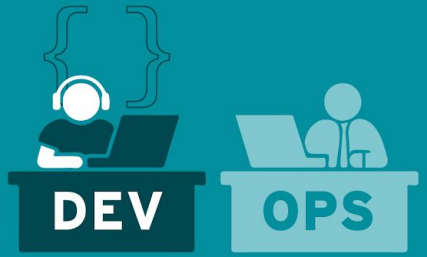


OPENSHIFT HANDS-ON @Microsoft

Développer et déployer une application Cloud-Native

Guillaume Estrem & Laurent Broudoux
AppDev Solution Architect
21 Février 2019

redhat.

LAB GUIDE

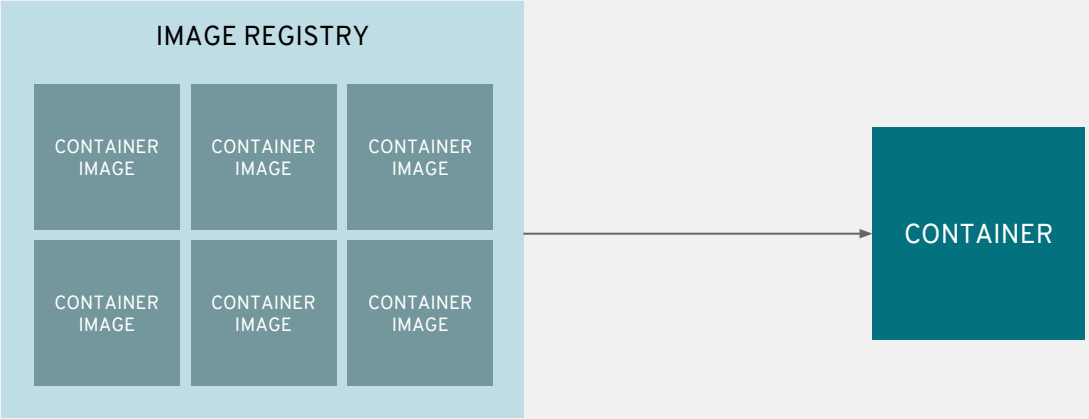
13h00			
	Lab 1	Getting Started	
	Lab 2	Deploying containers from an image	
	Lab 3	Deploying containers from sources	
	Lab 4	Monitoring application health	
14h30		PAUSE	14h45
	Lab 5	Distributed Tracing Configuration	
	Lab 6	Getting Application Metrics	
	Lab 7	Azure Service Broker	
	Lab 8	Continuous Delivery	
16h00			

OPENSHIFT CONCEPTS OVERVIEW

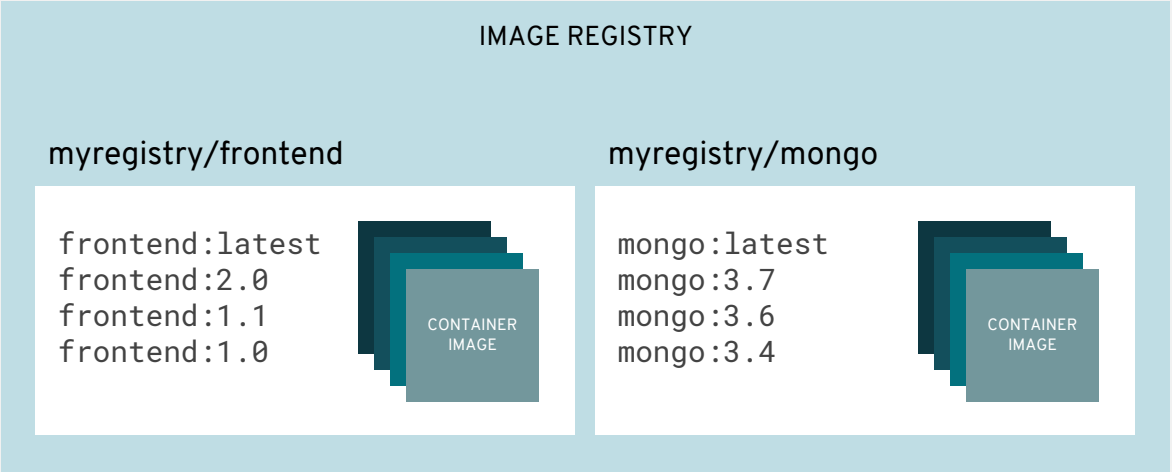
A container is the smallest compute unit

Containers are created from container images

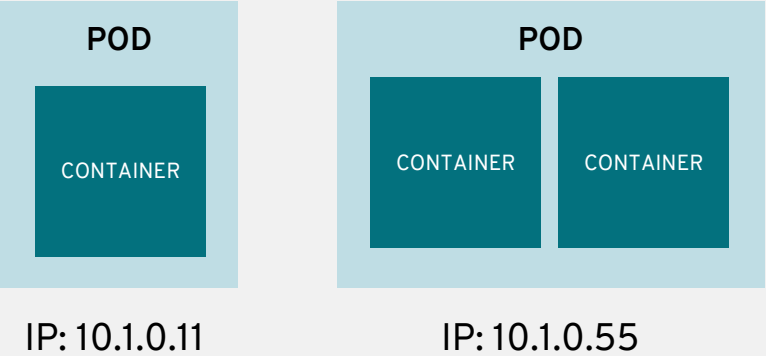
Container images are stored in an image registry



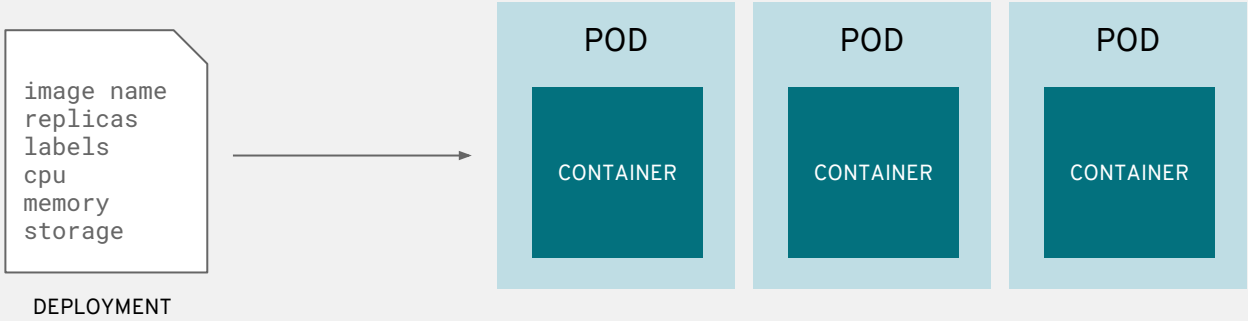
An image repository contains all versions of an image in the image registry



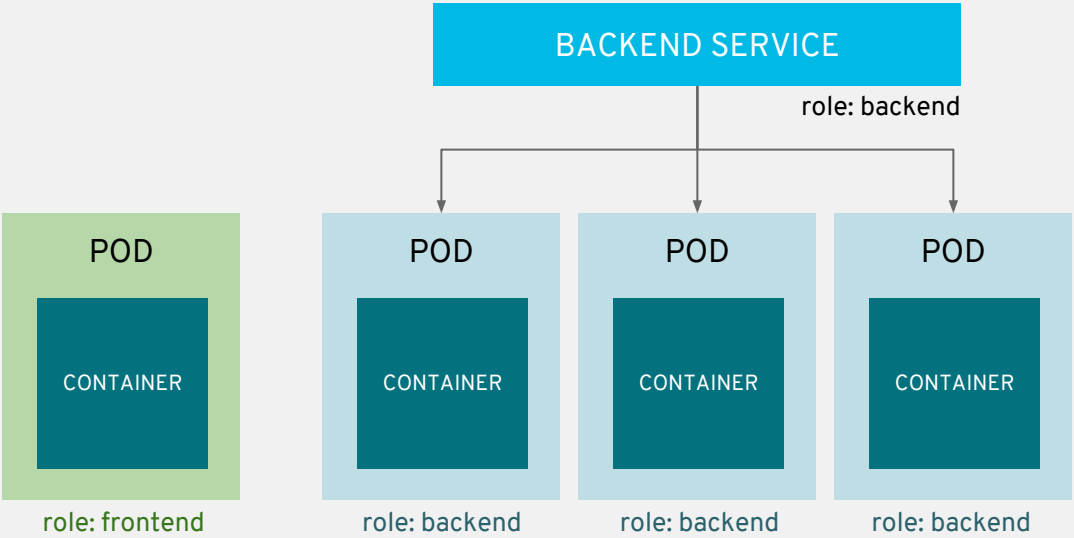
Containers are wrapped in pods which are units of deployment and management



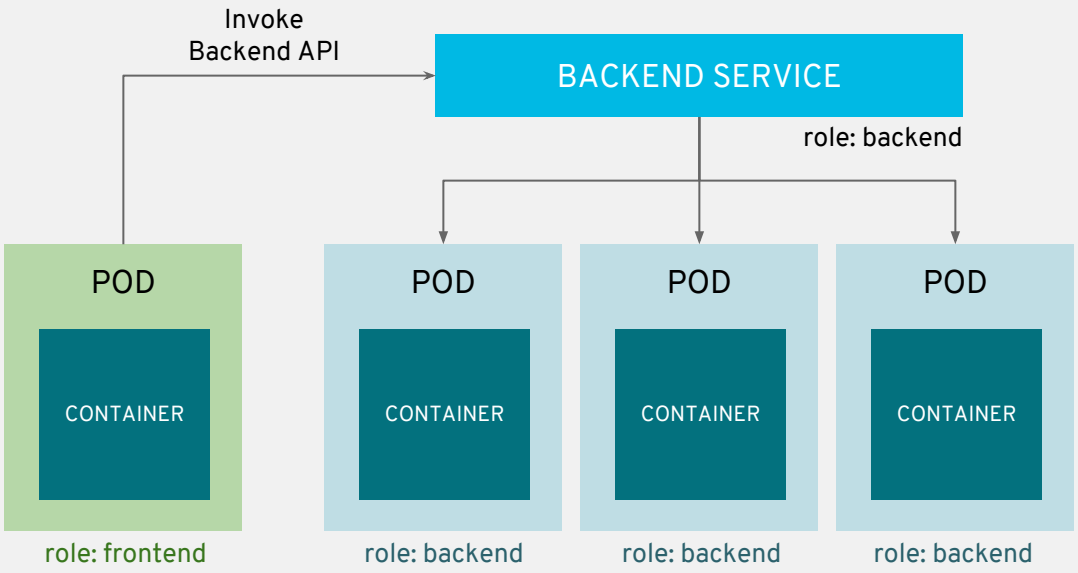
Pods configuration is defined in a `deployment`



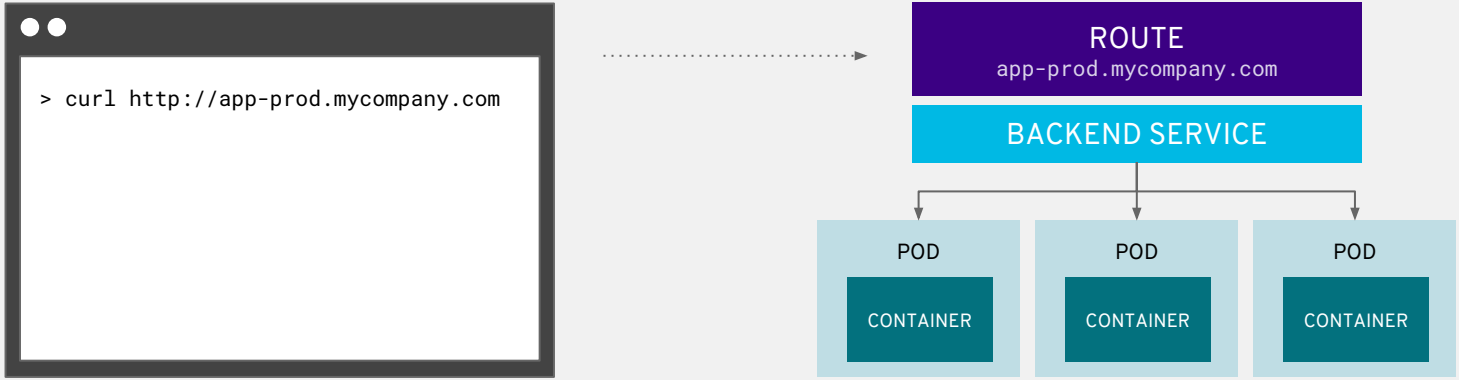
Services provide internal load-balancing and service discovery across pods



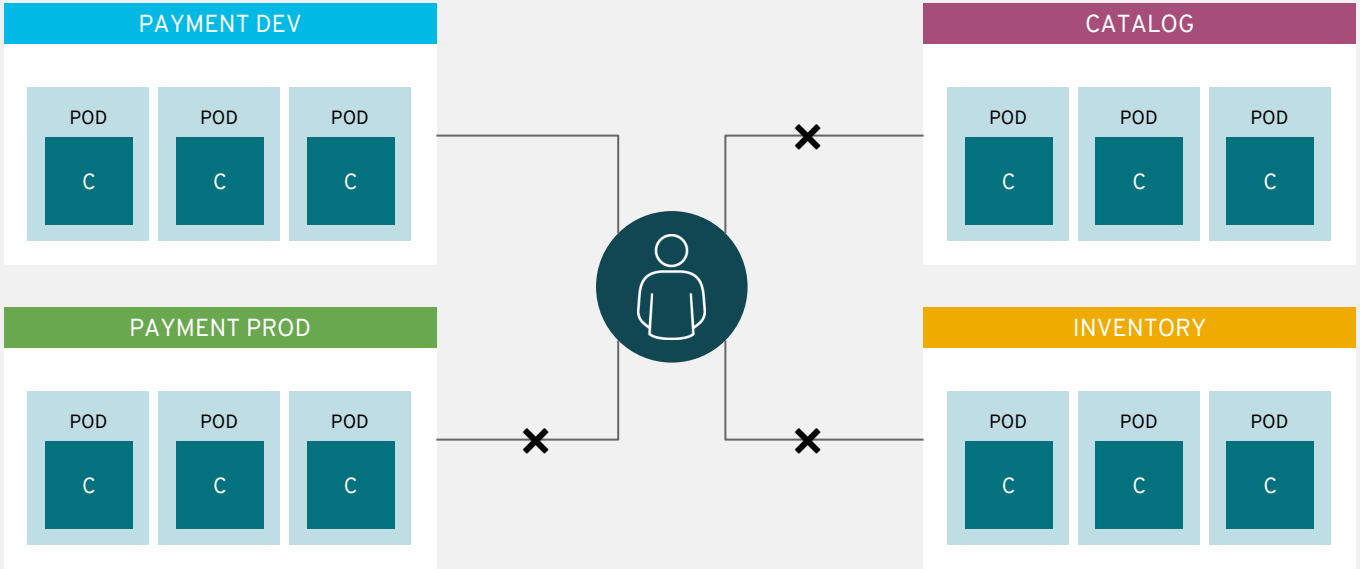
Apps can talk to each other via services



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

redhat.

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...

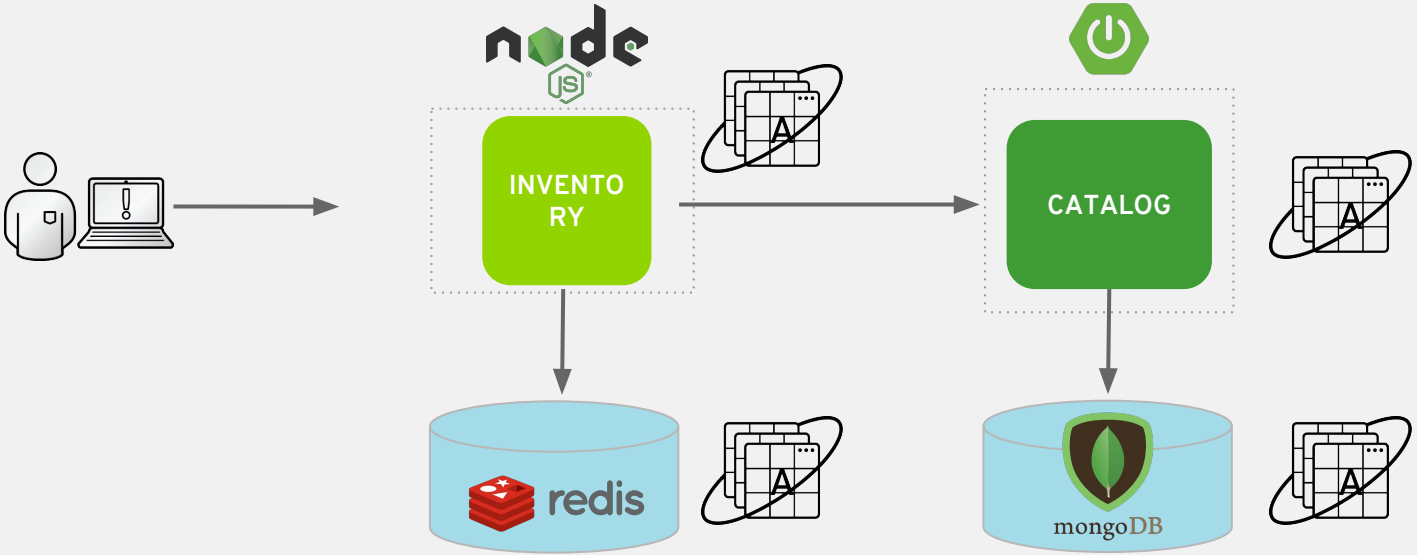
- Make sure you have a userId (userX). Each attendee has its own environment on OpenShift Container Platform
- Fork the GitHub repo https://github.com/lbroudoux/ocp-on-azure-workshop into your own GitHub and clone it in your home directory `/home/userX/` on the bastion
- Open a terminal and login into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

```
$ oc login https://masterdnscbmvtdzhvuqye.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



redhat.

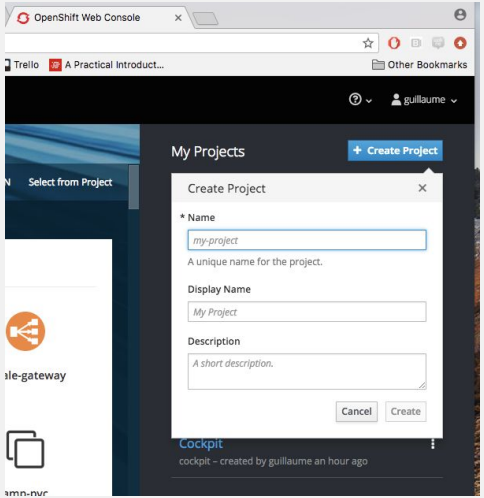
LAB 2

Deploy containers from an image

Create your development environment

Let’s go the Web Console

- Via the web console : https://masterdnscbmvtdzhvuqye.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`



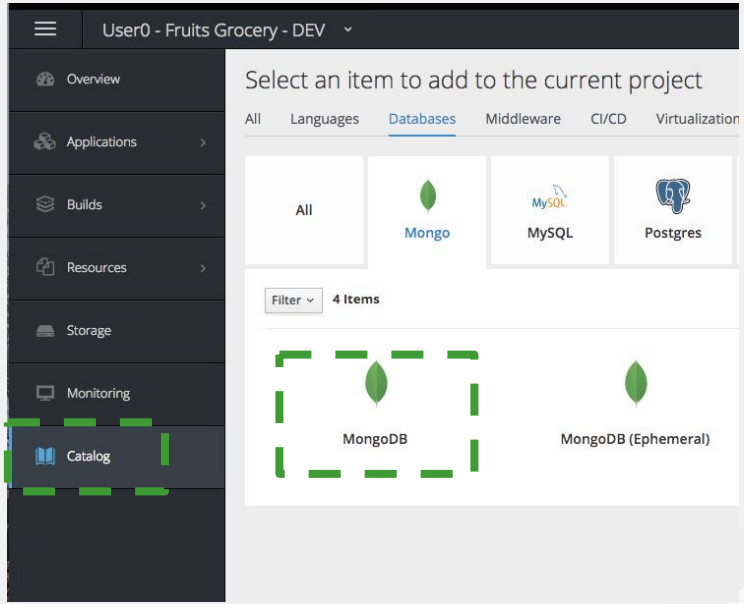
Deploy MongoDB database via the catalog

- Browse the service catalog and search for MongoDB

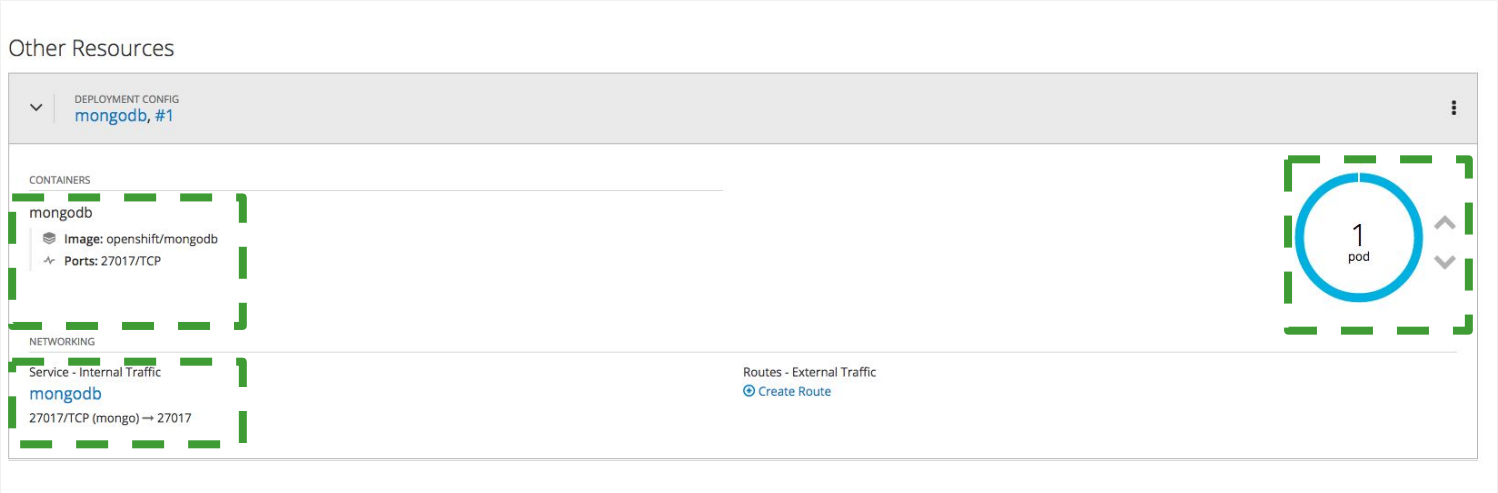
- Set MongoDb Database Name
  - Name : `fruitsdb`

- Save and label the deployment config with the command below

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



Check MongoDB deployment



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

Deploy Redis via the CLI

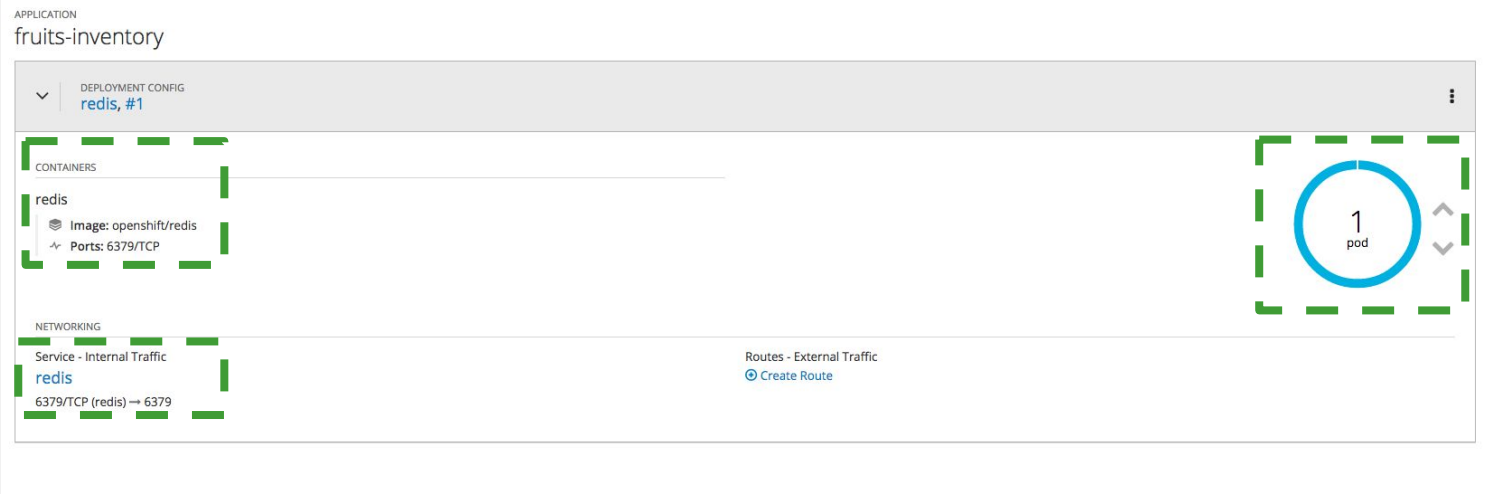
Let's do the deployment of Redis through the CLI rather than the Web console

```
$ 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

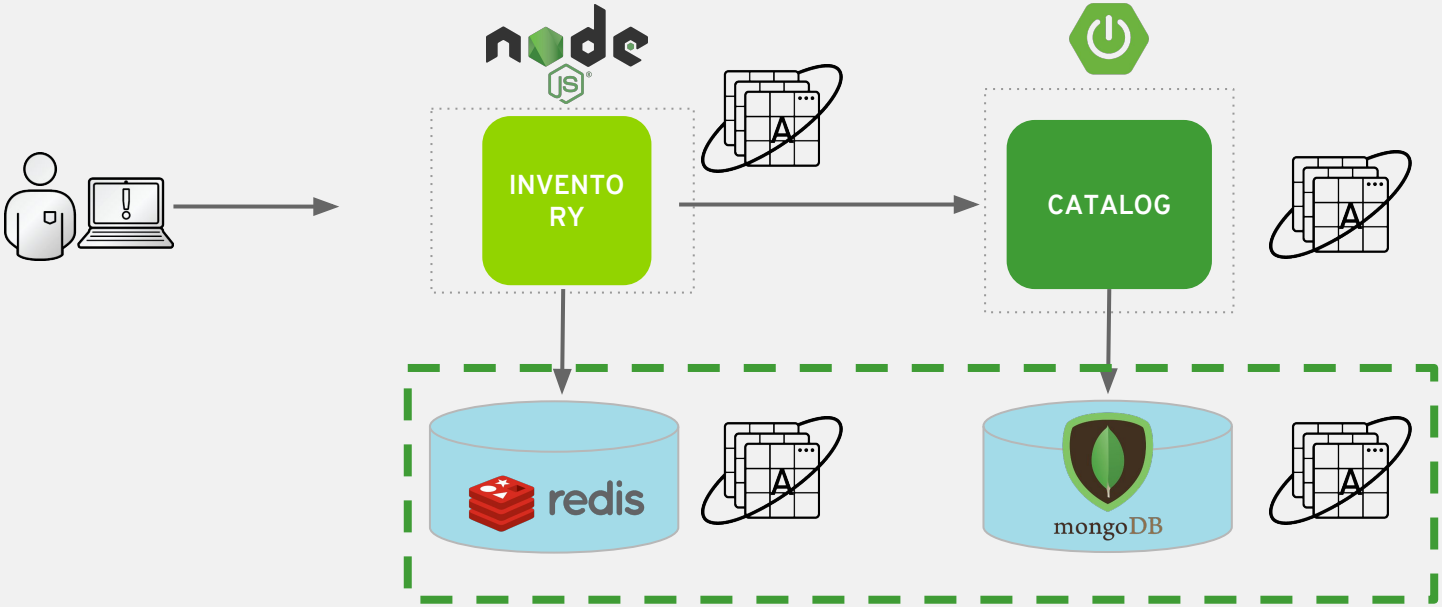
- "redis-persistent" is the template we use from the catalog
- We specify also a label (`app=fruits-inventory`) to select easily all resources related to fruits-inventory in our environment
- `DATABASE_SERVICE_NAME` is the service to reach all pods related to Redis

Check Redis deployment



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

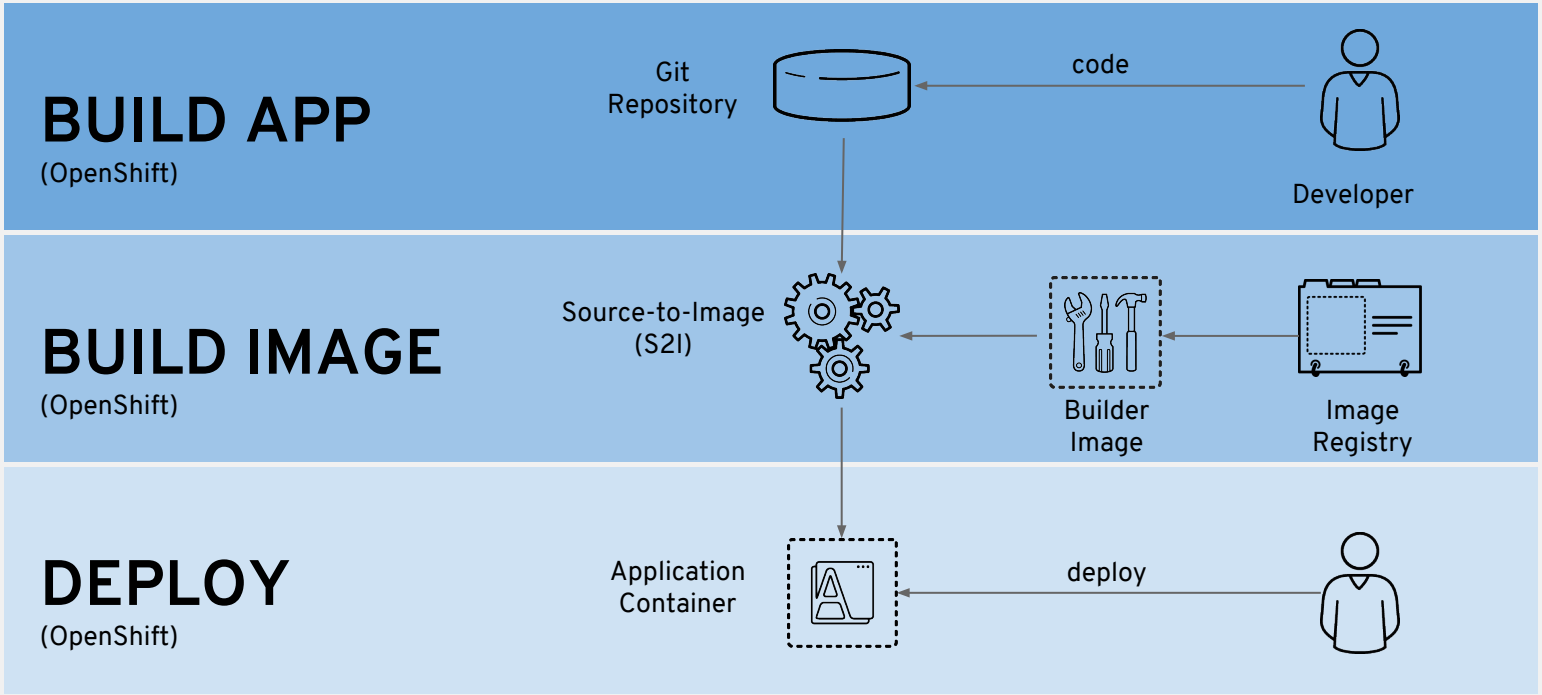
Grocery Store on OpenShift



LAB 3

Deploy containers from source

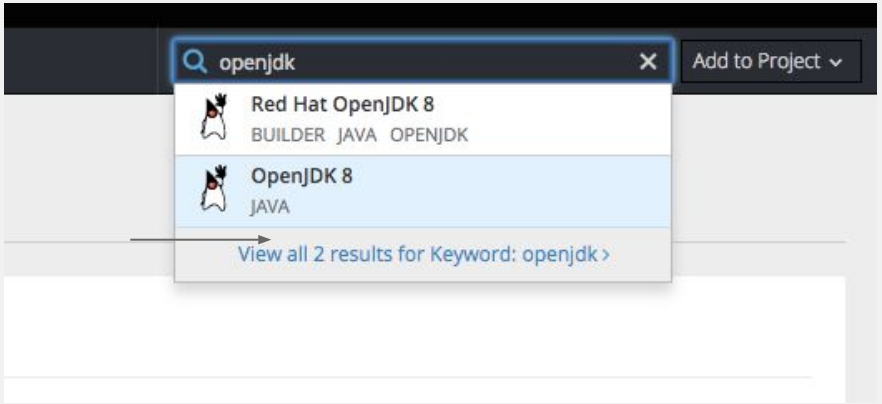
Deploy the fruits catalog with s2i strategy



Deploy the fruits catalog

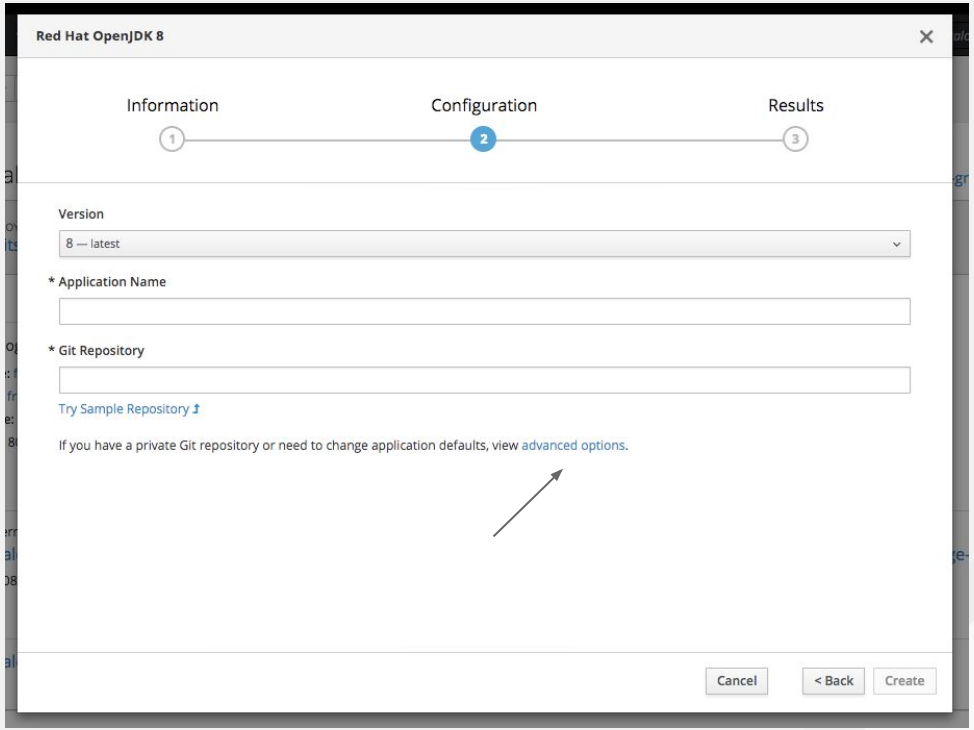
Our application is developed with Spring Boot. A powerful Java framework to build next-gen application and leverage Openshift capabilities.

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



Deploy the fruits catalog

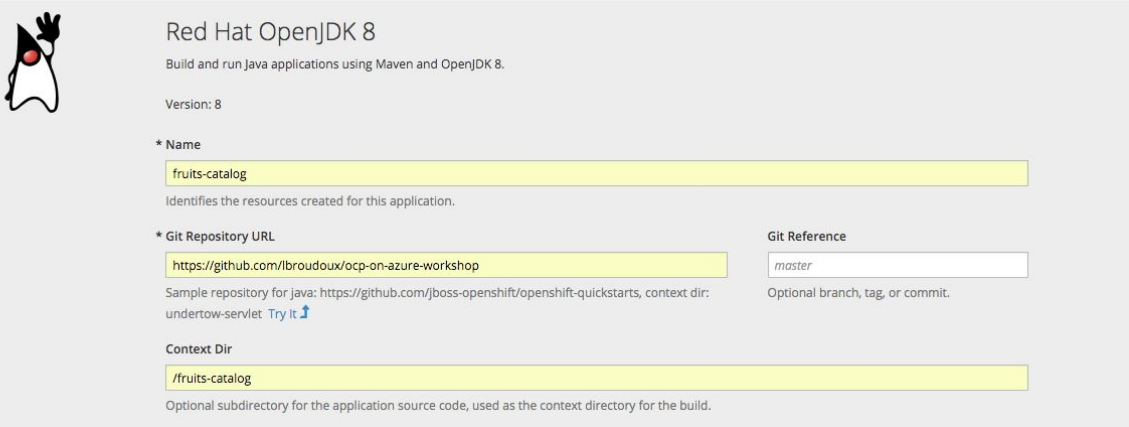
Let’s explore Advanced options to specify environment variables and extras things!



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

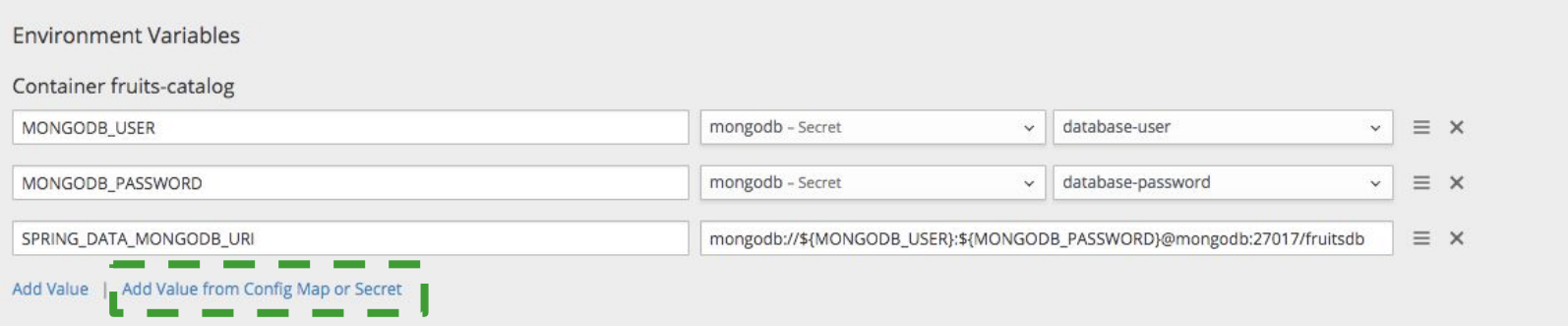


Deploy the fruits catalog

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`



Deploy the fruits catalog

Explore the application resources deployed

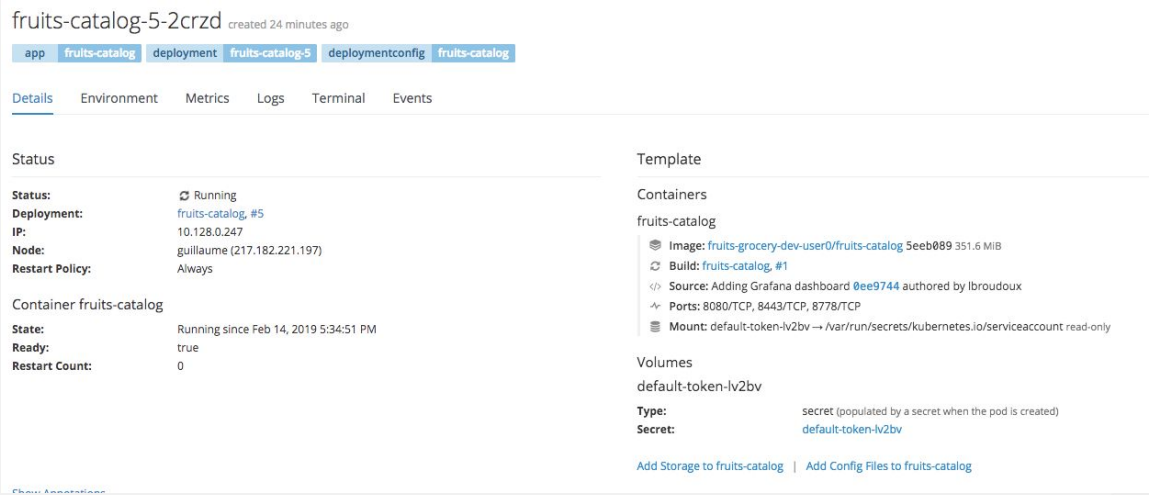
Container Image, Build used, ports, routes ...

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



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"}'
```

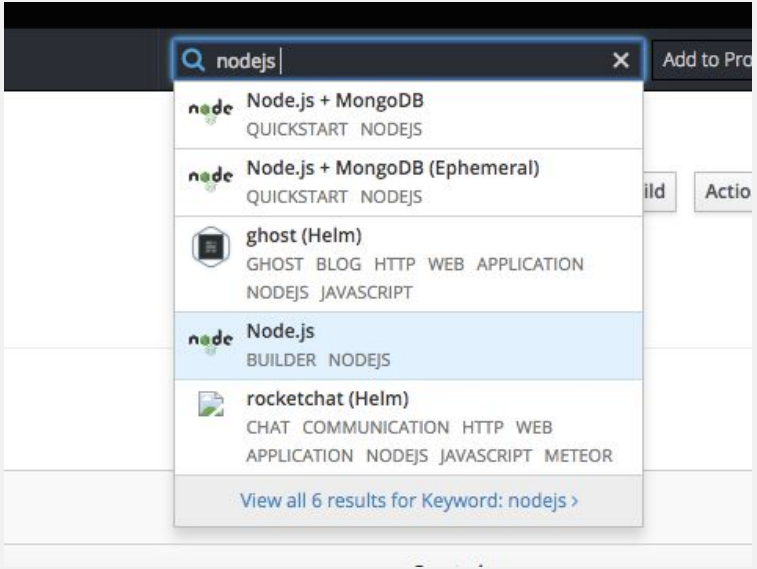
Get all fruits from the fruits-catalog

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

Deploy the Fruits inventory

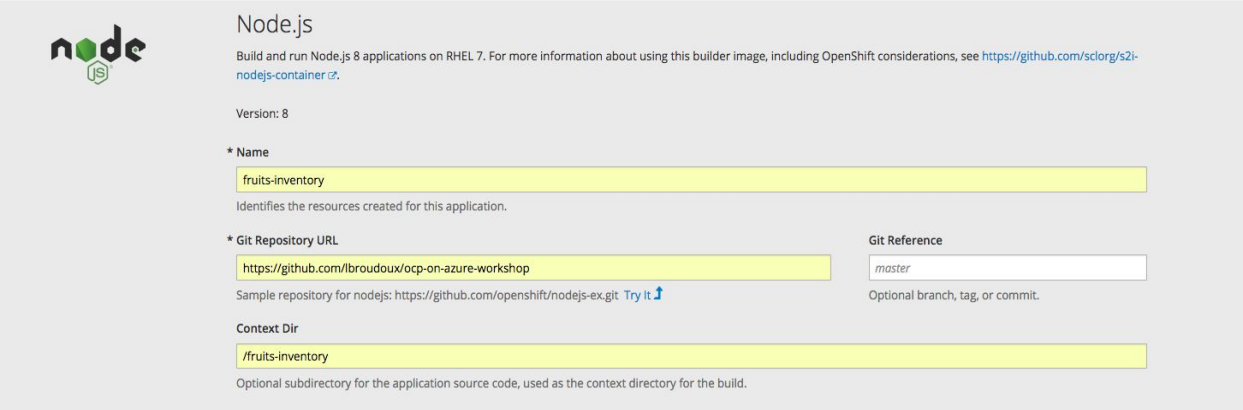
We use the same s2i strategy to build and deploy the app from source code

Choose the Node.JS image builder from the catalog



Deploy the Fruits inventory

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

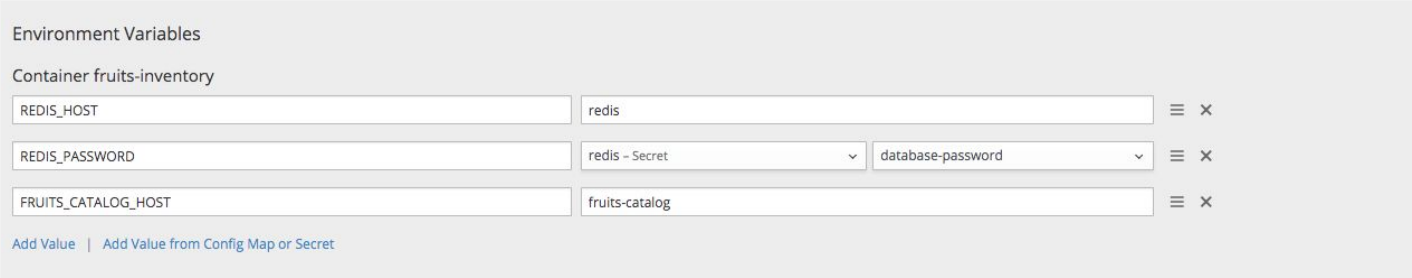


Deploy the Fruits inventory

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



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

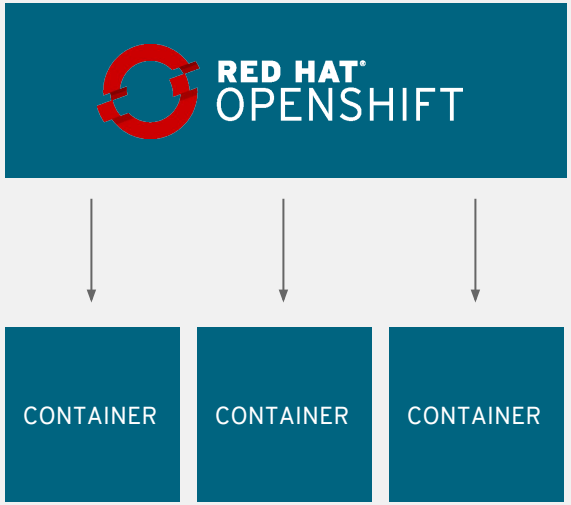
LAB 4

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.

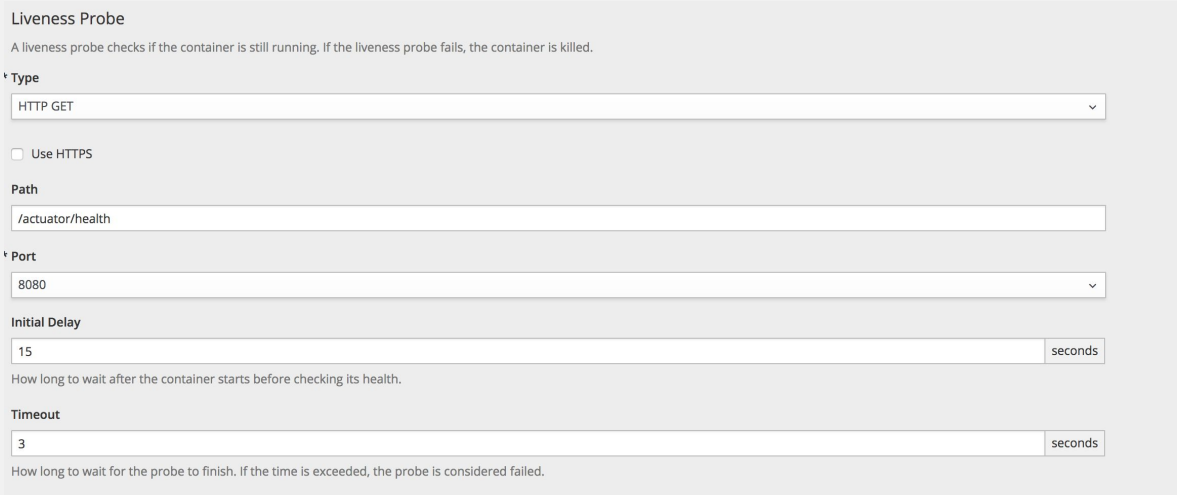
Remove Readiness Probe

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

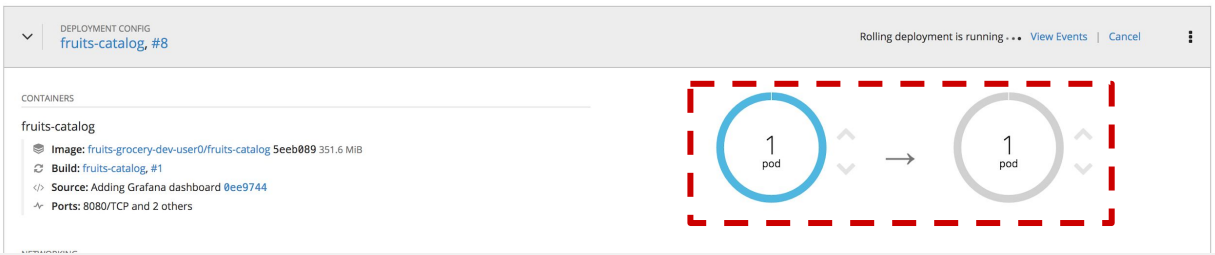
Is the app still running ? We use the same endpoint as the readiness for this example.



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

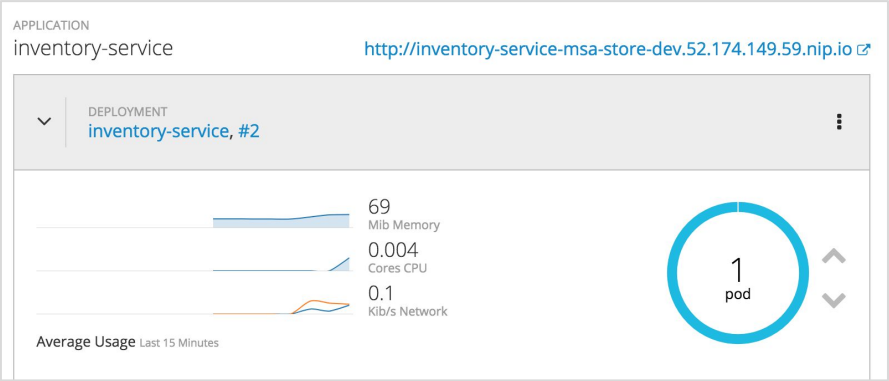
```
$ oc set probe dc/fruits-inventory --liveness --get-url=http://:8080/api/health/liveness
--initial-delay-seconds=60 --period-seconds=30

$ 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

Pods » inventory-service-2-dp8r2

inventory-service-2-dp8r2 created 14 days ago

Actions

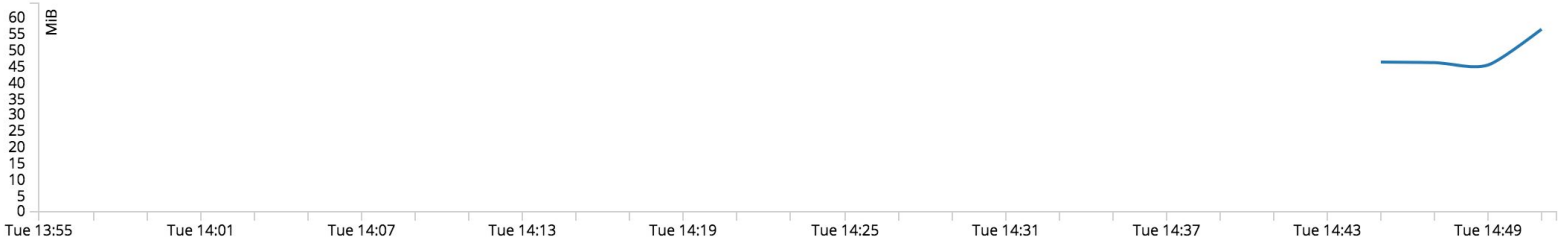
app inventory-service | deployment inventory-service-2 | deploymentconfig inventory-service

Details | Environment | Metrics | Logs | Terminal | Events

Container: inventory-service **Time Range:** Last hour

About Compute Resources

Memory



redhat.

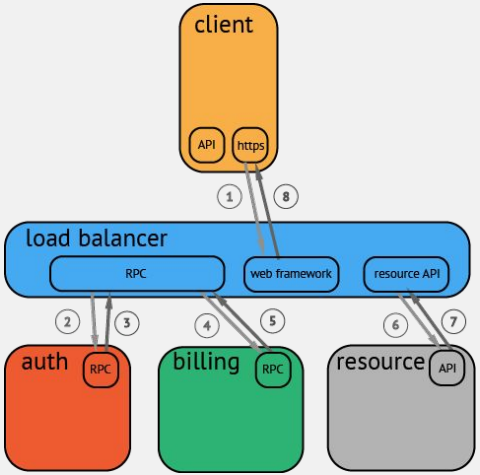
LAB 5

Distributed tracing configuration

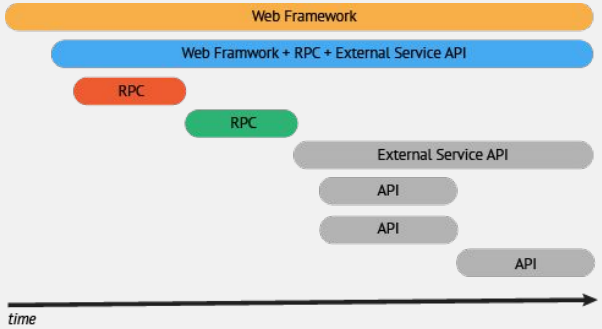
LAB 5: Distributed tracing configuration

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

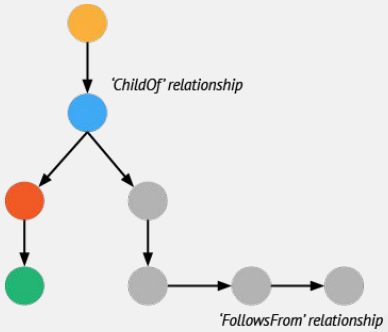
What is distributed tracing ?



Spans

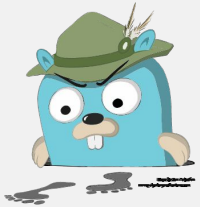


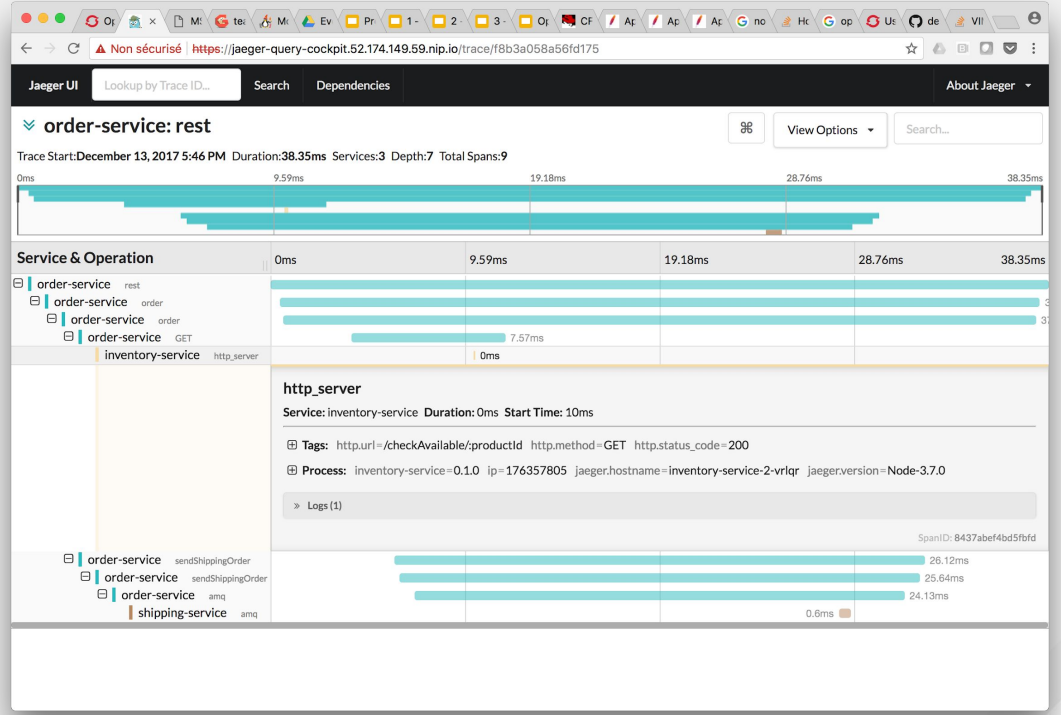
Relationships



Distributed tracing

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





Add Jaeger configuration to fruits-catalog

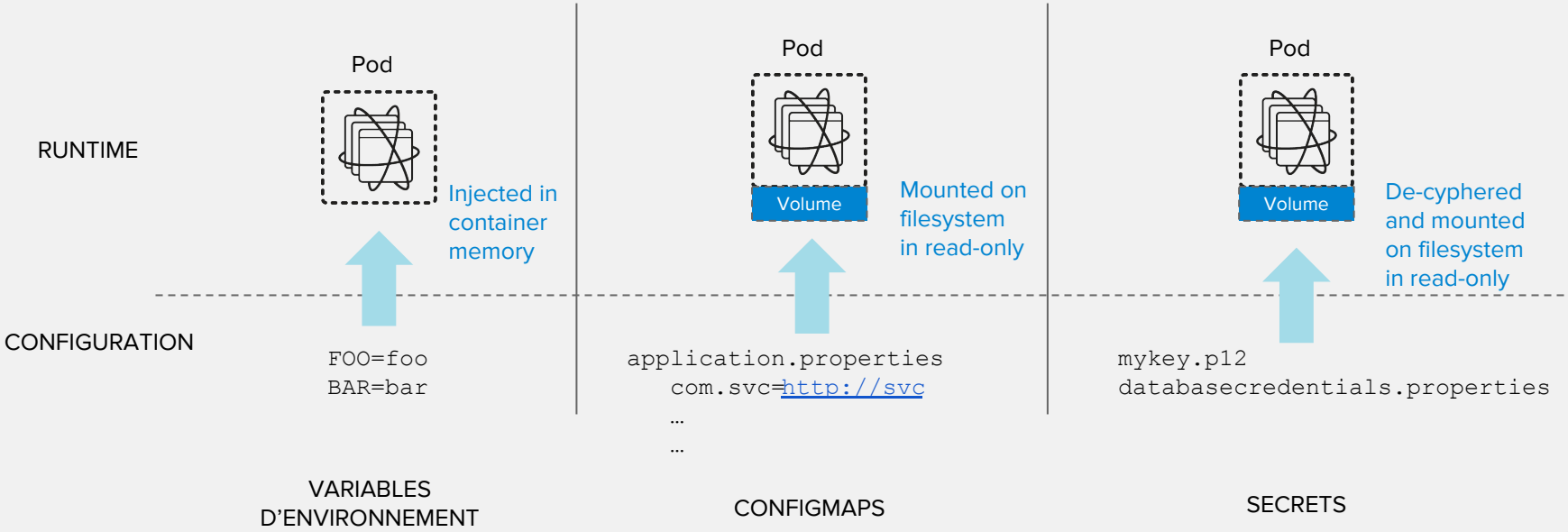
Before setting Jaeger in the fruits-catalog application, we have to add a specific role to the current project to view particular objects, especially ConfigMap ...

```
$ 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

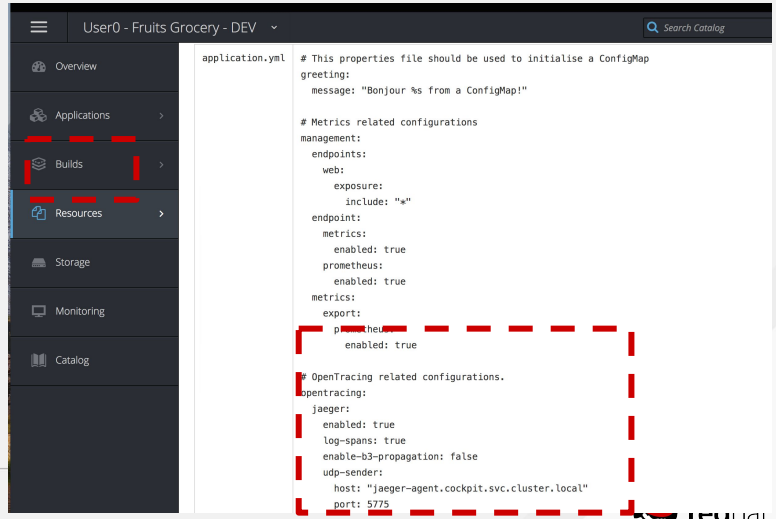
```
$ cd fruits-catalog
$ oc create configmap fruits-catalog-config --from-file=application.yml
```

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.

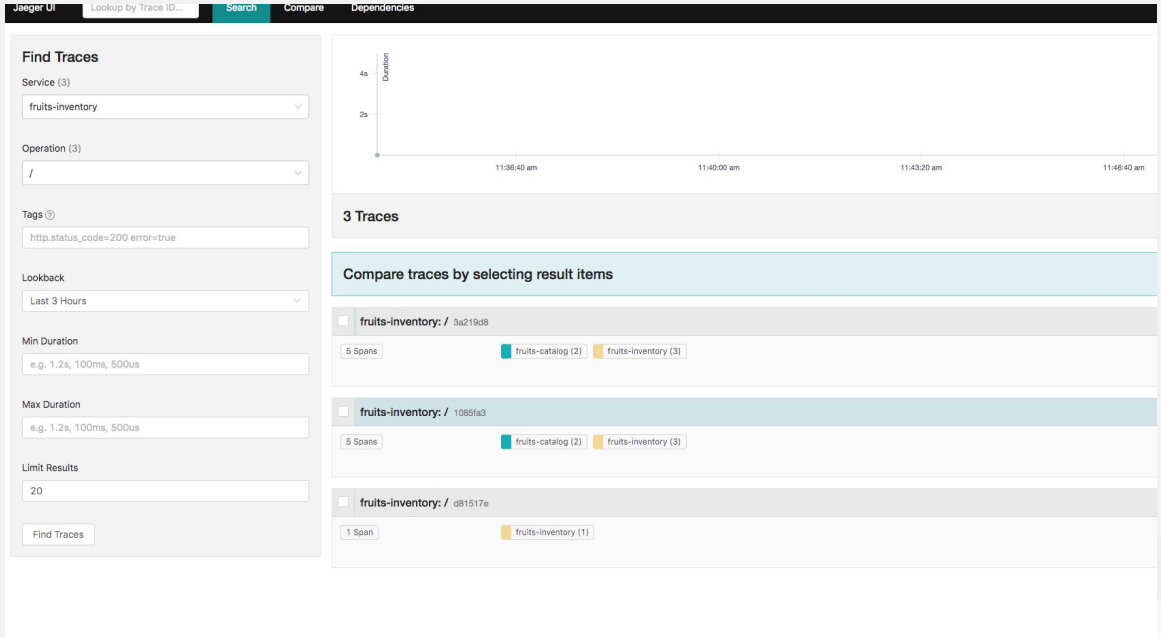
Get all fruits with their stock through the fruits-inventory API and jump to Jaeger to see the detailed trace

```
$ 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`.



Filter the right Jaeger trace

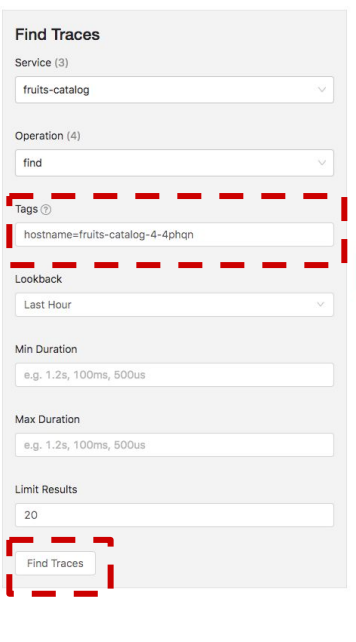
As we use a mutual Jaeger, you need to filter on your pod fruits-catalog hostname

```
$ 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



Explore the Jaeger trace

fruits-inventory: / 3a219d8

Search... | Trace Timeline

Trace Start February 20, 2019 11:49 AM | Duration 11ms | Services 2 | Depth 4 | Total Spans 5

0ms | 2.75ms | 5.5ms | 8.25ms | 11ms

Service & Operation

fruits-inventory /

/ — Service: fruits-inventory | Duration: 11ms | Start Time: 0ms

Tags: sampler.type = const | sampler.param = true | http.method = GET | span.kind = server | http.url = / | http.status_code = 200

Process: client-uuid = e537b64e-77fe-478d-a846-4c6dce2dff7d | fruits-inventory = 1.0.0 | hostname = fruits-inventory-2-g4c6k | ip = 10.131.2.7 | jaeger.version = Node-3.14.4

Logs (2)

SpanID: 3a219d8215c231f6

fruits-inventory fruits-catalog invocation — 9ms

fruits-catalog listFruits — 4.93ms

listFruits — Service: fruits-catalog | Duration: 4.93ms | Start Time: 3ms

Tags: http.status_code = 200 | http.url = http://fruits-catalog:8080/api/fruits | component = java-web-servlet | span.kind = server | http.method = GET

Process: hostname = fruits-catalog-4-4phqn | ip = 10.128.2.7 | jaeger.version = Java-0.32.0

Logs (4)

SpanID: acca53768a21723b

fruits-catalog find — 0.81ms

find — Service: fruits-catalog | Duration: 0.81ms | Start Time: 5ms

Tags: db.instance = fruitsdb | component = java-mongo | db.type = mongo | span.kind = client | db.statement = { "find" : "fruit" } | peer.hostname = mongodb | peer.port = 27017 | peer.ipv4 = 172.30.66.171

Process: hostname = fruits-catalog-4-4phqn | ip = 10.128.2.7 | jaeger.version = Java-0.32.0

SpanID: e401091f2e038527

fruits-inventory redis invocation — 1ms

redis invocation — Service: fruits-inventory | Duration: 1ms | Start Time: 9ms

Tags: http.url = http://redis:6379 | http.method = GET | span.kind = client | http.status_code = 200

Process: client-uuid = e537b64e-77fe-478d-a846-4c6dce2dff7d | fruits-inventory = 1.0.0 | hostname = fruits-inventory-2-g4c6k | ip = 10.131.2.7 | jaeger.version = Node-3.14.4

Logs (1)

SpanID: cdb54fdf75e5c415

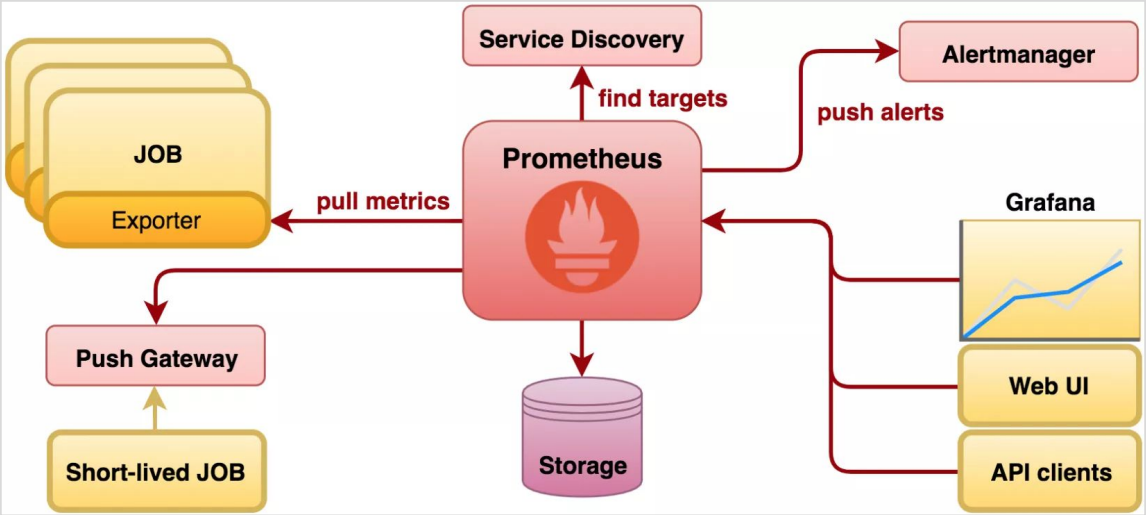
LAB 6

Getting application metrics

LAB 6: GETTING APPLICATION METRICS

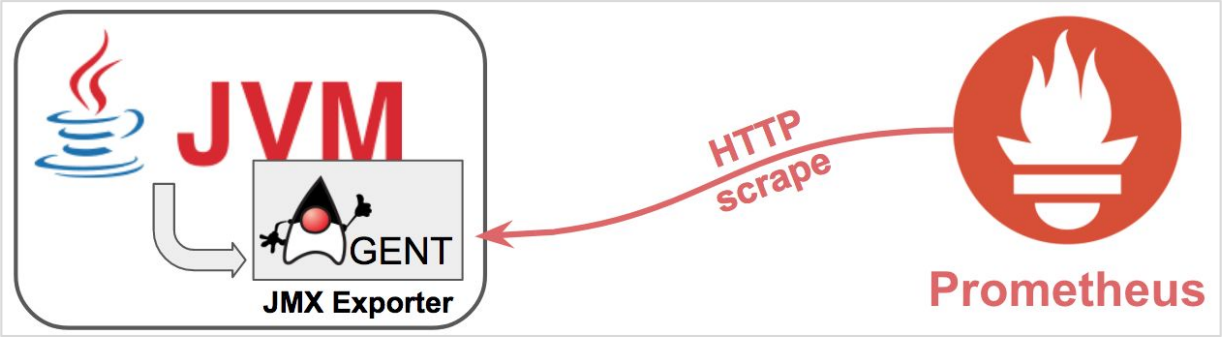
- Update Prometheus configuration
- Add Prometheus datasource in Grafana

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

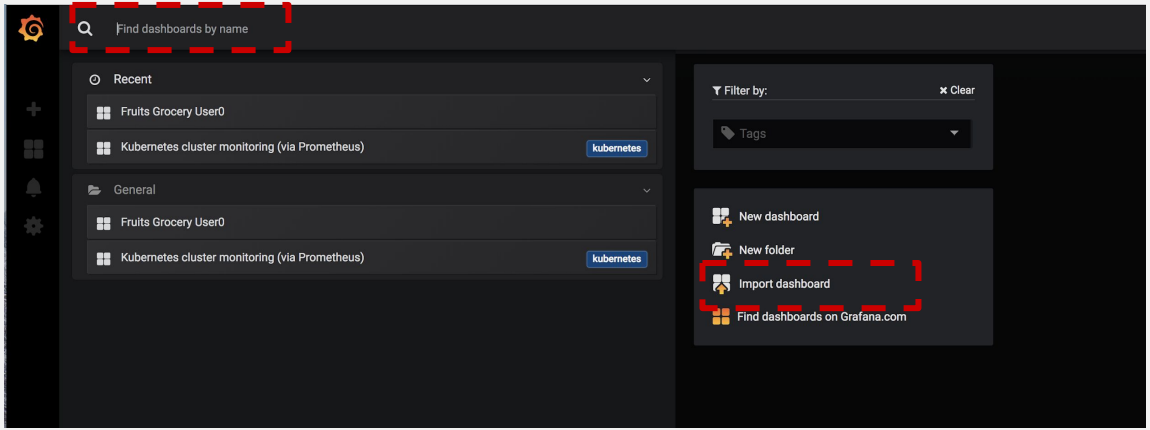
```
$ oc annotate service/fruits-catalog prometheus.io/scrape=true
prometheus.io/path=/actuator/prometheus prometheus.io/port=8080

$ 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

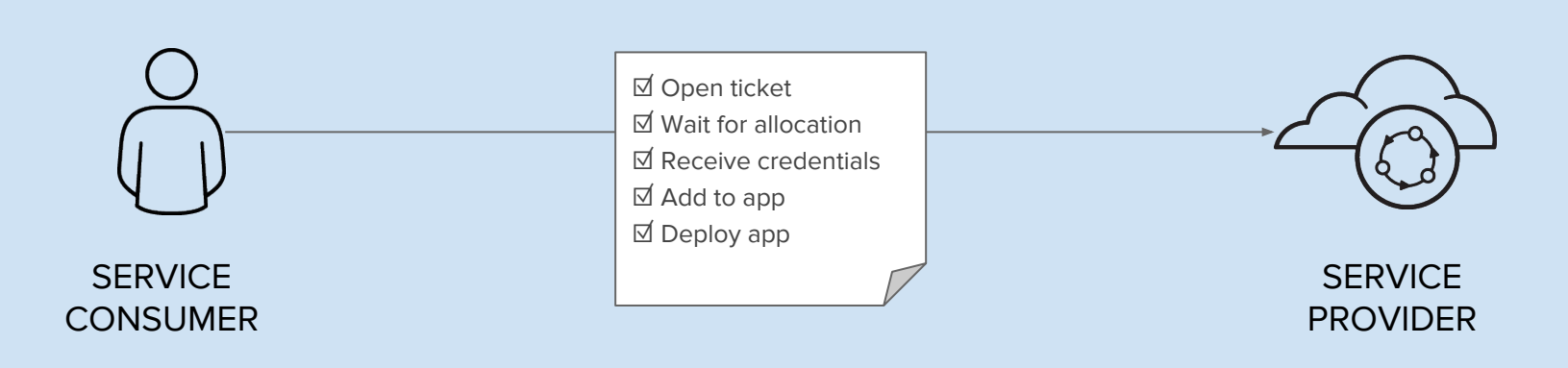
Grafana Dashboard example

LAB 7

Azure Service Broker

RED HAT®
OPENSHIFT
Container Platform

Why a service broker ?



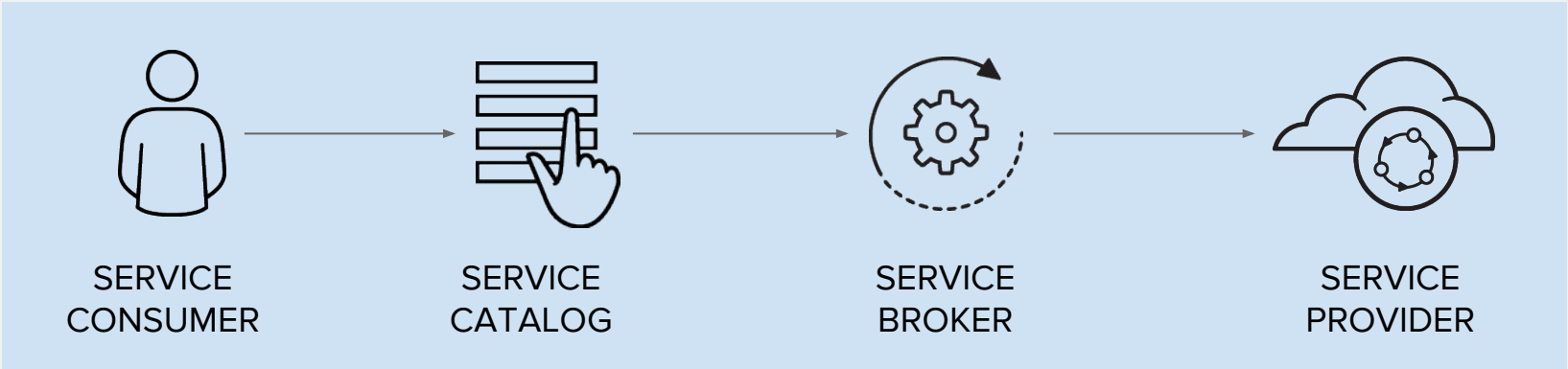
Manual, Time-consuming and Inconsistent

OPEN SERVICE BROKER API™

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

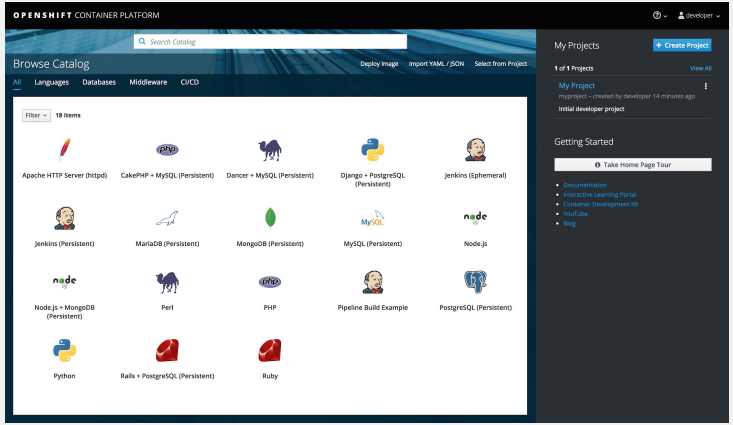
FUJITSU Pivotal IBM redhat Google SAP

What is a service broker ?

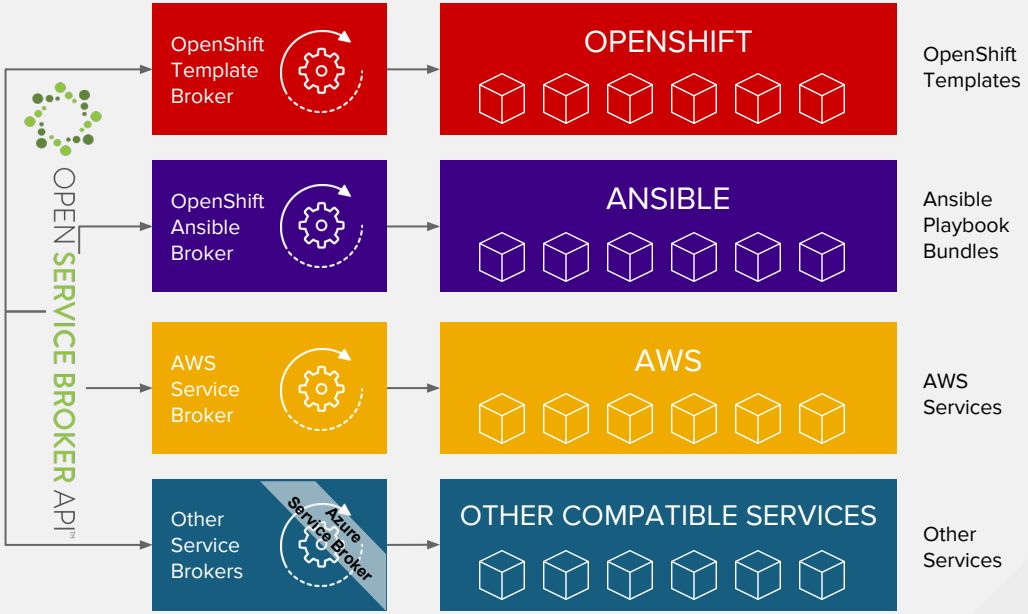


Automated, Standard and Consistent

OpenShift service catalog



OPENSHIFT SERVICE CATALOG



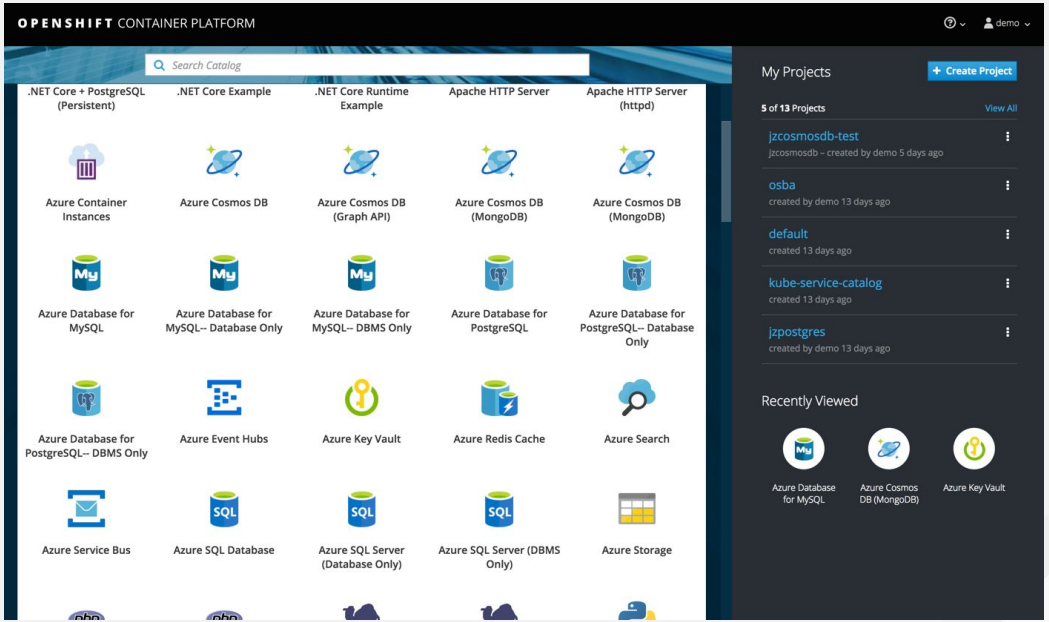
redhat.

OPEN SERVICE BROKER AZURE

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

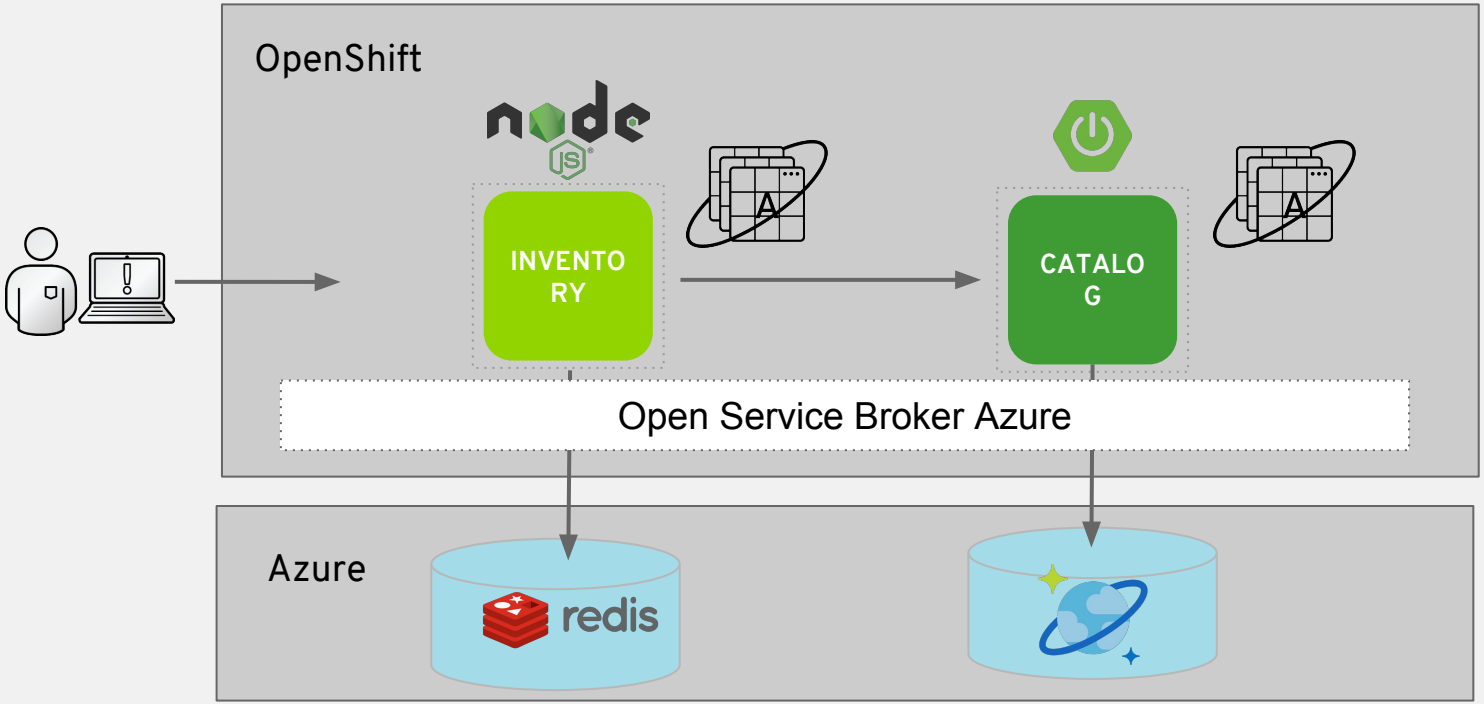
Supported services

- 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



redhat.

Grocery Store on OpenShift and Azure

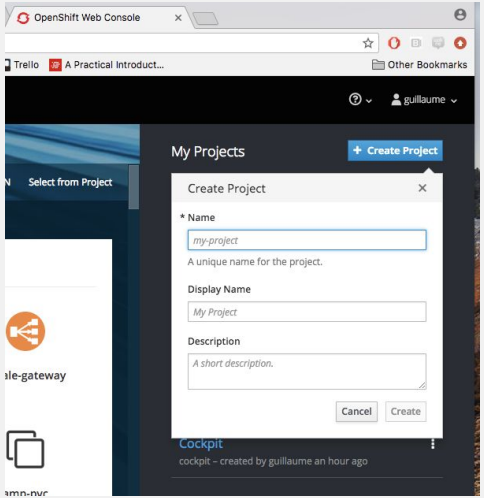


redhat.

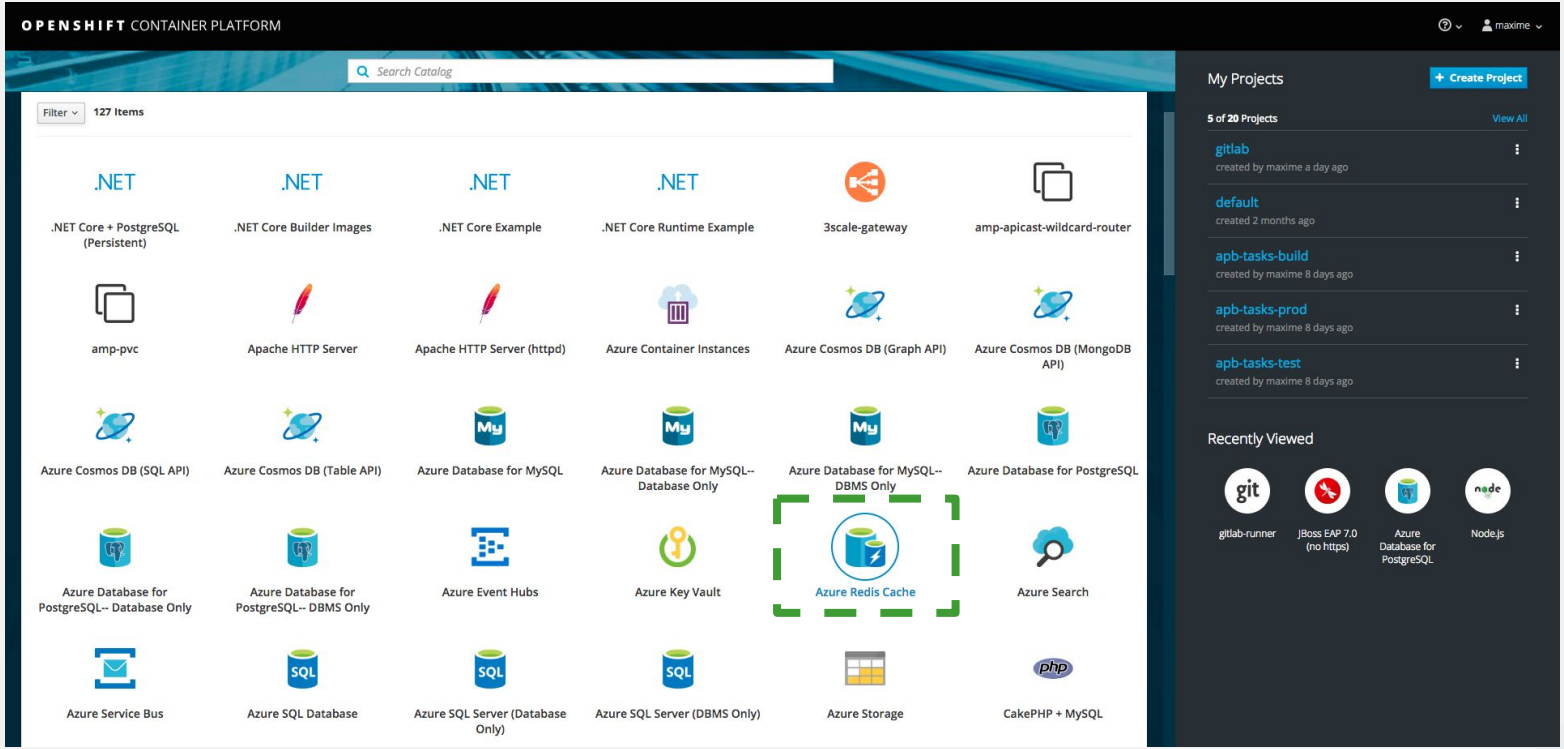
Create your production environment

Let’s go the Web Console

- Via the web console : https://masterdnscbmvtdzhvuqye.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

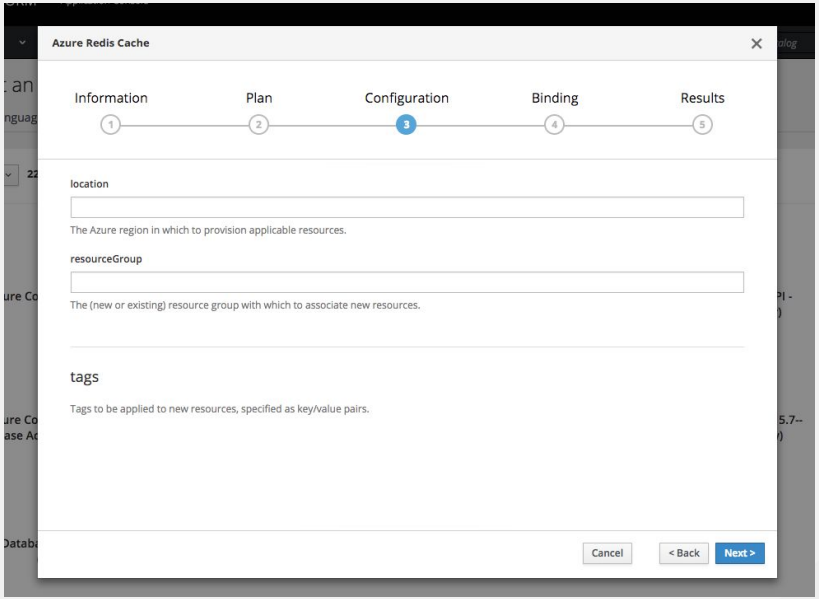


redhat.

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

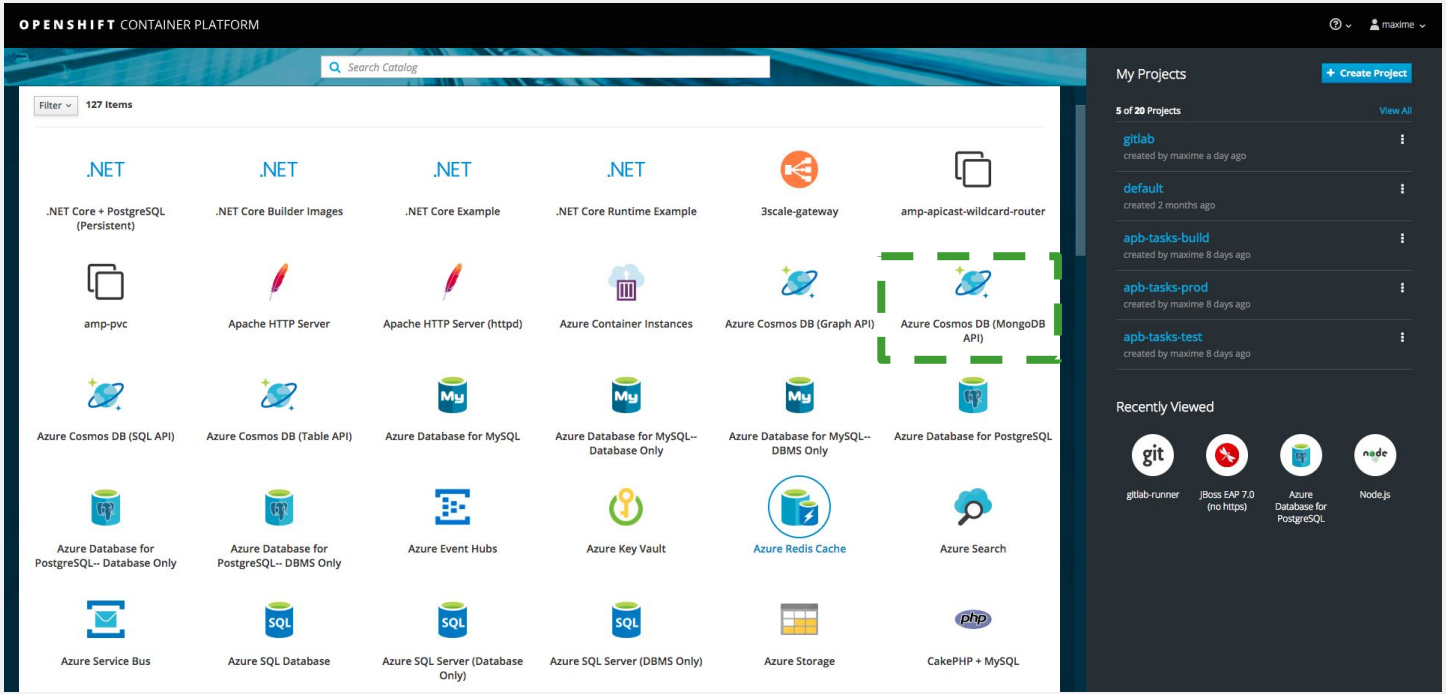
A Redis Cache instance has been provisioned in Azure through the Azure Service Broker

Filter by name... | All types | All locations | No grouping

2 items Show hidden types

NAME	TYPE	LOCATION
36fc4c0c-5b9b-4c8a-a213-02d8c1f3884e	Redis Cache	East US
b7ced0ba-2b7c-4dd3-9ece-6b76457062f0	Azure Cosmos DB account	East US

Deploy a Cosmo DB instance with OSBA



redhat.

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-rediscache-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-rediscache-clr54-kmdk8
created an hour ago

Pending

Delete

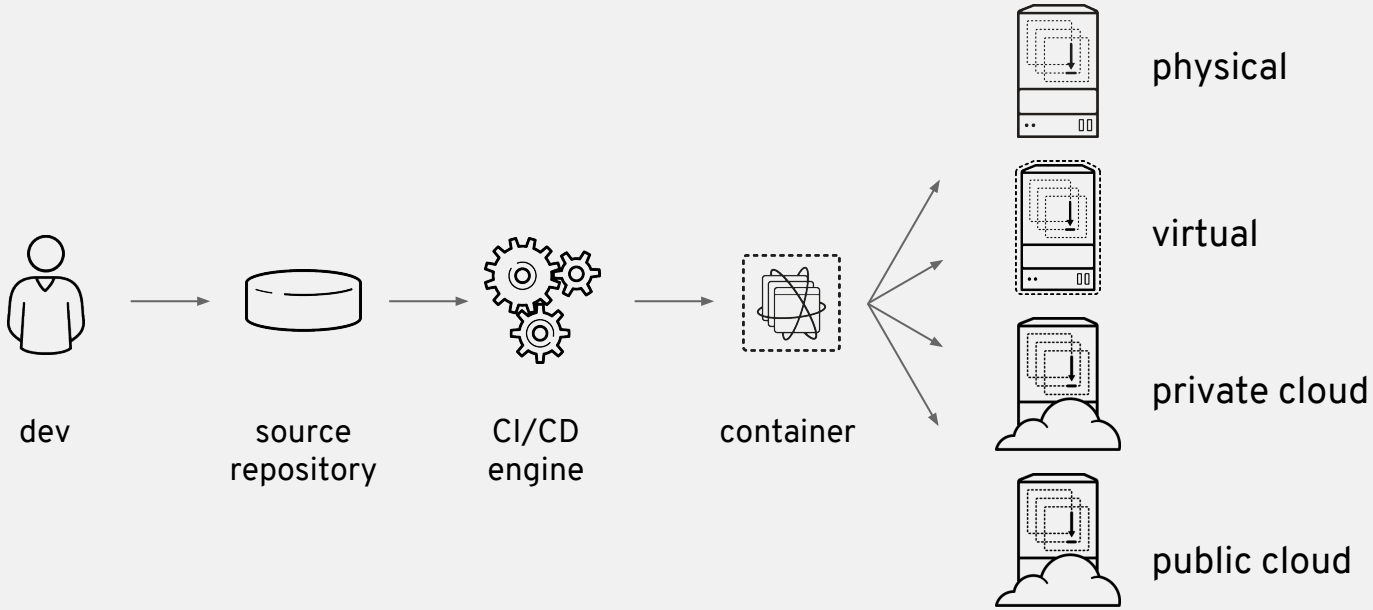
LAB 8

Continuous delivery

LAB 8: Automating Deployments Using Tags and Pipelines

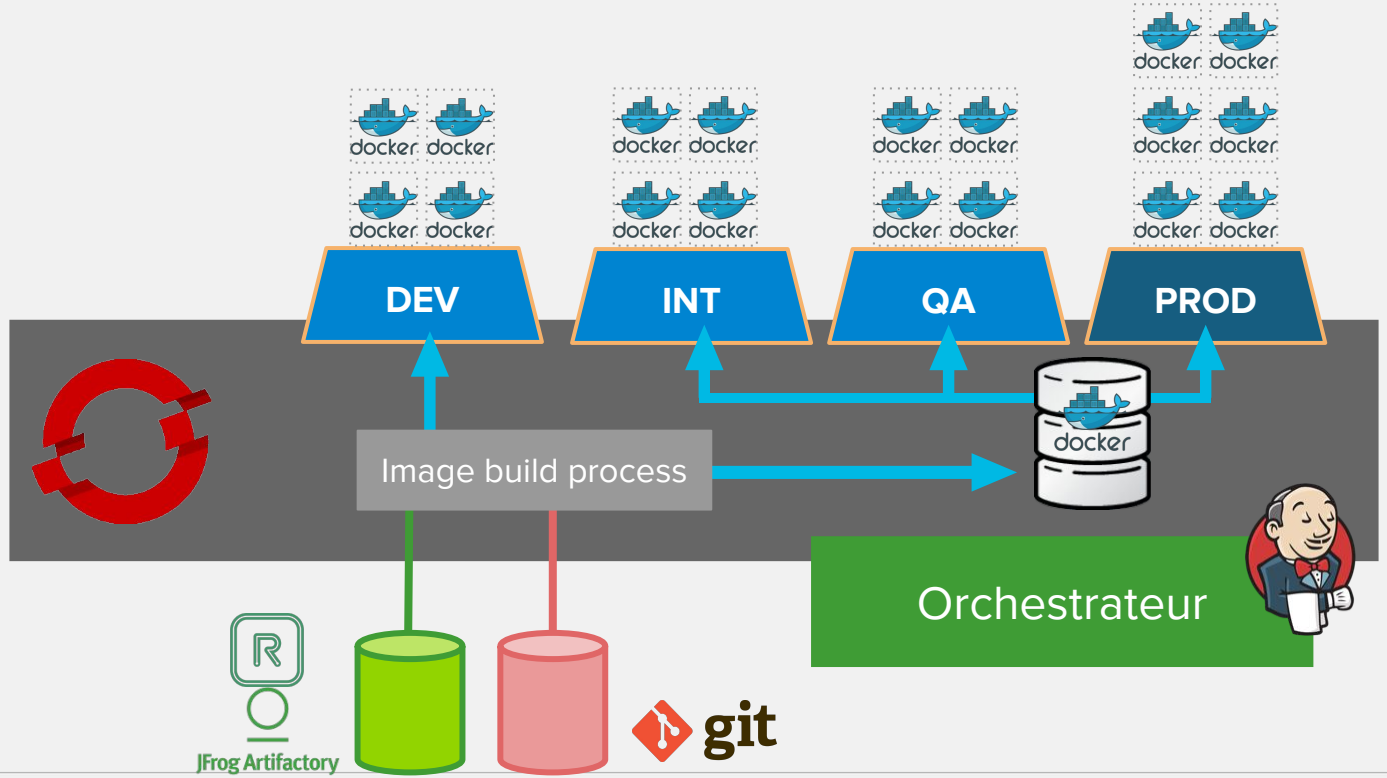
- Prepare a Production environment
- Explore the deployment configurations
- Promote images to production
- Create an OpenShift Jenkins Pipeline
- Add a Webhook to run the pipeline on every code change
- Change some code and review

Deployment pipeline



CI/CD with OpenShift

Rolling Upgrades
Blue/Green Deployments
A/B Testing



redhat.

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
```

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

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

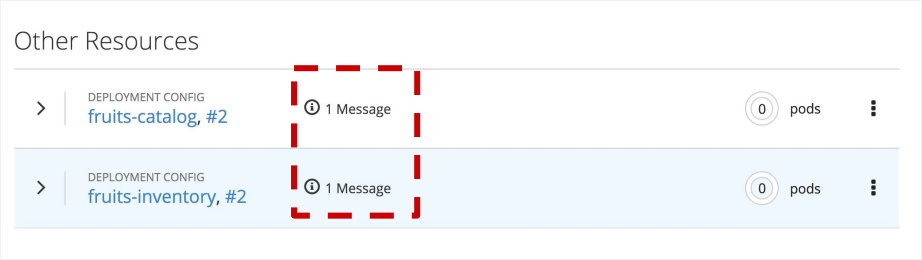
Prepare a Production environment

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

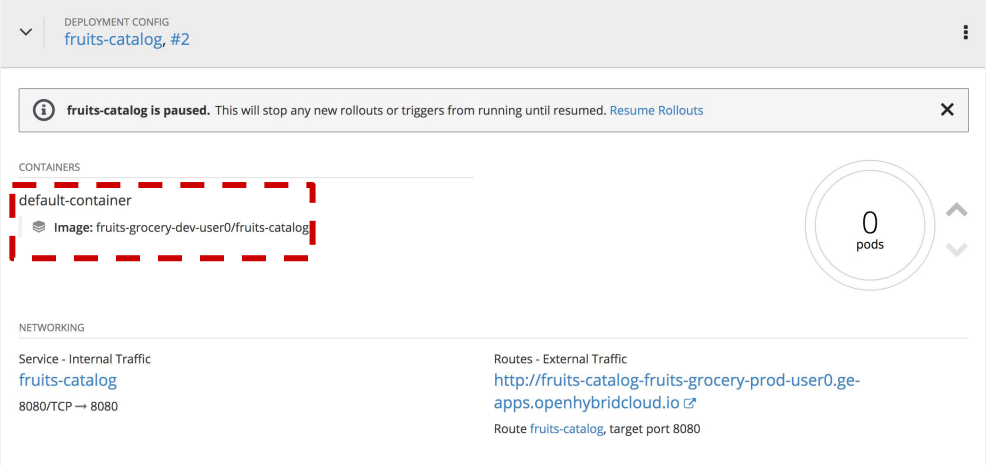
From overview on web console, check the deployment configuration All deployment are cancelled.



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 !

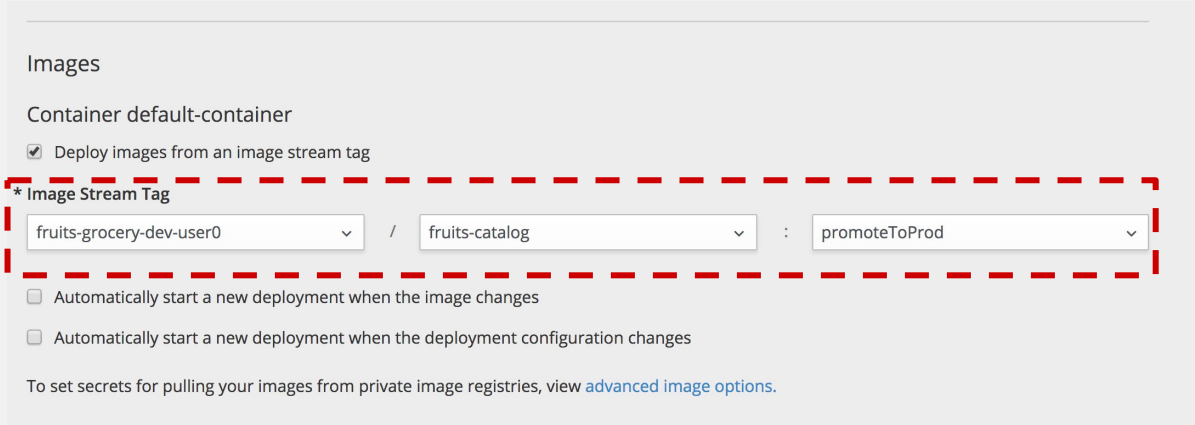


Explore the deployment configurations

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.

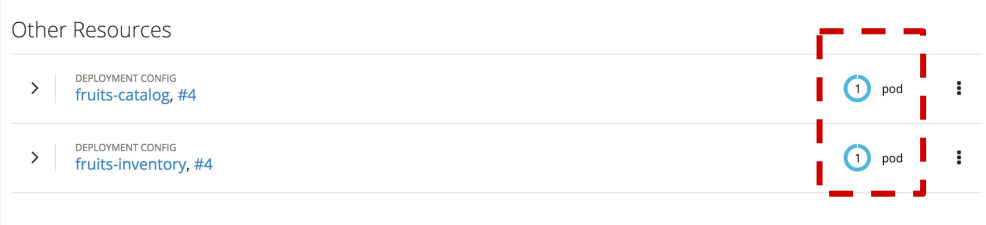
```
$ . /home/userX/deploy-prod.sh

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 !

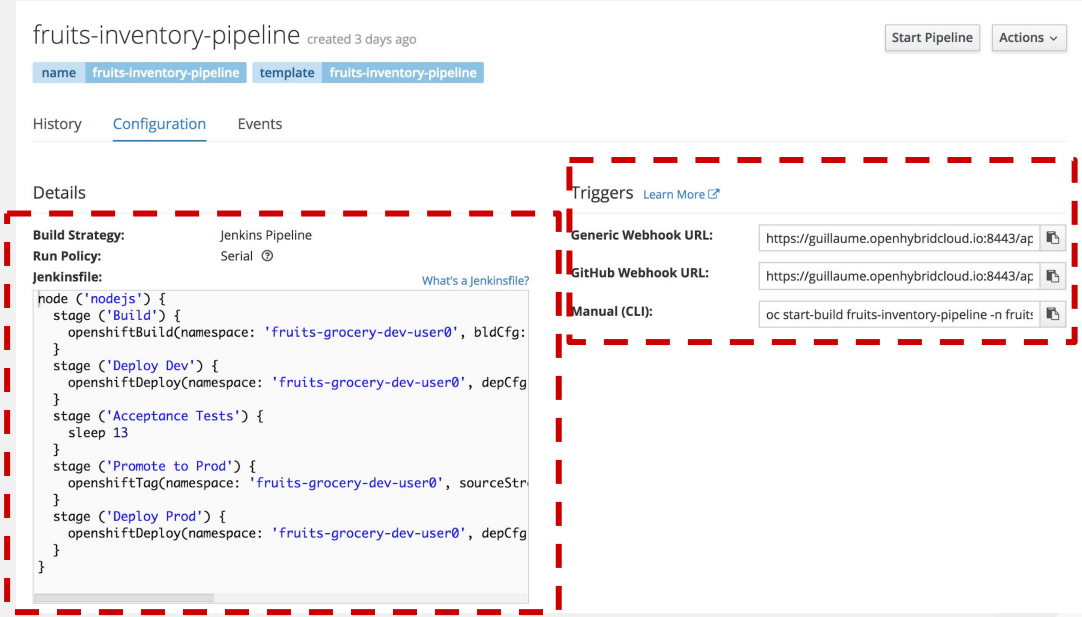
But wait … we have also created a pipeline. Just go to your development project.



Create an OpenShift Jenkins Pipeline

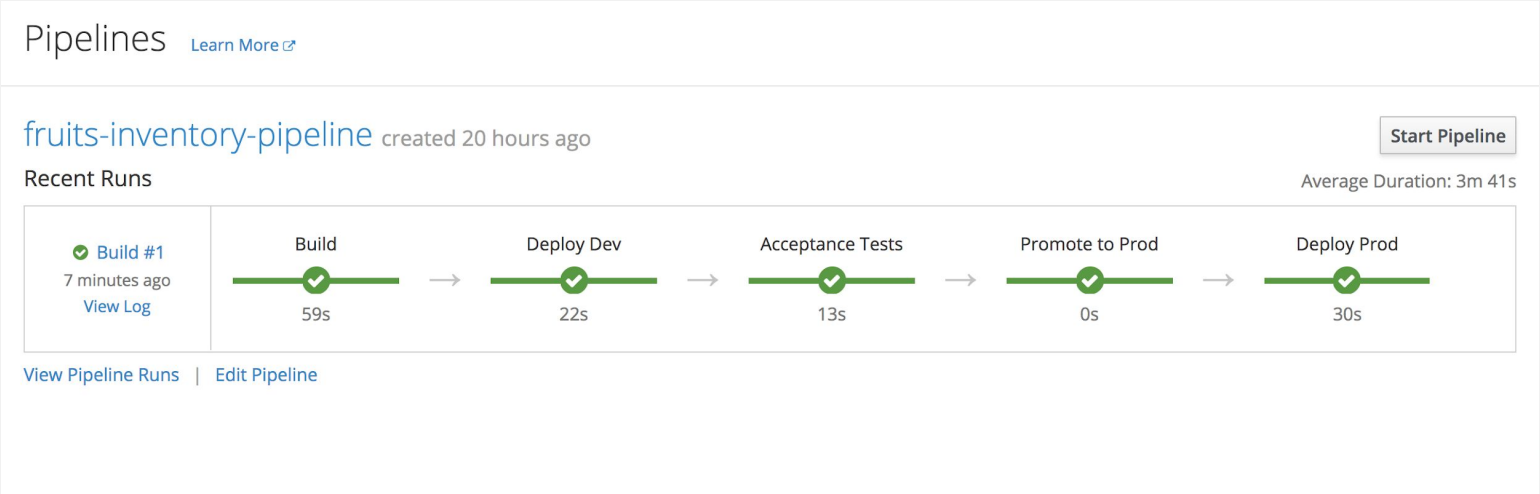
In your development project within the **Builds** section, **Pipelines** subsection, check that inventory-service-pipeline has been created.

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



Start your Jenkins pipeline

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



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

CONGRATULATIONS !

YOU’RE A CLOUD-NATIVE APPS DEVELOPER.

LEARN.OPENSHIFT.COM

OpenShift: Interactive Learning

Sécurisé | https://learn.openshift.com/introduction/

OPENSHIFT

Report an Issue

SIGN UP TO OPENSHIFT ONLINE FOR FREE

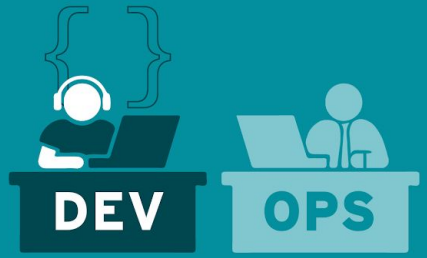
RED HAT OPENSHIFT

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

redhat.



THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos

redhat.