RED HAT® DEVELOPER PROGRAM

Kubernetes Introduction

@rafabene

benevides@redhat.com

Link → http://bit.ly/kubernetes-intro

Rafael Benevides
Director of Developer Experience at Red Hat
Apache DeltaSpike P.M.C

✉ benevides@redhat.com
@rafabene

Java Certifications:

SCJA / SCJP / SCWCD / SCBCD / SCEA

JBoss Certifications:

JBCD / JBCAA

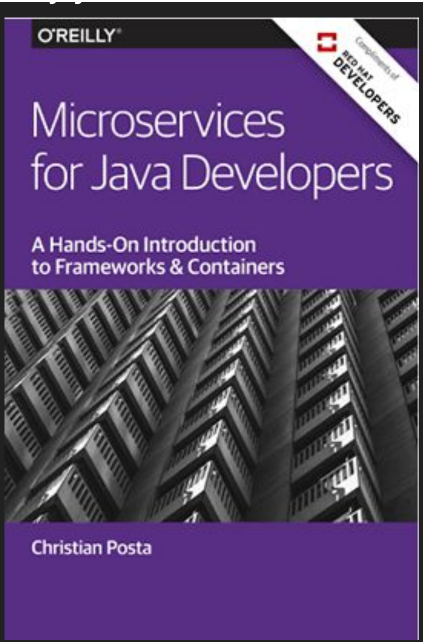
Red Hat Certifications:

OpenShift / Containers / Ansible

Other Certifications:

SAP Netweaver / ITIL / IBM Software Quality

bit.ly/javamicroservicesbook


O'REILLY

Compliments of RED HAT DEVELOPERS

Microservices for Java Developers

A Hands-On Introduction to Frameworks & Containers

Christian Posta

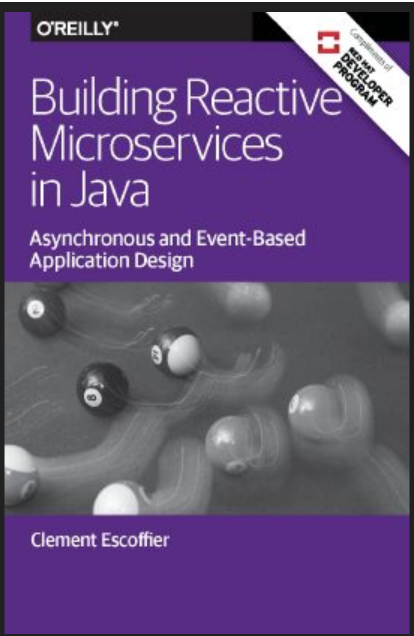
Free eBooks from developers.redhat.com

Microservices Introductory Materials

Demo: bit.ly/msa-instructions
Slides: bit.ly/microservicesdeepdive
Video Training: bit.ly/microservicesvideo
Kubernetes for Java Developers

Advanced Materials

bit.ly/reactivemicroservicesbook


O'REILLY

Compliments of RED HAT DEVELOPER PROGRAM

Building Reactive Microservices in Java

Asynchronous and Event-Based Application Design

Clement Escoffier

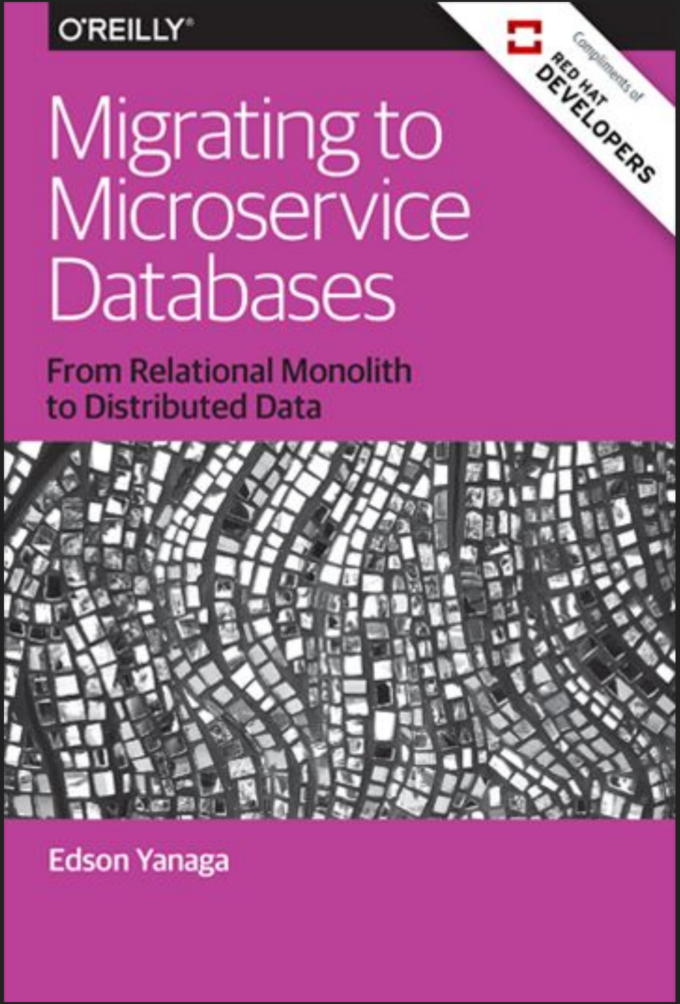
bit.ly/istio-tutorial

learn.openshift.com/servicemesh

bit.ly/faas-tutorial

learn.openshift.com/serverless

http://bit.ly/kubernetes-intro



O'REILLY®

Compliments of RED HAT DEVELOPERS

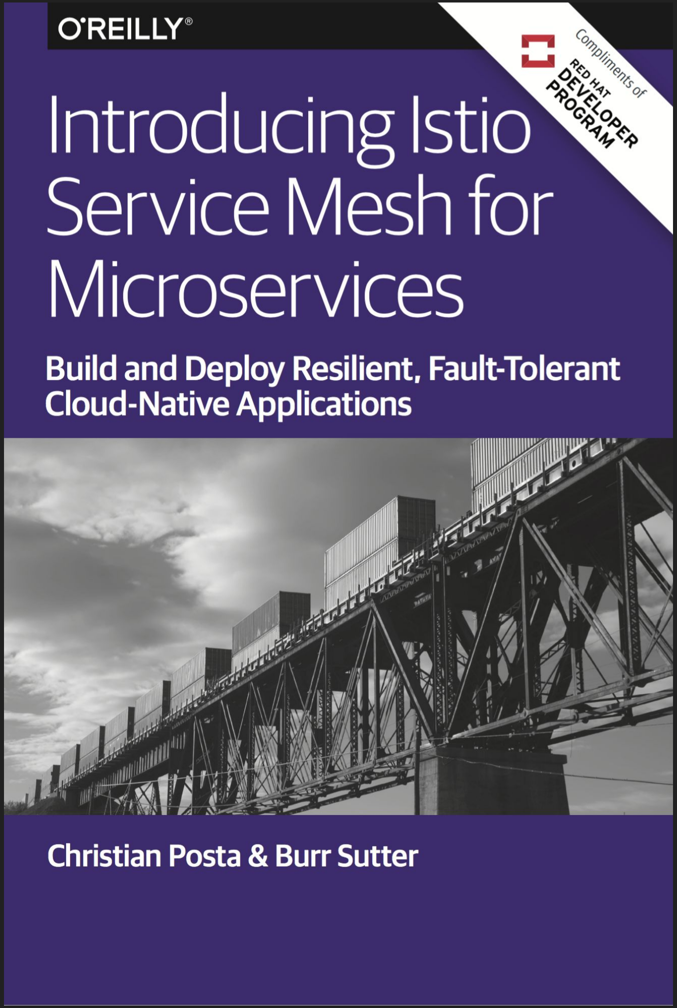
Migrating to Microservice Databases

From Relational Monolith to Distributed Data

Edson Yanaga

bit.ly/mono2microdb

@rafabene



O'REILLY®

Compliments of

RED HAT DEVELOPER PROGRAM

Introducing Istio Service Mesh for Microservices

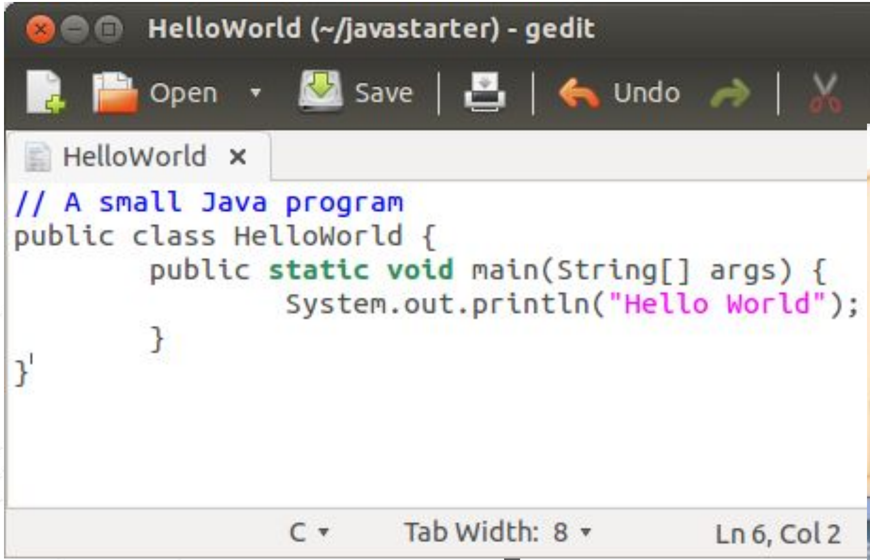
Build and Deploy Resilient, Fault-Tolerant Cloud-Native Applications

Christian Posta & Burr Sutter

bit.ly/istio-book

@rafabene

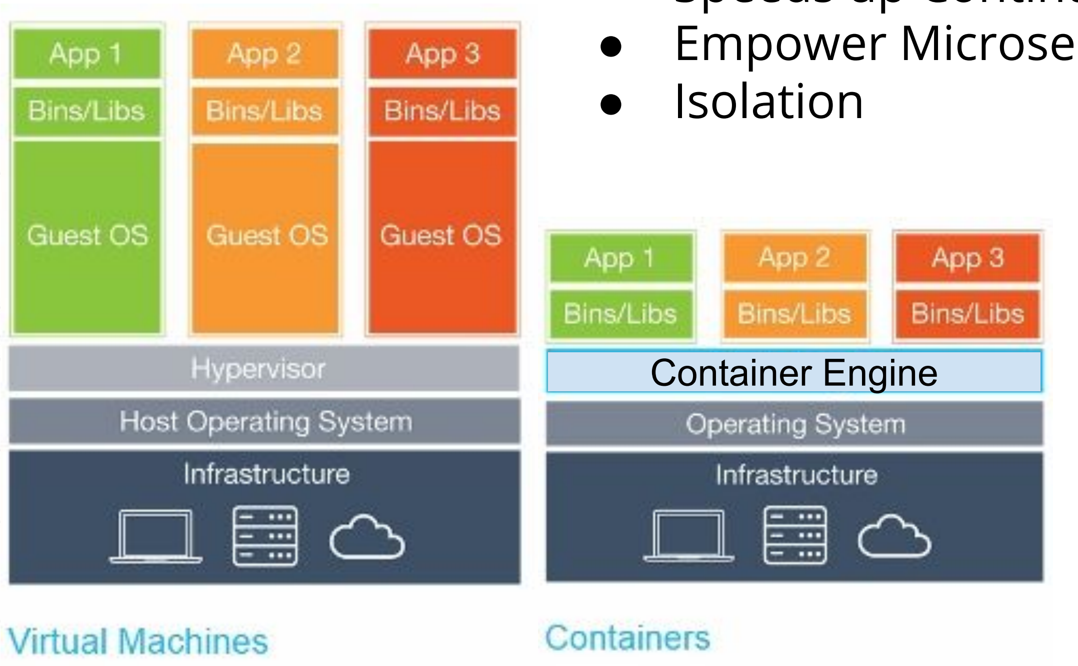
Why do you want to run your application inside containers?



```
// A small Java program
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

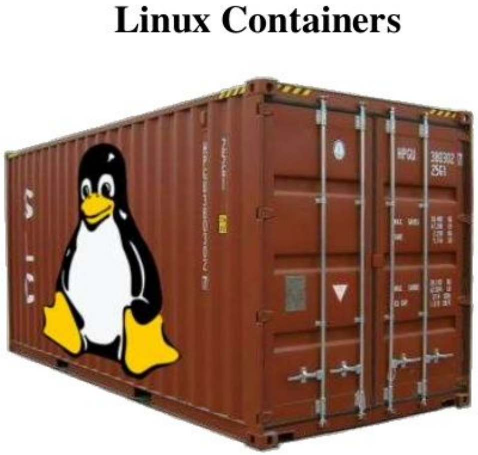
Container Advantages

- Lightweight footprint and minimal overhead,
- Portability across machines,
- Simplify DevOps practices,
- Speeds up Continuous Integration,
- Empower Microservices Architectures.
- Isolation



A way to run a Linux container:

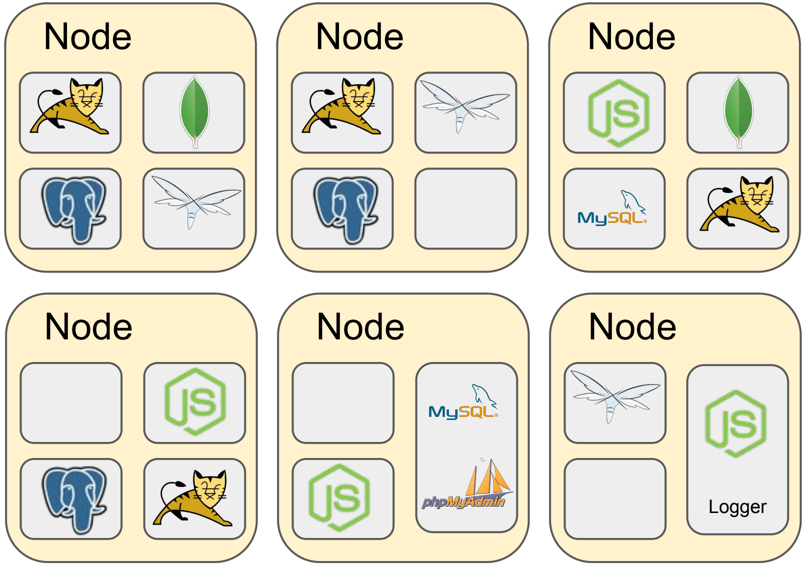
```
$ docker run -d <image-name>
```



A single and isolated Linux process running in a single machine

DevOps challenges for multiple containers

- How to scale?
- How to avoid port conflicts?
- How to manage them in multiple hosts?
- What happens if a host has a trouble?
- How to keep them running?
- How to update them?
- Where are my containers?



Meet Kubernetes

Greek for *“Helmsman”*; also the root of the word *“Governor”* *(from latin: gubernator)*

- Container **orchestrator**
- Supports **multiple cloud** and **bare-metal** environments
- Inspired by Google’s experience with containers
- **Open source**, written in **Go**

Manage **applications**, not machines

Open Source community

Project

Version 1.11

Hosted on GitHub

1700+ contributors

67,000+ commits

38,000+ GitHub stars

Partners

Red Hat

HP

IBM

Mesosphere

Microsoft

CoreOS

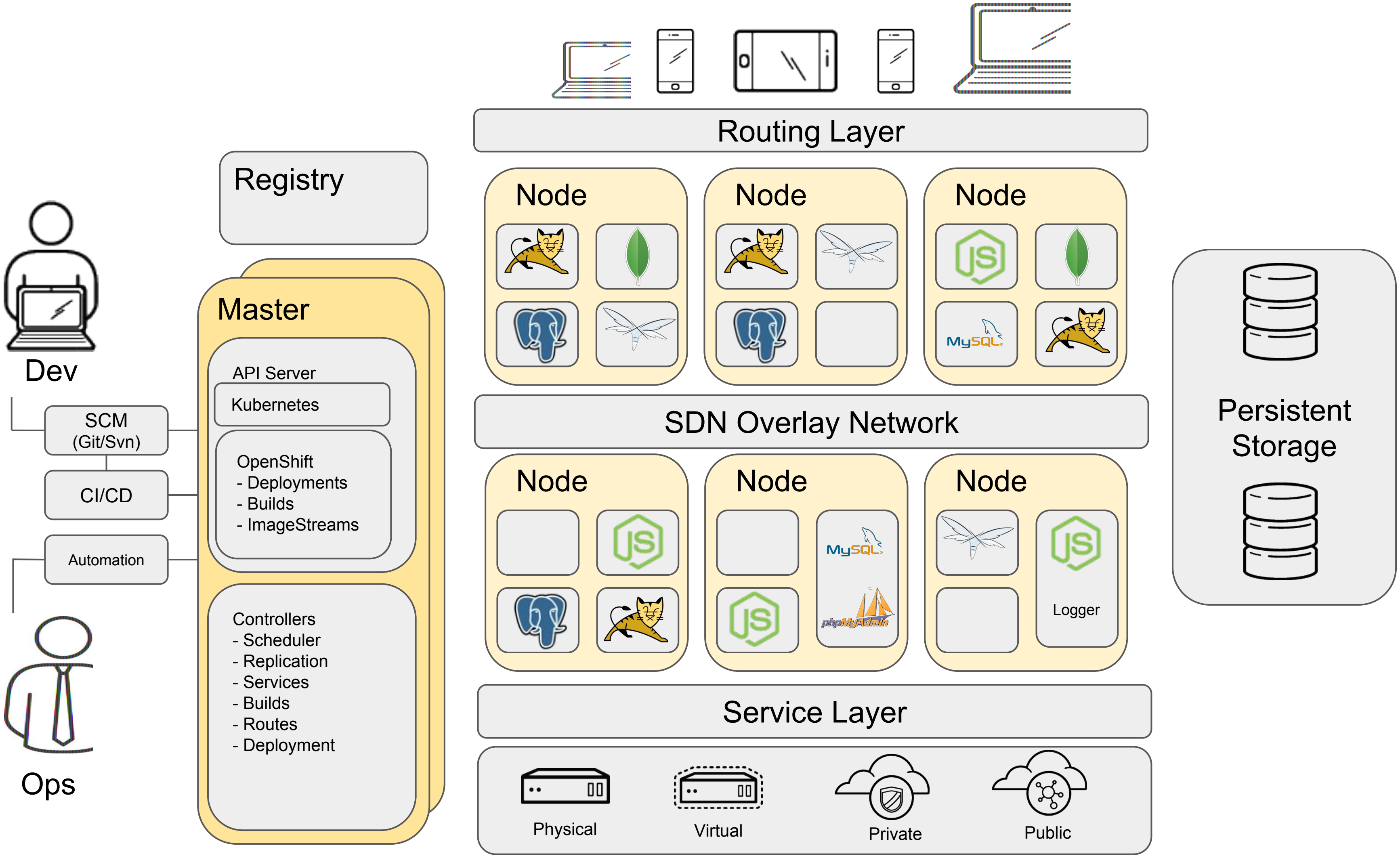
Pivotal

SaltStack

VMWare

http://kubernetes.io/

https://github.com/kubernetes/kubernetes



Routing Layer

Registry

Dev

SCM
(Git/Svn)

CI/CD

Automation

Ops

Master

API Server

Kubernetes

OpenShift
- Deployments
- Builds
- ImageStreams

Controllers
- Scheduler
- Replication
- Services
- Builds
- Routes
- Deployment

Node

Node

Node

SDN Overlay Network

Node

Node

Node

Logger

Service Layer

Physical

Virtual

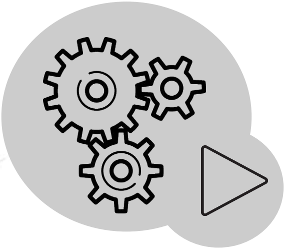
Private

Public

Persistent Storage

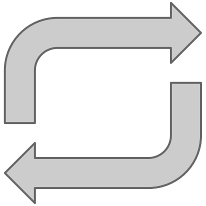
Kubernetes Concepts

Pod



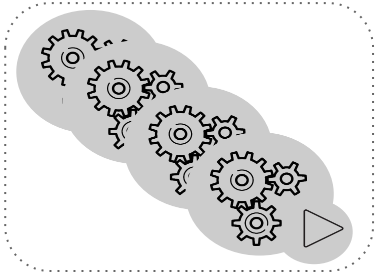
One or More Containers
Shared IP
Shared Storage Volume
Shared Resources
Shared Lifecycle

Replication Controller / Deployment



Ensures that a specified number of pod replicas are running at any one time

Service



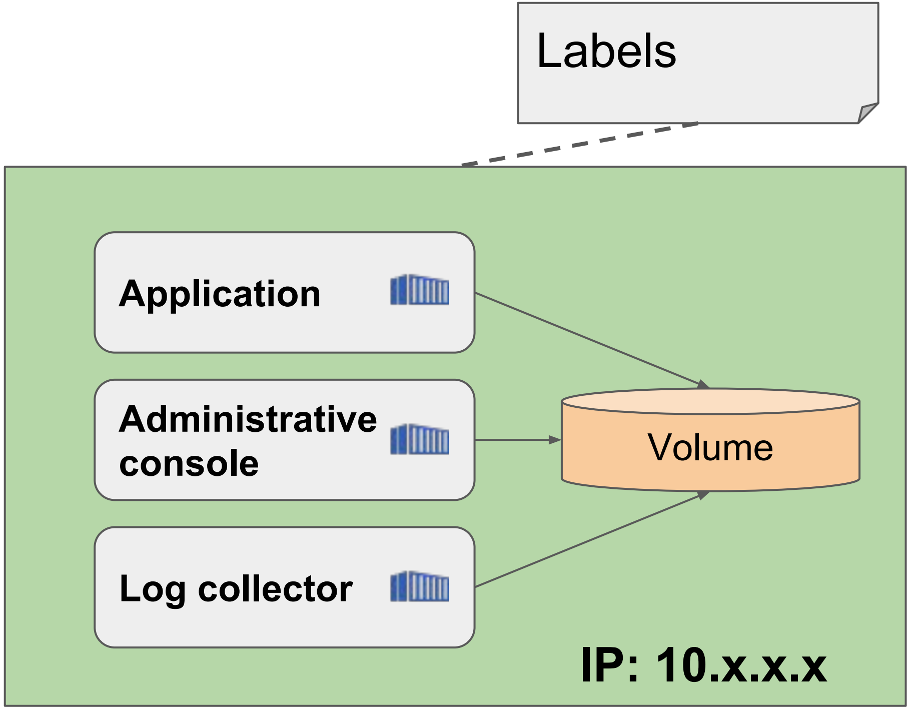
Grouping of pods, act as one, has stable virtual IP and DNS name

Label

Key/Value pairs associated with Kubernetes objects (e.g. env=production)

Concept: Pod

- Group of containers
- Live and die together
- Share:
  - IP
  - Secrets
  - Labels *
  - Volumes *

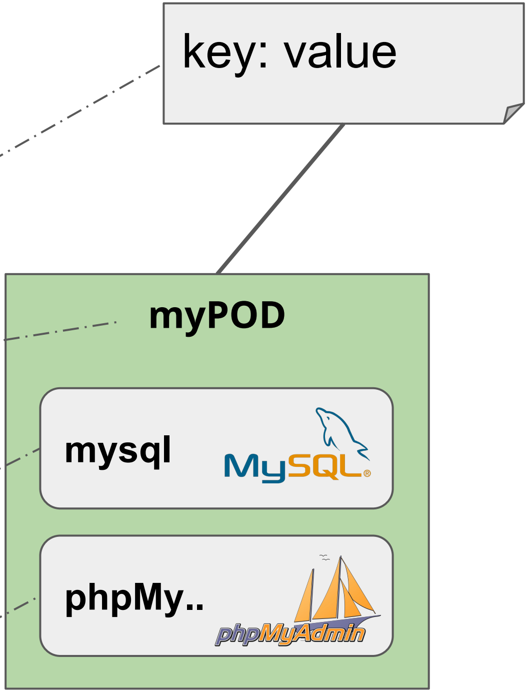


** we will talk about these concepts later*

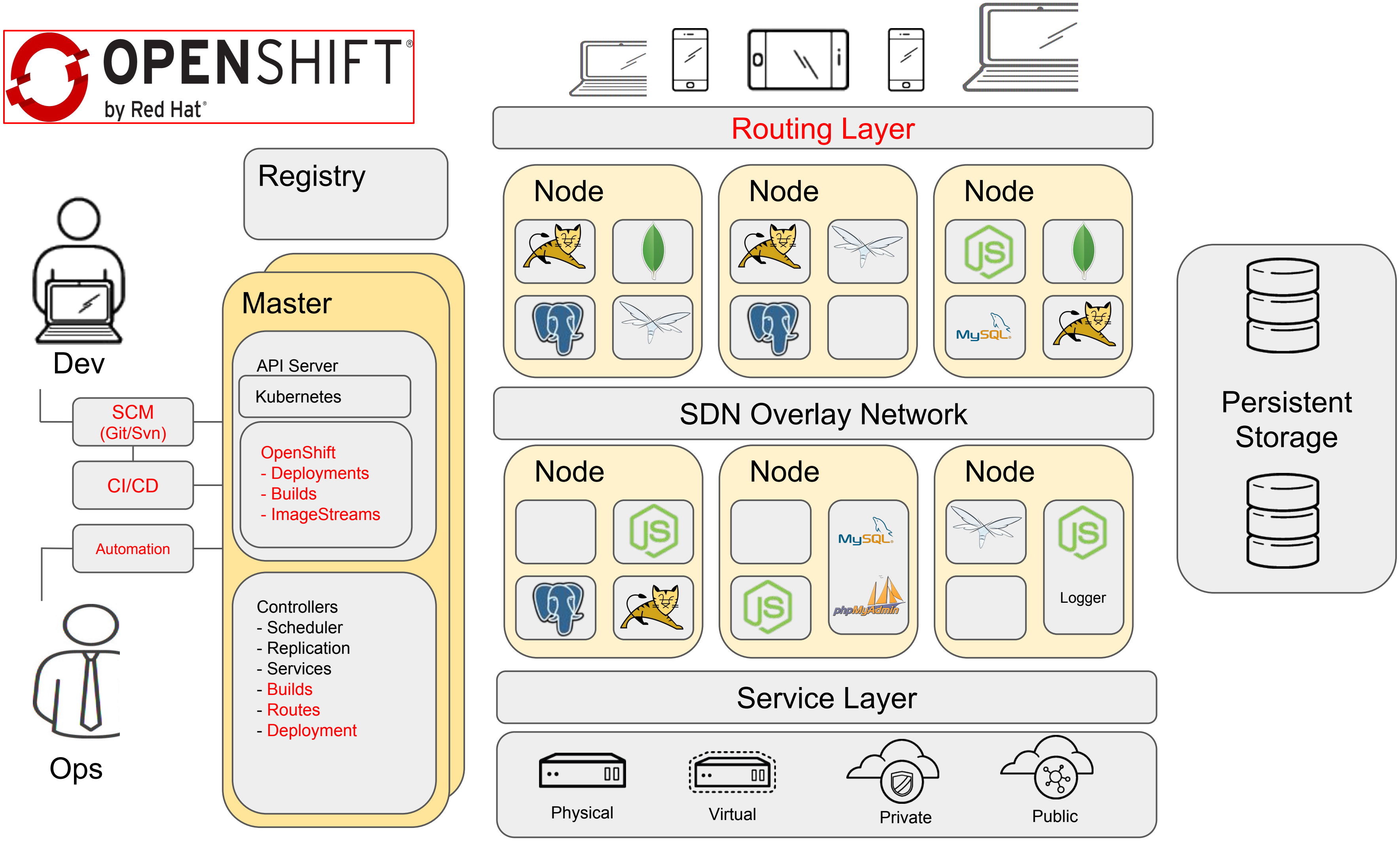
Concept: POD

Defining a POD as YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: myPod
  labels:
    key: value
spec:
  containers:
    - name: mysql
      image: username/image
    - name: phpMyAdmin
      image: username/image2
```



OPENSHIFT by Red Hat



Routing Layer

Registry

Dev

SCM (Git/Svn)

CI/CD

Automation

Ops

Master

API Server

Kubernetes

OpenShift
- Deployments
- Builds
- ImageStreams

Controllers
- Scheduler
- Replication
- Services
- Builds
- Routes
- Deployment

Node

Node

Node

SDN Overlay Network

Node

Node

Node

Logger

Service Layer

Physical

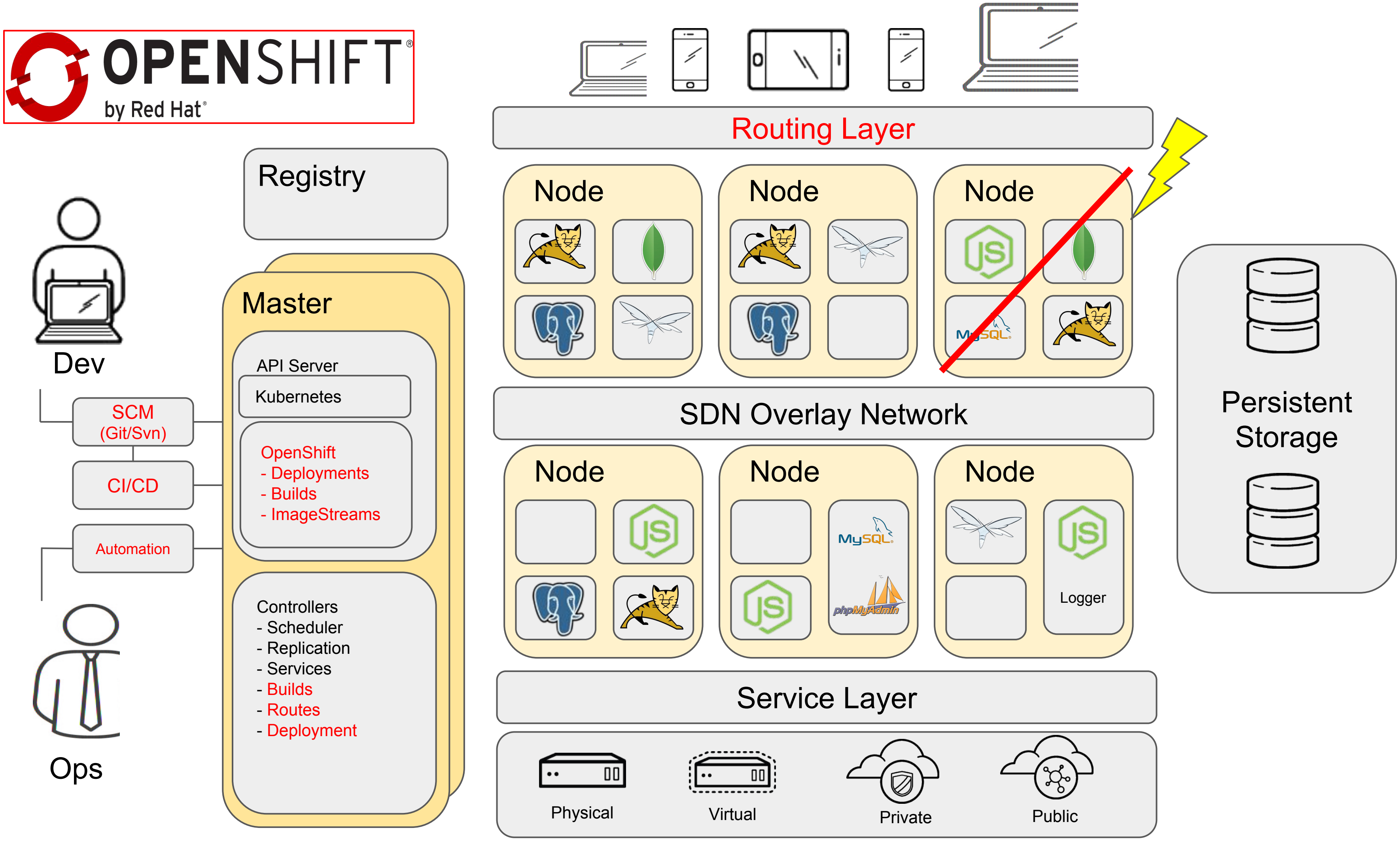
Virtual

Private

Public

Persistent Storage

OPENSHIFT by Red Hat



Registry

Dev

SCM (Git/Svn)

CI/CD

Automation

Ops

Master

API Server

Kubernetes

OpenShift
- Deployments
- Builds
- ImageStreams

Controllers
- Scheduler
- Replication
- Services
- Builds
- Routes
- Deployment

Routing Layer

Node

Node

Node

SDN Overlay Network

Node

Node

Node

Logger

Service Layer

Physical

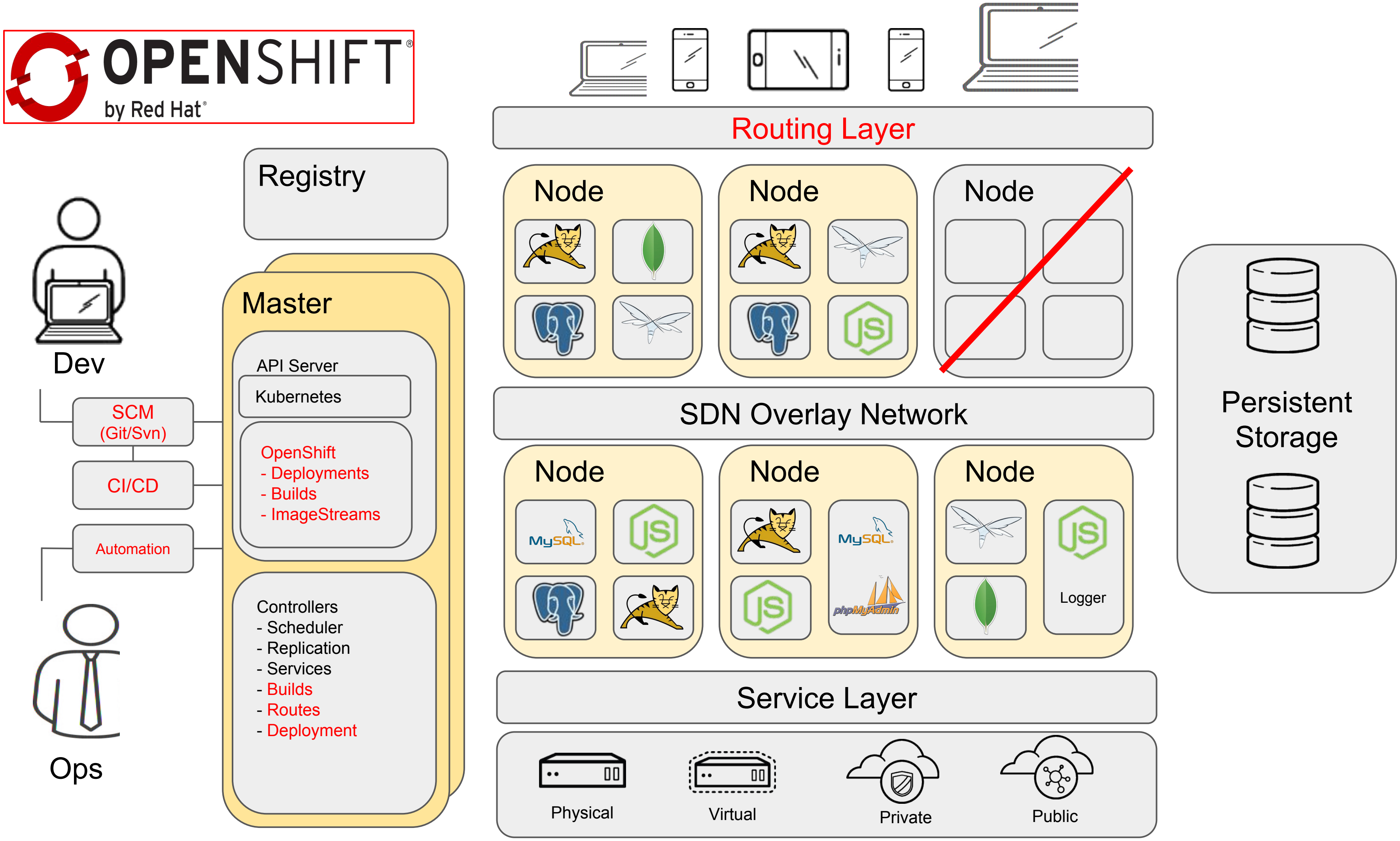
Virtual

Private

Public

Persistent Storage

OPENSHIFT by Red Hat



Registry

Dev

SCM (Git/Svn)

CI/CD

Automation

Ops

Master

API Server

Kubernetes

OpenShift
- Deployments
- Builds
- ImageStreams

Controllers
- Scheduler
- Replication
- Services
- Builds
- Routes
- Deployment

Routing Layer

Node

Node

Node

SDN Overlay Network

Node

Node

Node

Logger

Service Layer

Physical

Virtual

Private

Public

Persistent Storage

Concept: Replication Controllers / Deployment

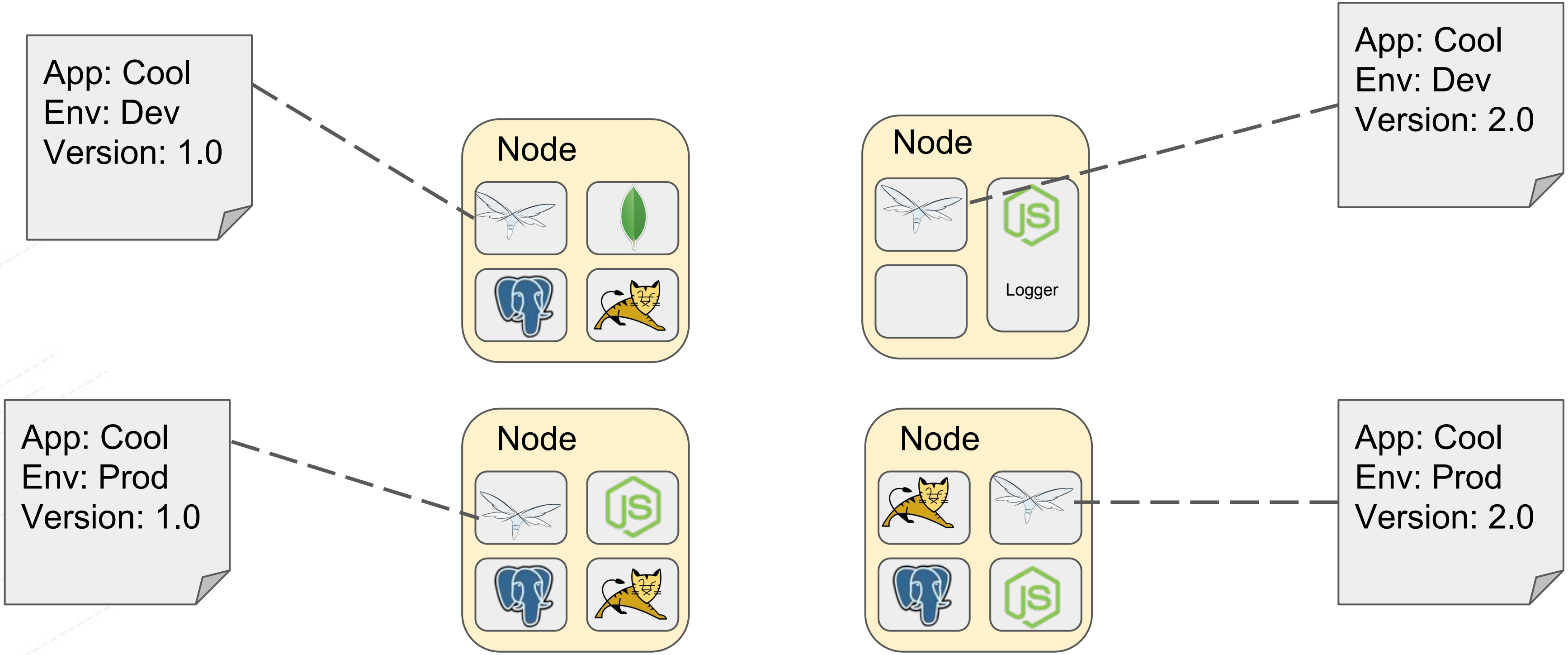
Defining a Deployment as YAML:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: myDeployment
spec:
  replicas: 4
  template:
    metadata:
    spec:
```

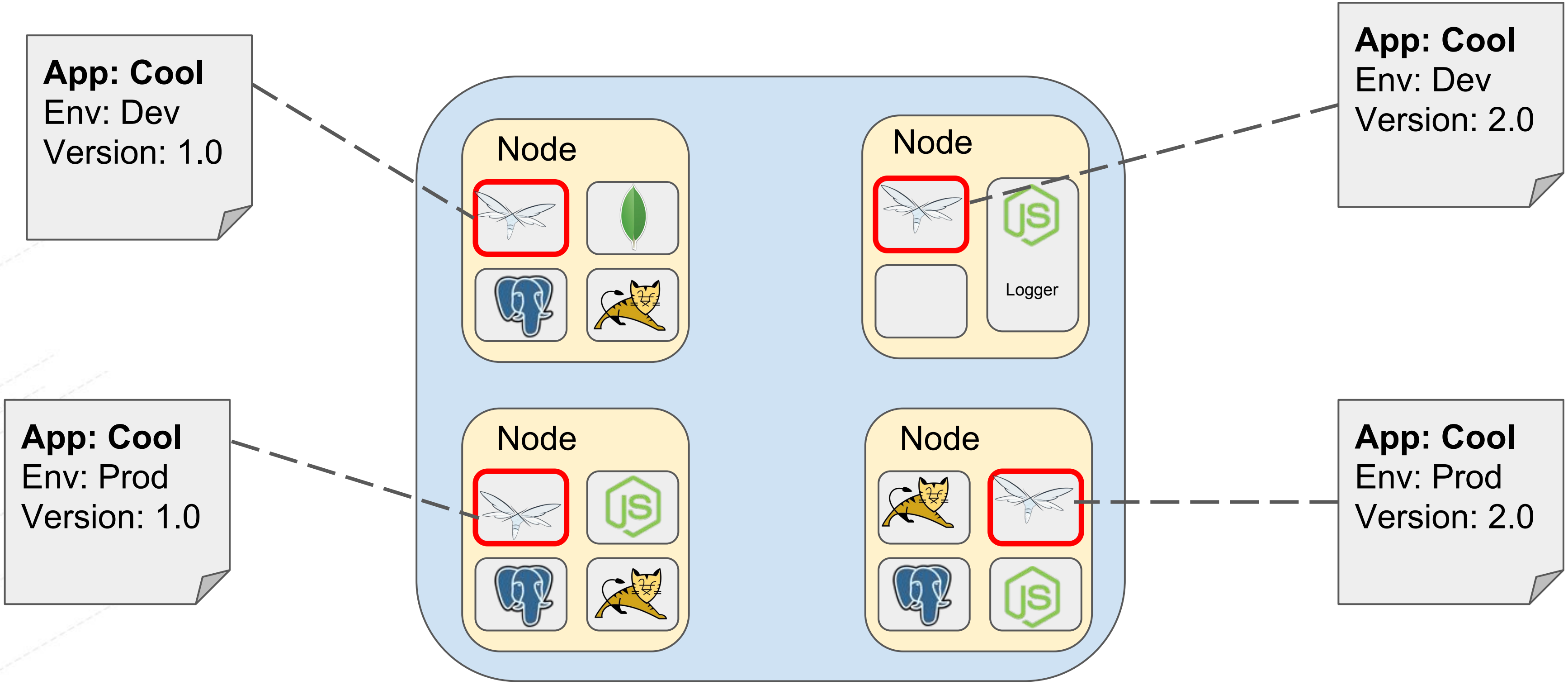
```
apiVersion: v1
kind: Pod
metadata:
  name: myPod
  labels:
    key: value
spec:
  containers:
    - name: myPod
      image: username/image
  ports:
      - name: http
        containerPort: 8080
```

Concept: Labels

Everything in Kubernetes can have a label



Concept: Labels



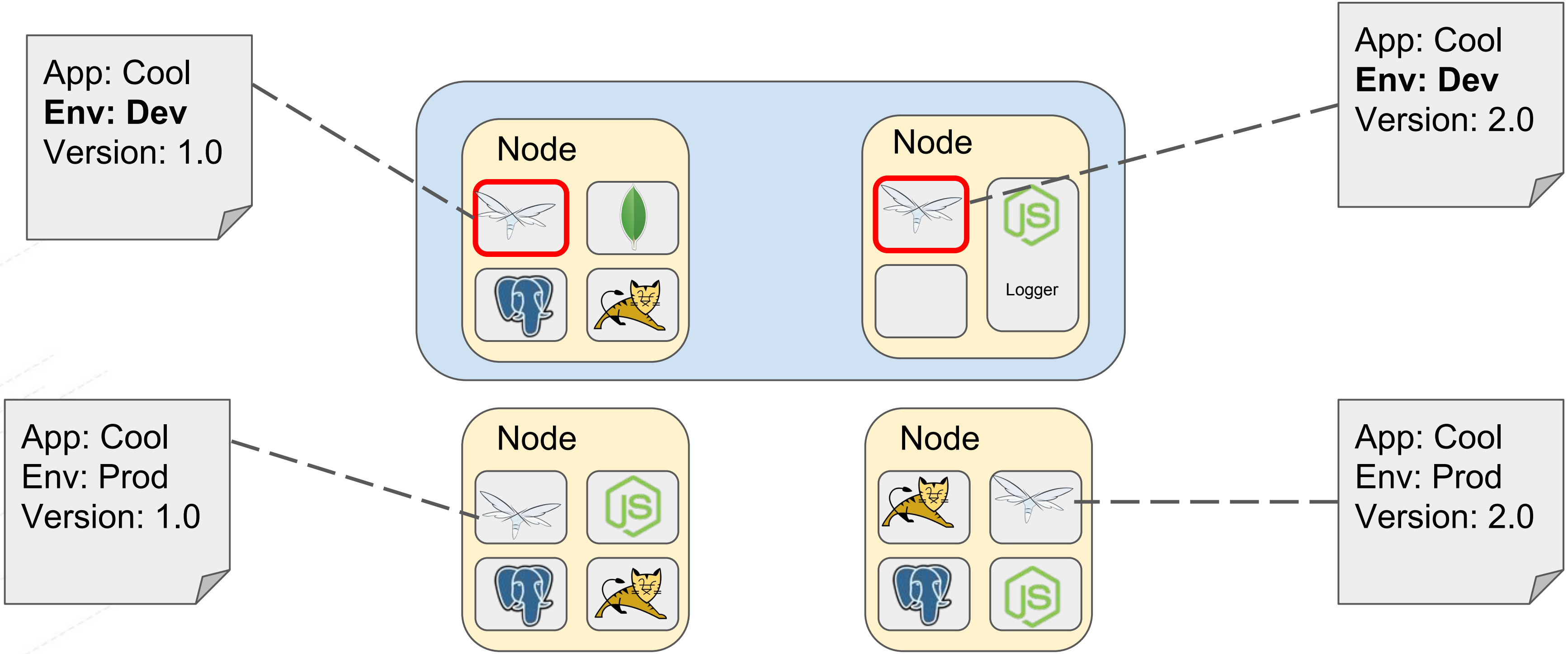
App: Cool
Env: Dev
Version: 1.0

App: Cool
Env: Dev
Version: 2.0

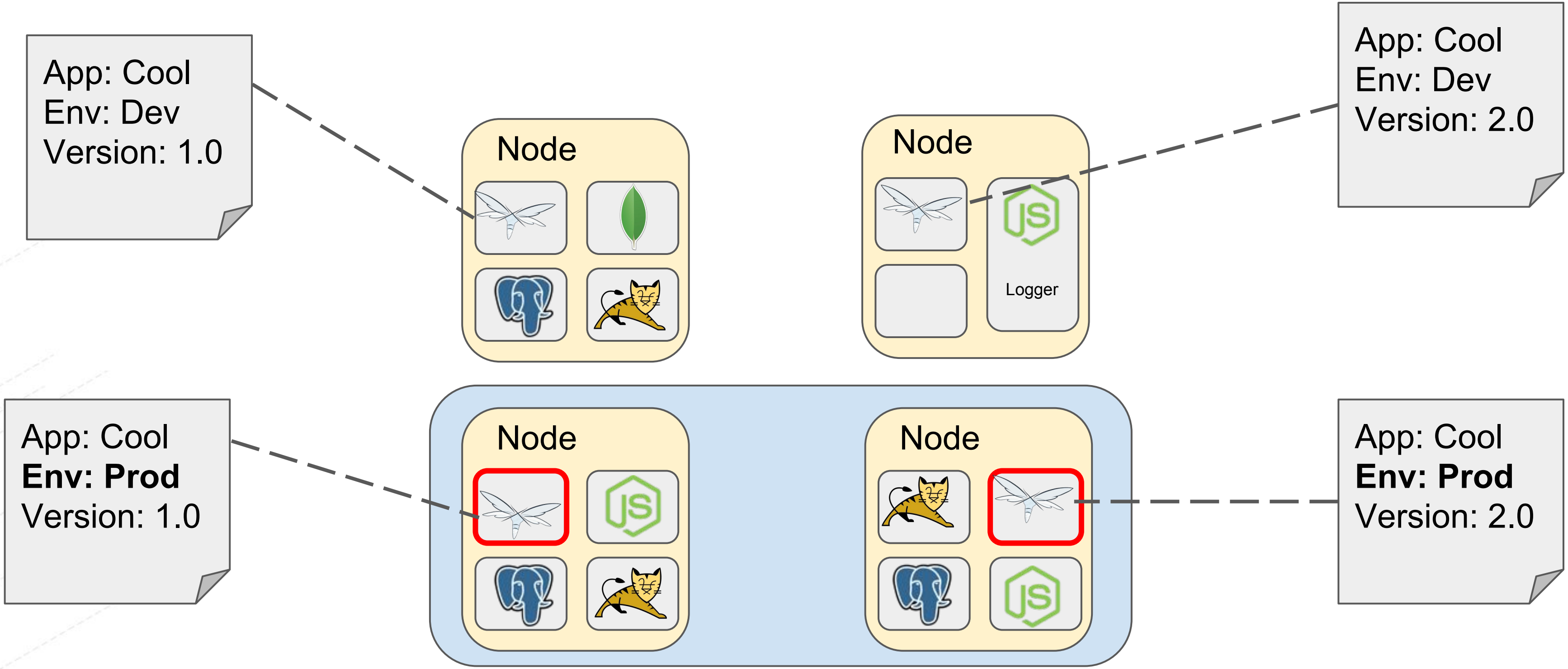
App: Cool
Env: Prod
Version: 1.0

App: Cool
Env: Prod
Version: 2.0

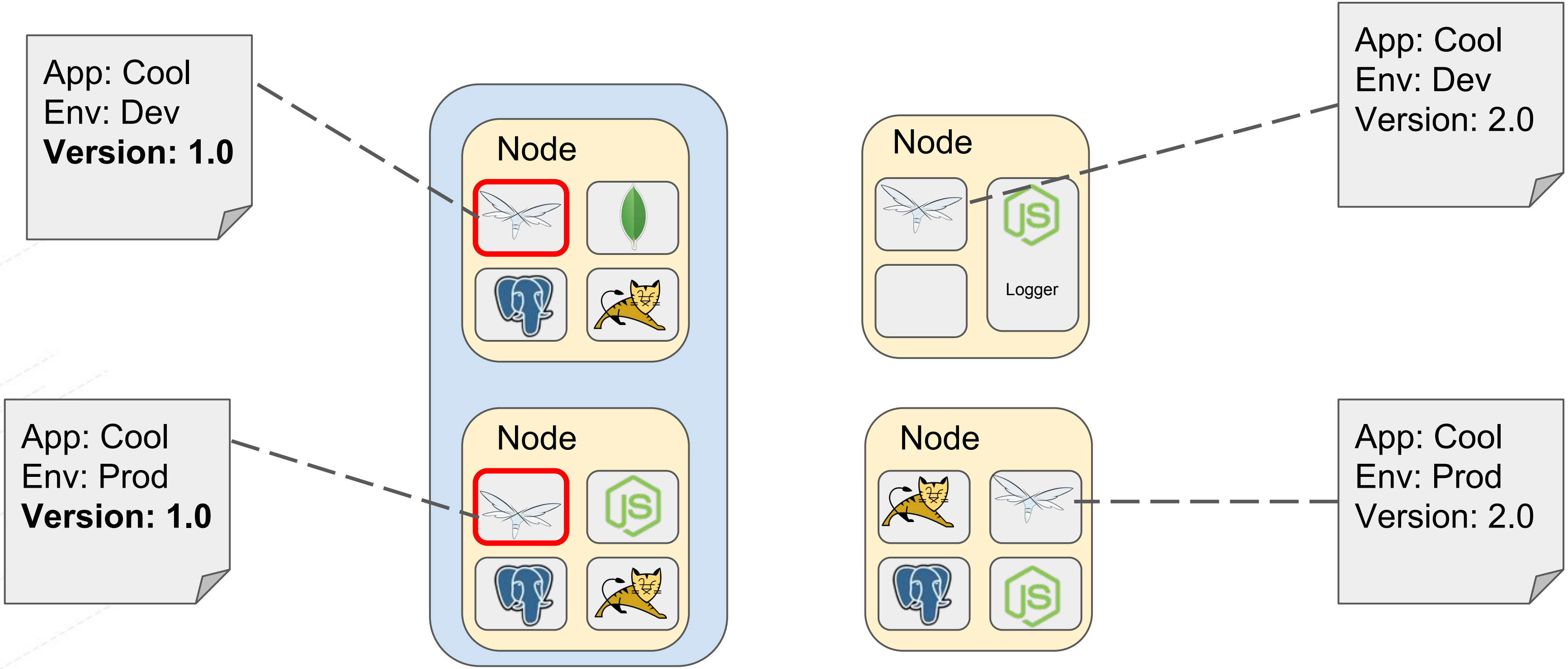
Concept: Labels



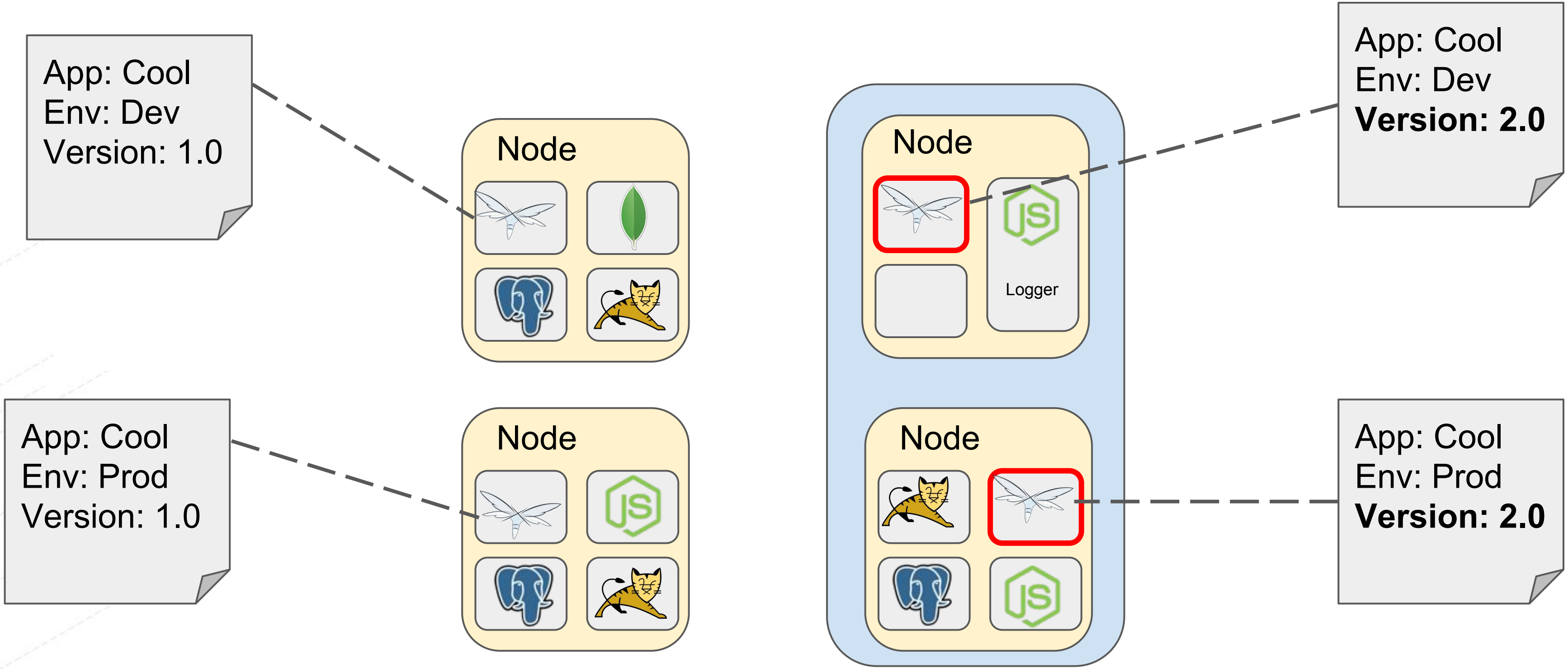
Concept: Labels



Concept: Labels



Concept: Labels



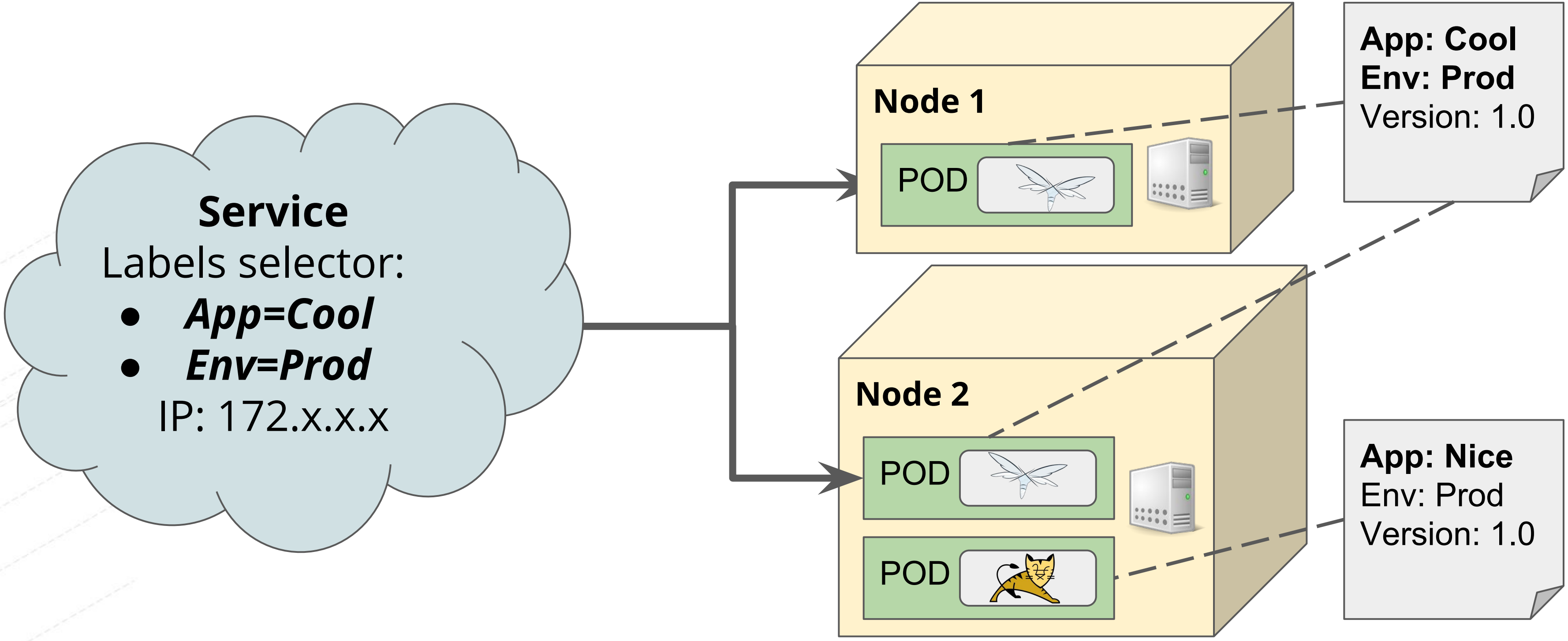
Concept: Labels

Defining Labels as YAML:
(can be placed in any object metadata)

```
metadata:
  name: objectName
  labels:
    App: Cool
    Env: Dev
    Version: 1.0
```

App: Cool
Env: Dev
Version: 1.0

Concept: Services



Service
Labels selector:
- ***App=Cool***
- ***Env=Prod***

IP: 172.x.x.x

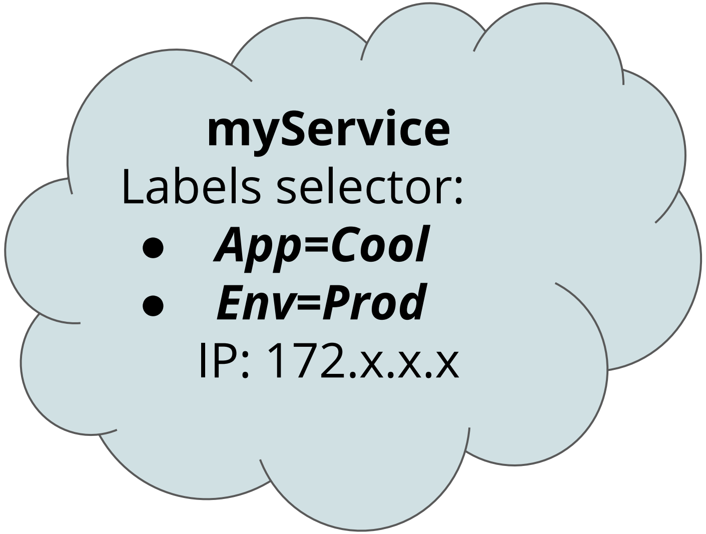
App: Cool
Env: Prod
Version: 1.0

App: Nice
Env: Prod
Version: 1.0

Concept: Services

Defining a Service as YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: myService
  labels:
        ...
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    App: Cool
    Env: Prod
```



myService

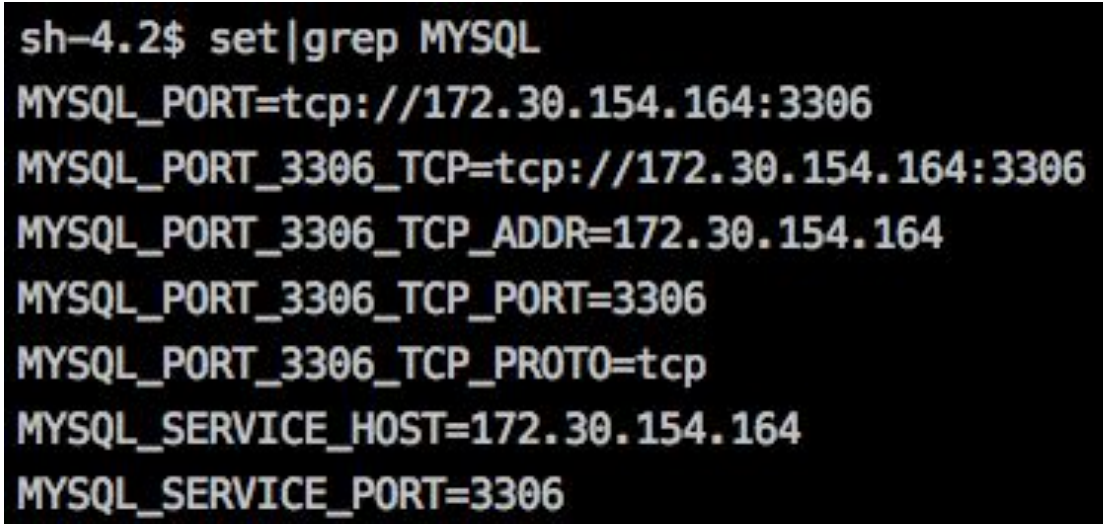
Labels selector:

- ***App=Cool***
- ***Env=Prod***

IP: 172.x.x.x

Service discovery inside Kubernetes

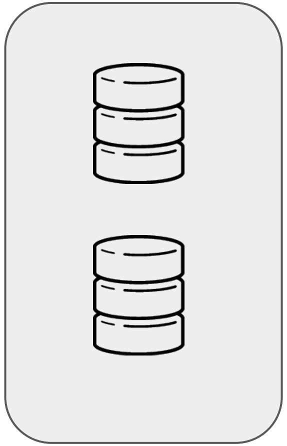
Using Environment variables:



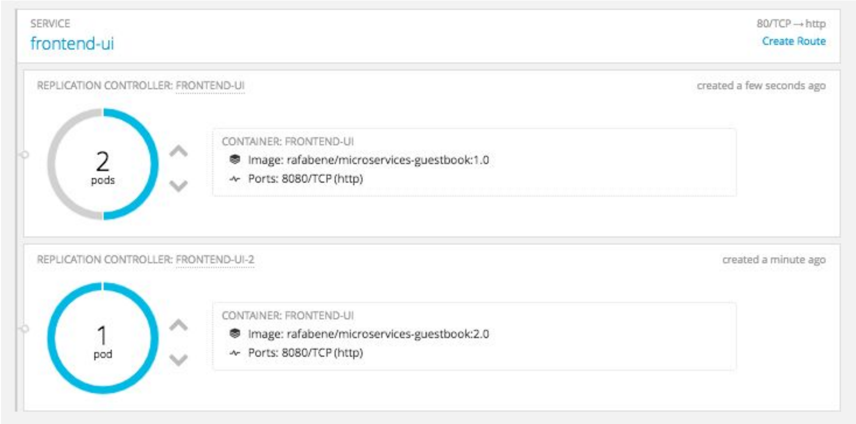
```
sh-4.2$ set|grep MYSQL
MYSQL_PORT=tcp://172.30.154.164:3306
MYSQL_PORT_3306_TCP=tcp://172.30.154.164:3306
MYSQL_PORT_3306_TCP_ADDR=172.30.154.164
MYSQL_PORT_3306_TCP_PORT=3306
MYSQL_PORT_3306_TCP_PROTO=tcp
MYSQL_SERVICE_HOST=172.30.154.164
MYSQL_SERVICE_PORT=3306
```

Using internal DNS: $ *ping mysql*

Other concepts



Persistent Volumes



SERVICE

frontend-ui

80/TCP → http

Create Route

REPLICATION CONTROLLER: FRONTEND-UI

created a few seconds ago

2 pods

CONTAINER: FRONTEND-UI

Image: rafabene/microservices-guestbook:1.0

Ports: 8080/TCP (http)

REPLICATION CONTROLLER: FRONTEND-UI-2

created a minute ago

1 pod

CONTAINER: FRONTEND-UI

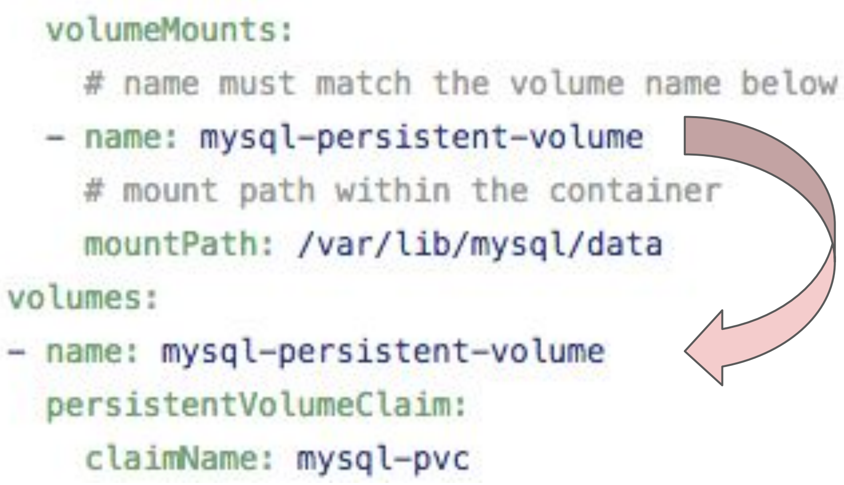
Image: rafabene/microservices-guestbook:2.0

Ports: 8080/TCP (http)

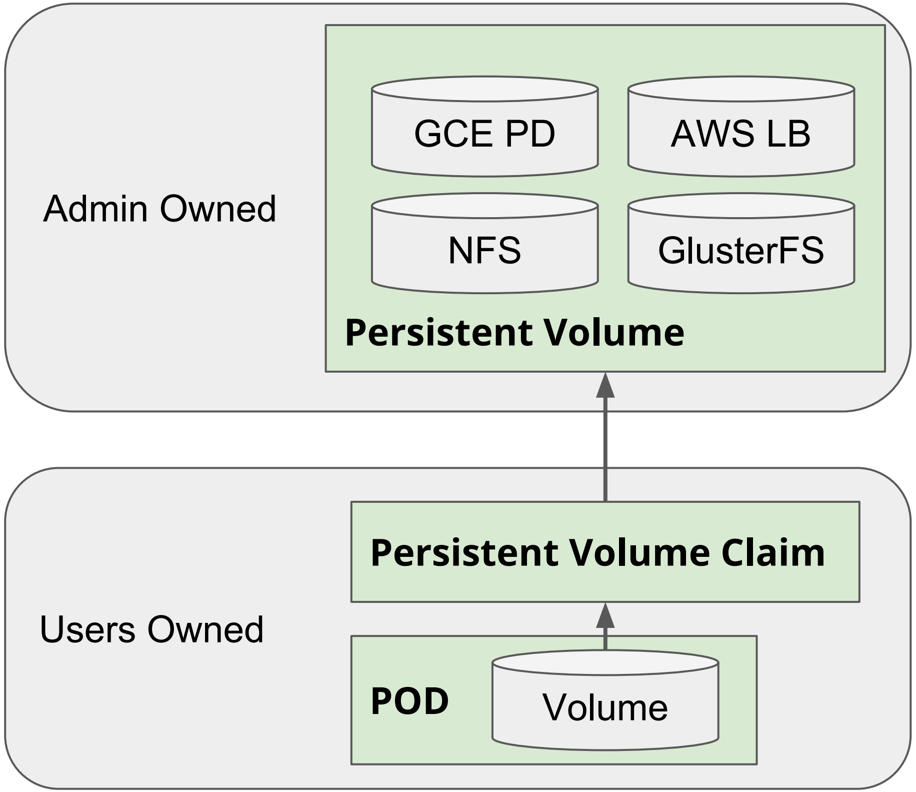
Rolling updates

Concept: Persistent Volumes

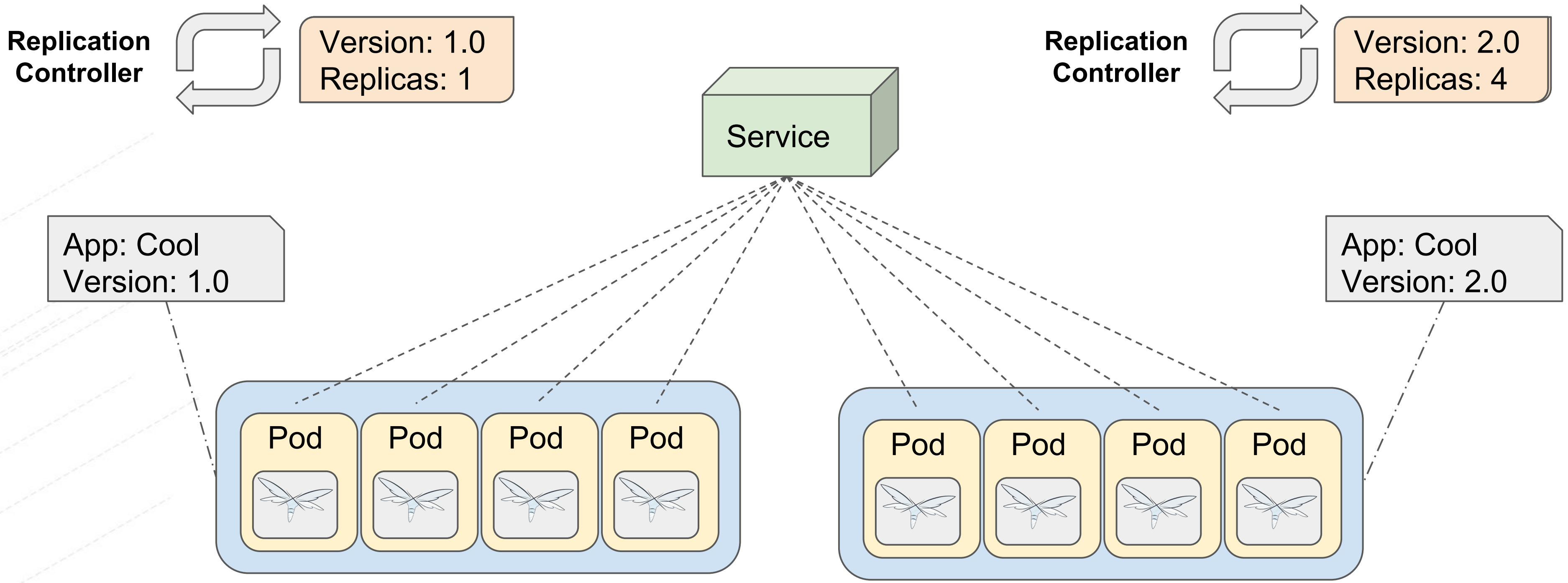
- Admin provisions them, Users claim them
- High-level abstraction
- Pods can mount PVCs as Volumes



```
  volumeMounts:
    # name must match the volume name below
  - name: mysql-persistent-volume
    # mount path within the container
    mountPath: /var/lib/mysql/data
volumes:
- name: mysql-persistent-volume
  persistentVolumeClaim:
    claimName: mysql-pvc
```



Deployment Concept: Rolling Updates



Replication Controller

Version: 1.0
Replicas: 1

Replication Controller

Version: 2.0
Replicas: 4

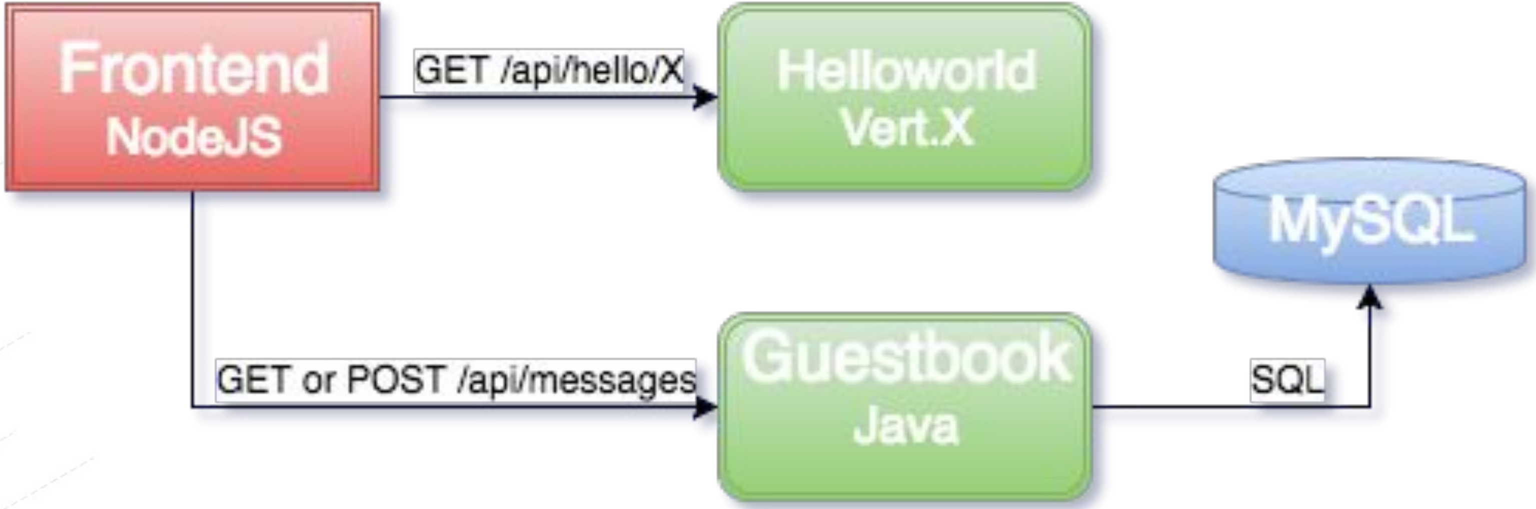
Service

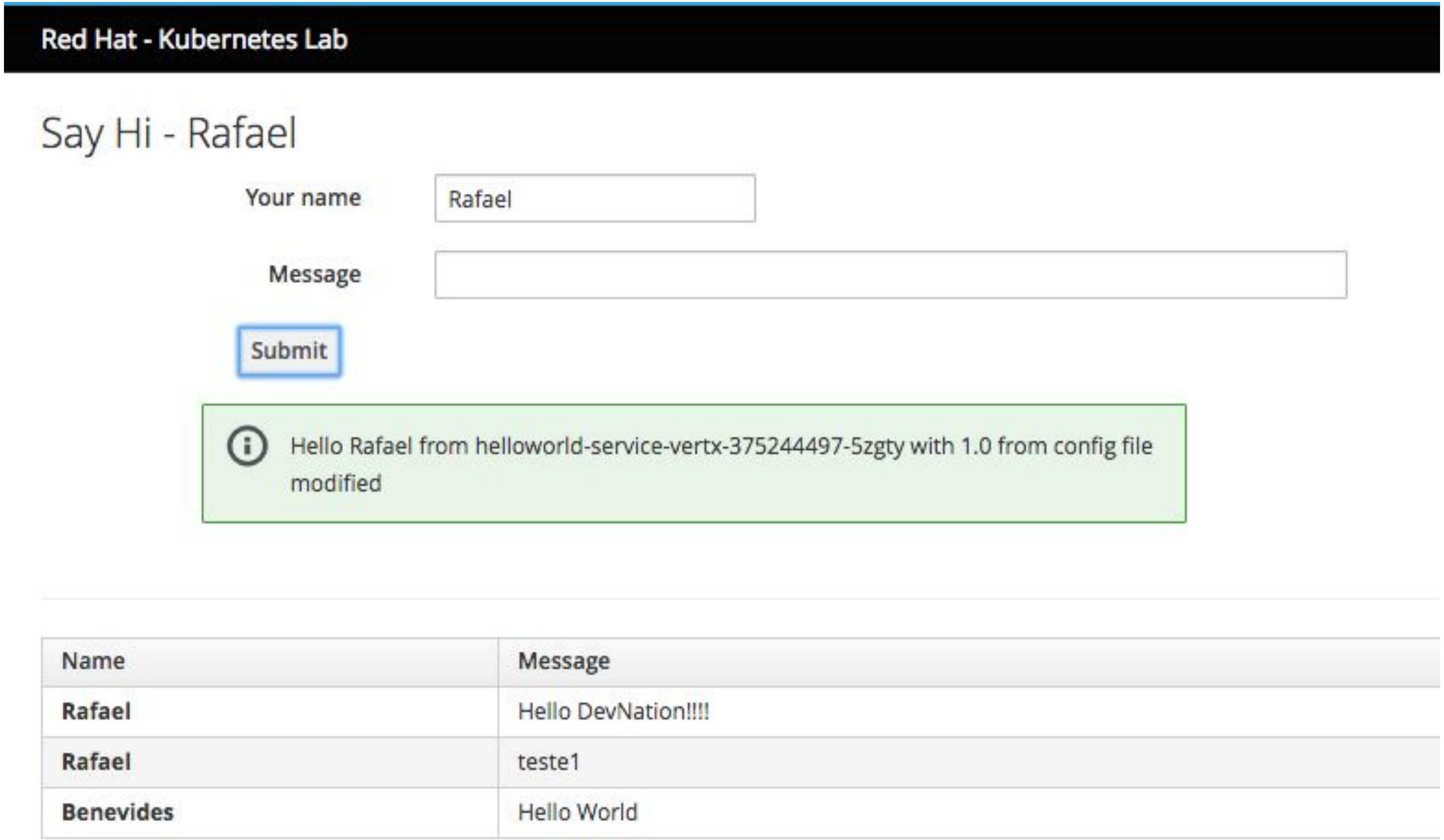
App: Cool
Version: 1.0

App: Cool
Version: 2.0

Kubernetes Example

Application Overview





Red Hat - Kubernetes Lab

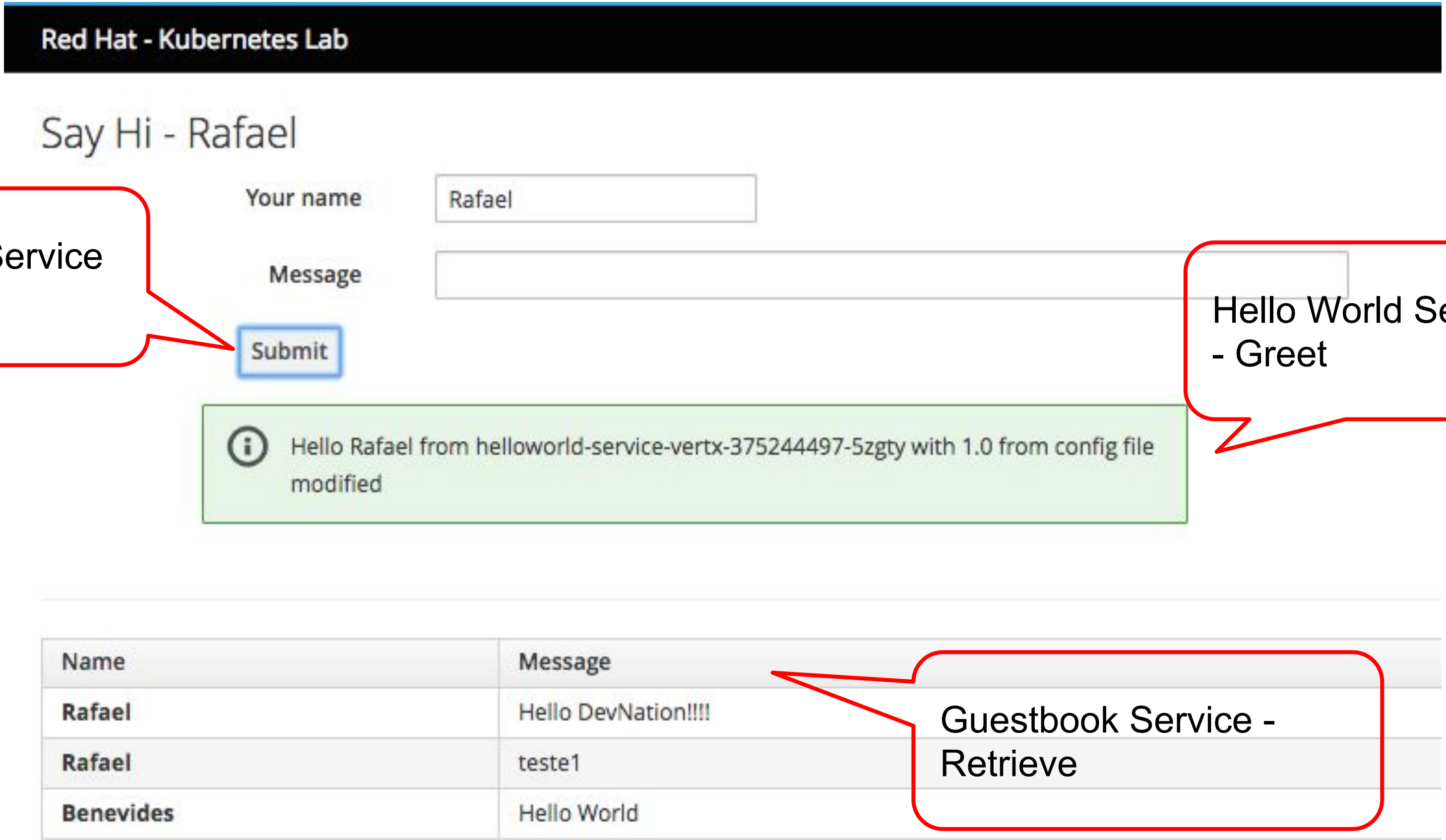
Say Hi - Rafael

Your name: Rafael

Message:

Hello Rafael from helloworld-service-vertx-375244497-5zgty with 1.0 from config file modified

Name	Message
Rafael	Hello DevNation!!!!
Rafael	teste1
Benevides	Hello World



Red Hat - Kubernetes Lab

Say Hi - Rafael

Your name: Rafael

Message:

Hello Rafael from helloworld-service-vertx-375244497-5zgty with 1.0 from config file modified

Name	Message
Rafael	Hello DevNation!!!!
Rafael	teste1
Benevides	Hello World

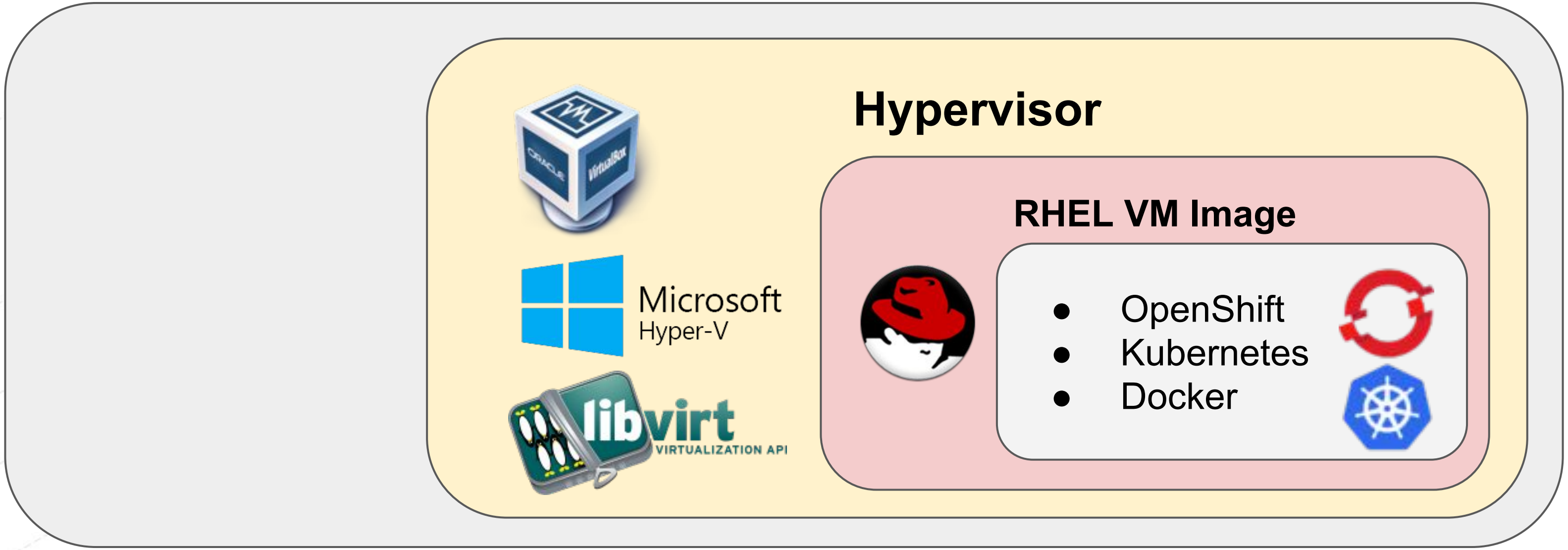
Guestbook Service - Create

Hello World Service - Greet

Guestbook Service - Retrieve

Lab infrastructure

minishift or CDK



Hypervisor

RHEL VM Image

- OpenShift
- Kubernetes
- Docker

http://developers.redhat.com/products/cdk/download/

Kubernetes lab

VERY IMPORTANT

http://bit.ly/kubernetes-lab

Follow me on the Setup environment section!

@RAFABENE

RED HAT®
DEVELOPER
PROGRAM

Kubernetes Introduction

@rafabene

benevides@redhat.com

Link → http://bit.ly/kubernetes-intro