



RED HAT DEVELOPERS

8 Steps to Becoming Awesome
with Kubernetes

Link

@burrsutter

burr@redhat.com

<https://developers.redhat.com/devnationlive/>

<http://bit.ly/8stepsawesome>

Your Journey to Awesomeness



Re-Org to
DevOps



Self-Service,
On-Demand,
Elastic
Infrastructure



Automation
Puppet, Chef,
Ansible,
Kubernetes



CI & CD
Deployment
Pipeline

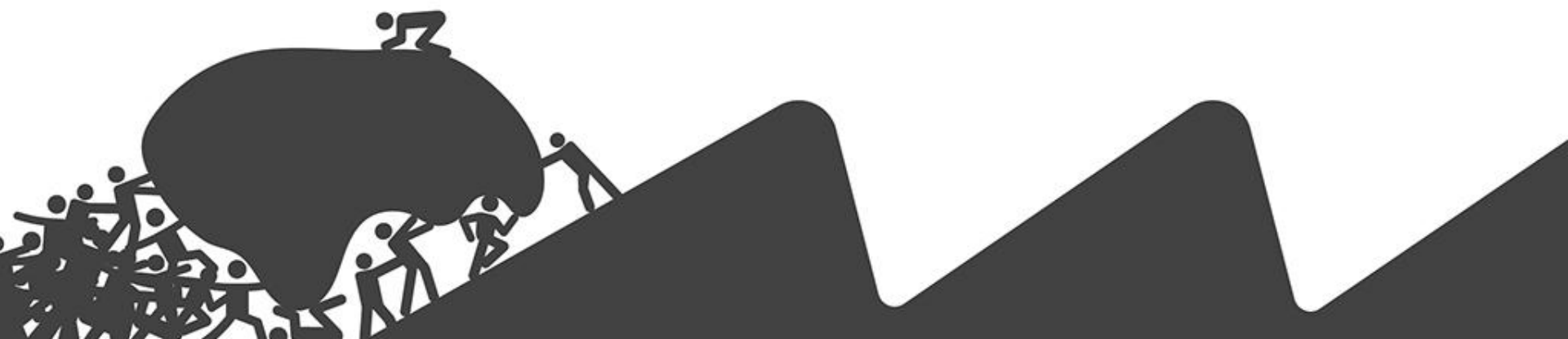


Advanced
Deployment
Techniques



Microservices







8 Steps

0 - Introduction

1 - Installation & Getting Started

2 - Building Images, Running Containers

3 - oc/kubectl exec magic

4 - Logs

5 - Service discovery & load-balancing

6 - Live & Ready

7 - Rolling updates, Canaries, Blue/Green

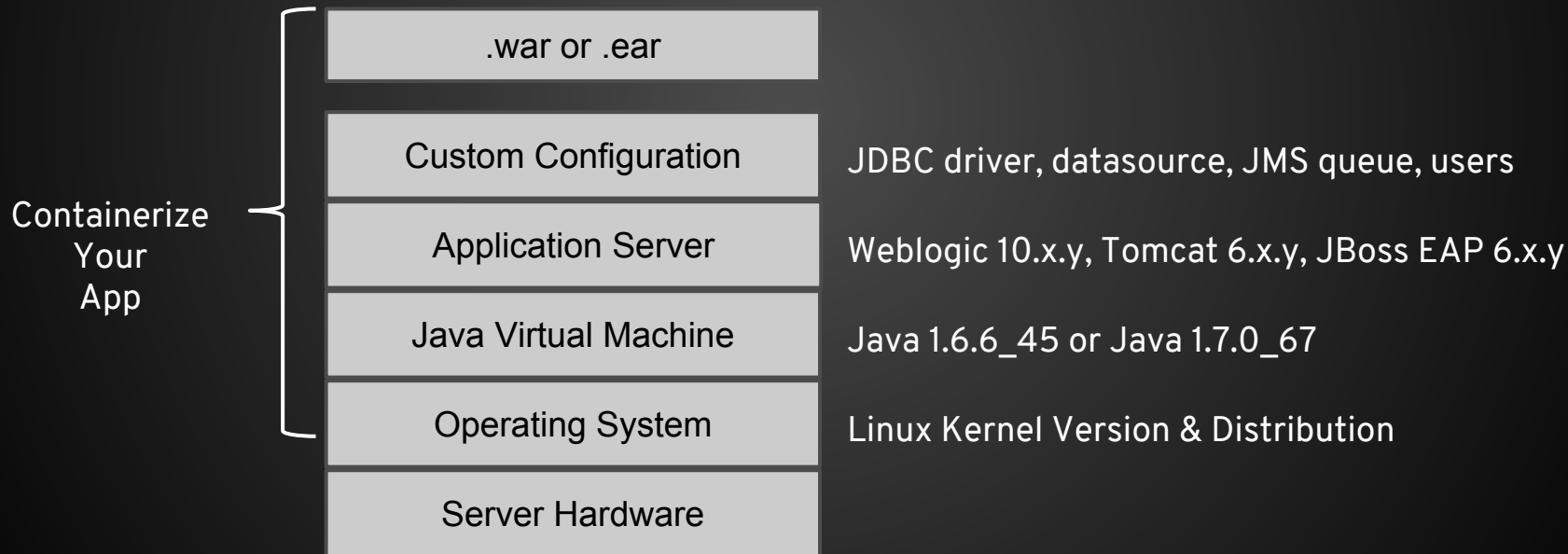
8 - Debugging

Bonus

Step 0: Introduction

A Challenge

Have you ever had “/” vs “\” break your app? Or perhaps needed a unique version of a JDBC driver? Or had a datasource with a slightly misspelled JNDI name? Or received a patch for the JVM or app server that broke your code?



Email

MyApp.war has been tested with the following

On my Windows 7 desktop

JDK 1.8.43

Wildfly 9

Configuration:

Datasource: MySQLDS

Tested with: mysql-connector-java-5.1.31-bin.jar

Production Environment

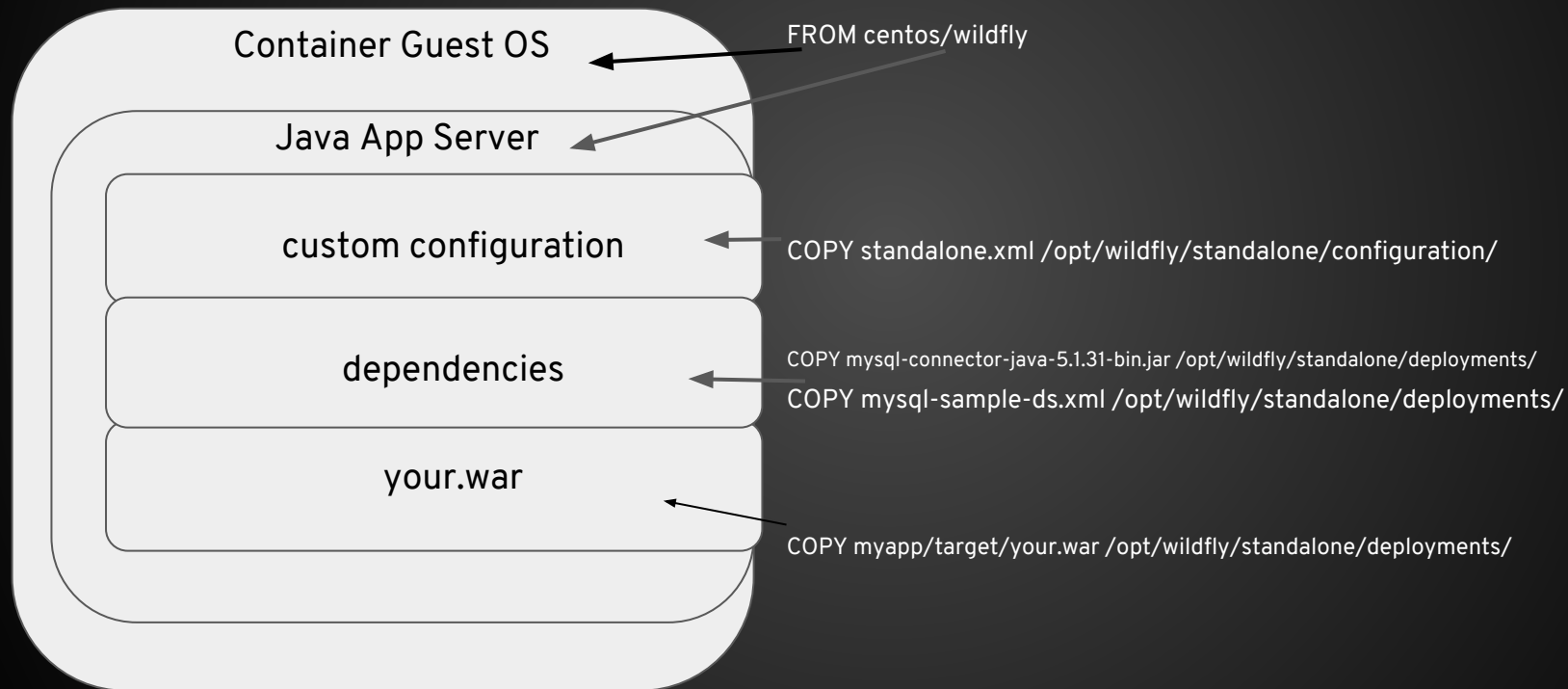
Red Hat Enterprise Linux 6.2

JRE 1.7.3

WebSphere 8.5.5

Oracle 9

Dockerfile



Docker Tutorial

bit.ly/docker-devnexus2017

DevOps Challenges for Multiple Containers

- How to scale?
- How to avoid port conflicts?
- How to manage them on multiple hosts?
- What happens if a host has 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

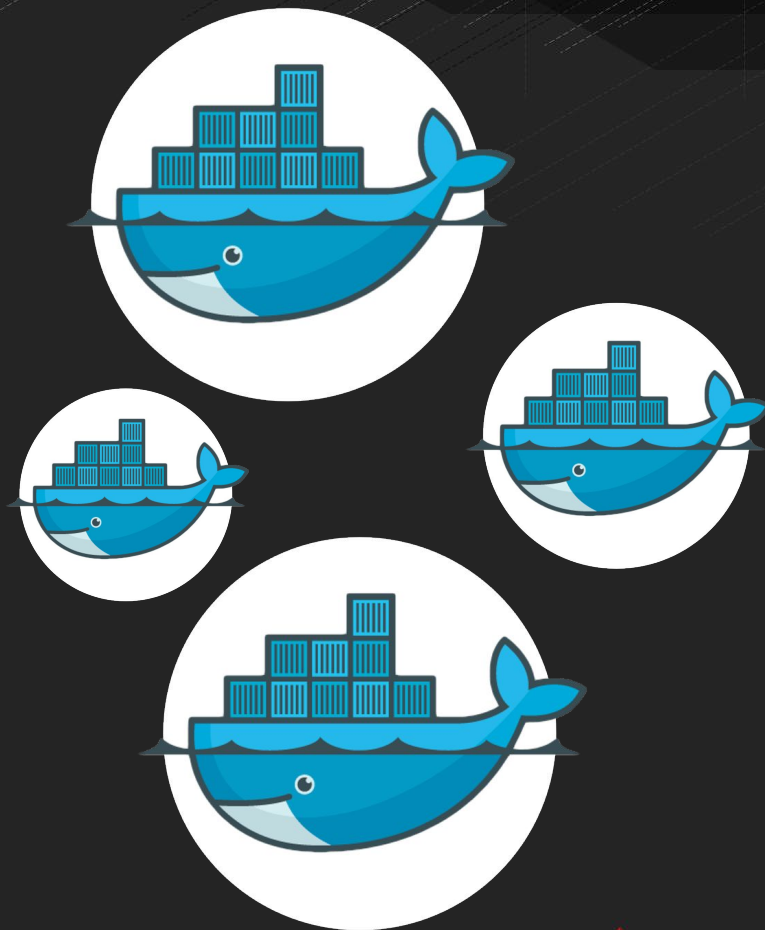


Pods

A group of whales is commonly referred to as a pod and a pod usually consists a group of whales that have bonded together either because of biological reasons or through friendships developed between two or more whales.

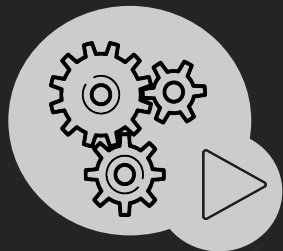
In many cases a typical whale pod consists of anywhere from 2 to 30 whales or more.*

*<http://www.whalefacts.org/what-is-a-group-of-whales-called/>



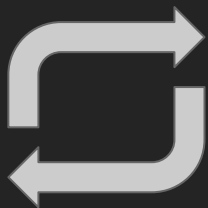
Kubernetes Cluster

Pod



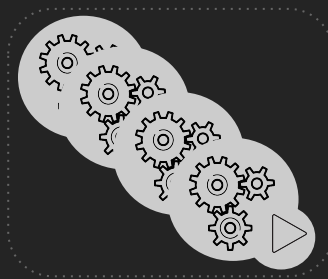
- ✓ 1+ containers
- ✓ Shared IP
- ✓ Shared storage volume
- ✓ Shared resources
- ✓ Shared lifecycle

Replication
Controller/
Deployment



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

Service



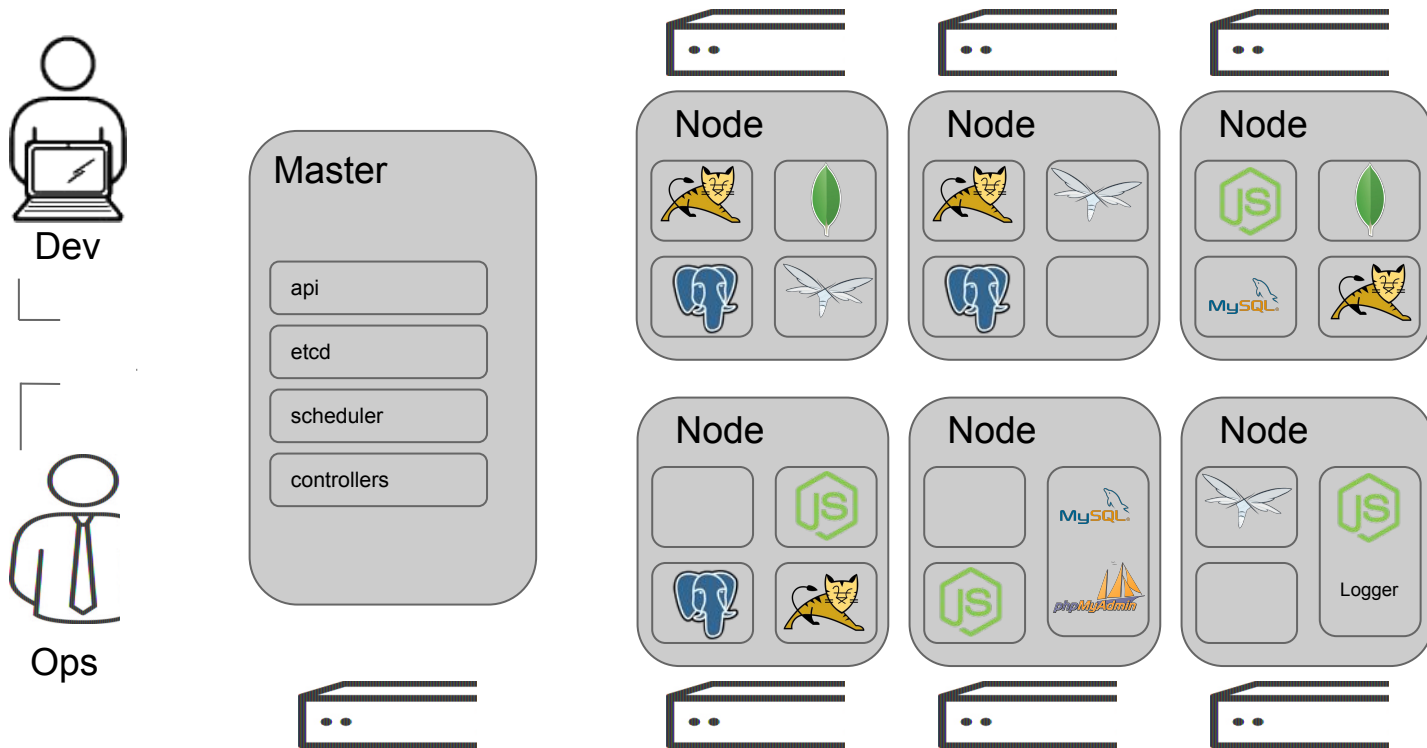
- ✓ Grouping of pods (acting as one) has stable virtual IP and DNS name

Label



- ✓ Key/Value pairs associated with Kubernetes objects (env=production)

Kubernetes Cluster



Key Kubernetes Capabilities

- Self-healing
- Horizontal Manual & Auto Scaling
- Automatic Restarting
- Scheduled across hosts
- Built-in load-balancer
- Rolling upgrades

Kubernetes Commands

<https://kubernetes.io/docs/user-guide/kubectl/>

```
kubectl get namespaces
```

```
kubectl get pods
```

```
kubectl run myvertex --image=burr/myvertex:v1 --port=8080
```

```
kubectl logs myvertex-kk605
```

```
kubectl expose deployment --port=8080 myvertex --type=LoadBalancer
```

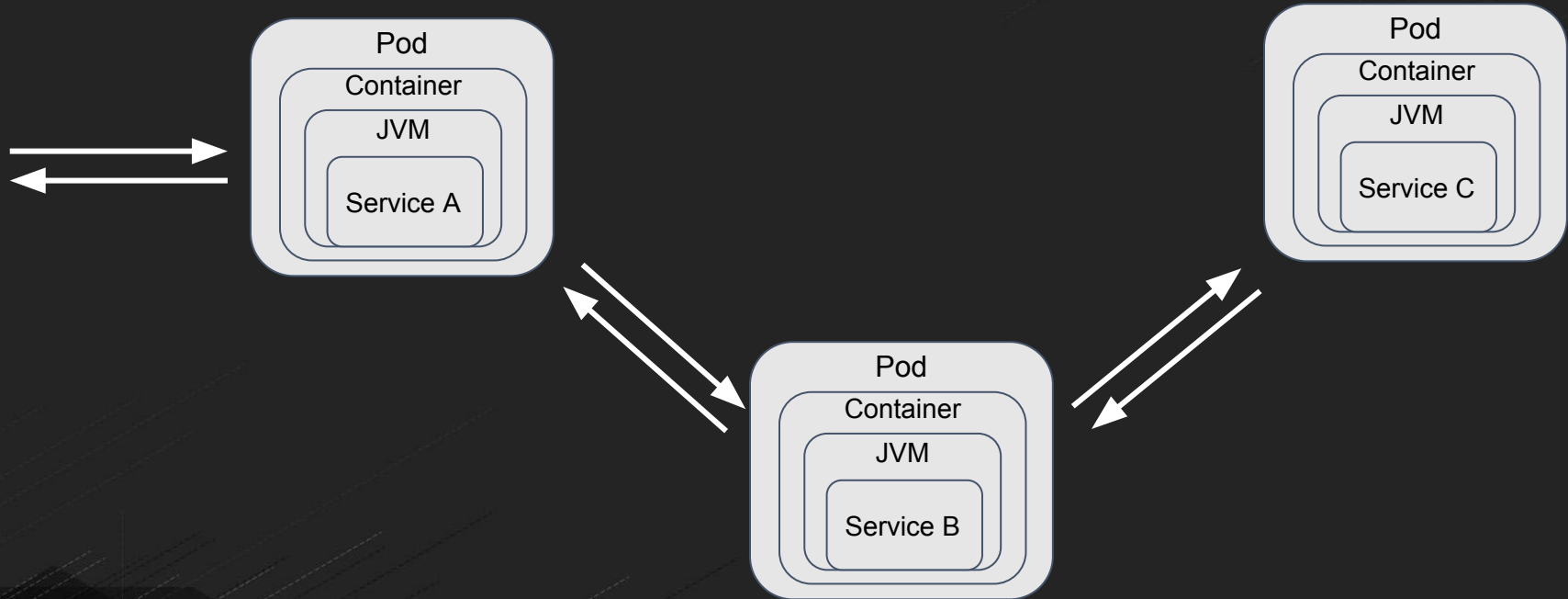
```
kubectl scale deployment myvertex --replicas=3
```

```
kubectl set image deployment/myvertex myvertex=burr/myvertex:v2
```

Kubernetes Tutorial

kubernetesbyexample.com

Microservices == Distributed Computing



java -jar myapp.jar

DropWizard

www.dropwizard.io

JAX-RS API

First to market

DropWizard Metrics

Embeddable servers:
Jetty



Spring Boot

projects.spring.io/spring-boot

Spring API
(@RestController)

'Starter' POMs:
start.spring.io

Embeddable servers:
Tomcat, Jetty, Undertow



WildFly Swarm

wildfly-swarm.io

MicroProfile.io

'Starter' POMs:
wildfly-swarm.io/generator

Embeddable servers:
WildFly (Undertow)



Vert.x

vertx.io

Reactive
Async/non-blocking

vertx run myhttp.java

HTTP, HTTP/2, TCP,
UDP, Websockets,
etc. (out of the box)





MICROPROFILE™

OPTIMIZING ENTERPRISE JAVA

- Defines **open source** Java **microservices** specifications
- Industry Collaboration - Red Hat, IBM, Payara, Tomitribe, London Java Community, SouJava, Oracle, Hazelcast, Fujitsu, SmartBear...
- **WildFly Swarm** is **Red Hat's** implementation
- Minimum footprint for Enterprise Java cloud-native services (v1.3) :

JSON-P 1.0

Health
Check

JWT
Propagation

Config

OpenAPI

REST
Client

CDI 1.2

JAX-RS 2.0

Fault
Tolerance

Metrics

Open
Tracing

Short History of Microservices



NETFLIX

OSS

NETFLIX

OSS

A NETFLIX ORIGINAL SERIES

HOUSE of CARDS



February 2013

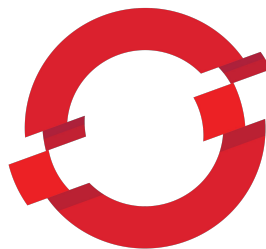
Java Microservices Platform circa 2015



NETFLIX Ribbon



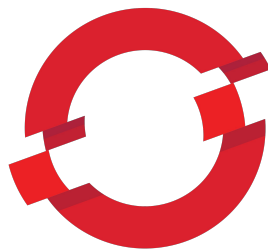
Better Microservices Platform circa 2016



OPENSIFT



Better Microservices Platform circa 2017



OPENSIFT



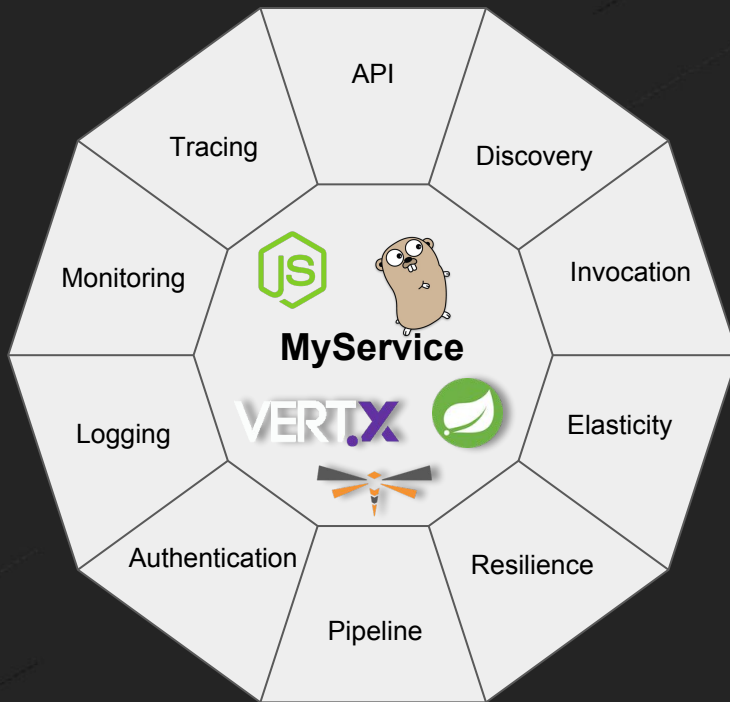
HYSTRIX
DEFEND YOUR APP

Better Microservices Platform circa 2018



Polyglot Microservices Platform circa 2018





OPENSIFT

Polyglot Microservices Platform



Kelsey Hightower ✓

@kelseyhightower

Following



2020 prediction: Monolithic applications will be back in style after people discover the drawbacks of distributed monolithic applications.

11:40 AM - 11 Dec 2017

1,318 Retweets 3,110 Likes



98 1.3K 3.1K



Tweet your reply



Burr Sutter @bursutter · 1h



Replying to @kelseyhightower

Folks are super excited about this thread - that does feel like we have entered the Trough of Disillusionment on Microservices

en.wikipedia.org/wiki/Hype_cycle

1 4

8 Steps

Step 1: Installation

Step 1 - Installation

Lots of Options

1. minikube (kubectl) (docs)
2. minishift (oc) (docs)
3. Google Cloud Platform
4. Openshift.com
5. AWS
6. oc cluster up (requires a docker daemon)
7. Red Hat DevSuite: Windows Installer + CDK
8. Many more...

<https://kubernetes.io/docs/setup/pick-right-solution/#turnkey-cloud-solutions>

Demo

Step 2: Building Images

Step 2: Building Images, Running Containers

1. `docker build + kubectl run or kubectl create -f deploy.yml`
2. Fabric8 maven plugin (fabric8.io)
3. Helm Charts - and Tiller
4. Kompose - converts docker-compose.yml to kubernetes yaml
5. Kedge - Concise Application Definitions for Kubernetes
6. Ansible - Playbooks for Kubernetes/Openshift deployment
7. s2i - source to image
8. podman - building Linux container images sans d-o-c-k-e-r

```
package stuff.demo;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController

public class MyRESTEndpoint {

    String hostname = System.getenv().getOrDefault("HOSTNAME", "unknown");

    @RequestMapping("/")
    public String index() {
        return "Hello Spring " + new java.util.Date() + " " + hostname;
    }
}
```

Fabric8 Maven Plugin

<https://maven.fabric8.io/#fabric8:setup>

oc new-project stuff (or kubectl create -f namespace.yml)

cd myspringbootapp/ or cd myvertxapp/

mvn clean compile package

mvn io.fabric8:fabric8-maven-plugin:3.5.37:setup

mvn fabric8:deploy

Demo

<https://github.com/burrsutter/kube4docker>

Step 3: oc or kubectl exec

Step 3: oc or kubectl exec

ssh into your containers and explore

```
kubectl get pods --namespace=microworld
```

```
kubectl exec -it --namespace=microworld $POD cat /sys/fs/cgroup/memory/memory.limit_in_bytes
```

or

```
kubectl exec -it --namespace=microworld microspringboot1-2-nz8f8 /bin/bash
```

```
ps -ef | grep java
```

Note: the following apply if using the fabric8 generated image, otherwise consult your Dockerfile

```
java -version
```

```
javac -version
```

```
# now find that fat jar
```

```
find / -name *.jar
```

```
cd /deployments (based on use of the fabric8 maven plugin)
```

```
ls
```

```
exit
```

Demo

Step 4: logs

Step 4: logs

```
System.out.println("Where am I?");
```

Spring Boot logs by default just go to the console.

```
kubectl get pods
```

```
kubectl logs microspringboot1-2-nz8f8
```

OR `./kubetail.sh`

<https://raw.githubusercontent.com/johanhaleby/kubetail/master/kubetail>

OR `stern` (brew install stern)

<https://github.com/wercker/stern>

Demo

Step 5: service discovery & load-balancing

Step 5: service discovery

1. Services are internal to the cluster and can be mapped to pods via a label selector
2. Just refer to a Service by its name, it is just DNS

```
String url = "http://producer:8080/";  
ResponseEntity<String> response =  
restTemplate.getForEntity(url, String.class);
```

Samples

<https://github.com/burrsutter/kube4docker/tree/master/discovery>

<https://github.com/redhat-developer-demos/microspringboot1>

<https://github.com/redhat-developer-demos/microspringboot2>

<https://github.com/redhat-developer-demos/microspringboot3>

Step 6: Live and Ready

Step 5: Live and Ready

```
spec:
  containers:
    - name: myvertx
      image: burr/myvertx:v1
      livenessProbe:
        httpGet:
          port: http
          path: /hello
        initialDelaySeconds: 10
        periodSeconds: 5
        timeoutSeconds: 3
      readinessProbe:
        httpGet:
          path: /hello
          port: 8080
        initialDelaySeconds: 10
        periodSeconds: 3
```

Demo

<http://movies.app.burr.red/>

<https://github.com/redhat-developer-demos/popular-movie-store/blob/master/src/main/java/org/workspace7/moviestore/controller/HomeController.java#L158-L159>

<https://github.com/redhat-developer-demos/popular-movie-store/blob/master/src/main/fabric8/deployment.yml#L38-L42>

Step 7: Rolling Updates, Blue/Green Canary

Step 7: Rolling, Blue/Green, Canary

bit.ly/msa-instructions

OpenShift Web Console x Microservices Frontend x Helloworld-MSA (Microservice x Burr

frontend-helloworld-msa.app.burr.red

Red Hat - Hello world MSA (Microservices Architecture)

SSO Browser as a client API Gateway Service chaining Hystrix Dashboard Jaeger Dashboard

Using an API Gateway

Refresh Results

API Gateway

- Blue - Aloha mai aloha-3-3kt96
- Blue - Hola de hola-2-xt8w
- Blue - Olá de ola-3-1vwzr
- Blue - Bonjour2 de bonjour-3-4gv0f

Refresh Results

```
graph LR; Browser[Browser] <--> Internet((Internet)); Internet <--> APIGateway[API Gateway]; APIGateway --> Hola[Hola (JAX-RS)]; APIGateway --> Bonjour[Bonjour]; APIGateway --> Aloha[Aloha]; APIGateway --> Ola[Ola];
```

Demo

Step 8: Debugging

Step 8: Debugging

<https://maven.fabric8.io/#fabric8:debug>

`mvn fabric8:deploy`

`mvn fabric8:debug`

<https://code.visualstudio.com/blogs/2017/09/28/java-debug>

<https://github.com/VeerMuchandi/openshift-local/blob/master/DebuggingUsingIDE.md>

Demo

Bonus: Istio



Istio - Sail

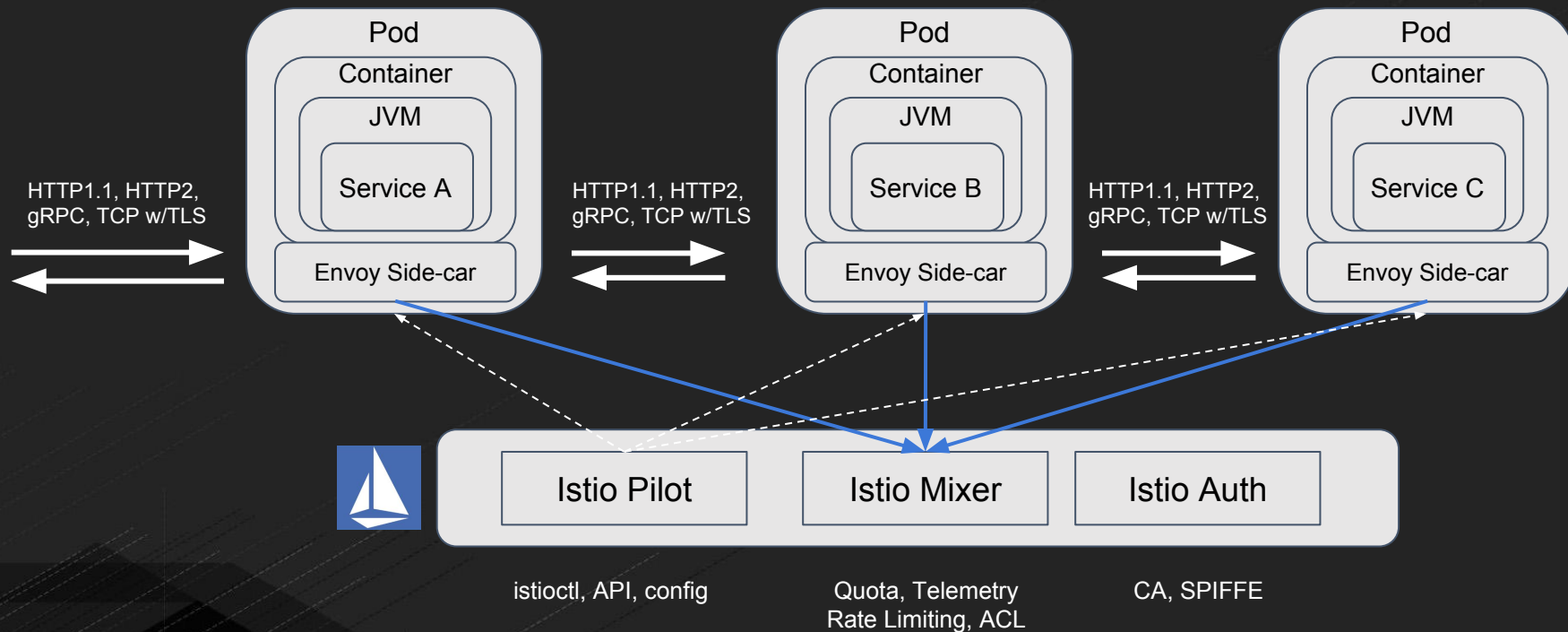
(Kubernetes - Helmsman or ship's pilot)

Next Gen Microservices - Service Mesh

Code Independent

- Intelligent Routing and Load-Balancing
 - A/B Tests
 - Canary Releases
 - Dark Launches
- Distributed Tracing
- Circuit Breakers
- Fine grained Access Control
- Telemetry, metrics and Logs
- Fleet wide policy enforcement

Istio Control Plane



The Comedy of Errors

Wikipedia Summary: The Comedy of Errors is one of **William Shakespeare's** early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

Book Details

Paperback:

200 pages

Publisher:

PublisherA

Language:

English

ISBN-10:

1234567890

ISBN-13:

123-1234567980

An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!

— Reviewer1 *Affiliation1*



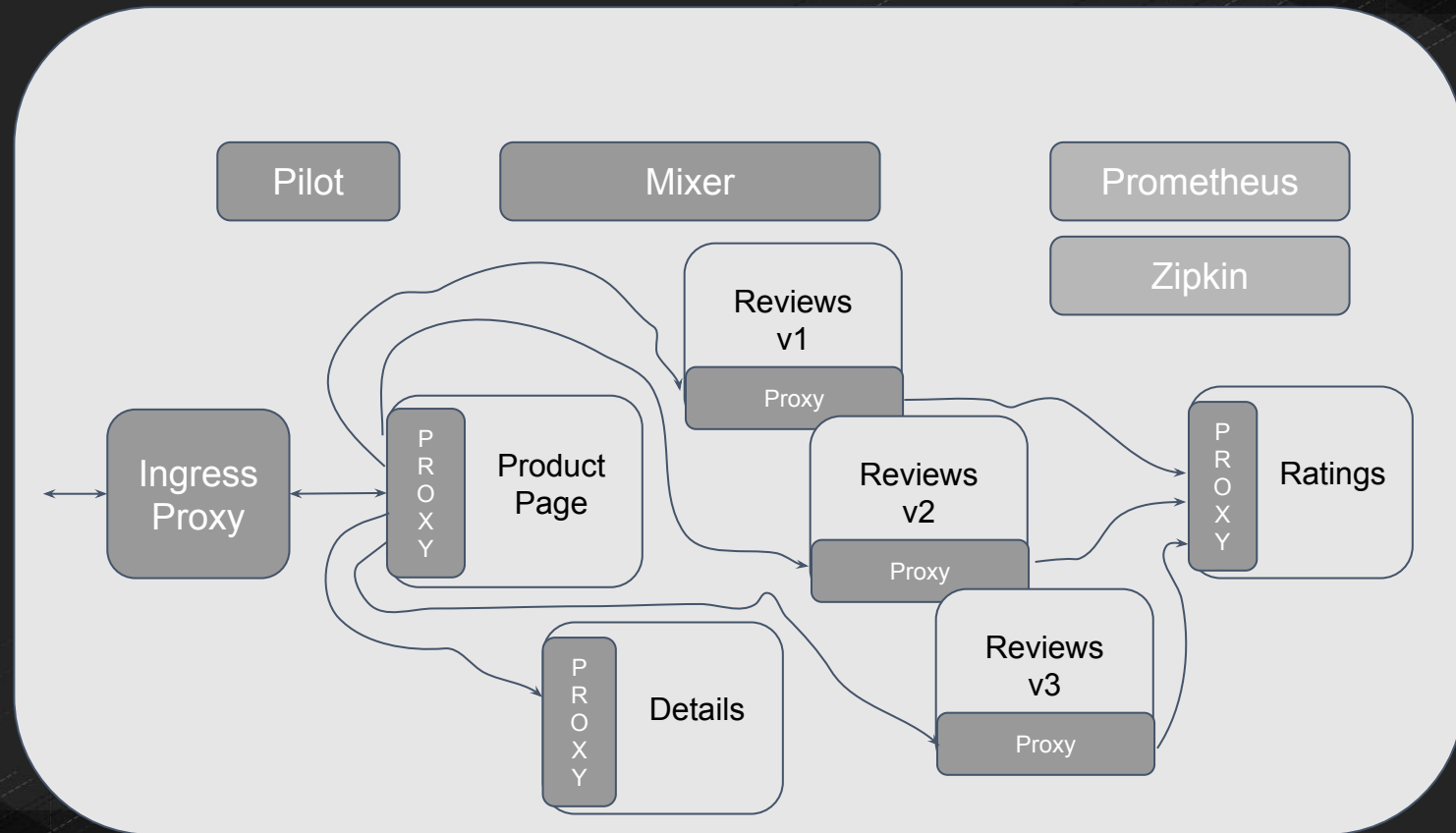
Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare.

— Reviewer2 *Affiliation2*



<https://istio.io/docs/samples/bookinfo.html>





https://www.youtube.com/watch?v=1SjJhwLrdbw&t=1s&index=11&list=PLuWlr4oKSRUYjjwDOaZOX-54X4_OIUUeS



bit.ly/istio-intro

bit.ly/istio-tutorial

Bonus: Eclipse Che

Container Native IDE - Eclipse Che

Allows you to launch a browser-based IDE inside of a Linux container that matches your production environment

<http://www.eclipse.org/che/docs/setup/openshift/index.html>

Create - OpenShift.ioOpenShift Eclipse Che | myvert.xVert.x HTTP Booster

Secure | https://che-burrrzinga-che.8a09.starter-us-east-2.openshiftapps.com/che/myvertxhello5-atanh

WorkspaceProjectEditAssistantRunGitProfileHelpEXECdev-machine: debug#1 00:56

Projects Explorer

Projects

Commands

myvertxhello5 (master)

src

licenses

main

fabric8

java

io.openshift.booster

HttpApplication.java

resources

test

target

Jenkinsfile

LICENSE

README.adoc

HttpApplication.java

2930313233343536373839404142434445464740:1

// Retrieve the port from the configuration, default to 8080.
config().getInteger("http.port", 8080), ar -> {
 if (ar.succeeded()) {
 System.out.println("Server starter on port " + ar.result().actualPort());
 }
 future.handle(ar.mapEmpty());
 }
}

private void greeting(RoutingContext rc) {
 String name = rc.request().getParam("name");
 if (name == null) {
 name = "World";
 }

 JsonObject response = new JsonObject()
 .put("content", String.format(template, name));
}

UTF-8Java

Debug

OpenJDK 64-Bit Server VM 1.8.0_144

Breakpoints:

HttpApplication.java:40

Frames:

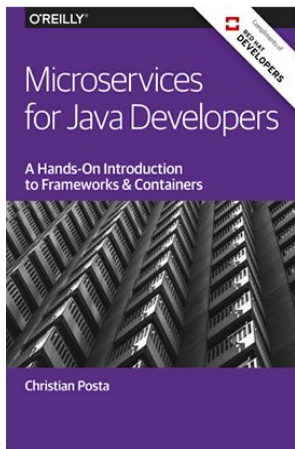
"vert.x-eventloop-thread-0"@3 in group "main": RUNNING
greeting(RoutingContext):40, HttpApplication
handle():-1, HttpApplication
handleContext(RoutingContext):217, io.vertx.ext.web.impl.RoutelImpl
iterateNext():78, io.vertx.ext.web.impl.RoutingContextImplBase
next():118, io.vertx.ext.web.impl.RoutingContextImpl

Variables:

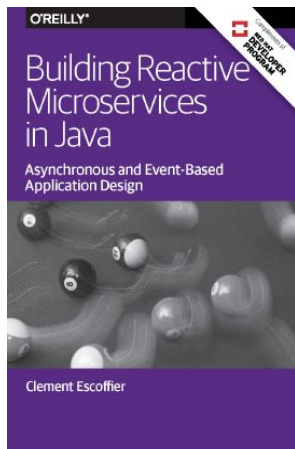
{HttpApplication.java:40}
> template="Hello, %s!"
> context=instance of io.vertx.core.
> vertx=instance of io.vertx.core.
> rc=instance of io.vertx.ext.web.i

EventsProcessesDebug

Resources



bit.ly/javamicroservicesbook

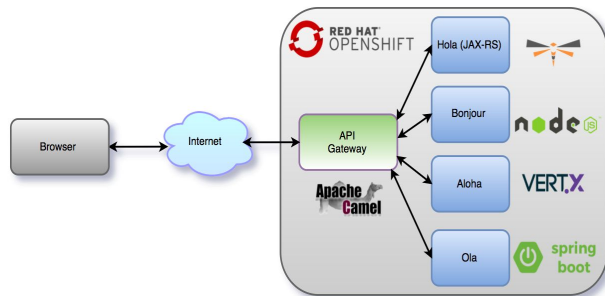


VERT.X

bit.ly/reactivemicroservicesbook



bit.ly/microservicesvideo

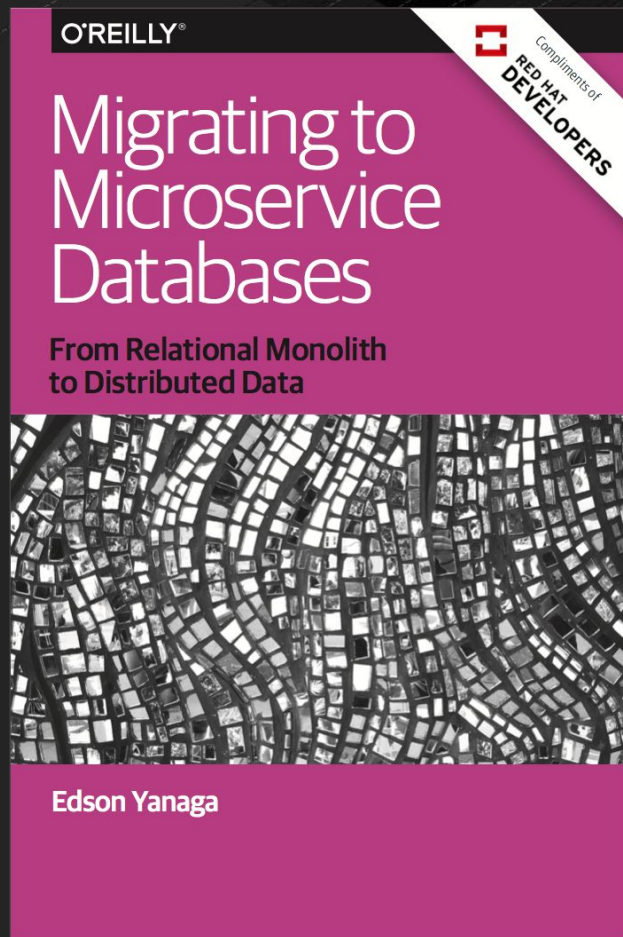


bit.ly/msa-instructions

There's an Elephant in the Room

Your Monolithic Database
(Your DBA will kill you.)

Download a copy
of Edson's book at
<http://bit.ly/mono2microdb>





RED HAT DEVELOPERS

8 Steps to Becoming Awesome
with Kubernetes

Link

@burrsutter

burr@redhat.com

<https://developers.redhat.com/devnationlive/>

<http://bit.ly/8stepsawesome>