Build your applications thinking on zero-downtime upgrades

OpenSouthCode 2017
Malaga, Spain
May 2017

Javier Ramirez
Software Maintenance Engineer
Red Hat

http://www.sombrerorojo.es
@javilinux
github.com/javilinux

Basada en una charla de mi compañero Jorge Morales

http://jorgemoral.es
@UnPOUcoDe
github.com/jorgemoralespou

THE WORLD'S LEADING PROVIDER OF OPEN SOURCE, ENTERPRISE I.T. SOLUTIONS

MORE THAN
90%
of the
FORTUNE
500
use
RED HAT
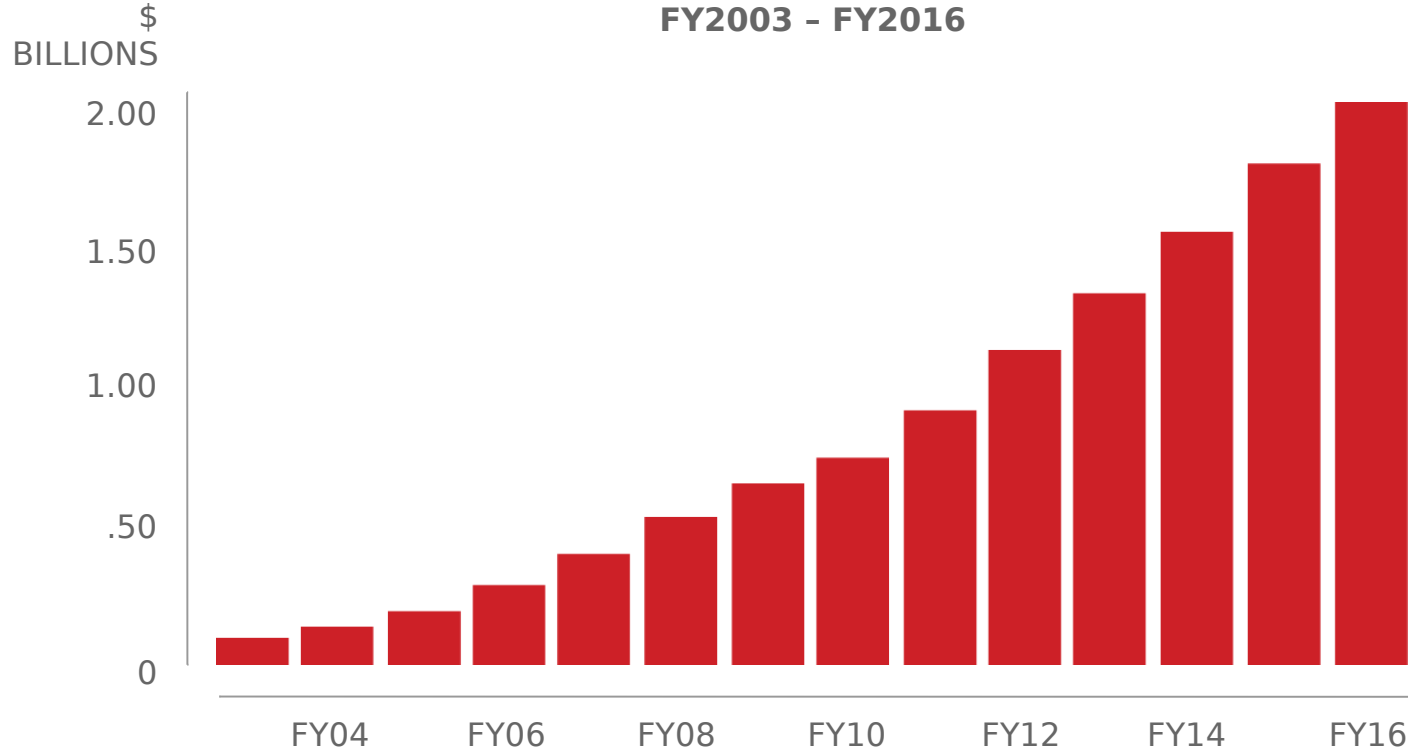PRODUCTS & SOLUTIONS*

~10,000
EMPLOYEES

85 35
OFFICES COUNTRIES

S&P
500
COMPANY

NYSE
RHT

THE FIRST
$2 BILLION
OPEN SOURCE COMPANY
IN THE WORLD

COMPANY REVENUE

FY2003 – FY2016

Build your applications thinking in zero-downtime upgrades

@UnPOUcoDe

OpenShift

Community Powered Innovation

RED HAT® OPENSHIFT

Introduction

Evolution of IT

Development Process	Application Architecture	Deployment & Packaging	Application Infrastructure
Waterfall	Monolithic	Physical Servers	Datacenter
Agile	N-Tier	Virtual Servers	Hosted
DevOps	**Microservices**	**Containers**	**Cloud**
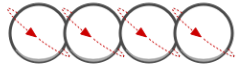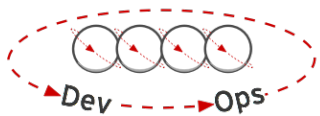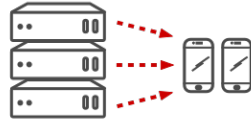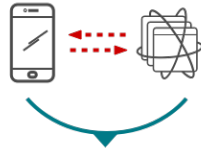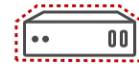
Build your applications thinking in zero-downtime upgrades

@javilinux
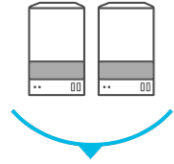
WHAT ARE CONTAINERS?

It depends who you ask

INFRASTRUCTURE

APPLICATIONS

- Sandboxed application processes on a shared Linux OS kernel

- Simpler, lighter, and denser than virtual machines

- Portable across different environments

- Package my application and all of its dependencies

- Deploy to any environment in seconds and enable CI/CD

- Easily access and share containerized components

Build your applications thinking in zero-downtime upgrades

@javilinux

CLEAR BOUNDARIES

Container

App

Controlled by
Developers

Operating
System

Virtual Machine

Controlled by
IT Operations

Hardware

COMMON LANGUAGE

DEVELOPERS

I.T. OPERATIONS
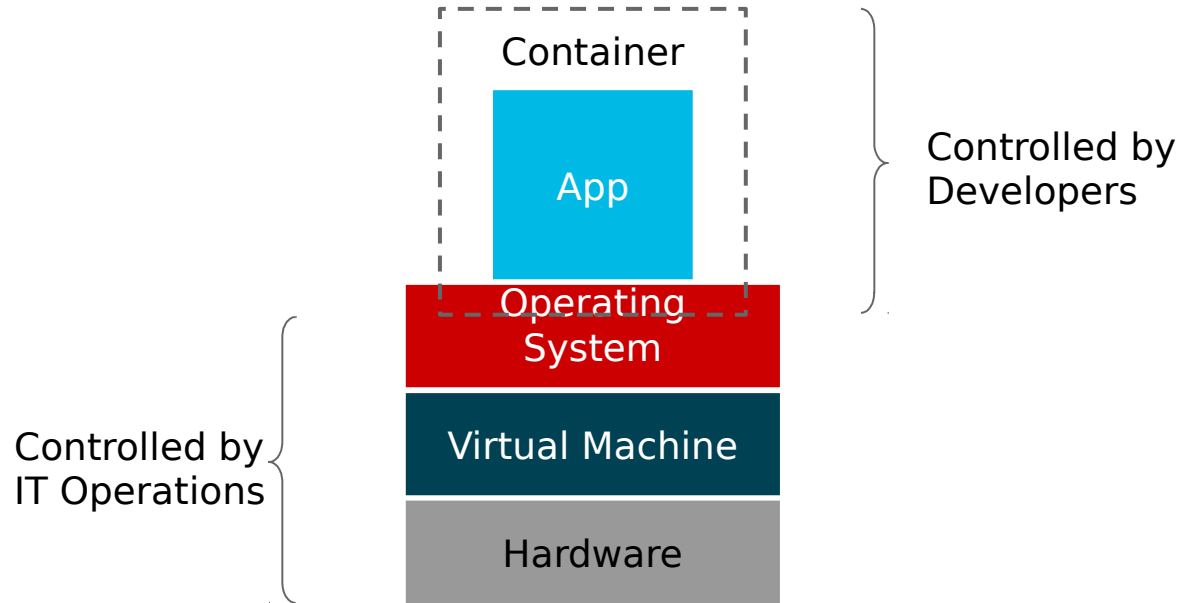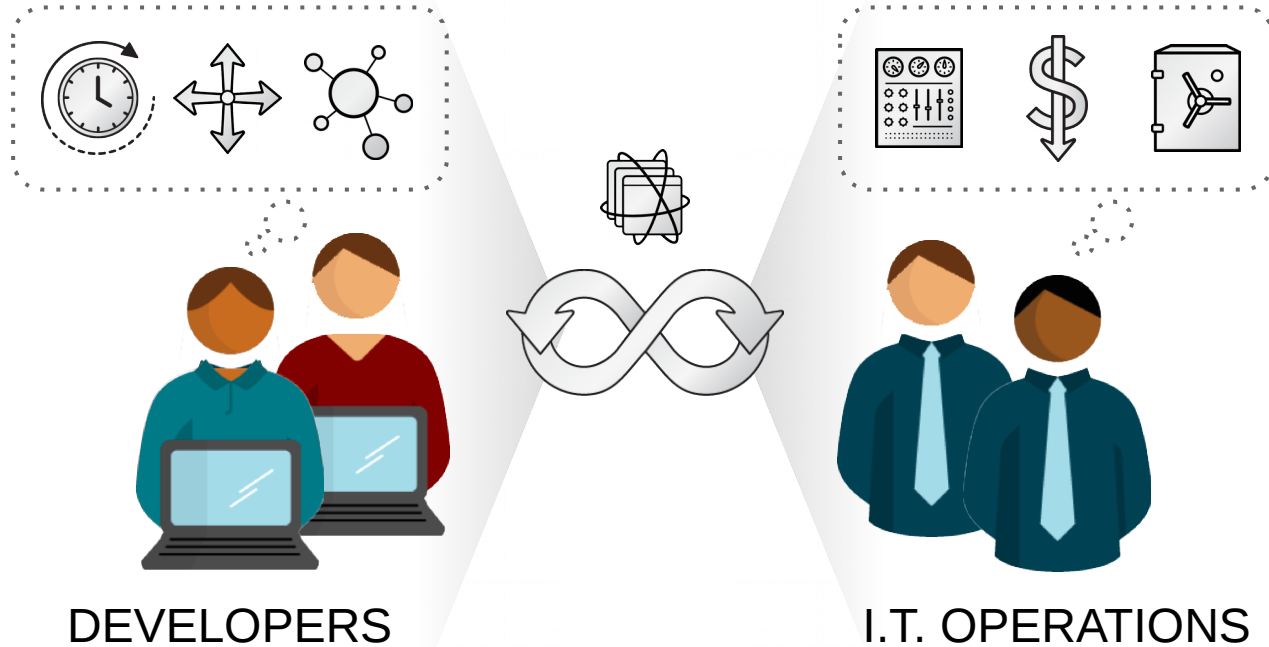
Docker is an open-source container packaging format and runtime that packages and run applications in "containers," allowing them to be portable among systems running Linux.
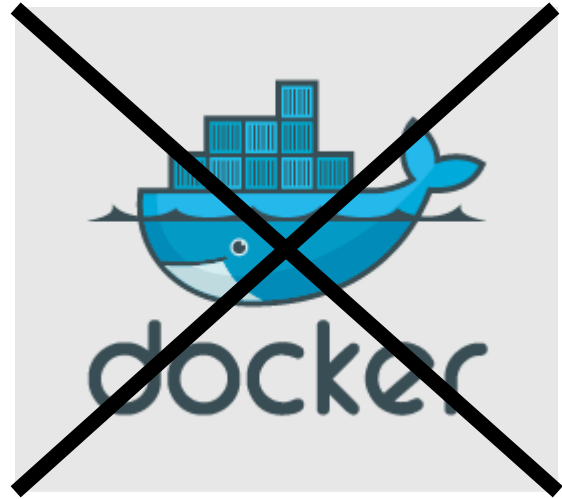
Kubernetes is an open-source system for automating deployment, operations, and scaling of containerized applications across multiple hosts

kubernetes

But you need more

Multi-tenancy	Teams and Collaboration
Routing & Load Balancing	Quota Management
CI/CD Pipelines	Image Build Automation
Role-based Authorization	Container Isolation
Capacity Management	Vulnerability Scanning
Infrastructure Visibility	Chargeback

The industry's most
secure
and comprehensive
enterprise-grade
container platform based
on industry standards,
Docker and Kubernetes.

To take full advantage of the platform one needs to know how it works

Workload types

ReplicaSets →DeploymentConfig/Deployments

StatefulSets (a.k.a PetSets)

DaemonSets

Jobs

CronJobs

more to come???

DeploymentConfig

A deployment configuration consists of the following key parts:

A **replication controller template** which describes the application to be deployed.

The default **replica count** for the deployment.

A **deployment strategy** which will be used to execute the deployment.

A set of **triggers** which cause deployments to be created automatically.

Deployment strategies

Recreate

Rolling

Custom

Recreate Deployment Strategy

Recreate Strategy for Deployment

A recreate deployment removes instances of the previous version of an application and replaces them with instances of the new version of the application.

The Recreate strategy has basic rollout behavior and supports lifecycle hooks for injecting code into the deployment process

When to use a Recreate Strategy

When you must run migrations or other data transformations before your new code starts.

When you do not support having new and old versions of your application code running at the same time.

When you want to use a RWO volume, which is not supported being shared between multiple replicas.

Definition of a Recreate Strategy

```
strategy:
  type: Recreate
  recreateParams:
    timeoutSeconds:
    pre: {}
    mid: {}
    post: {}
```

Workflow of a Recreate Strategy

1. Execute any **pre** lifecycle hook.

2. **Scale down** the previous deployment to zero.

3. Execute any **mid** lifecycle hook.

4. **Scale up** the new deployment.

5. Execute any **post** lifecycle hook.

Demo - Recreate Strategy (Code)

Execute a recreate deployment of an application with 3 instances

Demo en youtube

```
oc new-project recreatedemo

oc create -f
https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-
talk/master/demos/recreatedemo.yaml

oc tag deployment-example:v2 deployment-example:latest &&
oc logs -f dc/deployment-example
```

Allow for Zero downtime?

Rolling Deployment Strategy

Rolling Strategy for Deployment

A rolling deployment slowly replaces instances of the previous version of an application with instances of the new version of the application. A rolling deployment typically waits for new pods to become ready via a readiness check before scaling down the old components. If a significant issue occurs, the rolling deployment can be aborted.

All rolling deployments in OpenShift Origin are canary deployments; a new version (the canary) is tested before all of the old instances are replaced.

When to use Rolling Strategy

When you want to take no downtime during an application update.

When your application supports having old code and new code running at the same time.

Definition of a Rolling Strategy

```
strategy:
  type: Rolling
  rollingParams:
    timeoutSeconds:
    intervalSeconds:
    updatePeriodSeconds:
    maxSurge: ""
    maxUnavailable: ""
    pre: {}
    post: {}
```

Workflow of a Rolling Strategy

1. Execute any **pre** lifecycle hook.

2. **Scale up** the **new** replication controller based on the surge count.

3. **Scale down** the **old** replication controller based on the max unavailable count.

4. **Repeat this scaling** until the new replication controller has reached the desired replica count and the old replication controller has been scaled to zero.

5. Execute any **post** lifecycle hook.

Demo - Rolling Strategy ([Code](#))

Execute a Rolling deployment of an application with 3 instances
Demo en youtube

```
oc new-project rollingdemo

oc create -f
https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-
talk/master/demos/rollingdemo.yaml

oc tag deployment-example:v2 deployment-example:latest &&
oc logs -f dc/deployment-example
```

Allow for Zero downtime?

Custom Deployment Strategy

Custom Strategy for Deployment

The Custom strategy allows you to provide your own deployment behavior.

When to use Custom Strategy

When the default strategies does not work for you

When you want to alter how a default strategy works

Definition of a Custom Strategy

```
strategy:
  type: Custom
  customParams:
    Image: ""
    command:
    environment:
```

Alternative - Custom behaviour of default strategies

```
strategy:
  type: Rolling
  customParams:
    Image: ""
    command: "/bin/sh -c openshift-deploy --until=50%; echo
Halfway there; openshift-deploy; echo Complete"
    environment:
```

Demo - Custom strategy ([Code](#))

Execute a Custom Rolling deployment of an application with 4 instances showing a message half way through
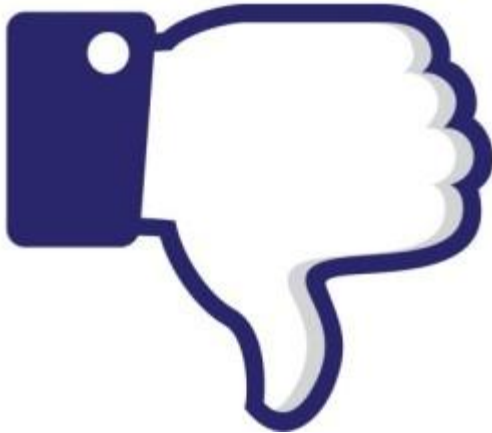Demo en youtube

oc new-project customdemo

oc create -f
https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-talk/master/demos/customdemo.yaml

oc tag deployment-example:v2 deployment-example:latest && oc logs -f dc/deployment-example

Allow for Zero downtime?

What's the problem now?

When a container starts it's not when the application is ready for getting requests.

Health checks

Know when the application is ready to accept requests (**readinessProbe**).

Know if the application is healthy (**livenessProbe**)

HTTP Check

The kubelet uses a web hook to determine the healthiness of the container. The check is deemed successful if the hook returns with 200 or 399

A HTTP check is ideal for complex applications that can return with a 200 status when completely initialized.

```
httpGet:
  path: /healthz
  port: 8080
initialDelaySeconds: 15
periodSeconds: 1
timeoutSeconds: 1
failureThreashold: 3
sucessThreshold: 1
```

Container execution Check

The kubelet executes a command inside the container. Exiting the check with status 0 is considered a success.

When health should be checked using a script or command

```
exec:
    command:
    - cat
    - /tmp/health
    initialDelaySeconds: 15
periodSeconds: 1
timeoutSeconds: 1
failureThreshold: 3
sucessThreshold: 1
```

TCP socket Check

The kubelet attempts to open a socket to the container. The container is only considered healthy if the check can establish a connection.

A TCP socket check is ideal for applications that do not start listening until initialization is complete.

```
tcpSocket:
  port: 8080
initialDelaySeconds: 15
periodSeconds: 1
timeoutSeconds: 1
failureThreashold: 3
sucessThreshold: 1
```

Readiness Probe

A readiness probe determines if a container is ready to service requests. If the readiness probe fails a container, the endpoints controller ensures the container has its IP address removed from the endpoints of all services.
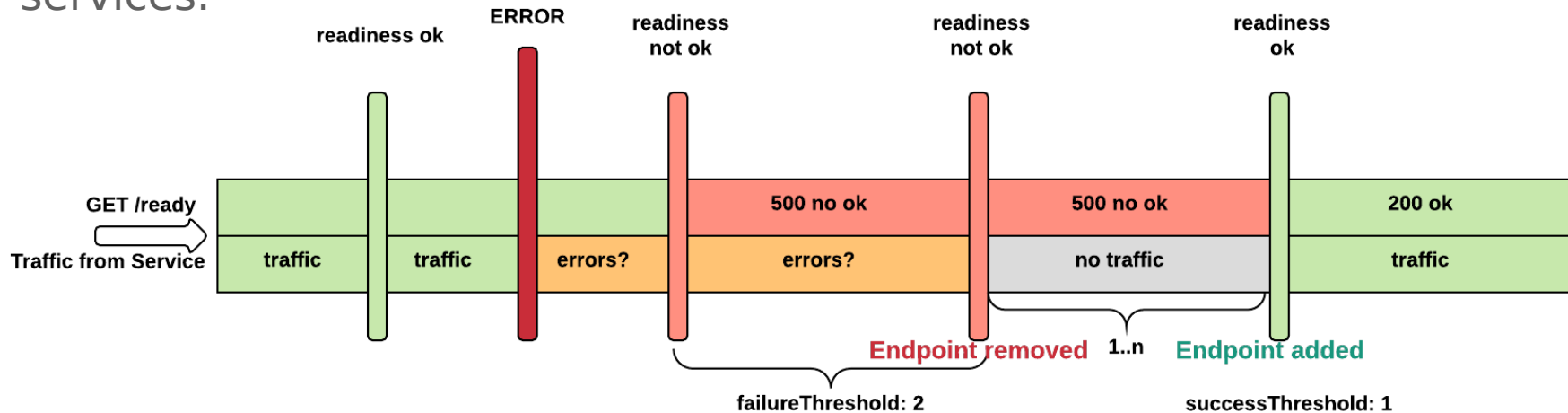
Demo - Readiness Probe (Code)

Switch readiness result OK/NOOK
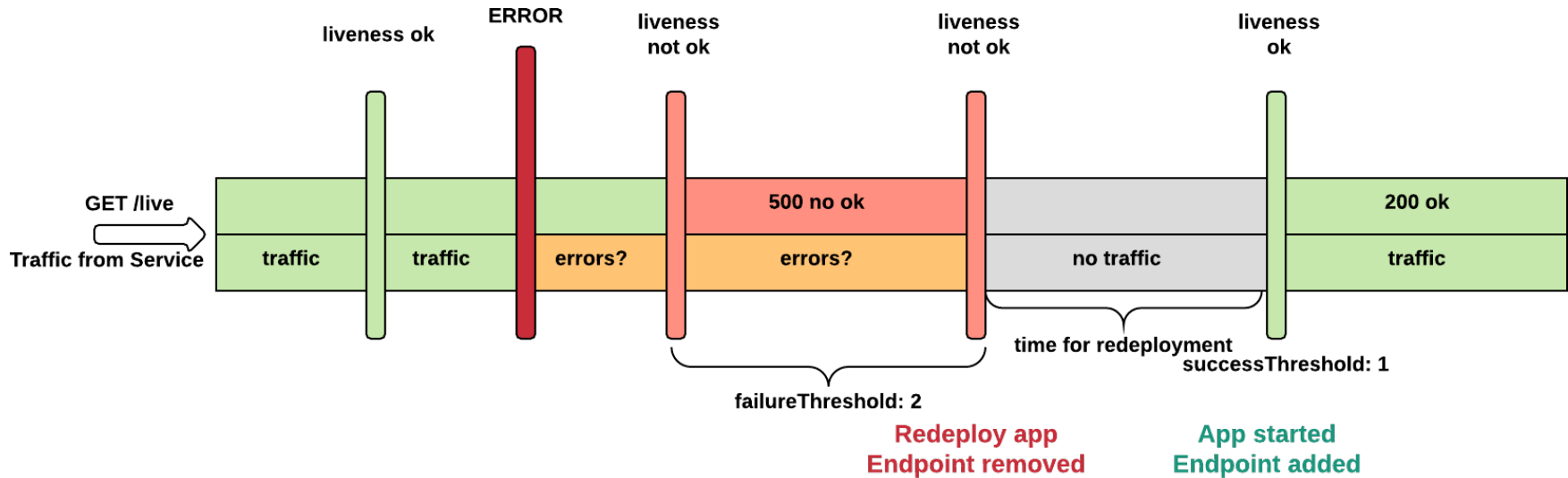Demo en youtube

oc new-project probesdemo

oc create -f
https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-
talk/master/demos/probesdemo.yaml

curl http://probesdemo-probesdemo.apps.lcup/ws/unsetready
curl http://probesdemo-probesdemo.apps.lcup/ws/setready

Liveness Probe

A liveness probe checks if the container in which it is configured is still running. If the liveness probe fails, the kubelet kills the container, which will be subjected to its restart policy.

Demo - Liveness Probe ([Code](#))

Switch Liveness resultNOOK
Demo en youtube

```
oc new-project probesdemo

oc create -f
https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-
talk/master/demos/probesdemo.yaml


curl http://probesdemo-probesdemo.apps.lcup/ws/unsetlive
```

Don't kill your current clients (a.k.a Graceful Shutdown)

On shutdown, OpenShift Origin will send a TERM signal to the processes in the container.

On receiving SIGTERM, an application:

> should continue to signal liveness

> may stop signalling readiness (in fact, when a Pod deletion starts, Kubernetes automatically removes the pod from services irrespective of any readiness probe result)

> should continue to accept new connections as long as they arrive

> should terminate cleanly if/when possible, e.g. when all connections have completed

After the graceful termination period expires, a process that has not exited will be sent the KILL signal, which immediately ends the process. (terminationGracePeriodSeconds defaults to 30 seconds).

Make sure signals get to the process. Use **exec**
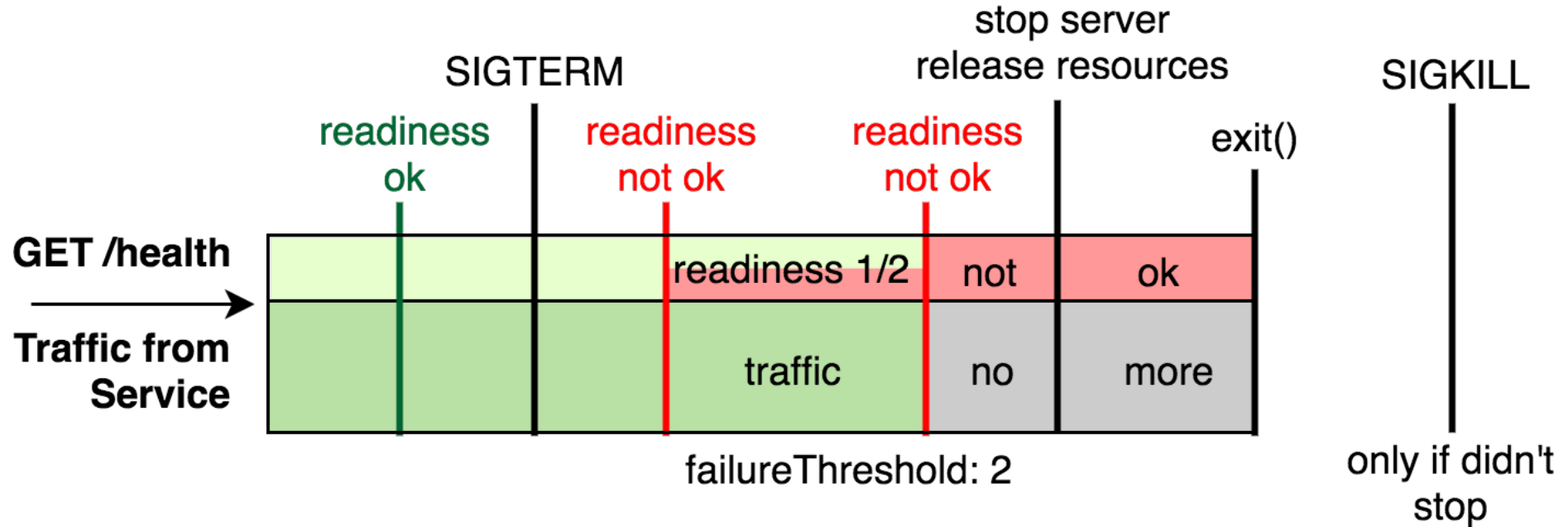
Make sure signals get to the process. Use **exec**

Demo - Propagate signal

Enter the container and kill 1
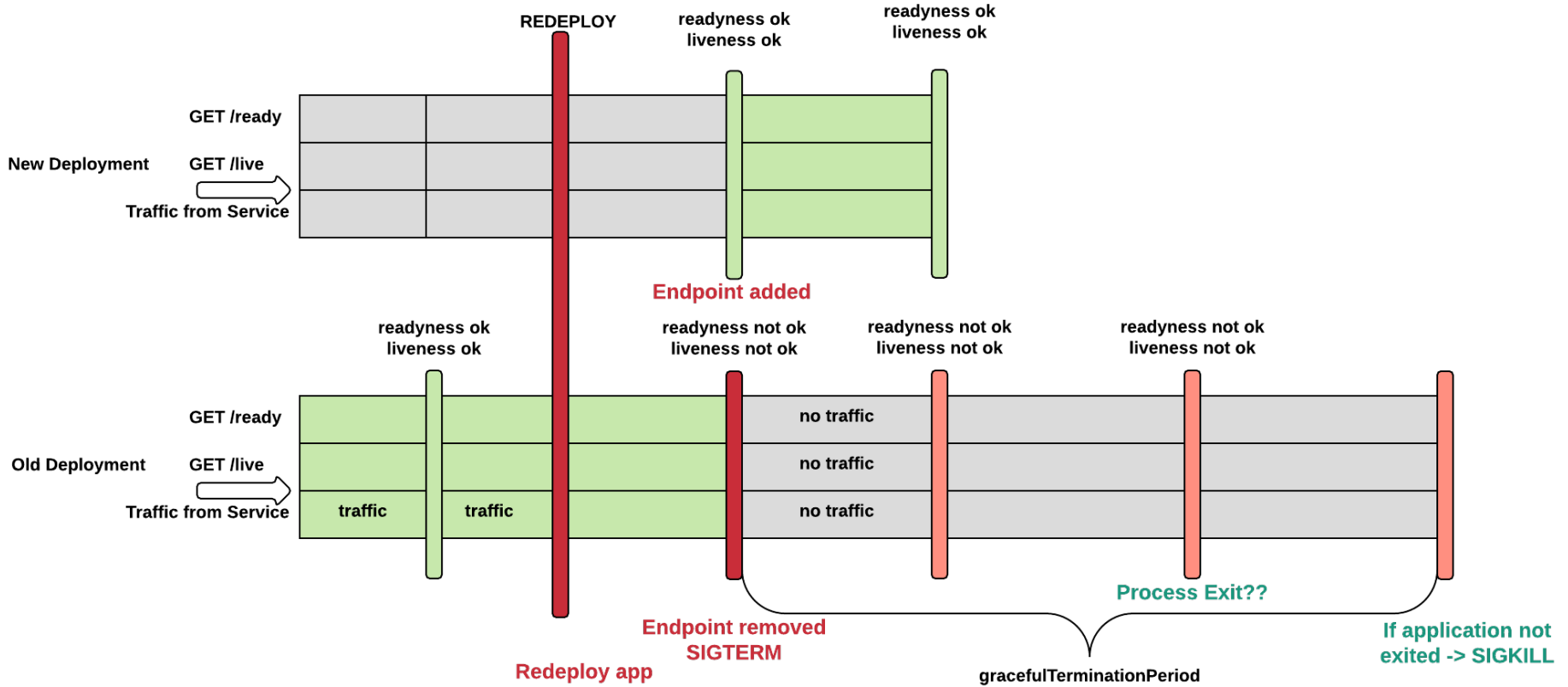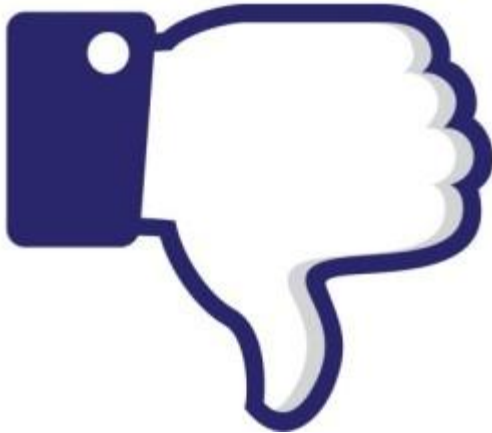
```
oc new-project probesdemo

oc create -f
https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-
talk/master/demos/probesdemo.yaml

oc rsh <pod>
kill 1
```
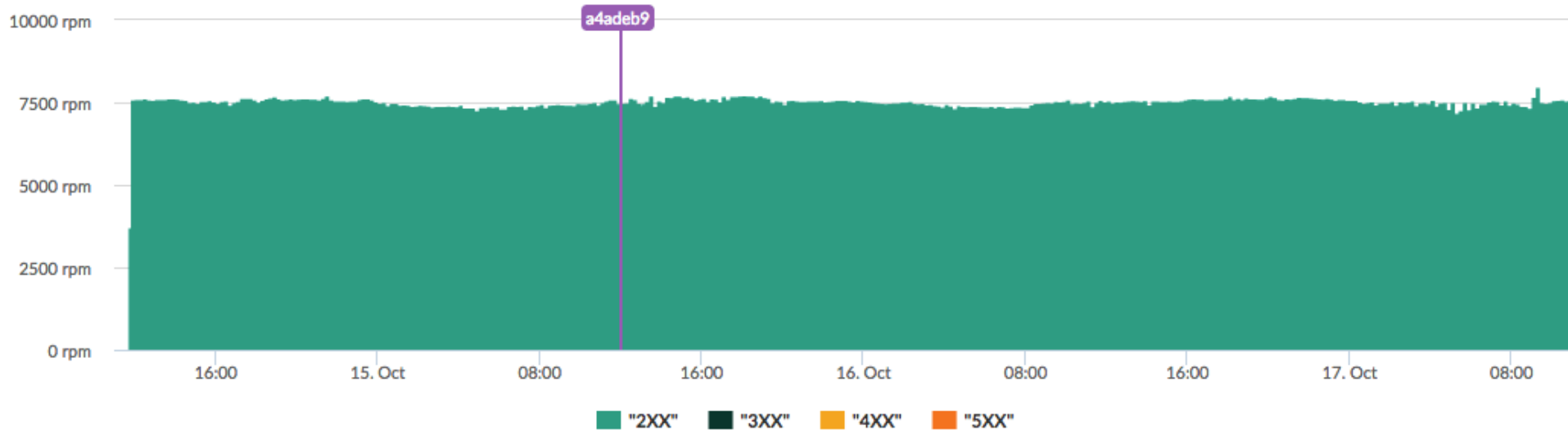
Allow for Zero downtime?

But deployments make it even better

RED HAT
OPENSHIFT

REDEPLOY

readyness ok
liveness ok

readyness ok
liveness ok

New Deployment

GET /ready

GET /live

Traffic from Service

Endpoint added

readyness ok
liveness ok

readyness not ok
liveness not ok

readyness not ok
liveness not ok

readyness not ok
liveness not ok

Old Deployment

GET /ready

GET /live

Traffic from Service

traffic

traffic

no traffic

no traffic

no traffic

Process Exit??

Endpoint removed
SIGTERM

Redeploy app

gracefulTerminationPeriod

If application not
exited -> SIGKILL

Build your applications thinking in zero-downtime upgrades

@UnPOUcoDe

Result graph

Demo - Zerodowntime (Code)

Deploy a new version of the application
Demo en youtube

```
oc new-project zerodowntimedemo
oc create -f https://raw.githubusercontent.com/jorgemoralespou/zerodowntime-
talk/master/demos/probesdemo.yaml

# Start load tool

httpress -n 10000 http://probesdemo-zerodowntime.apps.127.0.0.1.nip.io

# Redeploy lastest application again
oc deploy probesdemo --latest --follow
```
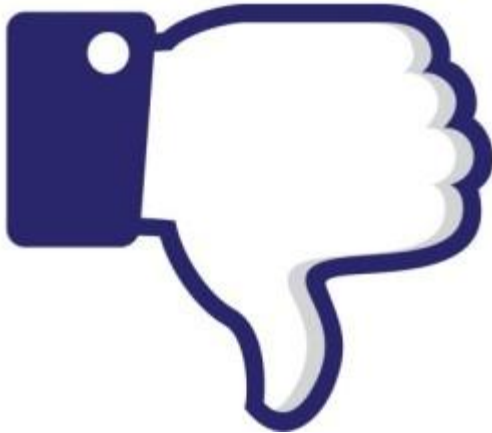
Allow for Zero downtime?

But what if?

My application upgrade needs change the database?

- If the change can be backwards compatible

  – Predeployment hook to update the database

  – Blue/Green Deployment - A/B Deployments

  – Pipelines orchestrated deployment (a.k.a Continuous Delivery Deployments)

- If the change can not be backwards compatible

  – No Zero-Downtime possible (with CAVEATS, but that's for another meetup)

Allow for Zero downtime?

Remember

Checklist

Appropriate Deployment strategy

- Recreate

- Rolling - > If possible

- Custom

Checklist

Healtchecks

- Liveness

- Readiness

  – HTTPGet

  – Command

  – TCPSocket

Checklist

- Propagate signals to your process

- Application/server Graceful Shutdown

Checklist

Data Involved

- Backwards compatibility

- Use lifecycle hooks

Checklist

Test under load

- Rollover

- Rollback

Checklist

Automate if there is a configuration change involved

- Pipelines to orchestrate the update

    - Rollover

    - Test

    - Rollback if error

Resources

https://github.com/jorgemoralespou/zerodowntime-talk

http://www.openshift.com

http://www.openshift.org

http://www.openshift.io

http://www.github.com/openshift

https://github.com/openshift-evangelists/oc-cluster-wrapper

Questions???