

# With Kubernetes, from AWS to Google Cloud Platform.



Wojciech Barczyński ([wojciech.barczynski@smacc.io](mailto:wojciech.barczynski@smacc.io))

# WOJCIECH BARCZYŃSKI

- Senior Software Engineer  
Lead of Warsaw Team - SMACC
- Before:  
System Engineer Lyke
- Before:  
1000+ nodes, 20 data centers with  
Openstack

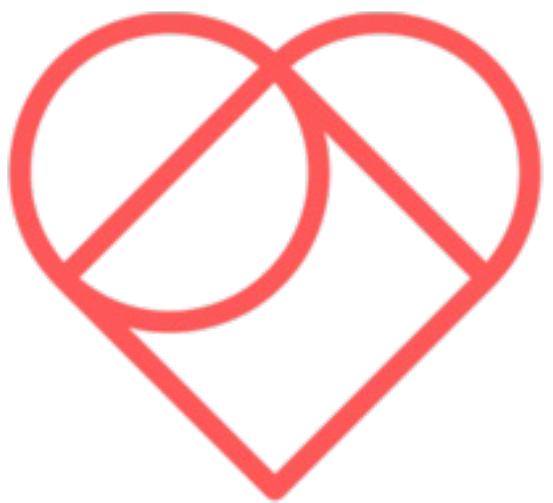


# BACKGROUND

- 10+ Developer and R&D
- E-commerce, Artificial Intelligence, Data Centers
- 1.5y with Kubernetes, 3+ AWS, 1+ GCE  
0+ ProfitBricks
- 3.5y with Openstack, 1000+ nodes, 21 data centers
- I do not like INFRA
- I do not like: puppet, chef, ansible,...

# STORY

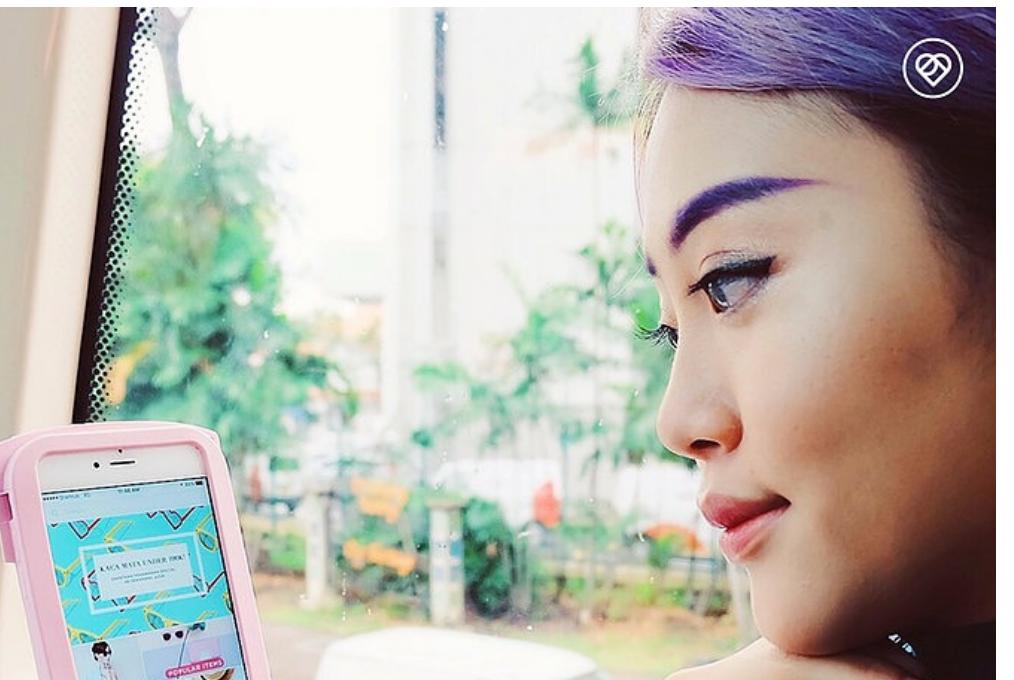
- Lyke - [12.2016 - 07.2017]
- SMACC - [10.2017 - present]



LYKE

# LYKE

- E-commerce
- Mobile-only
- 50k+ users
- 2M downloads
- Top 10 Fashion Apps  
w Google Play Store



# GOOD PARTS

- Fast Growth
- A/B Testing
- Data-driven
- Product Manager,  
UI Designer,  
Mobile Dev,  
and tester - one body



## CHALLENGES 12.2016

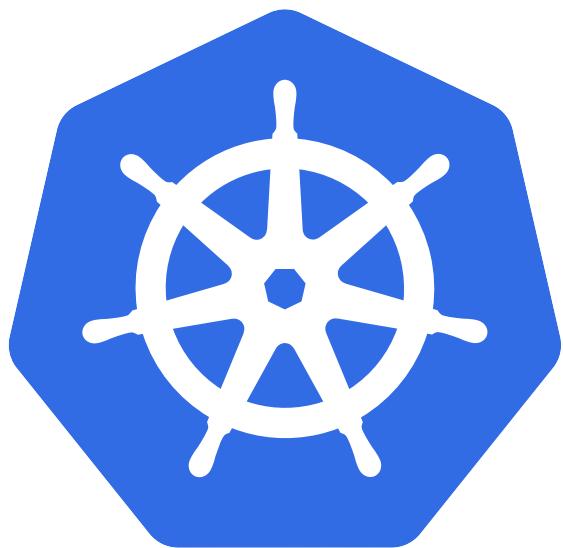
- 50+ VMs in Amazon, 1 VM - 1 App, idle machine
- Puppet, hilarious (manual) deployment process
- fear
- forgotten components
- sometimes performance issues

## APPROACH

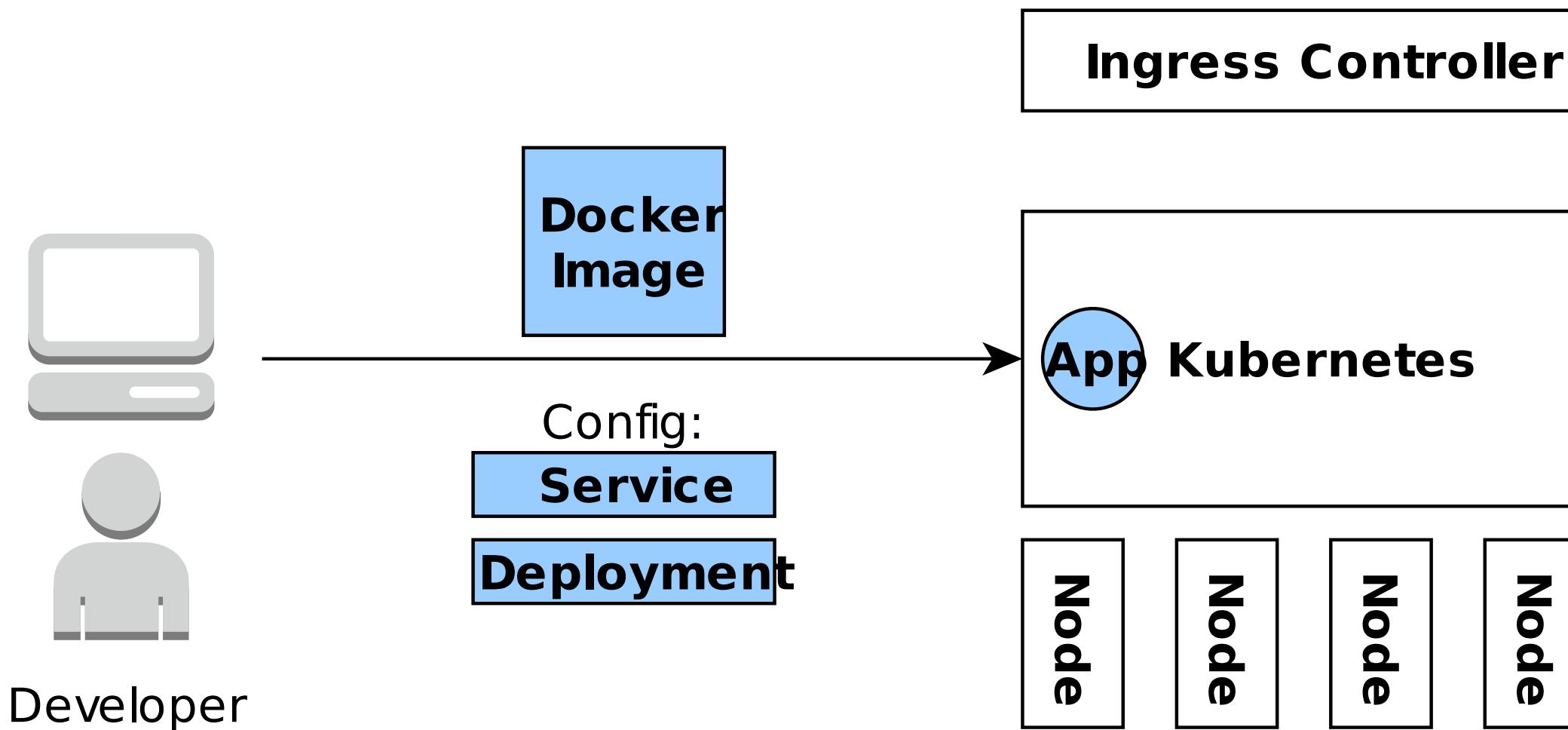
1. Simplify infrastructure
2. Change the Development practices
3. Change the work organization

# WHY KUBERENTES

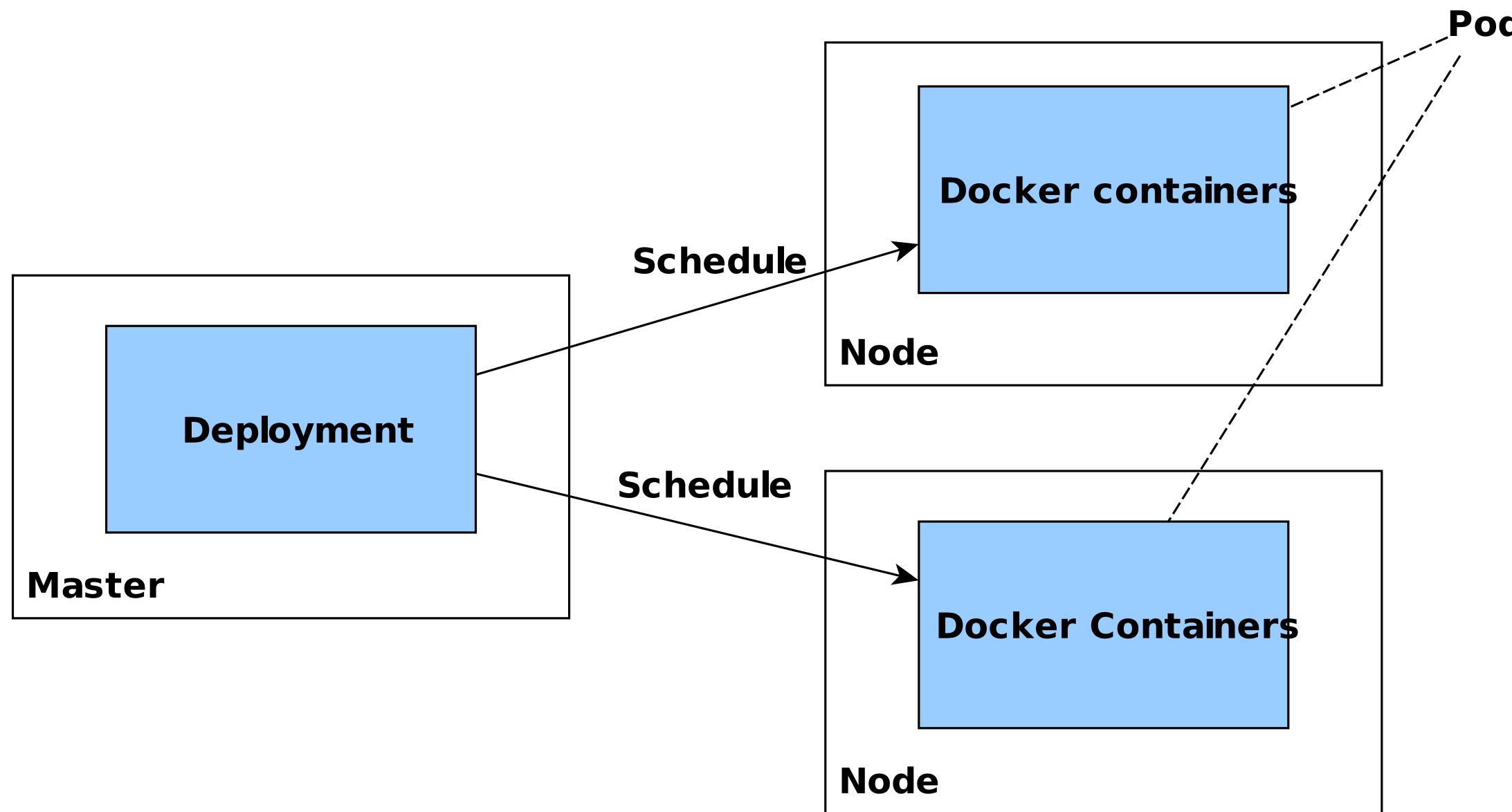
- Data center as a black-box
- Simple Semantic
- Batteries for your 12factor apps
- Meta-data support
- Service discovery
- Cloud-Provider independent, we can always move

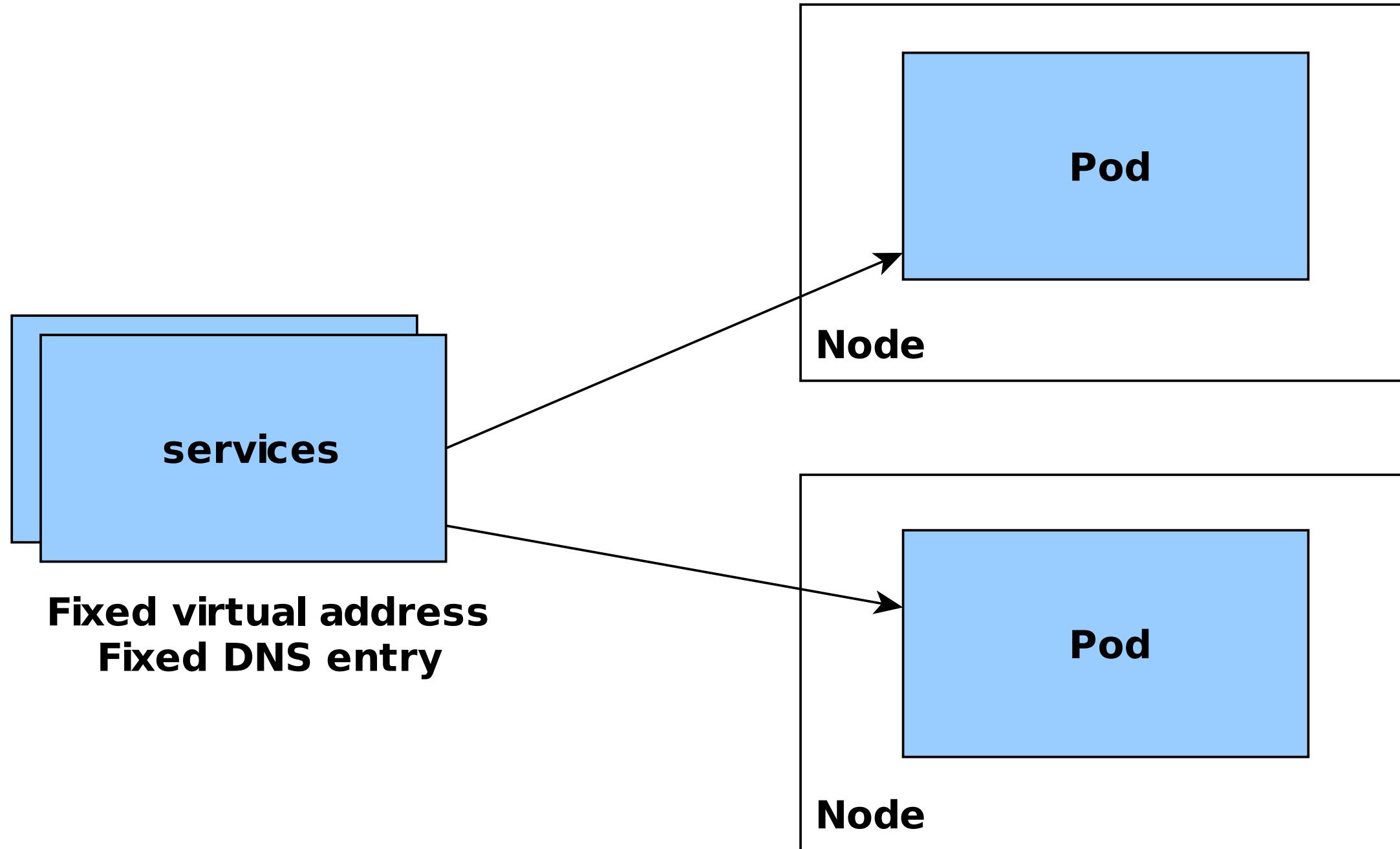


# KUBERNETES



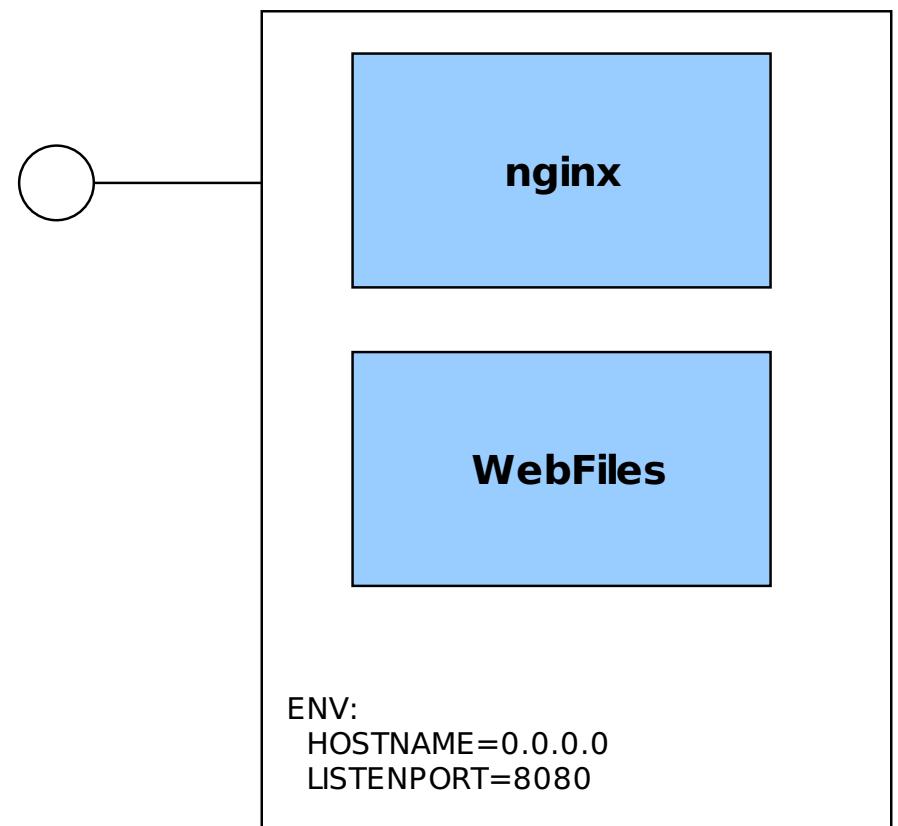
make docker\_push; kubectl create -f app-srv-dpl.yaml



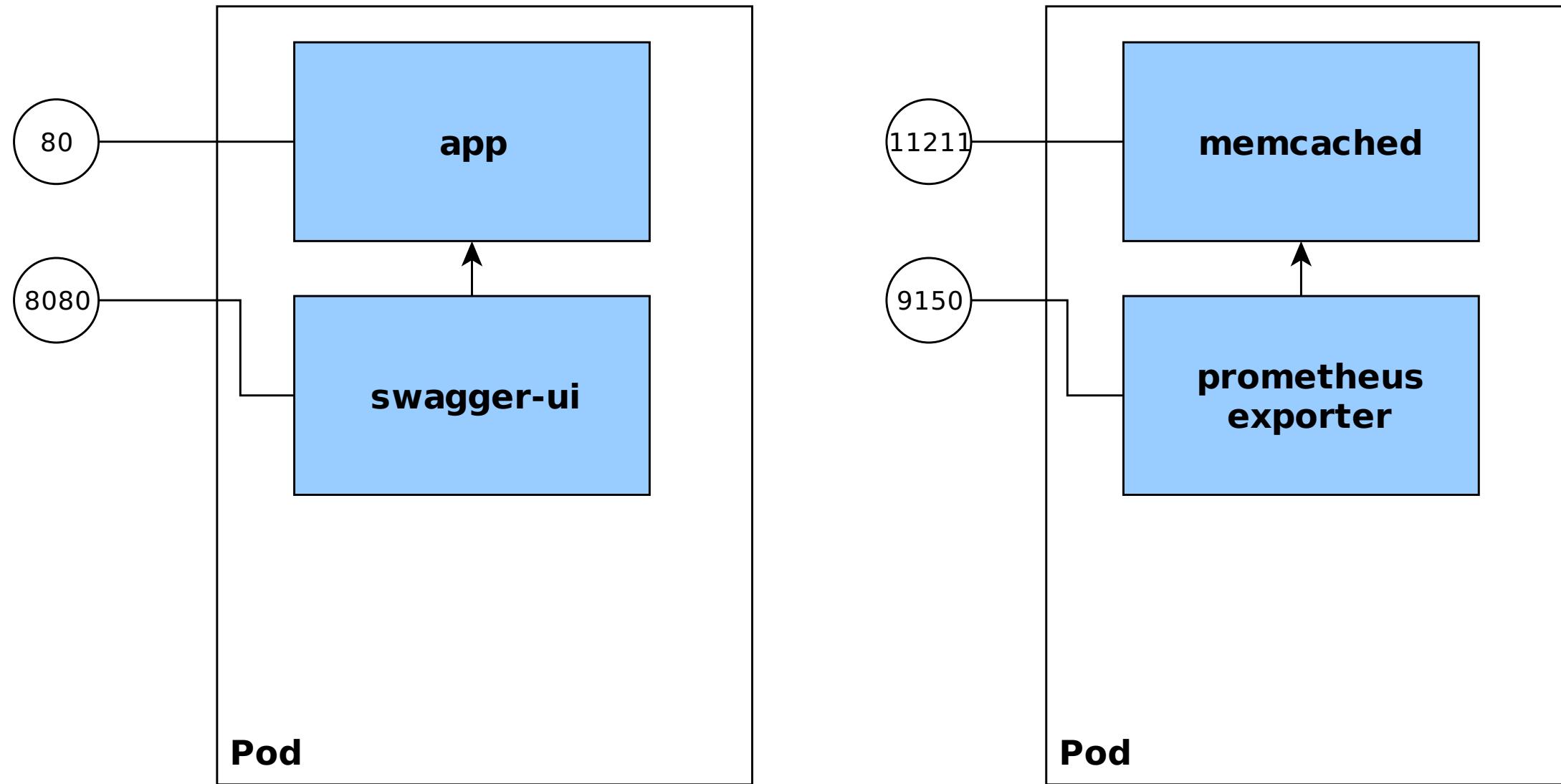


# PODS

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



# SIDE-CARS



## BASIC CONCEPTS

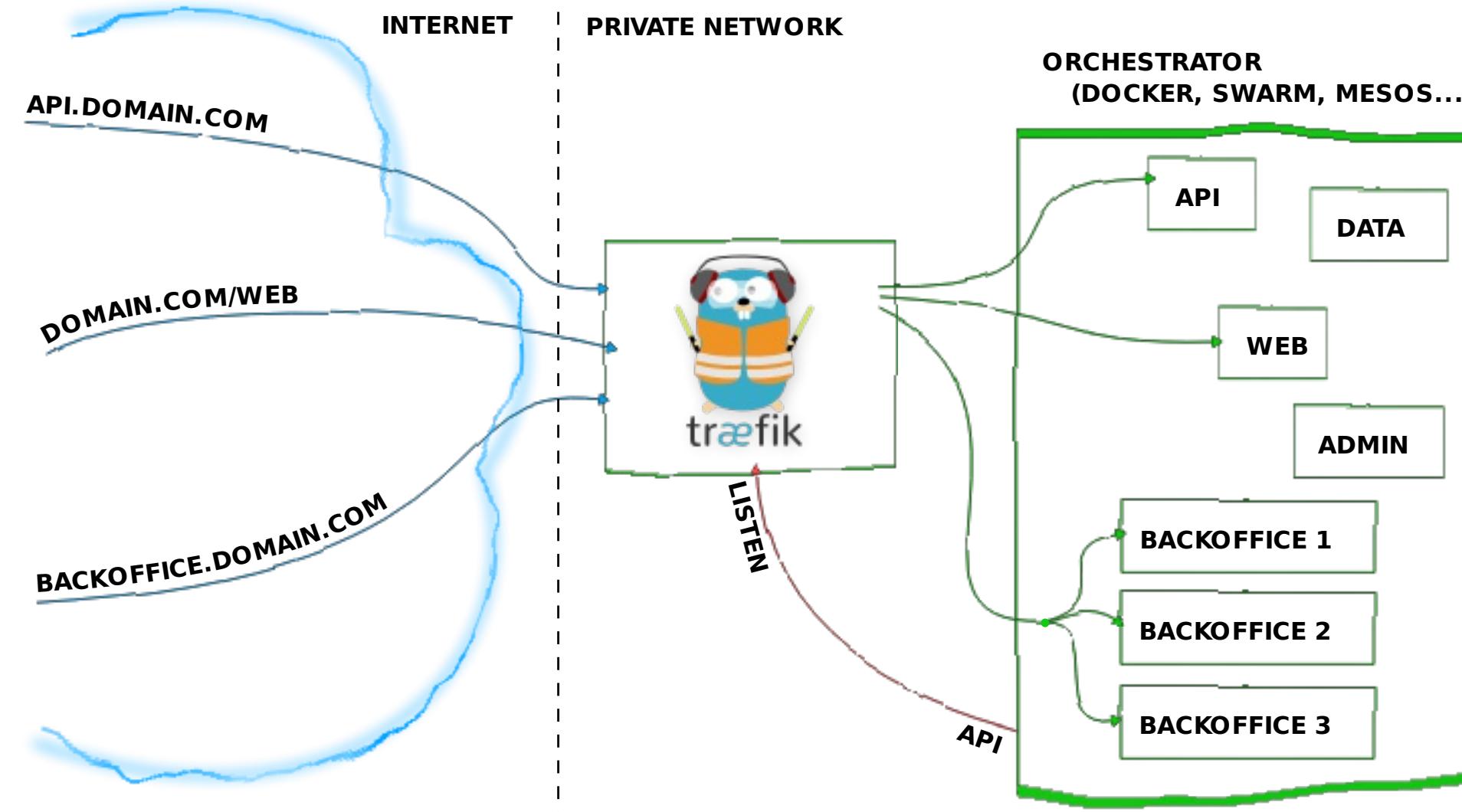
Name	Purpose	
Service	Interface	Entry point (Service Name)
Deployment	Factory	How many pods, which pods
Pod	Implementation	1+ docker running

# INGRESS CONTROLLER

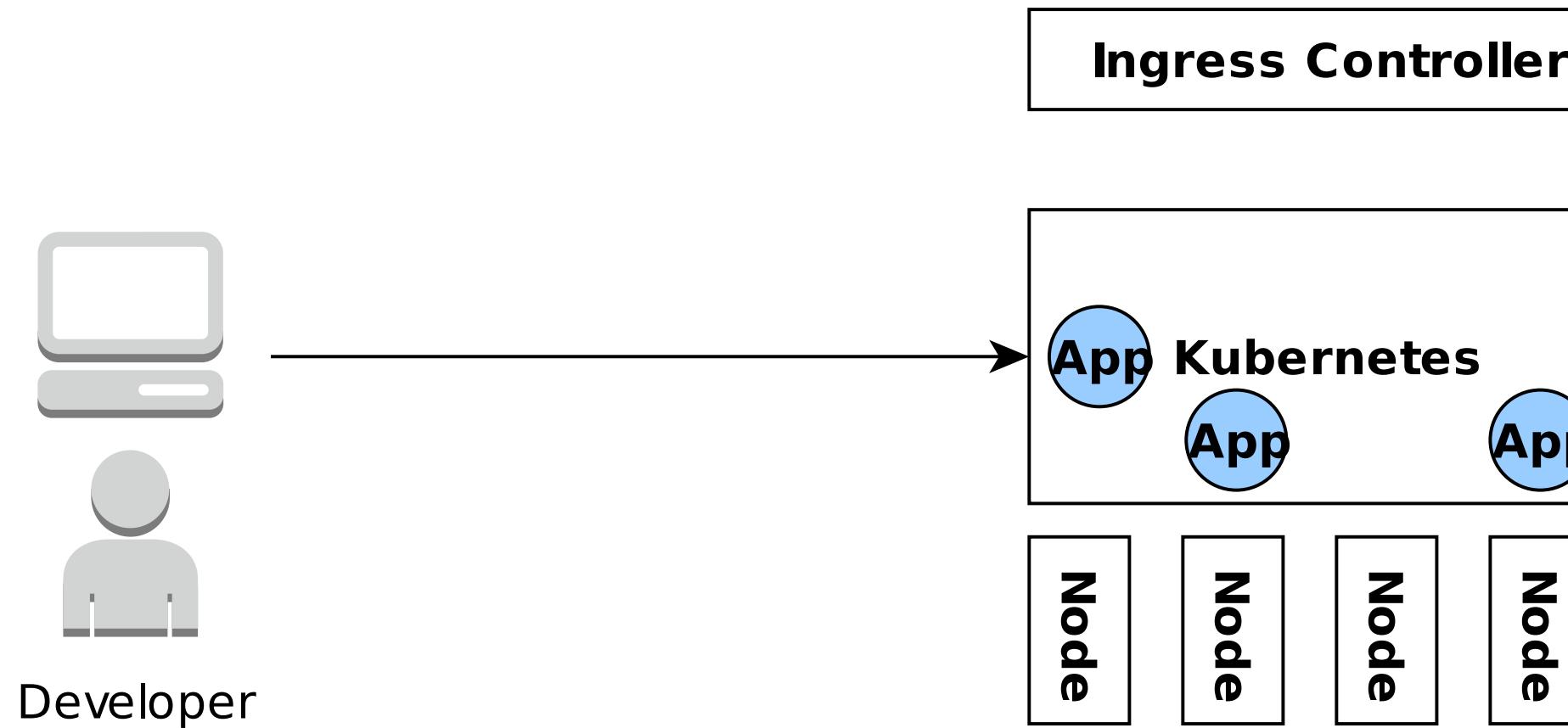
- `api.x.ee/v2/users` → service: `users-v2:80`
- `api.x.ee/v2/users/swagger-ui`  
    → service: `users-v2:8080`
- `api.x.ee` → service: `website`

Every service can have its own ingress config.

# INGRESS CONTROLLER

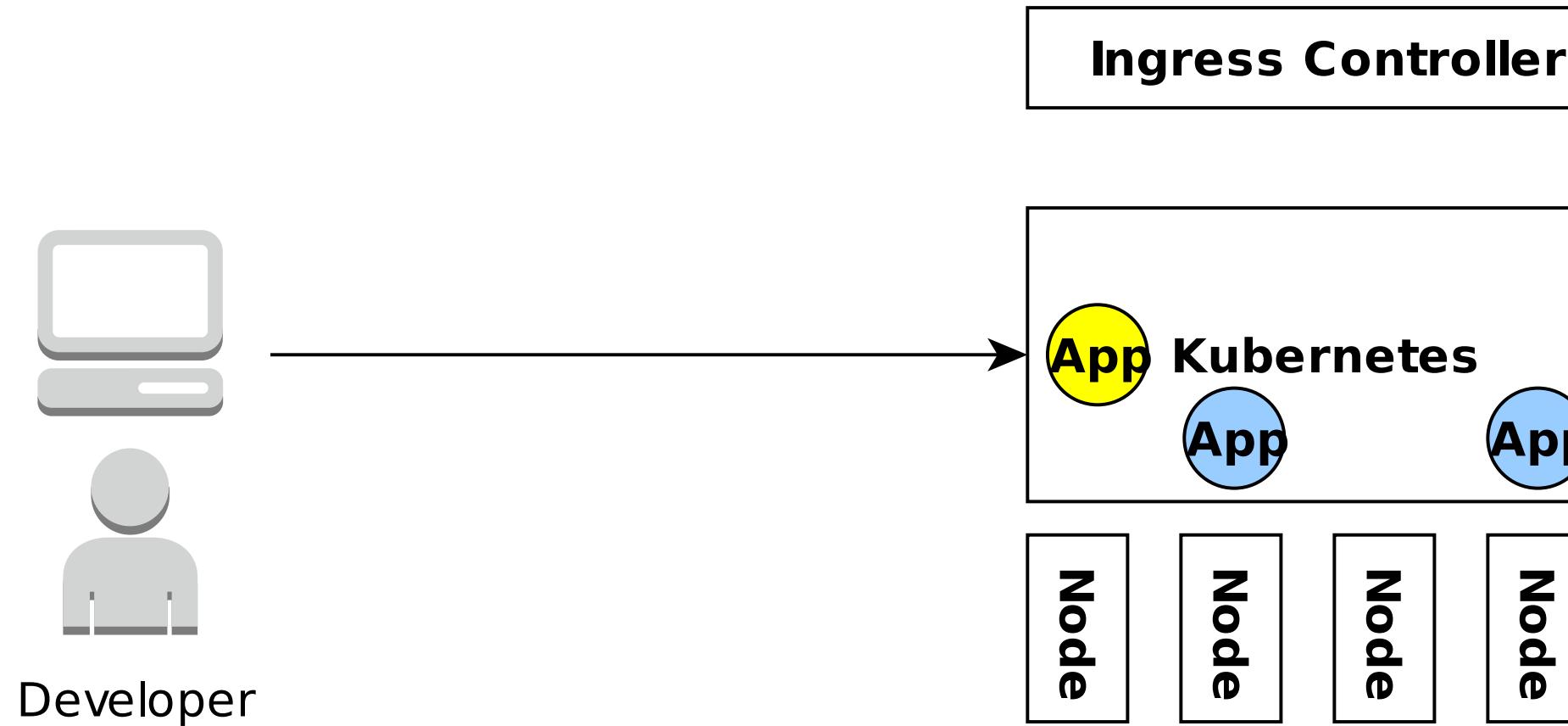


# SCALING



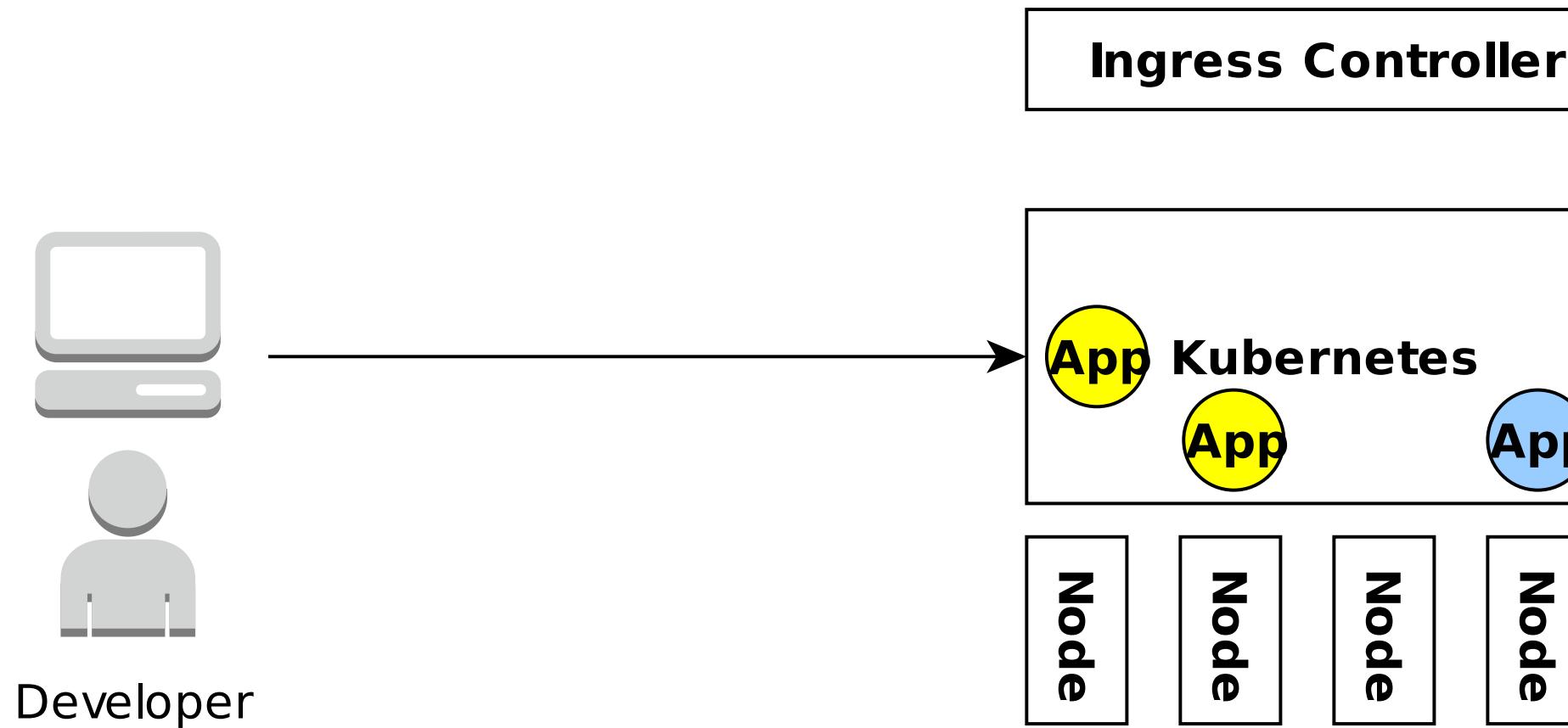
`kubectl scale --replicas=3 -f app-srv-dpl.yaml`

# ROLLING RELEASE



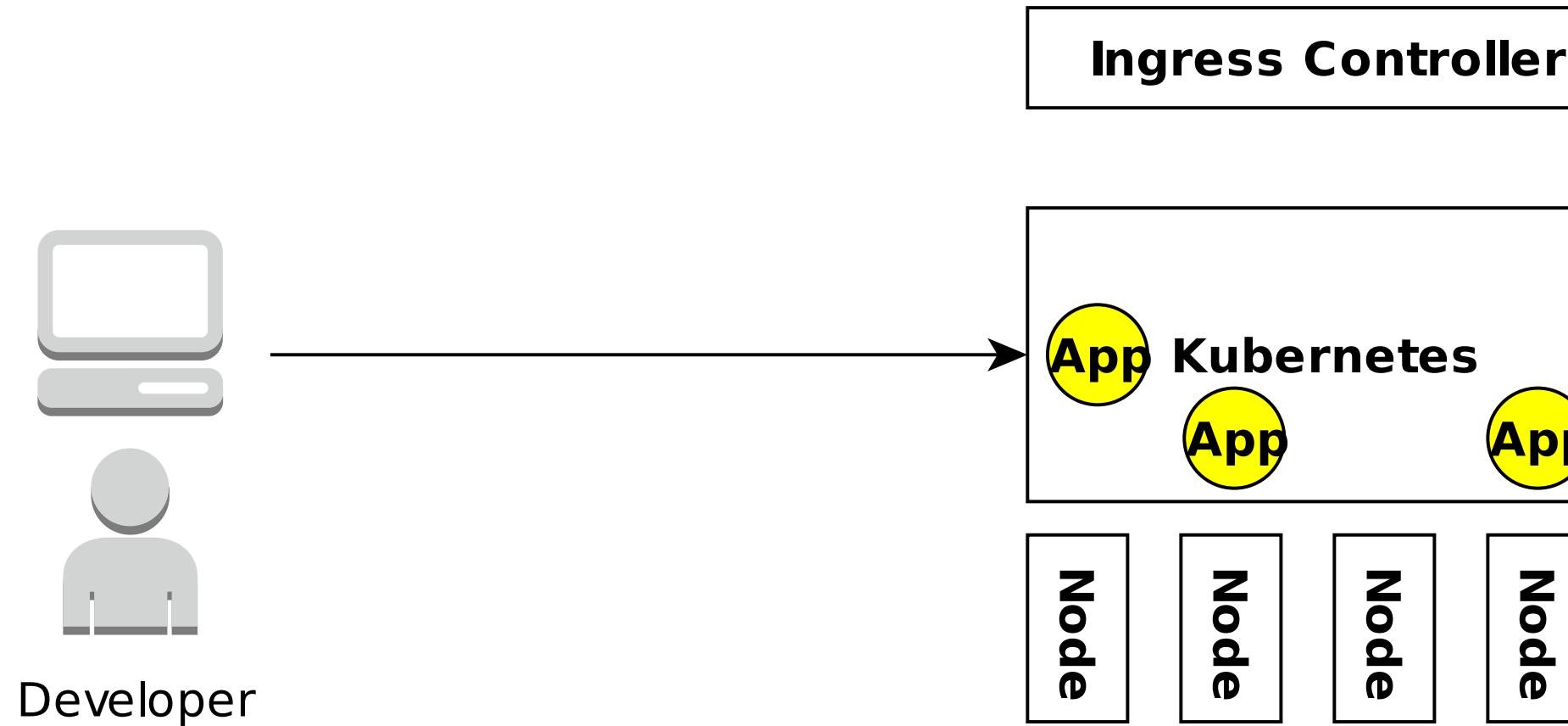
`kubectl set image deployment/app app=app:v2.0.0`

# ROLLING RELEASE



```
kubectl set image deployment/app app=app:v2.0.0
```

# ROLLING RELEASE



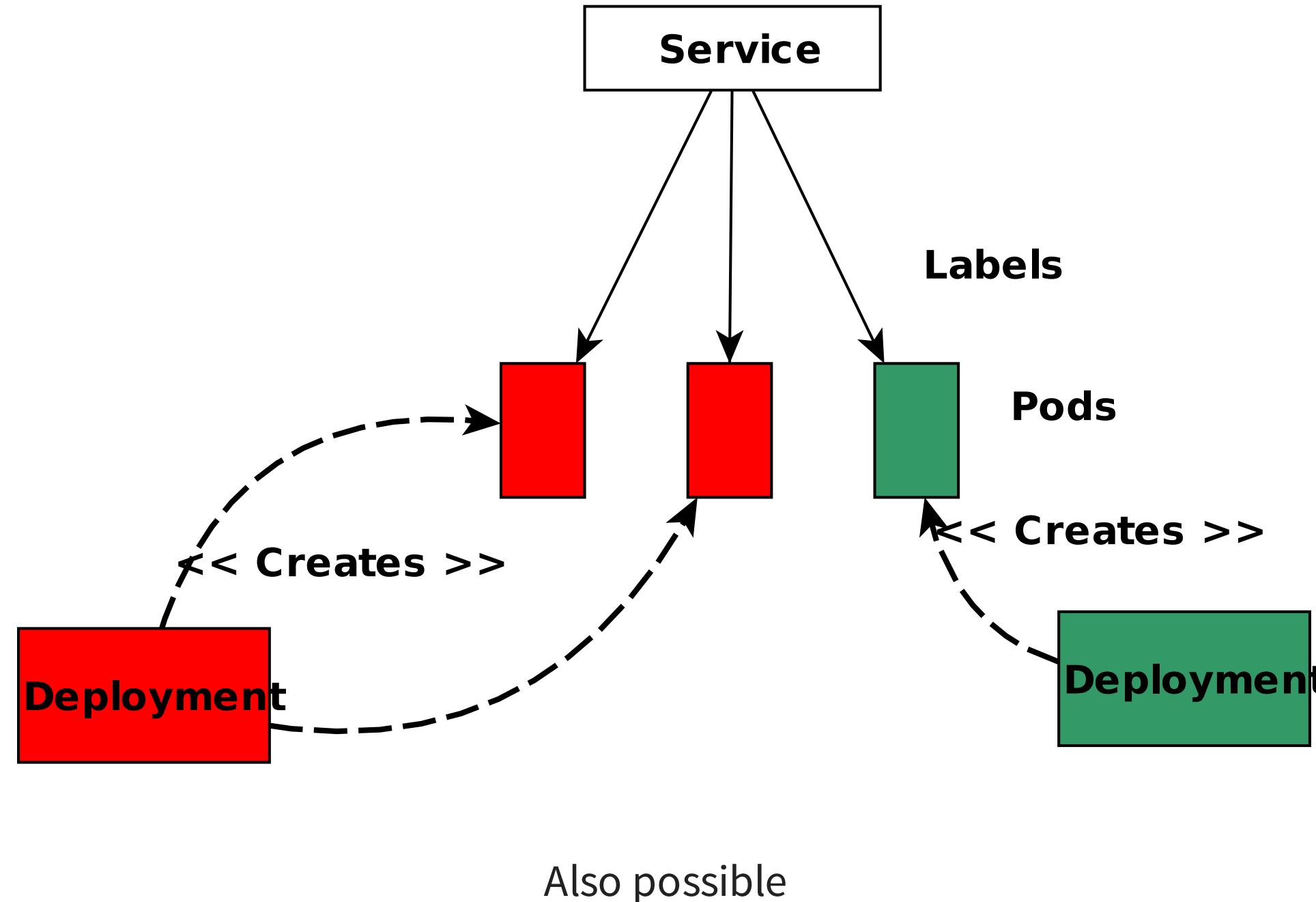
```
kubectl set image deployment/app app=app:v2.0.0
```

## ROLLING RELEASE

We used:

- Basic Rolling Releases per Component (focus on simplicity)
- A/B switches in Mobile App with Google Firebase  
BlueGreen Deployment - ingress controller

# ROLLING RELEASE WITH DEPLOYMENTS



# THE BEST PART

All the code lives in your git:

- integration with monitoring, alarming
- integration with ingress-controller
- smoke tests
- ...
- Devs can forget about infrastructure... almost
- Happy developers! ☺

DevOps Culture Dream!

# INGRESS RULE PER COMPONENT

cat tools/ingress-production.yaml

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: api-status
  namespace: production
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: api.example.com
    http:
      paths:
      - path: /status
        backend:
```

# INTEGRATION WITH PROMETHEUS

cat memcached-0-service.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: memcached-0
  labels:
    app: memcached
    kubernetes.io/name: "memcached"
    role: shard-0
    tier: backend
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/scheme: "http"
    prometheus.io/path: "metrics"
    prometheus.io/port: "9150"
```

<https://github.com/skarab7/kubernetes-memcached>

# USE LABELS IN ALERT RULES

```
ALERT ProductionAppServiceInstanceDown
  IF up { environment = "production", app =~ ".+" } == 0
  FOR 4m
  ANNOTATIONS {
    summary = "Instance of {{$labels.app}} is down",
    description = " Instance {{$labels.instance}} of app {{$labels.app}}
  }
```

AlertManager

# USE LABELS IN ALERT ROUTING

Call somebody if the label is `severity=page`:

```
---  
group_by: [cluster]  
# If an alert isn't caught by a route, send it to the pager.  
receiver: team-pager  
routes:  
- match:  
  severity: page  
  receiver: team-pager  
  
receivers:  
- name: team-pager  
  opsgenie_configs:  
  - api_key: $API_KEY  
    teams: example_team
```

AlertManager

# SIMPLIFY

- Google Identity-Aware-Proxy to protect all dashboards

# INFRA REPO

Terraform:

- Prometheus
- Grafana
- Centralized logging - EFK
- DNS entries
- Setup 3rd party, e.g., OpsGenie and StatusCake

# INFRA REPO

Config files:

- alertManager rules
- alertManager routing
- graph definitions

# INFRA REPO

Extras:

- TF for Google Identity-Aware-Proxy [\*]
- external services pointing to AWS instances with

kubectl

[\*] Setting up LB and Health-Checks

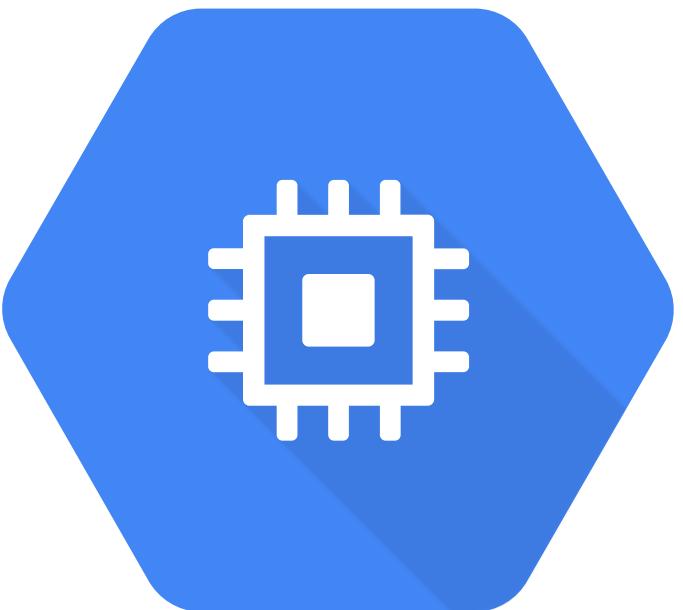
# SUM UP

1. Kubectl and Kubernetes config files [\*]
2. Terraform for the rest
3. Gopass for sharing secrets and configs

[\*] some components still needed manual intervention

# GOOGLE CONTAINER ENGINE

- Managed k8s
- Integrated with Google Cloud
- Identity-Aware Proxy
- Google 2-factor auth



## **MANAGED**

- Painless (1-2 second off-line) upgrades
- Scale-up and Scale-down
- No need to login to the nodes
- Google Docker Repository with no surprises

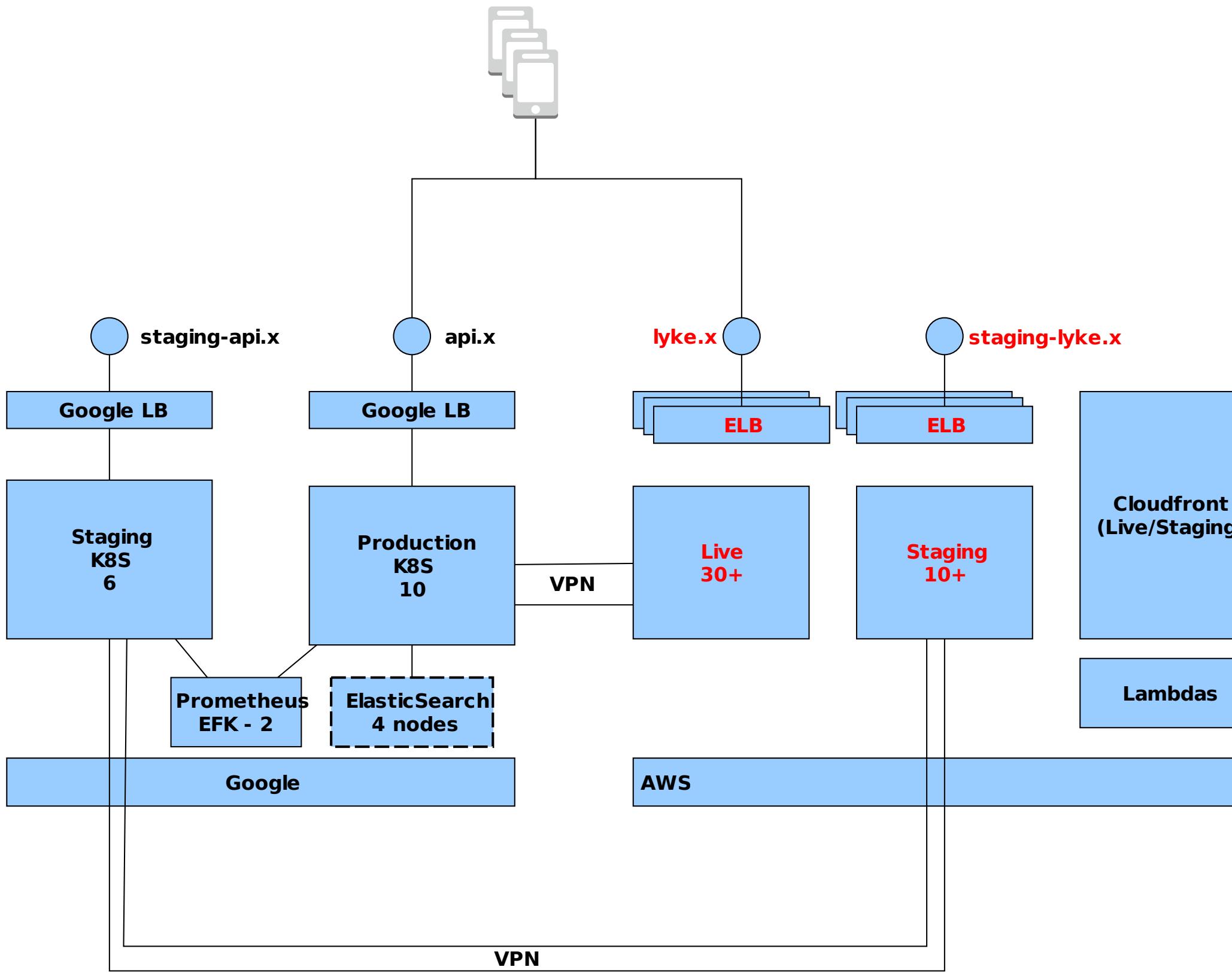
## INTEGRATION

- Load-Balancer and Health-checks
- Persistent volumes
- Graphic-card support added
- Sane Service accounts for CD and CI

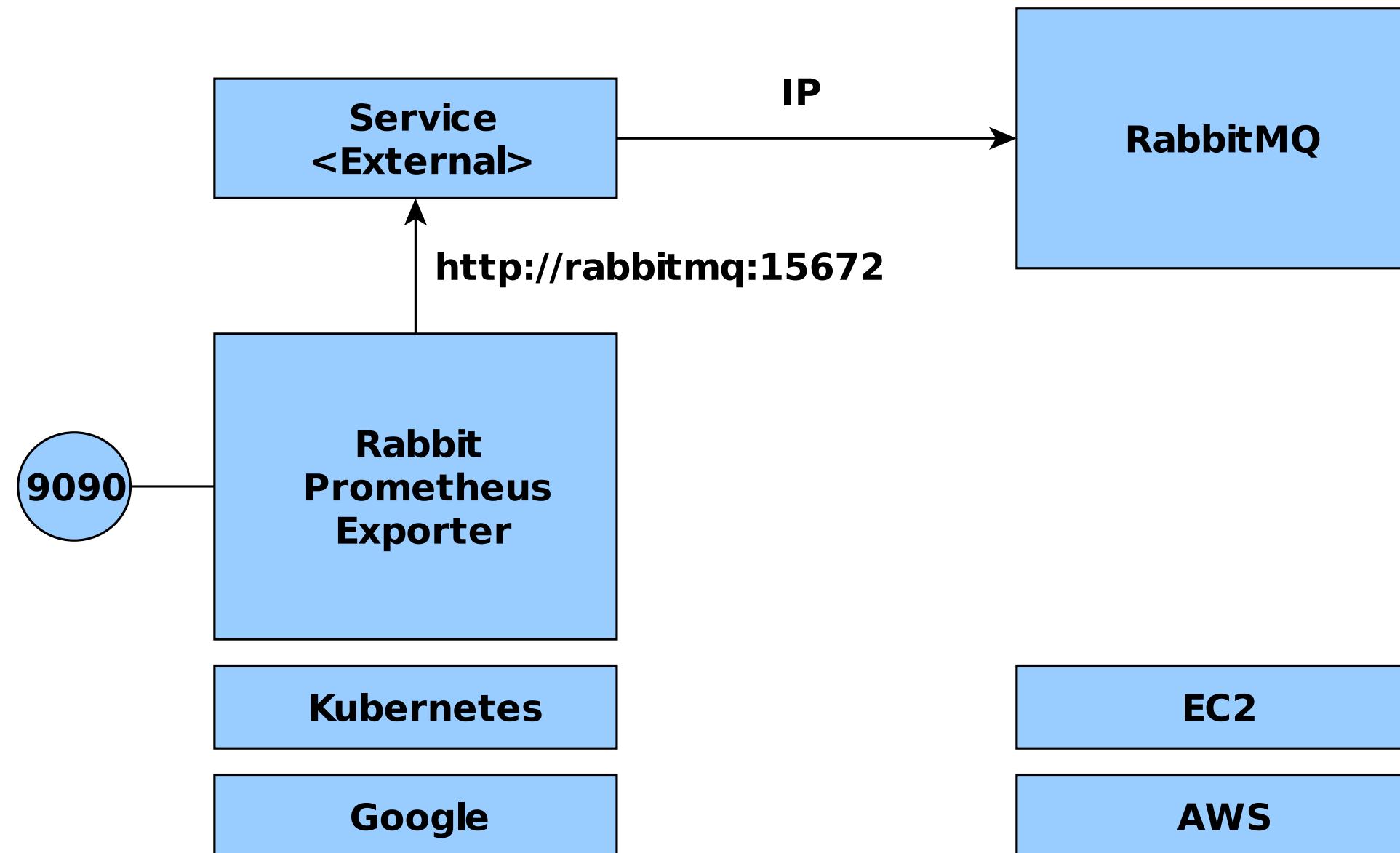
## OUR FINAL SETUP

- 2 clusters: staging and production
  - use Google Database [\*]
  - instance for Prometheus+AlertManager and EFK
  - 3rd parties: Opsgenie and StatusCake
  - Still use AWS components: Lambda and Cloudfront
- [\*] was planned

# Architecture During Migration



# NEW WITH OLD



## OLD WITH NEW

- Ad-hoc
- DNS entries to nodes
- Some talking over (secured) public endpoints

I wish I introduced HashiCorp Consul earlier

# **CHANGE THE DEVELOPMENT PRACTICES**

See [Conway's law](#)

# 1. CLEAN UP

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

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

## 2. GET BACK ALL THE KNOWLEDGE

- Puppet, Chef, ... → Dockerfile
- Check the 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
- Use: `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

# 5. CONTINUOUS DEPLOYMENT

Travis:

- test code, build docker, push to docker repo
- only run the rolling update:  
`kubectl set image deployment/api-status  
nginx=nginx:1.9.1`
- did not create any kubernetes artifacts [\*]

[\*]

## 6. KEEP IT RUNNING

Bridge the new with old:

- Use external services in Kubernetes
- Add Kubernetes services to your Service Discovery

[1]

[1] I evaluated feeding K8S events to HashiCorp consul

## 7. INTRODUCE SMOKE-TEST

```
TARGET_URL=127.0.0 make smoke_test
```

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

## 8. GOT FIRST MICRO-SERVICES

To offload the biggest components:

- Keep the light on of the old components
- New functionality delegated to micro-services

## 9. GET PERFORMANCE TESTING

- introduce wrk for evaluating performance (more like a check for dockers)
- load test the real system

# SERVICE SELF-CONSCIOUSNESS

Add to old services:

1. *metrics/*
2. *health/*
3. *info/*
4. *alertrules/ - PoC*

# CHANGE THE WORK ORGANIZATION

- From Scrum
- To Kanban

For the next talk

## WHAT I WOULD DO DIFFERENT

- More aggressive getting k8s components running in production
- Cut all PoC to 2 weeks max (consul, Vault, Hydra,...)
- Introduce HashiCorp in the old stack first
- Migrate the data to Google Mysql earlier

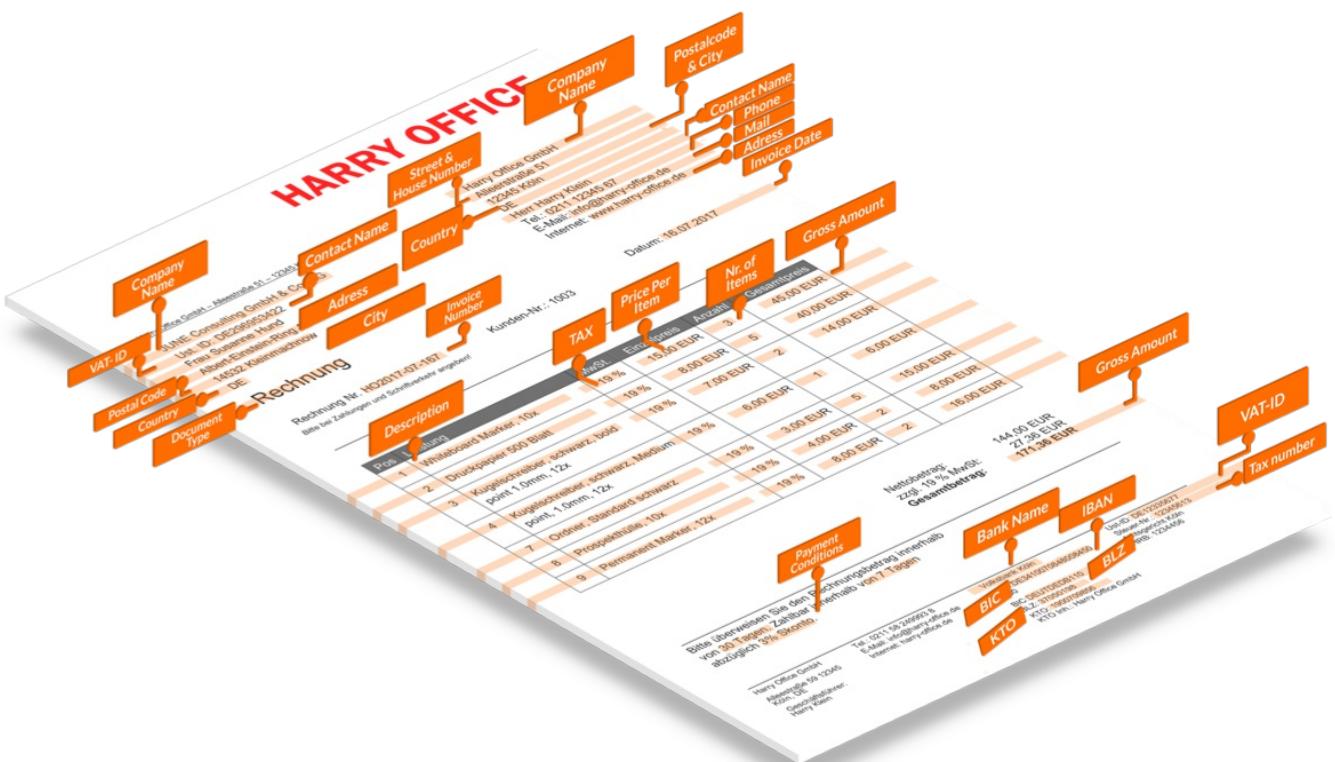
## **BUILDING NEW SYSTEMS**

- Ability to replicate user traffic

**SMACC**

# SMACC

- End-to-end Automation of Accounting
- Recognized AI FinTech Startup in Germany
- <https://www.smacc.io/en/>



# GREAT POINTS

- Strong AI and Data Science Team
- Deep know how on accounting and finance processes
- a lot of tested code, strong engineering



# CHALLENGE

- Before noneed for strong System Engineering
- Initial experiment with ECS, not amazing
- Few monolith oversized components
- Business Pressure on choosing a EU cloud provider
- Customers at the gates!

# KUBERNETES

- OAuth2 from Azure DS
- TravisCI for Continuous Deployment
- ProfitBrikcs S3
- Databases as separate nodes

# PROFITBRICKS

- Terraform provider [X]
- Basic Load Balancing (layer 3) [X]
- DE [X]
- Thanks to Kubernetes, we can easily move to Amazon EKS, Google CE, or Azure Kubernetes if needed

# NEXT STEPS

- Building new services around the core AI product
- Tech-debt in check, we can focus on building an effective (modern) architecture
- Evaluating new: [linkerd.io](https://linkerd.io), [istio.io](https://istio.io)
- missy - an auto-wiring microservice Open Source framework to forge our lesson-learnt into code  
(more on that next year :))

# NEXT STEPS

- More on our platform and OpenSource projects in one of the next meetups

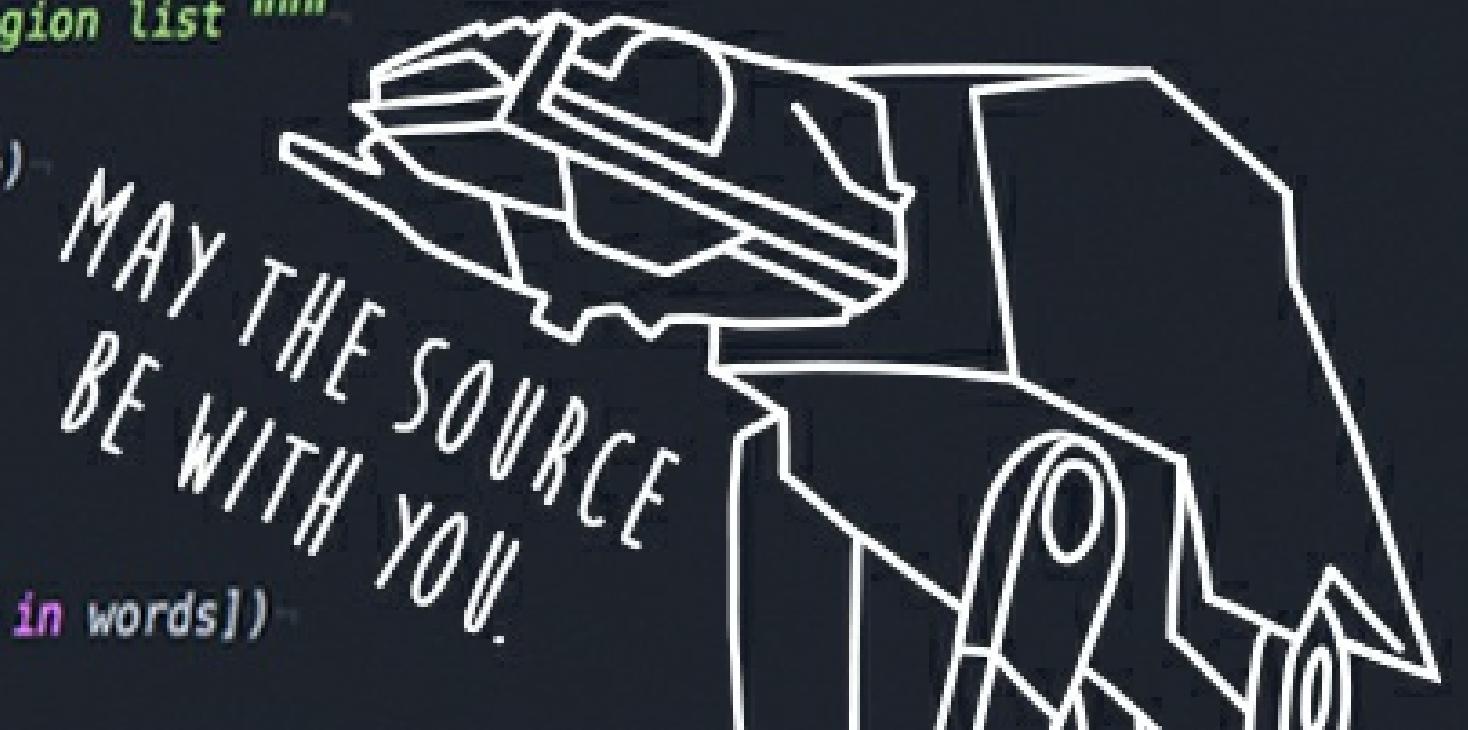


# SUMMARY

- Changes happens: Single point of entry, README.rst
- Kubernetes is the new Linux!
- Less important your cloud provider
- RAFT... CAP... in the end it is about people

# THANK YOU

```
123     def distance_matrix(regions):
124         """ Computes a distance matrix against a region list """
125         tuples = [r.as_tuple() for r in regions]
126         return cdist(tuples, tuples, region_distance)
127
128
129     def clusterize(words, **kwargs):
130         # TODO: write a cool docstring here
131         db = DBSCAN(metric="precomputed", **kwargs)
132         X = distance_matrix([Region.from_word(w) for w in words])
133         labels = [int(l) for l in db.fit_predict(X)]
```



From our developers, Warsaw Office in BL Astoria:



# QUESTIONS?

```
123 def distance_matrix(regions):
124     """ Computes a distance matrix against a region list """
125     tuples = [r.as_tuple() for r in regions]
126     return cdist(tuples, tuples, region_distance)
127
128
129 def clusterize(words, **kwargs):
130     # TODO: write a cool docstring here
131     db = DBSCAN(metric="precomputed", **kwargs)
132     X = distance_matrix([Region.from_word(w) for w in words])
133     labels = [int(l) for l in db.fit_predict(X)]
```



# BACKUP SLIDES

```
123 def distance_matrix(regions):
124     """ Computes a distance matrix against a region list """
125     tuples = [r.as_tuple() for r in regions]
126     return cdist(tuples, tuples, region_distance)
127
128
129 def clusterize(words, **kwargs):
130     # TODO: write a cool docstring here
131     db = DBSCAN(metric="precomputed", **kwargs)
132     X = distance_matrix([Region.from_word(w) for w in words])
133     labels = [int(l) for l in db.fit_predict(X)]
```



## **EVERYTHING RUNS IN POD**

Kubernetes administrated with kubernetes:

- everything run in pods
- e.g., you deploy your log collectors for k8s as pods:

<http://wbarczynski.pl/centralized-logging-for-kubernetes-with-fluentd-and-elasticsearch/>

# BASIC CONCEPTS

- config / secret → config and files
- ingress-controller → url pattern → service

# CONFIG

- env variables in deployment:

```
env:  
- name: SEARCH_ENGINE_USER  
  value: mighty_mouse
```

# CONFIG

- feed envs from configmaps:

```
env:  
- name: SEARCH_ENGINE_USER  
  valueFrom:  
    configMapKeyRef:  
      name: my-config  
      key: search.user
```

# CONFIG

- you can ship files using configmaps/secrets

```
kubectl create configmap my-config-file  
--from-file=config.json
```

# CONFIG

You can also run your own:

- HashiCorp Consul or etcd
- HashiCorp Vault

# THERE IS SO MUCH MORE

- resource quotas
- events in Kubernetes
- readiness probes
- liveness probes
- volumes
- stateful
- namespaces
- ...

# KUBERENTES

- Awesome command-line
- Resilient platform
- simple YAML files to setup your service,
- service discovery included
- annotations and metadata discovery included

**0.1 → 1.0**

Your component needs to get much more smarter.

# SERVICE SELF-CONSCIOUSNESS

Your endpoint:

- *metrics/*
- *alertrules/- [WIP]*
- *health/ or healthz/*
- *info/*

## DEEP LOOK INSIDE

- when I am ready to serve requests
- when I need to restart myself
- what to do when dependent services are down
- ...

## DEEP LOOK INSIDE

- Am I really stateless?
- Caching?
- fail-fast, start fast

## **RELATIONS WITH OTHERS**

- master-worker relationships
- waiting for other resources / services

# 12FACTOR APPS

- find services by name or URI
- move the important config to environment variables

# LOGGING

- logstash json format
- make configurable with ENV variable

EFK or ELK

# WHAT WITH YOUR DATABASES

- Keep it in a separated (k8s) cluster
- The best, go with DaaS
- With *Stateful*, you can run your db in k8s

Long discussion...

# MIGRATION OF ENV

Staging, production, canary, green/blue ...:

- If you have \$\$\$, have a separated k8s cluster
- If not, use Namespaces

# APPS IN NEW WORLD

- 12 factor apps (Heroku, 2012)
- much much smarter
- much faster
- much more predictable
- much harder to develop :D
- Forging experience into code [WIP]:  
<https://github.com/microdevs>