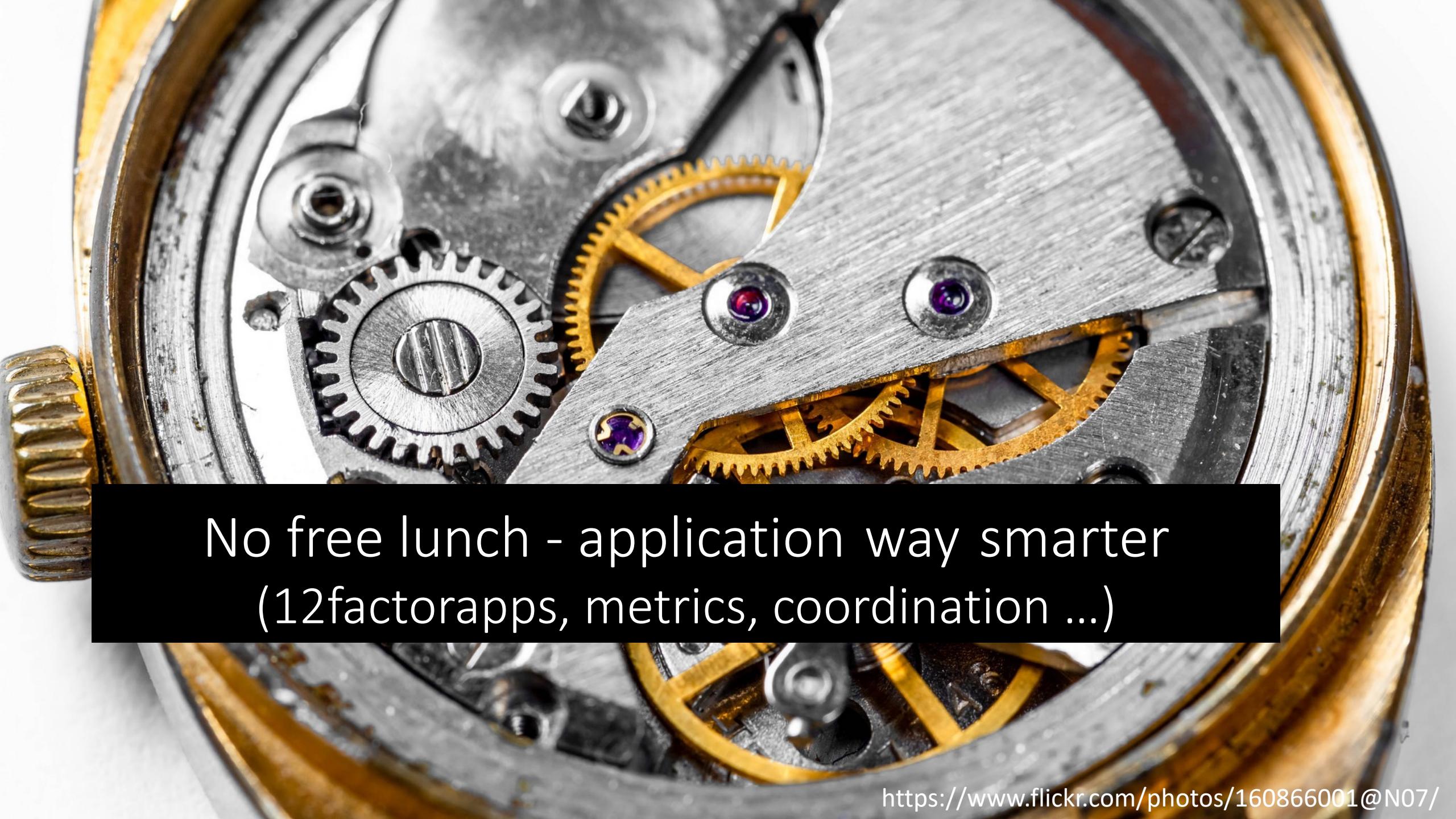


What should you know to start building Golang Services for Kubernetes?

Wojciech Barczyński
VP of Engineering



SPACELIFT



No free lunch - application way smarter
(12factorapps, metrics, coordination ...)

App development

- 12 factor apps
- Observability
- Liveness and Readiness probe
- Working with Kubernetes



12 factor apps

1. Inject the configuration with the env variables
2. Log to stdout (structured)
3. Scale up the app by adding new instance



12 factor apps

4. MTTR versus MTTF, fail fast
5. Assume every external call might fail
 - retries and timeouts

Observability

	Metrics	Logging	Tracing
CapEx	Medium	Low	High
OpEx	Low	High	Medium
Reaction	High	Medium	Low
Investigation	Low	Medium	High

Observability - logging

- structured

uber-go/zap and logrus

```
logger, _ := zap.NewProduction()
defer logger.Sync() // flushes buffer, if any
sugar := logger.Sugar()
sugar.Infof("failed to fetch URL",
    // Structured context as loosely typed key-value pairs.
    "url", url,
    "attempt", 3,
    "backoff", time.Second,
)
sugar.Infof("Failed to fetch URL: %s", url)
```

Observability - logging

- Pass the logger with Context

```
// A common pattern is to re-use fields between logging statements by re-using
// the logrus.Entry returned from WithFields()
contextLogger := log.WithFields(log.Fields{
    "common": "this is a common field",
    "other": "I also should be logged always",
})
```

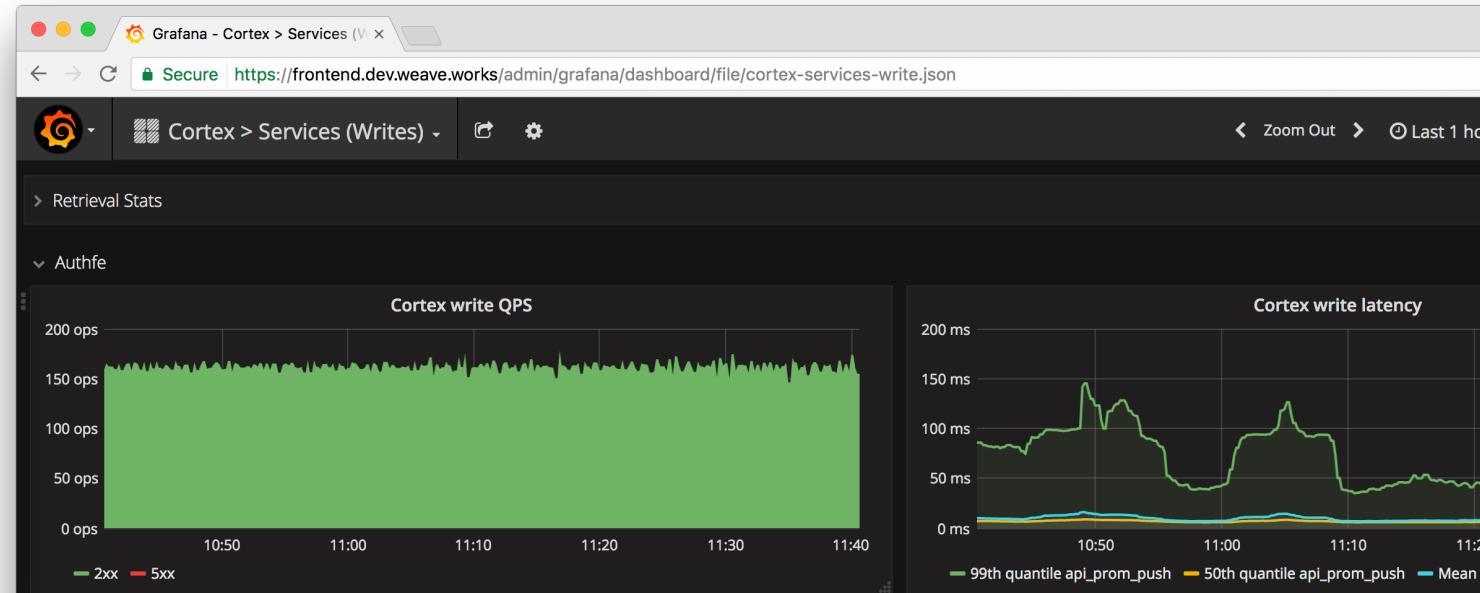
- Configurable with env variable - LOG_LEVEL

Observability - monitoring

Name	Label	Value
traefik_requests_total	code="200", method="GET"	3001



Prometheus



Observability - monitoring

- Use Prometheus Golang library
- Monitor yourself
- Monitor your dependencies
- Check the best practices -
<https://prometheus.io/docs/practices/naming/>

<https://prometheus.io/docs/guides/go-application/>

Observability - monitoring

- Annotate your service

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/port:    '9090'
    prometheus.io/path:   'metrics/'
spec:
  selector:
    app: my-app
  ports:
    - port: 8080
      target_port: 8080
```



Prometheus

<https://prometheus.io/docs/practices/naming/>

Observability - monitoring

- RED
 - Rate Error Duration
- USE
 - Utilization Saturation and Errors

Cooperation with K8S

- **Readiness probe**

- are you ready to server traffic?
- are my dependencies responding?

```
readinessProbe:  
  httpGet:  
    path: /ready  
    port: 8090
```

- **Liveness probe**

- am I alive?
- did I hanged

```
livenessProbe:  
  httpGet:  
    path: /healthz  
    port: 8090  
  initialDelaySeconds: 5  
  periodSeconds: 5
```

Cooperation with K8S

- **Readiness probe**

- are you ready to server traffic?
- are my dependencies responding?

```
readinessProbe:  
  httpGet:  
    path: /ready  
    port: 8090
```

- **Liveness probe**

- am I alive?
- did I h

```
livenessProbe:  
  httpGet:  
    path: /healthz  
    port: 8090  
    delaySeconds: 5  
    initialSeconds: 5
```

Start with Liveness.

Cooperation with K8S

When the probe fails:

- **Readiness probe**
 - Kubernetes stops (eventually) sending requests
- **Liveness probe**
 - Kubernetes SIGTERM followed by SIGKILL

When K8S stops your pod

1. Kubernetes sends SIGTERM
2. App should handle it
and set the readiness probe to not ready
3. Kubernetes stops sending requests
4. App should shutdown gracefully

```
func (s *Service) prepareShutdown(h Server) {
    signal.Notify(s.Stop, os.Interrupt, syscall.SIGTERM)
    <-s.Stop
    s.StatusNotReady()
    shutdown(h)
}
```

QoS – why k8s migrated my pod

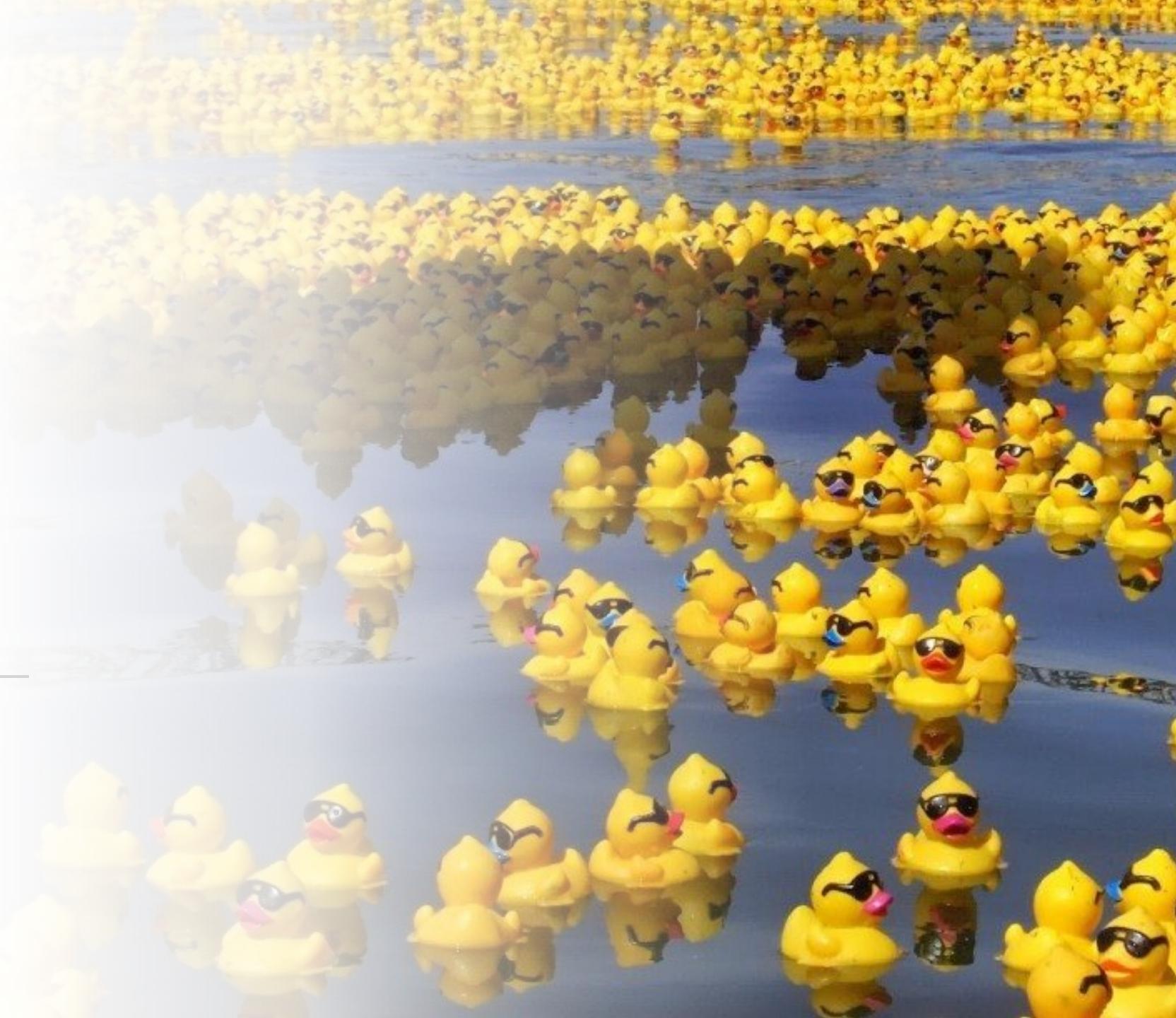
QoS:

- Guaranteed
- Burstable
- Best Effort

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "700m"
      requests:
        memory: "200Mi"
        cpu: "700m"
```

Smoke test & Readiness /Liveness probes

Does my app work?



How to choose the tech?



High performance

- Lead Time
- Deployment frequency
- Mean time to Recovery
- Change Fail Percent



High performance

- Commercial success
- ...
- Satisfaction of the employees



Our goal

- Lead time
- Deployment frequency



Kubernetes

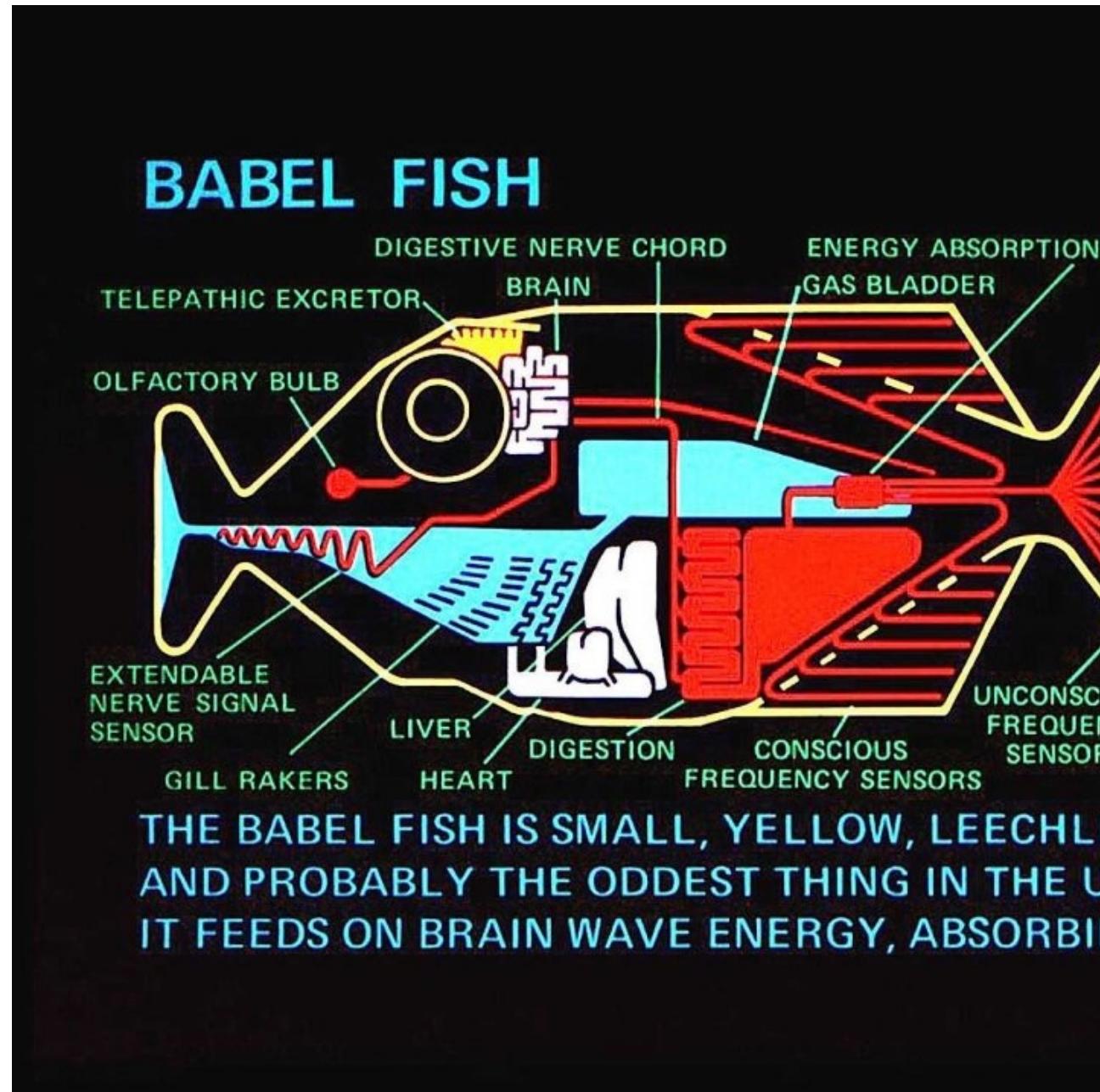
- Black (Blue) Box
- Almost invisible infrastructure
- Developer in the driving seat



[https://en.wikipedia.org/wiki/File:Dr_Who_\(316350537\).jpg](https://en.wikipedia.org/wiki/File:Dr_Who_(316350537).jpg)

Kubernetes

- Common language
- Artifacts
- Platform



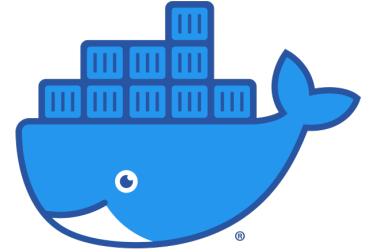
The way of Kubernetes



https://www.flickr.com/photos/bruno_brujah/

- In iterations
- Learn-as-you-go

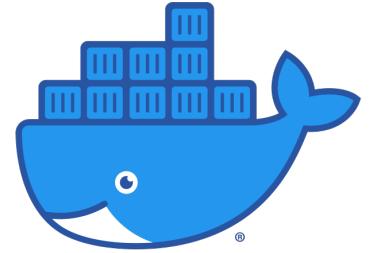
Docker



1. Use linters, for example, hadolint
2. Reduce size and numbers of layers:
.dockerignore && multi-stage

```
RUN apt-get -y update \
    && apt-get install -y vim -no-install-recommended \
    && rm -rf /var/lib/apt/lists/*
```

Docker



3. Do not leave secrets in your Docker
4. Use trivy to scan your Docker images



Manifest generation v1

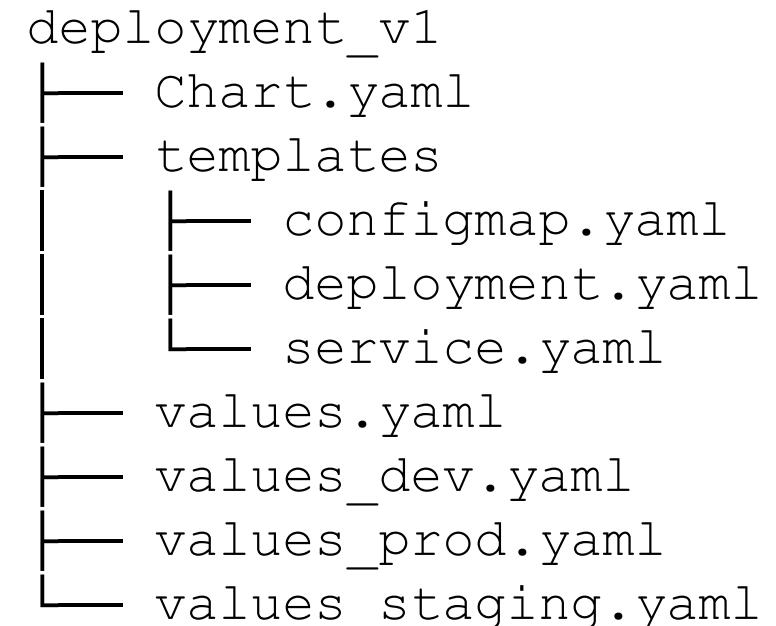
1. Helm as a templating engine
2. Apply the manifest with your CI/CD pipeline
3. Manual secret creation
or go with
[secrets-store-csi-driver.sigs.k8s.io](https://github.com/secrets-store-csi-driver/secrets-store-csi-driver.sigs.k8s.io)

```
deployment_v1
├── Chart.yaml
└── templates
    ├── configmap.yaml
    ├── deployment.yaml
    └── service.yaml
├── values.yaml
├── values_dev.yaml
├── values_prod.yaml
└── values_staging.yaml
```



Manifest generation v1

1. Keep templating to minimal YAGNI
2. Generated K8S manifests apply on the Kubernetes cluster
3. (alternative) use a transitive repo



Manifest generation v2+



Choose your adventure:

- Better secret management
- ArgoCD?
- Deployment strategies
- Dev env on demand

```
deployment_v1
├── Chart.yaml
└── templates
    ├── configmap.yaml
    ├── deployment.yaml
    └── service.yaml
├── values.yaml
├── values_dev.yaml
├── values_prod.yaml
└── values_staging.yaml
```

Manifest generation v2+



- What is a K8S without infrastructure?
- Next - opening Infrastructure-as-a-Code

```
deployment_v1
├── Chart.yaml
└── templates
    ├── configmap.yaml
    ├── deployment.yaml
    └── service.yaml
├── values.yaml
├── values_dev.yaml
├── values_prod.yaml
└── values_staging.yaml
```



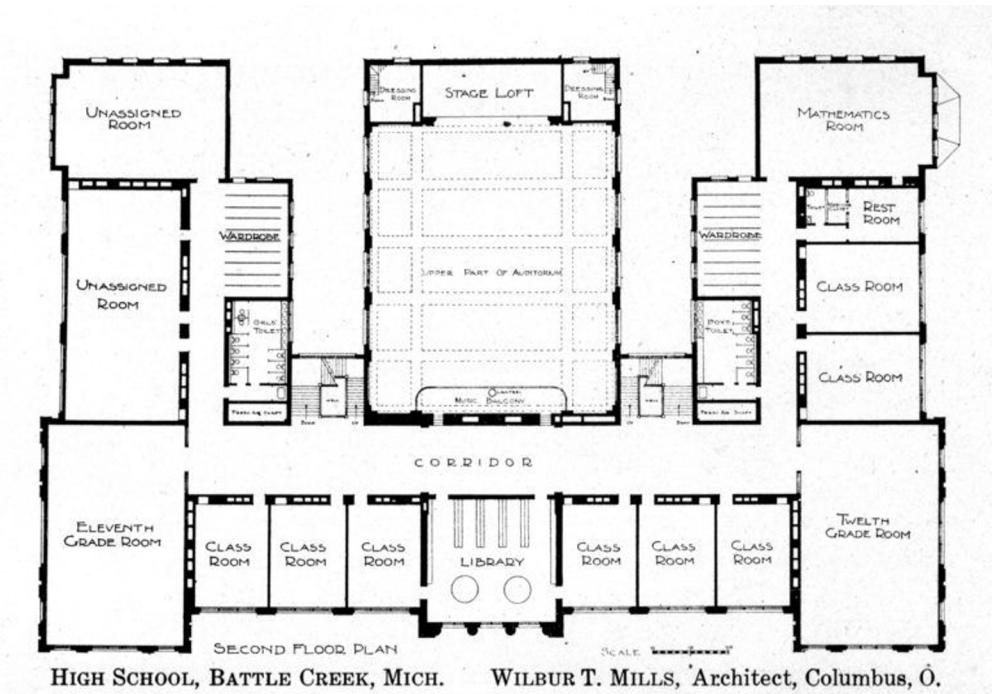
AWS CDK

Kubernetes
operators

Conventions over tools

New repository:

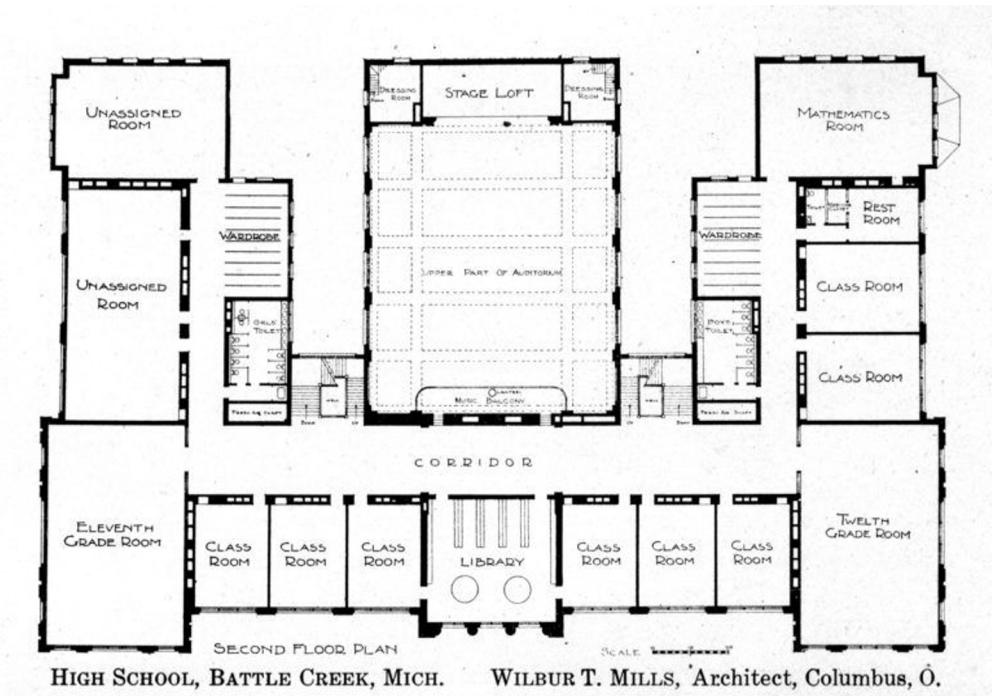
- Generator or copy&paste:
Makefile, ci.yaml,
Dockerfile, helm, ...
- Modify few things
- Ready & Go



Conventions over tools

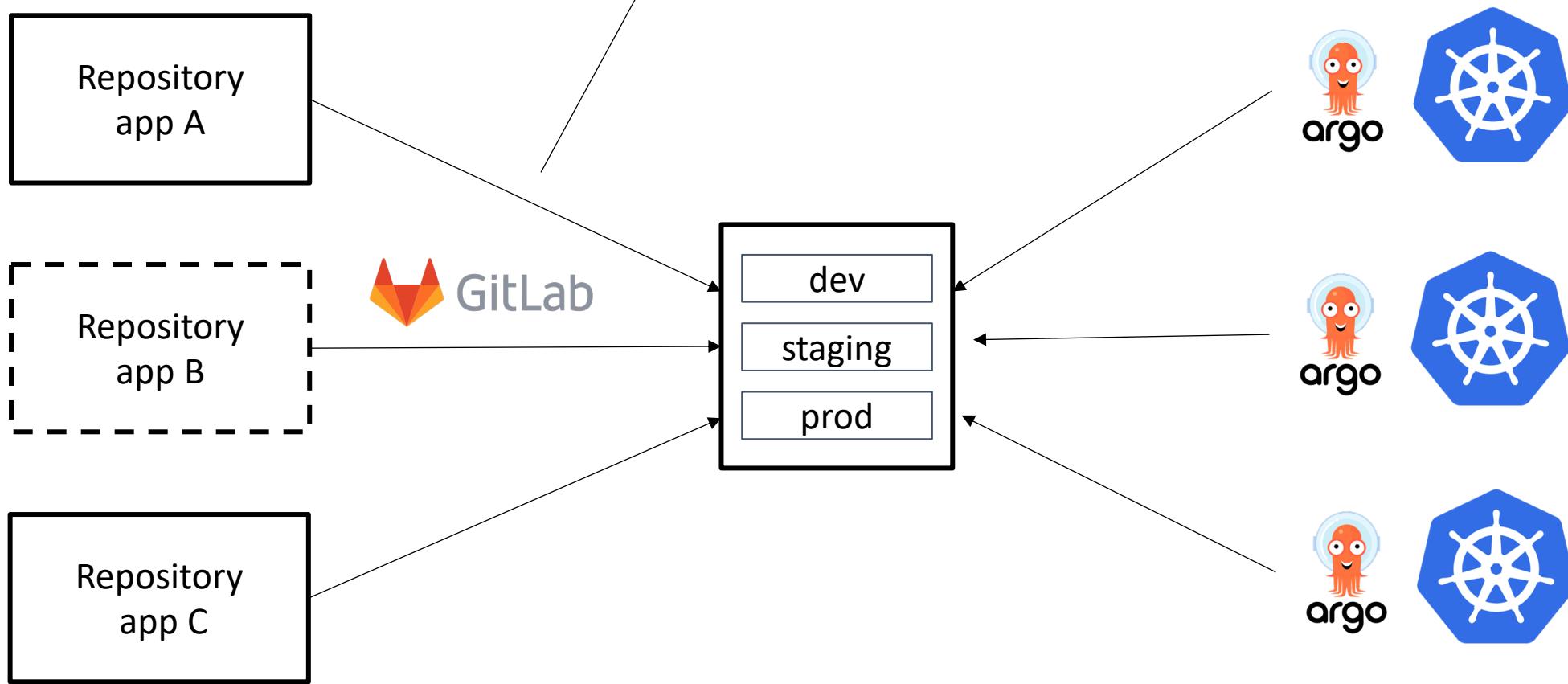
New repository:

- We want people to change the generated code
- We just ask for keeping the common conventions



v1.5

Generation scripts as the interface
between the component and the platform

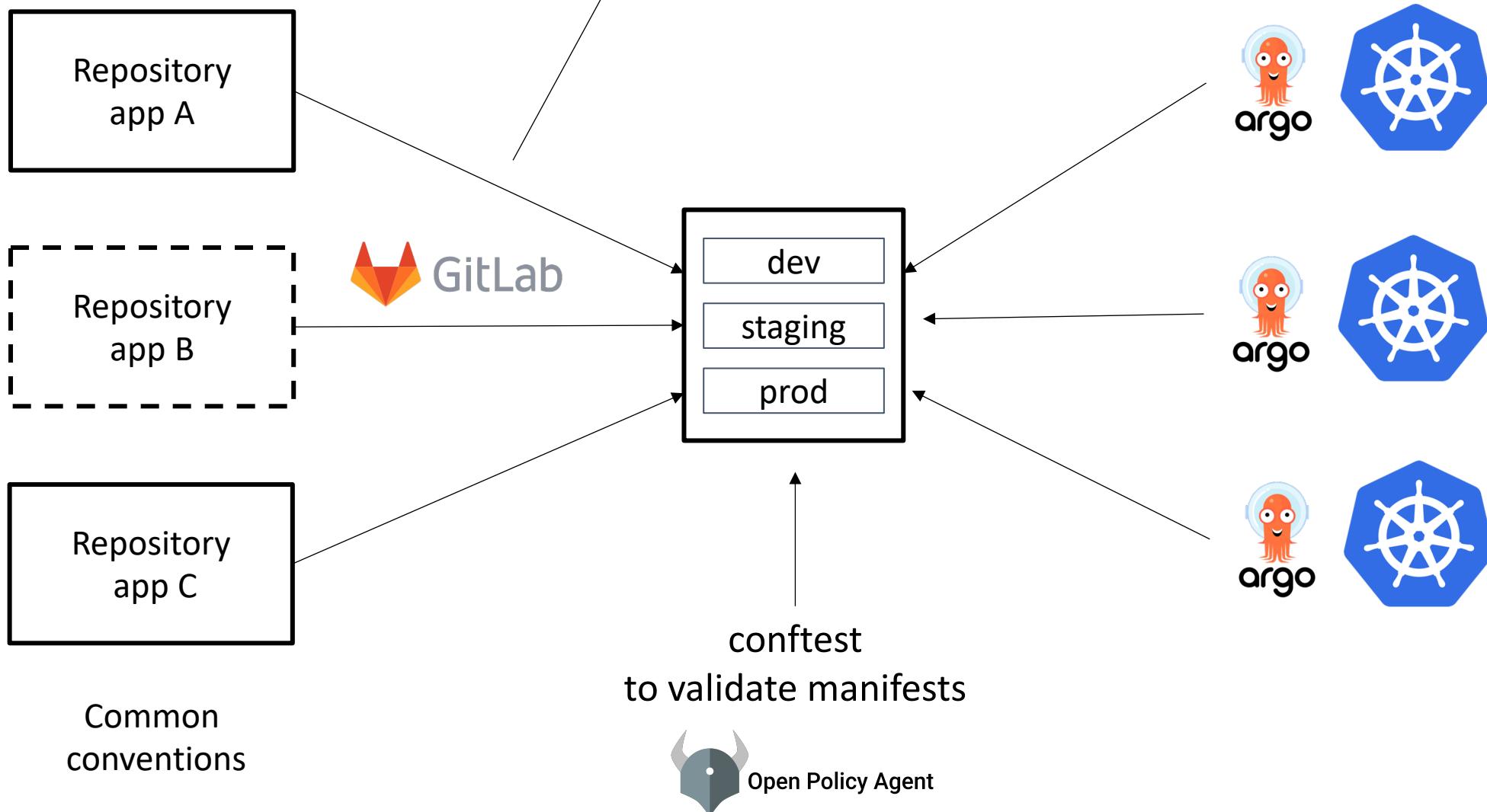


Common
conventions

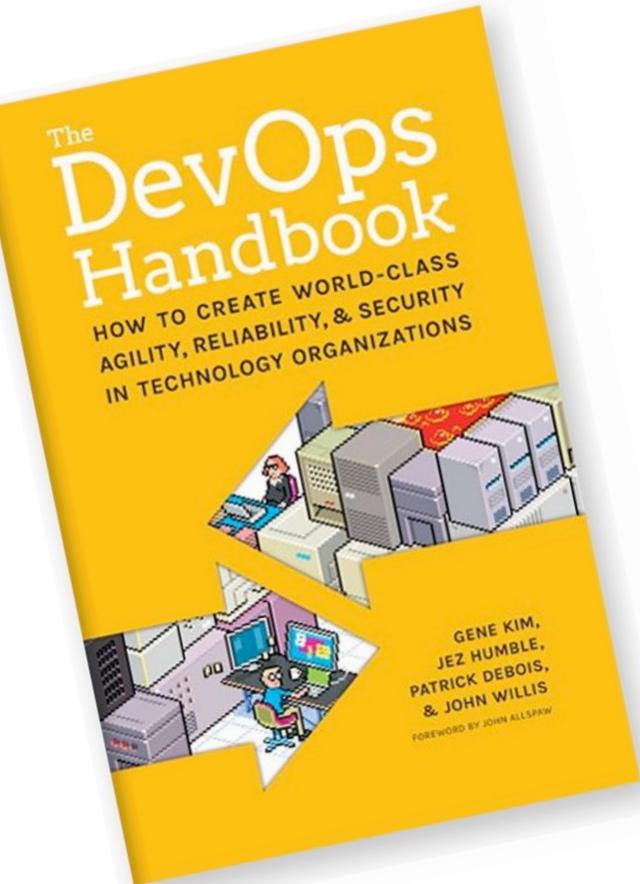
Transitive repo
with the current cluster
state for every environment.

v1.5

Generation scripts as the interface
between the component and the platform



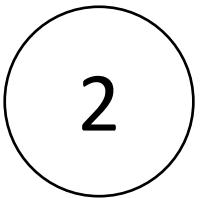
Recommended read



1

Concrete, do A, do B,
because C

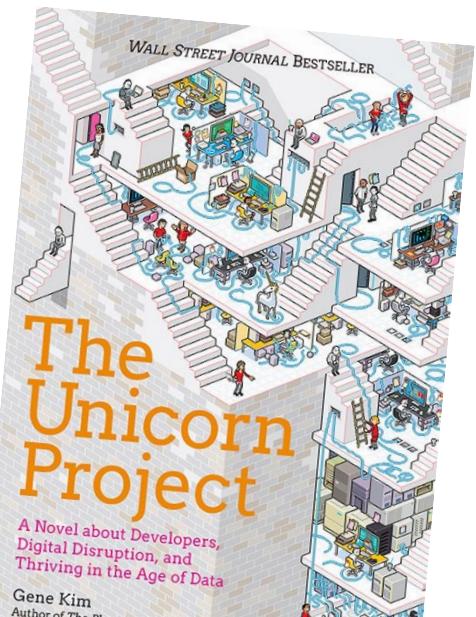
(DevOps here as culture, not an admin ☺)



The first 4 chapters,
the rest if you have time.

3

A story,
not as straight forward
as (1)



Questions?





SPACELIFT

Product ▾

Resources ▾

Company ▾

Pricing

Blog

Login

Hiring

Golang developers
passionate about
system engineering,
DevOps culture,
or building tools.

The most flexible management platform for Infrastructure as Code

Optimize your workflows, automate manual tasks, reduce number of errors, improve security and manage your infrastructure.

START FREE TRIAL



Stacks

Contexts

Policies

Worker Pools

Modules

Billing

Settings



John Doe
johndoe

Backup

Developers

1. Learn basic Kubernetes artifacts and commands
2. CI/CD should take over
3. Common conventions over common tools
- 4.

Rabbit holes everywhere...

Approach:

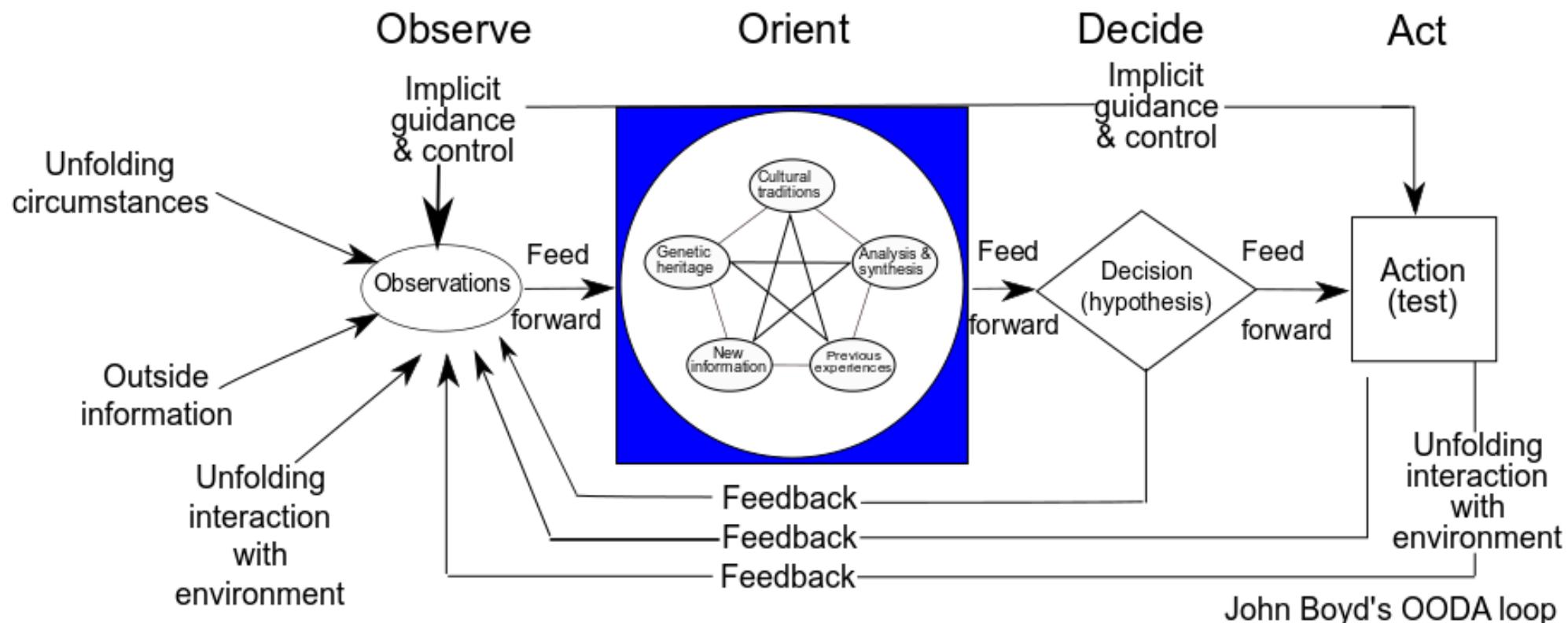
- The iteration, decision, deliver
- As soon as possible
to get into the cycle Patch Patch Patch

Alternative take:

- Lean v1/v2*

* <https://katemats.com/blog/lean-software-development-build-v1s-and-v2s>

OODA



<https://upload.wikimedia.org/wikipedia/commons/3/3a/ODA.Boyd.svg>