

# OPENSIFT 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

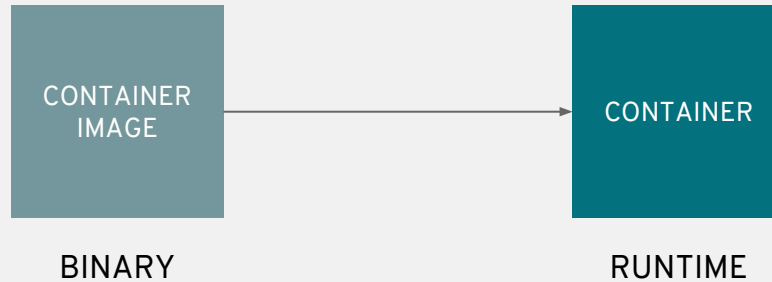
# OPENSIFT CONCEPTS OVERVIEW

A container is the smallest compute unit

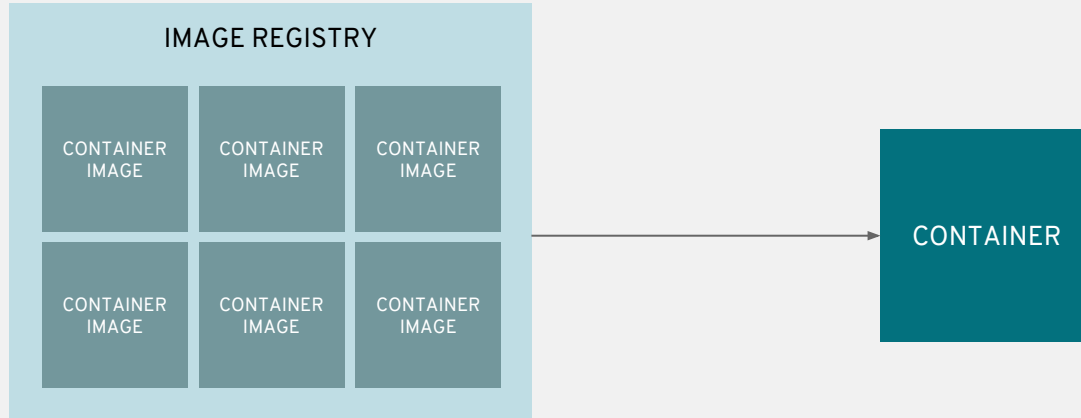


CONTAINER

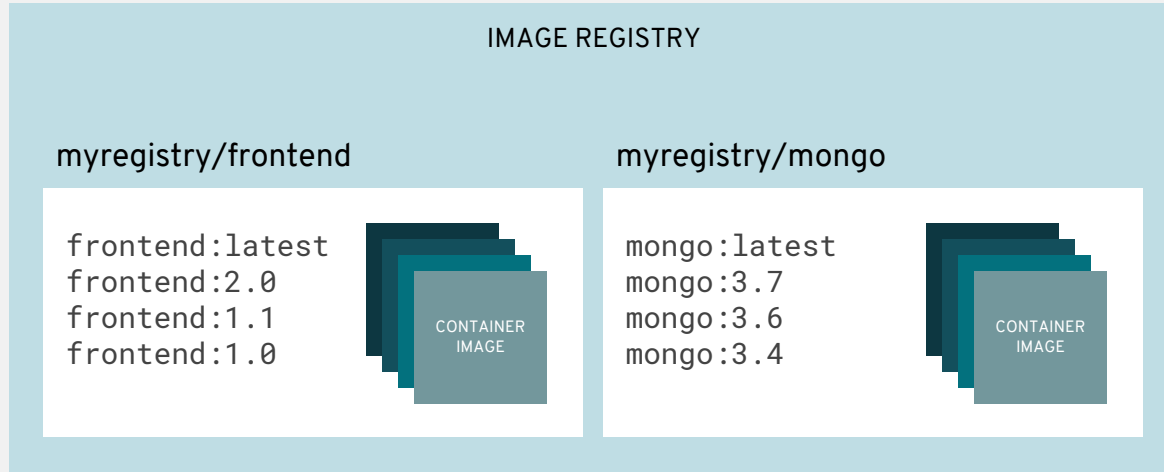
# Containers are created from container images



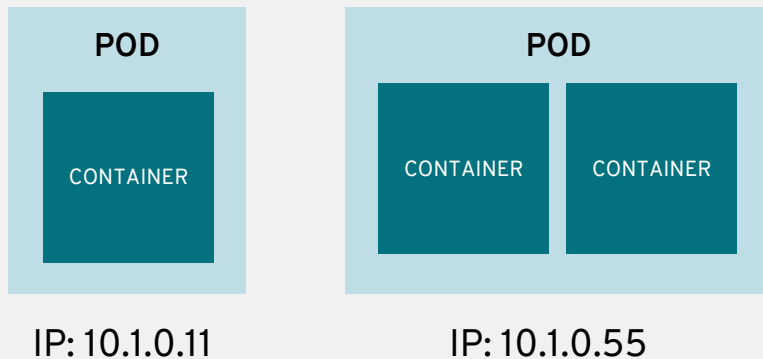
# Container images are stored in an image registry



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

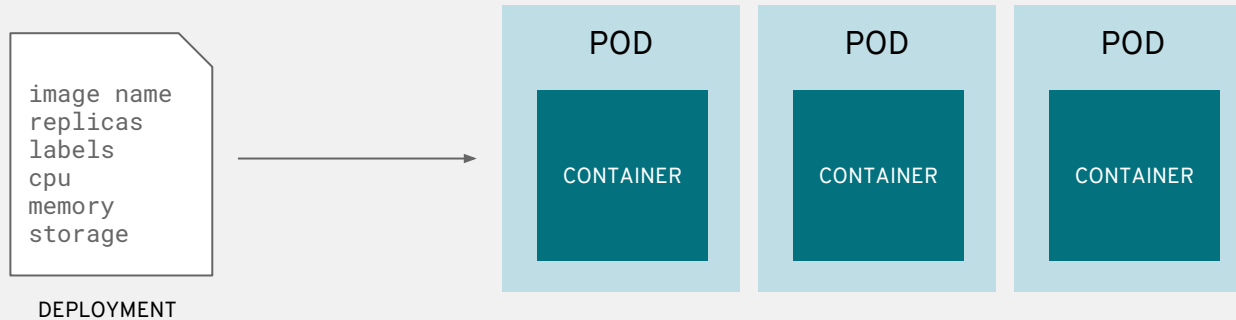


# Containers are wrapped in pods which are units of deployment and management

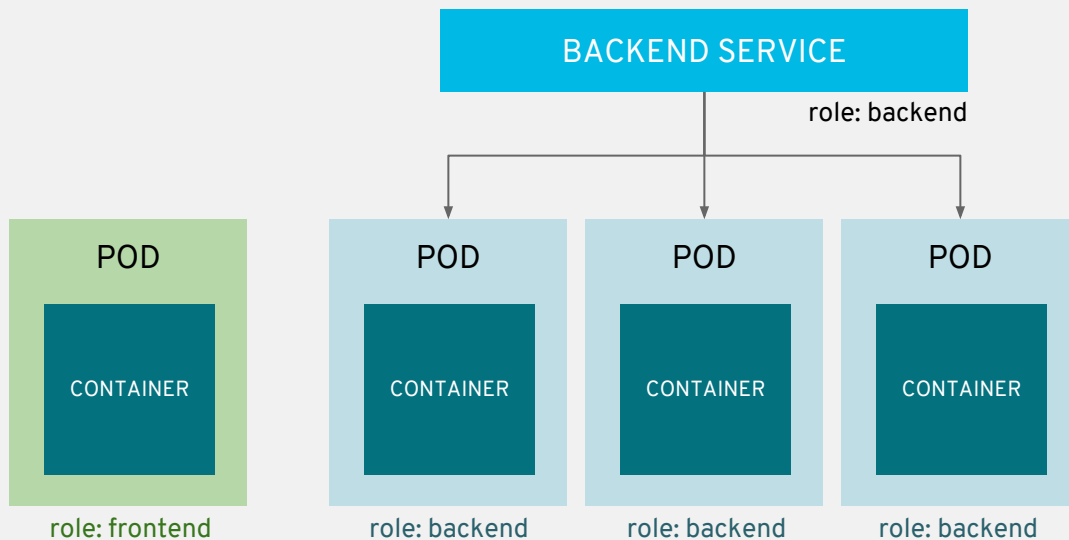




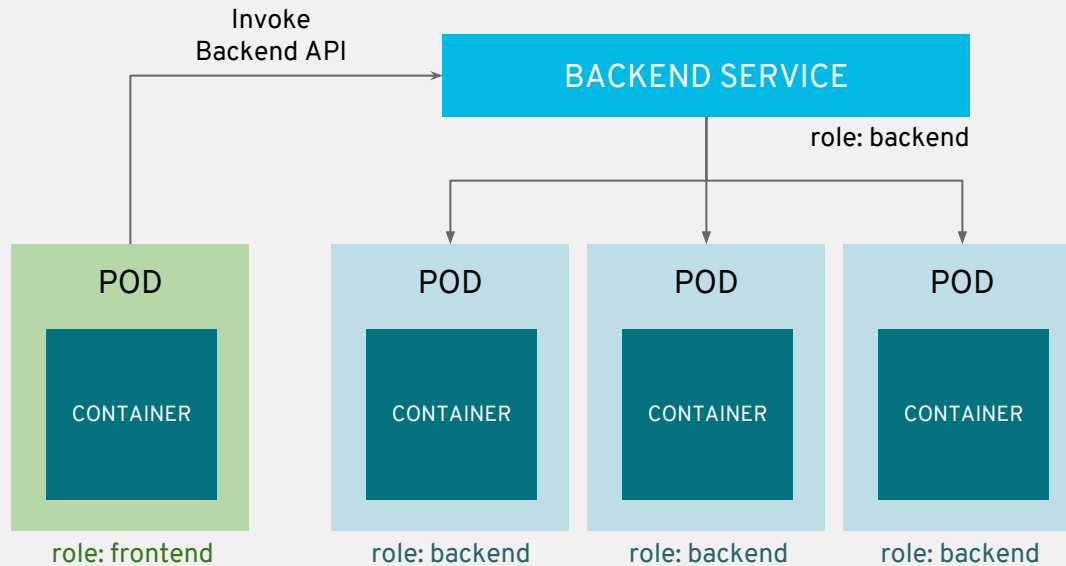
# Pods configuration is defined in a deployment



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

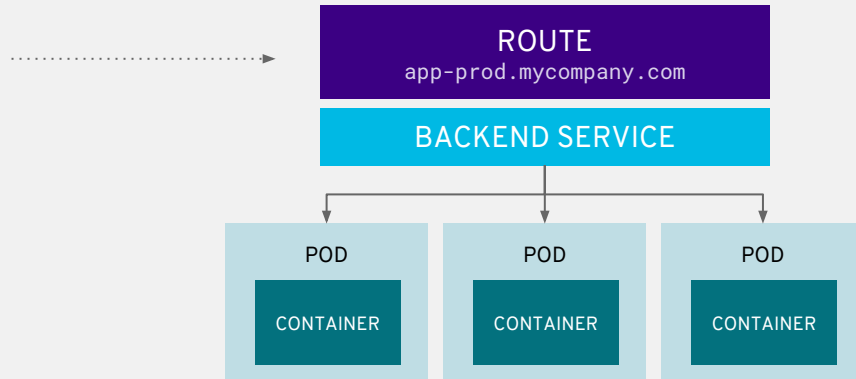


# Apps can talk to each other via services

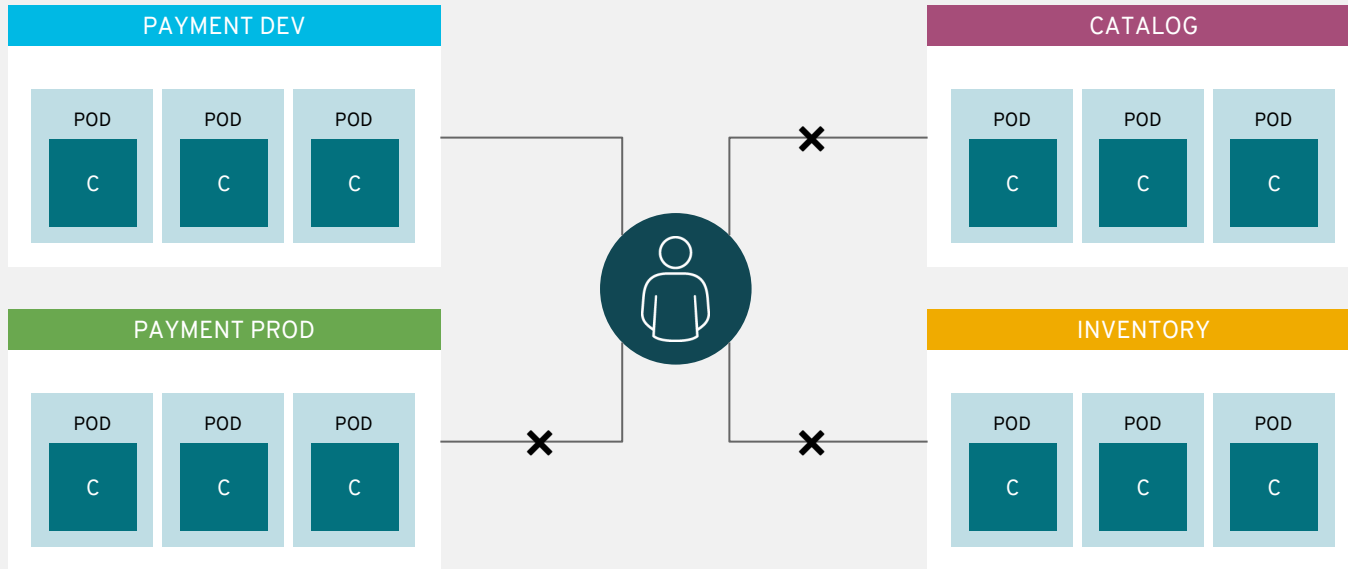


# Routes add services to the external load-balancer and provide readable urls for the app

```
> curl http://app-prod.mycompany.com
```



# 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://masterdnscbmvtzhuqye.francecentral.cloudapp.azure.com/ -u userX -p mypassword
```

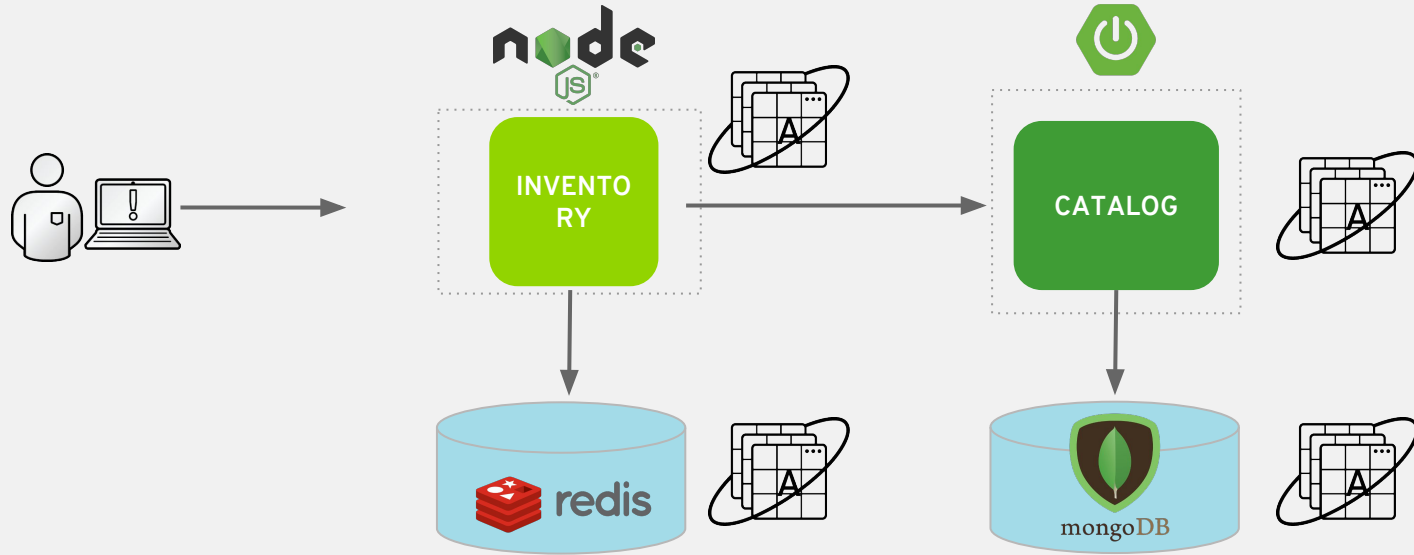
```
Login successful.
```

```
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

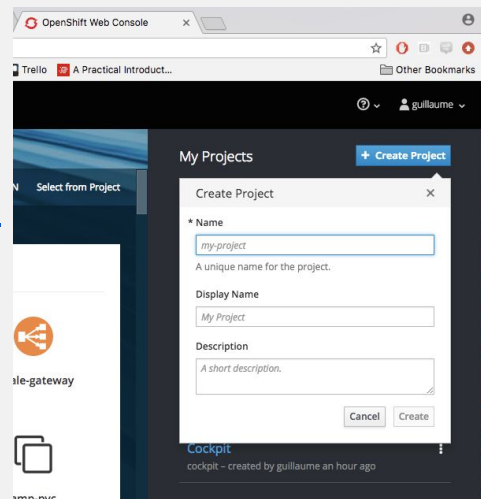
# Create your development environment

Let's go the Web Console

- Via the web console :

<https://masterdnsbmvtdzhvuqye.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

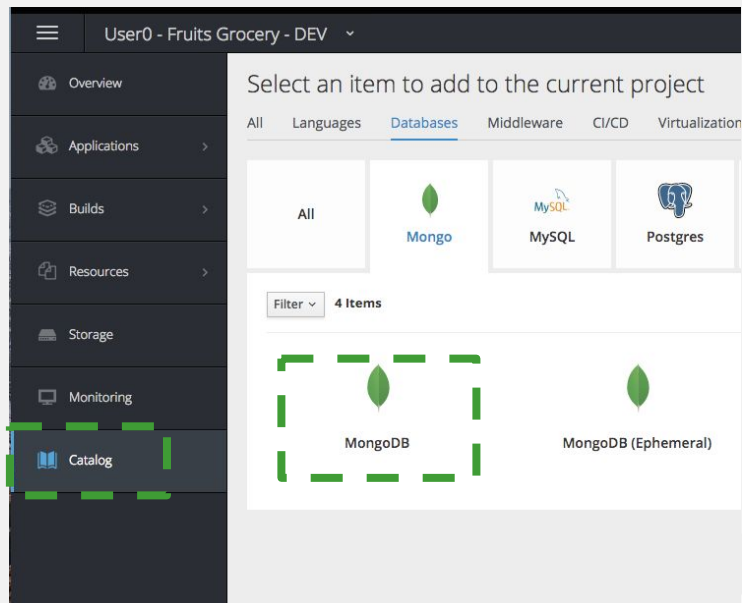
- Browse the service catalog and search for MongoDB

- 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



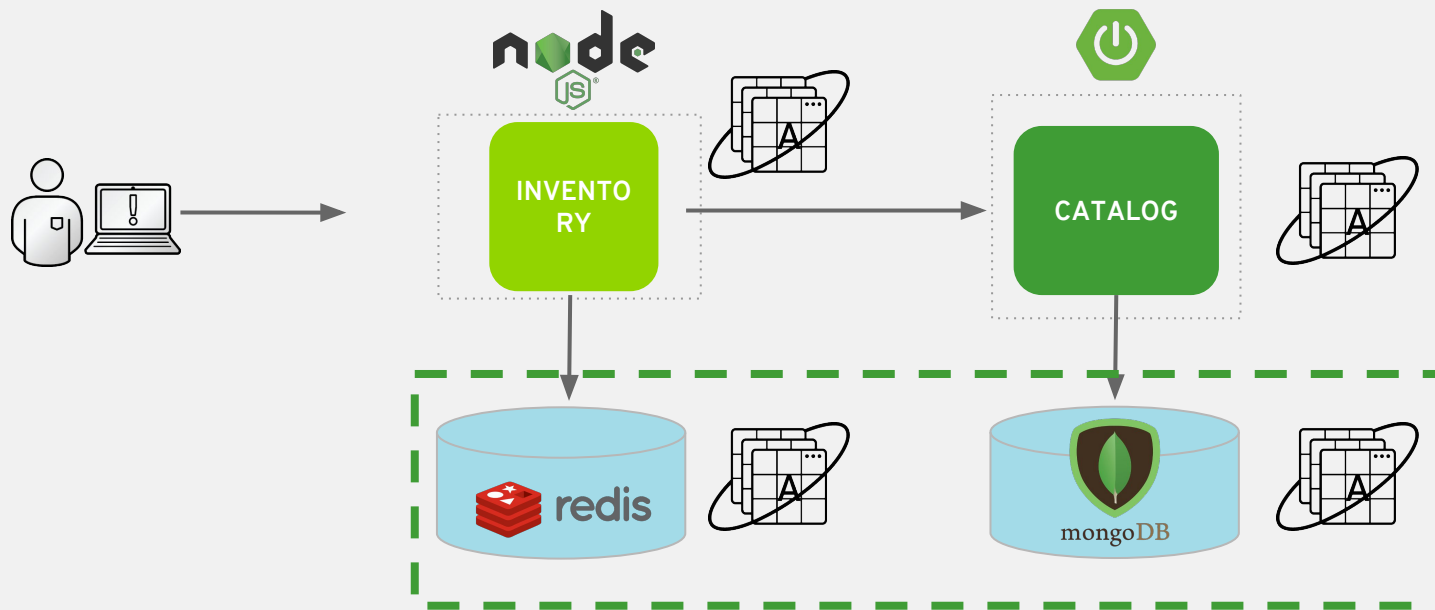
# Check Redis deployment

The screenshot displays the OpenShift console interface for an application named 'fruits-inventory'. The 'DEPLOYMENT CONFIG' section is expanded to show 'redis, #1'. Under the 'CONTAINERS' tab, the 'redis' container is detailed with the image 'openshift/redis' and port '6379/TCP'. The 'NETWORKING' section shows a 'Service - Internal Traffic' for 'redis' with port '6379/TCP (redis) -> 6379'. A 'Routes - External Traffic' section includes a 'Create Route' button. On the right, a pod status indicator shows '1 pod' running.

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



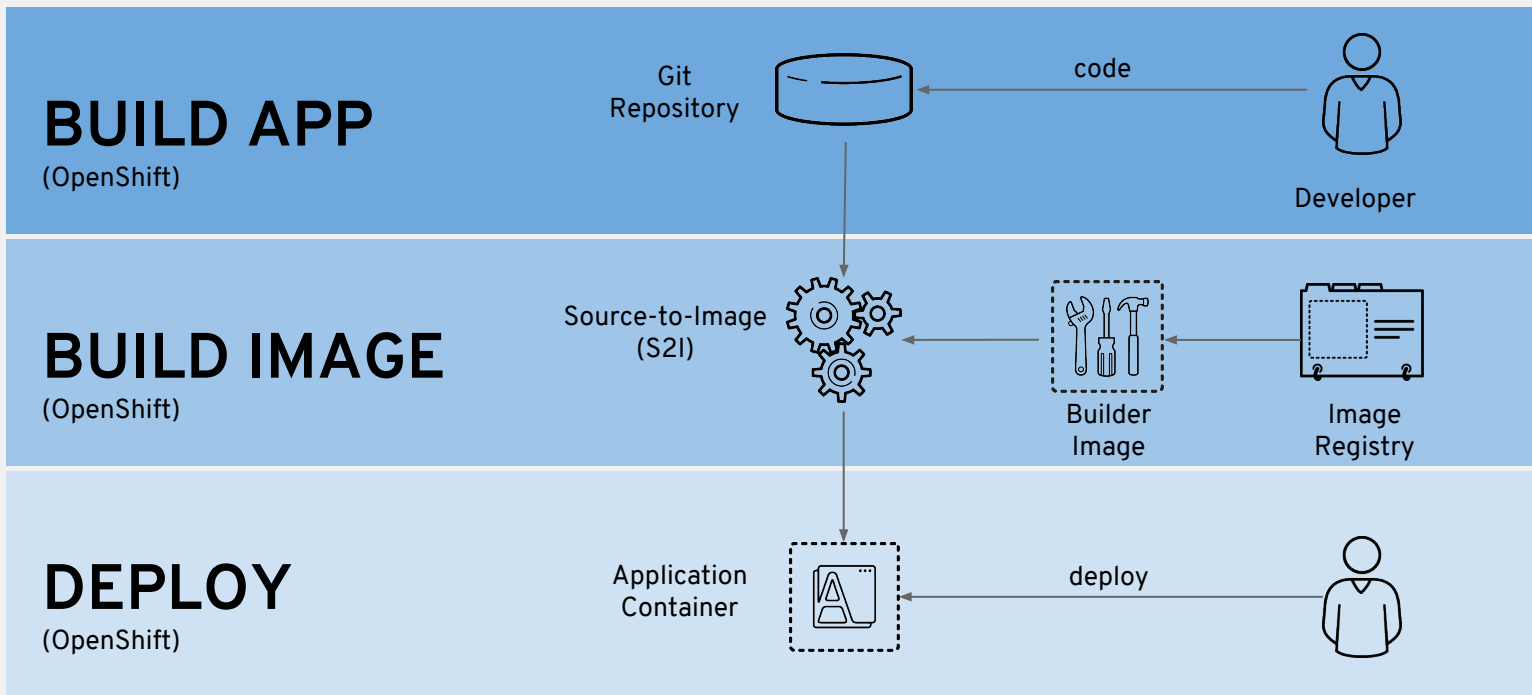
# Grocery Store on OpenShift



# LAB 3

## Deploy containers from source

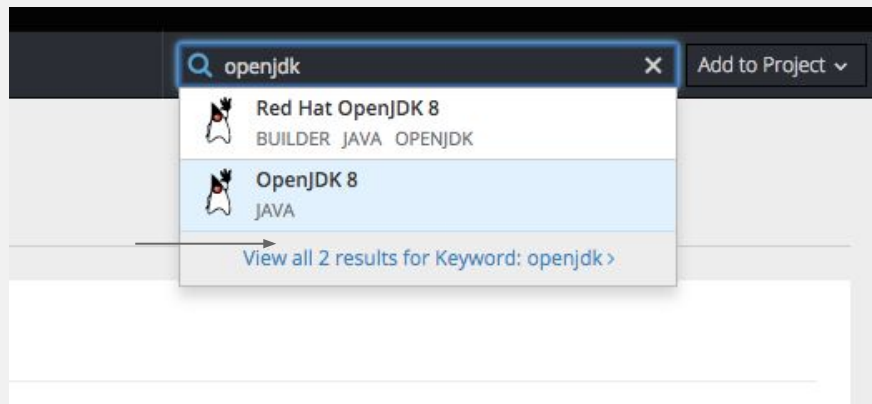
# Deploy the fruits catalog with s2i strategy



# Deploy the fruits catalog

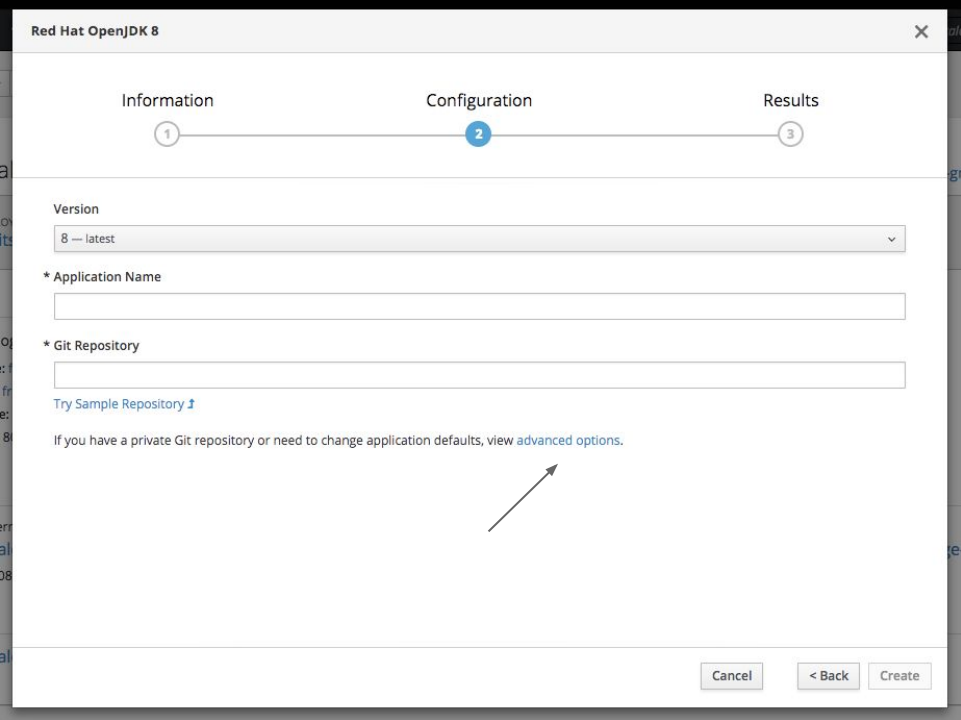
Our application is developed with Spring Boot. A powerful Java framework to build next-gen application and leverage Openshift capabilities.

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!



The screenshot shows a deployment wizard window titled "Red Hat OpenJDK 8". At the top, there is a progress bar with three steps: "Information" (1), "Configuration" (2, currently active), and "Results" (3). Below the progress bar, the "Configuration" section contains the following fields:


- Version:** A dropdown menu showing "8 -- latest".
- \* Application Name:** An empty text input field.
- \* Git Repository:** An empty text input field.
- Try Sample Repository:** A blue link with a downward arrow.
- Advanced options:** A text label with a blue link "view advanced options." and an arrow pointing to it.

At the bottom right of the window, there are three buttons: "Cancel", "< Back", and "Create".

# 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



Red Hat OpenJDK 8

Build and run Java applications using Maven and OpenJDK 8.

Version: 8

\* Name

Identifies the resources created for this application.

\* Git Repository URL

Sample repository for java: <https://github.com/jboss-openshift/openshift-quickstarts>, context dir: undertow-servlet [Try it ↕](#)

Git Reference

Optional branch, tag, or commit.

Context Dir

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

[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 displays the OpenShift console interface for the 'fruits-catalog' application. The top navigation bar shows the application name and a URL: <http://fruits-catalog-fruits-grocery-dev-user0.ge-apps.openhybridcloud.io>. Below this, the 'DEPLOYMENT CONFIG' section is expanded to show 'fruits-catalog, #5'. The 'CONTAINERS' section lists the 'fruits-catalog' container with details: Image: fruits-grocery-dev-user0/fruits-catalog 5eeb889 351.6 MiB, Build: fruits-catalog, #1, Source: Adding Grafana dashboard @ee9744, and Ports: 8080/TCP and 2 others. The 'NETWORKING' section shows 'Service - Internal Traffic' for 'fruits-catalog' with ports 8080/TCP (8080-tcp) → 8080 and 2 others. It also shows 'Routes - External Traffic' with the URL <http://fruits-catalog-fruits-grocery-dev-user0.ge-apps.openhybridcloud.io> and a route for 'fruits-catalog, target port 8080-tcp'. The 'BUILDS' section shows 'fruits-catalog' with a status of 'Build #1 is complete' created 2 hours ago. A blue circle with the number '1' and the label 'pod' is highlighted in the top right corner of the console view.

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 displays the configuration for a pod named 'fruits-catalog-5-2crzd', created 24 minutes ago. The breadcrumb navigation shows the path: app > fruits-catalog > deployment > fruits-catalog-5 > deploymentconfig > fruits-catalog. The 'Details' tab is selected, showing the following information:

**Status**

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

**Container fruits-catalog**

- State:** Running since Feb 14, 2019 5:34:51 PM
- Ready:** true
- Restart Count:** 0

**Template**

**Containers**

fruits-catalog

- Image:** fruits-grocery-dev-user0/fruits-catalog 5eeb089 351.6 MiB
- Build:** fruits-catalog, #1
- Source:** Adding Grafana dashboard @ee9744 authored by lbroudoux
- Ports:** 8080/TCP, 8443/TCP, 8778/TCP
- Mount:** default-token-iv2bv → /var/run/secrets/kubernetes.io/serviceaccount read-only

**Volumes**

default-token-iv2bv

- Type:** secret (populated by a secret when the pod is created)
- Secret:** default-token-iv2bv

[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"}
```

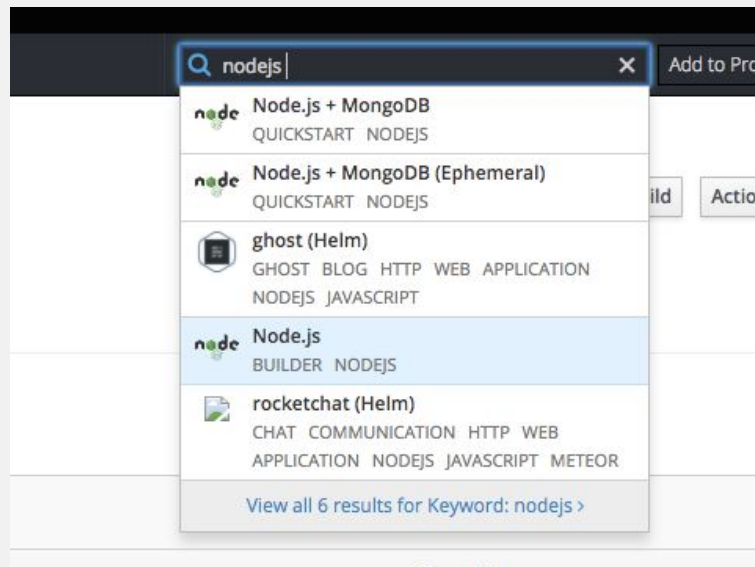
Get all fruits from the fruits-catalog

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

# Deploy the Fruits inventory

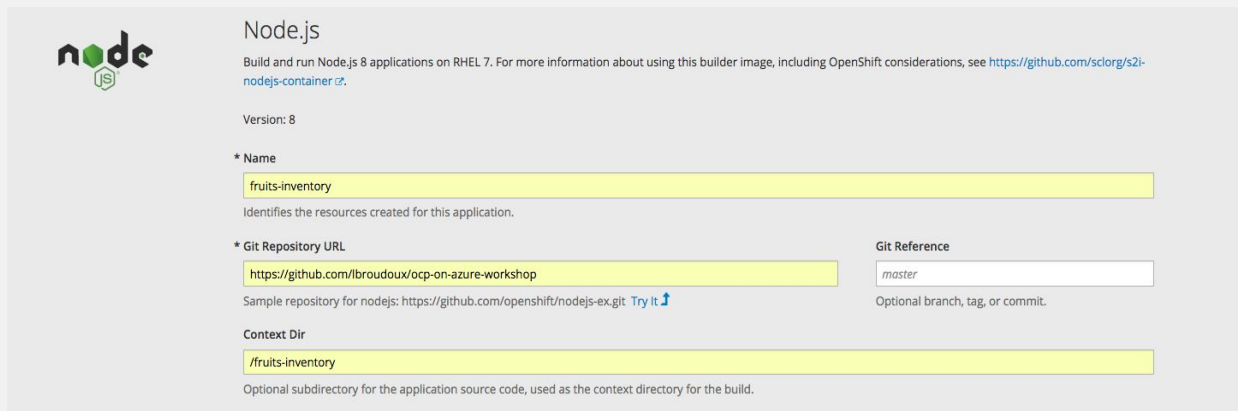
We use the same s2i strategy to build and deploy the app from source code

Choose the Node.JS image builder from the catalog



# Deploy the Fruits inventory

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



The screenshot shows the configuration interface for the Node.js builder image. It includes the Node.js logo, a description of the image, and several input fields for configuration. The fields are highlighted in yellow in the original image.

**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/sclorg/s2i-nodejs-container>.

Version: 8

\* Name  
  
Identifies the resources created for this application.

\* Git Repository URL  
  
Sample repository for nodejs: <https://github.com/openshift/nodejs-ex.git> [Try it](#) ↑

Git Reference  
  
Optional branch, tag, or commit.

Context Dir  
  
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

<input type="text" value="REDIS_HOST"/>	<input type="text" value="redis"/>	☰	×		
<input type="text" value="REDIS_PASSWORD"/>	<input type="text" value="redis - Secret"/>	▼	<input type="text" value="database-password"/>	☰	×
<input type="text" value="FRUITS_CATALOG_HOST"/>	<input type="text" value="fruits-catalog"/>	☰	×		

[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":"5c641f4d1890960016320d0","name":"Orange","origin":"Spain","quantity":"1230"}, {"id":"5c6422581890960016320d1","name":"Apple","origin":"France","quantity":"356"}]
```

You can also explore the deployment and the pod resources

# LAB 4

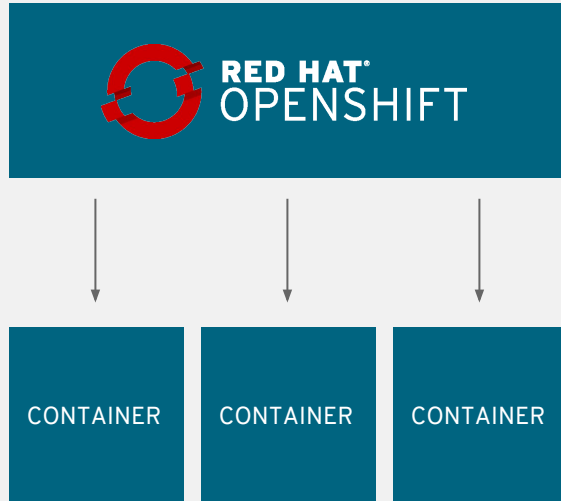
## Monitoring application health



# 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

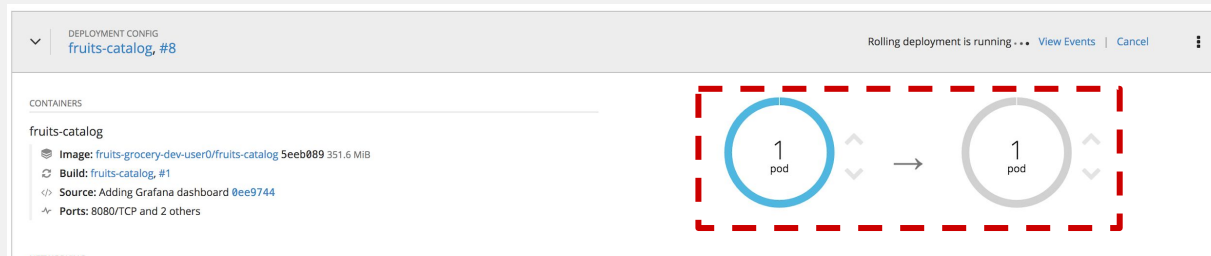
Initial Delay  
15 seconds  
How long to wait after the container starts before checking its health.

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!



The screenshot displays the OpenShift console interface for a deployment configuration named 'fruits-catalog, #8'. The status bar at the top indicates 'Rolling deployment is running...'. Below this, the 'CONTAINERS' section shows details for the 'fruits-catalog' pod, including its image, build, source, and ports. To the right, a diagram illustrates the rolling upgrade process: a light blue circle representing a pod is shown being replaced by a grey circle, with an arrow indicating the transition. The entire diagram is enclosed in a red dashed box.

# Add health checks with the CLI

We set an HTTP Request for both health checks

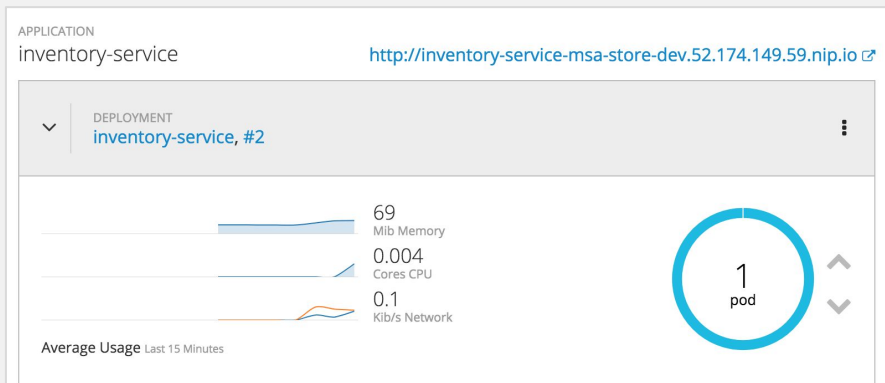
```
$ oc set probe dc/fruits-inventory --liveness --get-url=http://:8080/api/health/liveness  
--initial-delay-seconds=60 --period-seconds=30
```

```
$ oc set probe dc/fruits-inventory --readiness  
--get-url=http://:8080/api/health/readiness
```

# Monitoring pod metrics

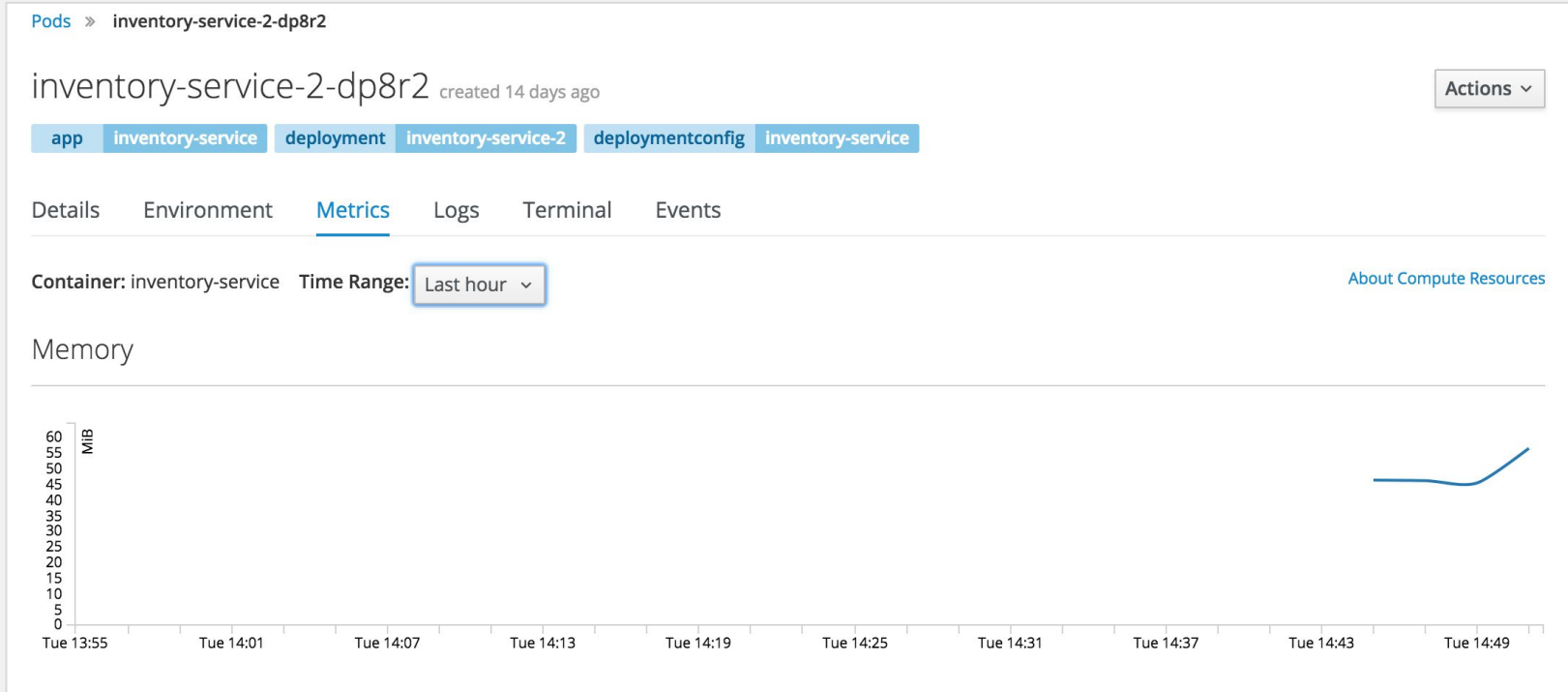
Metrics are another important aspect of monitoring applications which is required in order to gain visibility into how the application behaves and particularly in identifying issues.

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



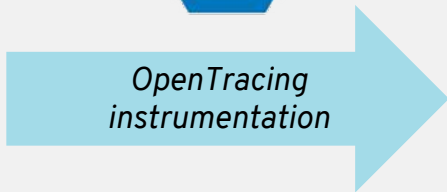
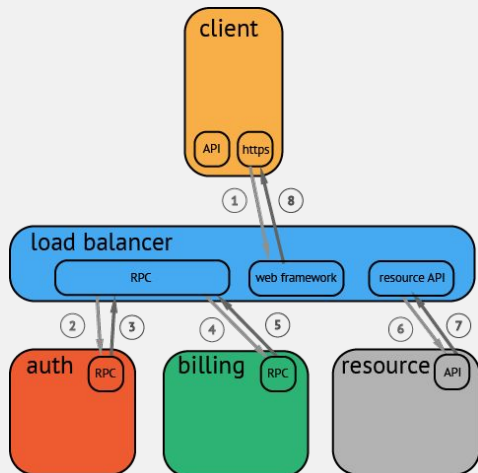
# LAB 5

## Distributed tracing configuration

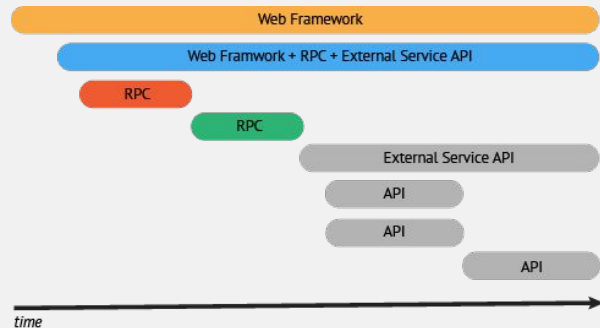
# LAB 5: Distributed tracing configuration

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

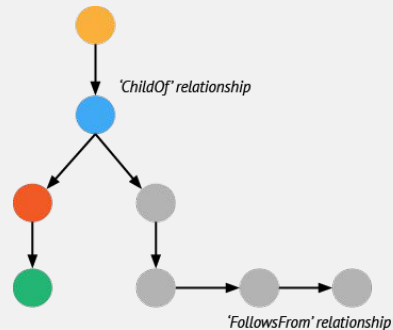
# What is distributed tracing ?



Spans

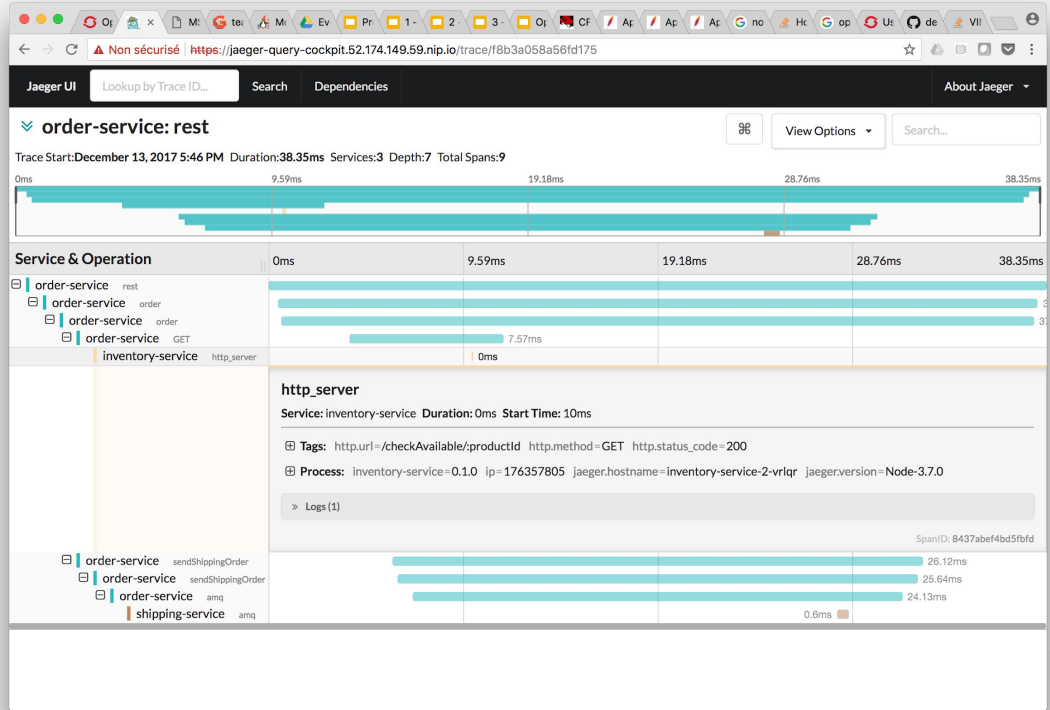
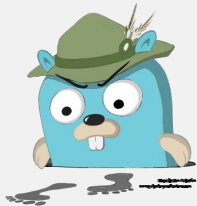


Relationships



# Distributed tracing

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



# Add Jaeger configuration to fruits-catalog

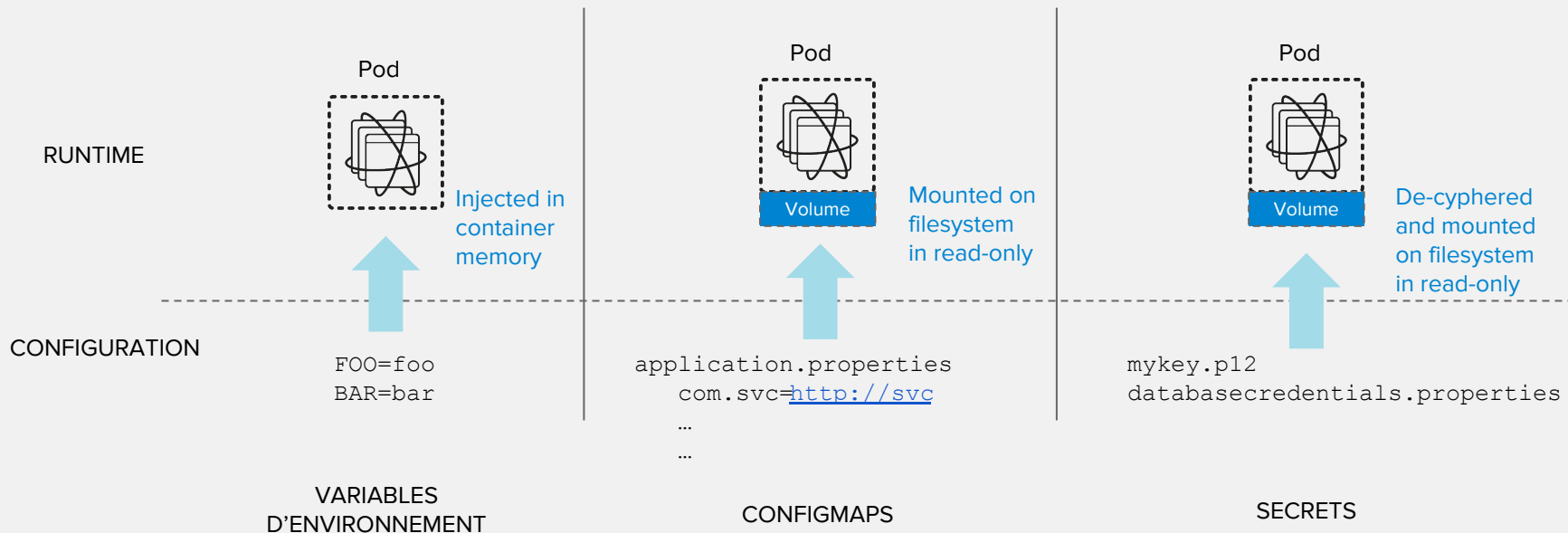
Before setting Jaeger in the fruits-catalog application, we have to add a specific role to the current project to view particular objects, especially ConfigMap ...

```
$ 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

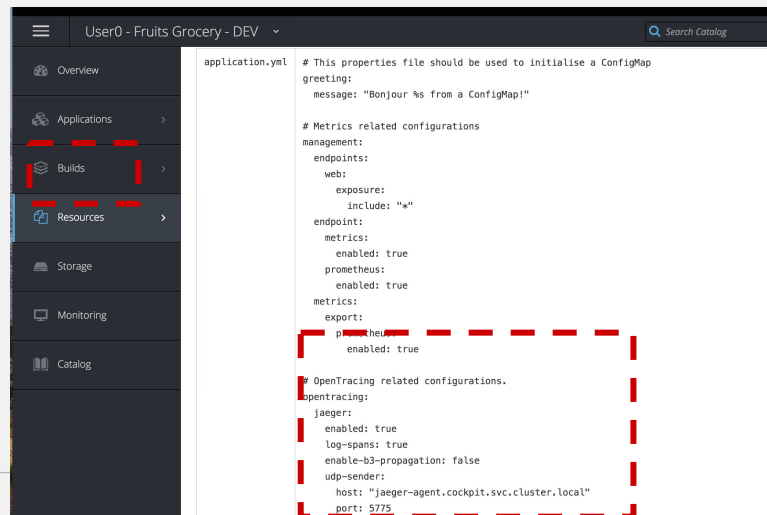
Create a configMap with the CLI

```
$ 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.

The screenshot displays the Jaeger UI interface. At the top, there are navigation tabs: "Jaeger UI", "Lookup by Trace ID...", "Search" (active), "Compare", and "Dependencies".

**Find Traces** section:

- Service (3): fruits-inventory
- Operation (3): /
- Tags (2): http.status\_code=200 error=true
- Lookback: Last 3 Hours
- Min Duration: e.g. 1.2s, 100ms, 500us
- Max Duration: e.g. 1.2s, 100ms, 500us
- Limit Results: 20
- Find Traces button

**Duration** graph: A line graph showing a single data point at 11:36:40 am with a duration of approximately 1.2s. The y-axis is labeled "Duration" with markers at 2s and 4s. The x-axis shows time from 11:36:40 am to 11:46:40 am.

**3 Traces** section:

Compare traces by selecting result items

- fruits-inventory: / 3a219d8
  - 5 Spans
    - fruits-catalog (2)
    - fruits-inventory (3)
- fruits-inventory: / 1085fa3
  - 5 Spans
    - fruits-catalog (2)
    - fruits-inventory (3)
- fruits-inventory: / d81517e
  - 1 Span
    - fruits-inventory (1)

# 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

Following filter criterias are :

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

Click on Find Traces

Find Traces

Service (3)  
fruits-catalog

Operation (4)  
find

Tags ⓘ  
hostname=fruits-catalog-4-4phqn

Lookback  
Last Hour

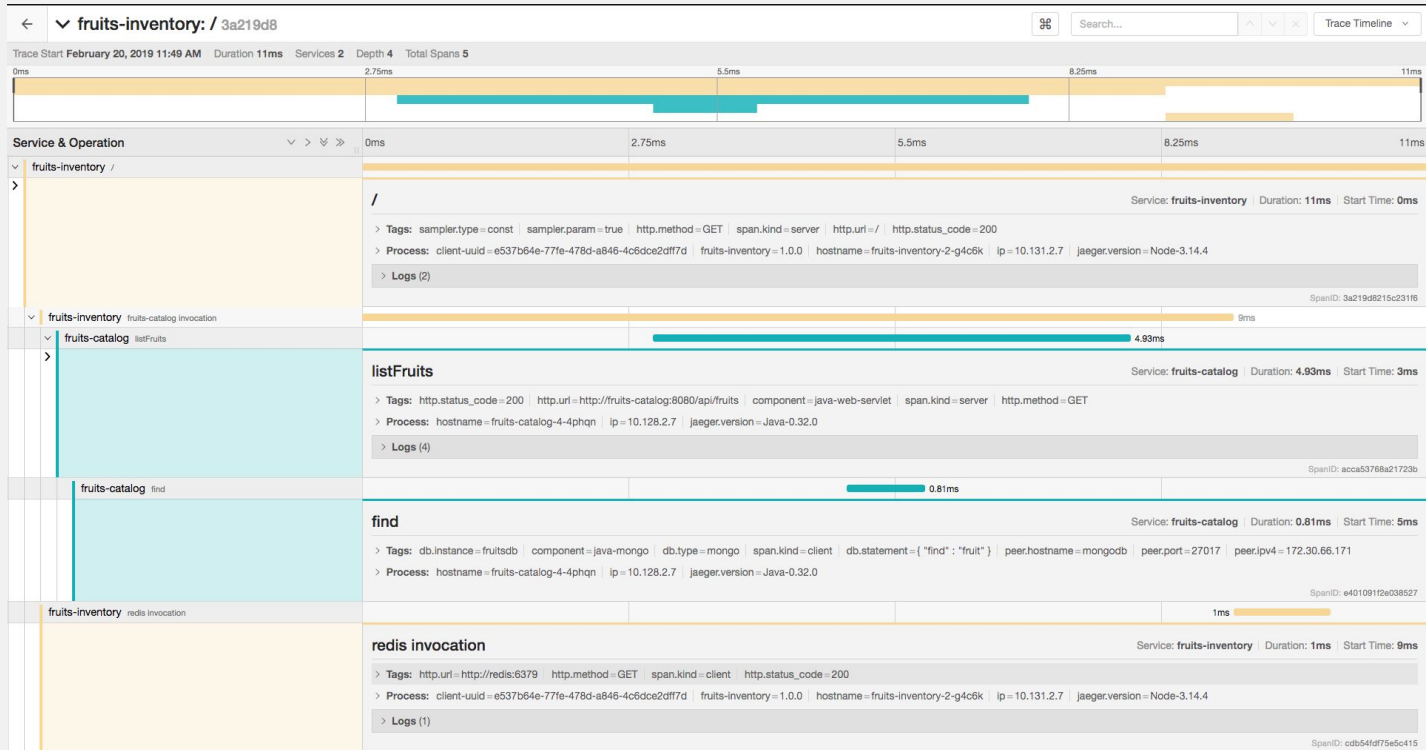
Min Duration  
e.g. 1.2s, 100ms, 500us

Max Duration  
e.g. 1.2s, 100ms, 500us

Limit Results  
20

Find Traces

# Explore the Jaeger trace



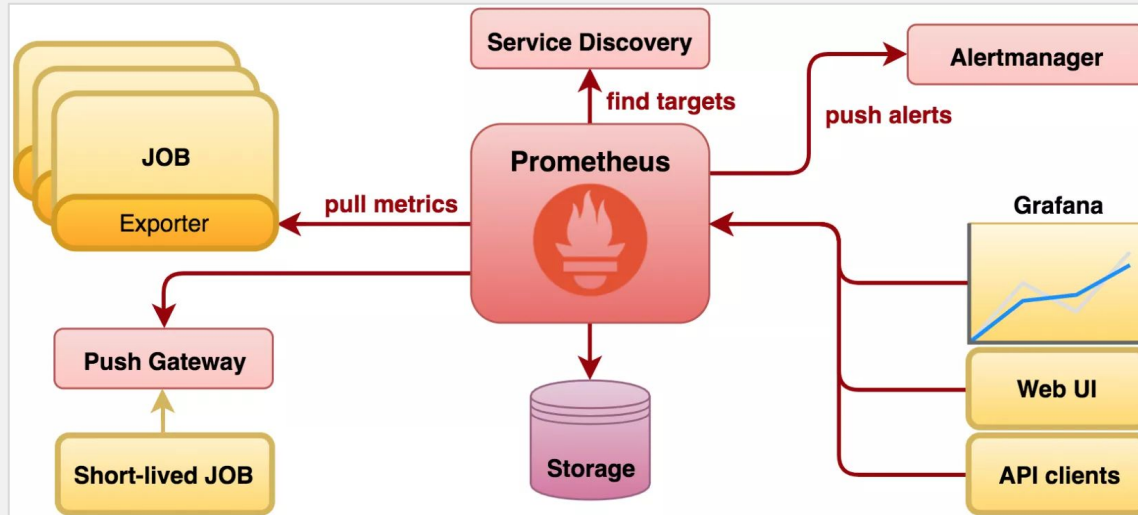
# LAB 6

## Getting application metrics

# LAB 6: GETTING APPLICATION METRICS

- Update Prometheus configuration
- Add Prometheus datasource in Grafana

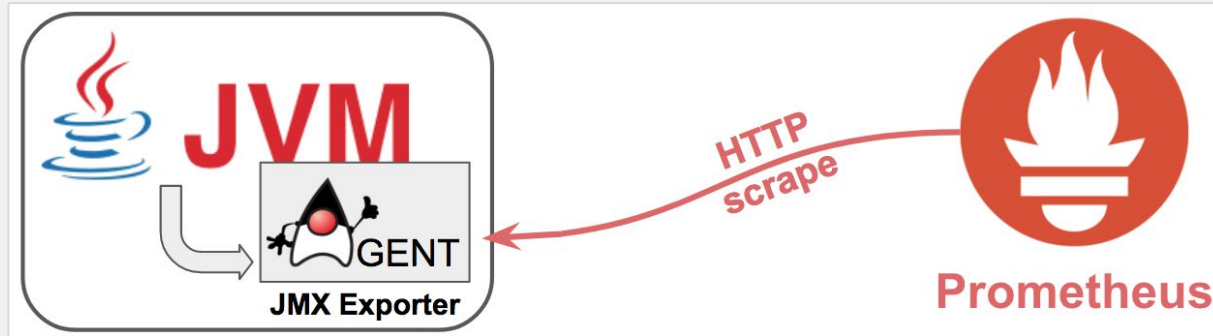
# Prometheus monitoring



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

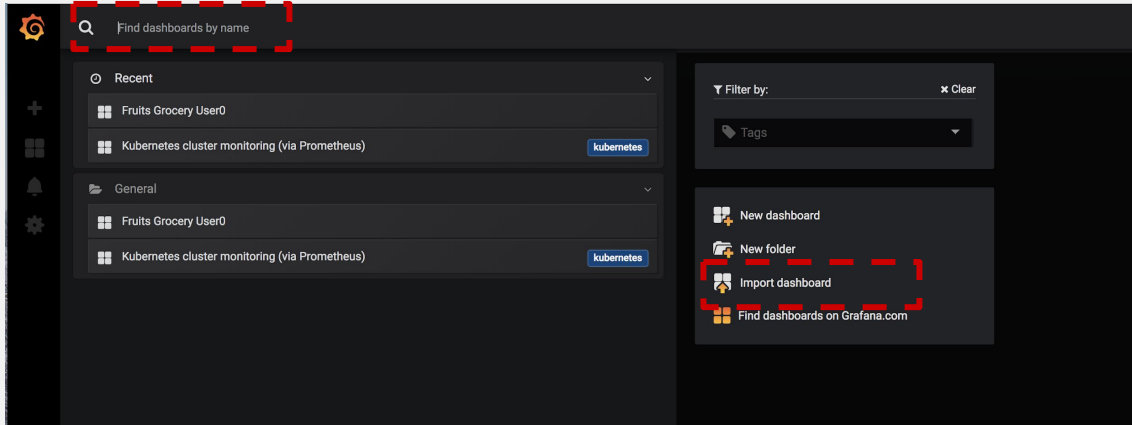
```
$ oc annotate service/fruits-catalog prometheus.io/scrape=true  
prometheus.io/path=/actuator/prometheus prometheus.io/port=8080
```

```
$ 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

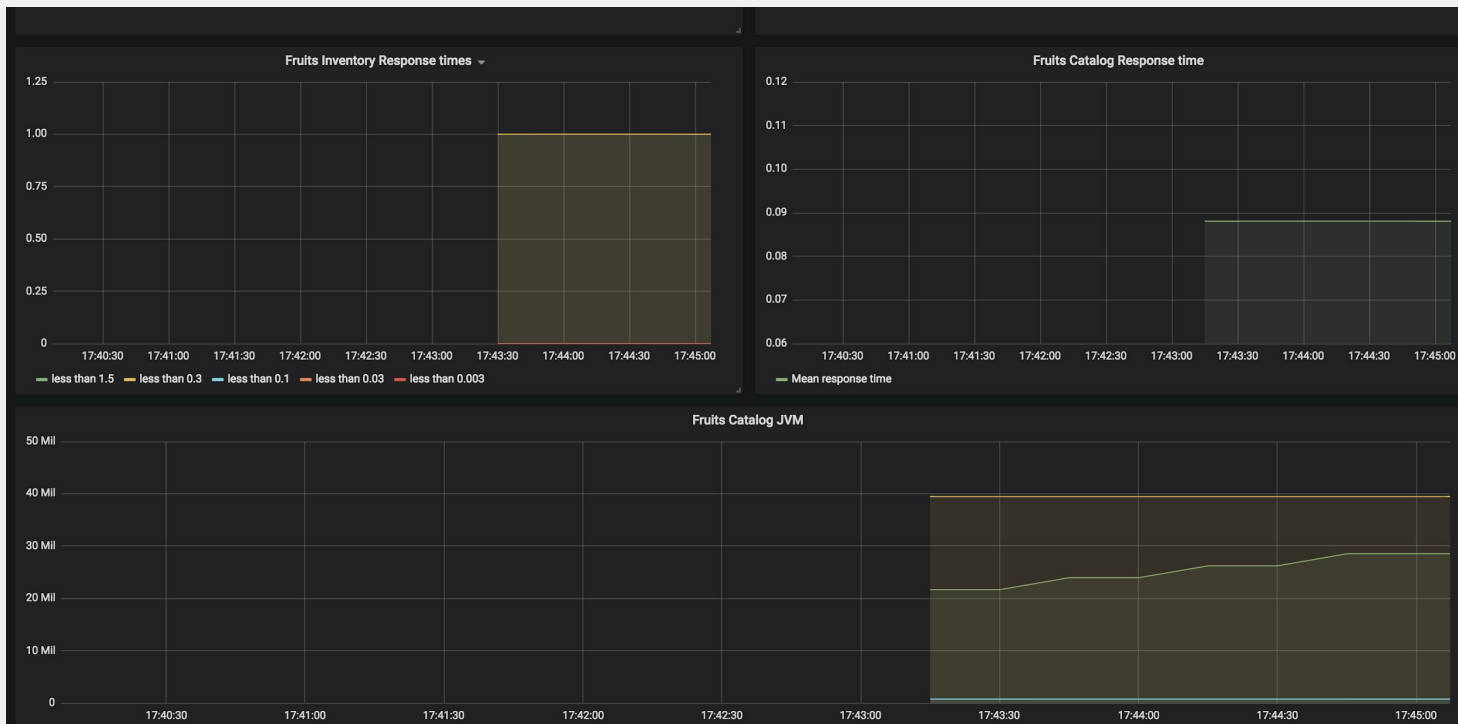
Change Dashboard name with your user ID

Options

Name	Fruits Grocery UserX	✓
Folder	General ▾	
Unique Identifier (uid)	value set	change
prometheus	Select a Prometheus data source	▾

 Import      Cancel

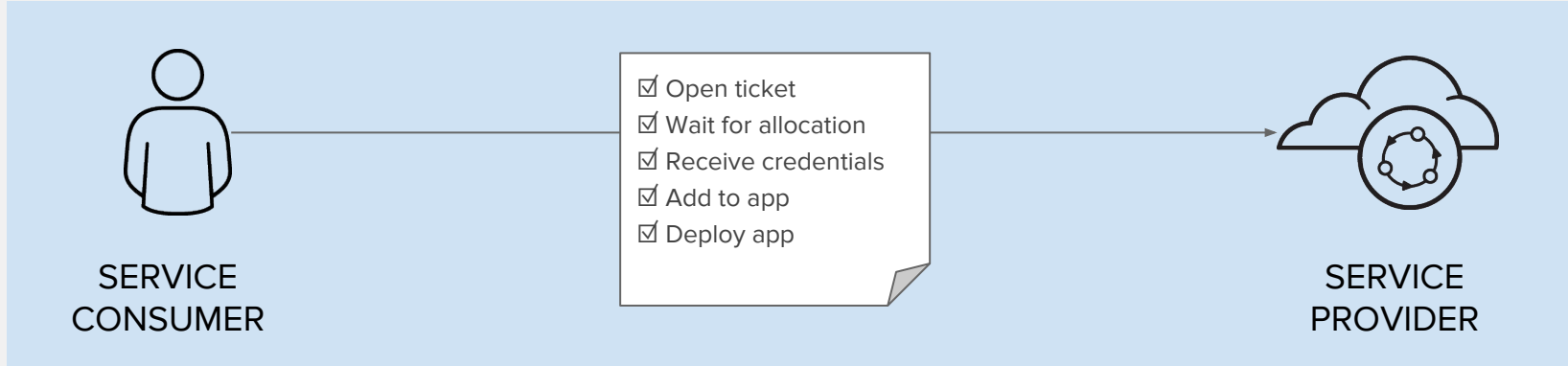
# Grafana Dashboard example



# LAB 7

## Azure Service Broker

# Why a service broker ?



Manual, Time-consuming and Inconsistent



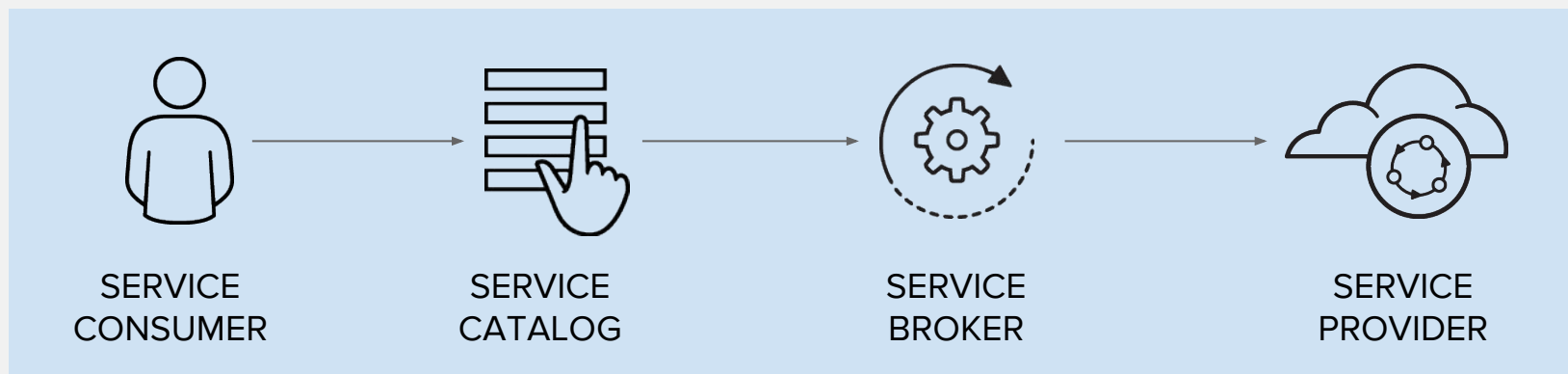


# OPEN SERVICE BROKER API™

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

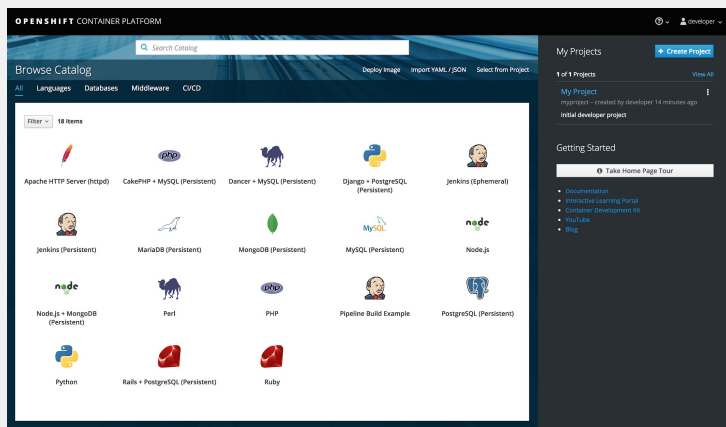


# What is a service broker ?

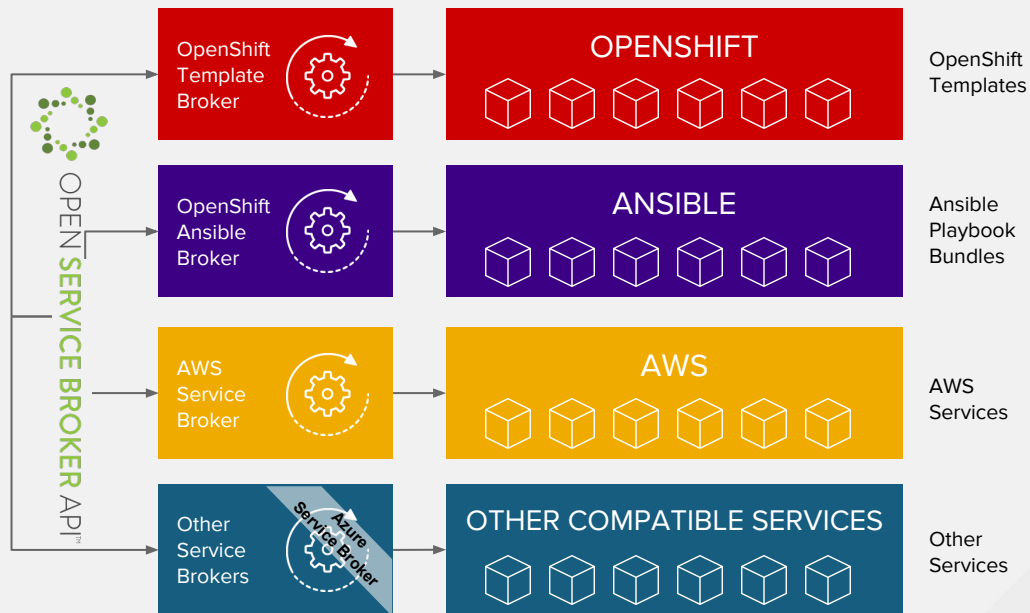


Automated, Standard and Consistent

# OpenShift service catalog



**OPENSHIFT SERVICE CATALOG**

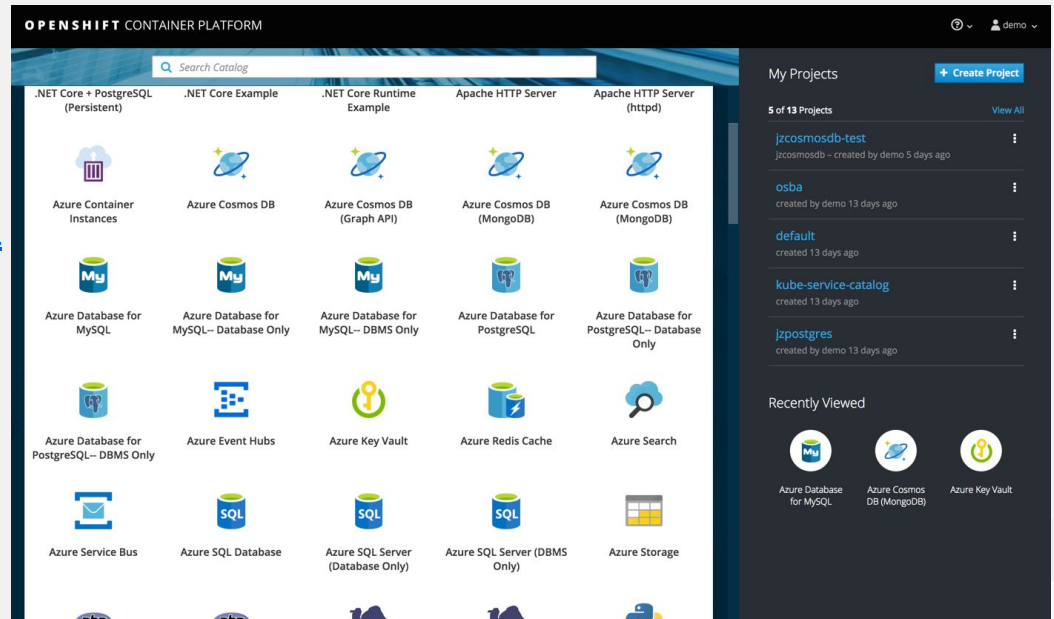


# OPEN SERVICE BROKER AZURE

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

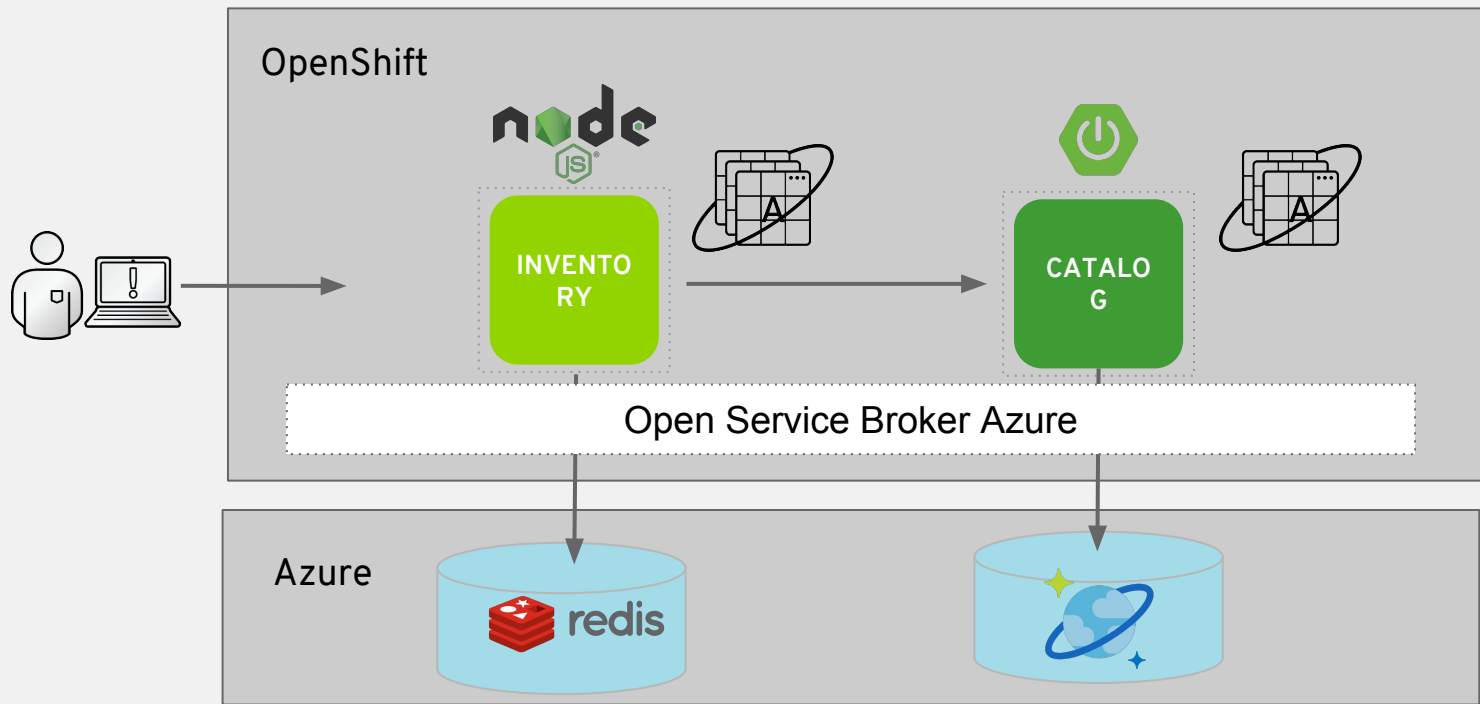
## Supported services

- [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



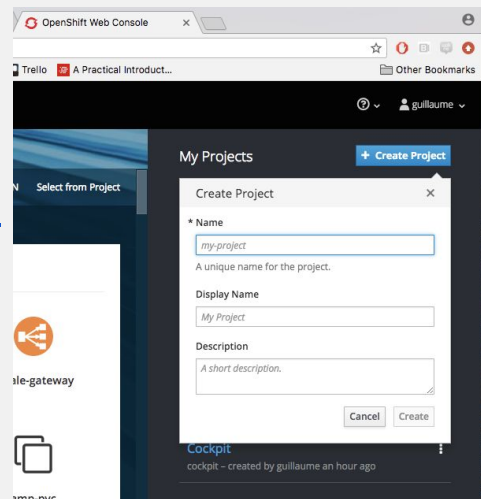
# Create your production environment

Let's go the Web Console

- Via the web console :

<https://masterdnsbmvtdzhvuqye.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**



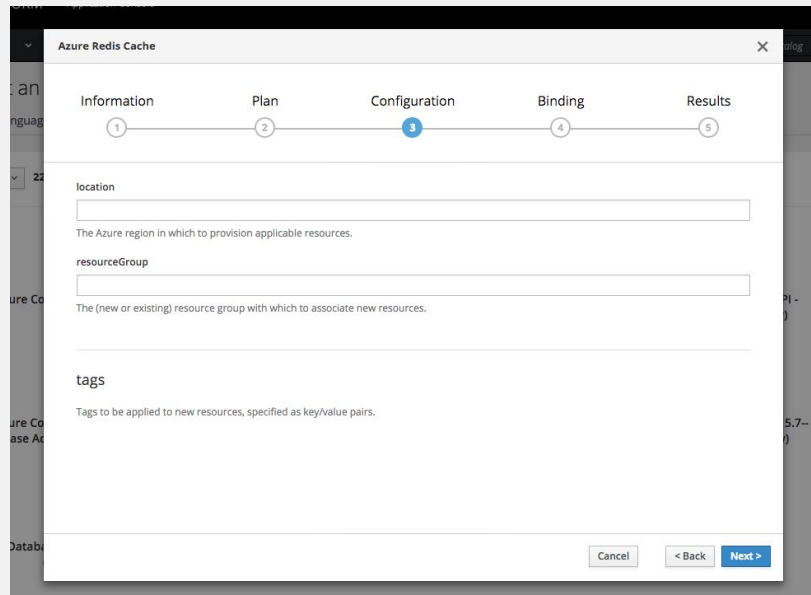
# Deploy a Redis Cache instance with Open Service Broker Azure

The screenshot displays the OpenShift Container Platform interface, specifically the service catalog. The header shows "OPENSIFT CONTAINER PLATFORM" and a search bar labeled "Search Catalog". A filter indicates "127 Items". The main area is a grid of service tiles. The "Azure Redis Cache" tile is highlighted with a green dashed border. Other visible services include .NET Core + PostgreSQL (Persistent), .NET Core Builder Images, .NET Core Example, .NET Core Runtime Example, 3scale-gateway, amp-apicast-wildcard-router, amp-pvc, Apache HTTP Server, 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 Search, Azure Service Bus, Azure SQL Database, Azure SQL Server (Database Only), Azure SQL Server (DBMS Only), Azure Storage, and CakePHP + MySQL. On the right sidebar, there is a "My Projects" section with a "Create Project" button and a list of projects: gitlab, default, apb-tasks-build, apb-tasks-prod, and apb-tasks-test. Below that is a "Recently Viewed" section with icons for gitlab-runner, JBoss EAP 7.0 (no https), Azure Database for PostgreSQL, and 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 :)



The screenshot shows the 'Azure Redis Cache' deployment wizard in the 'Configuration' step. The progress bar at the top indicates the current step is 3 out of 5. The configuration fields are as follows:

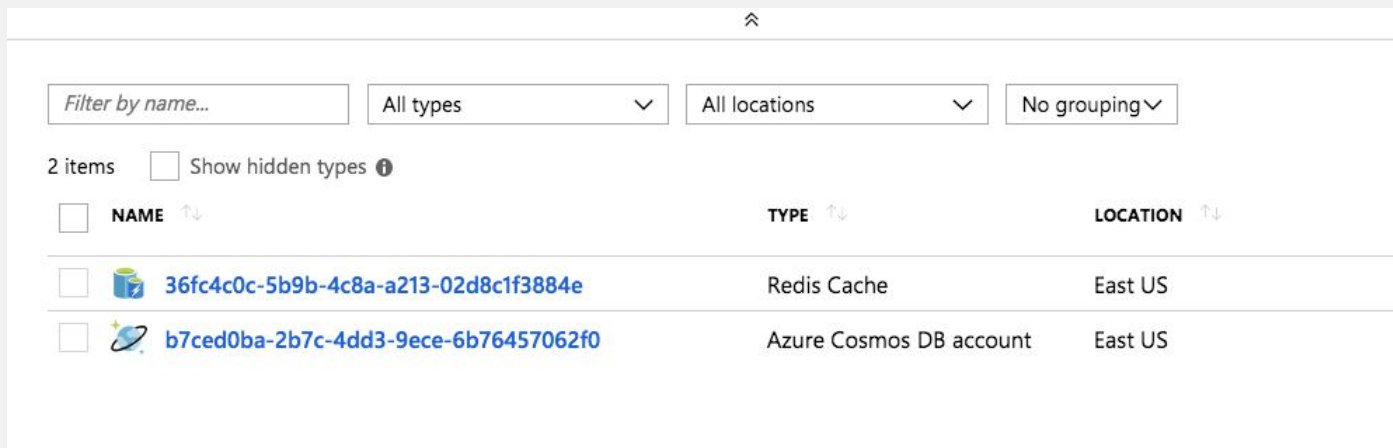
- location**: An empty text input field with the description: 'The Azure region in which to provision applicable resources.'
- resourceGroup**: An empty text input field with the description: 'The (new or existing) resource group with which to associate new resources.'
- tags**: An empty text input field with the description: 'Tags to be applied to new resources, specified as key/value pairs.'

At the bottom right, there are three buttons: 'Cancel', '< Back', and 'Next >'.





# Let's go to the backstage

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



The screenshot shows the Azure Service Broker interface. At the top, there are filter controls: a text input for "Filter by name...", a dropdown for "All types", a dropdown for "All locations", and a dropdown for "No grouping". Below the filters, it indicates "2 items" and a "Show hidden types" checkbox. The main content is a table with three columns: "NAME", "TYPE", and "LOCATION".

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

# Deploy a Cosmo DB instance with OSBA

The screenshot displays the OpenShift Container Platform interface. At the top, it says "OPENSHIFT CONTAINER PLATFORM" and "maxime". A search bar is present with the text "Search Catalog". Below the search bar, there are 127 items in the catalog. The items are displayed in a grid. The item "Azure Cosmos DB (MongoDB API)" is highlighted with a green dashed box. The right sidebar shows "My Projects" with a "+ Create Project" button and a list of 5 projects: "gitlab", "default", "apb-tasks-build", "apb-tasks-prod", and "apb-tasks-test". Below that, there is a "Recently Viewed" section with icons for "git", "gitlab-runner", "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.*

### Provisioned Services

▼ **Azure Cosmos DB (MongoDB API)**  
azure-cosmosdb-mongo-account-wnqrj

Azure Cosmos DB Database Account (MongoDB API)  
[View Documentation](#) [Get Support](#)

BINDINGS

**azure-cosmosdb-mongo-account-wnqrj-r7g6h**  
created 4 hours ago  
[Delete](#) | [View Secret](#)  
[Create Binding](#)

▼ **Azure Redis Cache**  
azure-redis-cache-clr54

**The service is not yet ready.** All associated ServiceBindings must be removed before this ServiceInstance can be deleted

Azure Redis Cache (Experimental)  
[View Documentation](#) [Get Support](#)

BINDINGS

**azure-redis-cache-clr54-kmdk8** ⌘ Pending  
created an hour ago  
[Delete](#)

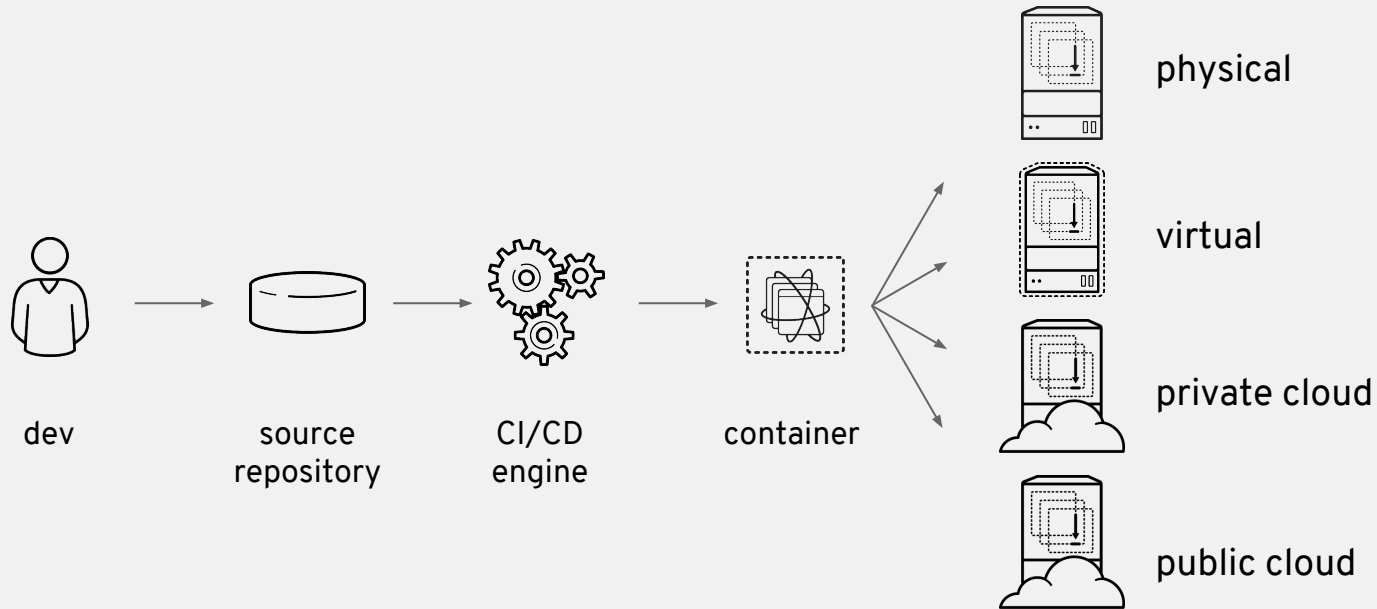
# LAB 8

## Continuous delivery

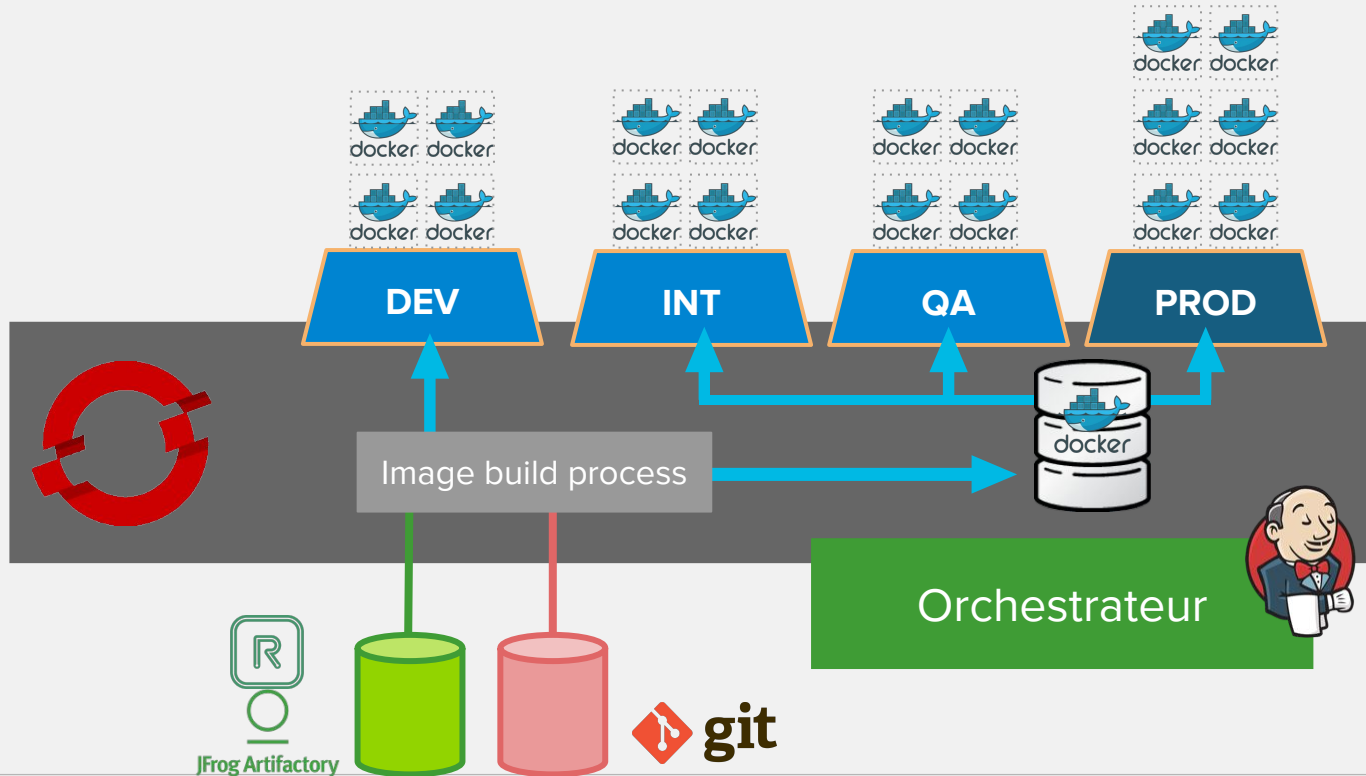
# LAB 8: Automating Deployments Using Tags and Pipelines

- 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



# 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/2TWsl5D>
- 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=		

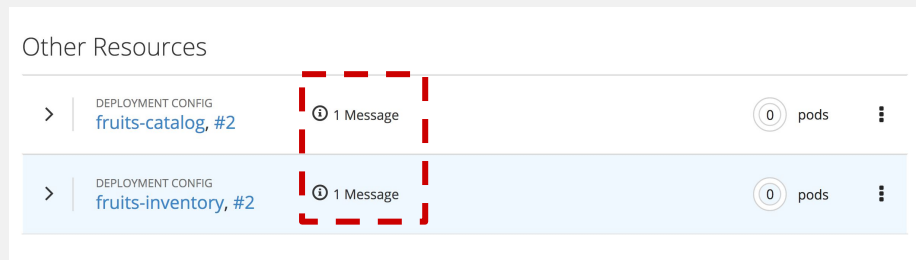
# Prepare a Production environment

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 !

DEPLOYMENT CONFIG  
fruits-catalog, #2

**fruits-catalog is paused.** This will stop any new rollouts or triggers from running until resumed. [Resume Rollouts](#)

CONTAINERS

default-container  
Image: fruits-grocery-dev-user0/fruits-catalog

0 pods

NETWORKING

Service - Internal Traffic  
fruits-catalog  
8080/TCP → 8080

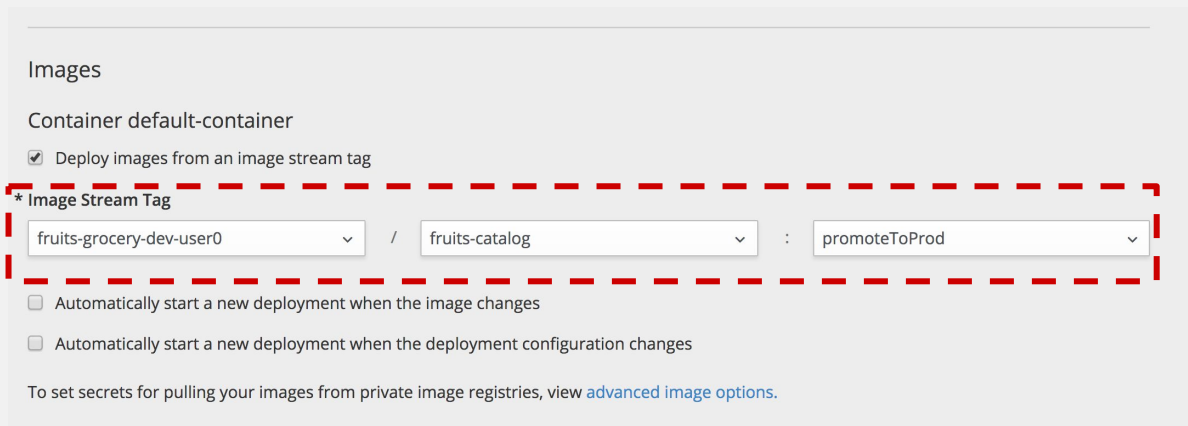
Routes - External Traffic  
<http://fruits-catalog-fruits-grocery-prod-user0-gs-apps.openhybridcloud.io>  
Route fruits-catalog, target port 8080

# 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 exist, deployment will fail !



Images

Container default-container

Deploy images from an image stream tag

\* Image Stream Tag

fruits-grocery-dev-user0 / fruits-catalog : promoteToProd

Automatically start a new deployment when the image changes

Automatically start a new deployment when the deployment configuration changes

To set secrets for pulling your images from private image registries, view [advanced image options](#).

# 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.

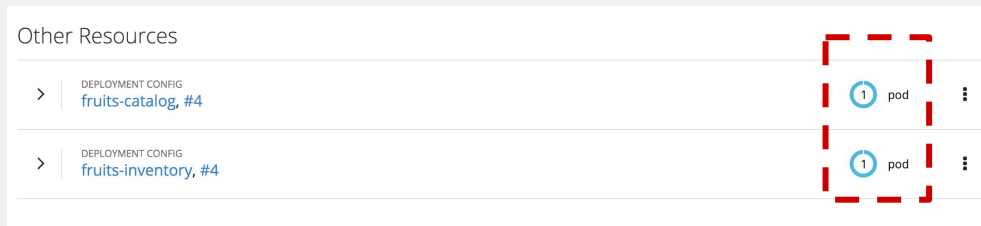
```
$ . /home/userX/deploy-prod.sh

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.





# 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 configuration page for a Jenkins pipeline named 'fruits-inventory-pipeline', created 3 days ago. The page has tabs for 'name', 'fruits-inventory-pipeline', 'template', and 'fruits-inventory-pipeline'. Below the tabs are sections for 'History', 'Configuration', and 'Events'. The 'Configuration' section is active and shows 'Details' and 'Triggers'.

**Details**

- Build Strategy:** Jenkins Pipeline
- Run Policy:** Serial
- Jenkinsfile:** [What's a Jenkinsfile?](#)

```
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
  }
}
```

**Triggers** [Learn More](#)

- Generic Webhook URL:**  [Copy](#)
- GitHub Webhook URL:**  [Copy](#)
- Manual (CLI):**  [Copy](#)

# 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

Recent Runs Average Duration: 3m 41s

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

[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 displaying the OpenShift Interactive Learning Portal. The browser's address bar shows the URL <https://learn.openshift.com/introduction/>. The page features the OpenShift logo and the Red Hat logo. A navigation bar includes a "Report an Issue" link and a prominent red button that says "SIGN UP TO OPENSHIFT ONLINE FOR FREE". The main content area has a dark background with a grid pattern and the text "RED HAT OPENSHIFT Interactive Learning Portal". Below this, a paragraph explains that the interactive learning scenarios provide a pre-configured OpenShift instance accessible from a browser. At the bottom, there are six dark gray buttons with red "START SCENARIO" labels, each representing a different learning scenario.

OPENSHIFT

Report an Issue

SIGN UP TO OPENSHIFT ONLINE FOR FREE

RED HAT OPENSHIFT

Interactive Learning Portal

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.

Getting Started with OpenShift for Developers

START SCENARIO

Logging in to an OpenShift Cluster

START SCENARIO

Deploying Applications From Images

START SCENARIO

Deploying Applications From Source

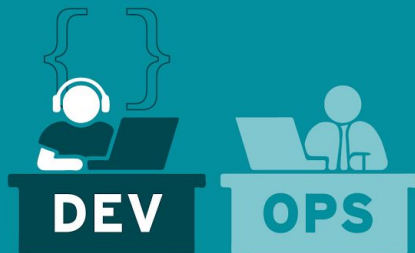
START SCENARIO

Using the CLI to Manage Resource Objects

START SCENARIO

Connecting to a Database Using Port Forwarding

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)

