- Ingress

kubernetes

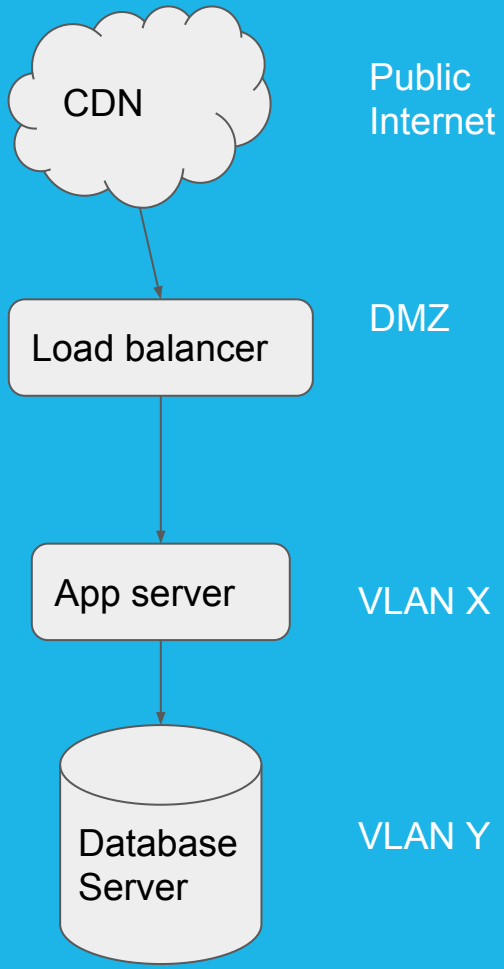
Deploying Kubernetes

Kubernetes is just another application

Puppet manages Kubernetes components

Use existing tools and knowledge

Run etcd, apiserver, controller-manager on 5 nodes in each cluster



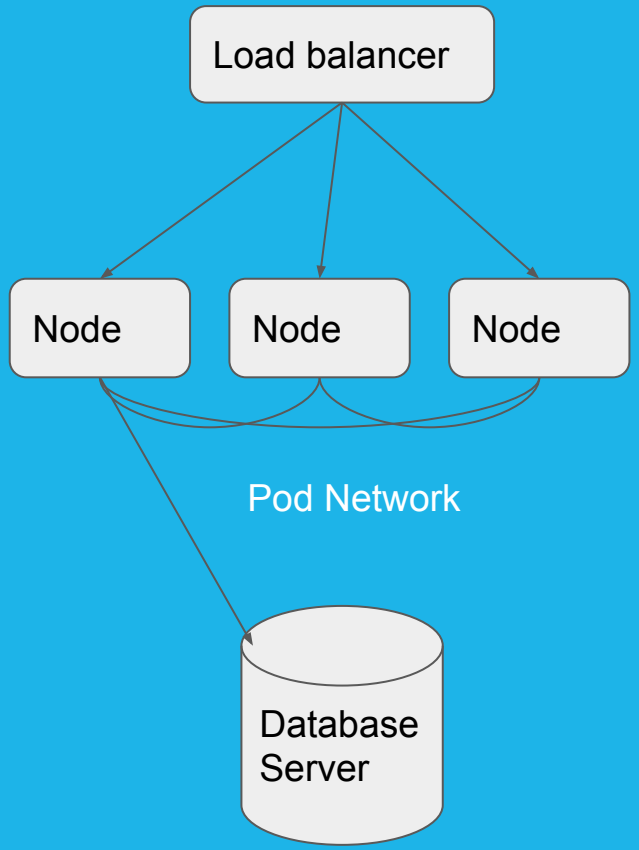
Current Networking

Fairly standard network

VLANs and ACLs

DMZ. VPN, MPLS, etc

Hardware switches, routers, firewalls, etc



Kubernetes Networking

Treat Kubernetes as Application Servers

“There’s no such thing as container networking.”

Keep it Simple

No Overlay

Layer 2 connectivity between hosts

Frontend/Backend networks

Host Routes

https://en.wikipedia.org/wiki/Lock_(security_device)

Authentication

OIDC via webhook

Need Groups Information

Custom app running on each apiserver

RBAC Roles

(Cluster)RoleBindings based on group

Kube-apiserver audit logs

Integration

Kubernetes is just another application

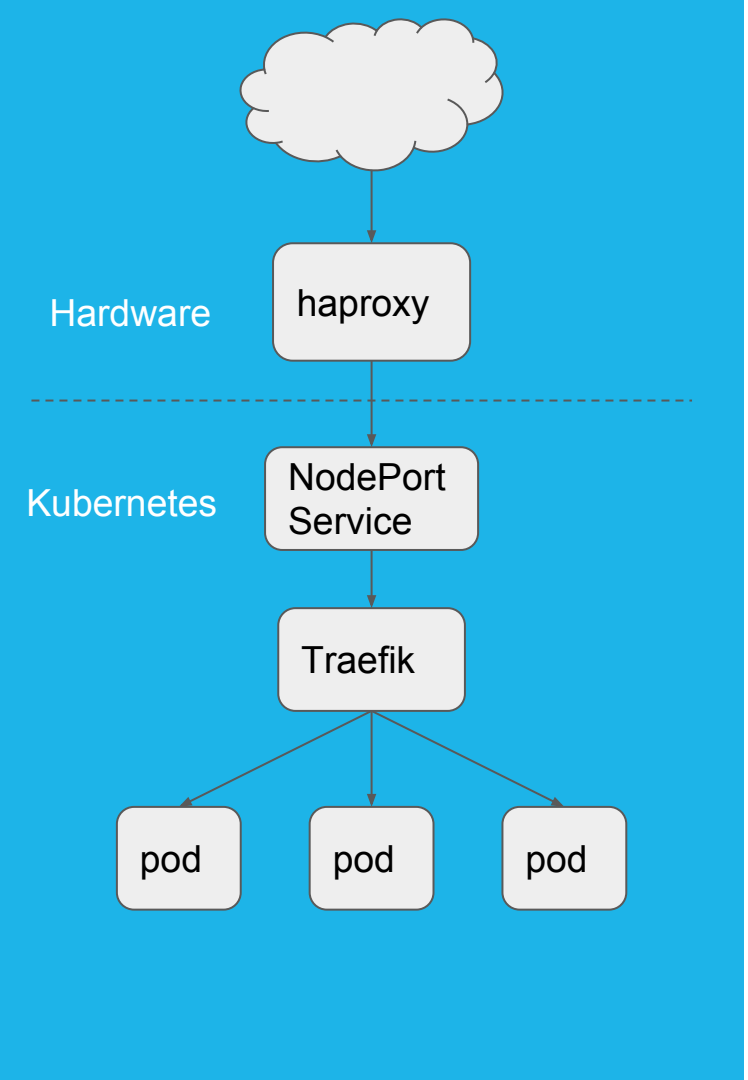
Needs to integrate with existing systems

Logs shipped via Fluentd and Filebeat

Zabbix

Prometheus/Alertmanager

Sealed Secrets



Ingress

HTTPS only

Internal and DMZ load balancers

Traefik for Ingress Controller

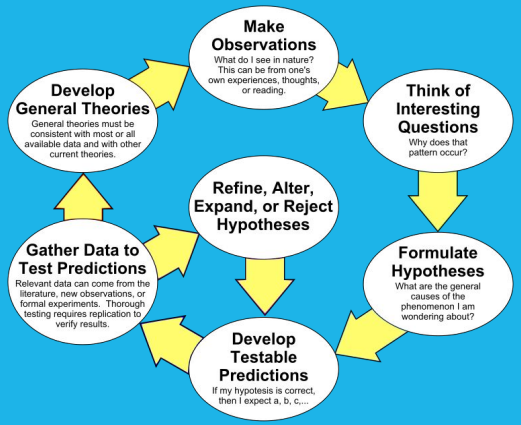
Traefik terminates TLS

Ingress Resources with labels

Phase Two: Application Deployment

Reliable builds and deployments

- What we tried
- What didn’t work
- What we are doing



https://en.wikipedia.org/wiki/Scientific_method

Failed Experiments

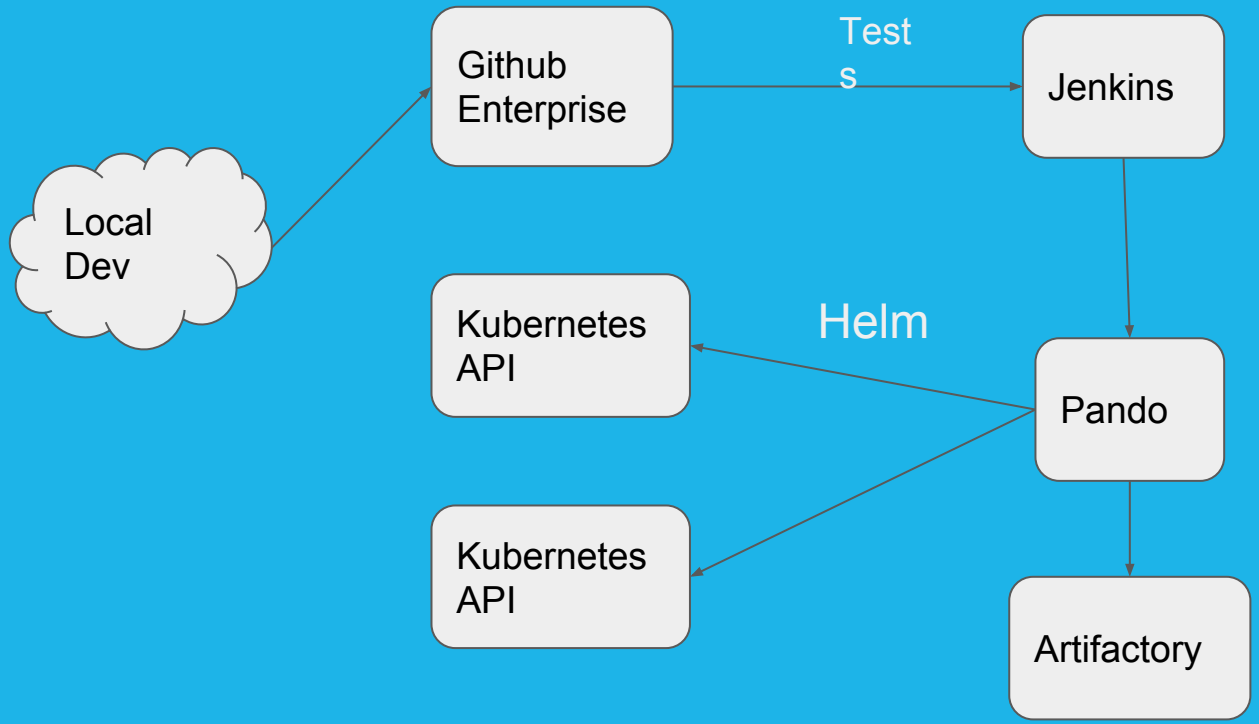
- Helm monorepo
- Repo per Chart
- Kubectl apply wrapper
- Monorepo of “raw” YAML

Currently

Simple wrapper around helm

Chart in app repo

What we are doing - for now

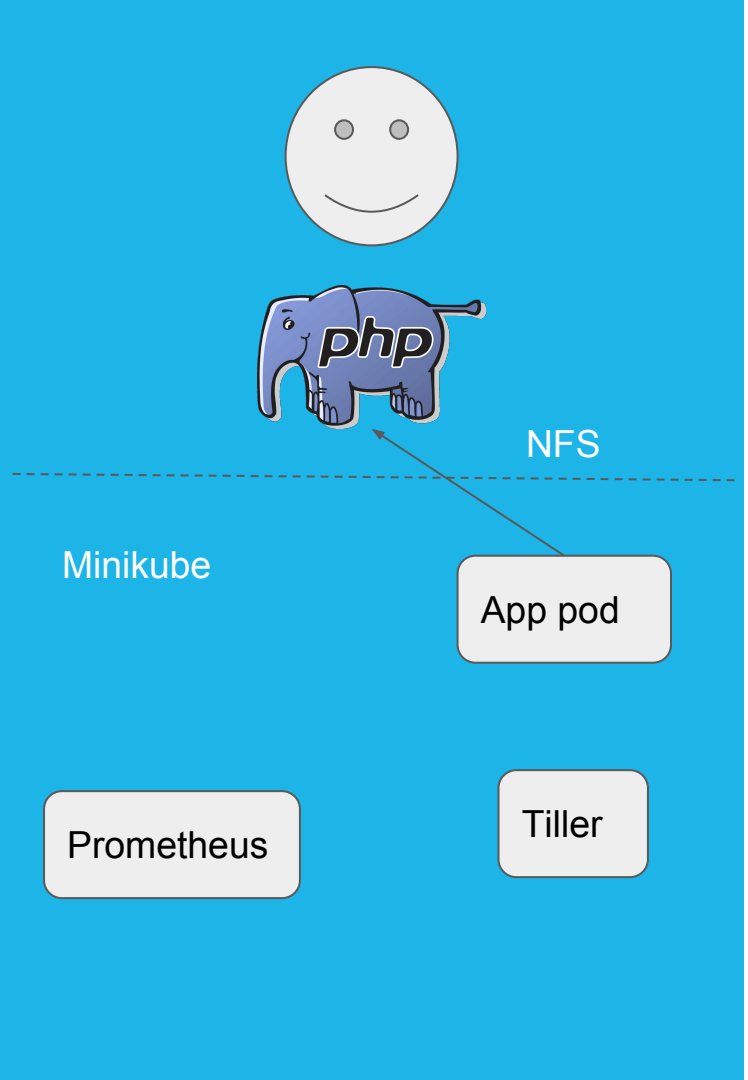


Automatically Build/Deploy on Merge To Master

Phase Three: Developer Experience

How to maintain similar developer workflow

- Local development
- Fast Feedback
- Dev/Prod parity



Local Dev Feedback Loop

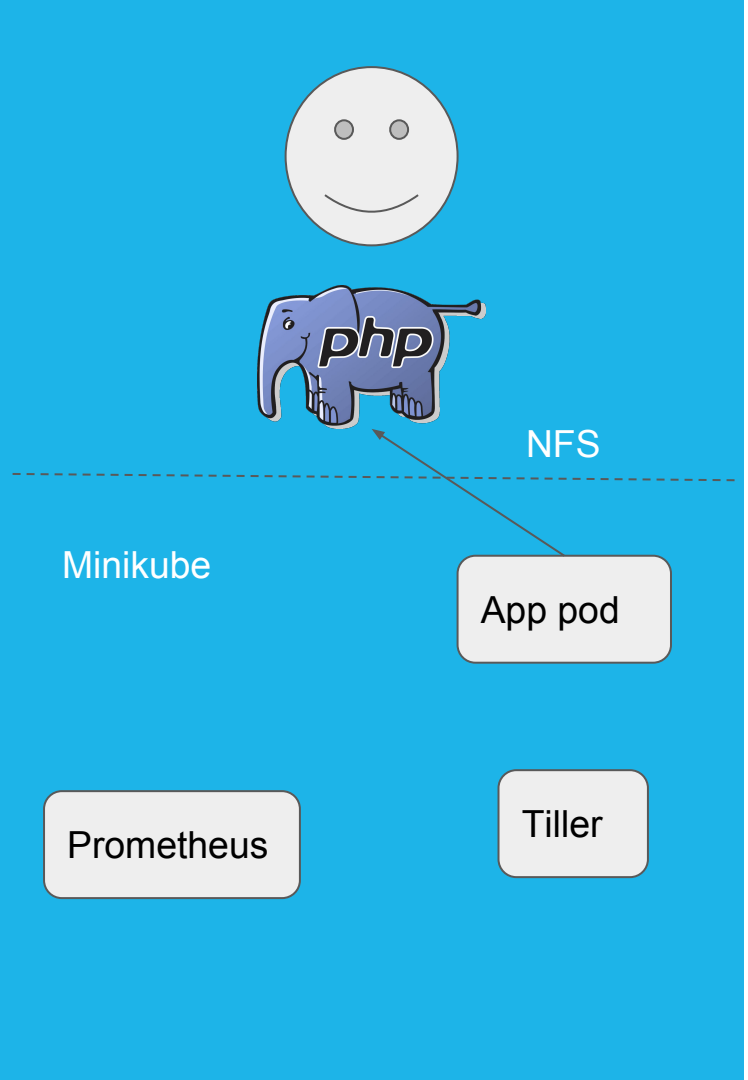
Edit PHP - see in browser

Building image each time was too slow

Minikube plus NFS

Dev can see metrics in local Prometheus

Pando used locally as well



Dev to Prod

Charts are part of application code

Pando tool expects a certain layout

Can iterate on application and deployment locally

Same tooling is used by Jenkins to deploy on merge to master

Phase 3.5: Deploying Real Applications

When we had enough tooling in place, we need people to actually use it

https://en.wikipedia.org/wiki/Shed

First Application

Tweak of “simple” snapshot app

The app is not so simple.

No one actually knows how it works.

Complete rewrite

PHP 7 with Headless Chrome

Bike-shedding

Tooling being written at same time.

https://en.wikipedia.org/wiki/Sawmill

Our Real First Application

Log Processing

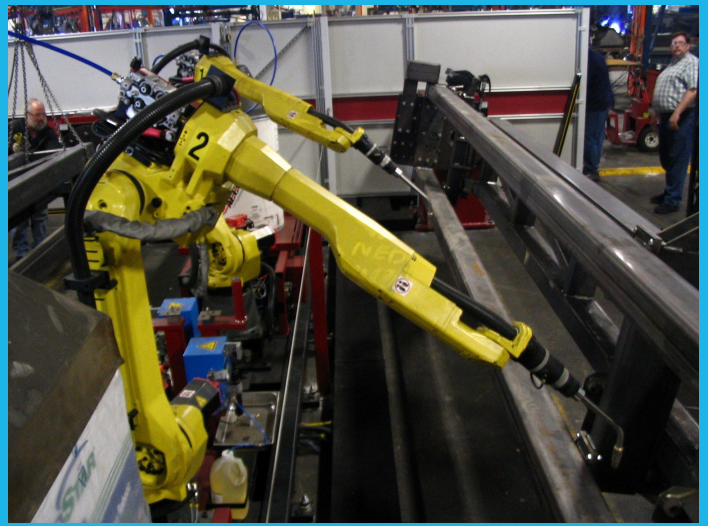
Already Containerized

Outgrew old infrastructure

Patient Team

Real load, not toy apps

Not directly user facing



https://en.wikipedia.org/wiki/Robot

Chat Bots

Slack migration

So many bots…

Different teams

Different languages

Good test of workflows and tooling

Feedback loop with developers

Oops...

One Cluster “full”

All CPU cores requested

“We don’t need to worry about capacity for a while…”

Request CPU versus used

Autoscale

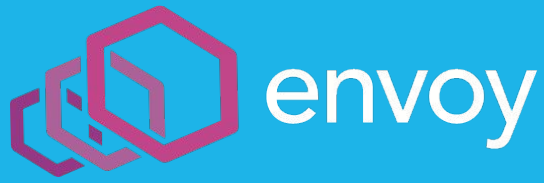
Future Work/Plans

Business Critical Applications

Updating a key business component

- Will be done on Kubernetes from day one
- Real money involved

Then, perhaps the monolith



gRPC Service Mesh

Standardize transport and calling mechanism

- TLS
- Service Authentication
- Service Discovery
- Circuit breakers

Currently, everyone does this per application

gRPC Services in PHP

Yes, it is possible.

Not ready to share, yet.

Stay tuned...

Reuse years of Application code

Questions?

I skimmed over details.

I skipped a few bumps and bruises.

I have a long list of “I wish we hads”