KUBERNETES

Wojciech Barczyński - Codility_

STORY

Kubernetes + Go + ...

- **Codility_** - remote recruitment platform
- **SMACC/Hypatos** - Fintech / ML
- **Lyke** - Mobile Fashion app / ML

Goal: continuous deployment

- Development: max 10 min
  Staging: max 15 min
  Production: max 20 min
- X to XX changes to production
- Infrastructure should be invisible

Goal: continuous deployment

- Frequent releases
- Higher dev satisfaction
- Less (big) bugs
- Faster feature delivery

see: Accelerate: The Science of Lean Software and DevOps

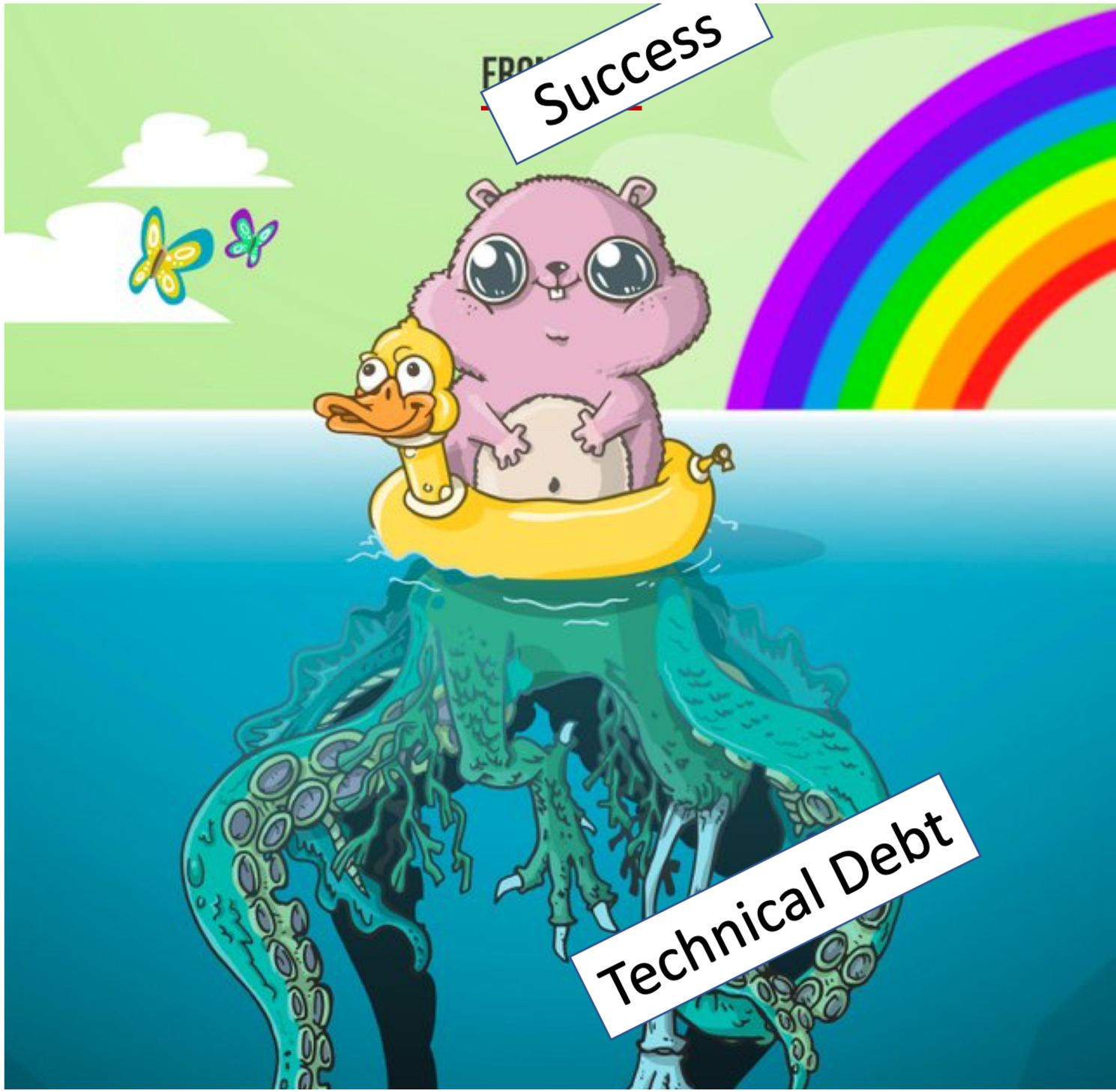
Slow delivery

Continuous Deployment?

Fear

Frustration

XX% Idle Machines



2016

SALTSTACK

ANSIBLE

CHEF

puppet

AWS
CloudFormation

HashiCorp
Terraform

Black (Blue) Box

Infrastructure (almost) invisible

Easy* Continuous Deployment

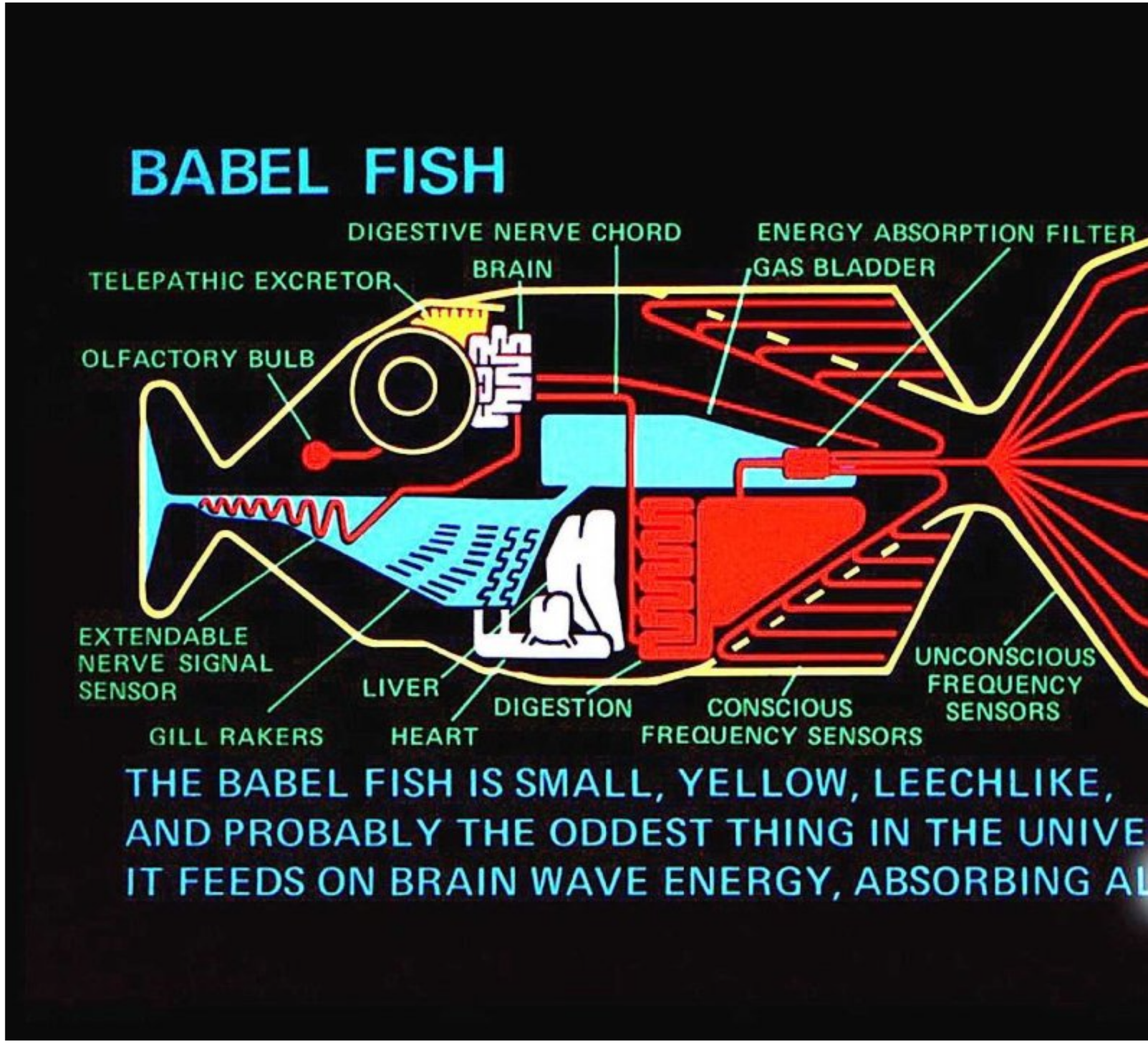
https://en.wikipedia.org/wiki/File:Dr_Who_(316350537).jpg

Common

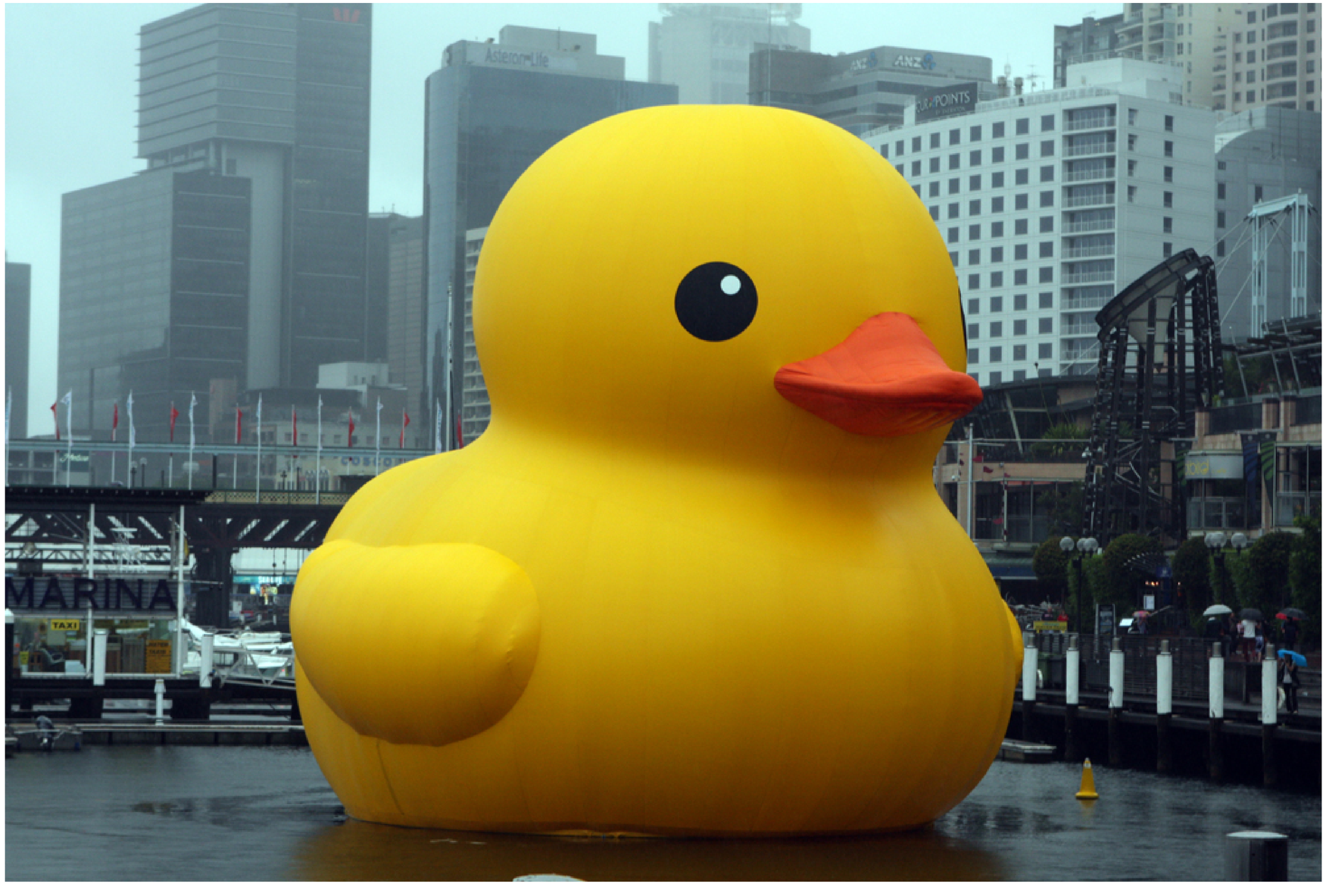
Language

Artifacts

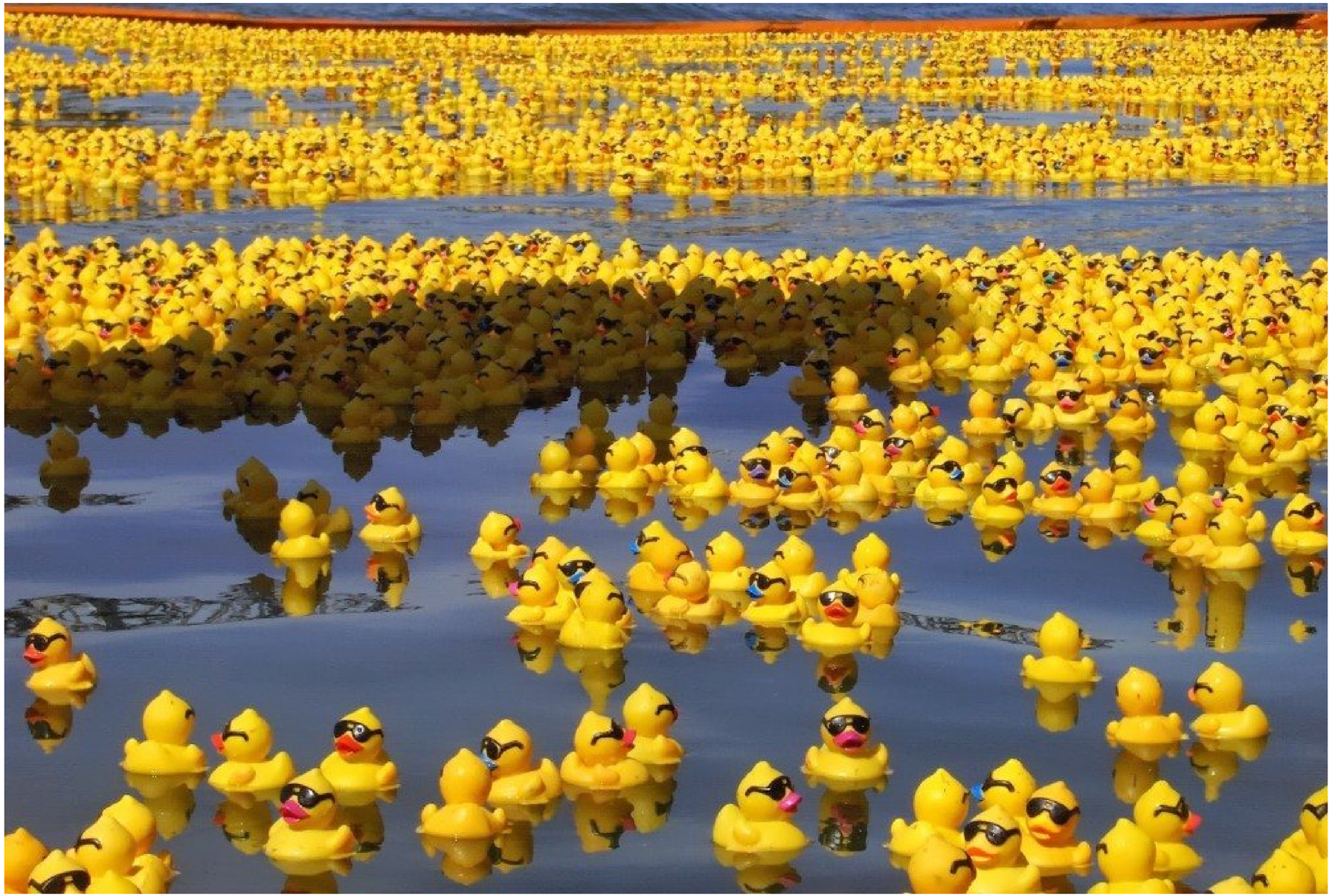
Platform



From monolit



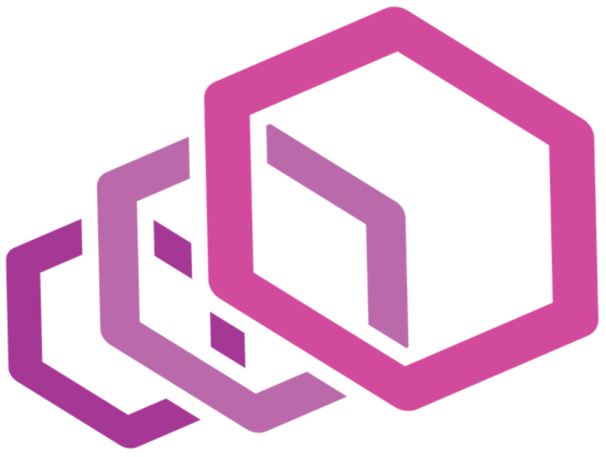
To {micro,macro}services



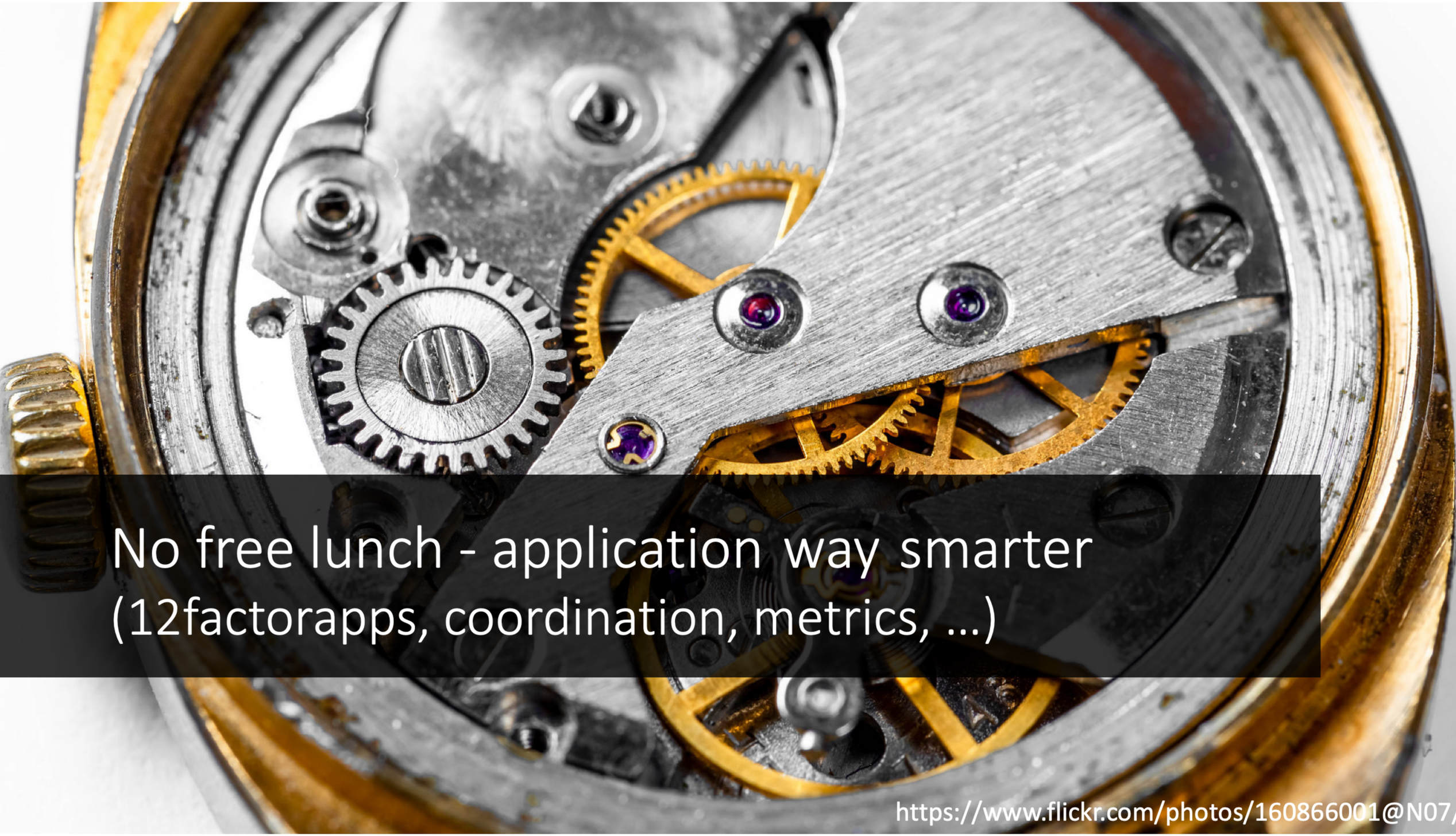
Learn-as-you-go

1. Deploy Cloud-Native app
2. Make a Hell of Mistakes
3. Get it right or Postpone

traefik



envoy



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

https://www.flickr.com/photos/160866001@N07/

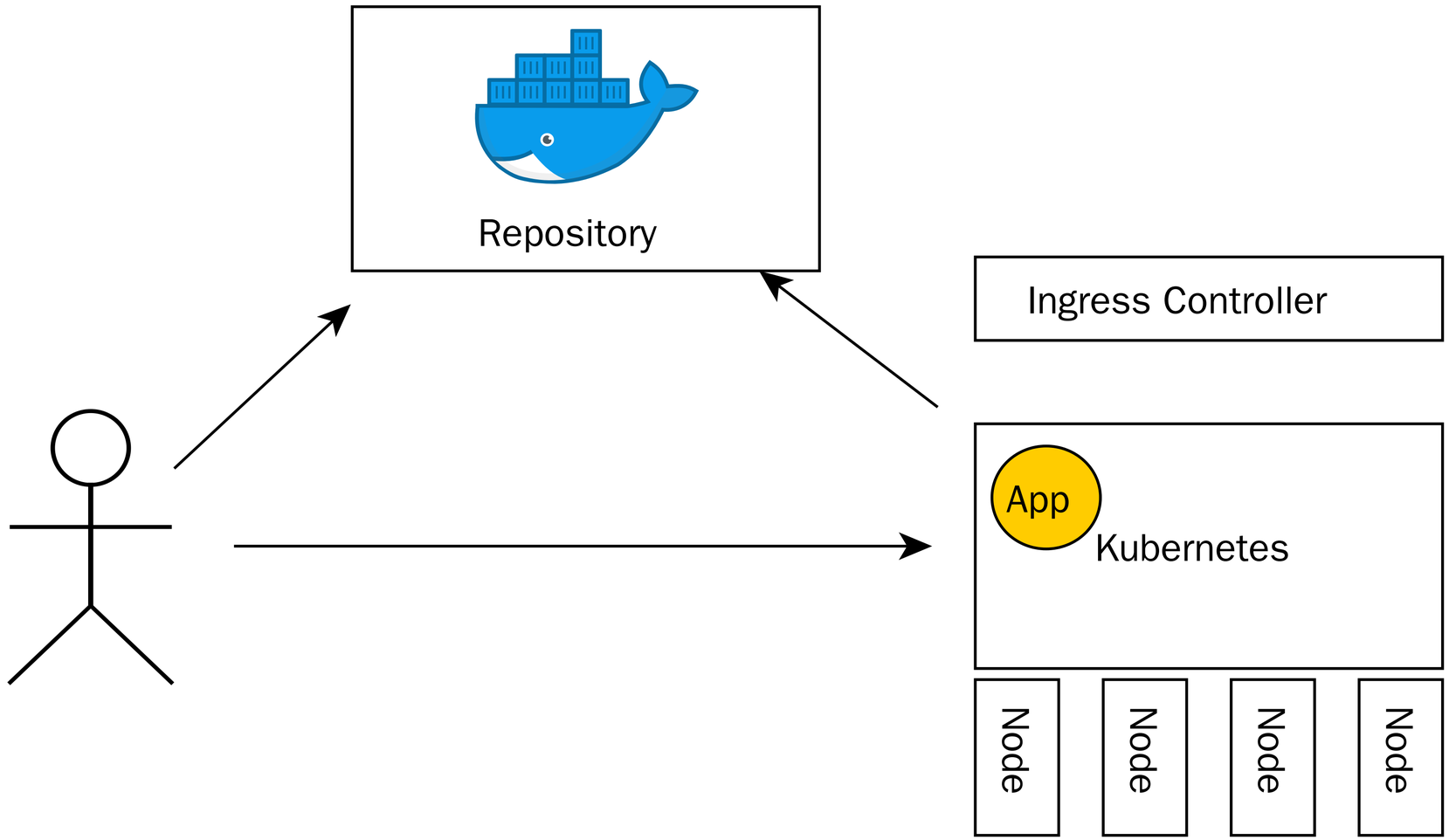
Kubernetes

- Container management
- Battery for 12factor apps
- Decople dev and admin work

Battery for 12factor apps

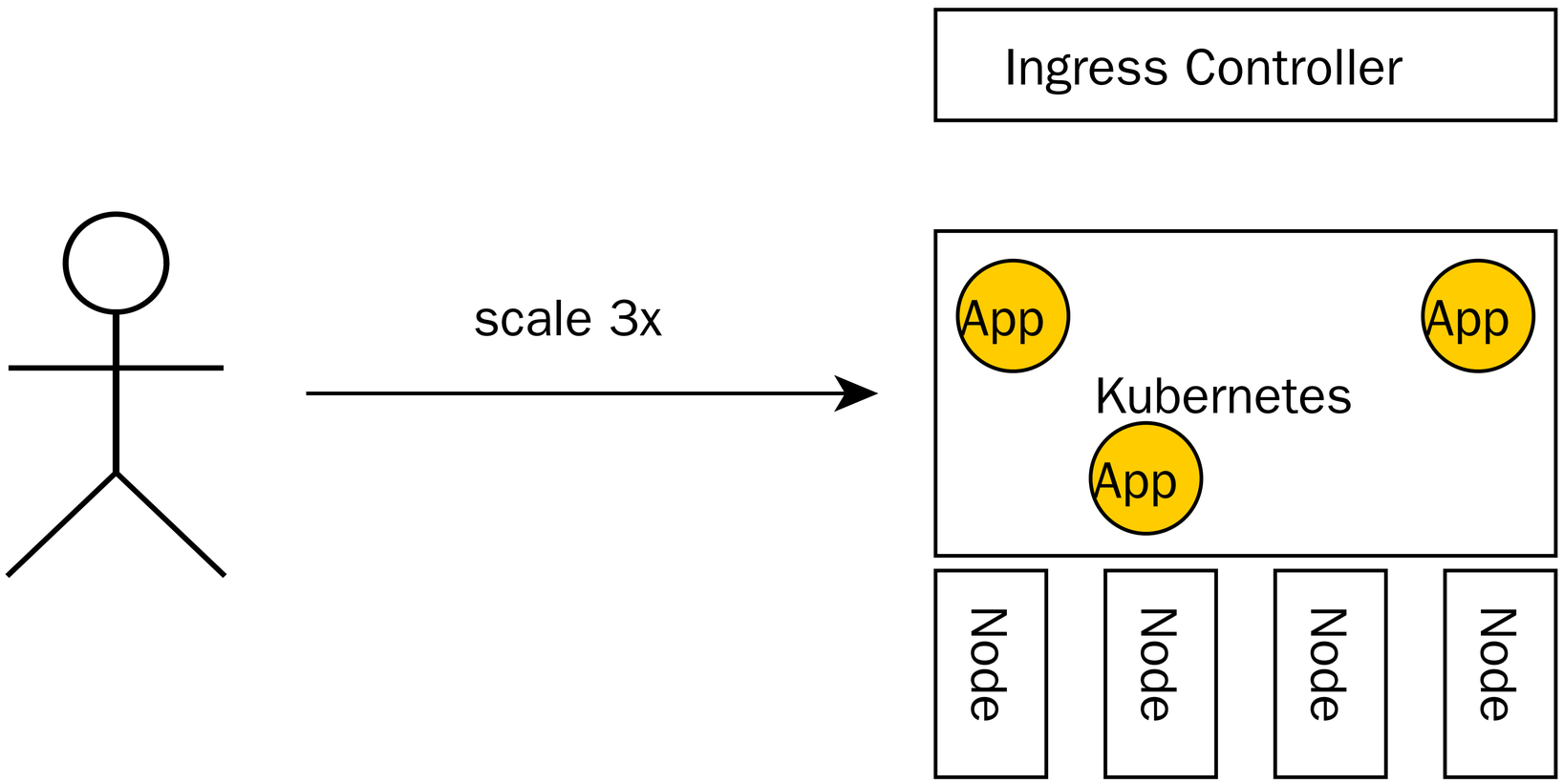
- Service Discovery
- Load Balancing
- Automatic restarts
- Lingua Franca

KUBERNETES



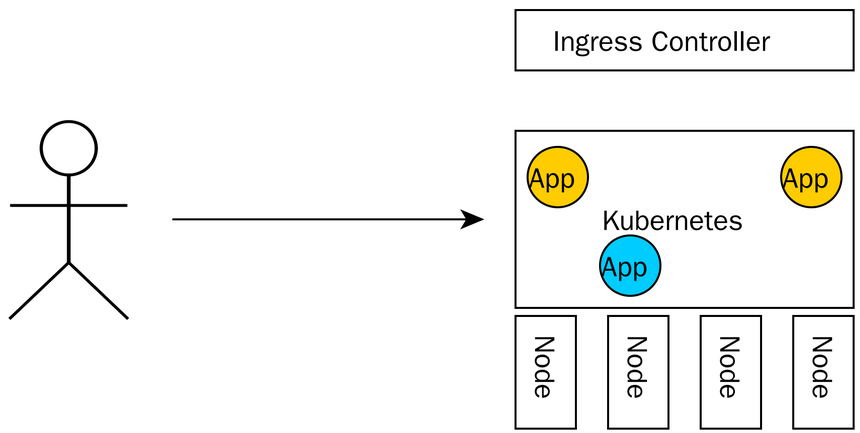

```
make docker_push; kubectl create -f app-srv-dpl.yaml
```

Scale up! Scale down!

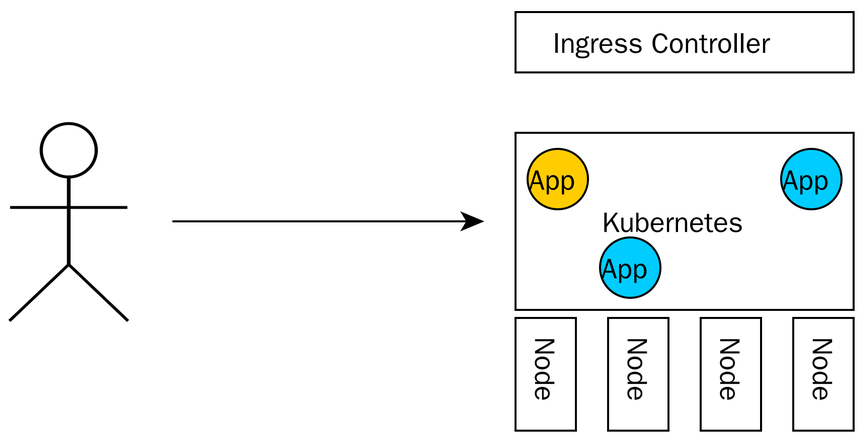



```
kubectl --replicas=3 -f app-srv-dpl.yaml
```

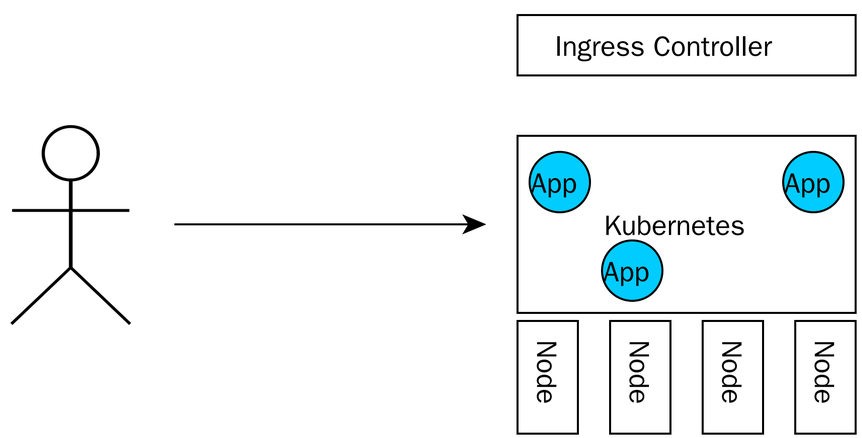
Example: rolling updates



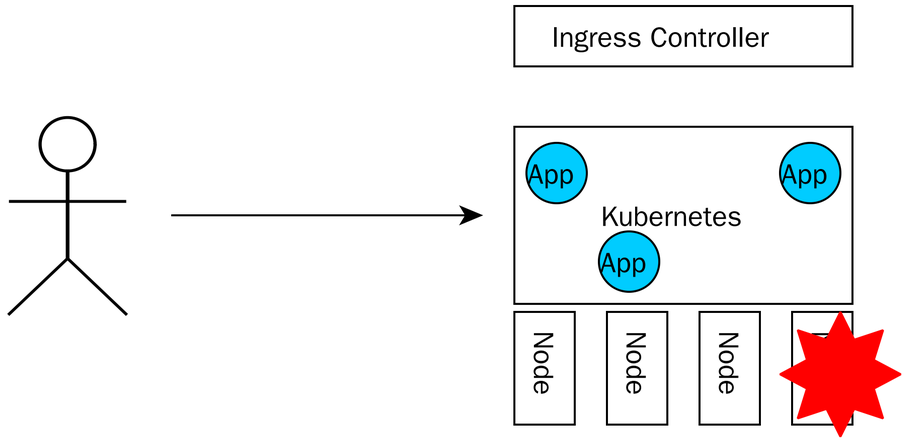
Example: rolling updates



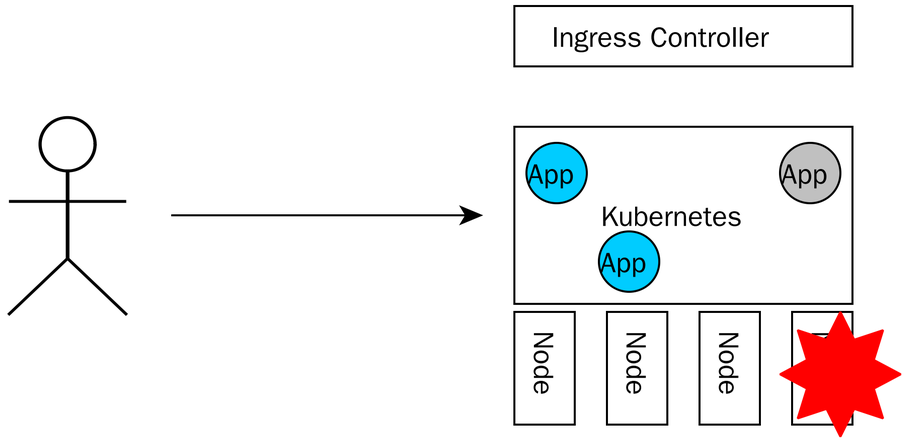
Example: rolling updates



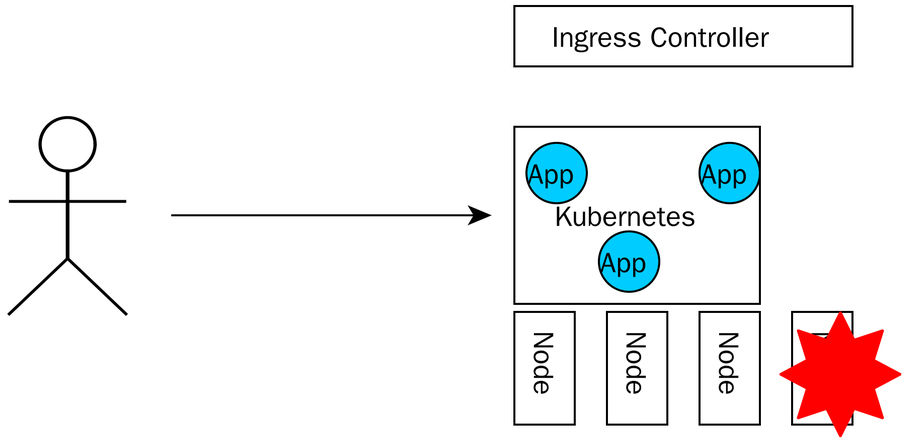
Resilient infra



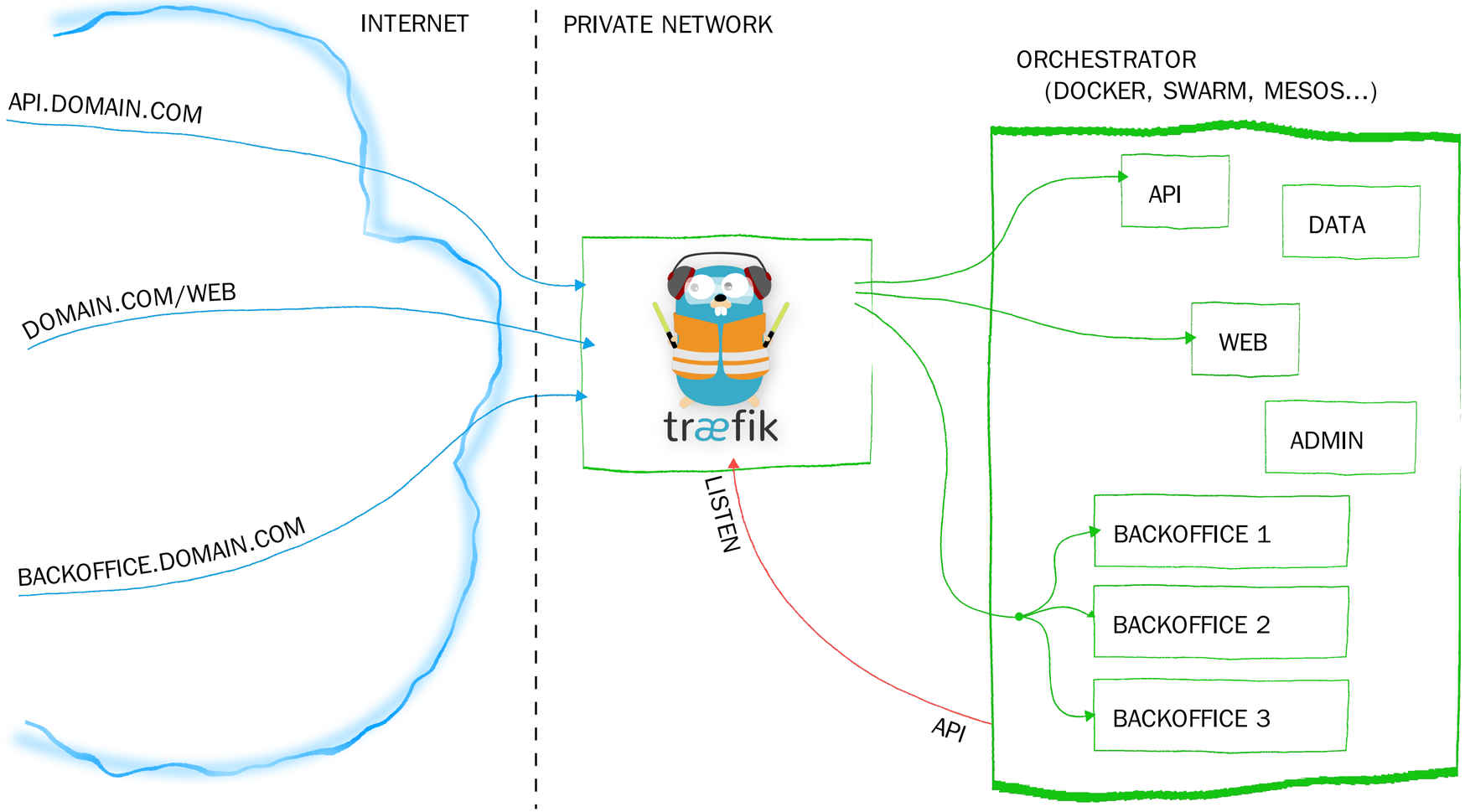
Resilient infra



Resilient infra



Ingress



Pluggable ingress controller.

Ingress

Integration with Cloud providers

- LoadBalancer will be provisioned from K8S

Ingress

Pattern	Target App Service
api.example.io/v1/users	users-v1
api.example.io/v2/users	users-v2
example.io	web

Patterns/Rules support wide routing options.

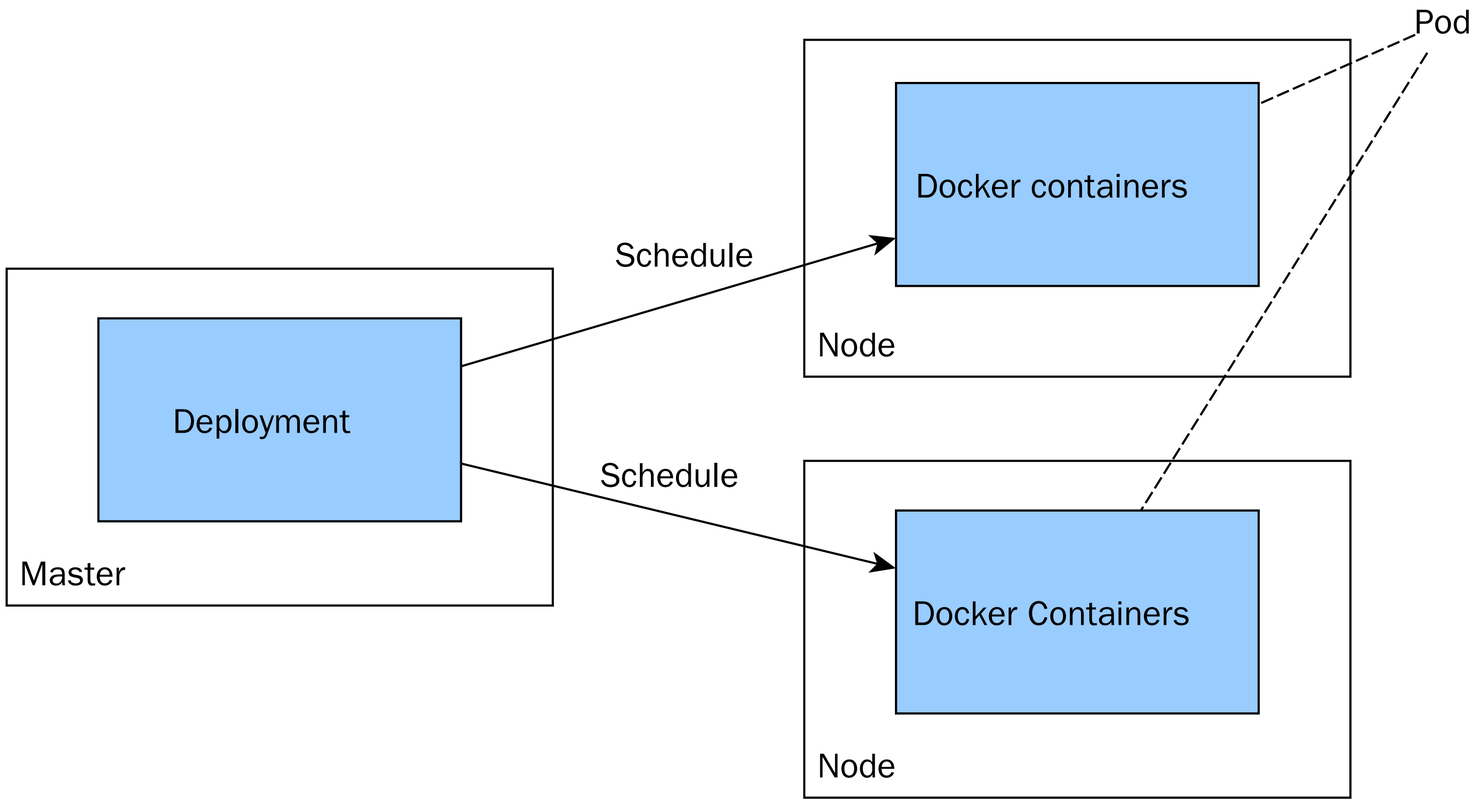
Basic concepts

Name	Purpose
Ingress	Customer interface
URL to Service	

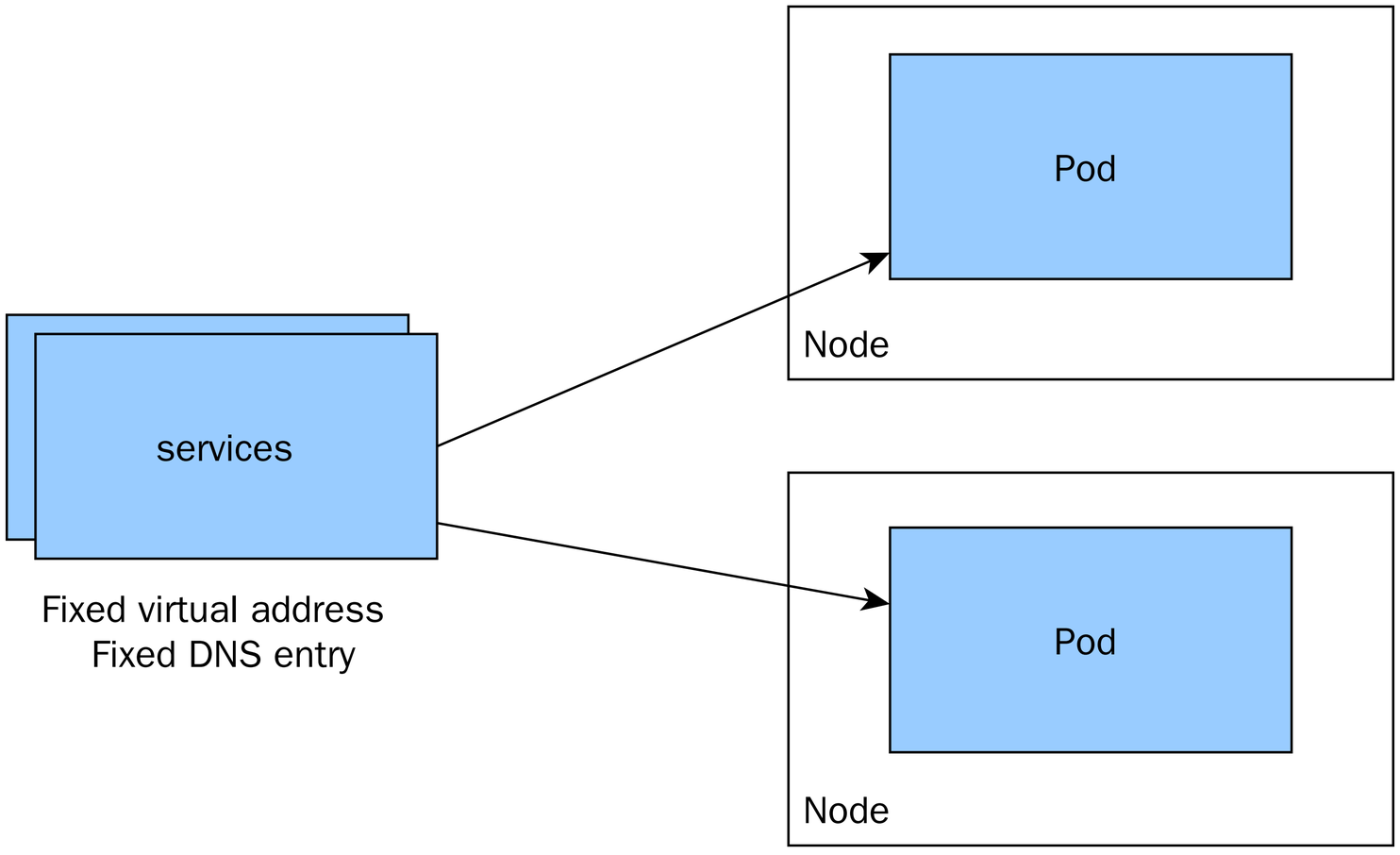
Basic concepts

Name	Purpose	
Service	Interface	Entry point (Service Name)
Deployment	Factory	How many pods, which pods
Pod	Implementation	1+ docker running

Deployment and pods



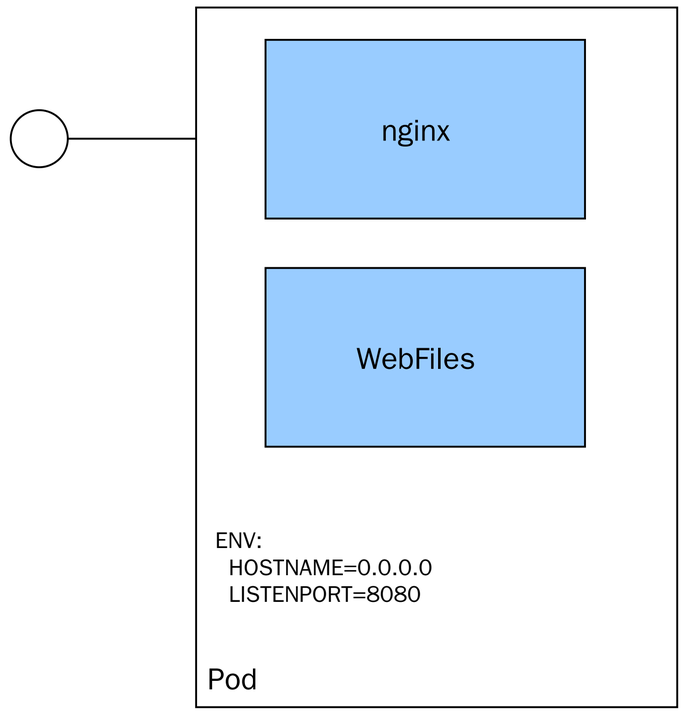
Service and pods



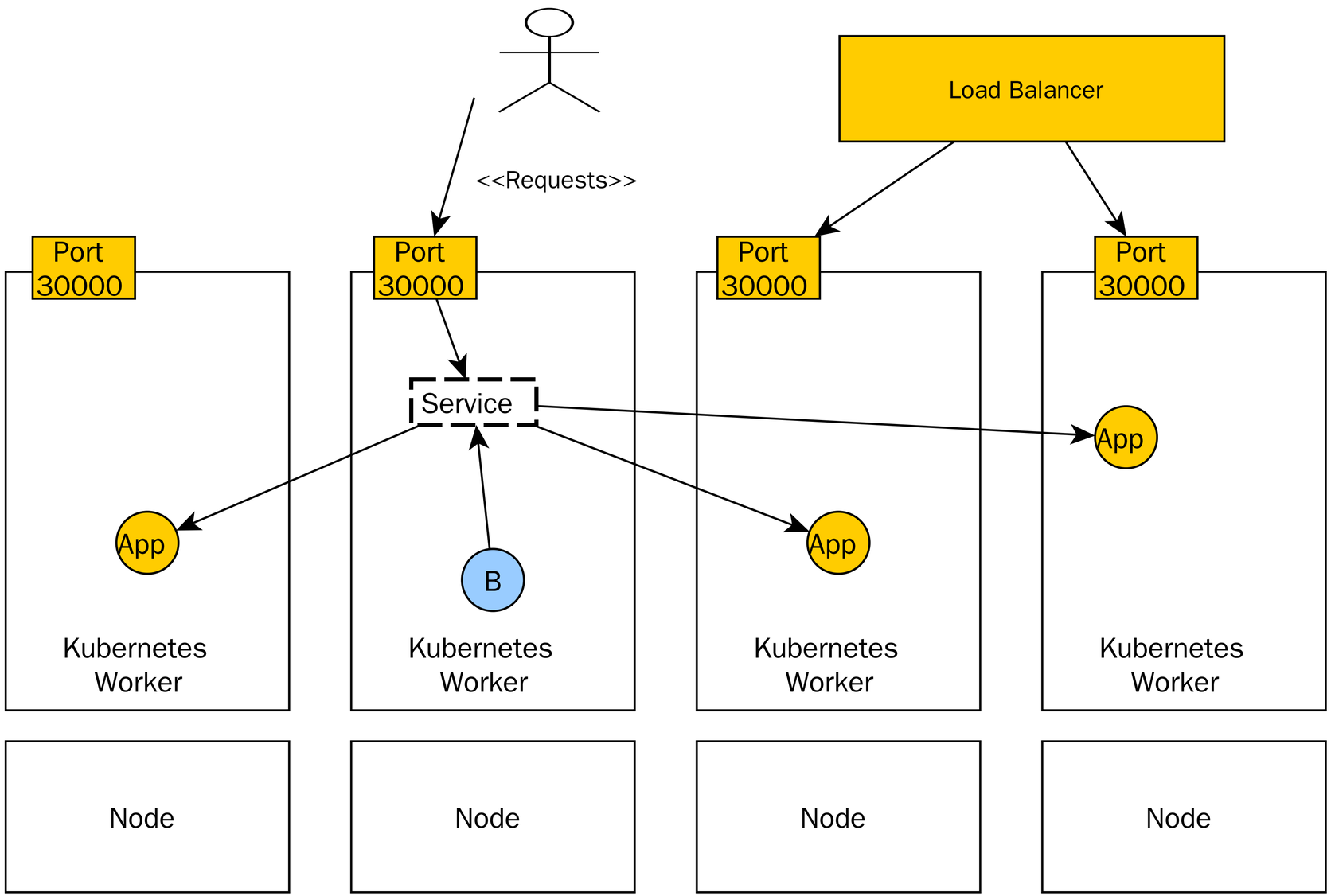
Service matches pods based on labels

Pods

- See each other on localhost
- Live and die together
- Can expose multiple ports



Load-balancing



Service Discovery and labeling

- names in DNS:
  `curl http://users/list`
- labels:
  `name=value`
- annotations:
  `prometheus.io/scrape: "true"`

Service discovery

- loosely couple components
- auto-wiring with logging and monitoring
- drop-in installation (traefik, prometheus, ...)

All hail the git!

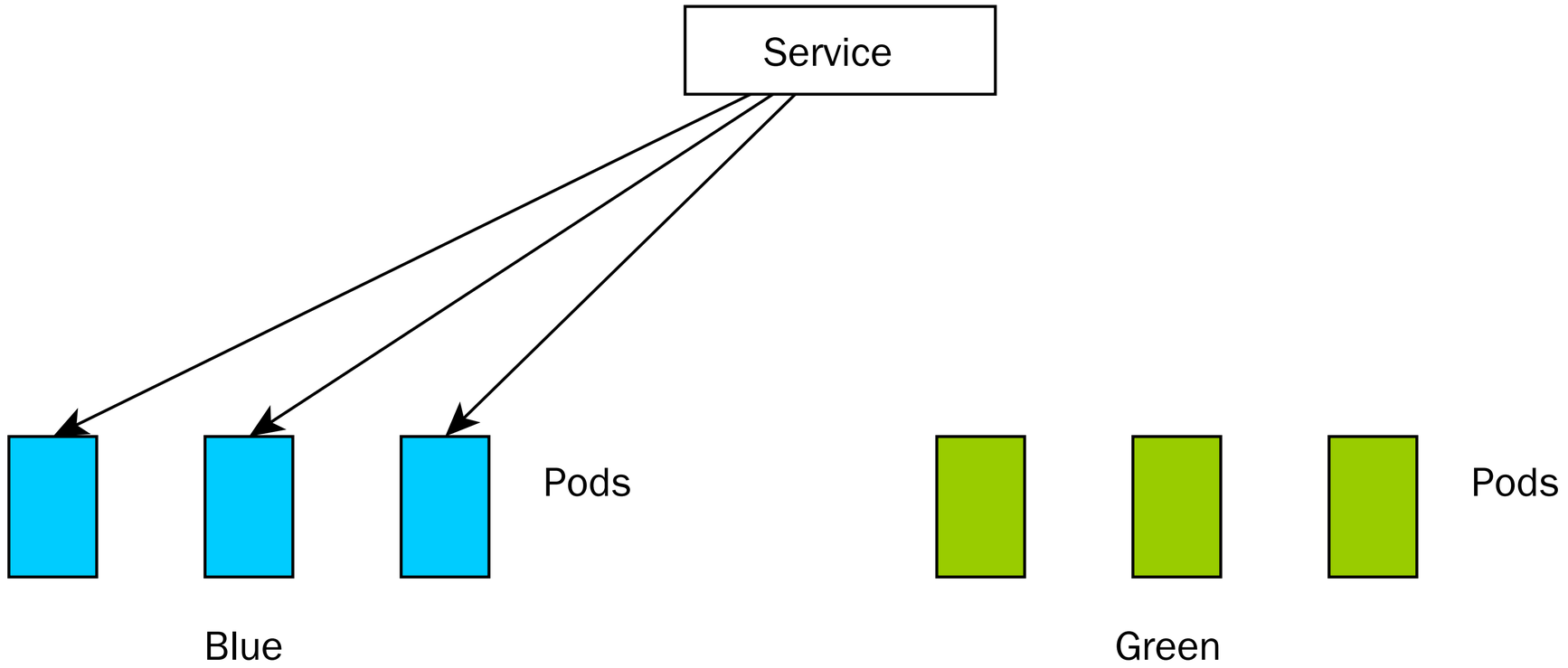
- Yaml
- integration:
  monitoring, alarming, ingress-controller
- ...
- Devs can forget about infra... almost
- DevOps Culture Dream!

Advanced!

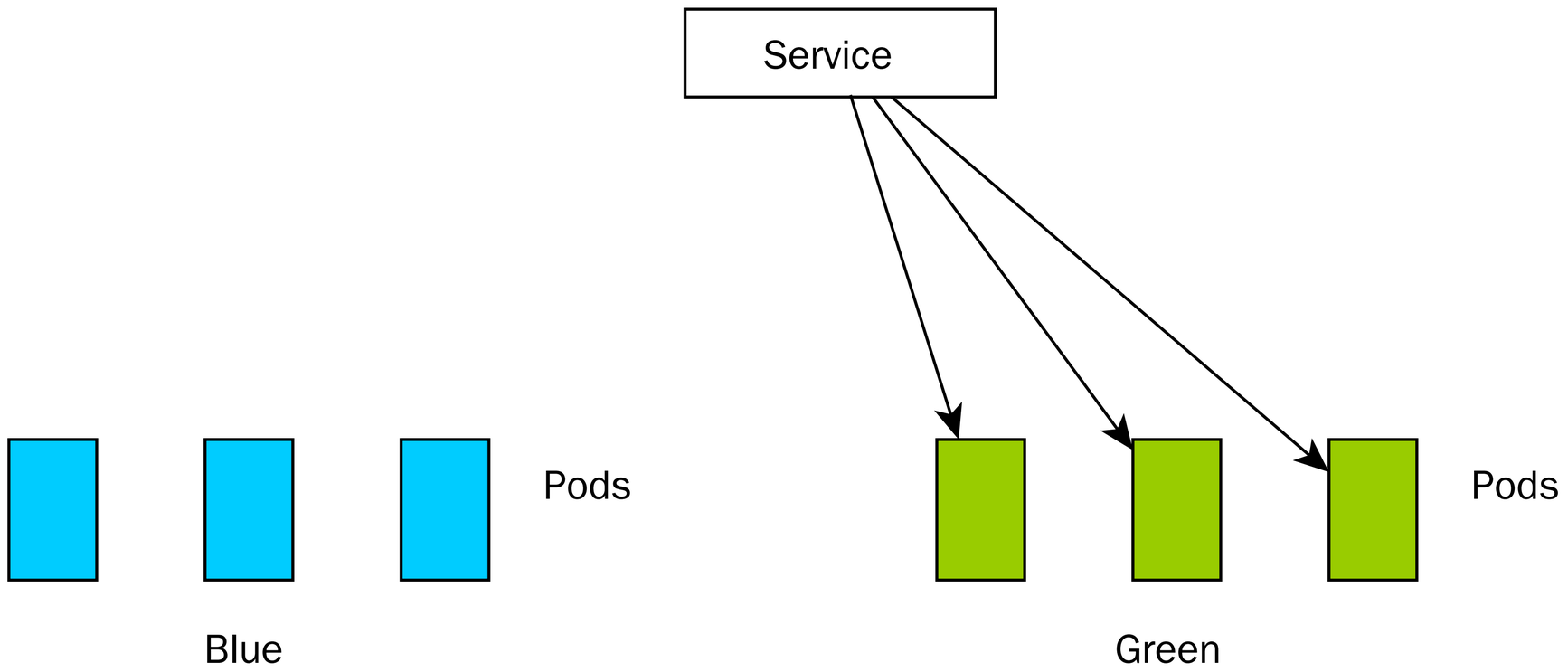
Sophisicated deployment strategies:

- rollout
- blue/green
- canary
- zero downtime deployments

DEMO - GREEN/BLUE



DEMO - GREEN/BLUE



DEMO - GREEN/BLUE

```
kubectl patch service api-status \
    -p '{"spec":{"selector": {"label": "green"} }}'
```

DEMO - GREEN/BLUE

```
kubectl patch service api-status \
    -p '{"spec":{"selector": {"label": "blue"} }}'
```

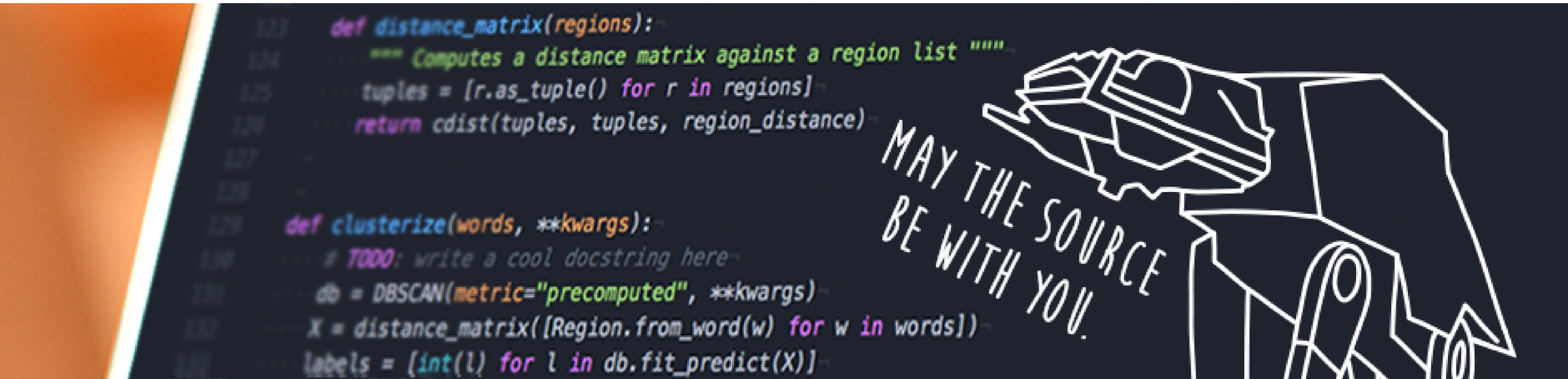
Effective Kubernetes

- Start with small iterations
- **Learn as-you-go**
- Keep Kubernetes Understable

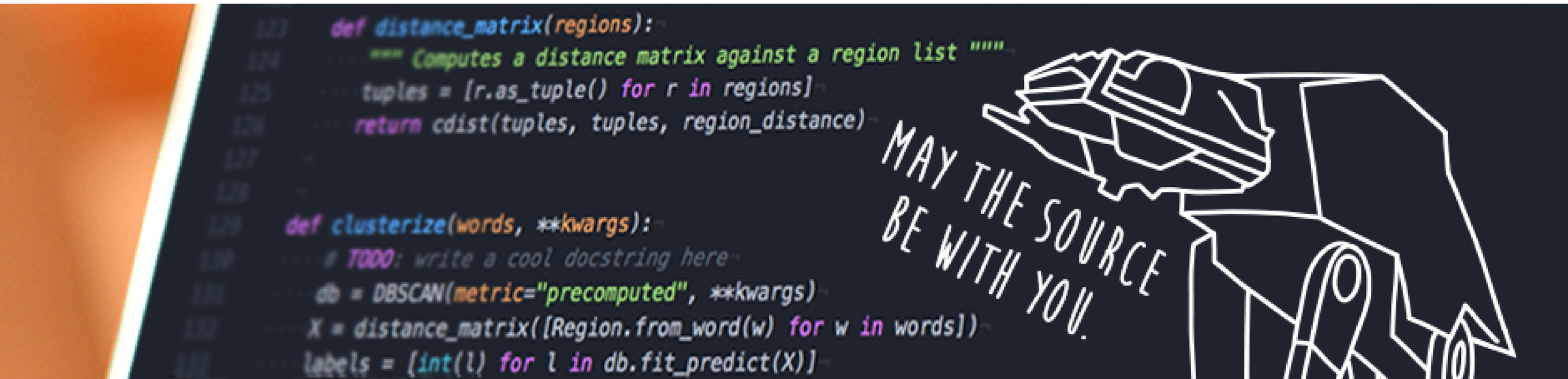
Effective Kubernetes

- Move configuration to runtime
- Do not terrorize your devs with K8S
- No free lunch... app must be smarter

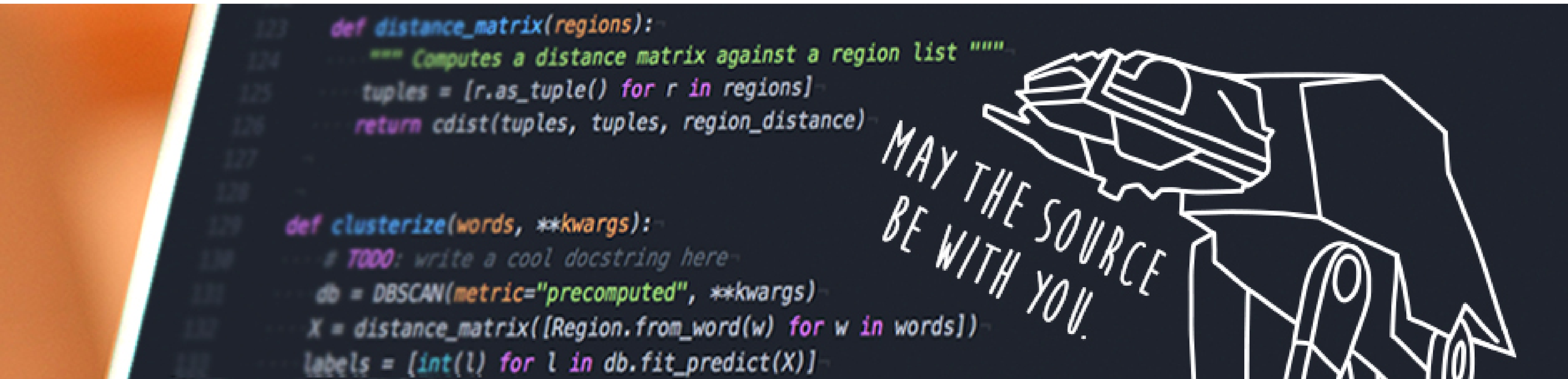
THANK YOU. QUESTIONS?



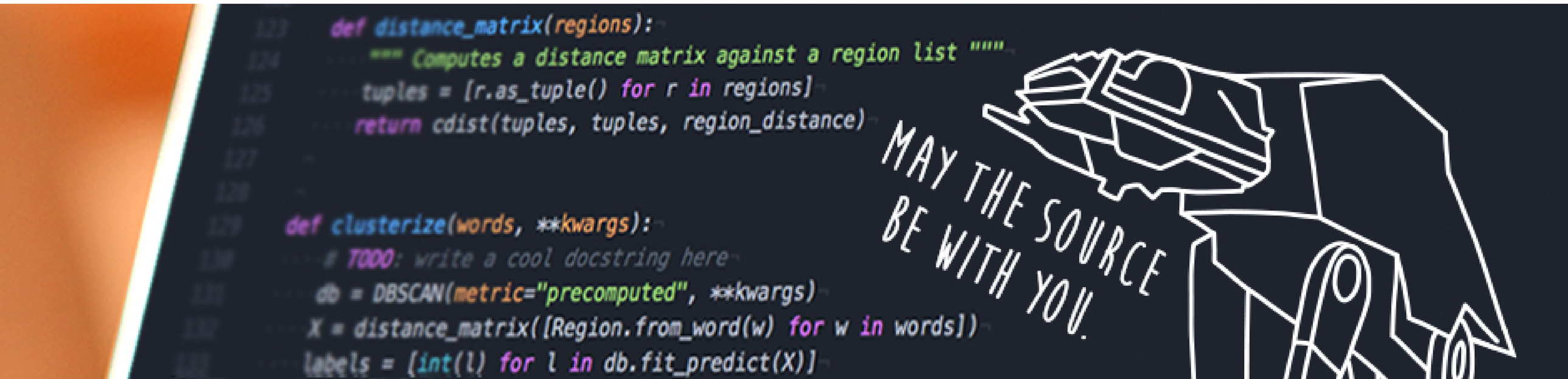
https://github.com/wojciech12/workshop_kubernetes_and_cloudnative



https://github.com/wojciech12/talk_zero_downtime_deployment_with_kubernetes



BACKUP SLIDES



INFRA TOOLS

- Prometheus + AlertMnager + Grafana
- Traefik
- Kafka - Yolean/kubernetes-kafka
- Vault on Etcd - banzaicloud/bank-vaults

DATA STORES

- Kafka - Yolean/kubernetes-kafka
- Etcd - coreos/etcd-operator
- DB: PSQL and Mongo

BACKUPS:

- old-school backups with ARK and Restic
- replications across clouds

1. CLEAN UP

- Single script for repo - Makefile [1]
- Resurrect the README

[1] With zsh or bash auto-completion plug-in in your terminal.

2. GET BACK ALL THE KNOWLEDGE

Extract from:

- Puppet, ... ➡ Dockerfile
- Running Instances ➡ Dockerfile, README.rst
- Nagios, ... ➡ README.rst + `checks/`

3. INTRODUCE RUN_LOCAL

- `make run_local`
- A nice section on how to run in README.rst
- with `docker-compose`

The most crucial point.

4. GET TO KUBERNETES

- `make kube_create_config`
- `make kube_apply`
- Generate the yaml files if your envs differ

secrets from gopass (password manager)

5. CONTINUOUS DEPLOYMENT

Travis:

- the same Makefile as dev use

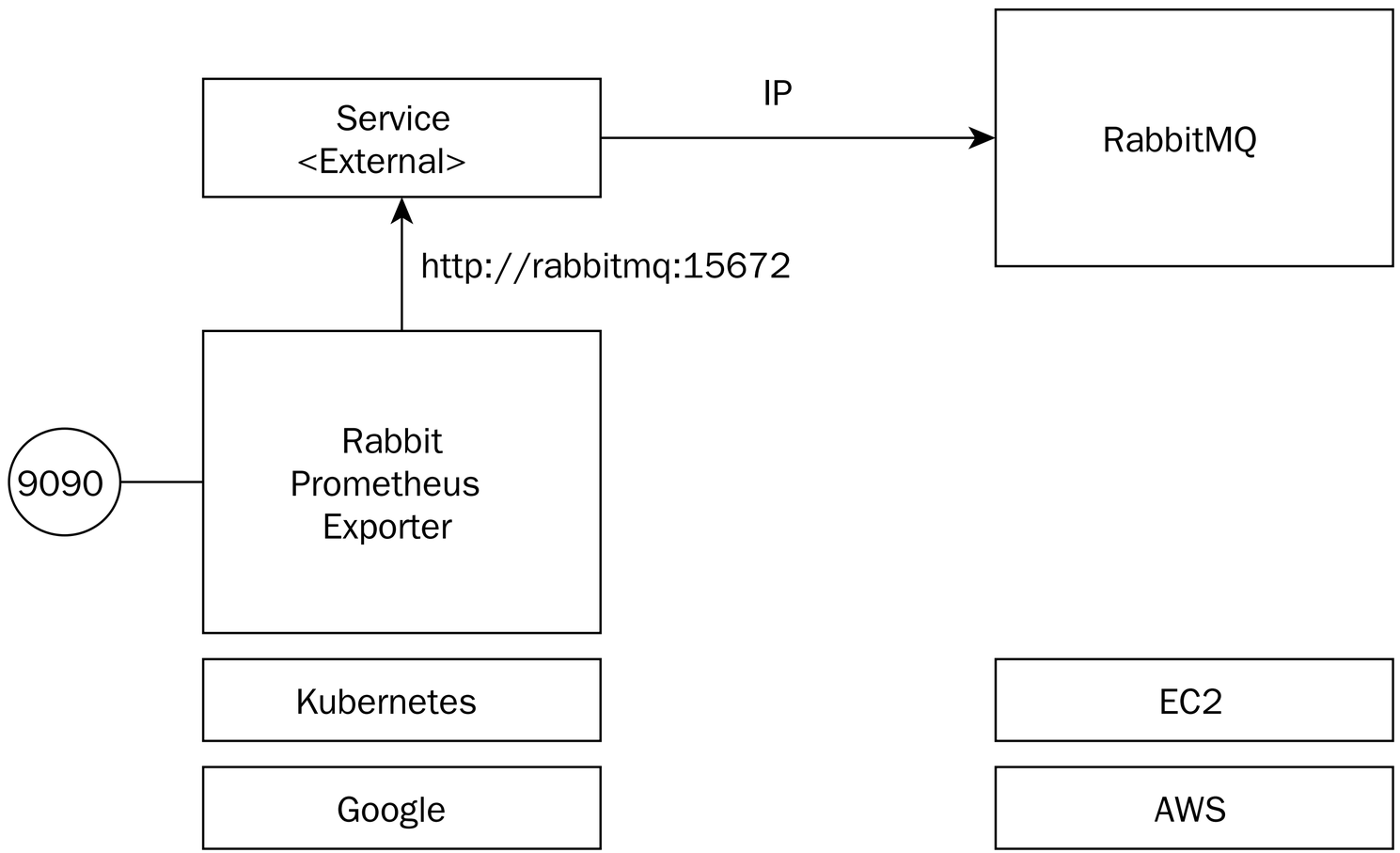
6. KEEP IT RUNNING

Bridge the new with old:

- Use External Services in Kubernetes
- Expose k8s in the Legacy [1]

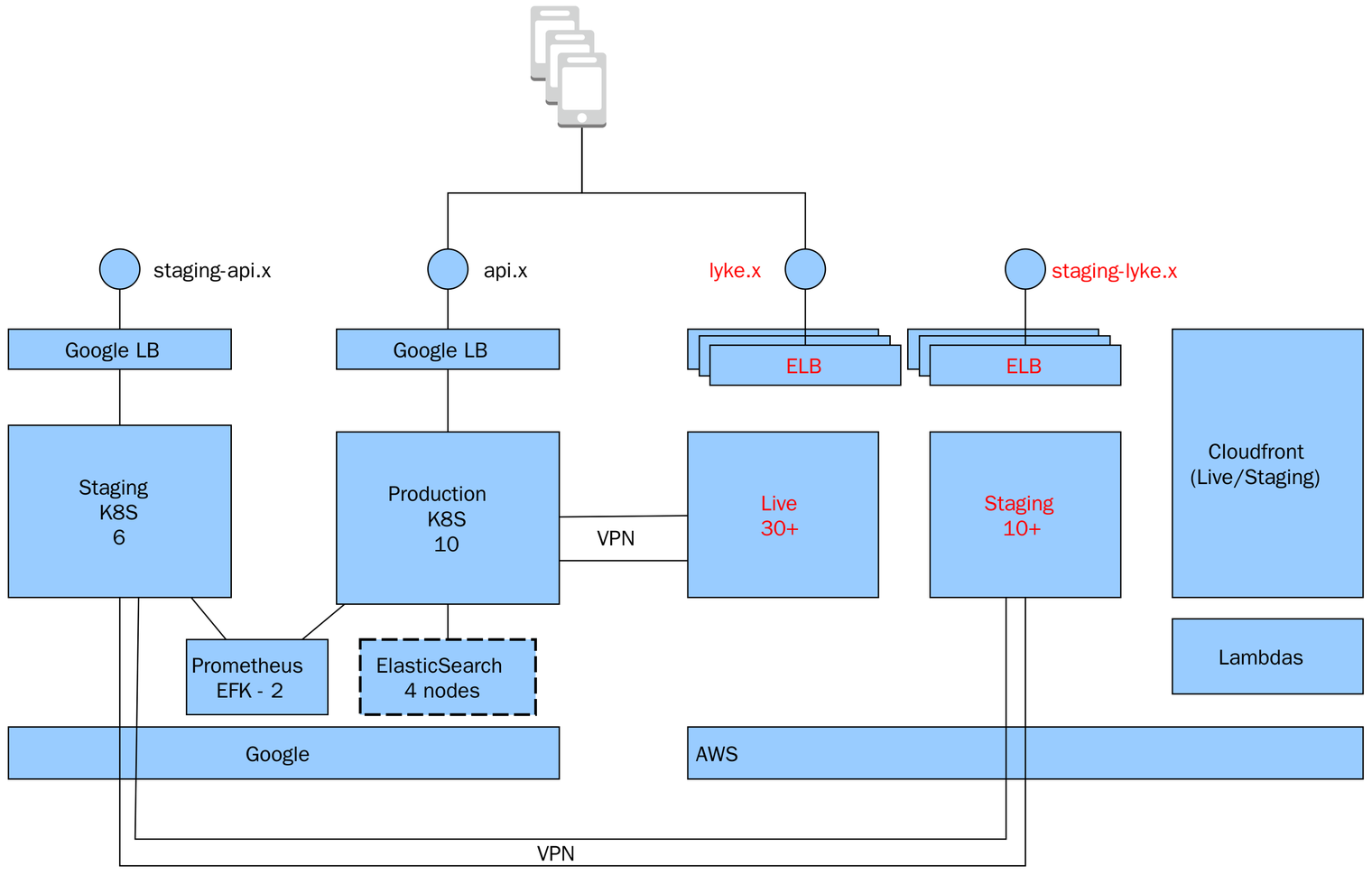
[1] hard coded IP:PORT, considered: K8S events to Consul

Bridge the new with old



Monitor legacy with new stack

Architecture During Migration



7. INTRODUCE SMOKE-TEST

```
TARGET_URL=127.0.0 make smoke_test
TARGET_URL=api.example.com/users make smoke_test
```

8. SERVICE SELF-AWARE

Add to old services:

1. *metrics/*
2. *health/*
3. *info/*

9. MOVE TO MICRO-SERVICES

Offload Legacy:

- Keep the lights on
- New functionality to micro-services

10. GET PERFORMANCE TESTING

- *wrk* for evaluating performance
- load test the real system