# Green Cloud Demo

Spring Cloud Brewery on OpenShift: Setup Guide

# Green Cloud Demo

1. Overview . . . . . 2
2. Demo Overview . . . . . 3
    2.1. Short History of Microservices . . . . . 3
    2.2. App Overview . . . . . 3
    2.3. Pre-Requisite . . . . . 3
    2.4. Docker Setup . . . . . 4
    2.5. Accessing the Applications . . . . . 4
3. Iteration I . . . . . 6
    3.1. Setup . . . . . 6
    3.2. Pre-Requisite . . . . . 7
    3.3. Deploy Applications . . . . . 7
4. Iteration II . . . . . 11
    4.1. Setup . . . . . 11
    4.2. Pre-Requisite . . . . . 11
    4.3. Deploy Applications . . . . . 12
    4.4. Acceptance Testing . . . . . 15
    4.5. Deprecated Modules . . . . . 15
5. Iteration III . . . . . 16
    5.1. Setup . . . . . 16
    5.2. Pre-Requisite . . . . . 17
    5.3. Deploy Applications . . . . . 17
    5.4. Open Issues . . . . . 20
    5.5. Deprecated Modules . . . . . 20
6. Resources . . . . . 21
    6.1. Documentation . . . . . 21
    6.2. Tools . . . . . 21
    6.3. Blogs . . . . . 22
    6.4. Demos . . . . . 22
    6.5. Videos . . . . . 23

# Chapter 1. Overview

**Green Cloud Demo** is your first step on how to migrate and optimize an existing Spring Boot application  to OpenShift.  The migration guides you with the process of how to migrate Spring Boot workload form other platform to OpenShift, i.e. the build process, the ideal platform (OpenShift), optimizing etc.,

# Chapter 2. Demo Overview

## 2.1. Short History of Microservices

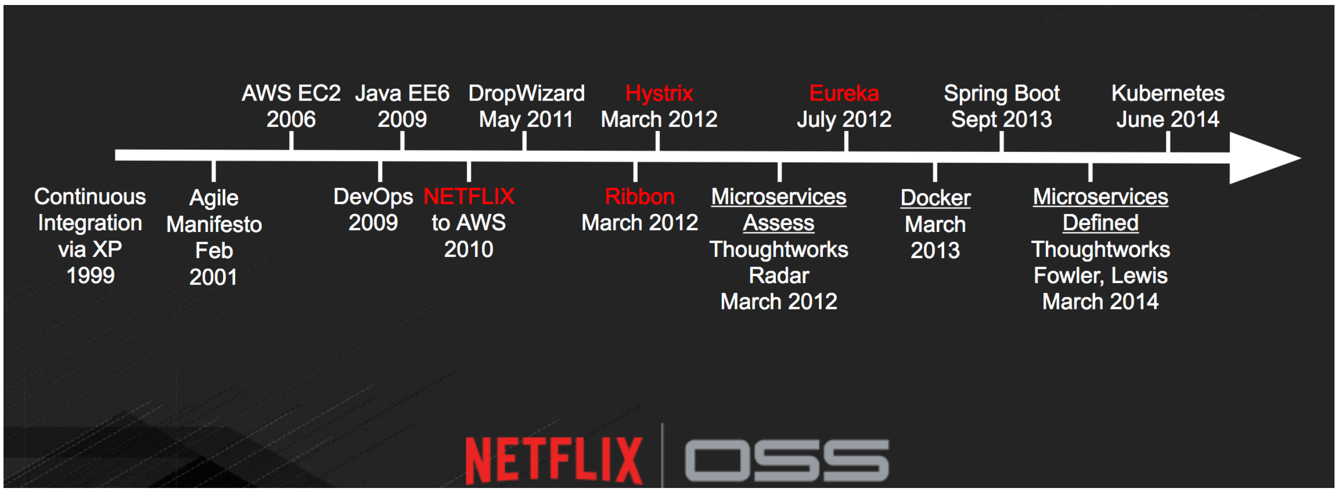


*Figure 1. Short History of Microservices*

Most of the Netflix OSS components listed above are pretty old and were optimized more for AWS. Some of those components are prone to some common DevOps pain-points for organizations starting to adopt DevOps. The migration that will be done as part of this demo will help in alleviating those possible DevOps pain-points and provide the organizations a direction on **How to migrate Spring Boot Application to OpenShift**

## 2.2. App Overview

In this demo, the Spring Cloud Samples - Brewery was migrated and optimized for OpenShift, during the process of migration the original Spring Cloud Samples - Brewery was modified to make it deployable on to Kubernetes or OpenShift.

The application was migrated iteratively,

- ☑ Iteration I - As-is deployment of the Spring Cloud Samples - Brewery with no code change. The application build process was be modified to enable easier deployment of application on Openshift
- ☑ Iteration II - Use native Kubernetes / OpenShift features such as service discovery, loadbalancing & externalization of the config
- ☑ Iteration III - Optimizing stacks on ttps://www.openshift.com[OpenShift], like Apache Artemis instead of RabbitMQ, using OpenTracing and Jaeger

## 2.3. Pre-Requisite

You have a OpenShift cluster running locally using **minishift** or **CDK**, or have access to **OpenShift Container Platform**

Check the Tools section for more details

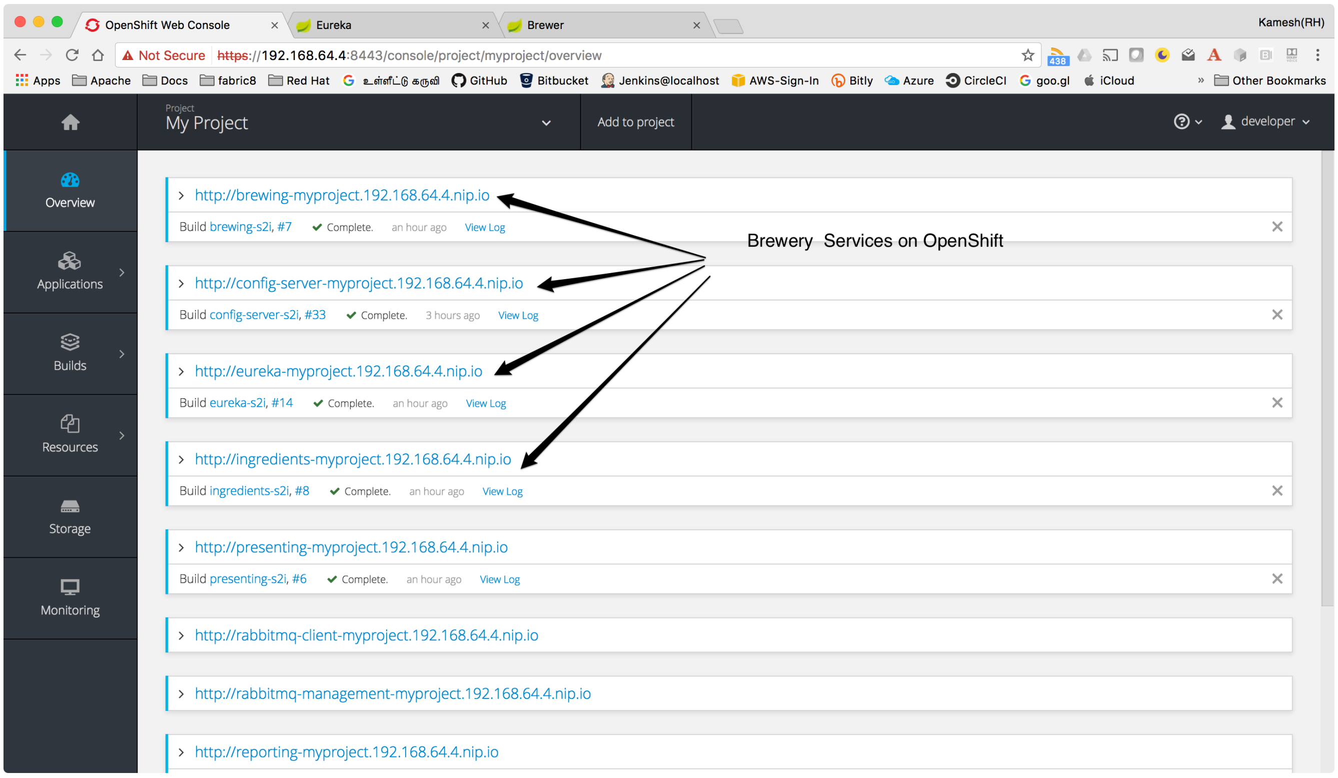
- At least 7Gb of RAM is required to run the Brewery application, atleast for Iteration I

## 2.4. Docker Setup

Before doing any deployment, its recommended to do `eval $(minishift docker-env)` from your current shell, to set up the DOCKER environment variables, that will be needed by the fabric8 maven plugin to deploy application on OpenShift

## 2.5. Accessing the Applications

You can view the application urls from OpenShift Web Console. A successful deployment will have all the applications running with single pod. The following screenshots shows how the Eureka will look like when all the clients registers with it



http://brewing-myproject.192.168.64.4.nip.io
Build brewing-s2i, #7 Complete. an hour ago View Log

http://config-server-myproject.192.168.64.4.nip.io
Build config-server-s2i, #33 Complete. 3 hours ago View Log

http://eureka-myproject.192.168.64.4.nip.io
Build eureka-s2i, #14 Complete. an hour ago View Log

http://ingredients-myproject.192.168.64.4.nip.io
Build ingredients-s2i, #8 Complete. an hour ago View Log

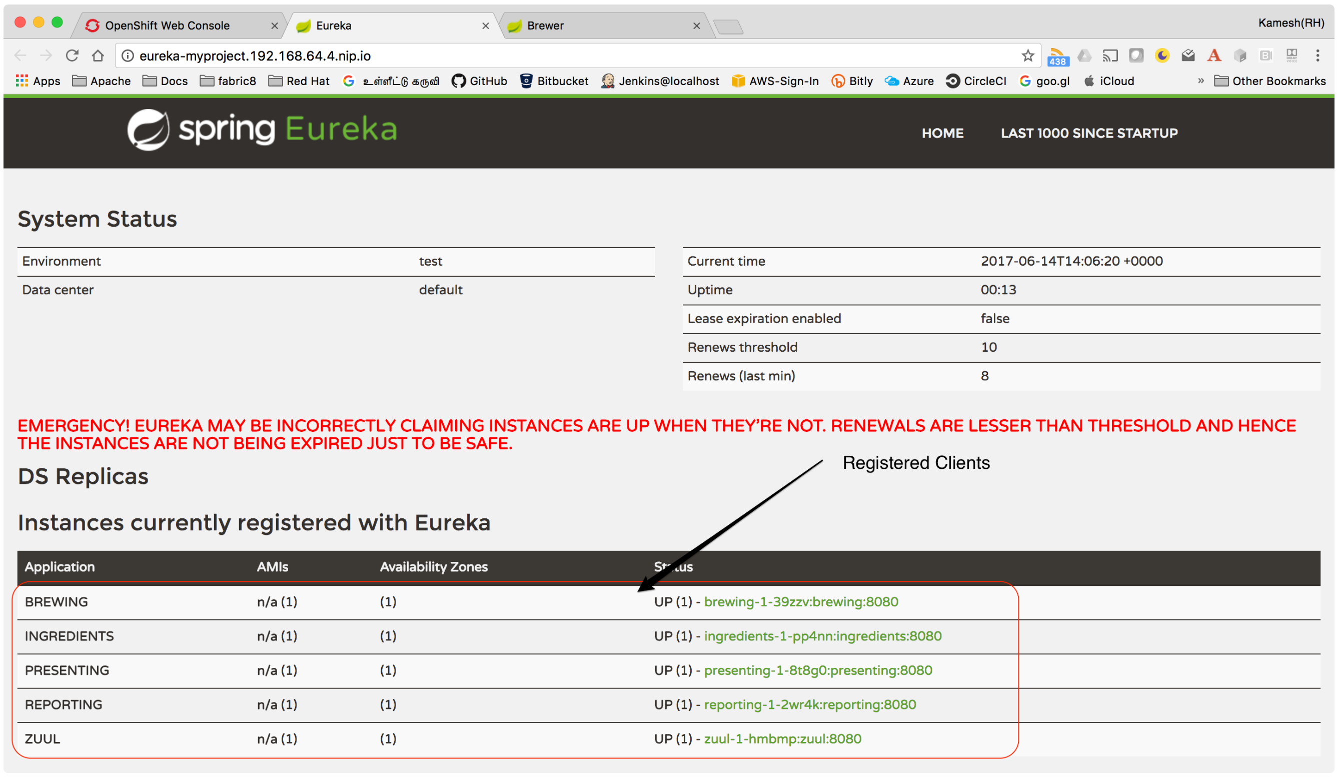
http://presenting-myproject.192.168.64.4.nip.io
Build presenting-s2i, #6 Complete. an hour ago View Log

http://rabbitmq-client-myproject.192.168.64.4.nip.io

http://rabbitmq-management-myproject.192.168.64.4.nip.io

http://reporting-myproject.192.168.64.4.nip.io

*Figure 2. Brewery Services*


OpenShift Web Console
Eureka
Brewer
Kamesh(RH)
eureka-myproject.192.168.64.4.nip.io
Apps Apache Docs fabric8 Red Hat உள்ளீட்டு கருவி GitHub Bitbucket Jenkins@localhost AWS-Sign-In Bitly Azure CircleCI goo.gl iCloud Other Bookmarks
spring Eureka
HOME
LAST 1000 SINCE STARTUP
Registered Clients

System Status

| | |
|---|---|
| Environment | test |
| Data center | default |

| | |
|---|---|
| Current time | 2017-06-14T14:06:20 +0000 |
| Uptime | 00:13 |
| Lease expiration enabled | false |
| Renews threshold | 10 |
| Renews (last min) | 8 |

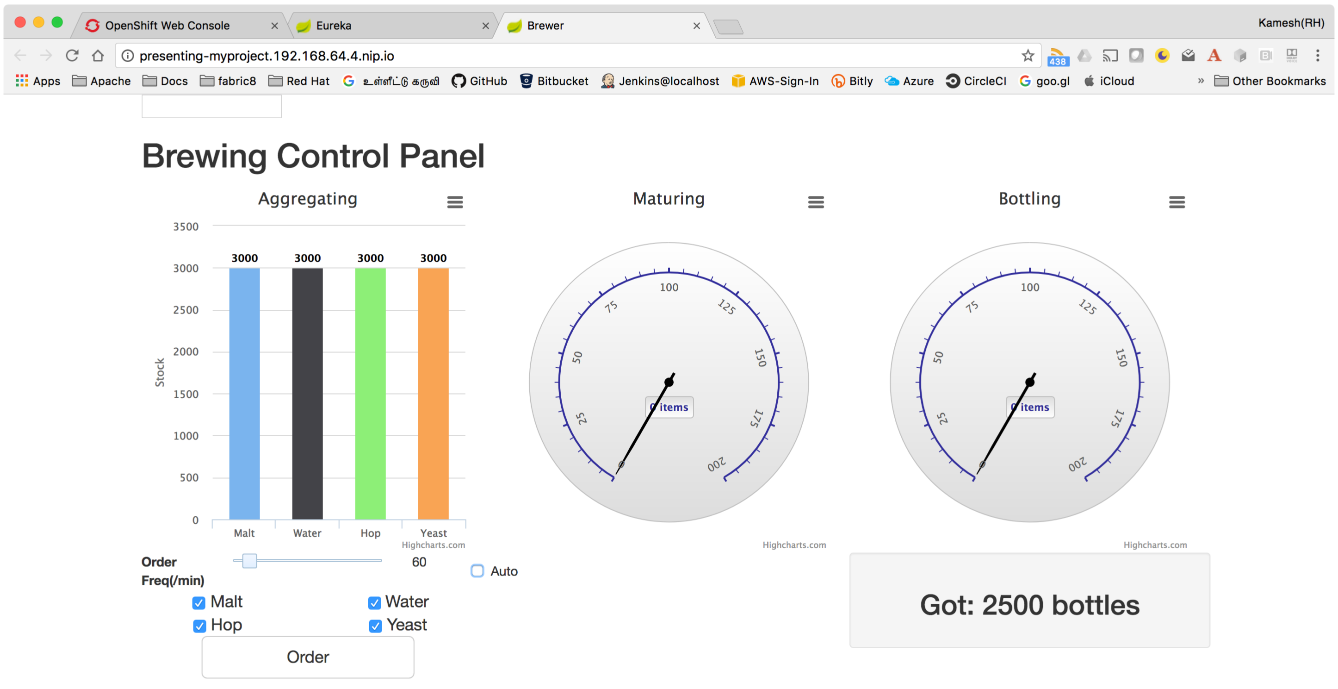
EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| BREWING | n/a (1) | (1) | UP (1) - brewing-1-39zzv:brewing:8080 |
| INGREDIENTS | n/a (1) | (1) | UP (1) - ingredients-1-pp4nn:ingredients:8080 |
| PRESENTING | n/a (1) | (1) | UP (1) - presenting-1-8t8g0:presenting:8080 |
| REPORTING | n/a (1) | (1) | UP (1) - reporting-1-2wr4k:reporting:8080 |
| ZUUL | n/a (1) | (1) | UP (1) - zuul-1-hmbmp:zuul:8080 |

*Figure 3. Eureka on OpenShift*



Got: 2500 bottles

*Figure 4. Brewer App*

# Chapter 3. Iteration I

The Iteration-1 is supposed to be the as-is deployment of the *Brewery* application on to Kubernetes or Openshift. This will have all the components from the original https://github.com/spring-cloud-samples/brewery with modifications required :

- Use the Fabric8 Maven Plugin to generate the Kubernetes/OpenShift resources needed to deploy the applications on the platform
- Design OpenShift templates to deploy RabbitMQ and the different servers such as Eureka, Config-Server, Zipkin etc.,to have them deployed on to Openshift

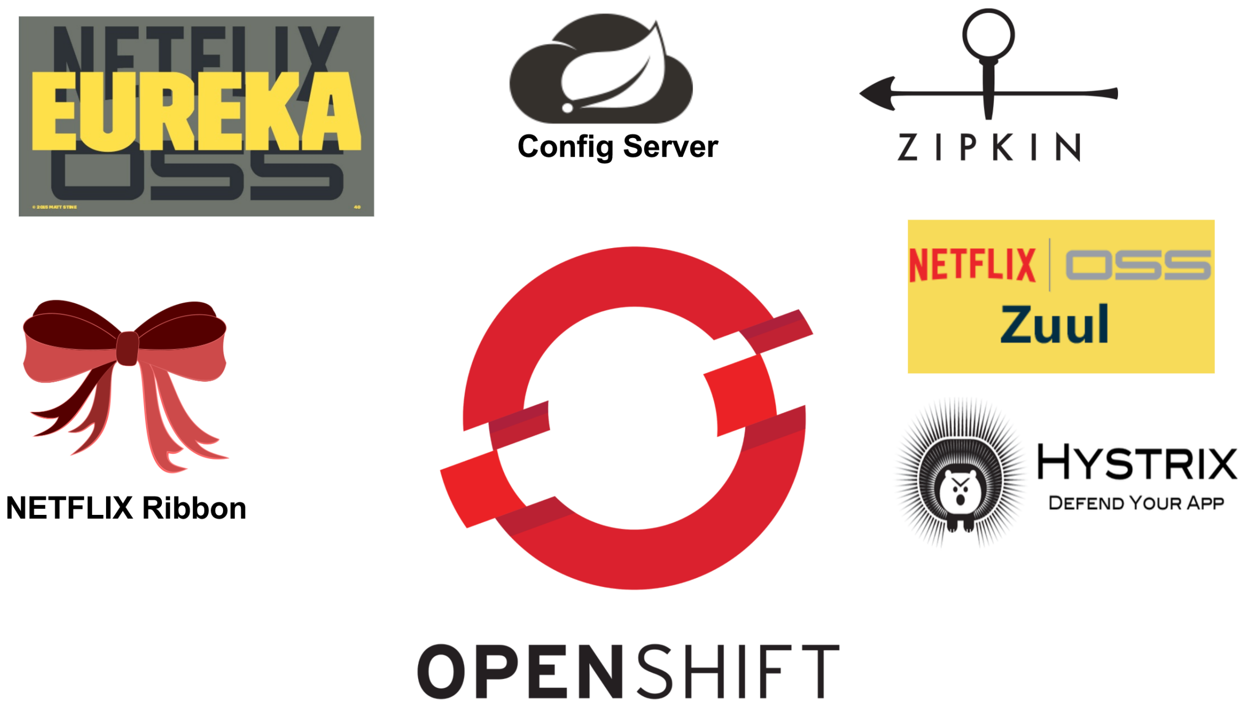


*Figure 5. Java Microservices*

The picture above shows a typical Cloud Native Spring Boot Application deployment with one big difference being the undelying platform being OpenShift in place of usual CloudFoundry

## 3.1. Setup

### 3.1.1. Clone

```
git clone -b iteration-1 https://github.com/redhat-developer-demos/brewery.git
```

Through out this document we will call the directory where the project was cloned as *$PROJECT_HOME*

## 3.2. Pre-Requisite

### 3.2.1. RabbitMQ

The RabbimtMQ container image when run is run using `root` user (UID `0`). The OpenShift by default does not allow container images to be run with UID `0`. In order to allow that we need to define polices and attach to Kubernetes Service Account, to have better clarity and control we create a separate service account called **brewery** and add the required Security Context Constraints(SCC) to allow running the container as UID `0` user. The following commands adds the required SCC to the service account **brewery**,

```
oc adm policy add-scc-to-user privileged -z brewery ①

oc adm policy add-scc-to-user anyuid -z brewery  ①
```

① **brewery** service is created when RabbitMQ is deployed

## 3.3. Deploy Applications

*Table 1. Application List*

| | Application | Folder | Remarks |
|---|---|---|---|
| | RabbitMQ | **$PROJECT_HOME**/extras/rabbitmq | Message Broker - https://www.rabbitmq.com/ |
| | common | **$PROJECT_HOME**/common | Common shared library |
| | Eureka | **$PROJECT_HOME**/eureka | Service Registry - https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance |
| | Config Server | **$PROJECT_HOME**/config-server | Centralized Configuration Server - https://cloud.spring.io/spring-cloud-config/spring-cloud-config.html |
| | Zipkin Server | **$PROJECT_HOME**/zipkin-server | Distributed Tracing system |
| | Zuul | **$PROJECT_HOME**/zuul | Java based Proxy |
| | Ingredients | **$PROJECT_HOME**/ingredients | |

| | Application | Folder | Remarks |
|---|---|---|---|
| | Reporting | **$PROJECT_HOME**/reporting | |
| | Brewing | **$PROJECT_HOME**/brewing | |
| | Presenting | **$PROJECT_HOME**/presenting | |

### 3.3.1. Building

Brewery application uses gradle for build, we will leverage on the same to get the application artifacts ready. To build the applicaiton run the following command

```
./gradlew -DWHAT_TO_TEST="SLEUTH_STREAM" clean build ①
./mvnw -N install ②
```

① We will be using Spring Cloud Sleuth for sending trace information to Zipkin

② This will install the brewery parent pom in local maven repository

### 3.3.2. Deploying to OpenShift

As part of this lift and shift i.e to make existing application to work **as-is**, there is certain order of application deployment might be required. The order below is not hard rule but the best known working order during the migration effort.

#### RabbitMQ

Go to the directory **$PROJECT_HOME/extras/rabbitmq**, and execute the following command

```
./mvnw -Dfabric8.mode=kubernetes clean fabric8:deploy
```

This will take some time to get it running as the deployment needs to download the `rabbitmq` docker image

#### Config Server

Go to the directory **$PROJECT_HOME/config-server**, and execute the following command

```
./mvnw clean fabric8:deploy
```

Since this is the first Java application to be deployed, it may take some time to download the necessary images from docker hub.

**Eureka**

Go to the directory **$PROJECT_HOME/eureka**, and execute the following command

```
./mvnw clean fabric8:deploy
```

**Zipkin Server**

Go to the directory **$PROJECT_HOME/zipkin-server**, and execute the following command

```
./mvnw clean fabric8:deploy
```

**Zuul**

Go to the directory **$PROJECT_HOME/zuul**, and execute the following command

```
./mvnw clean fabric8:deploy
```

**Ingredients**

Go to the directory **$PROJECT_HOME/ingredients**, and execute the following command

```
./mvnw clean fabric8:deploy
```

**Reporting**

Go to the directory **$PROJECT_HOME/reporting**, and execute the following command

```
./mvnw clean fabric8:deploy
```

**Brewing**

Go to the directory **$PROJECT_HOME/brewing**, and execute the following command

```
./mvnw clean fabric8:deploy
```

**Presenting**

Go to the directory **$PROJECT_HOME/presenting**, and execute the following command

```
./mvnw clean fabric8:deploy
```

# Chapter 4. Iteration II

The Iteration II will deprecate few of the NetFlix OSS components that are superflous inside Kubernetes or OpenShift. The following sections shows how to get the Iteration II deployed on to Kubernetes or OpenShift. This iteration uses the Spring Cloud Kubernetes - the Spring Cloud based discovery client for Kubernetes

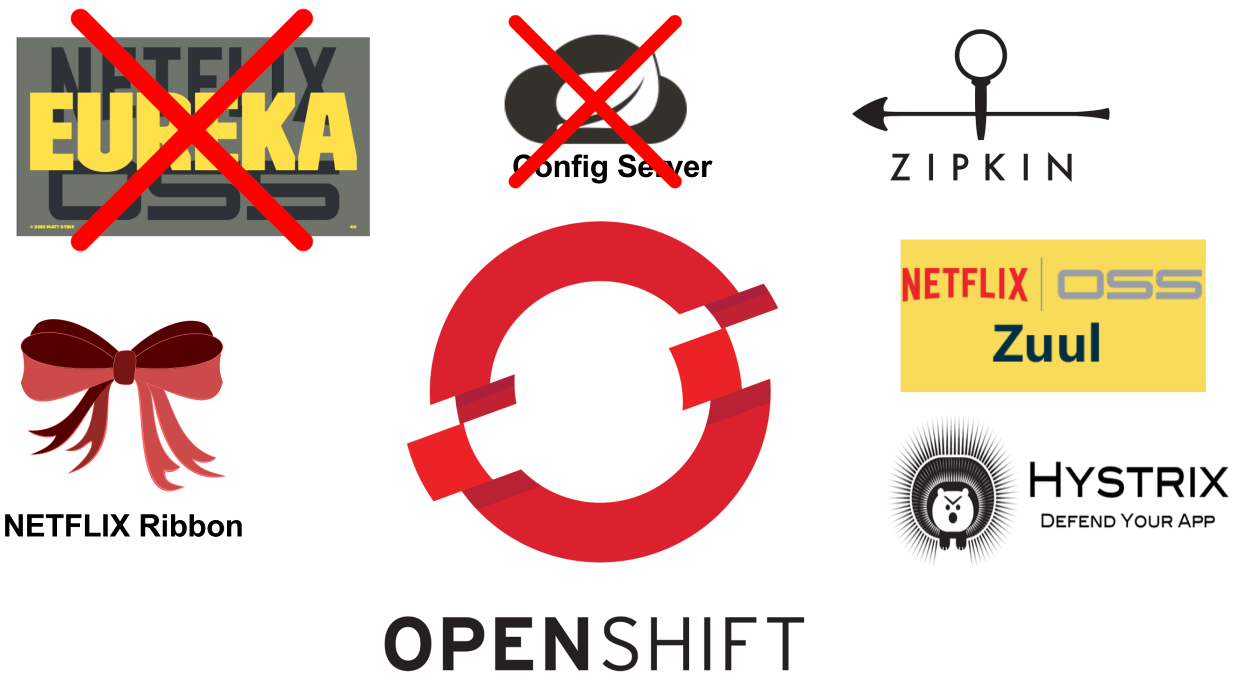


*Figure 6. Java Microservices*

The picture above shows how Java Microservices Platform - Iteration-1 has evolved post deployment to OpensShift, especially how components like Eureka and Config Server became superflous and has been deprecated.

## 4.1. Setup

### 4.1.1. Clone

```
git clone -b iteration-2 https://github.com/redhat-developer-demos/brewery.git
```

Through out this document we will call the directory where the project was cloned as *$PROJECT_HOME*

## 4.2. Pre-Requisite

### 4.2.1. General

The spring-cloud-kubernetes library used in the project requires the `default` service account to have view permissions, to enable that we execute the following command,

```
oc policy add-role-to-user view -z default -n $(oc project -q)
```

The Service Account `default` does not have any permission, in order to allow `default` SA to lookup the ConfigMaps within the namespace, it need to be added with `view` role

### 4.2.2. RabbitMQ

The RabbimtMQ container image when run is run using `root` user (UID `0`). The OpenShift by default does not allow container images to be run with UID `0`. In order to allow that we need to define polices and attach to Kubernetes Service Account, to have better clarity and control we create a separate service account called **brewery** and add the required Security Context Constraints(SCC) to allow running the container as UID `0` user. The following commands adds the required SCC to the service account **brewery**,

```
oc adm policy add-scc-to-user privileged -z brewery ①

oc adm policy add-scc-to-user anyuid -z brewery  ①
```

① **brewery** service is created when RabbitMQ is deployed

## 4.3. Deploy Applications

*Table 2. Application List*

| | Application | Folder | Remarks |
|---|---|---|---|
| | RabbitMQ | **$PROJECT_HOME**/extras/rabbitmq | Message Broker - https://www.rabbitmq.com/ |
| | common | **$PROJECT_HOME**/common | Common shared library |
| | common-zipkin-stream | **$PROJECT_HOME**/ommon-zipkin-stream | Common shared library for the projects that uses the Sleuth Zipkin Stream for tracing |
| **X** | eureka | **$PROJECT_HOME**/eureka | Application will use Kubernetes Services |

|  | Application | Folder | Remarks |
|---|---|---|---|
| X | config-server | **$PROJECT_HOME**/config-server | Application will use Kubernetes ConfigMaps |
|  | Zipkin Server | **$PROJECT_HOME**/zipkin-server | Distributed Tracing system |
|  | Zuul | **$PROJECT_HOME**/zuul | Java based Proxy |
|  | Ingredients | **$PROJECT_HOME**/ingredients |  |
|  | Reporting | **$PROJECT_HOME**/reporting |  |
|  | Brewing | **$PROJECT_HOME**/brewing |  |
|  | Presenting | **$PROJECT_HOME**/presenting |  |

### 4.3.1. Building

The Iteration II of the brewery application has migrated all the projects to Apache Maven based build.

To build the application run the following command:

```
./mvnw -N install ①
./mvnw clean install ②
```

① This will install the brewery parent pom in local maven repository

② This will build the applications, if the minishift or OpenShift cluster is running, this will trigger `s2i` builds of the respective application as well

### 4.3.2. Deploying to OpenShift

The following section details on how to deploy the Iteration II to OpenShift.

Ensure that all Pre-Requisite are done before starting deployment.

### RabbitMQ

#### Local Deployment

Go to the directory **$PROJECT_HOME/extras/rabbitmq**, and execute the following command

```
./mvnw -Dfabric8.mode=kubernetes clean fabric8:deploy
```

#### External Cloud Deployment

Sometimes you might have access to docker socket typical case when deploying to external cloud, in those cases you can run the following set of commands,

```
./mvnw clean fabric8:resource
oc apply -f target/classes/META-INF/fabric8/openshift.yml
```

This will take some time to get it running as the deployment needs to download the `rabbitmq` docker image

### Zipkin Server

Go to the directory **$PROJECT_HOME/zipkin-server**, and execute the following command

```
./mvnw fabric8:deploy
```

### Zuul

Go to the directory **$PROJECT_HOME/zuul**, and execute the following command

```
./mvnw fabric8:deploy
```

### Ingredients

Go to the directory **$PROJECT_HOME/ingredients**, and execute the following command

```
./mvnw fabric8:deploy
```

### Reporting

Go to the directory **$PROJECT_HOME/reporting**, and execute the following command

```
./mvnw fabric8:deploy
```

**Brewing**

Go to the directory **$PROJECT_HOME/brewing**, and execute the following command

```
./mvnw fabric8:deploy
```

**Presenting**

Go to the directory **$PROJECT_HOME/presenting**, and execute the following command

```
./mvnw fabric8:deploy
```

## 4.4. Acceptance Testing

The **$PROJECT_HOME/acceptance-tests** holds the test cases for testing the application. To perform we need to have have some ports forwarded from Kubernetes/OpenShift to localhost(where you build the application)

```
oc port-forward zipkin-1-06wmt 9411:8080 ①
oc port-forward presenting-1-wzhfn 9991:8080 ②
```

① forward port 8080 from Zipkin pod to listen on localhost:9411

② forward port 8080 from Presenting pod to listen on localhost:9991

Please update the pod names based on your local deployment

To run acceptance testing, execute following command from $PROJECT_HOME,

```
./mvnw clean test
```

## 4.5. Deprecated Modules

As part of Iteration-II the following modules have been deprecated,

- Eureka
- Config Server
- common-zipkin
- common-zipkin-old
- zookeeper
- docker

# Chapter 5. Iteration III

The Iteration III is more of optimizing the Iteration II on OpenShift. This iteration has fair bit of code change that is required when porting the application to use

- Apache Artemis in place of RabbitMQ
- OpenTracing for tracing and  Jaeger in place of Zipkin
- Native Kubernetes based Load Balancing instead of Ribbon

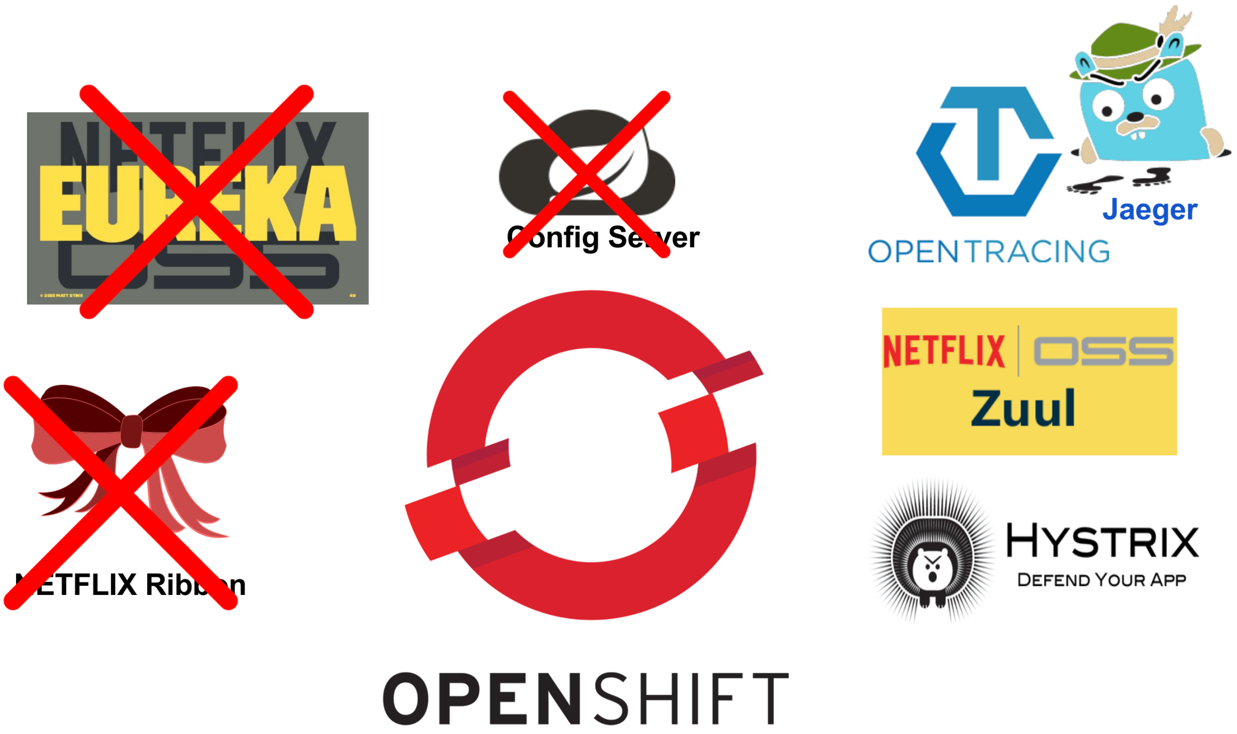


*Figure 7. Java Microservices*

The picture above shows how ,Java Microservices Platform - Iteration-2 has been further optimized for OpenShift, with notable difference of deprecating Netflix Ribbon, as application can rely on Kubernetes / Openshift for client side load balancing.  As part of the optimization from Iteration II, many modules/projects are deprecated.

## 5.1. Setup

### 5.1.1. Clone

```
git clone -b iteration-3 https://github.com/redhat-developer-demos/brewery.git
```

Through out this document we will call the directory where the project was cloned as *$PROJECT_HOME*

## 5.2. Pre-Requisite

### 5.2.1. General

The spring-cloud-kubernetes library used in the project requires the `default` service account to have view permissions, to enable that we execute the following command,

```
oc policy add-role-to-user view -z default -n $(oc project -q)
```

## 5.3. Deploy Applications

*Table 3. Application List*

| | Application | Folder | Remarks |
|---|---|---|---|
| | Apache Artemis | **$PROJECT_HOME**/extras/apache-artemis | Message Broker - https://activemq.apache.org/artemis/ |
| | common | **$PROJECT_HOME**/common | Common shared library, the shared libraries dependencies are updated to leverage RHOAR 1.5.7 BOM |
| **X** | common-zipkin-stream | **$PROJECT_HOME**/common-zipkin-stream | Since this iteration has moved all the tracing Components to OpenTracing, this module is deprecated/obsolete as no Sleuth Stream will be used, instead the project will use Open Tracing Java modules |
| **X** | eureka | **$PROJECT_HOME**/eureka | Application will use Kubernetes Services |
| **X** | config-server | **$PROJECT_HOME**/config-server | Application will use Kubernetes ConfigMaps |
| **X** | zipkin-server | **$PROJECT_HOME**/zipkin-server | Distributed Tracing system |
| | Jaeger Server | Jaeger already provides OpenShift manifests to deploy the same in OpenShift | Jaeger, a high performing OpenTracing based implementation of Distributed Tracing |

| | Application | Folder | Remarks |
|---|---|---|---|
| | Zuul | **$PROJECT_HOME**/zuul | Java based Proxy |
| | Ingredients | **$PROJECT_HOME**/ingredients | |
| | Reporting | **$PROJECT_HOME**/reporting | |
| | Brewing | **$PROJECT_HOME**/brewing | |
| | Presenting | **$PROJECT_HOME**/presenting | |

### 5.3.1. Building

The Iteration II of the brewery application has migrated all the projects to Apache Maven based build, to build the application run the following command

```
./mvnw -N install ①
./mvnw clean install ②
```

① This will install the brewery parent pom in local maven repository

② This will build the applications, if the minishift or OpenShift cluster is running, this will trigger `s2i` builds of the respective application as well

### 5.3.2. Deploying to OpenShift

As part of this lift and shift of existing application, to make it work as-is, there is certain order of applicaiton deployment might be required. The following section explains the deployment of the application in the same order as expected ( you can expriment with it if you like :) )

❗ Ensure that all Pre-Requisite are done before starting deployment.

#### Apache Artemis

Starting this iteration, the application will be using Apache Artemis as message broker in place of RabbitMQ, the following sections details on deploying Apache Artemis on OpenShift

##### Local Deployment

Go to the directory **$PROJECT_HOME/extras/apache-artemis**, and execute the following command

```
./mvnw -Dfabric8.mode=kubernetes clean fabric8:deploy
```

#### External Cloud Deployment

Sometimes you might have access to docker socket typical case when deploying to external cloud, in those cases you can run the following set of commands,

```
./mvnw clean fabric8:resource
oc apply -f target/classes/META-INF/fabric8/openshift.yml
```

This will take some time to get it running as the deployment needs to download the `apache-artemis` docker image

### Jaeger Server

The Jaeger distribution provides the OpenShift deployment manifests to deploy Jaeger, as part of this demo the all-in-one deployment will be used.

```
cd $PROJECT_HOME/extras/jaeger
oc process -f jaeger-all-in-one-template.yml  | oc create -f -
```

Please use the template from sources for all-in-deployment of Jaeger, any version above 0.8.0 is not able to display order traces as expected

### Zuul

Go to the directory **$PROJECT_HOME/zuul**, and execute the following command

```
./mvnw fabric8:deploy
```

### Ingredients

Go to the directory **$PROJECT_HOME/ingredients**, and execute the following command

```
./mvnw fabric8:deploy
```

### Reporting

Go to the directory **$PROJECT_HOME/reporting**, and execute the following command

```
./mvnw fabric8:deploy
```

**Brewing**

Go to the directory **$PROJECT_HOME/brewing**, and execute the following command

```
./mvnw fabric8:deploy
```

**Presenting**

Go to the directory **$PROJECT_HOME/presenting**, and execute the following command

```
./mvnw fabric8:deploy
```

## 5.4. Open Issues

### 5.4.1. Tracing

The traces generated by OpenTracing right now has order traces broken because right now there is no instrumentation for Spring Streams. The instrumentation is under development.

### 5.4.2. Acceptance Testing

As this iteration has lot of module updates and replacements, the old acceptance tests does not hold good. The automated Arquillain based automated tests development is in progress, this section will be updated with needed details once its in place.

## 5.5. Deprecated Modules

As part of Iteration-III the following modules have been deprecated,

- Eureka
- Config Server
- common-zipkin
- common-zipkin-old
- common-zipkin-stream
- zipkin-server
- zookeeper
- docker

# Chapter 6. Resources

## 6.1. Documentation

- Maven Properties
- Kubernetes
- Openshift

## 6.2. Tools

There are two main tools that is always used with Spring Boot on OpenShift

- RedHat Container Development Kit
- fabric8 maven plugin

You can also find related tools and downloads from https://developers.redhat.com

This page lists some commonly used commands, tips/tricks and some trouble shooting tips around these tools.

This is not meant to replace original setup guide, please refer to the original setup guides for detailed setup of these tools.

### 6.2.1. Minishift

```
oc login --server <your openshift master server url> -u developer
```

Logging into OpenShift from cli using the user developer and the default password is developer.

```
eval $(minishift oc-env)
```

Sets the right path to the OpenShift cli

```
minishift console
```

Opens the OpenShift web console in the default browser

### 6.2.2. fabric8 maven plugin

```
eval $(minishift docker-env)
```

This is very important command that you need to run before the first maven build, as this allows setting some important docker variables

```
mvn io.fabric8:fabric8-maven-plugin:3.5.30:setup
```

Sets up the fabric8 maven plugin in the current maven project.

In the above code 3.5.30 is used as version, please update to version that suits your needs

```
mvn fabric8:deploy
```

Deploys the current maven project into OpenShift

```
mvn fabric8:undeploy
```

UnDeploys the current maven project from OpenShift

```
mvn fabric8:debug
```

Setups up port forward to debug the current OpenShift project

```
mvn fabric8:run
```

Quick deploy the current maven project to OpenShift, runs in foreground and undeploys once CTRL + C is used to terminate current process in foreground.

## 6.3. Blogs

Getting started with Spring Boot on OpenShift

Spring Boot and OAuth2 with Keycloak - RHD Blog

Configuring Spring Boot Application on Kubernetes - RHD Blog

Configuring Spring Boot on Kubernetes with ConfigMap - RHD Blog

Configuring Spring Boot on Kubernetes with Secrets - RHD Blog

## 6.4. Demos

Spring Boot on OpenShift 101 Quickstart SprigBoot on OpenShift

Step by Step approach to make an existing CloudFoundry ready Spring Boot application and its related workloads to OpenShift Green Cloud Demo

How to do a stateful Canary deployment using Spring Boot and Infnispan Popular Movie Store

How to do integration Test of Spring Boot Application on OpenShift Spring Boot Integration Test on OpenShift

Using Kubernetes ConfigMaps with Spring Boot Configure Kubernetes ConfigMaps with Spring Boot

Using Kubernetes Secrets with Spring Boot Configure Kubernetes Secrets with Spring Boot

## 6.5. Videos

Quick Start Spring Boot on OpenShift

Debug Spring Boot Application on OpenShift