



# OPENSHIFT HANDS-ON @Microsoft

Développer et déployer une application Cloud-Native

Guillaume Estrem & Laurent Broudoux  
AppDev Solution Architect  
21 Février 2019



# LAB GUIDE

13h00

- Lab 1 Getting Started
- Lab 2 Deploying containers from an image
- Lab 3 Deploying containers from sources
- Lab 4 Monitoring application health

14h30

PAUSE

14h45

- Lab 5 Distributed Tracing Configuration
- Lab 6 Getting Application Metrics
- Lab 7 Azure Service Broker
- Lab 8 Continuous Delivery

16h00

# OPENSHIFT CONCEPTS OVERVIEW



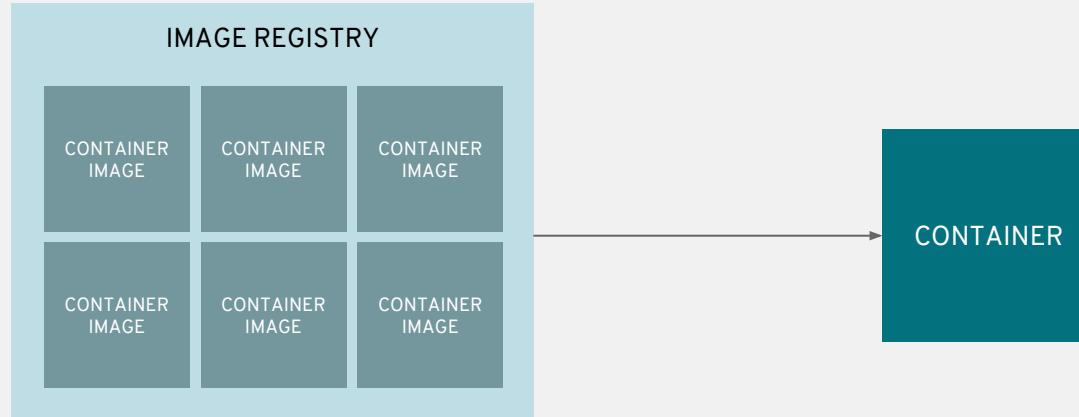
# A container is the smallest compute unit



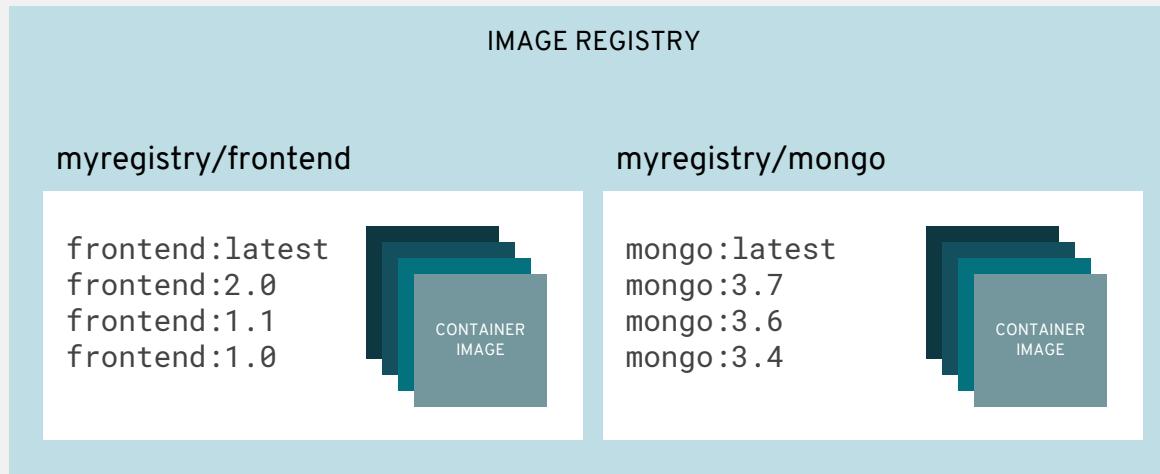
# Containers are created from container images



# Container images are stored in an image registry



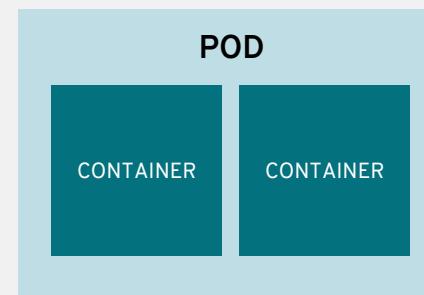
# An image repository contains all versions of an image in the image registry



# Containers are wrapped in pods which are units of deployment and management

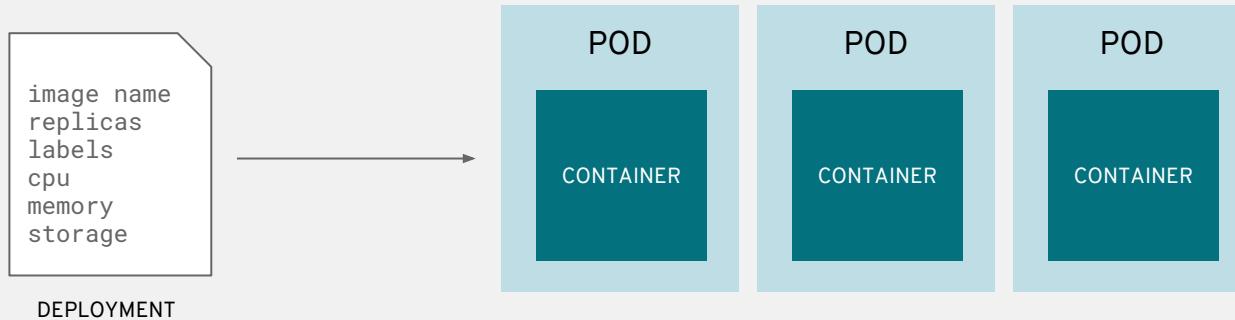


IP: 10.1.0.11

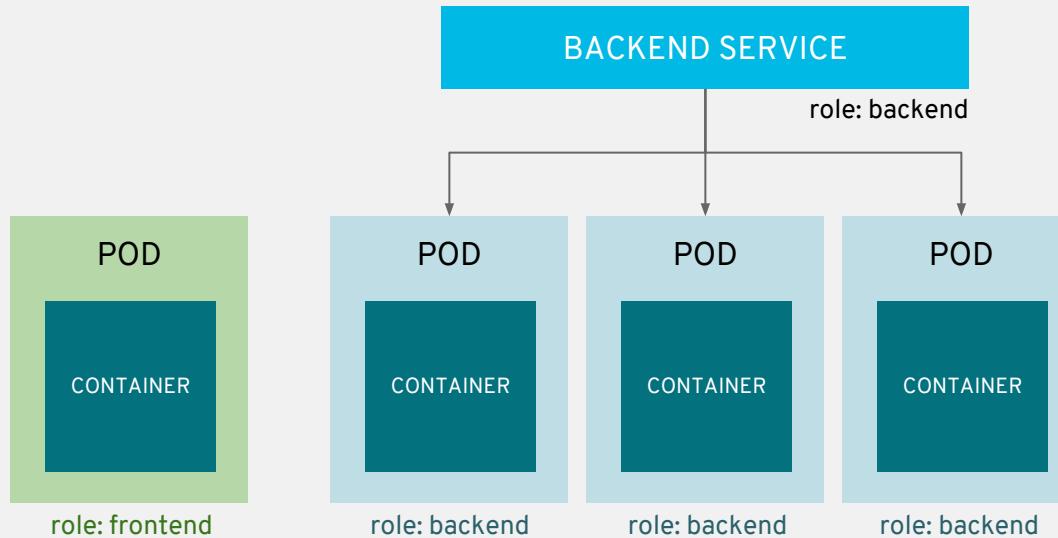


IP: 10.1.0.55

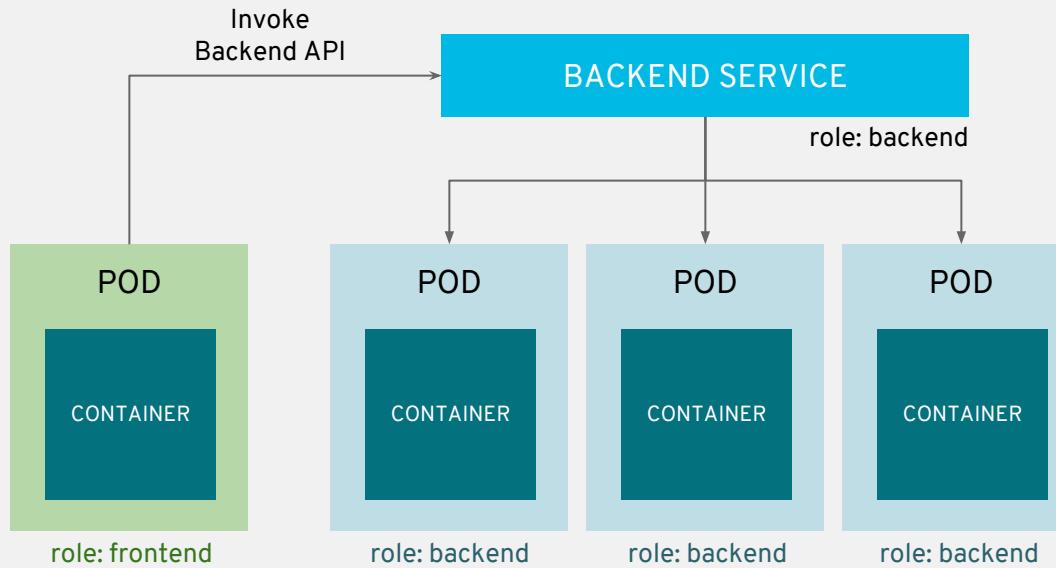
# Pods configuration is defined in a deployment



# Services provide internal load-balancing and service discovery across pods



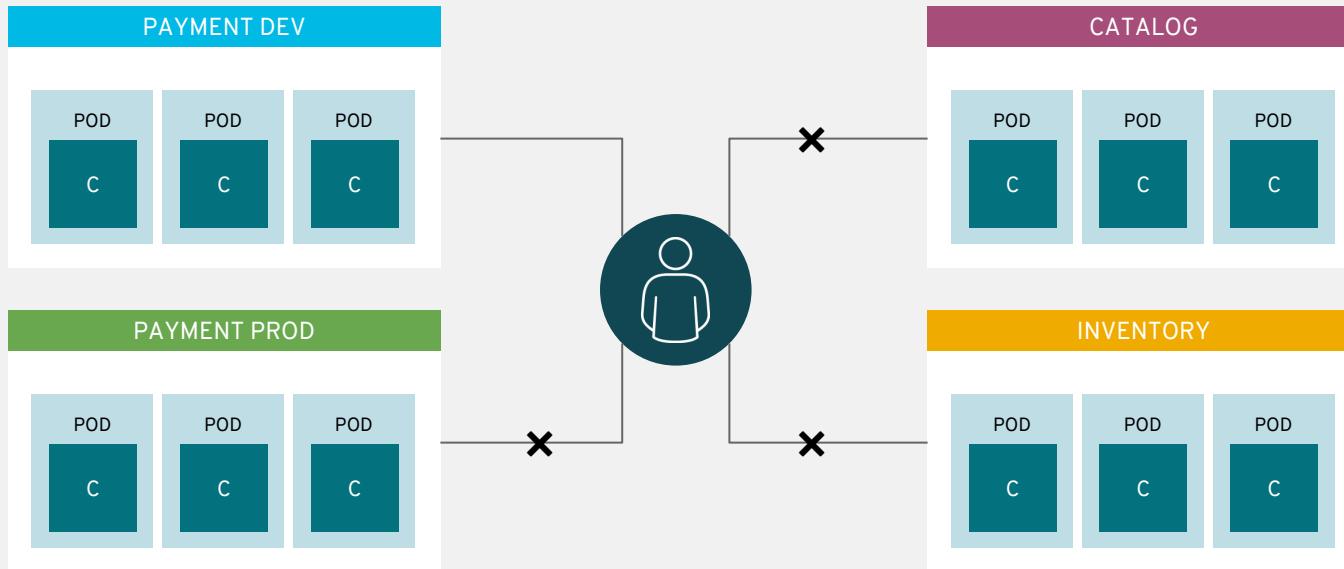
# Apps can talk to each other via services



# Routes add services to the external load-balancer and provide readable urls for the app



# Projects isolate apps across environments, teams, groups and departments



# LAB 1

## Getting started

# Pick your user ID

Go to <http://bit.ly/ocp-on-azure> and assign your name to a user available. This user will be your identity during the workshop.  
Don't use your neighbour user ;)

# Connect via SSH to the bastion

The bastion contains all tools needed for the following workshop.

Open your terminal and execute the following command :

```
$ ssh userX@52.143.152.215
```

For Windows users, download and install Putty :

<https://www.ssh.com/ssh/putty/windows/install>

# Before starting...

- Make sure you have a userId (**userX**). Each attendee has its own environment on OpenShift Container Platform
- Fork the GitHub repo <https://github.com/lbroudoux/ocp-on-azure-workshop> into your own GitHub and clone it in your home directory /home/userX/ on the bastion
- Open a terminal and login into Openshift with the following credentials

```
$ oc login https://masterdnsccbvtzdhvuyqe.francecentral.cloudapp.azure.com/ -u userX -p mypassword
```

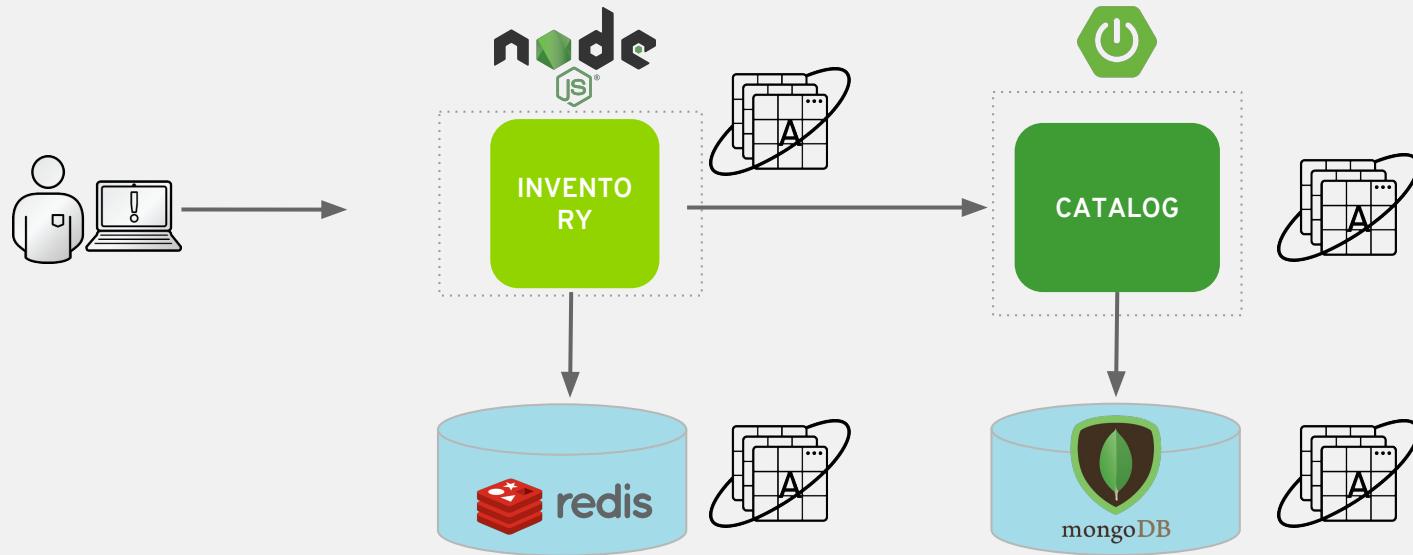
```
Login successful.
```

```
You have access to the following projects and can switch between them with 'oc project <projectname>':
```

# APPLICATION ARCHITECTURE OVERVIEW



# Grocery Store on OpenShift



# LAB 2

Deploy containers from an image

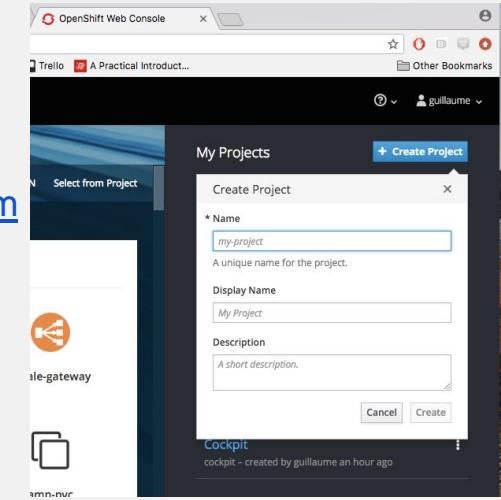
# Create your development environment

Let's go the Web Console

- Via the web console :

<https://masterdnsccbmvtdzhvuqye.francecentral.cloudapp.azure.com>

- Login with the same credentials
- Create a Project with the following informations
  - Name : **fruits-grocery-dev-userX**
  - Display Name: **UserX - Fruits Grocery - Dev**

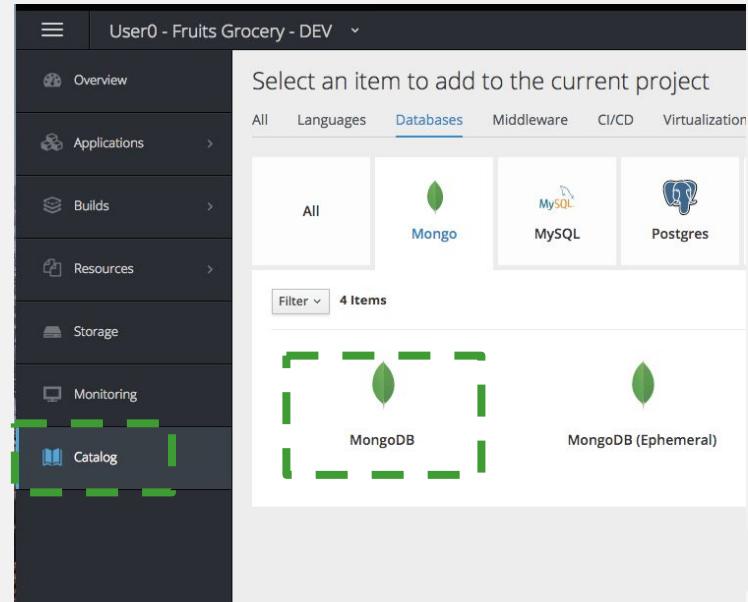


# Deploy MongoDB database via the catalog

- Browse the service catalog and search for MongoDB

- Set MongoDb Database Name
  - Name : fruitsdb
- Save and label the deployment config with the command below

```
$ oc label dc/mongodb app=fruits-catalog
```



# Check MongoDB deployment

Other Resources

DEPLOYMENT CONFIG  
**mongodb, #1**

CONTAINERS

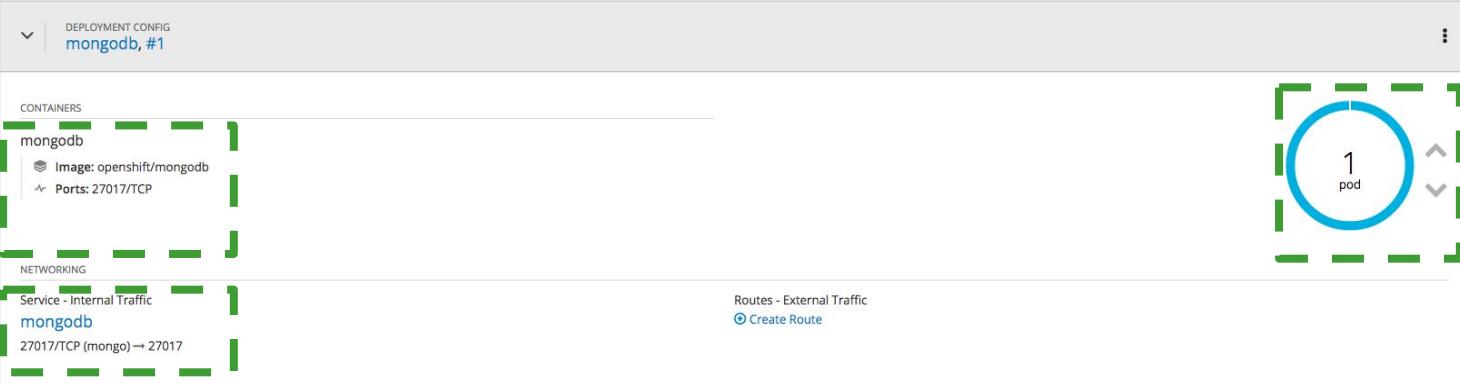
- mongodb
  - Image: openshift/mongodb
  - Ports: 27017/TCP

NETWORKING

- Service - Internal Traffic
  - mongodb**
    - 27017/TCP (mongo) → 27017

Routes - External Traffic  
Create Route

1 pod



One Pod is running. Explore the objects created by OpenShift : image used, TCP port opened and service created

# Deploy Redis via the CLI

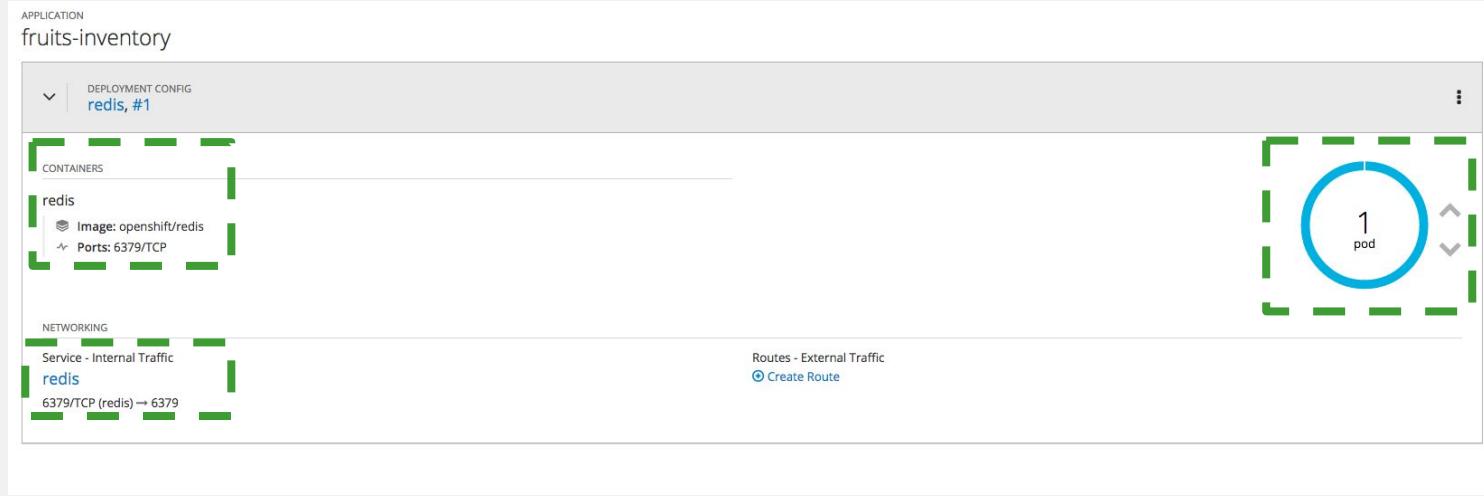
Let's do the deployment of Redis through the CLI rather than the Web console

```
$ oc new-app redis-persistent --name=redis -p DATABASE_SERVICE_NAME=redis -l app=fruits-inventory -n fruits-grocery-dev-userX
```

Quick overview of the command line

- “redis-persistent” is the template we use from the catalog
- We specify also a label (`app=fruits-inventory`) to select easily all resources related to fruits-inventory in our environment
- `DATABASE_SERVICE_NAME` is the service to reach all pods related to Redis

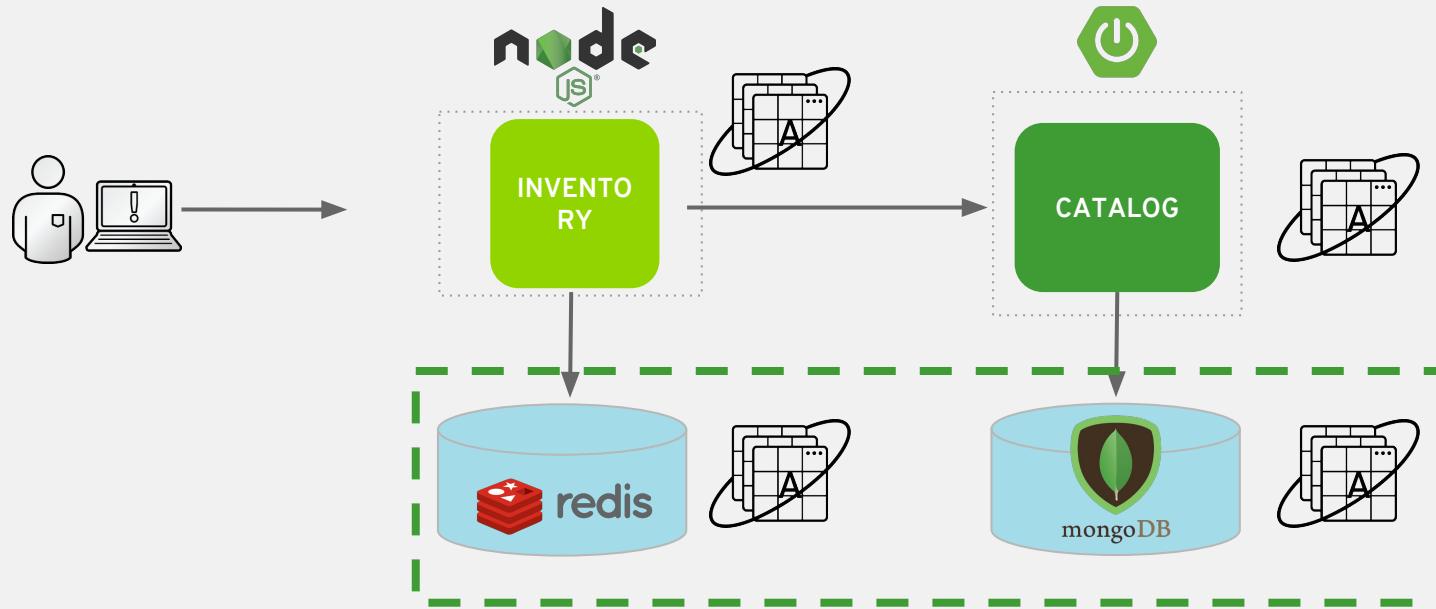
# Check Redis deployment



One Pod is running. Explore the objects created by OpenShift : image used, TCP port opened and service created



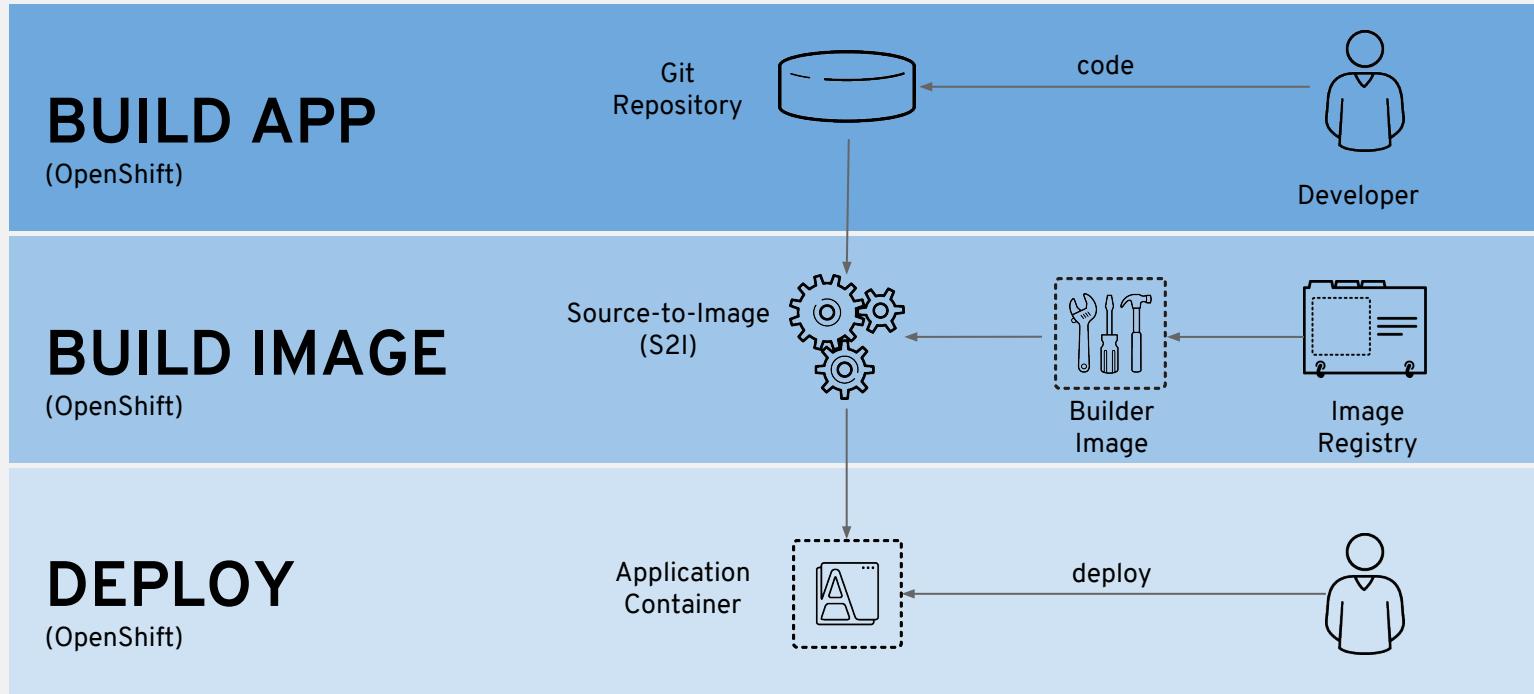
# Grocery Store on OpenShift



# LAB 3

Deploy containers from source

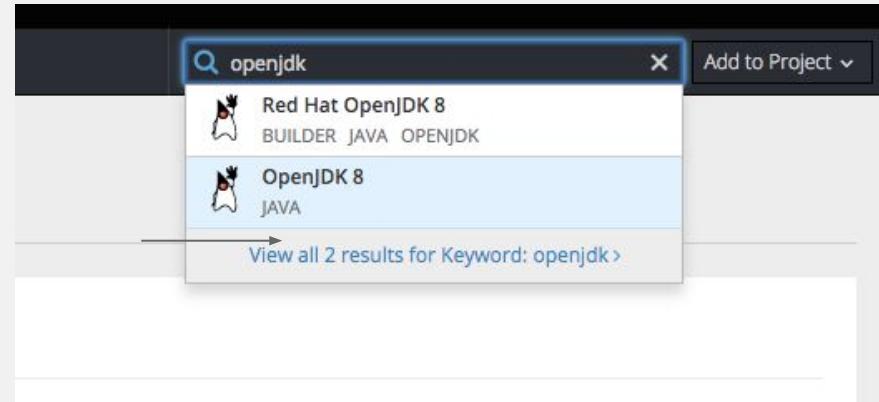
# Deploy the fruits catalog with s2i strategy



# Deploy the fruits catalog

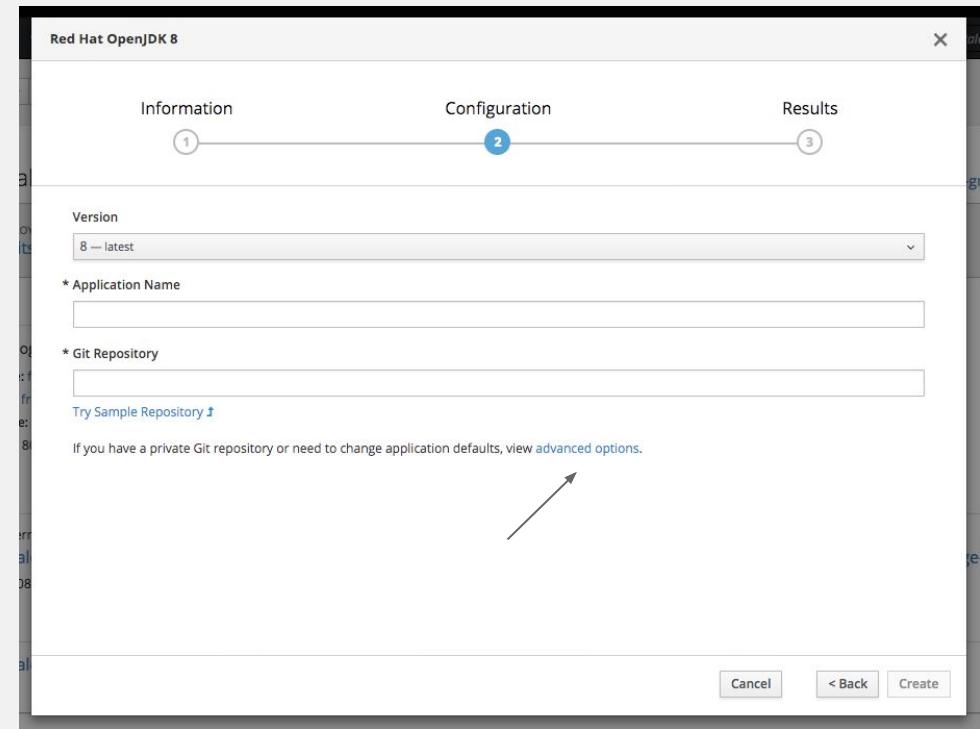
Our application is developed with Spring Boot. A powerful Java framework to build next-gen application and leverage Openshift capabilities.

Let's use the official Red Hat OpenJDK 8 image Builder to create our container Image from the source code.



# Deploy the fruits catalog

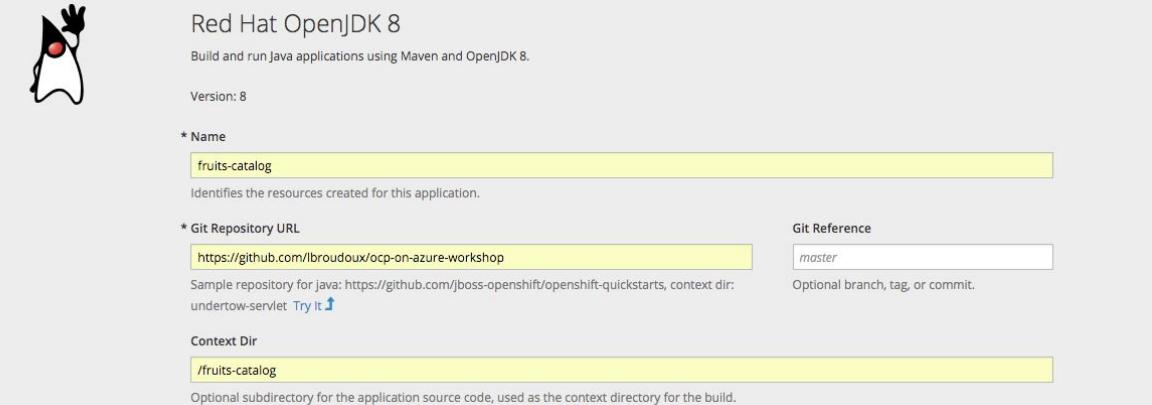
Let's explore Advanced options to specify environment variables and extras things!



# Deploy the fruits catalog

Complete source code informations to build the SpringBoot app in Openshift

- **Name :**  
fruits-catalog
- **Context Dir :**  
/fruits-catalog
- **Your Git repo URL**



The screenshot shows the configuration page for a new application named 'fruits-catalog'. The application is based on 'Red Hat OpenJDK 8' and is intended to 'Build and run Java applications using Maven and OpenJDK 8.' The version is set to 8. The 'Name' field is filled with 'fruits-catalog', which is described as identifying the resources created for this application. The 'Git Repository URL' is set to <https://github.com/lbroudoux/ocp-on-azure-workshop>, with a note that it is a sample repository for Java. The 'Git Reference' is set to 'master'. The 'Context Dir' is set to '/fruits-catalog', which is described as an optional subdirectory for the application source code used as the context directory for the build.

# Deploy the fruits catalog

Set environment variables for database credentials and URI

- MONGODB\_USER - pick the right secret
- MONGODB\_PASSWORD - pick the right secret
- SPRING\_DATA\_MONGODB\_URI :  
`mongodb://${MONGODB_USER}:${MONGODB_PASSWORD}@mongodb:27017/fruitsdb`

MongoDB credentials are located in a secret named `mongodb`

Environment Variables

Container fruits-catalog

MONGODB_USER	mongodb - Secret	database-user	x
MONGODB_PASSWORD	mongodb - Secret	database-password	x
SPRING_DATA_MONGODB_URI	mongodb://\${MONGODB_USER}:\${MONGODB_PASSWORD}@mongodb:27017/fruitsdb		x

Add Value | Add Value from Config Map or Secret

# Deploy the fruits catalog

Explore the application resources deployed

Container Image,  
Build used, ports,  
routes ...

The screenshot shows the OpenShift application details page for 'fruits-catalog'. The top navigation bar includes 'APPLICATION', 'fruits-catalog', and a URL: <http://fruits-catalog-fruits-grocery-dev-user0.ge-apps.openhybridcloud.io>. The main content area is divided into sections: 'DEPLOYMENT CONFIG' (fruits-catalog, #5), 'CONTAINERS' (fruits-catalog, showing image, build, source, and ports), 'NETWORKING' (Service - Internal Traffic, fruits-catalog, port mapping), and 'BUILDS' (fruits-catalog). A large green dashed box encloses the container and networking sections. To the right, a blue circle labeled '1 pod' is shown within a green dashed box, with a tooltip indicating 'Routes - External Traffic' and the URL <http://fruits-catalog-fruits-grocery-dev-user0.ge-apps.openhybridcloud.io>. A note at the bottom right states 'Build #1 is complete created 2 hours ago'.

Click on the blue circle to explore the pod instance

# Deploy the fruits catalog

Explore the pod configuration

- Check Environment variables
- Access to the terminal
- Explore application logs
- Visualize metrics

The screenshot shows the OpenShift web console interface for a pod named 'fruits-catalog-5-2crzd' created 24 minutes ago. The pod is part of a deployment named 'fruits-catalog'.

**Status:** Running  
**Deployment:** fruits-catalog, #5  
**IP:** 10.128.0.247  
**Node:** guillaume (217.182.221.197)  
**Restart Policy:** Always

**Container fruits-catalog**

<b>State:</b>	Running since Feb 14, 2019 5:34:51 PM
<b>Ready:</b>	true
<b>Restart Count:</b>	0

**Template**

**Containers**  
fruits-catalog

- Image: fruits-grocery-dev-user0/fruits-catalog 5eeb089 351.6 MB
- Build: fruits-catalog, #1
- Source: Adding Grafana dashboard 0ee9744 authored by lbroudoux
- Ports: 8080/TCP, 8443/TCP, 8778/TCP
- Mount: default-token-lv2bv → /var/run/secrets/kubernetes.io/serviceaccount read-only

**Volumes**  
default-token-lv2bv

- Type: secret (populated by a secret when the pod is created)
- Secret: default-token-lv2bv

[Add Storage to fruits-catalog](#) | [Add Config Files to fruits-catalog](#)

# Deploy the fruits catalog

Test the fruits catalog

Insert fruits in your catalog microservices via the fruits-catalog API

```
$ curl `oc get route/fruits-catalog -o template --template={{.spec.host}}`/api/fruits  
-XPOST -H "Content-Type: application/json" -d '{"name":"Orange", "origin":"Spain"}'  
  
$ curl `oc get route/fruits-catalog -o template --template={{.spec.host}}`/api/fruits  
-XPOST -H "Content-Type: application/json" -d '{"name":"Apple", "origin":"France"}'
```

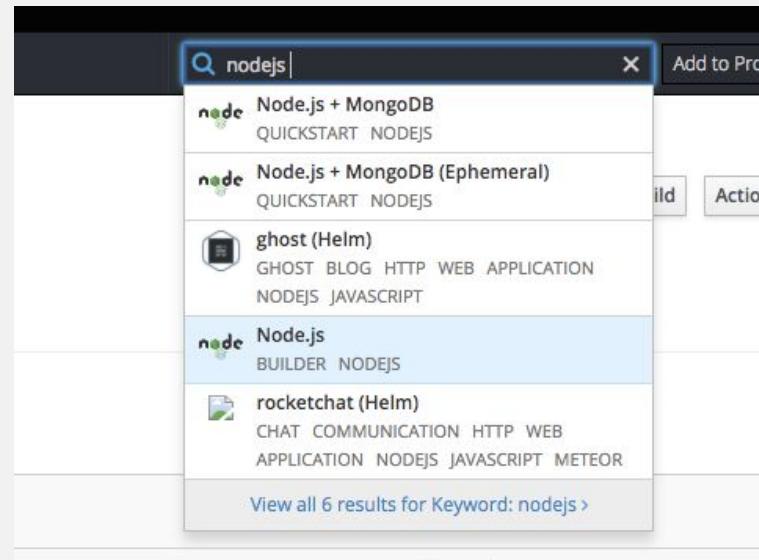
Get all fruits from the fruits-catalog

```
$ curl `oc get route/fruits-catalog -o template --template={{.spec.host}}`/api/fruits -v
```

# Deploy the Fruits inventory

We use the same s2i strategy to build and deploy the app from source code

Choose the Node.JS image builder from the catalog



# Deploy the Fruits inventory

- Name : fruits-inventory
- Context Dir : /fruits-inventory
- Your Git Repo URL



Node.js

Build and run Node.js 8 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see <https://github.com/scrlorg/s2i-nodejs-container>.

Version: 8

\* Name  
fruits-inventory

Identifies the resources created for this application.

\* Git Repository URL  
<https://github.com/lbroudoux/ocp-on-azure-workshop>

Sample repository for nodejs: <https://github.com/openshift/nodejs-ex.git> Try it ↗

Git Reference  
master

Optional branch, tag, or commit.

Context Dir  
/fruits-inventory

Optional subdirectory for the application source code, used as the context directory for the build.

# Deploy the Fruits inventory

Set environment variables to access Redis Cache component already containerized

- REDIS\_HOST - **redis**
- REDIS\_PASSWORD - pick the right secret
- FRUITS\_CATALOG\_HOST - **fruits-catalog**

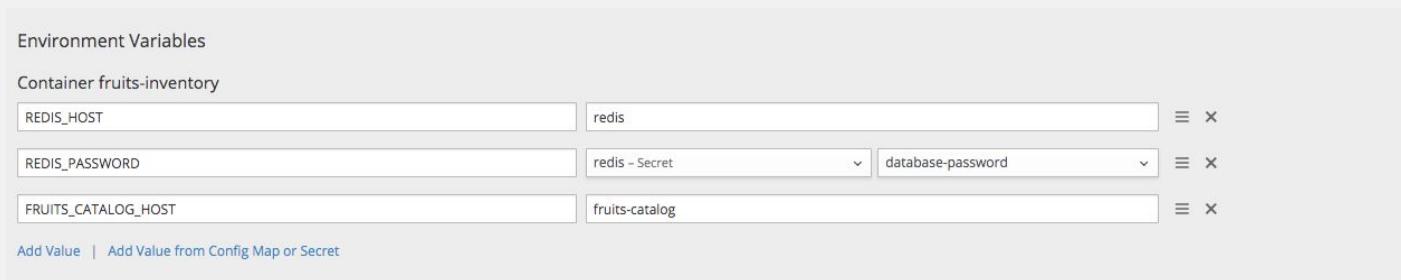
Set environment variables as described

Environment Variables

Container fruits-inventory

REDIS_HOST	redis	≡	x			
REDIS_PASSWORD	redis - Secret	▼	database-password	▼	≡	x
FRUITS_CATALOG_HOST	fruits-catalog	≡	x			

Add Value | Add Value from Config Map or Secret



# Deploy the Fruits inventory

Check that the component works properly with Redis cache

Let's get all fruits in the Grocery Store with their quantity

```
$ curl `oc get route/fruits-inventory -o template --template={{.spec.host}}`/api/fruits  
[{"id": "5c641f4d18909600016320d0", "name": "Orange", "origin": "Spain", "quantity": "1230"}, {"id": "5c64225818909600016320d1", "name": "Apple", "origin": "France", "quantity": "356"}]
```

You can also explore the deployment and the pod resources

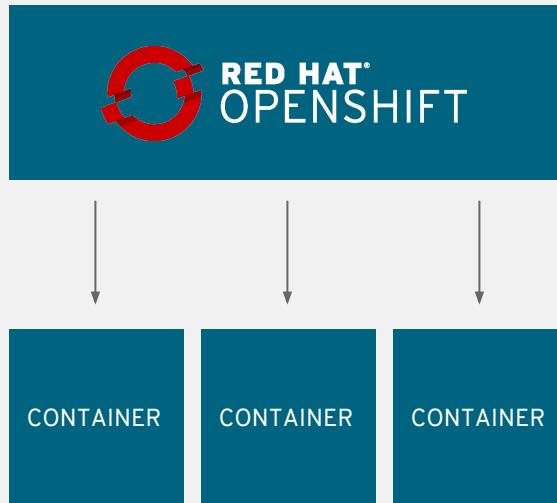
# LAB 4

## Monitoring application health

# LAB 4: Monitoring Application Health

- Review Health endpoints in services
- Add health probes to inventory-service
- Add health probes to shop-ui front-end
- Explore pod metrics

# HEALTH PROBES



## PROBE TYPES

Is it ready?  
Is it alive?

## PROBE CHECKS

HTTP  
Shell Command  
TCP Port

# Health probes

There are two type of health probes available in OpenShift: [liveness probes and readiness probes](#).

*Liveness probes* are to know when to restart a container and *readiness probes* to know when a Container is ready to start accepting traffic.

Health probes also provide crucial benefits when automating deployments with practices like rolling updates in order to remove downtime during deployments. A readiness health probe would signal OpenShift when to switch traffic from the old version of the container to the new version so that the users don't get affected during deployments.

# Add Health check to fruits catalog

We can do it through the web console or the CLI

Set HTTP request to check **readiness**. An endpoint is already defined in the fruits catalog.

Readiness Probe

A readiness probe checks if the container is ready to handle requests. A failed readiness probe means that a container should not receive any traffic from a proxy, even if it's running.

\* Type  
HTTP GET

Use HTTPS

Path  
/actuator/health

\* Port  
8080

Initial Delay  
15 seconds

How long to wait after the container starts before checking its health.

Timeout  
3 seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.

[Remove Readiness Probe](#)



We use business and technical endpoints provided natively by the **actuator** Spring Boot library. This library will be used in others labs ;)

# Add health check to Fruits catalog

## Add the liveness probe

Is the app still running ?  
We use the same endpoint as the readiness for this example.

Liveness Probe

A liveness probe checks if the container is still running. If the liveness probe fails, the container is killed.

Type

HTTP GET

Use HTTPS

Path

/actuator/health

Port

8080

Initial Delay

15 seconds

How long to wait after the container starts before checking its health.

Timeout

3 seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.



# Save and check rolling upgrade strategy

Click **Save** and then click the **Overview** button in the left navigation.

You will notice that `fruits-catalog` pod is getting restarted and it stays light blue for a while. This is a sign that the pod(s) have not yet passed their readiness checks and it turns blue when it's ready!



# Add health checks with the CLI

We set an HTTP Request for both health checks

```
$ oc set probe dc/fruits-inventory --liveness --get-url=http://:8080/api/health/liveness  
--initial-delay-seconds=60 --period-seconds=30  
  
$ oc set probe dc/fruits-inventory --readiness  
--get-url=http://:8080/api/health/readiness
```

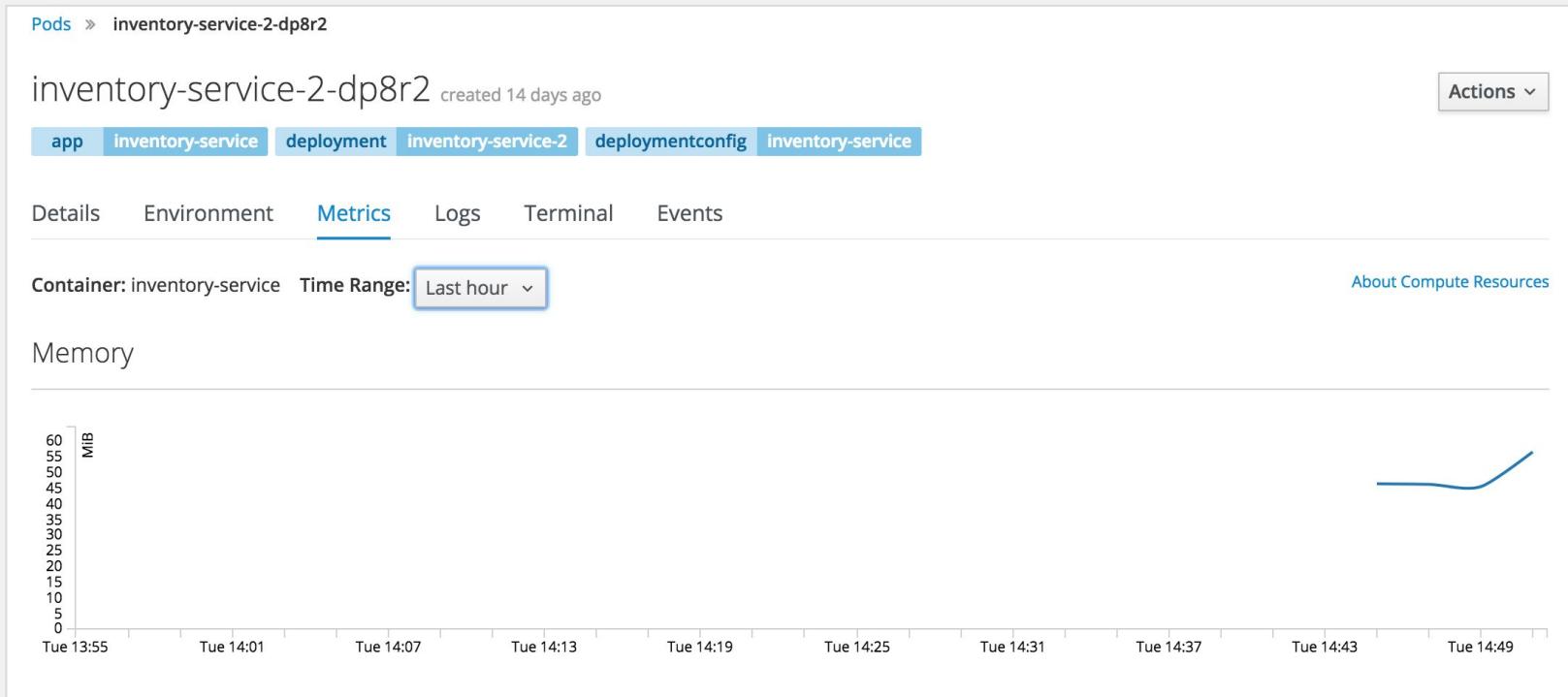
# Monitoring pod metrics

Metrics are another important aspect of monitoring applications which is required in order to gain visibility into how the application behaves and particularly in identifying issues.

OpenShift provides container metrics out-of-the-box and displays how much memory, cpu and network each container has been consuming over time. In the project overview, you can see three charts near each pod that shows the resource consumption by that pod.



# Monitoring pod metrics



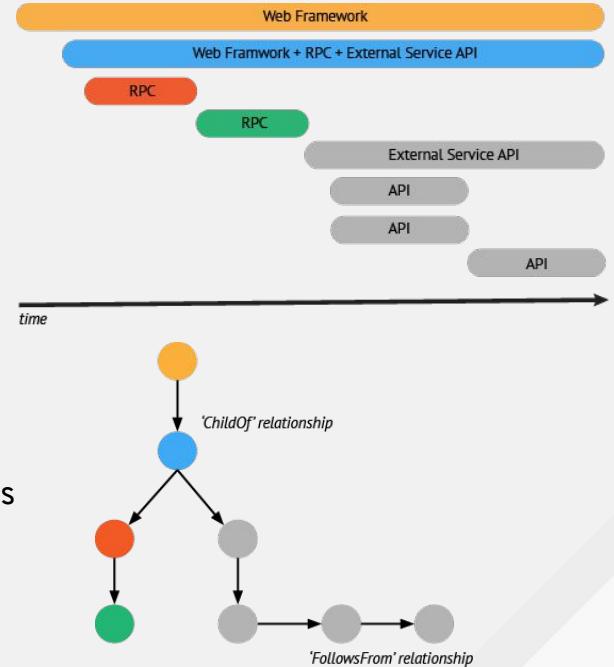
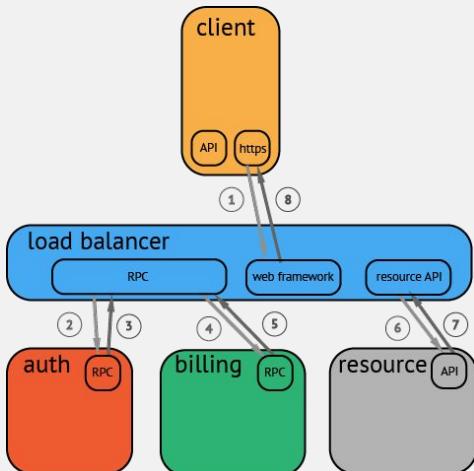
# LAB 5

## Distributed tracing configuration

# LAB 5: Distributed tracing configuration

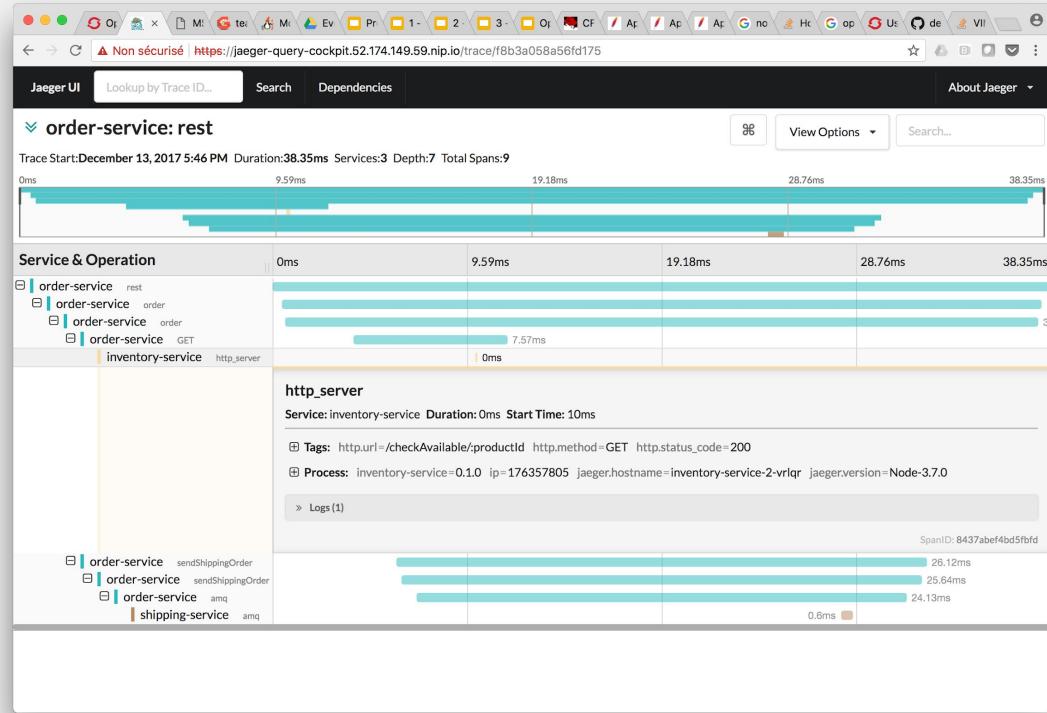
- Externalize and manage application configuration
- Add Jaeger configuration to fruits-catalog
- Explore distributed traces

# What is distributed tracing ?



# Distributed tracing

Jaeger is an OpenTracing implementation and is available in the Cockpit environment.



# Add Jaeger configuration to fruits-catalog

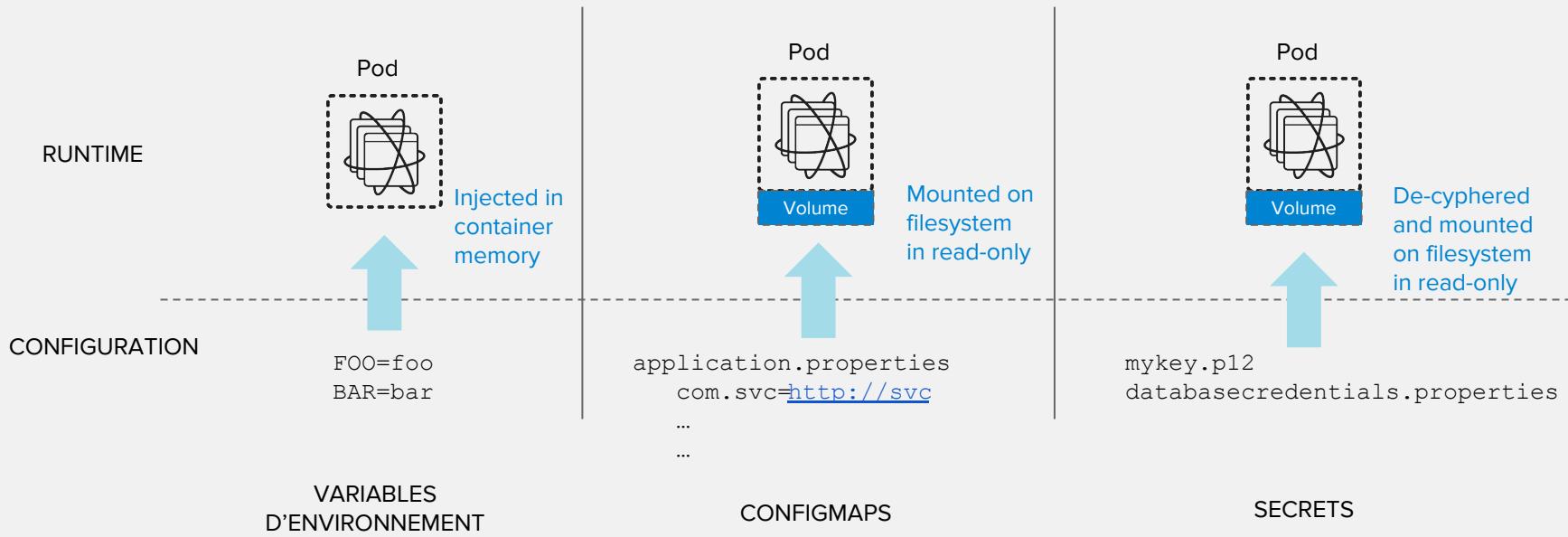
Before setting Jaeger in the fruits-catalog application, we have to add a specific role to the current project to view particular objects, especially ConfigMap ...

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

# ConfigMap in OpenShift

- Config maps inject config data into containers
- Config maps can hold
  - Properties (key-value pairs)
  - Files (JSON, XML, etc)
- Containers see config maps as
  - Files on the filesystem
  - Environment variables
- Secrets are like config maps for sensitive data
  - Credentials, certificates, SSH keys, etc

# Configuration management



# Add Jaeger configuration to fruits-catalog

Create a configMap with the CLI

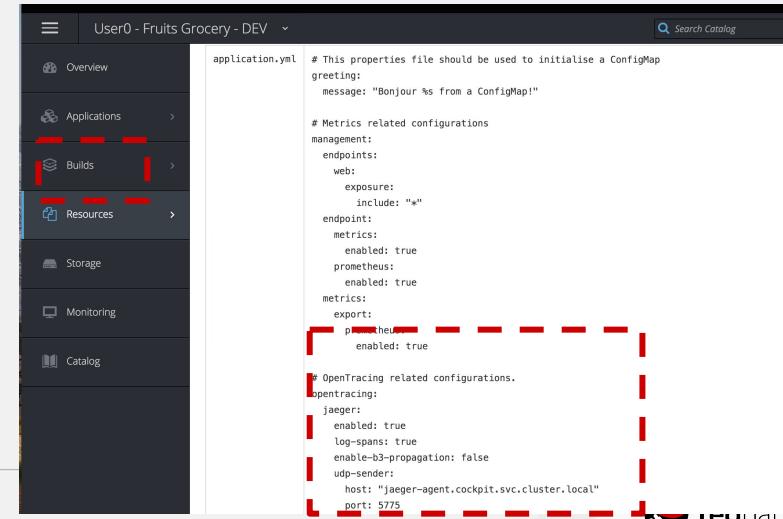
```
$ cd fruits-catalog  
$ oc create configmap fruits-catalog-config --from-file=application.yml
```

Edit ConfigMap ( Actions > Edit Yaml ) created and set Jaeger host as :

jaeger-agent.cockpit.svc.cluster.local

Click Add to Application

Now pod is redeploying



# Add Jaeger host to fruits-inventory

A Jaeger tracer is already set for all invocations in fruits-inventory.  
We set the Jaeger host as environment variable

```
$ oc set env dc/fruits-inventory JAEGER_HOST=jaeger-agent.cockpit.svc.cluster.local
```

A new deployment is created.

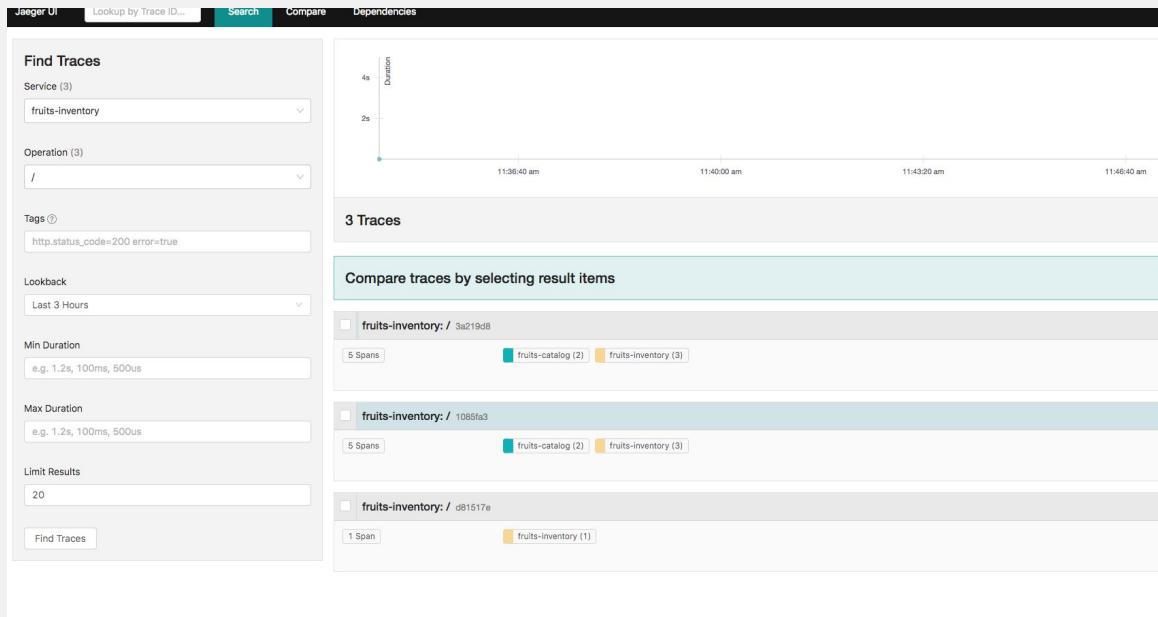
Get all fruits with their stock through the fruits-inventory API and jump to Jaeger to see the detailed trace

```
$ curl `oc get route/fruits-catalog -o template --template={{.spec.host}}`/api/fruits -v
```

# Explore Distributed Traces with Jaeger

Go to Jaeger console via  
<https://bit.ly/2BKWuTN>

Jaeger is deployed on Openshift in  
an other project name cockpit.



# Filter the right Jaeger trace

As we use a mutual Jaeger, you need to filter on your pod fruits-catalog hostname

```
$ oc get pods -l app=fruits-catalog
NAME           READY   STATUS    RESTARTS   AGE
fruits-catalog-4-4phqn   1/1     Running   0          40m
```

The image shows the 'Find Traces' interface of the Jaeger UI. It has several input fields:

- Service (3):** fruits-catalog
- Operation (4):** find
- Tags (1):** hostname=fruits-catalog-4-4phqn
- Lookback:** Last Hour
- Min Duration:** e.g. 1.2s, 100ms, 500us
- Max Duration:** e.g. 1.2s, 100ms, 500us
- Limit Results:** 20

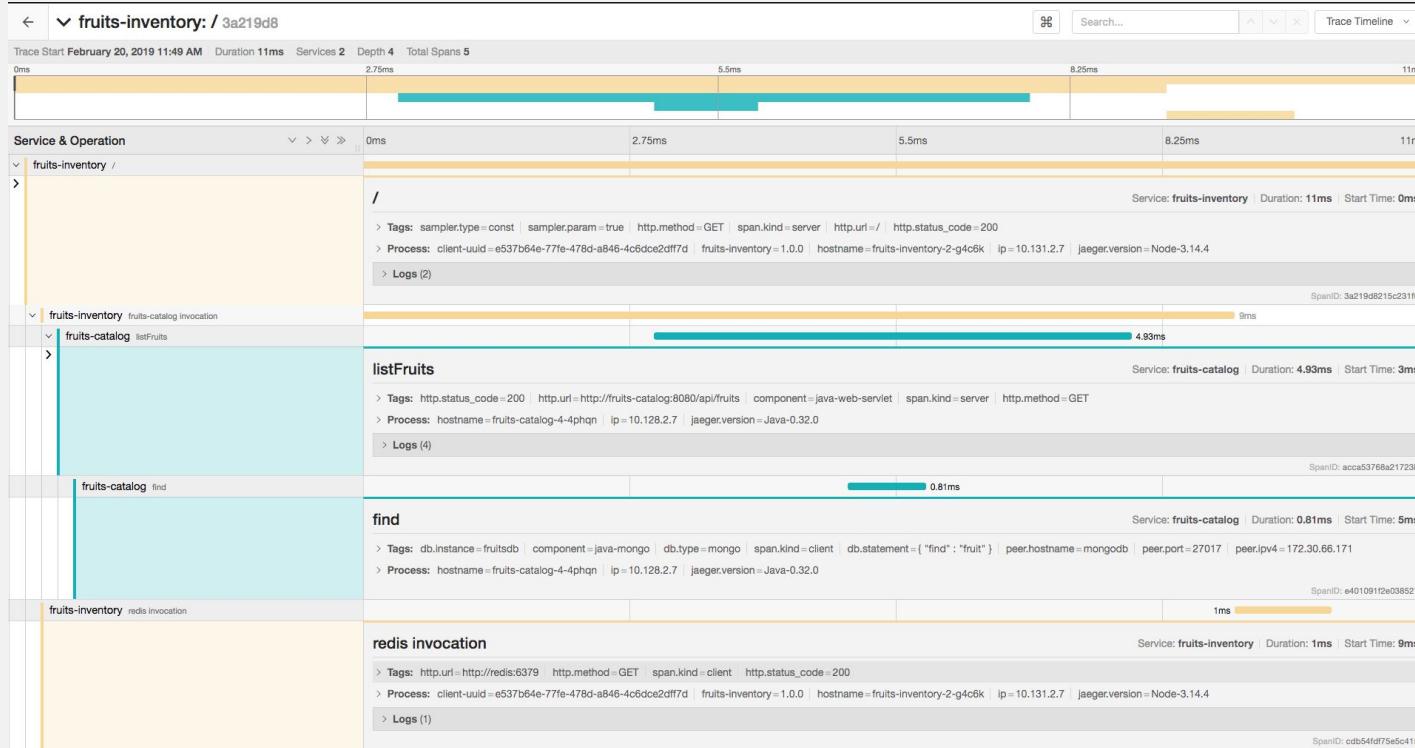
A red dashed box highlights the 'Tags' field, and another red dashed box highlights the 'Find Traces' button at the bottom.

Following filter criterias are :

- Services : fruits-inventory
- Operation: /
- Tags: hostname=fruits-catalog-4-4phqn

Click on Find Traces

# Explore the Jaeger trace



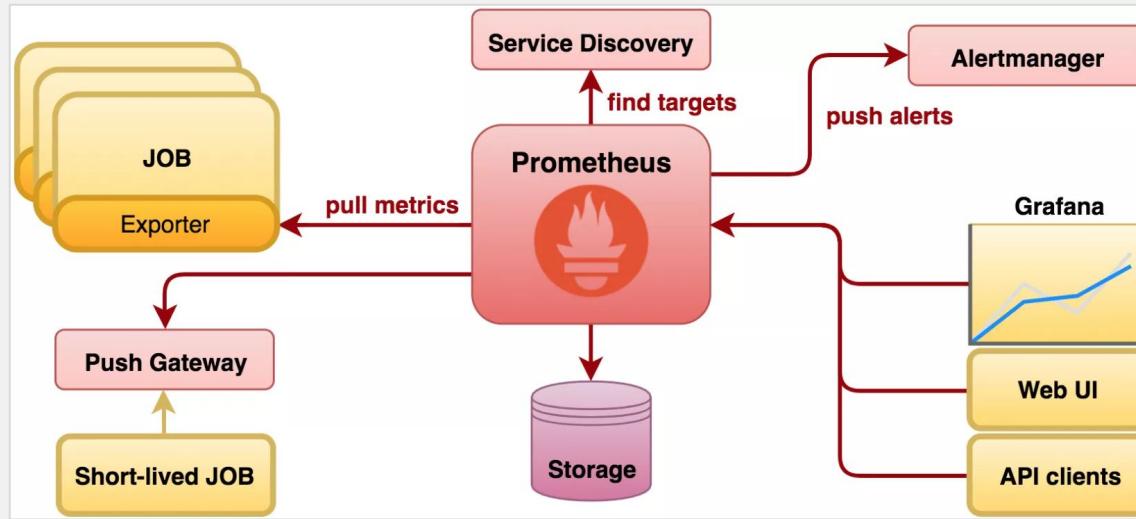
# LAB 6

## Getting application metrics

# LAB 6: GETTING APPLICATION METRICS

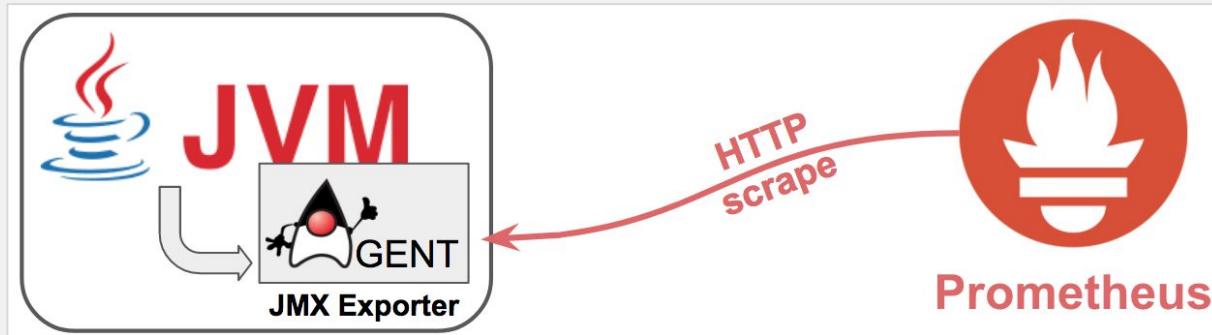
- Update Prometheus configuration
- Add Prometheus datasource in Grafana

# Prometheus monitoring



OpenShift now provides Prometheus templates for automated deployment. One instance is available into a `cockpit` project. A Grafana instance on same project.

# Prometheus monitoring



For a quick run, we'll use JMX Exporter Prometheus Agent that expose JMX metrics as Prometheus endpoints. This is already configured into fruits-catalog thanks to actuator library.  
A middleware Prometheus is added in fruits-inventory

# Check Prometheus metrics in deployed pods

Access to the pod terminal with oc rsh command

```
$ oc rsh dc/fruits-catalog # Now logging in fruits-catalog pod
$ curl http://localhost:8080/actuator/prometheus
... # TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="direct",} 82807.0
jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
...
$ curl http://localhost:8080/actuator/metrics # display metrics available
{"names": ["jvm.memory.max", "jvm.threads.states", "process.files.max",
"jvm.gc.memory.promoted" ...
```

# Check Prometheus console now ...

Go to Prometheus console : <https://prometheus-cockpit.52.143.158.219.nip.io/> in the target menu

Nothing is sent by fruits-catalog and fruits-inventory Prometheus console !

Prometheus scraps by default /metrics endpoint on port 9900. Our 2 back-ends expose a different Prometheus endpoint.

We need to annotate our application Kubernetes services to be discovered by Prometheus

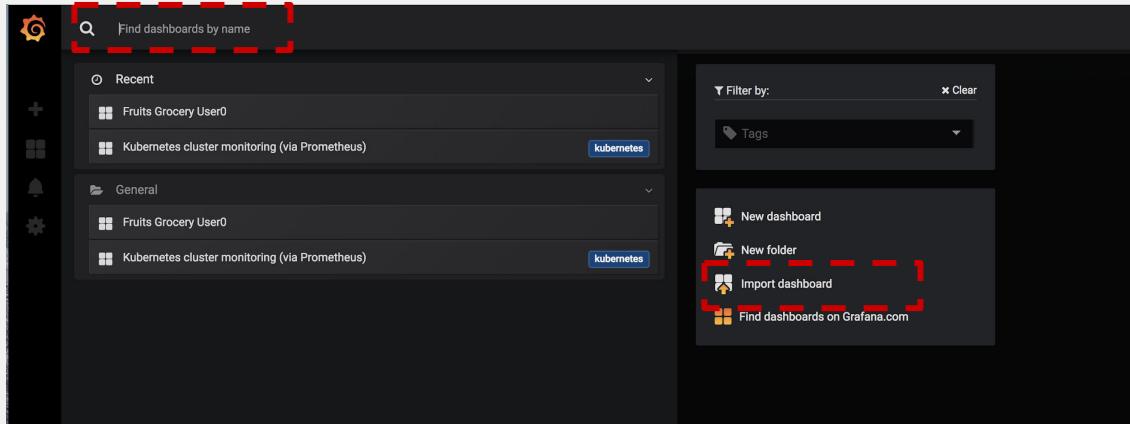
```
$ oc annotate service/fruits-catalog prometheus.io/scrape=true  
prometheus.io/path=/actuator/prometheus prometheus.io/port=8080
```

```
$ oc annotate service/fruits-inventory prometheus.io/scrape=true prometheus.io/port=8080
```

# Import Grafana Dashboard

Grafana URL : <https://grafana-cockpit.52.143.158.219.nip.io/>

Click on New Dashboard and Import Dashboard



Copy and paste the following json:

<https://raw.githubusercontent.com/lbroudoux/ocp-on-azure-workshop/master/grafana-dashboard-user0.json>

# Import Grafana Dashboard

Change Dashboard name with your user ID

Options

Name  ✓

Folder

Unique Identifier (uid)  change

prometheus

 Import

Cancel

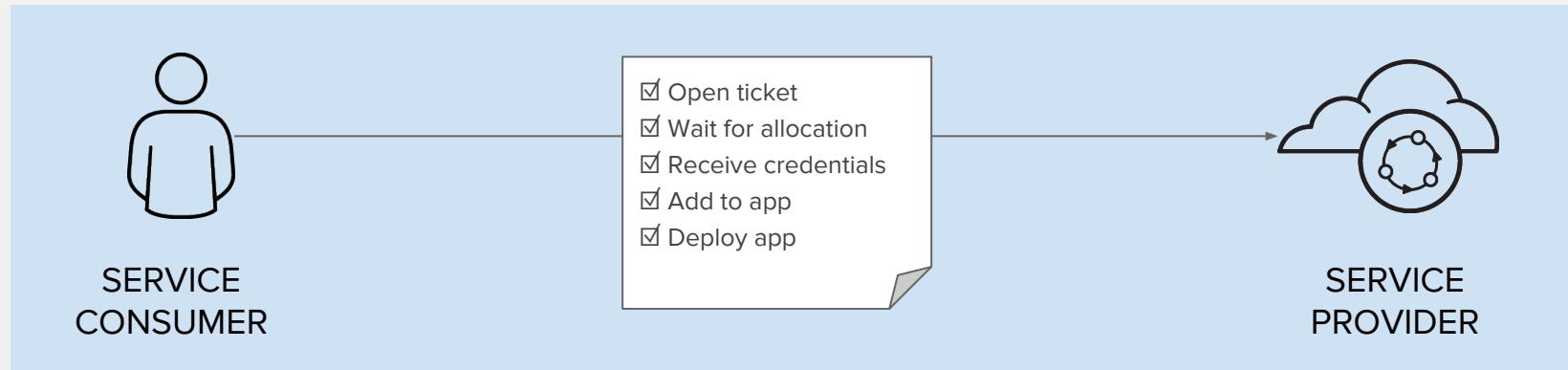
# Grafana Dashboard example



# LAB 7

## Azure Service Broker

# Why a service broker ?



Manual, Time-consuming and Inconsistent



OPEN SERVICE BROKER API™

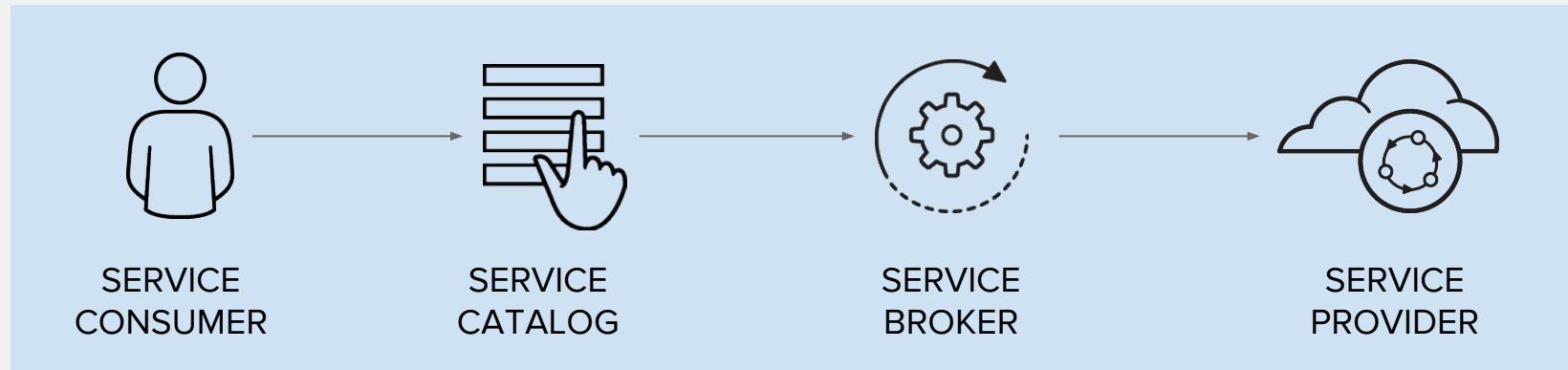
A multi-vendor project to standardize how services are consumed on cloud-native platforms across service providers

FUJITSU Pivotal.

IBM redhat.

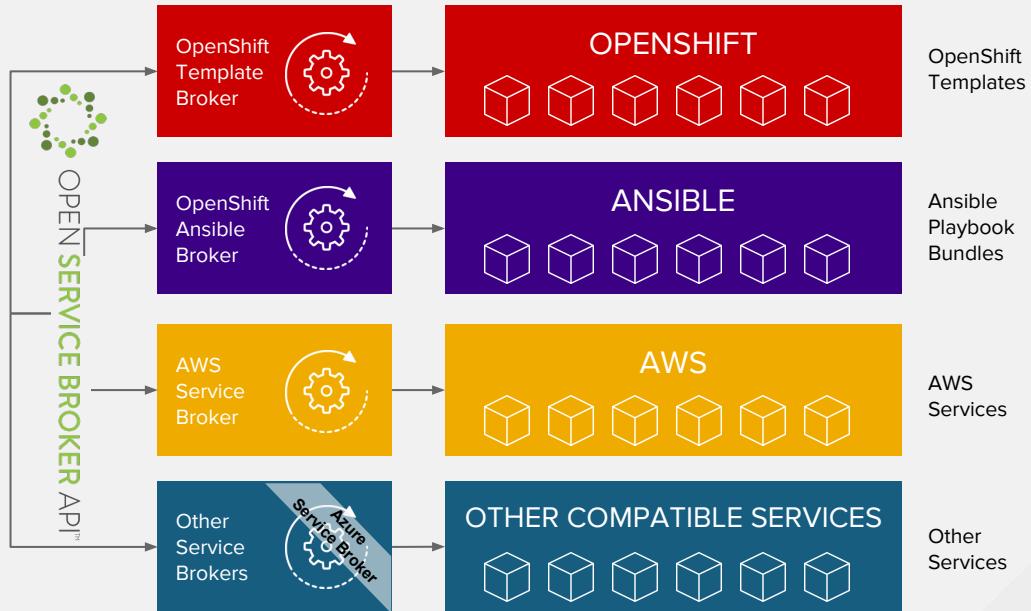
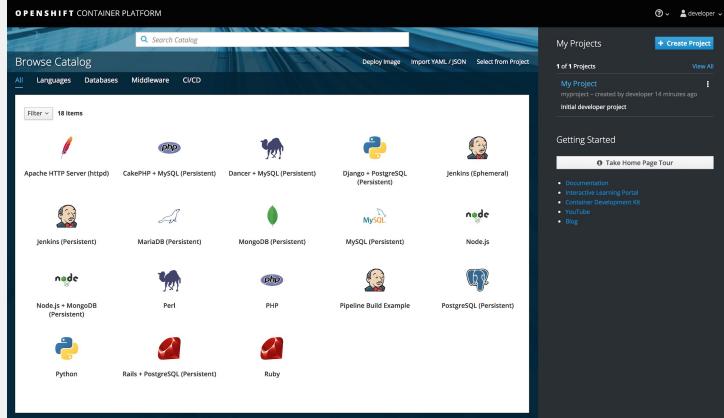
Google SAP®

# What is a service broker ?



Automated, Standard and Consistent

# OpenShift service catalog

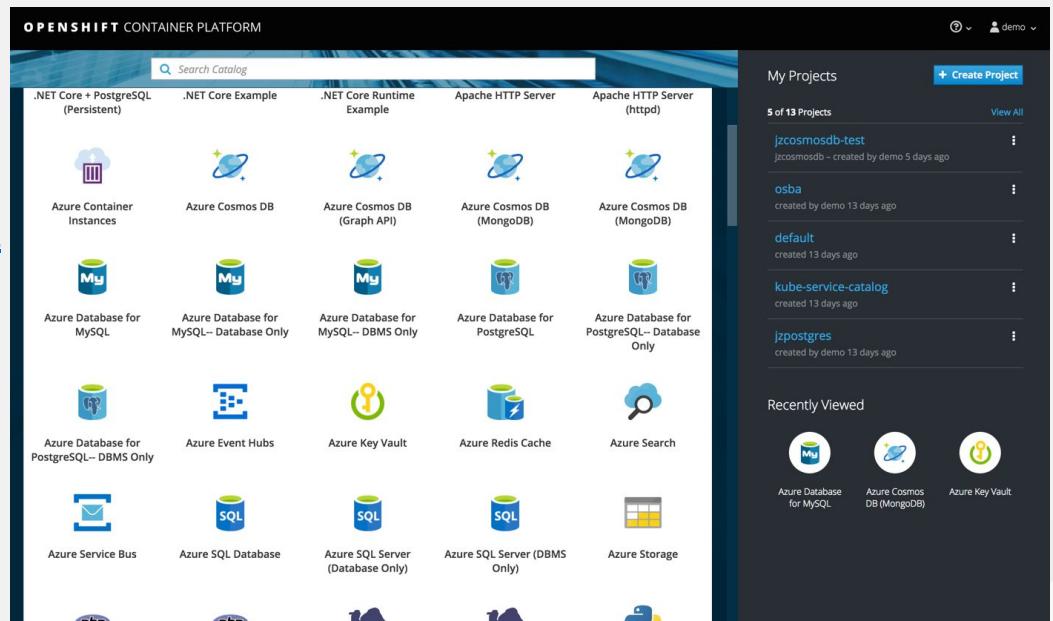


# OPEN SERVICE BROKER AZURE

<https://github.com/Azure/open-service-broker-azure>

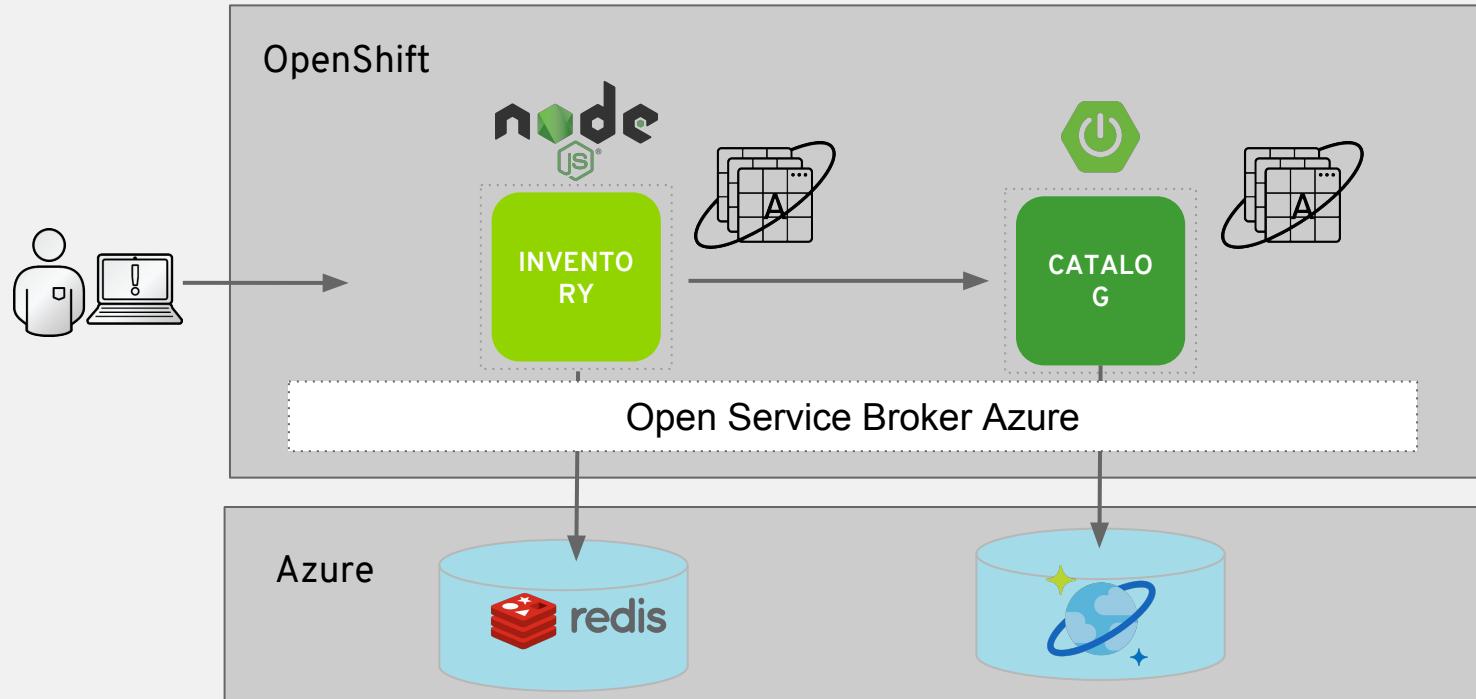
## Supported services

- [Azure Container Instances](#)
- [Azure CosmosDB](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)
- [Azure Event Hubs](#)
- [Azure Key Vault](#)
- [Azure Redis Cache](#)
- [Azure SQL Database](#)
- [Azure Search](#)
- [Azure Service Bus](#)
- [Azure Storage](#)





# Grocery Store on OpenShift and Azure



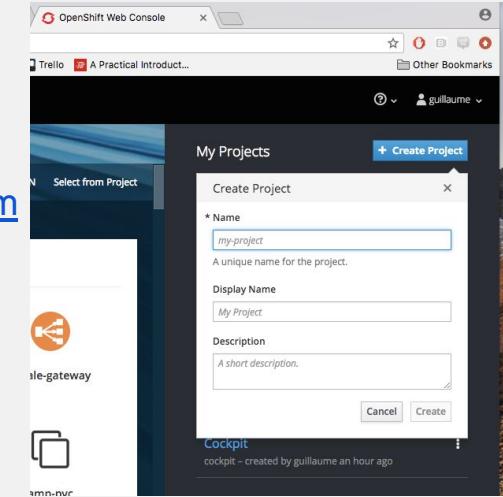
# Create your production environment

Let's go the Web Console

- Via the web console :

<https://masterdnsccbmvtdzhvuqye.francecentral.cloudapp.azure.com>

- Login with the same credentials
- Create a Project with the following informations
  - Name : **fruits-grocery-prod-userX**
  - Display Name: **UserX - Fruits Grocery - Prod**



# Deploy a Redis Cache instance with Open Service Broker Azure

OPENSHIFT CONTAINER PLATFORM

Search Catalog

Filter ▾ 127 Items

.NET .NET .NET .NET 3scale-gateway amp-apicast-wildcard-router

.NET Core + PostgreSQL (Persistent) .NET Core Builder Images .NET Core Example .NET Core Runtime Example

amp-pvc Apache HTTP Server Apache HTTP Server (httpd) Azure Container Instances

Azure Cosmos DB (Graph API) Azure Cosmos DB (MongoDB API)

Azure Cosmos DB (SQL API) Azure Cosmos DB (Table API) Azure Database for MySQL

Azure Database for MySQL- Database Only Azure Database for MySQL-- DBMS Only Azure Database for PostgreSQL

Azure Database for PostgreSQL-- Database Only Azure Database for PostgreSQL-- DBMS Only Azure Event Hubs

Azure Key Vault Azure Redis Cache Azure Search

Azure Service Bus Azure SQL Database Azure SQL Server (Database Only) Azure SQL Server (DBMS Only) Azure Storage CakePHP + MySQL

My Projects

+ Create Project

5 of 20 Projects View All

- gitlab created by maxime a day ago
- default created 2 months ago
- apb-tasks-build created by maxime 8 days ago
- apb-tasks-prod created by maxime 8 days ago
- apb-tasks-test created by maxime 8 days ago

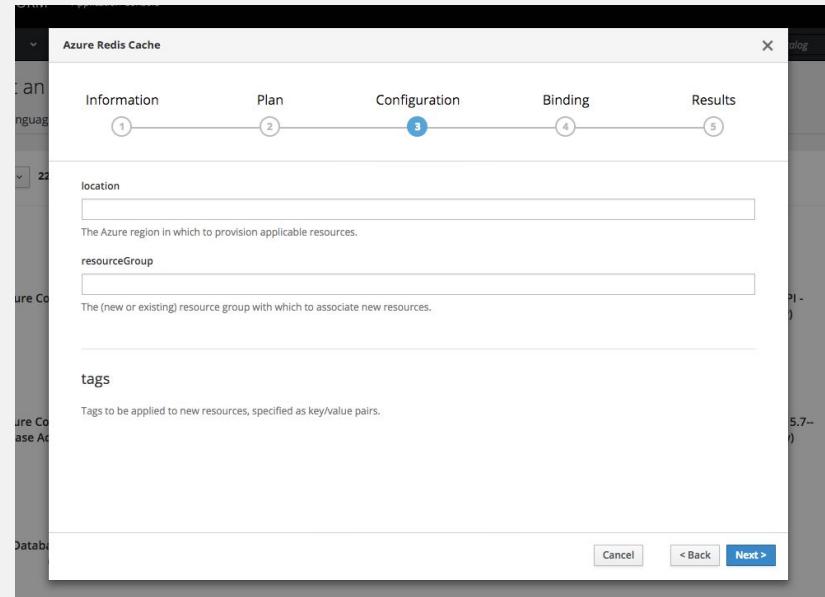
Recently Viewed

- gitlab-runner
- JBoss EAP 7.0 (no https)
- Azure Database for PostgreSQL
- Node.js

# Deploy Redis Cache DB with OSBA

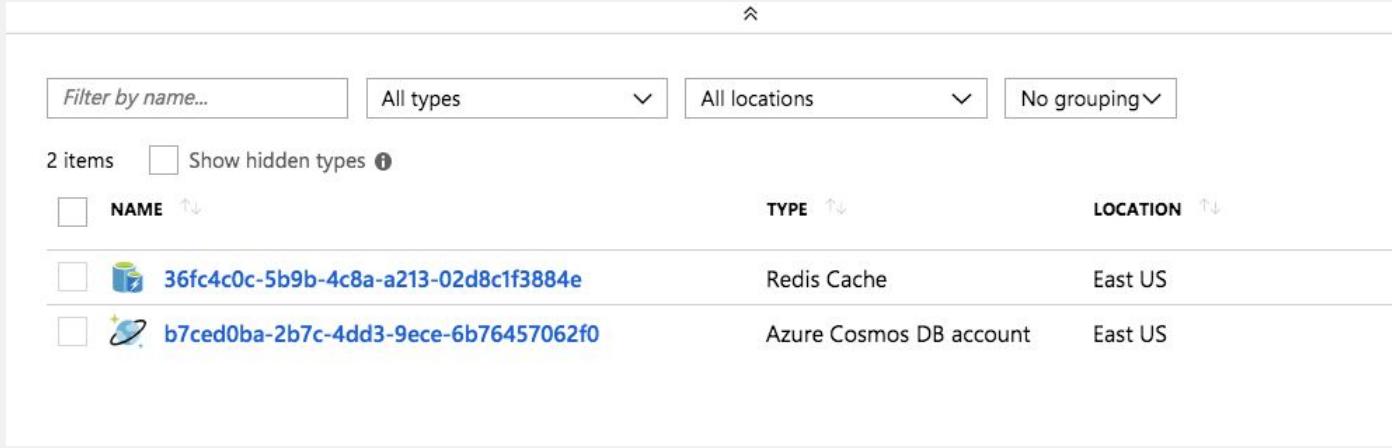
Complete the following settings

- Select a Plan
  - Basic Tier
- Configuration
  - location : **eastus**
  - resourceGroup: **osba**
- Bindings
  - Don't bind to secrets. We will do it Manually :)



# Let's go to the backstage

A Redis Cache instance has been provisioned in Azure through the Azure Service Broker



The screenshot shows a list of service instances in the Azure portal. At the top, there are filters for 'Filter by name...', 'All types', 'All locations', and 'No grouping'. Below the filters, it says '2 items' and has a checkbox for 'Show hidden types'. The table has columns for NAME, TYPE, and LOCATION. There are two entries:

NAME	TYPE	LOCATION
<a href="#">36fc4c0c-5b9b-4c8a-a213-02d8c1f3884e</a>	Redis Cache	East US
<a href="#">b7ced0ba-2b7c-4dd3-9ece-6b76457062f0</a>	Azure Cosmos DB account	East US

# Deploy a Cosmo DB instance with OSBA

The screenshot shows the OpenShift Container Platform Catalog interface. On the left, there's a search bar labeled "Search Catalog" and a filter dropdown set to "127 Items". Below the search bar, there are several categories of services:

- .NET**: .NET Core + PostgreSQL (Persistent), .NET Core Builder Images, .NET Core Example, .NET Core Runtime Example.
- Cloud Services**: 3scale-gateway, amp-apicast-wildcard-router.
- Azure Services**: amp-pvc, Apache HTTP Server, Apache HTTP Server (https), Azure Container Instances, Azure Cosmos DB (Graph API), Azure Cosmos DB (MongoDB API).
- Database Services**: Azure Cosmos DB (SQL API), Azure Cosmos DB (Table API), Azure Database for MySQL, Azure Database for MySQL - Database Only, Azure Database for MySQL - DBMS Only, Azure Database for PostgreSQL.
- General Services**: Azure Event Hubs, Azure Key Vault, Azure Redis Cache, Azure Search.
- Platform Services**: Azure Service Bus, Azure SQL Database, Azure SQL Server (Database Only), Azure SQL Server (DBMS Only), Azure Storage, CakePHP + MySQL.

On the right side of the interface, there's a sidebar titled "My Projects" which lists 5 of 20 projects, including "gitlab", "default", "apb-tasks-build", "apb-tasks-prod", and "apb-tasks-test". Below the projects, there's a "Recently Viewed" section with icons for "git", "JBoss EAP 7.0 (no https)", "Azure Database for PostgreSQL", and "Node.js".

# Deploy a Cosmo DB with OSBA

Complete the following settings

## Configuration

- defaultConsistencyLevel = Session
- allowedIPRanges = 0.0.0.0/0 . Then click Add and then click the X
- Location : eastus
- resourceGroup : osba

## Binds:

- Add secrets bindings
- Service Broker will retrieve credentials CosmoDB instance from Azure

# Our two services provisioned !

**Redis** and **CosmoDB** are provisioned asynchronously in Azure via the Open Service Broker.

You can consume both services through OpenShift via the binding mechanism.

*\*Due to OSBA implementation  
Redis stays in Pending status.*

The screenshot shows the 'Provisioned Services' section in the Azure portal. It lists two services: 'Azure Cosmos DB (MongoDB API)' and 'Azure Redis Cache'. The 'Azure Cosmos DB' service has a single binding named 'azure-cosmosdb-mongo-account-wnqrj-r7g6h' created 4 hours ago. The 'Azure Redis Cache' service has a binding named 'azure-rediscache-clr54-kmdk8' created an hour ago, with a note indicating it is 'Pending' status due to OSBA implementation.

Service	Binding	Status
Azure Cosmos DB (MongoDB API)	azure-cosmosdb-mongo-account-wnqrj-r7g6h	Created 4 hours ago
Azure Redis Cache	azure-rediscache-clr54-kmdk8	Pending

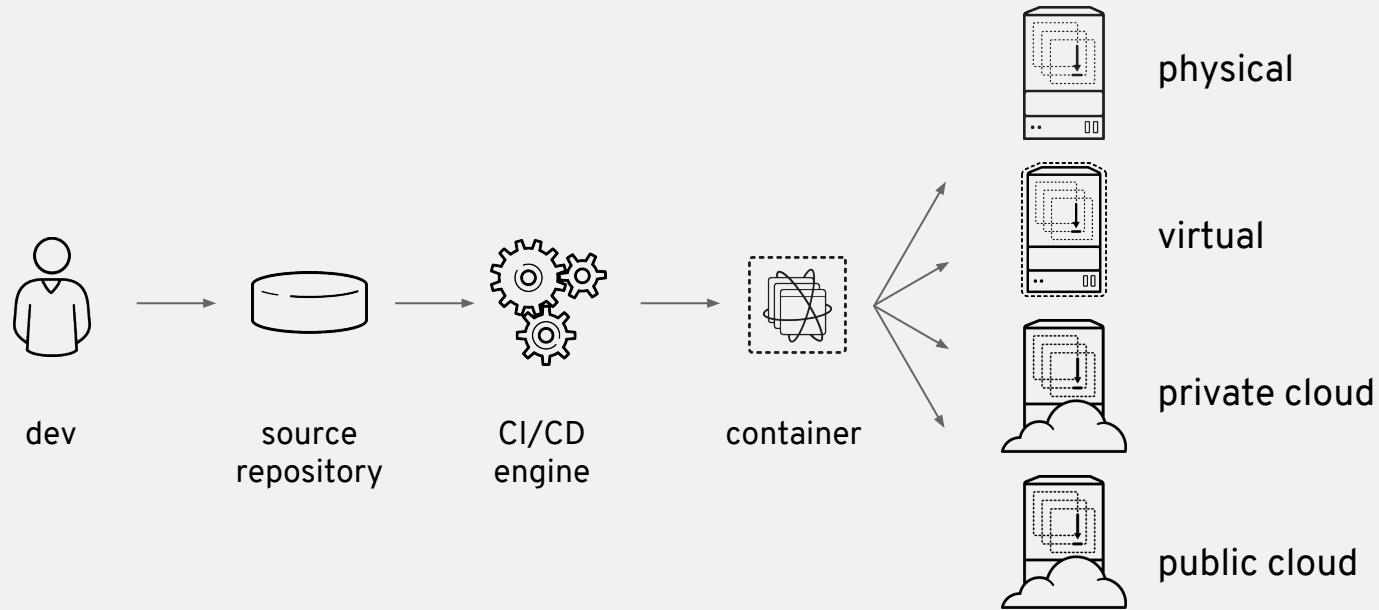
# LAB 8

## Continuous delivery

# LAB 8: Automating Deployments Using Tags and Pipelines

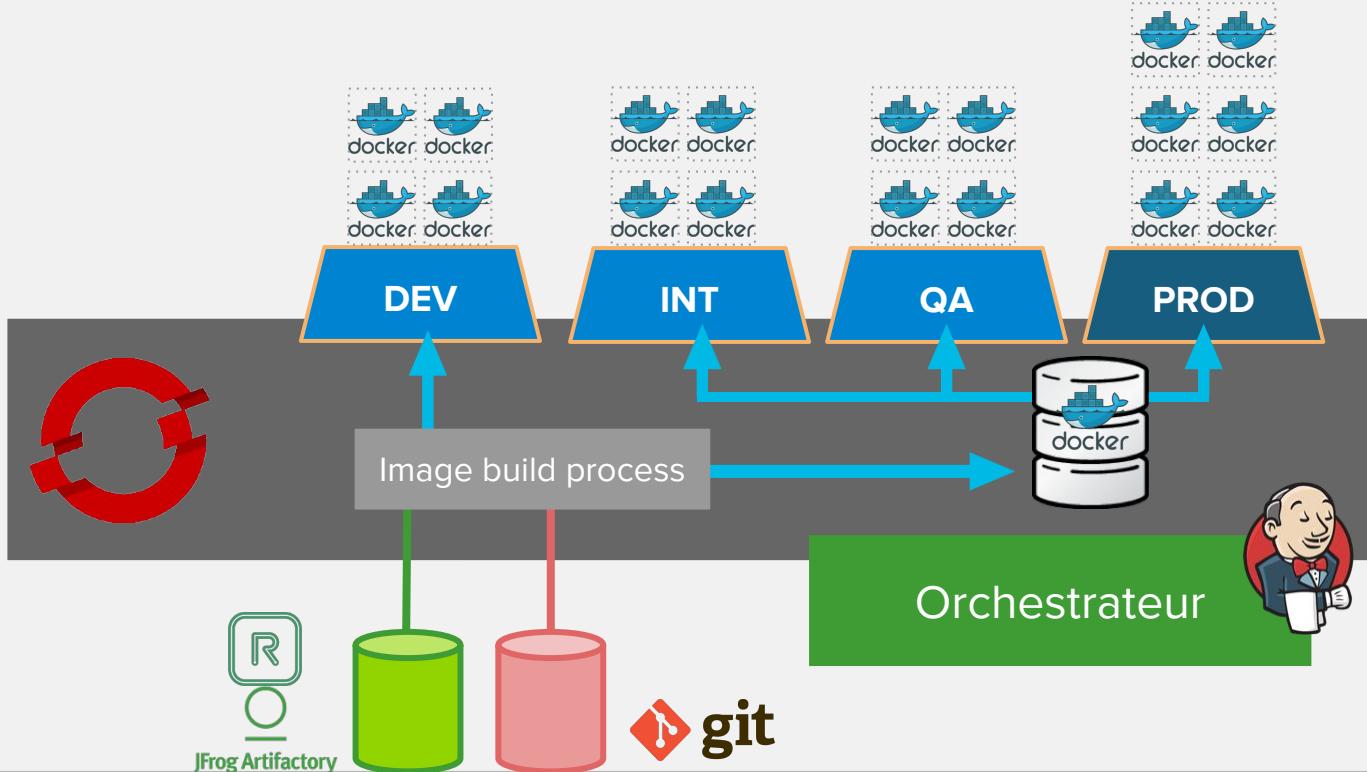
- Prepare a Production environment
- Explore the deployment configurations
- Promote images to production
- Create an OpenShift Jenkins Pipeline
- Add a Webhook to run the pipeline on every code change
- Change some code and review

# Deployment pipeline



Rolling Upgrades  
Blue/Green Deployments  
A/B Testing

# CI/CD with OpenShift



# OpenShift Pipelines

- CI/CD workflow via Jenkins
- Pipelines are started, monitored, and managed similar to other builds
- Auto-provisioning of Jenkins server
- On-demand Jenkins slaves
- Embedded Jenkinsfile or in Git repo

```
pipeline {
    agent {
        label 'maven'
    }
    stages {
        stage('build app') {
            steps {
                git url: 'https://git/app.git'
                sh "mvn package"
            }
        }
        stage('build image') {
            steps {
                script {
                    openshift.withCluster() {
                        openshift.startBuild("...")
                    }
                }
            }
        }
    }
}
```

# Create Redis Cache secrets

- Go back to the spreadsheet : <https://bit.ly/2TWsI5D>
- Update REDIS\_HOST and REDIS\_PASSWORD environment variables from prepare\_prod.sh file with the values from the spreadsheet

```
$ vi /home/userX/prepare-prod.sh
```

```
2 | export REDIS_HOST=36fc4c0c-5b9b-4c8a-a213-02d8c1f3884e.redis.cache.windows.net
3 | export REDIS_PASSWORD=nusoAxF3Ae+RHvkhhKMxruPpwnO+A6Xn5rkLMaSlkmw=
```

User	Password	REDIS_HOST	REDIS_PASSWORD		
user0	P@ssword-User0	36fc4c0c-5b9b-4c8a-a213-02d8c1f3884e.redis.cache.windows.net	nusoAxF3Ae+RHvkhhKMxruPpwnO+A6Xn5rkLMaSlkmw=		
--	--	--	--	--	--

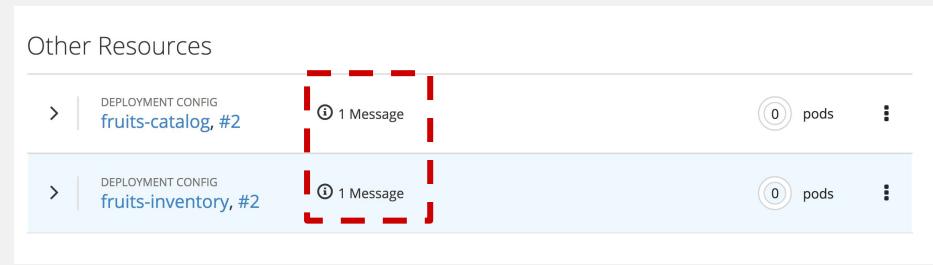
# Prepare a Production environment

A wrap-up script has been prepared for you. It will contains all resources created previously in the Development project.

```
$ ./prepare-prod.sh
```

# Explore the deployment configurations

From overview on web console,  
check the deployment  
configuration All deployment are  
cancelled.



# Explore the deployment configurations

Clicking on a deployment configuration, you should see that there's no automatic trigger defined for deployment.

You shall also notice that the image used for deployment is coming from your development project !

The screenshot shows the 'DEPLOYMENT CONFIG' page for the 'fruits-catalog' deployment configuration. At the top, a message states: 'fruits-catalog is paused. This will stop any new rollouts or triggers from running until resumed. [Resume Rollouts](#)'.

**CONTAINERS**: A single container named 'default-container' is listed, using the image 'fruits-grocery-dev-user0/fruits-catalog'. A red dashed box highlights this container entry.

**NETWORKING**: It shows a service named 'fruits-catalog' with internal traffic on port 8080/TCP and external traffic on port 8080, both mapped to target port 8080. A route is also listed: 'http://fruits-catalog-fruits-grocery-prod-user0.apps.openhybridcloud.io'.

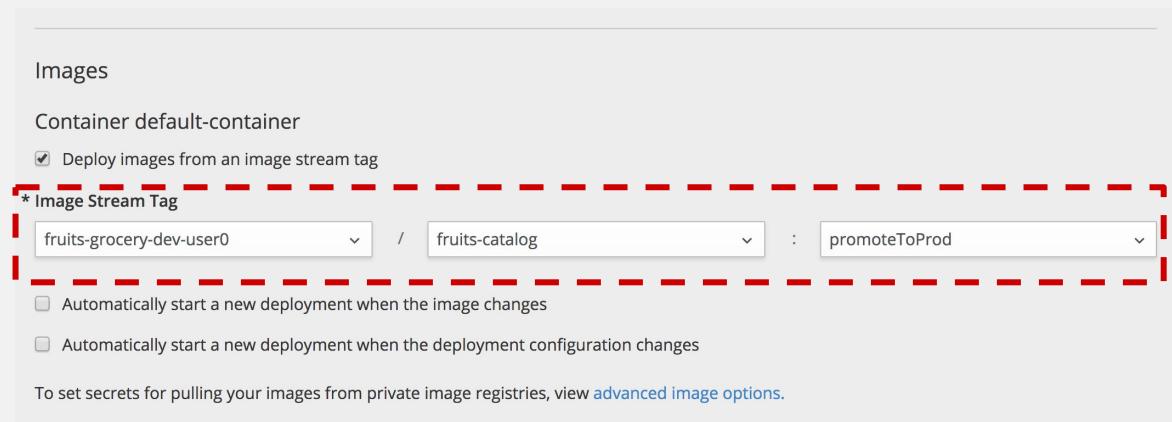
To the right, a circular icon indicates '0 pods'.

# Explore the deployment configurations

Access detailed configuration by choosing **Edit** in **Actions** menu.

Check that the image referenced into your dev project has the `:promoteToProd` tag.

Because this tag does not exists, deployment will fail !



# Promote images to production

The wrap-up script can be used again here through a new command. The command will tag all images from development streams and rollout all the deployments.

```
$ . /home/userX/deploy-prod.sh

Tag fruits-grocery-dev-user0/fruits-catalog:promoteToProd set to
fruits-grocery-dev-user0/fruits-catalog@sha256:5eeb089a5df9aa55b4e80c581014a674c1e2f7e902c92a3f5c48e0df4155e95
7.
Tag fruits-grocery-dev-user0/fruits-inventory:promoteToProd set to
fruits-grocery-dev-user0/fruits-inventory@sha256:29a17627c330a5568f6a956ffddc5f7c3e17ab4839e22085899b7eb0328
9705a.
deploymentconfig "fruits-catalog" rolled out
deploymentconfig "fruits-inventory" rolled out
```

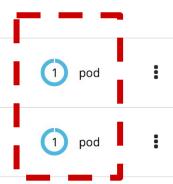
# Promote images to production

Check deployment are successful !

But wait ... we have also created a pipeline. Just go to your development project.

Other Resources

- > DEPLOYMENT CONFIG fruits-catalog, #4
- > DEPLOYMENT CONFIG fruits-inventory, #4



The diagram shows a dashed red square containing two blue circles, each with the number '1' and the word 'pod' next to it, representing two pods in a deployment.

# Create an OpenShift Jenkins Pipeline

In your development project within the **Builds** section, **Pipelines** subsection, check that `inventory-service-pipeline` has been created.

Triggers with webhooks provide a full developer experience to automate its deployment from a local env to production environment

The screenshot shows the Jenkins Pipeline configuration page for a pipeline named "fruits-inventory-pipeline". The pipeline was created 3 days ago. The "Configuration" tab is selected. The "Details" section displays the Jenkinsfile content:

```
node('nodejs') {
    stage('Build') {
        openshiftBuild(namespace: 'fruits-grocery-dev-user0', bldCfg: '')
    }
    stage('Deploy Dev') {
        openshiftDeploy(namespace: 'fruits-grocery-dev-user0', depCfg: '')
    }
    stage('Acceptance Tests') {
        sleep 13
    }
    stage('Promote to Prod') {
        openshiftTag(namespace: 'fruits-grocery-dev-user0', sourceStr: '')
    }
    stage('Deploy Prod') {
        openshiftDeploy(namespace: 'fruits-grocery-dev-user0', depCfg: '')
    }
}
```

The "Triggers" section contains three entries:

- Generic Webhook URL: <https://guillaume.openhybridcloud.io:8443/api/jenkins-webhook>
- GitHub Webhook URL: <https://guillaume.openhybridcloud.io:8443/api/jenkins-webhook>
- Manual (CLI): `oc start-build fruits-inventory-pipeline -n fruits`

# Start your Jenkins pipeline

We deploy the fruits-inventory application from Dev to Prod with complex tests ...

Pipelines [Learn More ↗](#)

**fruits-inventory-pipeline** created 20 hours ago [Start Pipeline](#) Average Duration: 3m 41s

Recent Runs

	Build	Deploy Dev	Acceptance Tests	Promote to Prod	Deploy Prod
<a href="#">Build #1</a> 7 minutes ago <a href="#">View Log</a>	59s	22s	13s	0s	30s

[View Pipeline Runs](#) | [Edit Pipeline](#)

Check Jenkins pipeline job logs via the Jenkins console. Click on “View Log”



**CONGRATULATIONS !**

**YOU'RE A CLOUD-NATIVE APPS**

**DEVELOPER.**

# LEARN.OPENSHIFT.COM

The screenshot shows a web browser window for the 'OpenShift: Interactive Learning' page at <https://learn.openshift.com/introduction/>. The page features a dark background with a grid of white squares. At the top, there's a navigation bar with the OpenShift logo, a 'Report an Issue' button, and a 'SIGN UP TO OPENSFIFT ONLINE FOR FREE' button. The main heading is 'Interactive Learning Portal'. Below it, a sub-headline reads: 'Our Interactive Learning Scenarios provide you with a pre-configured OpenShift instance, accessible from your browser without any downloads or configuration. Use it to experiment, learn OpenShift and see how we can help solve real-world problems.' Six 'START SCENARIO' buttons are arranged in two rows of three, each linking to a different learning module.

Getting Started with OpenShift for Developers	Logging in to an OpenShift Cluster	Deploying Applications From Images
START SCENARIO	START SCENARIO	START SCENARIO
Deploying Applications From Source	Using the CLI to Manage Resource Objects	Connecting to a Database Using Port Forwarding
START SCENARIO	START SCENARIO	START SCENARIO



# THANK YOU



[plus.google.com/+RedHat](https://plus.google.com/+RedHat)



[facebook.com/redhatinc](https://facebook.com/redhatinc)



[linkedin.com/company/red-hat](https://linkedin.com/company/red-hat)



[twitter.com/RedHatNews](https://twitter.com/RedHatNews)



[youtube.com/user/RedHatVideos](https://youtube.com/user/RedHatVideos)