Kubernetes

Introduction

WOJCIECH BARCZYŃSKI

- (hiring) Senior Software Engineer
  Lead of Warsaw Team - SMACC
- System Engineer background
- Interests:
  working software
- Hobby:
  teaching software engineering

BACKGROUND

- A top AI FinTech ➡ microservices and k8s
- Before renew tech stack of a top Indonesian mobile ecommerce
- 3.5y with Openstack, 1000+ nodes, 21 data centers
- I do not like INFRA :D

KUBERNETES

- Kubernetes - greek for helmsman
- Run and Manages containers
- Inspired by Google's Borg
- Integrated with AWS, GCP, Azure
- Becoming an integration platform for large ecosystem

Manages Applications not Machines!

GOALS

- Utilized resources nearly 100%
- Get to applications/services mindset
- Enforce loosely couple software - 12factor apps, Amazon-API approach
- Best practises included, e.g., name service, metadata discovery, ...

CURRENT WINNER

« Amazon joined Kubernetes
on 10.08.2017 »

WHY KUBERNETES?

- Data Center as a Black Box
- Batteries for your (12factor) apps

WHY KUBERNETES?

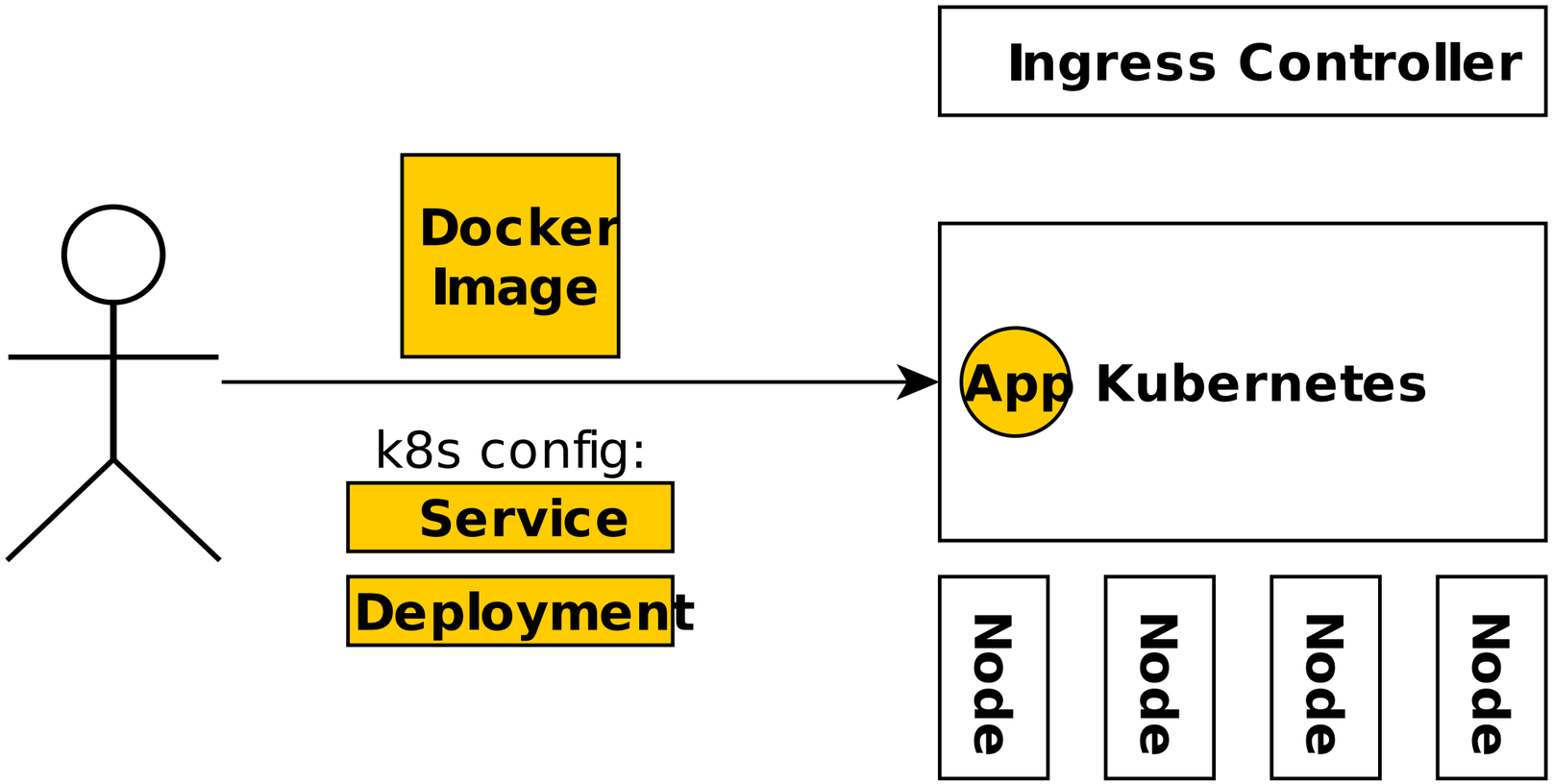
- Give you complete control over your application with simple *yaml* config files
- Use *labels* to auto-wire your app to monitoring, logging, and alarming
- Let you to, almost forget, about the infrastructure

Batteries

- Load Balancing
- Name Service Discovery
- Metadata and Annotation support
- Decoupled interface and implementation
- Labeled based matching

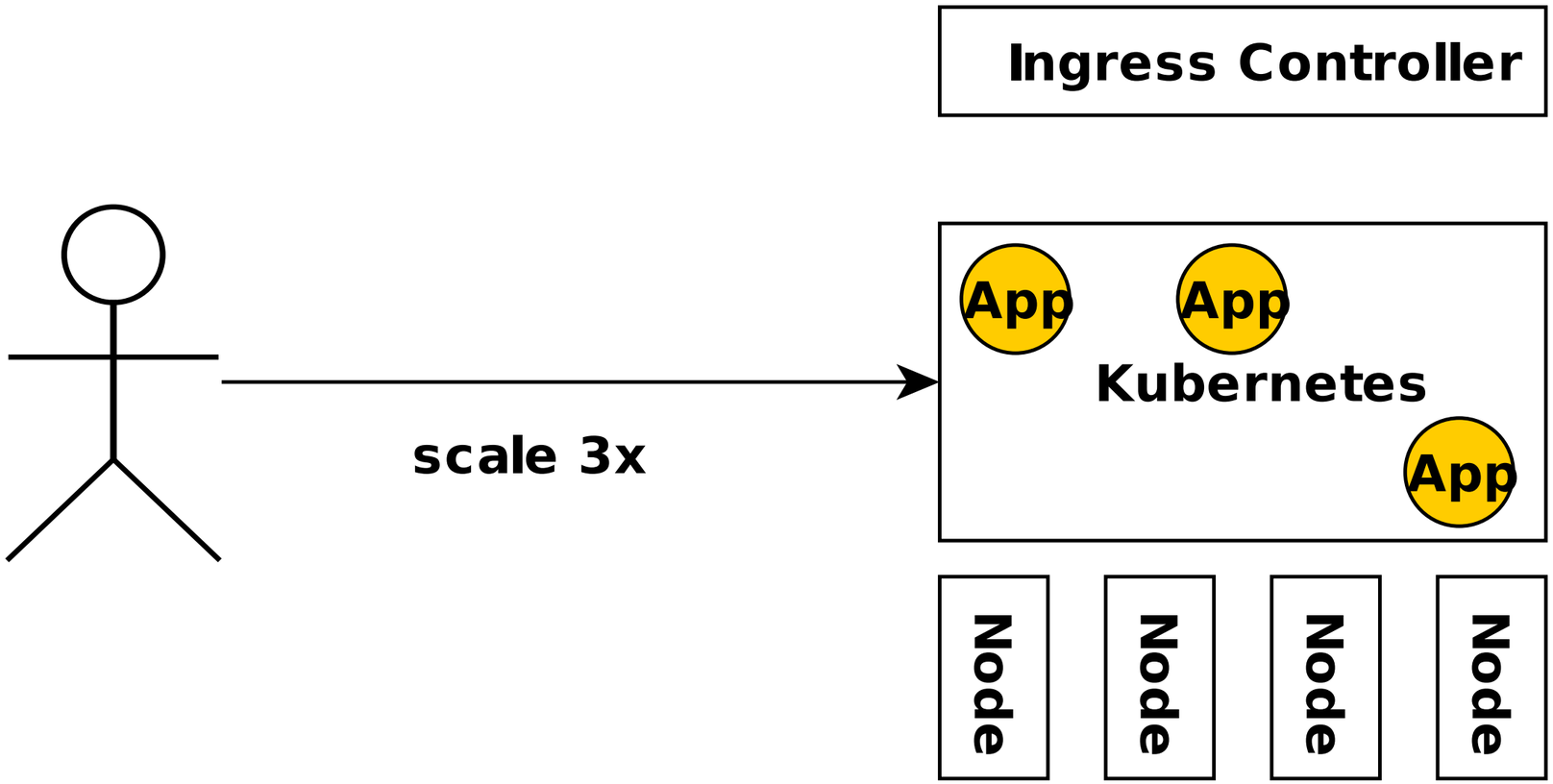
DATA CENTER AS A BLACK BOX

KUBERNETES




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

SCALE UP! SCALE DOWN!

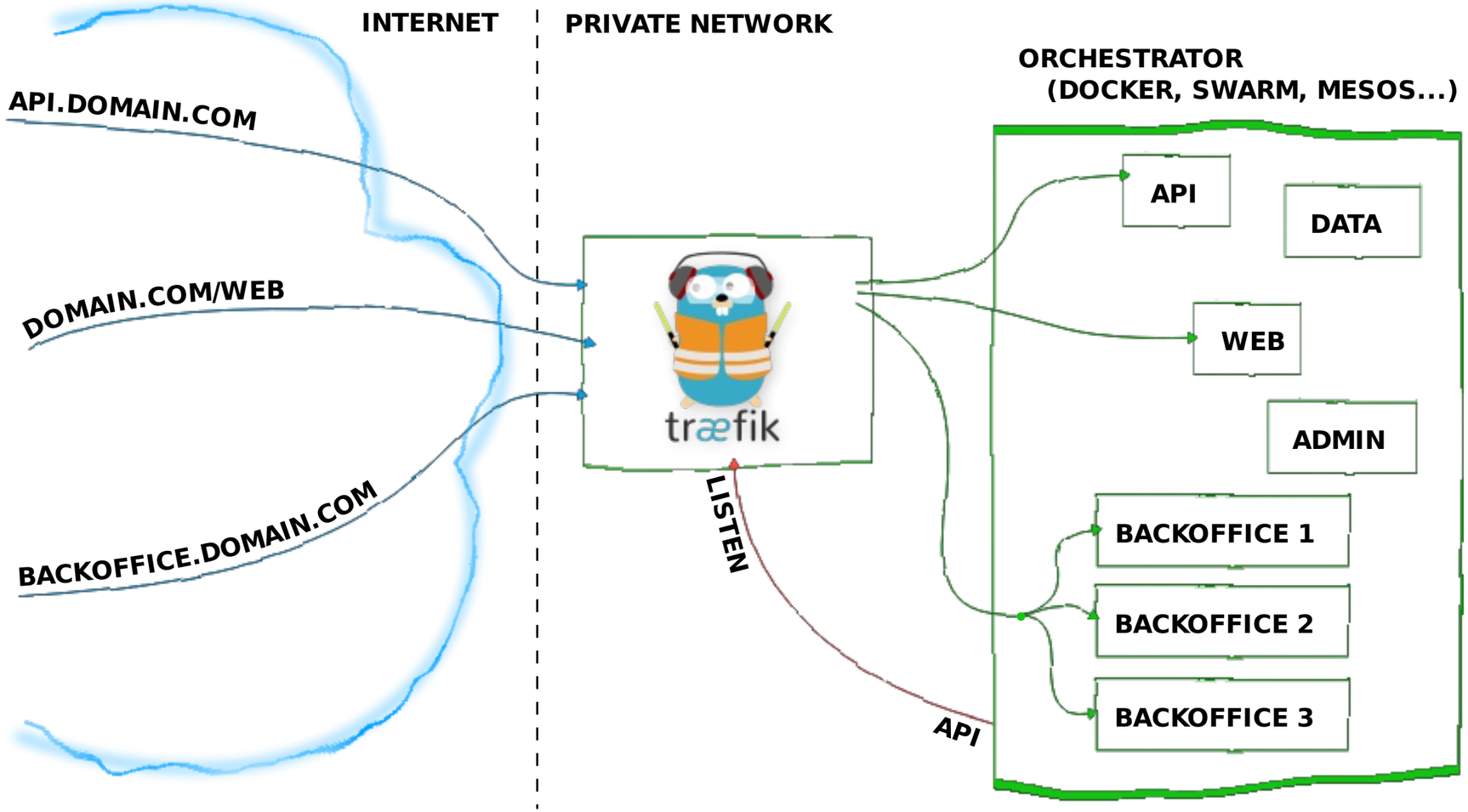


kubectl --replicas=3 -f app-srv-dpl.yaml

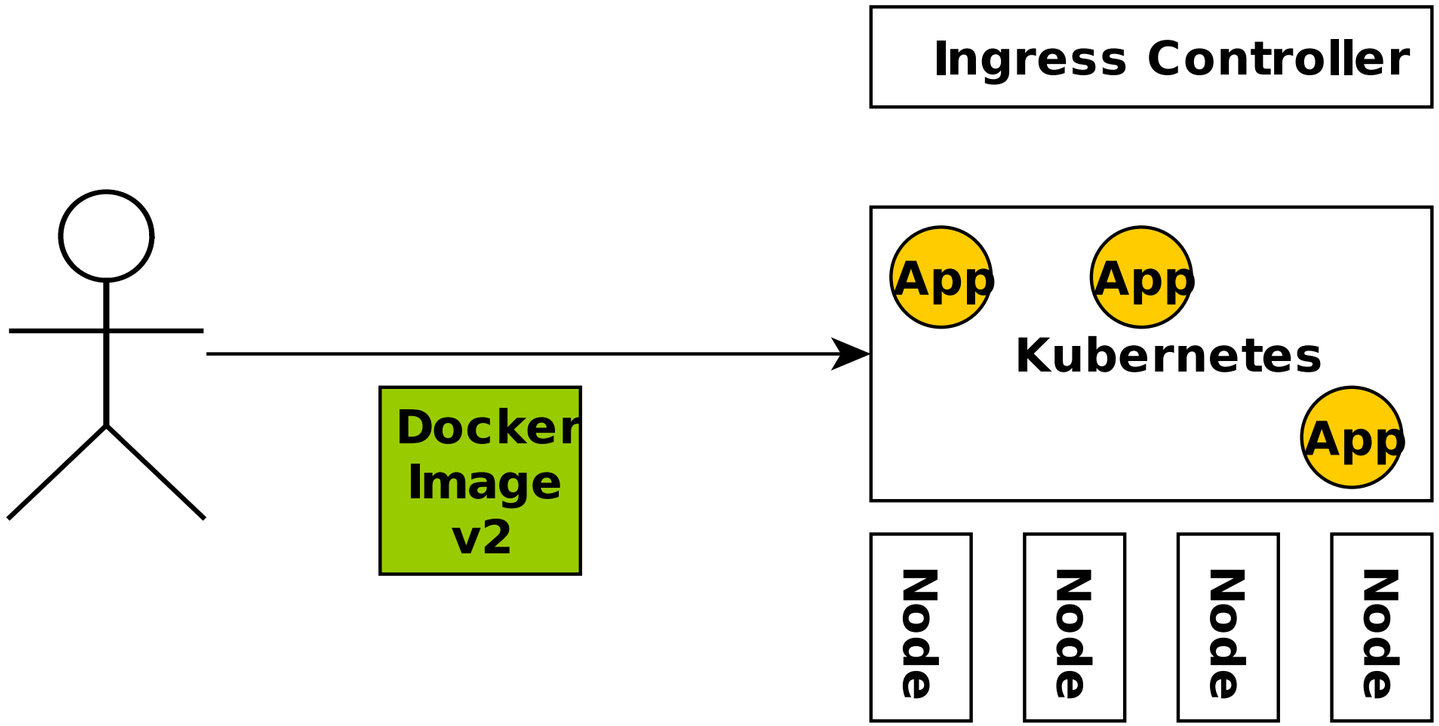
INGRESS CONTROLLER

- api.smacc.io/v1/users ➡ service: users-v1
- api.smacc.io/v2/users ➡ service: users-v2
- api.smacc.io/accounts ➡ service: accounts
- smacc.io ➡ service: website

INGRESS CONTROLLER

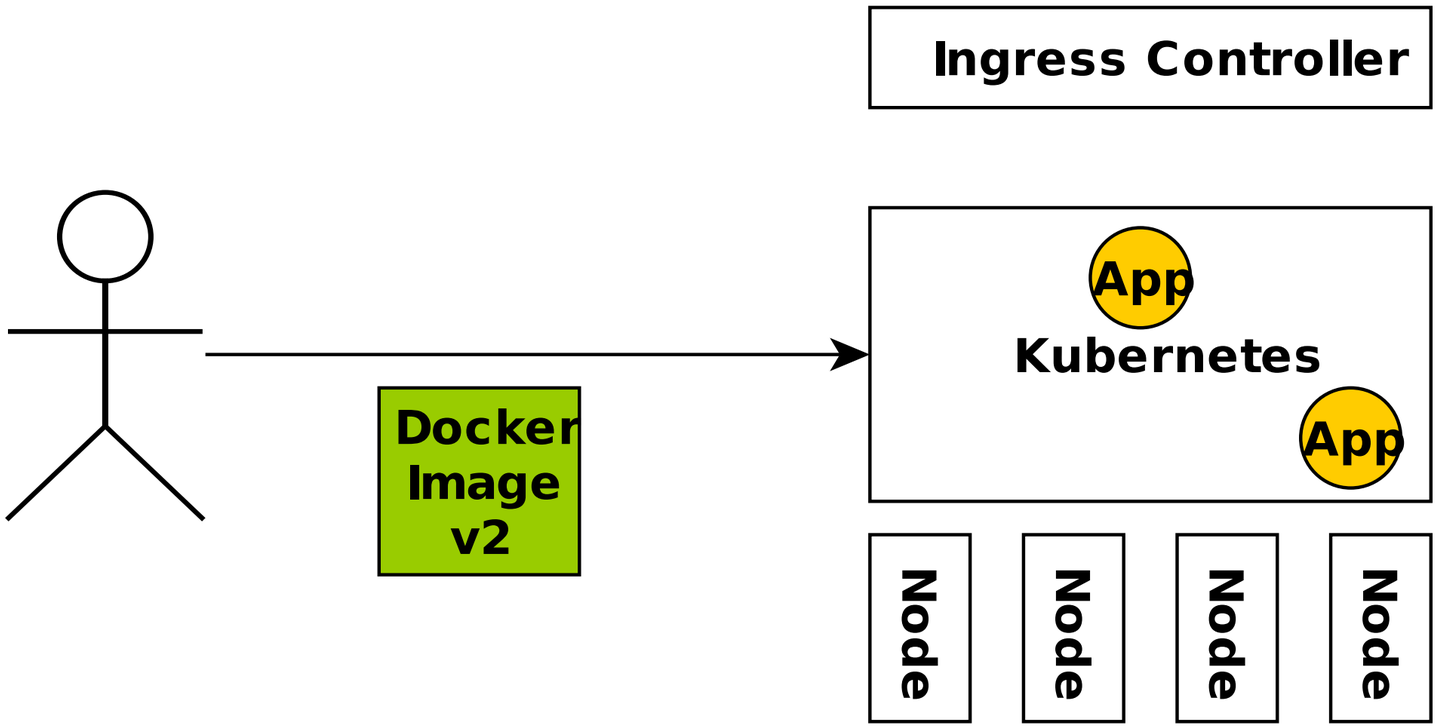


ROLLING UPDATES!

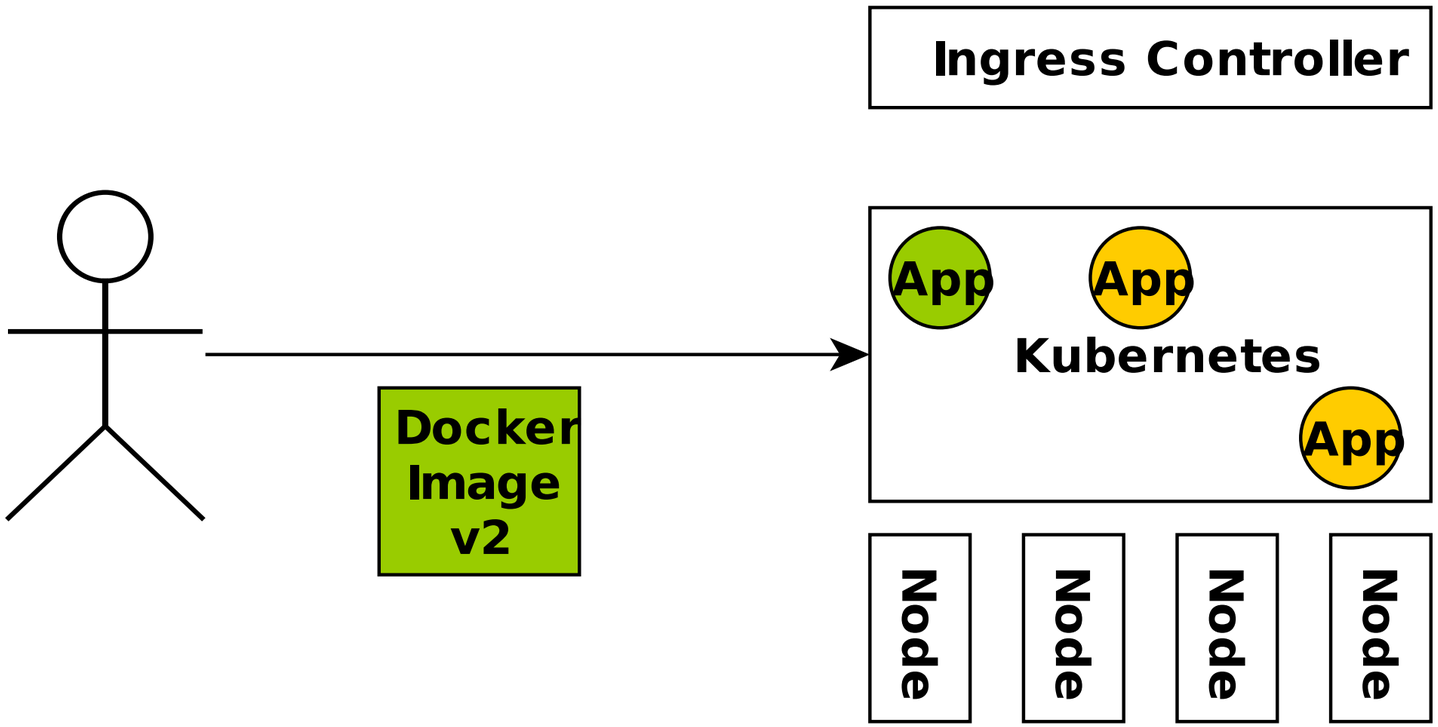



```
kubectl set image deployment/app app=app:v2.0.0
```

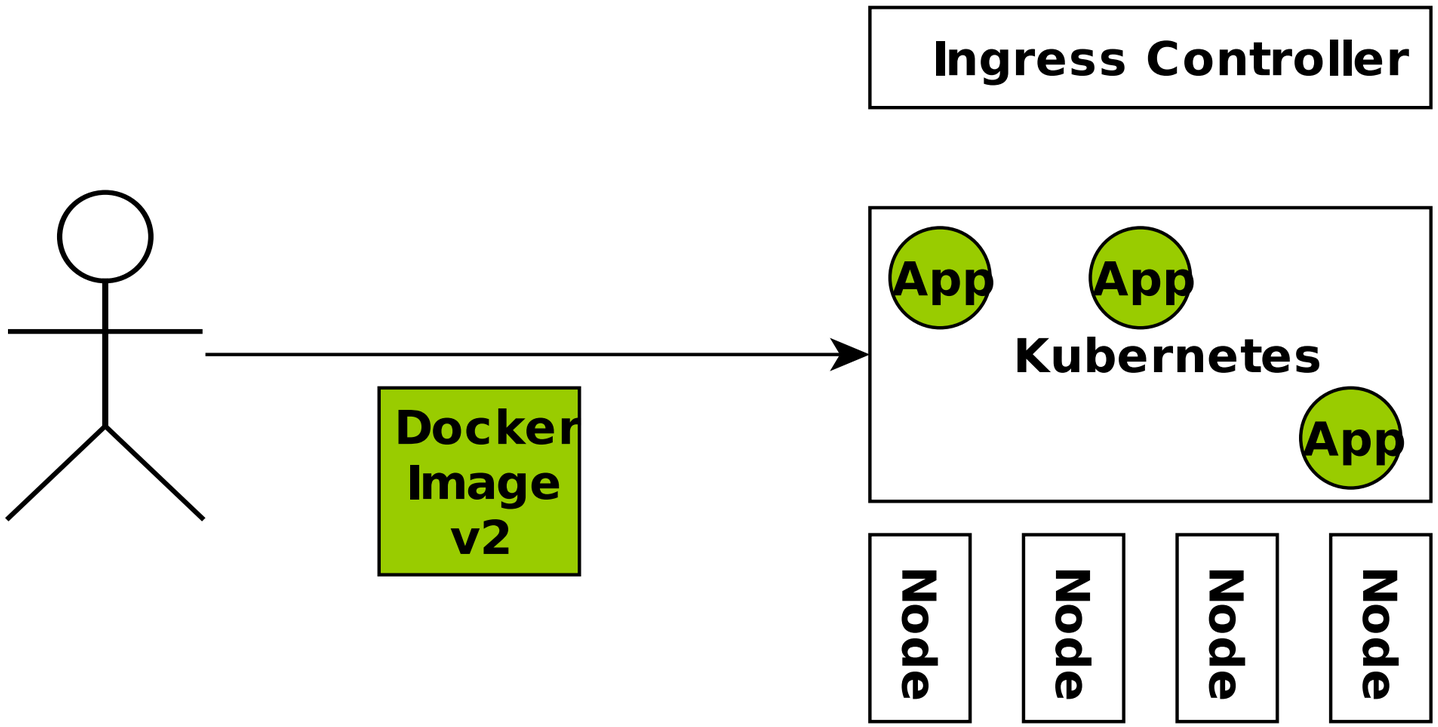
ROLLING UPDATES!



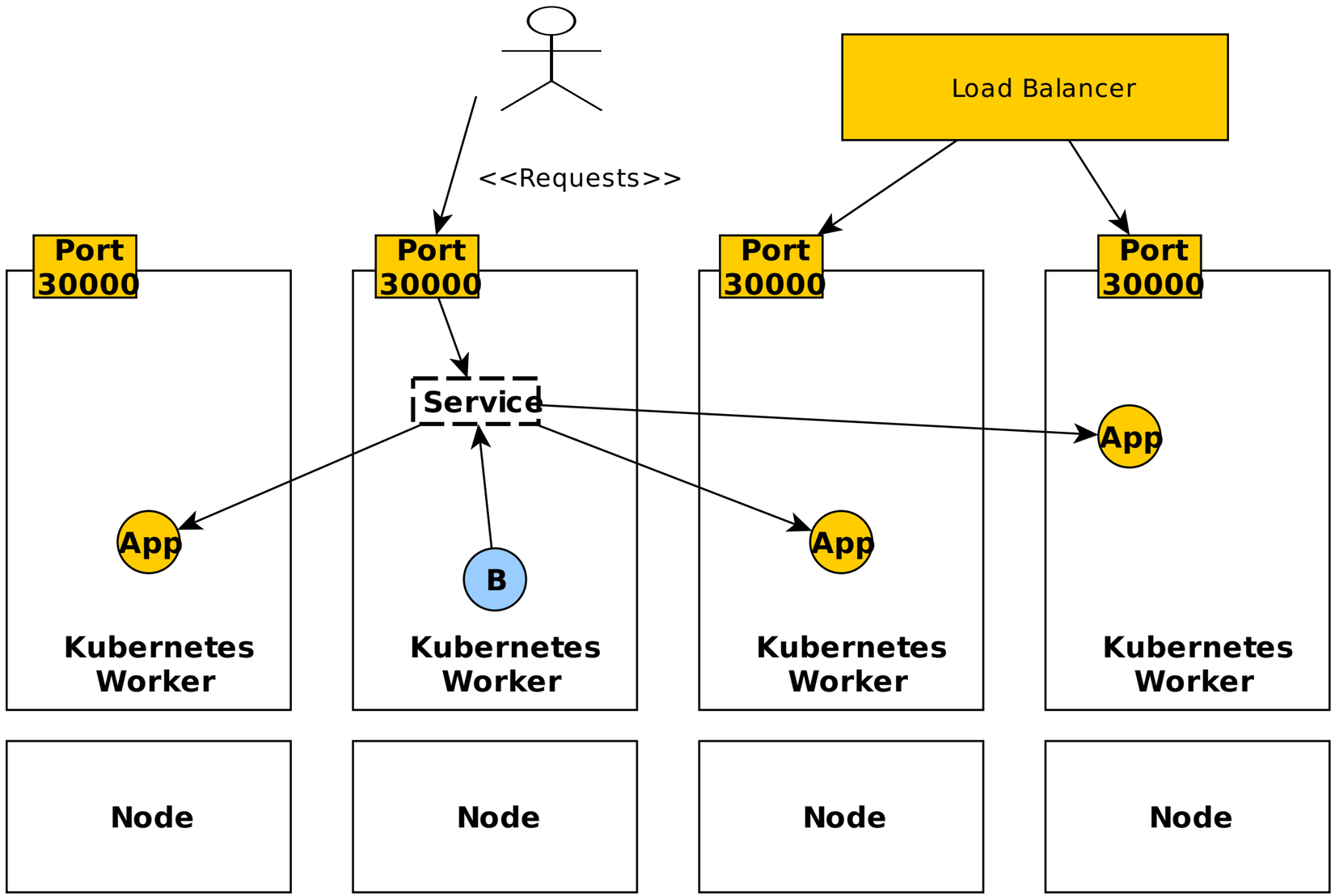
ROLLING UPDATES!



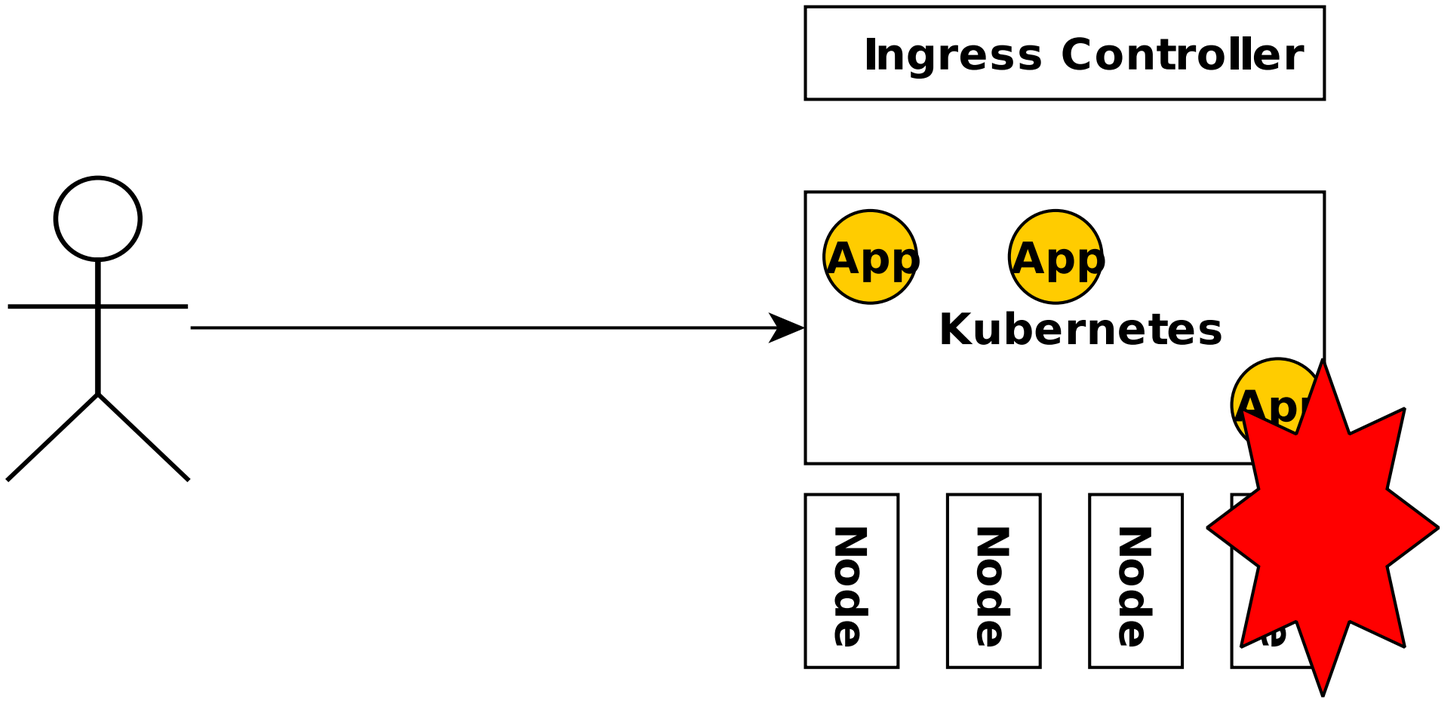
ROLLING UPDATES!



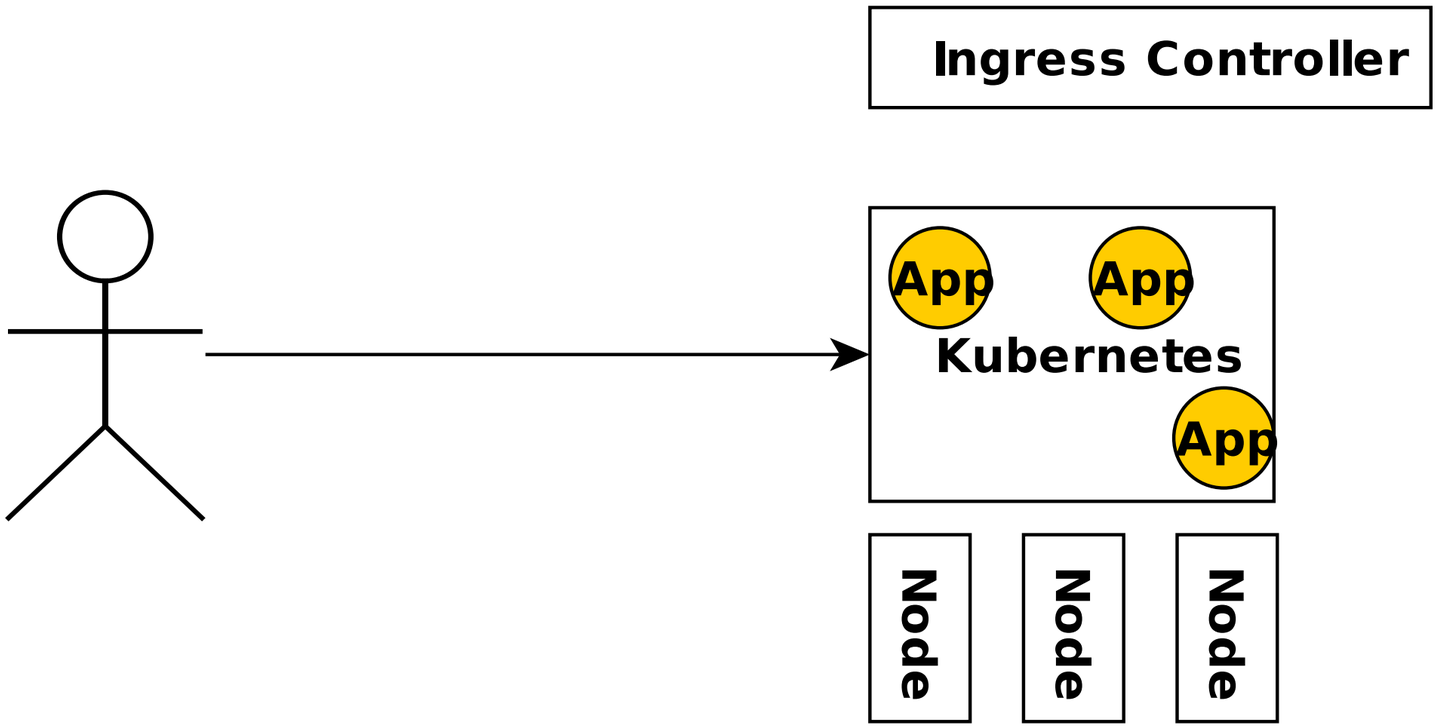
LOAD BALANCING



RESISTANCE!



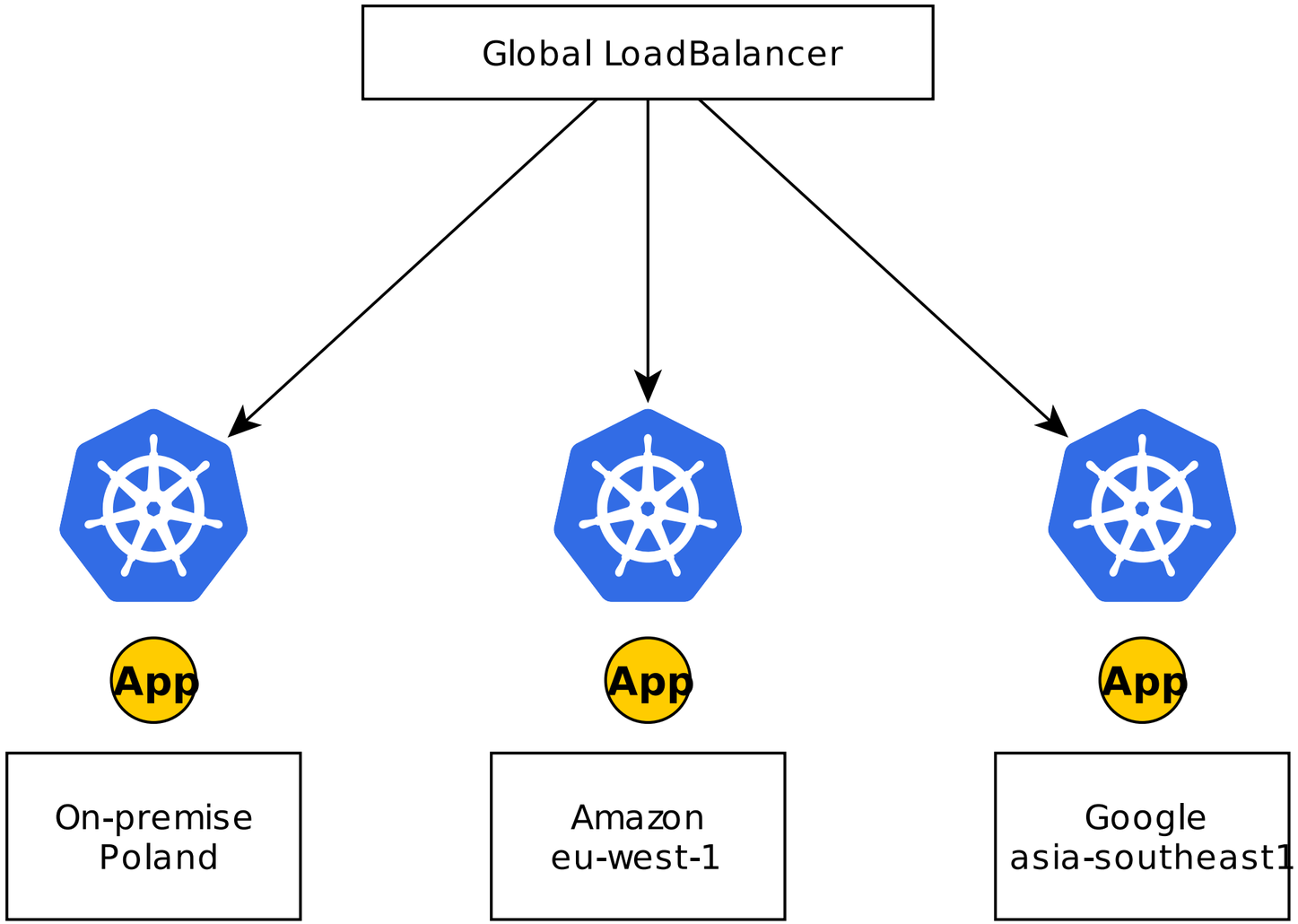
RESISTANCE!



RESISTANCE!

- When the node dies in flames
- When other apps (with higher guaranteed quotas) eats all memory
- When you need to drain nodes before upgrade
- You can easily scale up, create machine and join it to cluster (easier with kops or on GCE)

FEDERATION


Global LoadBalancer
App
App
App
On-premise
Poland
Amazon
eu-west-1
Google
asia-southeast1

MUCH MORE

Plug-and-play integrations:

- integration with AWS, Google Cloud Platform, and Azure
- multiple drivers for network, storage,...
- you can run on minikube

MUCH MORE

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

Name	Purpose	
Service	Interface	Service Name, port, labels, annotations
Deployment	Factory	How many pods with which docker images, labels
Pod	Implementation	1+ docker images running in 1 pod

BASIC CONCEPTS

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

SERVICE

service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: api-status
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: api-status
```

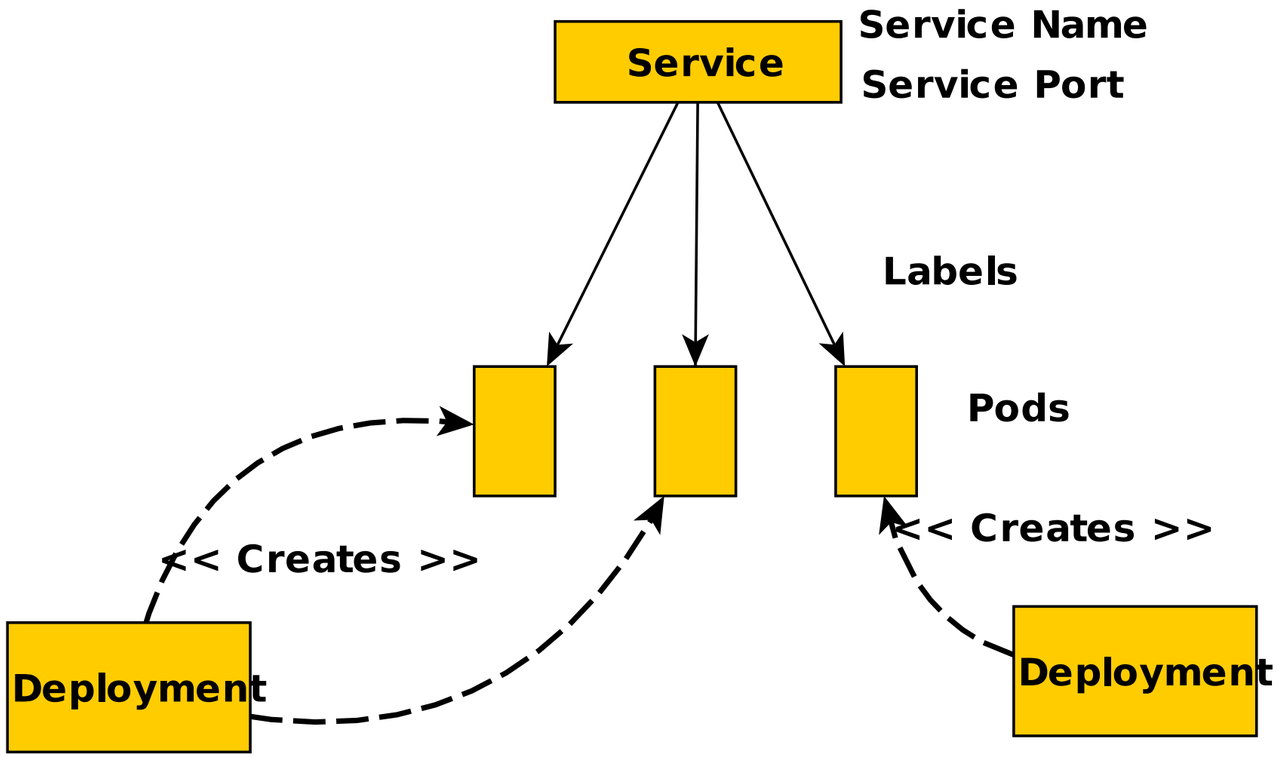
SERVICE

```
# create the service and deployment
kubectl create -f api-status-srv.yaml
kubectl create -f api-status-dpl.yaml

# get to a running docker (in a pod)
kubectl -it exec app-999-8zh1p /bin/bash

# check whether name service works
curl http://api-status/health
OK
```

BASIC CONCEPTS



deployment.yaml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: api-status-nginx
  app: api-status
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: api-status-nginx
        app: api-status
    spec:
      containers:
      - name: nginx
```

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

METADATA AND ANNOTATIONS

- Auto-wiring
- Precise discovery
- Reporting
- Labeling targets for security scans
- Labeling critical services for oncall (see alertmanager)

MONITORING WITH KUBERNETES

- You deploy a memcached
- Exposed its prometheus metrics on `metrics/`
- How to ship metrics?

ANNOTATIONS!

memcached-0-deployment.yaml

```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

INGRESS CONTROLLER WITH TRAEFIK?

ANNOTATIONS!

Use traefik instead of built-in reverse proxy

```
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:
            serviceName: api-status
```

LABELS!

Monitoring rule that uses labels:

```
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

LABELS!

Call sb 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

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

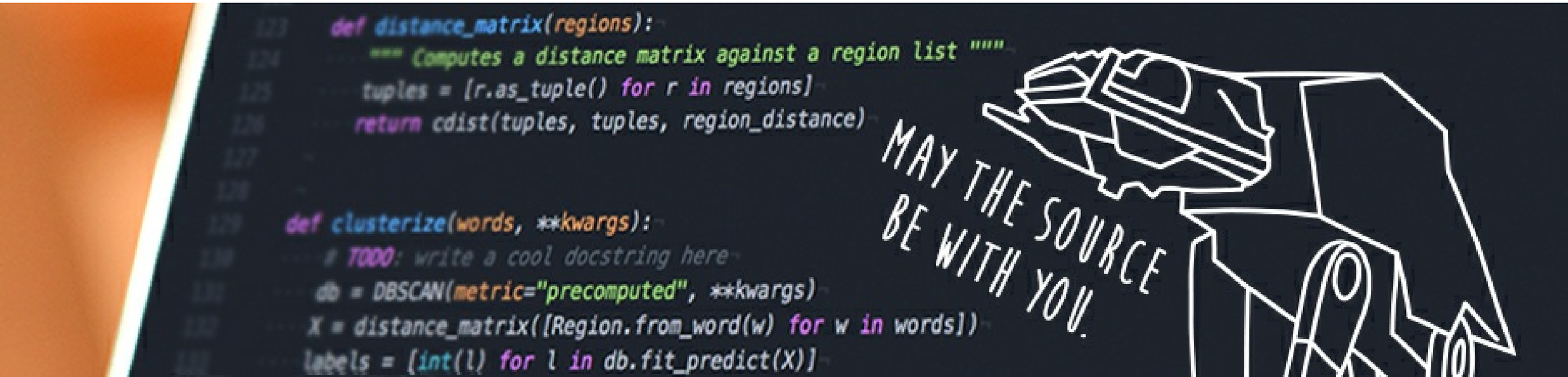
Staging, production, canary, green/blue ...:

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

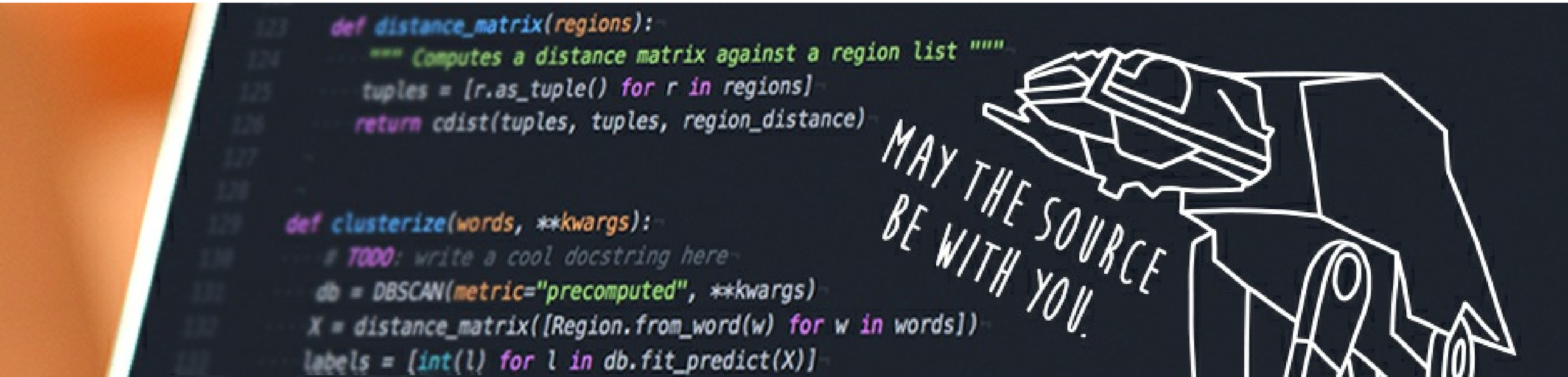
APPS IN NEW WORLD

- 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

THANK YOU


def distance_matrix(regions):
""" Computes a distance matrix against a region list """
tuples = [r.as_tuple() for r in regions]
return cdist(tuples, tuples, region_distance)
def clusterize(words, **kwargs):
# TODO: write a cool docstring here
db = DBSCAN(metric="precomputed", **kwargs)
X = distance_matrix([Region.from_word(w) for w in words])
labels = [int(l) for l in db.fit_predict(X)]
MAY THE SOURCE
BE WITH YOU.

(hiring) Wojciech Barczyński
(wojciech.barczynski@smacc.io)



Backup slides

6 + 1 STEPS

The big 1 - making your app smarter

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

Simple components:

- test code, build docker, push to docker repo
- run the rolling update:
  `kubectl set image deployment/api-status nginx=nginx:1.9.1`
- I use TravisCI

5. CONTINUOUS DEPLOYMENT

Complex components:

- with label-based matching, the sky is the limit

6. KEEP IT RUNNING

Brigde the new with old:

- You can add your external servies to the k8s Name Service
- You can bridge Kubernetes services to your Service Discovery [1]

[1] You can subscribe to K8S events to keep, e.g., your consul in sync