# Documentación Openshift Udemy

Usuario: jose

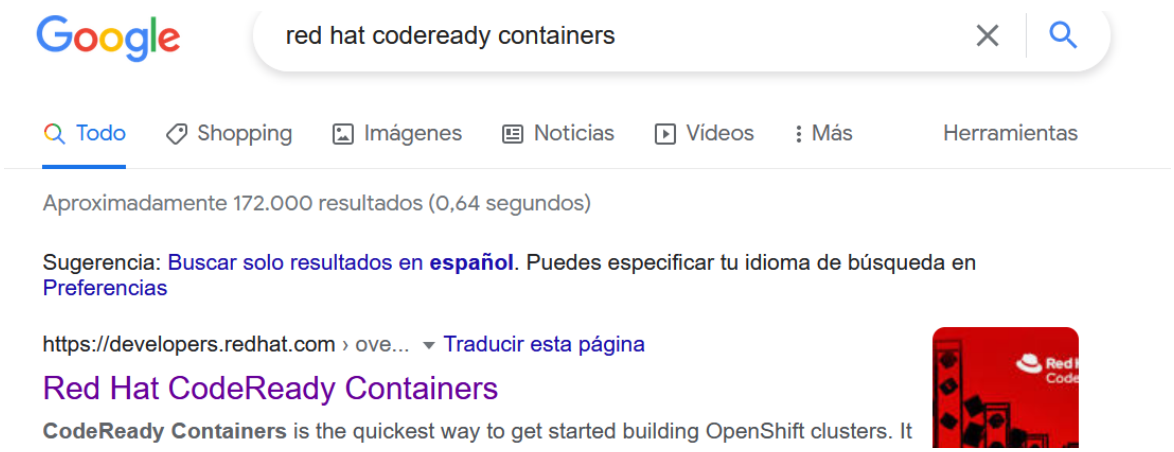
Contraseña: jose123

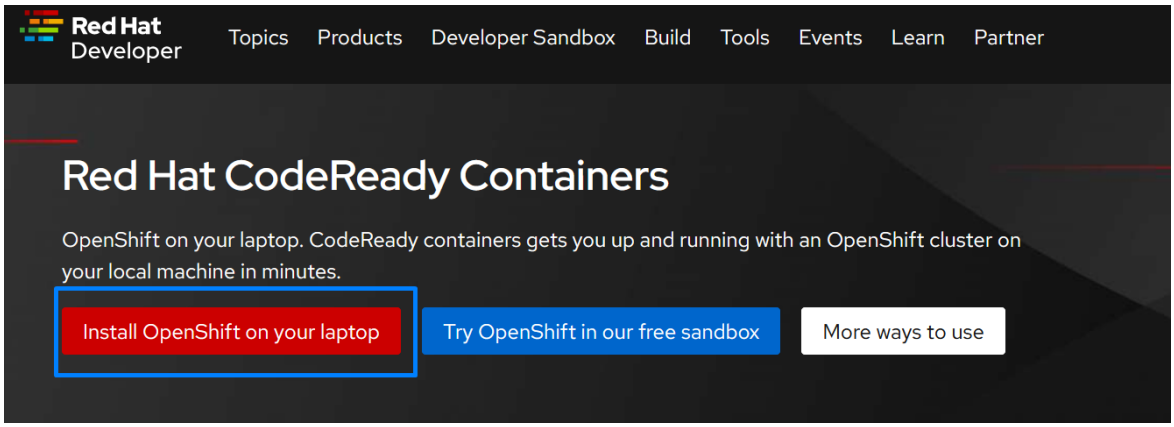
Máquina con mínimo 8GB de RAM y 40 GB de disco, S.O que voy a usar CentOS

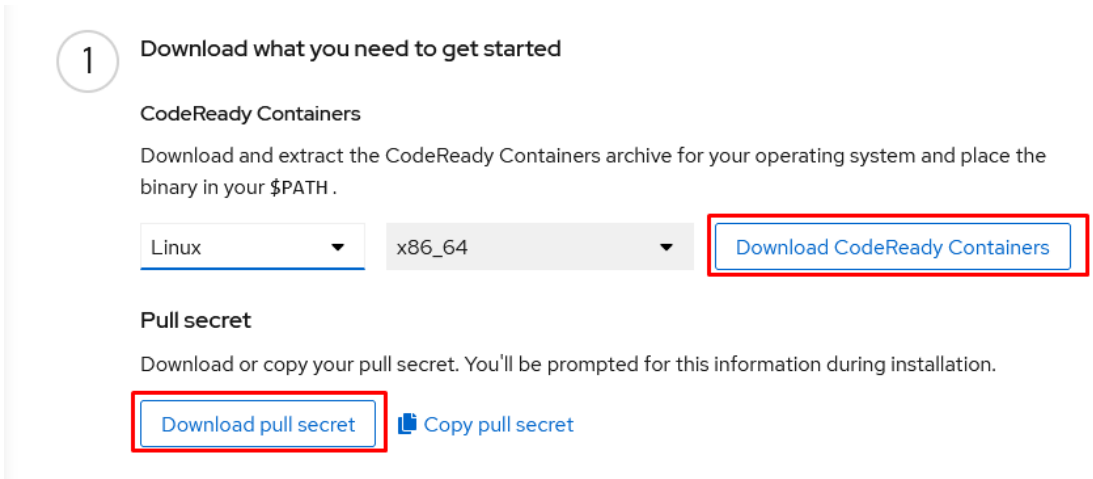
Es necesario activar la virtualización y añadirle algún Core a la máquina

Error virtualización: https://www.youtube.com/watch?v=dkMWvENtGBk

## Instalar y preparar RedHat CodeReady Containers



A continuación, instalamos estos dos paquetes desde el root

```
[jose@jose ~]$ su -
Contraseña:
Último inicio de sesión:jue abr 21 16:33:22 CEST 2022en pts/0
[root@jose ~]# yum install NetworkManager
[root@jose ~]# yum install libvirt
```

IMPORTANTE SALIR DE ROOT

Ahora descomprimimos el archivo descargado en el home

```
[jose@jose ~]$ tar xvf /home/jose/Descargas/crc-linux-amd64.tar.xz
```

Lo renombramos

```
[jose@jose ~]$ ls crc*
crc  LICENSE
[jose@jose ~]$ mv crc-linux-2.0.1-amd64/ crc
```

Yo la voy a mover al home

```
[jose@jose ~]$ cd crc/
```

Antes de empezar vamos a darle permisos de root a nuestro usuario

```
[root@jose ~]# cd /etc/
[root@jose etc]# nano sudoers
```

```
##
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
jose    ALL=(ALL)       ALL
```

Salir del root y añadir el PATH a bashrc

```
[jose@jose ~]$ nano .bashrc
```

```
# User specific aliases and functions
export PATH=$PATH:/home/jose/crc
```

Una vez hecho esto salimos del root y seguimos con el siguiente comando

```
[jose@jose crc]$ ./crc setup
```

A continuación (El pull secret debe estar en el home)

```
[jose@jose ~]$ crc start -p ~/pull-secret
```

Es aconsejable guarda los datos en un txt que nos da al final el start (Yo lo he llamado datos.txt)

```
Started the OpenShift cluster.

The server is accessible via web console at:
  https://console-openshift-console.apps-crc.testing

Log in as administrator:
  Username: kubeadmin
  Password: 3Tmx6-Pw7n3-7Q7ez-DZPYN

Log in as user:
  Username: developer
  Password: developer

Use the 'oc' command line interface:
  $ eval $(crc oc-env)
  $ oc login -u developer https://api.crc.testing:6443
```

# Configurar el cliente OC

Debemos de ejecutar los siguientes comandos

```
[jose@jose ~]$ crc oc-env
export PATH="/home/jose/.crc/bin/oc:$PATH"
# Run this command to configure your shell:
# eval $(crc oc-env)
[jose@jose ~]$ eval $(crc oc-env)
```

Así ya tendríamos acceso al cliente oc

```
oc status
```

oc → muestra una lista de comandos

```
oc version
```

Para que funcione correctamente tenemos que logarnos y hacer algunas configuraciones

```
[jose@jose ~]$ oc login -u developer -p developer https://api.crc.testing:6443
```

```
[jose@jose ~]$ crc config set pull-secret-file /home/jose/pull-secret
Successfully configured pull-secret-file to /home/jose/pull-secret
```

```
[jose@jose ~]$ nslooku api.crc.testing

bash: nslooku: no se encontró la orden...
[jose@jose ~]$ nslookup api.crc.testing
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   api.crc.testing
Address: 192.168.130.11

[jose@jose ~]$ crc ip
192.168.130.11
```

Nos logamos con el admin

Probamos los siguientes comandos

oc get nodes

oc get nodes -o wide

crc -console --credentials

## Acceder a la consola web (Gráfica)

crc console --url

crc console

## Crear un proyecto (KUBERNETES)

Nos conectamos con el usuario administrador (kubeadmin)

oc new-project desa1

para verlo oc get Project desa1

## Crear un proyecto en modo declarativo desde un fichero YAML (KUBERNETES)

El archivo. yaml debe de contener la información

```
[openshift@curso proyectos]$ oc apply -f proyecto.yaml
project.project.openshift.io/desa2 created
[openshift@curso proyectos]$ oc get projects -l tipo=desa
NAME    DISPLAY NAME                                        STATUS
desa2   Ejemplo de creacion de una proyecto Openshift       Active
```

Borrar un proyecto

```
[openshift@curso ~]$ oc delete project desa
project.project.openshift.io "desa3" delete
[openshift@curso ~]$ oc get projects
NAME
default
desa1
desa2
```

Para no tener problema de permisos es necesario ejecutar el siguiente comando en cada uno de los proyectos que creemos durante el curso

Hay que modificar "default" por el nombre del proyecto....

oc adm policy add-scc-to-user anyuid -z default

## Crear un POD (KUBERNETES)

```
[openshift@curso ~]$ oc project desa1
Now using project "desa1" on server "https://api.crc.testing:6443".
[openshift@curso ~]$ oc run --generator=run-pod/v1 nginx --image=nginx
pod/nginx created
```

```
[openshift@curso ~]$ oc describe pod nginx
```

Podemos ver

```
[openshift@curso ~]$ oc rsh nginx
# ls -l
total 8
drwxr-xr-x.   2 root root 4096 Feb 24 00:00 bin
drwxr-xr-x.   2 root root    6 Feb  1 17:09 boot
```

## Crear un POD con Manifest (DOCKER) (KUBERNETES)

Azul → nombre de usuario de dockerhub

Rojo → Nombre que le damos a la imagen

```
[openshift@curso crearPODconManifest]$ sudo docker build -t apasoft/nginx-openshift .
[sudo] password for openshift:
Sending build context to Docker daemon 3.584 kB
Step 1/7 : FROM ubuntu:12.04
 ---> 5b117edd0b76
Step 2/7 : MAINTAINER Apasoft Formacion "apasoft.formacion@gmail.com"
 ---> Using cache
 ---> 165dac39e424
Step 3/7 : RUN apt-get update
 ---> Using cache
 ---> 72060b7ded82
Step 4/7 : RUN apt-get install -y nginx
 ---> Using cache
 ---> b89091342d98
Step 5/7 : RUN echo 'Ejemplo de POD para el curso de OPENSHIFT de Apasoft Formacion' > /usr/sha
 ---> Using cache
 ---> 369a0839f877
Step 6/7 : ENTRYPOINT /usr/sbin/nginx -g daemon off;
```

Nos logeamos en Docker

```
openshift@curso crearPODconManifest]$ sudo docker login
```

Subimos la imagen

```
[openshift@curso crearPODconManifest]$ sudo docker push apasoft/nginx-openshift
```

Creamos el yaml y ejecutamos

```
[openshift@curso crearPODconManifest]$ oc apply -f nginx.yaml
pod/nginx-openshift created
[openshift@curso crearPODconManifest]$ oc get pod
NAME              READY   STATUS              RESTARTS   AGE
nginx-openshift   0/1     ContainerCreating   0          7s
```

Los ficheros los da el curso

## Crear deployment (KUBERNETES)

Los ficheros los da el curso

RECORDAR HACERLO DESDE EL PROYECTO CREADO

```
master-->oc apply -f deploy.yaml
deployment.apps/example created
master-->oc get all
```

IMPORTANTE QUE NO COINCIDA CON EL PUERTO 80

## Crear un servicio (KUBERNETES)

Los ficheros los da el curso

```
master-->kubectl apply -f servicio.yaml
```

```
master-->oc get svc
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
servicio   NodePort   172.30.79.235   <none>        8080:30005/TCP   10s
```

Para probar esto vemos la IP y el puerto

```
master-->crc ip
192.168.130.11
master-->oc get svc
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
servicio   NodePort   172.30.79.235   <none>        8080:30005/TCP   106s
```

Y en internet

# Despliegue desde una imagen

Es necesario indicar la siguiente opción al ejecutar el comando. Ya que si no nos generará un deploymentConfig (Pasa desde las versiones 4.5 de Openshift)

**--as-deployment-config**

LA IMAGEN DEBE ESTAR SUBIDA A DOCKERHUB Y DEBEMOS ESTAR LOGEADOS

```
master-->oc new-app apasoft/blog
```

Ver configuración y características

is (ImageStream)

```
master-->oc get is
NAME   IMAGE REPOSITORY                                                    TAGS     UPDATED
blog   default-route-openshift-image-registry.apps-crc.testing/desa1/blog   latest   4 minut
ago
master-->oc describe is blog
Name:                   blog
Namespace:              desa1
Created:                5 minutes ago
Labels:                 app=blog
                        app.kubernetes.io/component=blog
                        app.kubernetes.io/instance=blog
Annotations:            openshift.io/generated-by=OpenShiftNewApp
                        openshift.io/image.dockerRepositoryCheck=2020-03-20T19:10:26Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/desa1/blog
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

latest
  tagged from apasoft/blog

  * apasoft/blog@sha256:9c8d3cedcd722ae6a9508ca532102501dfb796600c53a61d5178cdfaf22e02ed
      5 minutes ago
```

```
master-->oc get deploymentconfig
NAME   REVISION   DESIRED   CURRENT   TRIGGERED BY
blog   1          1         1         config,image(blog:latest)
master-->oc get dc
NAME   REVISION   DESIRED   CURRENT   TRIGGERED BY
blog   1          1         1         config,image(blog:latest)
```

```
master-->oc describe dc blog
Name:                   blog
Namespace:              desa1
Created:                8 minutes ago
Labels:                 app=blog
                        app.kubernetes.io/com
                        app.kubernetes.io/ins
```

El POD marcado es el importante los otros son pasos generados para llegar al marcado.

```
master-->oc get rs
No resources found in desa1 namespace.
master-->oc get rc
NAME      DESIRED   CURRENT   READY   AGE
blog-1    1         1         1       9m50s
master-->oc get rc -o wide
NAME      DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES
                                                SELECTOR
blog-1    1         1         1       10m   blog         apasoft/blo
08ca532102501dfb796600c53a61d5178cdfaf22e02ed   deployment=blog-1,
master-->oc get pod
NAME            READY   STATUS      RESTARTS   AGE
blog-1-deploy   0/1     Completed   0          10m
blog-1-g89rd    1/1     Running     0          10m
master-->oc get svc
NAME   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
blog   ClusterIP   172.30.145.190   <none>        8080/TCP   11m
```

Continuamos

```
[openshift@curso deploymentDeclarativo]$ oc expose svc blog
route.route.openshift.io/blog exposed
[openshift@curso deploymentDeclarativo]$ oc get route blog
NAME   HOST/PORT                      PATH   SERVICES   PORT       TERMINATION   WILDCARD
blog   blog-desa1.apps-crc.testing           blog       8080-tcp                 None
```

Prueba


Podemos escalarlo

```
[openshift@curso deploymentDeclarativo]$ oc scale --replicas=3 dc blog
deploymentconfig.apps.openshift.io/blog scaled
[openshift@curso deploymentDeclarativo]$ oc get pod
NAME            READY   STATUS              RESTARTS   AGE
blog-1-deploy   0/1     Completed           0          134m
blog-1-g89rd    1/1     Running             1          134m
blog-1-hbznv    0/1     ContainerCreating   0          7s
blog-1-mt5ft    0/1     ContainerCreating   0          7s
```

Ver lo que se ha creado

```
[openshift@curso ~]$ oc get all -o name -l app=blog
replicationcontroller/blog-1
service/blog
deploymentconfig.apps.openshift.io/blog
imagestream.image.openshift.io/blog
route.route.openshift.io/blog
```

Borrar

```
[openshift@curso ~]$ oc delete all -o name -l app=blog
```

# Práctica WordPress y MYSQL

El MYSWL debe la 5.7 o dará error

El PDF con los comandos está en la carpeta (Para crear el MYSQL y el WordPress poner todo el comando junto)

oc new-app mysql:5.7 --name=mysql1 -e MYSQL_ROOT_PASSWORD=secret -e MYSQL_USER=usu1 -e MYSQL_PASSWORD=secret MYSQL_DATABASE=wordpress

TODOS LOS COMANDOS EN EL PDF

Este comando está mal en el PDF

oc expose svc/wordpress1

Con el ultimo comando del PDF vemos la ruta si la copiamos y pegamos en el navegador veremos si está todo correcto


# Despliegue de una aplicación desde código fuente en un repositorio GIT

Primero creamos la aplicación especificando la ruta donde está en git. Lo marcado es opcional

```
[openshift@curso ~]$ oc new-app python:3.5~https://github.com/apasofttraining/blog  --name blog1
```

Podemos ver propiedades (Suele tardar)

```
[openshift@curso ~]$ oc get bc blog1
NAME    TYPE     FROM   LATEST
blog1   Source   Git    1
[openshift@curso ~]$ oc get build
NAME      TYPE     FROM          STATUS    STARTED              DURATION
blog1-1   Source   Git@f386268   Running   About a minute ago
[openshift@curso ~]$ oc get pod
NAME            READY   STATUS      RESTARTS   AGE
blog-1-954pv    1/1     Running     0          14h
blog-1-deploy   0/1     Completed   0          14h
blog-1-xqhlf    1/1     Running     0          13h
blog1-1-build   1/1     Running     0          94s
```

```
[openshift@curso ~]$ oc get pods
NAME             READY   STATUS      RESTARTS   AGE
blog-1-954pv     1/1     Running     0          14h
blog-1-deploy    0/1     Completed   0          14h
blog-1-xqhlf     1/1     Running     0          14h
blog1-1-build    0/1     Completed   0          31m
blog1-1-deploy   0/1     Completed   0          21m
blog1-1-sqck4    1/1     Running     0          21m
```

Crear el route

```
[openshift@curso ~]$ oc expose svc blog1
route.route.openshift.io/blog1 exposed
```

Prueba

```
[openshift@curso ~]$ oc describe route blog1
Name:                   blog1
Namespace:              desa1
Created:                18 seconds ago
Labels:                 app=blog1
                        app.kubernetes.io/component=blog1
                        app.kubernetes.io/instance=blog1
Annotations:            openshift.io/host.generated=true
Requested Host:         blog1-desa1.apps-crc.testing
                          exposed on router default (host apps-crc.testing)
```


## Despliegue de una aplicación desde Docker file

Creamos la ampliación con la ruta, indicar en la estrategia que es Docker

```
[openshift@curso ~]$ oc new-app --name blog3 --strategy=docker https://github.com/apasofttraining/blo
g
```

Propiedades y características

```
[openshift@curso ~]$ oc get bc
NAME    TYPE     FROM   LATEST
blog1   Source   Git    1
blog2   Source   Git    1
blog3   Docker   Git    1
[openshift@curso ~]$ oc get builds
NAME      TYPE     FROM          STATUS     STARTED             DURATION
blog1-1   Source   Git@f386268   Complete   4 hours ago         3m59s
blog2-1   Source   Git@f386268   Complete   About an hour ago   7m52s
blog3-1   Docker   Git@f386268   Running    41 seconds ago
```

```
[openshift@curso ~]$ oc get pods
NAME              READY   STATUS      RESTARTS   AGE
blog-1-954pv      1/1     Running     0          23h
blog-1-deploy     0/1     Completed   0          23h
blog-1-xqhlf      1/1     Running     0          23h
blog1-1-4pf6b     1/1     Running     0          3h28m
blog1-1-build     0/1     Completed   0          3h32m
blog1-1-deploy    0/1     Completed   0          3h28m
blog2-1-build     0/1     Completed   0          81m
blog2-1-deploy    0/1     Completed   0          73m
blog2-1-hl7ff     1/1     Running     0          73m
blog3-1-build     1/1     Running     0          88s
[openshift@curso ~]$ oc log blog3-1-build
```

Podemos ver los logs

```
[openshift@curso ~]$ oc logs blog3-1-build
```

Crear el route

```
[openshift@curso ~]$ oc expose svc blog3
route.route.openshift.io/blog3 exposed
```

Para probarlo y después escribir lo marcado en el navegador

```
[openshift@curso ~]$ oc get route blog3
NAME    HOST/PORT                      PATH   SERVICES   PORT       TERMINATION   WILDCARD
blog3   blog3-desa1.apps-crc.testing          blog3      8080-tcp                 None
```

# Variables

Lo primero será crear un nuevo proyecto

```
[openshift@curso Variables]$ oc new-project variables
```

```
[openshift@curso Variables]$ oc apply -f dc_variables.yaml
```

Configuración

```
[openshift@curso Variables]$ oc get dc
NAME                REVISION   DESIRED   CURRENT   TRIGGERED BY
ejemplo-variables   1          2         0         config
[openshift@curso Variables]$ oc get rc
NAME                    DESIRED   CURRENT   READY   AGE
ejemplo-variables-1     2         2         2       8s
[openshift@curso Variables]$ oc get pod
NAME                          READY   STATUS      RESTARTS   AGE
ejemplo-variables-1-7s6f5     1/1     Running     0          10s
ejemplo-variables-1-deploy    0/1     Completed   0          14s
ejemplo-variables-1-ll4hc     1/1     Running     0          10s
```

Conectarnos ssh al POD

```
[openshift@curso Variables]$ oc rsh ejemplo-variables-1-7s6f5
```

Prueba

```
$ env
KUBERNETES_SERVICE_PORT=443
KUBERNETES_PORT=tcp://172.30.0.1:443
NODE_VERSION=4.4.2
HOSTNAME=ejemplo-variables-1-7s6f5
HOME=/
TERM=xterm-256color
KUBERNETES_PORT_443_TCP_ADDR=172.30.0.1
PROPIETARIO=Apasoft Training
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
NOMBRE=CURSO DE OPENSHIFT
KUBERNETES_PORT_443_TCP_PORT=443
NPM_CONFIG_LOGLEVEL=info
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP=tcp://172.30.0.1:443
KUBERNETES_SERVICE_HOST=172.30.0.1
PWD=/
$ echo $NOMBRE
CURSO DE OPENSHIFT
$
```

# Modificar variables

Entramos en el editor de variables

```
[openshift@curso Variables]$ oc edit dc ejemplo-variables
```

Si cambiamos algo, lo vuelve a lanzar y le cambia el nombre

```
[openshift@curso Variables]$ oc get pods
NAME                          READY   STATUS      RESTARTS   AGE
ejemplo-variables-1-deploy    0/1     Completed   0          3m54s
ejemplo-variables-2-9xm6d     1/1     Running     0          100s
ejemplo-variables-2-deploy    0/1     Completed   0          104s
ejemplo-variables-2-ljml9     1/1     Running     0          95s
[openshift@curso Variables]$ oc rsh ejemplo-variables-2-ljml9
```

# Comandos interesantes de variables

Listar variables

```
[openshift@curso Variables]$ oc set env dc/ejemplo-variables --list
```

Añadir variables

```
[openshift@curso Variables]$ oc set env dc/ejemplo-variables RESPONSABLE=pedro
deploymentconfig.apps.openshift.io/ejemplo-variables updated
```

Actualizar variables

```
[openshift@curso Variables]$ oc set env dc/ejemplo-variables --overwrite RESPONSABLE=Rosa
deploymentconfig.apps.openshift.io/ejemplo-variables updated
```

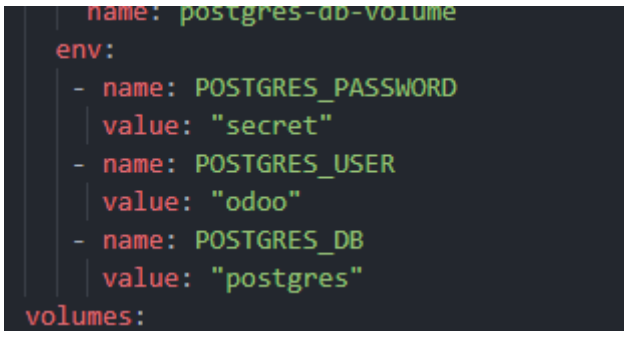
Para borrar una variable

```
[openshift@curso Variables]$ oc set env dc/ejemplo-variables RESPONSABLE-
```

# Ejemplo Odoo y Postgres

Los ficheros nos lo proporcionan el curso, La práctica es completamente manual

Lo importante son la configuración de las variables




Creamos la aplicación

```
openshift=>oc apply -f dep_postgres.yaml
deploymentconfig.apps.openshift.io/postgres-db created
openshift=>oc get dc
NAME                REVISION   DESIRED   CURRENT   TRIGGERED BY
ejemplo-variables   7          2         2         config
postgres-db         1          1         0         config
openshift=>oc get rc
NAME                   DESIRED   CURRENT   READY   AGE
ejemplo-variables-1    0         0         0       15h
ejemplo-variables-2    0         0         0       15h
ejemplo-variables-3    0         0         0       15h
ejemplo-variables-4    0         0         0       15h
ejemplo-variables-5    0         0         0       15h
ejemplo-variables-6    0         0         0       15h
ejemplo-variables-7    2         2         2       15h
postgres-db-1          1         1         0       16s
```

Probamos que es correcto y entramos dentro

```
openshift=>oc get pod | grep postg
postgres-db-1-6f75d          1/1     Running     0          36s
postgres-db-1-deploy         0/1     Completed   0          45s
openshift=>oc rsh postgres-db-1-6f75d
```

Entramos dentro de la bbdd ponemos el usuario y la bbdd que nos queremos conectar

```
# psql -U odoo postgres
psql (11.7 (Debian 11.7-2.pgdg90+1))
Type "help" for help.

postgres=# \ll
Invalid command \ll. Try \? for help.
postgres=# \l
                                  List of databases
   Name    | Owner | Encoding |  Collate   |   Ctype    | Access privileges
-----------+-------+----------+------------+------------+-------------------
 postgres  | odoo  | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | odoo  | UTF8     | en_US.utf8 | en_US.utf8 | =c/odoo          +
           |       |          |            |            | odoo=CTc/odoo
 template1 | odoo  | UTF8     | en_US.utf8 | en_US.utf8 | =c/odoo          +
           |       |          |            |            | odoo=CTc/odoo
(3 rows)
```

Nos salimos con exit y exit otra vez para estar en el Openshift. Listar variables

```
openshift=>oc set env dc/postgres-db --list
# deploymentconfigs/postgres-db, container postgres-db
POSTGRES_PASSWORD=secret
POSTGRES_USER=odoo
POSTGRES_DB=postgres
```

Ahora creamos el servicio

```
openshift=>oc apply -f servicio_postgres.yaml
service/postgres-svc created
```

```
openshift=>oc get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
postgres-svc   ClusterIP   172.30.17.131   <none>        5432/TCP   5s
openshift=>oc describe svc postgres-svc
Name:              postgres-svc
Namespace:         variables
Labels:            app=postgres-db
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"
":"postgres-db"},"name":"postgres-svc","namespace":"variabl...
Selector:          deploymentconfig=postgres-db
Type:              ClusterIP
IP:                172.30.17.131
Port:              5432-tcp  5432/TCP
TargetPort:        5432/TCP
Endpoints:         10.128.0.213:5432
Session Affinity:  None
Events:            <none>
```

Configuración de las variables del archivo de odoo (El archivo está en los recursos del curso)

```
containers:
- env:
  - name: HOST
    value: postgres-svc
  - name: PASSWORD
    value: secret
  - name: USER
    value: odoo
```

Levantamos el odoo

```
openshift=>oc apply -f odoo.yaml
deploymentconfig.apps.openshift.io/odoo created
```

Detalles

```
openshift=>oc get dc
NAME                REVISION   DESIRED   CURRENT   TRIGGERED BY
ejemplo-variables   7          2         2         config
odoo                1          1         0         config
postgres-db         1          1         1         config
```

```
openshift=>oc get pod | grep odoo
odoo-1-deploy                0/1     Completed   0          43s
odoo-1-wbcmn                 1/1     Running     0          35s
openshift=>
```

Ya tendríamos el postgre con sus servicios y el odoo funcionando

Creamos el servicio de odoo

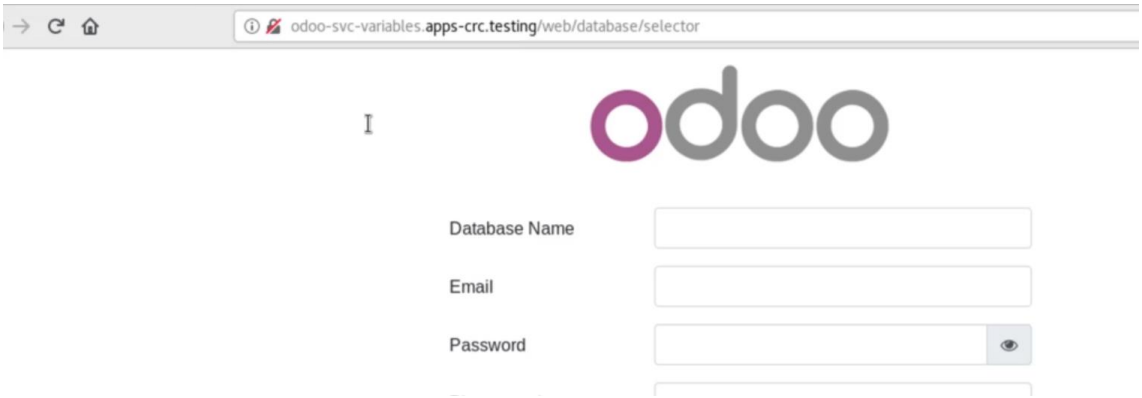
```
openshift=>oc apply -f servicio-odoo.yaml
service/odoo-svc created
```

Lo exponemos

```
openshift=>oc expose svc odoo-svc
route.route.openshift.io/odoo-svc exposed
```

Para probarlo

```
openshift=>oc get route
NAME       HOST/PORT                                PATH   SERVICES   PORT       TERMINATION   WILDCA
odoo-svc   odoo-svc-variables.apps-crc.testing             odoo-svc   8069-tcp                 None
```




## Config Maps

Crear un Config map

```
openshift=>oc create configmap cf1 --from-literal=NOMBRE=apasof --from-literal=PROPIETARIO='Apasoft
Training'
configmap/cf1 created
```

```
openshift=>oc describe cm cf1
Name:         cf1
Namespace:    variables
Labels:       <none>
Annotations:  <none>

Data
====
NOMBRE:
----
apasof
PROPIETARIO:
----
Apasoft Training
```

Ahora si modificamos un fichero y sustituimos (Seria como importar una config map)

```
env:
- name: NOMBRE
  value: "CURSO DE OPENSHIFT"
- name: PROPIETARIO
  value: "Apasoft Training"
```

```
envFrom:
- configMapRef:
    name: cf1
```

Importante que esté bien anidado

Lo suyo sería hacer esto antes de lanzar los archivos YAML

# ConfigMaps con ficheros. Ejemplo con Odoo y Postgres

Los recursos los proporciona el curso. Borrar antes de hacer esto (PODS, Deployment Config, porque no pueden llamarse igual)

```
openshift=>cat postgres.properties
POSTGRES_PASSWORD=secret
POSTGRES_USER=odoo
POSTGRES_DB=postgres
```

Creamos un ConfigMaps el cual llamamos Postgres-cm y le decimos que el archivo de donde tiene que coger los valores se llama postgres.properties

```
openshift=>oc create configmap postgres-cm --from-env-file postgres.properties
configmap/postgres-cm created
```

```
openshift=>oc describe cm postgres-cm
Name:         postgres-cm
Namespace:    variables
Labels:       <none>
Annotations:  <none>

Data
====
POSTGRES_DB:
----
postgres
POSTGRES_PASSWORD:
----
secret
POSTGRES_USER:
----
odoo
Events:  <none>
openshift=>oc get cm
NAME          DATA   AGE
cf1           2      99m
postgres-cm   3      20s
```

Editar el archivo YAML (deb_postgres.yaml)

```
    envFrom:
      - configMapRef:
          name: postgres-cm
  volumes:
```

Lo aplicamos

```
openshift=>oc apply -f dep_postgres.yaml
deploymentconfig.apps.openshift.io/postgres-db created
```

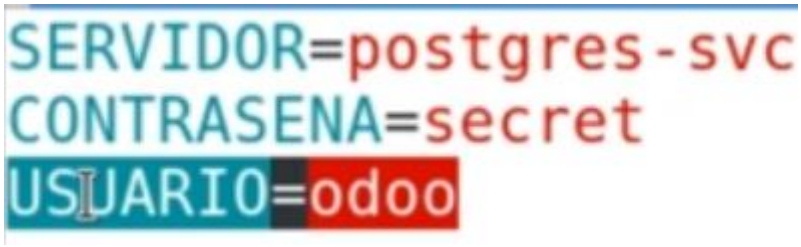
Prueba

```
openshift=>oc get dc
NAME          REVISION   DESIRED   CURRENT   TRIGGERED BY
postgres-db   1          1         1         config
openshift=>oc get rc
NAME            DESIRED   CURRENT   READY   AGE
postgres-db-1   1         1         0       8s
openshift=>oc get pod
NAME                    READY   STATUS    RESTARTS   AGE
postgres-db-1-deploy    1/1     Running   0          13s
postgres-db-1-w867x     1/1     Running   0          10s
```

Creamos el servicio

```
openshift=>oc apply -f servicio_postgres.yaml
service/postgres-svc created
openshift=>oc get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
postgres-svc   ClusterIP   172.30.249.92   <none>        5432/TCP   5s
```

Fichero de configuración de odoo




Creamos el ConfigMaps para odoo

```
openshift=>oc create configmap odoo-cm --from-env-file odoo.properties
configmap/odoo-cm created
```

```
openshift=>oc describe cm odoo-cm
Name:         odoo-cm
Namespace:    variables
Labels:       <none>
Annotations:  <none>

Data
====
CONTRASENA:
----
secret
SERVIDOR:
----
postgres-svc
USUARIO:
----
odoo
Events:  <none>
```

Esta vez el YAML será distinto ya que las propiedades no se llaman igual, aquí le indicamos que propiedad del ConfigMaps debe escoger para cada propiedad que necesita su configuración

```
    containers:
    - env:
      - name: HOST
        valueFrom:
         configMapKeyRef:
           name: odoo-cm
           key: SERVIDOR
      - name: PASSWORD
        valueFrom:
         configMapKeyRef:
           name: odoo-cm
           key: CONTRASENA
      - name: USER
        valueFrom:
         configMapKeyRef:
           name: odoo-cm
           key: USUARIO
```

Hacemos el Deploy Config

```
openshift=>oc apply -f odoo.yaml
deploymentconfig.apps.openshift.io/odoo created
```

Prueba

```
openshift=>oc get dc
NAME          REVISION   DESIRED   CURRENT   TRIGGERED BY
odoo          1          1         0         config
postgres-db   1          1         1         config
openshift=>oc get rc
NAME            DESIRED   CURRENT   READY   AGE
odoo-1          1         1         1       13s
postgres-db-1   1         1         1       46m
openshift=>oc get pod
NAME                    READY   STATUS      RESTARTS   AGE
odoo-1-cmp9t            1/1     Running     0          15s
odoo-1-deploy           0/1     Completed   0          18s
postgres-db-1-deploy    0/1     Completed   0          46m
postgres-db-1-w867x     1/1     Running     0          46m
```

Creamos el servicio

```
openshift=>oc apply -f servicio-odoo.yaml
service/odoo-svc created
```

Prueba

```
openshift=>oc get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
odoo-svc       ClusterIP   172.30.42.119   <none>        8069/TCP,8071/TCP,8072/TCP   4s
postgres-svc   ClusterIP   172.30.249.92   <none>        5432/TCP                     45m
```

Y ahora debemos crear la ruta, así podríamos verlo desde internet, ya que está le asigna IP y puerto

```
openshift=>oc expose svc odoo-svc
route.route.openshift.io/odoo-svc exposed
```

Prueba

```
openshift=>oc get route
NAME       HOST/PORT                              PATH   SERVICES   PORT       TERMINATION   WILDCARD
odoo-svc   odoo-svc-variables.apps-crc.testing           odoo-svc   8069-tcp                 None
```


## Crear un secret

Creamos un secret (from-literal es para añadir propiedades, secret-cm es el nombre que le damos)

```
openshift=>oc create secret generic secret-cm --from-literal=usuario=usu1 --from-literal=password=se
cret
secret/secret-cm created
```

Si nos fijamos no podemos ver los valores que hemos asignado

```
openshift=>oc describe secret secret-cm
Name:         secret-cm
Namespace:    variables
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  6 bytes
usuario:   4 bytes
```

Ahora editamos el fichero (Recurso del curso)

```
envFrom:
- secretRef:
    name: secret-cm
```

Hacemos el Deployment Config

```
openshift=>oc apply -f dc_secret.yaml
deploymentconfig.apps.openshift.io/ejemplo-secrets created
```

```
openshift=>oc get pods
NAME                       READY   STATUS      RESTARTS   AGE
ejemplo-secrets-1-deploy   0/1     Completed   0          14s
ejemplo-secrets-1-zr555    1/1     Running     0          11s
```

# Secrets declarativos

Primero borramos el Deployment Config (dc) de odoo, también se borra el POD, etc. Lo que no borra es el servicio el cuál no vamos a borrarlo ni tampoco el route

```
openshift=>oc delete dc odoo
deploymentconfig.apps.openshift.io "odoo" deleted
```

Ahora crearíamos el fichero secreto1.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: secreto1
type: Opaque
data:
  PASSWORD: c2VjcmV0Cg==
```

Para saber que valor poner en la password debemos convertir la contraseña a base 64, en esté caso como la contraseña del Postgres era secret convertimos esa

```
openshift=>echo secret | base64
c2VjcmV0Cg==
```

Creamos el secret y podemos ver que está creado

```
openshift=>oc apply -f secreto1.yaml
secret/secreto1 created
openshift=>oc get secret
NAME                       TYPE                                  DATA   AGE
builder-dockercfg-cdcrm    kubernetes.io/dockercfg               1      25h
builder-token-hzbmg        kubernetes.io/service-account-token   4      25h
builder-token-jx6wg        kubernetes.io/service-account-token   4      25h
default-dockercfg-9nmdj    kubernetes.io/dockercfg               1      25h
default-token-npxzs        kubernetes.io/service-account-token   4      25h
default-token-p6j97        kubernetes.io/service-account-token   4      25h
deployer-dockercfg-kjjml   kubernetes.io/dockercfg               1      25h
deployer-token-m8c4f       kubernetes.io/service-account-token   4      25h
deployer-token-mbd5d       kubernetes.io/service-account-token   4      25h
secret-cm                  Opaque                                2      16m
secreto1                   Opaque                                1      4s
```

Ahora editamos el archivo de configuración odoo.yaml en el cual le decimos que la contraseña la coja del secret que hemos creado, lo demás de un ConfigMaps

```
- env:
  - name: HOST
    valueFrom:
      configMapKeyRef:
        name: odoo-cm
        key: SERVIDOR
  - name: PASSWORD
    valueFrom:
      secretKeyRef:
        name: secreto1
        key: PASSWORD
  - name: USER
    valueFrom:
      configMapKeyRef:
        name: odoo-cm
        key: USUARIO
```

Creamos el dc

```
openshift=>oc apply -f odoo.yaml
deploymentconfig.apps.openshift.io/odoo created
```

Prueba

```
openshift=>oc get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                       AGE
odoo-svc      ClusterIP   172.30.42.119   <none>        8069/TCP,8071/TCP,8072/TCP    38m
postgres-svc  ClusterIP   172.30.249.92   <none>        5432/TCP                      83m
openshift=>oc get route
NAME       HOST/PORT                                PATH   SERVICES   PORT       TERMINATION   WILDCARD
odoo-svc   odoo-svc-variables.apps-crc.testing             odoo-svc   8069-tcp                 None
```


## Crear una ImageStream desde new-app

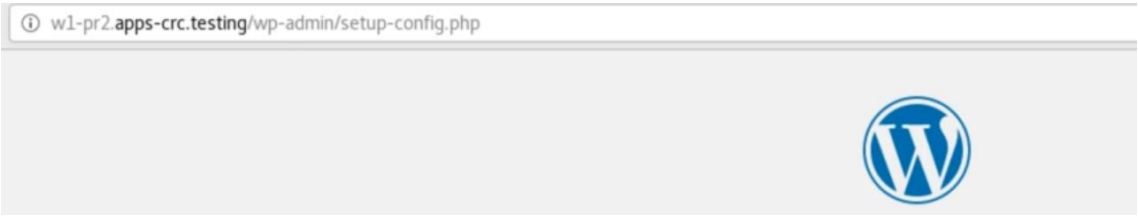
Con el comando new-app, nos crearía automáticamente el ImageStream

```
[openshift@curso ~]$ oc new-app wordpress --name=w1
 Found container image 0d205d4 (7 days old) from Docker Hub for "wordpress"

 * An image stream tag will be created as "w1:latest" that will track this image
 * This image will be deployed in deployment config "w1"
 * Port 80/tcp will be load balanced by service "w1"
   * Other containers can access this service through the hostname "w1"
 * This image declares volumes and will default to use non-persistent, host-local
```

Creamos la route

```
[openshift@curso ~]$ oc expose svc/w1
route.route.openshift.io/w1 exposed
[openshift@curso ~]$ oc get route
NAME   HOST/PORT                  PATH   SERVICES   PORT     TERMINATION   WILDCARD
w1     w1-pr2.apps-crc.testing           w1         80-tcp                 None
```

Si lo probamos en internet nos lo habría creado




Para ver los ImageStream que tenemos

```
[openshift@curso ~]$ oc get is
NAME   IMAGE REPOSITORY                                                TAGS     UPDATED
w1     default-route-openshift-image-registry.apps-crc.testing/pr2/w1   latest   2 minutes ago
```

Y para ver uno con detalle

```
[openshift@curso ~]$ oc describe is w1
Name:                   w1
Namespace:              pr2
Created:                3 minutes ago
Labels:                 app=w1
                        app.kubernetes.io/component=w1
                        app.kubernetes.io/instance=w1
Annotations:            openshift.io/generated-by=OpenShiftNewApp
                        openshift.io/image.dockerRepositoryCheck=2020-04-09T14:19:28Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/pr2/w1
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

latest
  tagged from wordpress
```

A la imagen que apunta sale al final

```
latest
  tagged from wordpress

  * wordpress@sha256:6c6f664c9e1f6d88aa279ef2b7e1d889edf12483dd38face9affaede20b2cf67
```

## Crear una ImageStream que apunte a una imagen externa con import-image

Ponemos el comando y le indicamos el nombre y el from es decir de donde va a coger la imagen, importante poner el confirm al final si no da error

```
[openshift@curso ~]$ oc import-image cassandra:latest --from="docker.io/cassandra:latest" --confirm
```

```
[openshift@curso ~]$ oc get is
NAME        IMAGE REPOSITORY                                                  TAGS     UPDATED
cassandra   default-route-openshift-image-registry.apps-crc.testing/pr2/cassandra   latest   2 minutes ago
```

El ImageStream es como crear una referencia

Si hacemos un ejemplo, veremos que tarda porque ahora es cuando descarga la imagen.

Como podemos observar tenemos dos IS, una la nueva y otra la de cassandra que creamos antes

```
[openshift@curso ~]$ oc new-app cassandra:latest --name=c1
--> Found image 9243b2c (13 days old) in image stream "pr2/cassandra" under tag "latest" for "cassandra:late

    * This image will be deployed in deployment config "c1"
    * Ports 7000/tcp, 7001/tcp, 7199/tcp, 9042/tcp, 9160/tcp will be load balanced by service "c1"
      * Other containers can access this service through the hostname "c1"
    * This image declares volumes and will default to use non-persistent, host-local storage.
      You can add persistent volumes later by running 'oc set volume dc/c1 --add ...'
    * WARNING: Image "pr2/cassandra:latest" runs as the 'root' user which may not be permitted by your clust
ministrator

--> Creating resources ...
    imagestreamtag.image.openshift.io "c1:latest" created
    deploymentconfig.apps.openshift.io "c1" created
    service "c1" created
--> Success
```

```
[openshift@curso ~]$ oc get is
NAME        IMAGE REPOSITORY                                                     TAGS     UPDATED
c1          default-route-openshift-image-registry.apps-crc.testing/pr2/c1       latest
cassandra   default-route-openshift-image-registry.apps-crc.testing/pr2/cassandra latest   6 minutes ago
w1          default-route-openshift-image-registry.apps-crc.testing/pr2/w1       latest   2 hours ago
```

Nos puede salir el siguiente error al poner esto

```
[openshift@curso ~]$ oc describe is c1
```

```
  ! error: Import failed (InternalError): Internal error occurred: image-registry.openshift-image-registry.svc
00/pr2/cassandra:latest: Get https://image-registry.openshift-image-registry.svc:5000/v2/: x509: certificate s
ed by unknown authority
```

PARA SOLUCIONARLO HAY UN FICHERO EN LA CARPETA DONDE SE EXPLICA

Si somos administradores podemos ver donde está mapeada la IS

```
[openshift@curso ~]$ oc get images | grep cassandra
sha256:f5372637c06b5ef55facf34db63eafaf86db3613c13d67804711162832d4fd5d  docker.io/cassandra@sha256:f5372637c06b
5ef55facf34db63eafaf86db3613c13d67804711162832d4fd5d
[openshift@curso ~]$ oc adm top images | grep sha256:f5372637c06b5ef55facf34db63eafaf86db3613c13d67804711162832d4
fd5d
sha256:f5372637c06b5ef55facf34db63eafaf86db3613c13d67804711162832d4fd5d pr2/cassandra (latest)
                                        <none>                          Deployment: pr2/c1-1 no
  123MiB
```

Prueba de funcionamiento

```
[openshift@curso ~]$ oc get pods
NAME          READY   STATUS      RESTARTS   AGE
c1-1-deploy   0/1     Completed   0          96m
c1-1-w42mj    1/1     Running     0          95m
w1-1-457sn    1/1     Running     0          4h
w1-1-deploy   0/1     Completed   0          4h
[openshift@curso ~]$ oc exec -it c1-1-w42mj bash
root@c1-1-w42mj:/# cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

## Crear una ImageStream interna

Para ello importamos la imagen junto con el tag (versión)

```
[openshift@curso ~]$ oc import-image mariadb:10.2 --confirm
```

Aquí podemos ver que la ha importado con el tag

```
[openshift@curso ~]$ oc get is
NAME      IMAGE REPOSITORY                                                    TAGS   UPDAT
ED
mariadb   default-route-openshift-image-registry.apps-crc.testing/images/mariadb   10.2   45 se
conds ago
```

Con este comando vemos también el tag y hacia donde está apuntando

```
[openshift@curso ~]$ oc describe is mariadb
Name:                   mariadb
Namespace:              images
Created:                About a minute ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2020-04-10T20:24:16Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/images/mar
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

10.2
  tagged from mariadb:10.2

  * mariadb@sha256:d7fe669341e505bc79f289e33ccac496a0c2ab24af5a74703193853225O7f755
      About a minute ago
```

Ahora hacemos el dc con el nombre de IS que hemos creado, seguido del nombre que le queramos poner al dc, seguido de una variable que es obligatorio indicar en mariadb

```
[openshift@curso ~]$ oc new-app mariadb:10.2 --name=maria1 -e MYSQL_ROOT_PASSWORD=secret
```

```
[openshift@curso ~]$ oc get is
NAME      IMAGE REPOSITORY                                                    TAGS   UPDAT
ED
maria1    default-route-openshift-image-registry.apps-crc.testing/images/maria1    10.2
mariadb   default-route-openshift-image-registry.apps-crc.testing/images/mariadb   10.2   19 mi
nutes ago
```

## Etiquetar ImageStream comando oc tag

Importamos la imagen

```
[openshift@curso ~]$ oc import-image odoo:13 --confirm
imagestream.image.openshift.io/odoo imported
```

Con este comando le agregamos la etiqueta latest

```
[openshift@curso ~]$ oc tag odoo:13 odoo:latest
Tag odoo:latest set to odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a
65a8.
```

Y vemos que ahora tiene 2 etiquetas

```
[openshift@curso ~]$ oc get is
NAME      IMAGE REPOSITORY                                                    TAGS
UPDATED
maria1    default-route-openshift-image-registry.apps-crc.testing/images/maria1    10.2
mariadb   default-route-openshift-image-registry.apps-crc.testing/images/mariadb   10.2
44 minutes ago
odoo      default-route-openshift-image-registry.apps-crc.testing/images/odoo      latest,13
12 seconds ago
```

Si usamos el comando describe vemos que tenemos 1 imagen con dos tags uno latest y otro 13

```
[openshift@curso ~]$ oc describe is odoo
Name:                   odoo
Namespace:              images
Created:                About a minute ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2020-04-10T21:07:40Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/images/odoo
Image Lookup:           local=false
Unique Images:          1
Tags:                   2

latest
  tagged from odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a65a8

  * odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a65a8
      35 seconds ago

13
  tagged from odoo:13

  * odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a65a8
```

Si le asignamos un tags a otra imagen (Versión 12)

```
[openshift@curso ~]$ oc tag docker.io/odoo:12 odoo:12
Tag odoo:12 set to docker.io/odoo:12.
```

Veremos que tenemos 3 tags

```
[openshift@curso ~]$ oc get is
NAME      IMAGE REPOSITORY                                                    TAGS          UP
DATED
maria1    default-route-openshift-image-registry.apps-crc.testing/images/maria1    10.2
mariadb   default-route-openshift-image-registry.apps-crc.testing/images/mariadb   10.2          Ab
out an hour ago
odoo      default-route-openshift-image-registry.apps-crc.testing/images/odoo      12,latest,13  16
 seconds ago
```

Pero si usamos el describe

```
[openshift@curso ~]$ oc describe is odoo
```

Veremos que tenemos 2 imágenes y 3 tags

```
Unique Images:          2
Tags:                   3

latest
  tagged from odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a65a8

  * odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a65a8
      2 minutes ago

12
  tagged from docker.io/odoo:12

  * docker.io/odoo@sha256:88cc016ea24e76ba9ebc8147e0f2247a85fb783b81436de334198f3865388440
      34 seconds ago

13
  tagged from odoo:13

  * odoo@sha256:3491c33af3989c1b7b624e5e17b3298f33d9f1c28035715681ef0204c30a65a8
```

## Crear una ImageStream desde un YAML

El fichero esta en los recursos, el contenido es:

```yaml
apiVersion: v1
kind: ImageStream
metadata:
  name: mi-web
spec:
  tags:
  - name: "1.0"
    from:
      kind: DockerImage
      name: apasoft/web
```

Lo lanzamos

```
[openshift@curso imageStreamYAML]$ oc apply -f is.yaml
imagestream.image.openshift.io/mi-web created
```

Y se ha creado

```
[openshift@curso imageStreamYAML]$ oc get is
NAME      IMAGE REPOSITORY                                                     TAGS
maria1    default-route-openshift-image-registry.apps-crc.testing/images/maria1    10.2
mariadb   default-route-openshift-image-registry.apps-crc.testing/images/mariadb   10.2
mi-web    default-route-openshift-image-registry.apps-crc.testing/images/mi-web    1.0
odoo      default-route-openshift-image-registry.apps-crc.testing/images/odoo      prod,12,latest,13
odoo1     default-route-openshift-image-registry.apps-crc.testing/images/odoo1     prod
```

Como vemos está mapeada correctamente

```
[openshift@curso imageStreamYAML]$ oc describe is mi-web
Name:                   mi-web
Namespace:              images
Created:                31 seconds ago
Labels:                 <none>
Annotations:            kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"imag
metadata":{"annotations":{},"name":"mi-web","namespace":"images"},"spec":{"tags":[{"from":{"
,"name":"1.0"}]}}

                        openshift.io/image.dockerRepositoryCheck=2020-04-10T22:08:05Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/images/mi-we
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

1.0
  tagged from apasoft/web

  * apasoft/web@sha256:1e7eab580f7a263375a02a676783dc0dc0c3490ab3e6abc0b56a3ff39cefadec
```

Si le agregamos un tag

```
[openshift@curso imageStreamYAML]$ oc tag mi-web:1.0 mi-web:ultima
Tag mi-web:ultima set to mi-web@sha256:1e7eab580f7a263375a02a676783dc0dc0c3490ab3e6abc0b56a
```

Veremos que tiene dos tag con el comando describe

Ahora creamos una aplicación

```
[openshift@curso imageStreamYAML]$ oc new-app mi-web:ultima --name=web1
```

```
[openshift@curso imageStreamYAML]$ oc get is
NAME      IMAGE REPOSITORY                                                     TAGS
maria1    default-route-openshift-image-registry.apps-crc.testing/images/maria1    10.2
mariadb   default-route-openshift-image-registry.apps-crc.testing/images/mariadb   10.2
mi-web    default-route-openshift-image-registry.apps-crc.testing/images/mi-web    ultima,1.
odoo      default-route-openshift-image-registry.apps-crc.testing/images/odoo      prod,12,l
odoo1     default-route-openshift-image-registry.apps-crc.testing/images/odoo1     prod
web1      default-route-openshift-image-registry.apps-crc.testing/images/web1      ultima
```

Creamos la ruta

```
[openshift@curso imageStreamYAML]$ oc expose svc web1
route.route.openshift.io/web1 exposed
```

Vemos la url

```
[openshift@curso imageStreamYAML]$ oc get route
NAME   HOST/PORT                     PATH   SERVICES   PORT     TERMINATION   WILDCARD
web1   web1-images.apps-crc.testing         web1       80-tcp                 None
```

## Actualizar de forma automática un ImageStream

Lo primero es crear una imagen (Por ejemlo a través de un dockerfile, está subido a los recursos del curso)

```
[openshift@curso imagenScheduler]$ cat Dockerfile
FROM alpine
CMD echo 'Hola, estas en la imagen con la versión inicial de la aplicacion' && exec sleep infinity
```

Lo subimos a nuestro DockerHub

```
[openshift@curso imagenScheduler]$ sudo docker build -t apasoft/imagen1 .
[sudo] password for openshift:
Sending build context to Docker daemon 2.048 kB
Step 1/2 : FROM alpine
 ---> a187dde48cd2
Step 2/2 : CMD echo 'Hola, estas en la imagen con la versión inicial de la aplicacion' && exec sle
 ---> Running in 8ce9968d6524
 ---> c2f89247e049
Removing intermediate container 8ce9968d6524
Successfully built c2f89247e049
[openshift@curso imagenScheduler]$ sudo docker push apasoft/imagen1
```

Ahora la importamos (IMPORTANTE EL Schedule, esto hará que dentro de 15 min pregunte si hay algún cambio)

```
[openshift@curso imagenScheduler]$ oc import-image apasoft/imagen1 --scheduled=true --confirm
```

Creamos una aplicación

```
[openshift@curso imagenScheduler]$ oc new-app imagen1 --name=im1
```

Si modificamos el Docker File

```
FROM alpine
CMD echo 'Hola, estas en la imagen con la versión segunda de la aplicacion' && exec sleep infinity
```

Ahora primero la actualizamos y luego lo subimos

```
[openshift@curso imagenScheduler]$ sudo docker build -t apasoft/imagen1 .
Sending build context to Docker daemon 2.048 kB
Step 1/2 : FROM alpine
 ---> a187dde48cd2
Step 2/2 : CMD echo 'Hola, estas en la imagen con la versión segunda de la aplicacion' && exec sle
 ---> Running in 68c0c2f7a961
 ---> 5acb473abd49
Removing intermediate container 68c0c2f7a961
Successfully built 5acb473abd49
[openshift@curso imagenScheduler]$ sudo docker push apasoft/imagen1
The push refers to a repository [docker.io/apasoft/imagen1]
beee9f30bc1f: Layer already exists
latest: digest: sha256:79d6792a97fc8938da7da1b4bee65e9b557fbc656d53067eedca74e611054922 size: 528
```

Y ahora habría que esperar a que se actualice dentro de 15 min

```
[openshift@curso imagenScheduler]$ oc describe is imagen1
Name:                   imagen1
Namespace:              default
Created:                2 minutes ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2020-04-11T12:22:37Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/default/imagen1
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

latest
  updates automatically from registry apasoft/imagen1

  * apasoft/imagen1@sha256:ccd07d776894fbd4bb6d2d9bcc0337355da3a67755291efc93063caf6f02e630
      2 minutes ago
```

Tras 15 min vemos que se ha actualizado

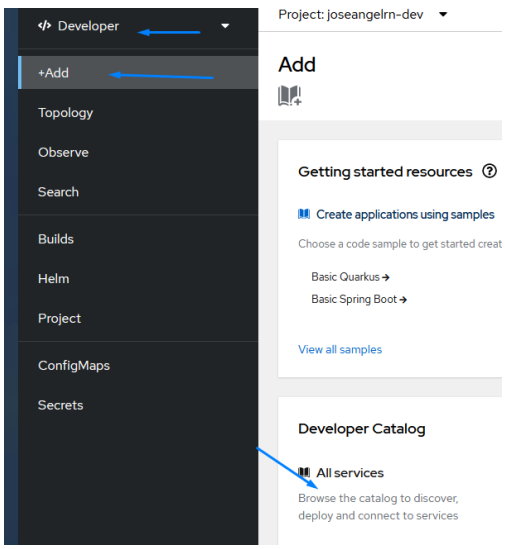
```
[openshift@curso imagenScheduler]$ oc describe is imagen1
Name:                   imagen1
Namespace:              default
Created:                About an hour ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2020-04-11T12:38:41Z
Image Repository:       default-route-openshift-image-registry.apps-crc.testing/default/im
Image Lookup:           local=false
Unique Images:          2
Tags:                   1

latest
  updates automatically from registry apasoft/imagen1

  * apasoft/imagen1@sha256:79d6792a97fc8938da7da1b4bee65e9b557fbc656d53067eedca74e61105492
      53 minutes ago
    apasoft/imagen1@sha256:ccd07d776894fbd4bb6d2d9bcc0337355da3a67755291efc93063caf6f02e63
      About an hour ago
```

# BuildConfigs

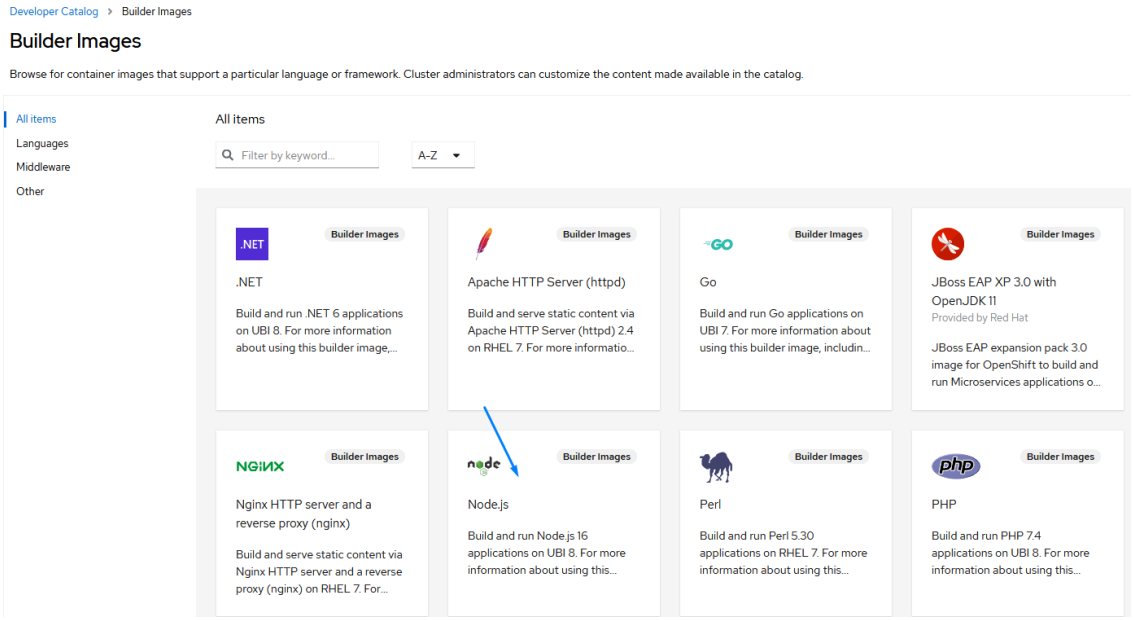
Podemos crear un nuevo proyecto

```
[openshift@curso ~]$ oc new-project builds
```

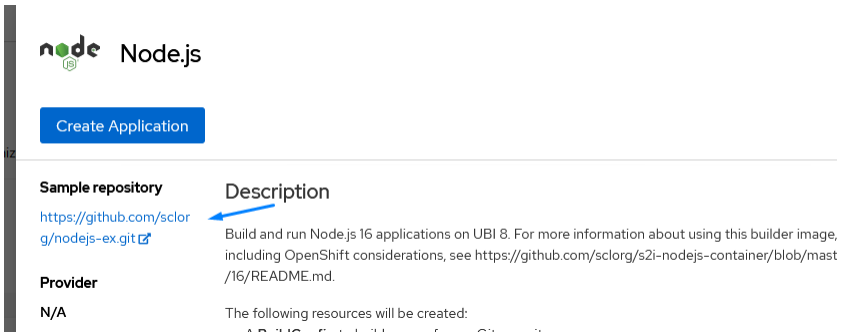
Desde la interfaz gráfica, en modo developer



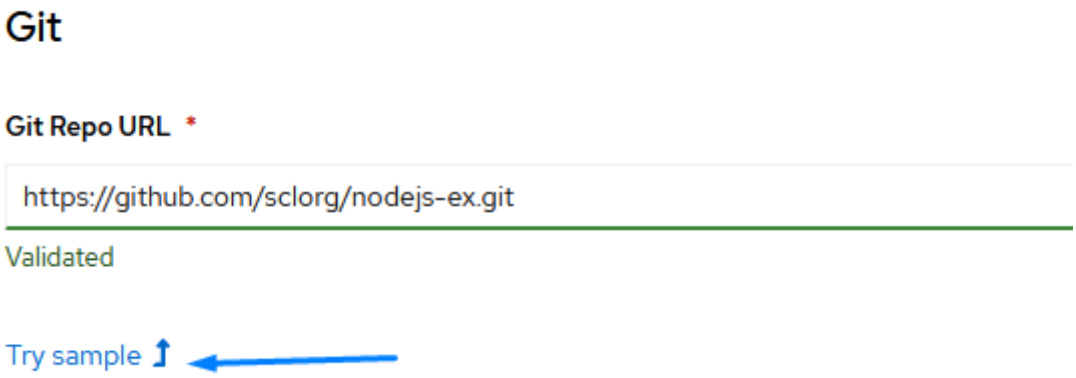
Creamos un builder



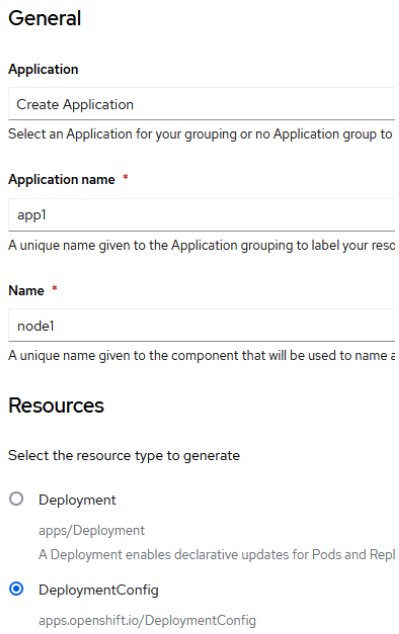

Vamos a usar el ejemplo que viene




Le damos a crear aplicación, seleccionamos el ejemplo que viene




Completamos los campos




Podemos ver que se ha creado

```
[openshift@curso ~]$ oc get bc
NAME    TYPE     FROM   LATEST
node1   Source   Git    1
[openshift@curso ~]$ oc get build
NAME      TYPE     FROM          STATUS    STARTED          DURATION
node1-1   Source   Git@a096bd2   Running   31 seconds ago
[openshift@curso ~]$ oc get pods
NAME            READY   STATUS    RESTARTS   AGE
node1-1-build   1/1     Running   0          47s
[openshift@curso ~]$ oc get is
NAME    IMAGE REPOSITORY                                                      TAGS   UPDATED
node1   default-route-openshift-image-registry.apps-crc.testing/builds/node1
```

Con el siguiente comando podemos meter el archivo YAML del build en otro archivo para modificarlo crear el nuestro propio

```
[openshift@curso ~]$ oc get bc node1 -o yaml > bc.yaml
```

Borramos las primeras líneas, como las de anotaciones y demás.

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: node1
  namespace: builds
spec:
  failedBuildsHistoryLimit: 5
  nodeSelector: null
  output:
    to:
      kind: ImageStreamTag
      name: node1:latest
  postCommit: {}
  resources: {}
```

# Crear un nuevo build con GIT (new-build)

```
openshift=>oc new-build  nodejs~https://github.com/sclorg/nodejs-ex.git --name=node2
```

Comprobaciones

```
openshift=>oc get bc
NAME    TYPE     FROM   LATEST
node1   Source   Git    1
node2   Source   Git    1
openshift=>oc get build
NAME      TYPE     FROM          STATUS     STARTED              DURATION
node1-1   Source   Git@a096bd2   Complete   19 hours ago         1m39s
node2-1   Source   Git@a096bd2   Complete   About a minute ago   1m8s
openshift=>oc get pods
NAME             READY   STATUS      RESTARTS   AGE
node1-1-b5vh6    1/1     Running     1          19h
node1-1-build    0/1     Completed   0          19h
node1-1-deploy   0/1     Completed   0          19h
node2-1-build    0/1     Completed   0          87s
openshift=>oc get dc
NAME    REVISION   DESIRED   CURRENT   TRIGGERED BY
node1   1          1         1         config,image(node1:latest)
```

```
openshift=>oc get is
NAME    IMAGE REPOSITORY                                                    TAGS     UPDA
TED
node1   default-route-openshift-image-registry.apps-crc.testing/builds/node1   latest   19 h
ours ago
node2   default-route-openshift-image-registry.apps-crc.testing/builds/node2   latest   42 s
econds ago
x1      default-route-openshift-image-registry.apps-crc.testing/builds/x1      latest   9 mi
nutes ago
```

Como podemos ver no ha creado el dc (Deployment Config)

# Otros comandos de build

Arrancar otro build

```
openshift=>oc start-build node2
build.build.openshift.io/node2-2 started
```

```
openshift=>oc get builds
NAME      TYPE     FROM          STATUS     STARTED          DURATION
node1-1   Source   Git@a096bd2   Complete   20 hours ago     1m39s
node2-1   Source   Git@a096bd2   Complete   32 minutes ago   1m8s
node2-2   Source   Git@a096bd2   Running    16 seconds ago
```

Para cancelar un build (Hemos hecho un start nuevo por eso es el 3)

```
openshift=>oc cancel-build node2-3
build.build.openshift.io/node2-3 marked for cancellation, waiting to be cancelled
build.build.openshift.io/node2-3 cancelled
```

Borrar un build config

```
openshift=>oc delete bc node1
buildconfig.build.openshift.io "node1" deleted
```

# Build con DockerFile inline

El DockerFile está en los recursos del curso

```
openshift=>oc apply -f bc-dockerfile.yaml
buildconfig.build.openshift.io/docker-input created
openshift=>oc get bc
NAME         TYPE     FROM         LATEST
docker-input   Docker   Dockerfile   1
node2          Source   Git          3
```

No puede crearlo porque en este tipo hay que crear el ImageStream antes

```
openshift=>oc create imagestream docker-input
imagestream.image.openshift.io/docker-input created
```

```
openshift=>oc start-build docker-input
build.build.openshift.io/docker-input-2 started
```

```
openshift=>oc get build
NAME             TYPE     FROM          STATUS                       STARTED          DURATION
node2-1          Source   Git@a096bd2   Complete                     22 hours ago     1m8s
node2-2          Source   Git@a096bd2   Complete                     22 hours ago     56s
node2-3          Source   Git@a096bd2   Cancelled (CancelledBuild)   21 hours ago     32s
docker-input-1   Docker   Dockerfile    Running                      58 seconds ago
docker-input-2   Docker   Dockerfile    New
openshift=>oc delete build docker-input-1
build.build.openshift.io "docker-input-1" deleted
```

```
openshift=>oc new-app docker-input --name=d1
```

```
openshift=>oc expose dc/d1 --port=80
service/d1 exposed
```

```
openshift=>oc get pods
NAME                   READY   STATUS      RESTARTS   AGE
d1-1-deploy            0/1     Completed   0          79s
d1-1-jzwn6             1/1     Running     0          75s
docker-input-2-build   0/1     Completed   0          3m56s
node1-1-b5vh6          1/1     Running     1          41h
node1-1-deploy         0/1     Completed   0          41h
node2-1-build          0/1     Completed   0          22h
node2-2-build          0/1     Completed   0          22h
openshift=>oc logs d1-1-jzwn6
Hola, estas probando un dockerfile inline
```

# Builds desde sistemas de fichero local

Los recursos los proporciona el curso, lanzamos el fichero de configuración

```
openshift=>oc apply -f bc-binary.yaml
buildconfig.build.openshift.io/binary created
```

Lanzamos un fichero que contiene el IS, si no lanzamos este dará error

```
openshift=>oc apply -f is.yaml
imagestream.image.openshift.io/binary created
```

```
openshift=>oc get build
NAME              TYPE     FROM          STATUS
ON
node2-1           Source   Git@a096bd2   Complete
node2-2           Source   Git@a096bd2   Complete
node2-3           Source   Git@a096bd2   Cancelled (CancelledBuild)
docker-input-2    Docker   Dockerfile    Complete
binary-1          Source   Git@a9484e3   Running
```

Esperamos a que se complete, levantamos una app

```
openshift=>oc new-app binary --name=b1
```

Creamos la ruta

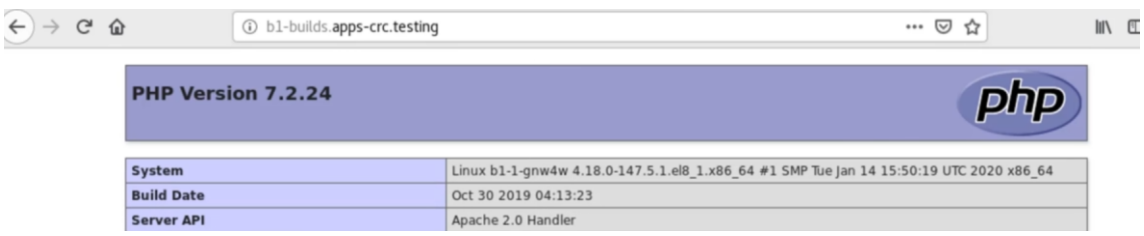
```
    'oc expose svc/b1'
    Run 'oc status' to view your app.
openshift=>oc expose svc/b1
route.route.openshift.io/b1 exposed
```

Si nos da el siguiente error es porque la maquina se ha quedado sin memoria

```
openshift=>oc get pods | grep b1

b1-1-deploy              0/1     OOMKilled   0          32s
b1-1-gnw4w               1/1     Running     0          28s
```

Lo probamos

```
openshift=>oc get route
NAME    HOST/PORT                      PATH   SERVICES   PORT       TERMINATION   WILDCARD
b1      b1-builds.apps-crc.testing            b1         8080-tcp                 None
```




Si ahora queremos modificar el código, si entramos en el directorio fuente y en index.php (Venia junto a los recursos del curso) que es el archivo modificado

Rojo = Comprimido Verde = Directorio azul = archivo,

```
openshift=>oc start-build binary --from-dir fuente --from-file --from-archive
```

Salimos del directorio fuente

```
openshift=>cd ..
openshift=>oc start-build binary --from-dir fuente
Uploading directory "fuente" as binary input for the build ...
```

Y ahora si creo una nueva aplicación utilizará la nueva imagen

```
openshift=>oc new-app binary --name=b2
```

Exponemos la ruta

```
openshift=>oc expose svc/b2
route.route.openshift.io/b2 exposed
```

```
openshift=>oc get route
NAME    HOST/PORT                        PATH   SERVICES   PORT       TERMINATION   WILDCARD
b1      b1-builds.apps-crc.testing              b1         8080-tcp                 None
b2      b2-builds.apps-crc.testing              b2         8080-tcp                 None
doc1    doc1-builds.apps-crc.testing            doc1       80                       None
node1   node1-builds.apps-crc.testing           node1      8080-tcp                 None
```

Probamos


## Triggers ImageChange

Los tres ficheros usados nos lo proporcionan en el curso

Este Trigger se lanza cuando se produce algún cambio en las imágenes que referencia el BuildConfig

```yaml
triggers:
- generic:
    secretReference:
      name: pythonapp-generic-webhook-secret
  type: Generic
- github:
    secretReference:
      name: pythonapp-github-webhook-secret
```

Lanzamos el BuildConfig

```
openshift=>oc apply -f bc-python.yaml
buildconfig.build.openshift.io/pythonapp created
```

Vemos que no se ha creado porque no tiene la imagen

```
openshift=>oc get bc
NAME           TYPE     FROM         LATEST
binary         Source   Git          2
docker-input   Docker   Dockerfile   2
node2          Source   Git          3
pythonapp      Source   Git          0
```

Creamos la imagen que está en el otro fichero

```
openshift=>oc apply -f is.yaml
imagestream.image.openshift.io/mi-imagen created
```

Y veremos que se ha disparado automáticamente (Este es un ejemplo de Trigger tipo imagen)

```
openshift=>oc get bc
NAME           TYPE     FROM         LATEST
binary         Source   Git          2
docker-input   Docker   Dockerfile   2
node2          Source   Git          3
pythonapp      Source   Git          1
```

Ahora da otro problema de que no encuentra la imagen

```
openshift=>oc get builds
NAME             TYPE     FROM            STATUS                        STARTED         DURATION
node2-1          Source   Git@a096bd2     Complete                      2 days ago      1m8s
node2-2          Source   Git@a096bd2     Complete                      2 days ago      56s
node2-3          Source   Git@a096bd2     Cancelled (CancelledBuild)    2 days ago      32s
docker-input-2   Docker   Dockerfile      Complete                      31 hours ago    1m10s
binary-1         Source   Git@a9484e3     Complete                      24 hours ago    1m10s
binary-2         Source   Binary@a9484e3  Complete                      24 hours ago    1m6s
pythonapp-1      Source   Git             New (InvalidOutputReference)
```

Para ello creamos la imagen

```
openshift=>oc create imagestream pythonapp
imagestream.image.openshift.io/pythonapp created
```

Y se lanzará el otro Trigger de imagen

```
openshift=>oc get builds
NAME             TYPE     FROM            STATUS                        STARTED         DURATION
node2-1          Source   Git@a096bd2     Complete                      2 days ago      1m8s
node2-2          Source   Git@a096bd2     Complete                      2 days ago      56s
node2-3          Source   Git@a096bd2     Cancelled (CancelledBuild)    2 days ago      32s
docker-input-2   Docker   Dockerfile      Complete                      31 hours ago    1m10s
binary-1         Source   Git@a9484e3     Complete                      24 hours ago    1m10s
binary-2         Source   Binary@a9484e3  Complete                      24 hours ago    1m6s
pythonapp-1      Source   Git             Running                       6 seconds ago
```

Vemos que ya está funcionando

```
openshift=>oc get is -o name
imagestream.image.openshift.io/b1
imagestream.image.openshift.io/b2
imagestream.image.openshift.io/binary
imagestream.image.openshift.io/d1
imagestream.image.openshift.io/docker-input
imagestream.image.openshift.io/mi-imagen
imagestream.image.openshift.io/node1
imagestream.image.openshift.io/node2
imagestream.image.openshift.io/pythonapp
```

# Rollout

Con el siguiente comando estamos diciendo de volver lanzar un DP (Deployment Config) del último DP que tengamos (Si hay 3 versiones pues de la última) Para hacer un DP de la última versión

El DP se llama php1

```
[openshift@curso ~]$ oc rollout latest dc/php1
deploymentconfig.apps.openshift.io/php1 rolled out
```

Mostrar el estado del rollout

```
[openshift@curso ~]$ oc  rollout status dc/php1
replication controller "php1-2" successfully rolled out
```

Muestra los rollout que he hecho

```
[openshift@curso ~]$ oc rollout history dc/php1
deploymentconfig.apps.openshift.io/php1
REVISION        STATUS          CAUSE
1               Complete        config change
2               Complete        manual change
```

Nos pone los datos de la revisión que le indiquemos

```
[openshift@curso ~]$ oc rollout history dc/php1 --revision=2
deploymentconfig.apps.openshift.io/php1 with revision #2
Pod Template:
  Labels:       deployment=php1-2
        deploymentconfig=php1
  Annotations:  openshift.io/deployment-config.latest-version: 2
        openshift.io/deployment-config.name: php1
        openshift.io/deployment.name: php1-2
        openshift.io/generated-by: OpenShiftNewApp
  Containers:
   php1:
    Image:      image-registry.openshift-image-registry.svc:5000/
b53c7ac69e73963cfb082c04ebad7519c537
    Ports:      8080/TCP, 8443/TCP
    Host Ports: 0/TCP, 0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>
```

# Rollback (Retroceder a una versión anterior)

Modificamos algo del archivo (El archivo está subido a GIT) y una vez modificado

```
[openshift@curso ~]$ oc start-build bc/php1
build.build.openshift.io/php1-2 started
```

Vemos que ya tiene varias revisiones

```
[openshift@curso ~]$ oc get dc
NAME    REVISION   DESIRED   CURRENT   TRIGGERED BY
perl    1          1         1         config,image(perl:latest)
php1    3          1         1         config,image(php1:latest)
[openshift@curso ~]$ oc get pods
NAME             READY   STATUS      RESTARTS   AGE
perl-1-8kq6f     1/1     Running     0          5h5m
perl-1-build     0/1     Completed   0          5h10m
perl-1-deploy    0/1     Completed   0          5h5m
php1-1-build     0/1     Completed   0          60m
php1-1-deploy    0/1     Completed   0          58m
php1-2-build     0/1     Completed   0          4m8s
php1-2-deploy    0/1     Completed   0          39m
php1-3-chl5b     1/1     Running     0          2m52s
php1-3-deploy    0/1     Completed   0          3m
```

Vemos el historial

```
[openshift@curso ~]$ oc rollout history dc/php1
deploymentconfig.apps.openshift.io/php1
REVISION        STATUS          CAUSE
1               Complete        config change
2               Complete        manual change
3               Complete        image change
```

Para volver a una versión anterior, en este caso la 2

```
[openshift@curso ~]$ oc rollout undo dc/php1 --to-revision=2
deploymentconfig.apps.openshift.io/php1 rolled back
```

Vemos que ahora tenemos 4 revisiones

```
[openshift@curso ~]$ oc rollout history dc/php1
deploymentconfig.apps.openshift.io/php1
REVISION        STATUS          CAUSE
1               Complete        config change
2               Complete        manual change
3               Complete        image change
4               Complete        image change
```

## Rolling Update

Mantienen durante cierto tiempo la versión antigua hasta que se actualiza a la nueva, es la más habitual. Mantiene los antiguos mientras crea los nuevos.

Lo primero que haremos será escalar la aplicación para que haya suficientes PODS para ver claro el ejemplo

```
[openshift@curso ~]$ oc scale dc/php1 --replicas=10
deploymentconfig.apps.openshift.io/php1 scaled
```

Importar asegurarse que la estrategia es de tipo replica

```
[openshift@curso ~]$ oc edit dc php1
```

```
spec:
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    deploymentconfig: php1
  strategy:
    activeDeadlineSeconds: 21600
    resources: {}
    rollingParams:
      intervalSeconds: 1
      maxSurge: 25%
      maxUnavailable: 25%
      timeoutSeconds: 600
      updatePeriodSeconds: 1
    type: Rolling
  template:
```

Lanzamos un nuevo deployment

```
[openshift@curso ~]$ oc rollout latest dc/php1
deploymentconfig.apps.openshift.io/php1 rolled out
```

Podemos ver que los PODS nuevos se van creando y los antiguos terminando

```
[openshift@curso ~]$ oc get pods
NAME            READY   STATUS              RESTARTS   AGE
perl-1-build    0/1     Completed           0          7h55m
php1-1-build    0/1     Completed           0          3h45m
php1-1-deploy   0/1     Completed           0          3h43m
php1-2-build    0/1     Completed           0          169m
php1-2-deploy   0/1     Completed           0          3h24m
php1-3-deploy   0/1     Completed           0          168m
php1-4-2p9gc    1/1     Terminating         0          2m23s
php1-4-5hgdn    0/1     Terminating         0          2m23s
php1-4-bqlkp    1/1     Terminating         0          162m
php1-4-deploy   0/1     Completed           0          162m
php1-4-v7zwp    1/1     Terminating         0          2m23s
php1-5-2mmpt    1/1     Running             0          19s
php1-5-6722j    1/1     Running             0          19s
php1-5-7kzd9    1/1     Running             0          14s
php1-5-8w8cf    1/1     Running             0          27s
php1-5-deploy   1/1     Running             0          32s
php1-5-fz4ln    1/1     Running             0          14s
php1-5-ll2lk    0/1     ContainerCreating   0          7s
php1-5-rj7pg    1/1     Running             0          23s
php1-5-vtgt4    1/1     Running             0          27s
php1-5-whnfx    1/1     Running             0          23s
php1-5-xwtjh    1/1     Running             0          27s
[openshift@curso ~]$
```

Como vemos ya solo están los nuevos

```
[openshift@curso ~]$ oc get pods
NAME            READY   STATUS      RESTARTS   AGE
perl-1-build    0/1     Completed   0          7h55m
php1-1-build    0/1     Completed   0          3h45m
php1-1-deploy   0/1     Completed   0          3h43m
php1-2-build    0/1     Completed   0          169m
php1-2-deploy   0/1     Completed   0          3h24m
php1-3-deploy   0/1     Completed   0          168m
php1-4-deploy   0/1     Completed   0          162m
php1-5-2mmpt    1/1     Running     0          27s
php1-5-6722j    1/1     Running     0          27s
php1-5-7kzd9    1/1     Running     0          22s
php1-5-8w8cf    1/1     Running     0          35s
php1-5-deploy   0/1     OOMKilled   0          40s
php1-5-fz4ln    1/1     Running     0          22s
php1-5-ll2lk    1/1     Running     0          15s
php1-5-rj7pg    1/1     Running     0          31s
php1-5-vtgt4    1/1     Running     0          35s
php1-5-whnfx    1/1     Running     0          31s
php1-5-xwtjh    1/1     Running     0          35s
```

## Propiedad importante del Rolling Update

Azul → máximo de PODS (Si hay 10 réplicas puede haber 12 como máximo mientras se hace el Update), rojo → mínimo de PODS (Si tiene 10 réplicas puede haber mínimo 8)

```
strategy:
  activeDeadlineSeconds: 21600
  resources: {}
  rollingParams:
    intervalSeconds: 1
    maxSurge: 20%
    maxUnavailable: 20%
    timeoutSeconds: 600
    updatePeriodSeconds: 1
  type: Rolling
```

## Recreate (Otro tipo de Update)

La diferencia con el anterior es que este primero termina los PODS antiguos y lñuego lanza los nuevos y el anterior, lo hace de forma progresiva

Primero editamos el DC

```
[openshift@curso ~]$ oc edit dc php1
```

```
spec:
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    deploymentconfig: php1
  strategy:
    type: Recreate
  template:
```

Lanzamos un nuevo DC

```
[openshift@curso ~]$ oc rollout latest dc/php1
deploymentconfig.apps.openshift.io/php1 rolled out
```

Podemos ver que va terminando los PODS antiguos, pero no crea los nuevos hasta que acaba con los antiguos

```
[openshift@curso ~]$ oc get pods | grep -e Running -e Terminating
php1-12-dtx77    1/1     Terminating   0          67m
php1-12-hl4gx    1/1     Terminating   0          67m
php1-12-hpcb7    1/1     Terminating   0          67m
php1-12-l8zvv    0/1     Terminating   0          67m
php1-12-qmh2c    1/1     Terminating   0          67m
php1-12-rbfrs    1/1     Terminating   0          68m
php1-12-rp4kl    1/1     Terminating   0          68m
php1-12-skmg4    1/1     Terminating   0          68m
php1-12-sqgkc    1/1     Terminating   0          67m
php1-12-w5gqv    1/1     Terminating   0          68m
php1-13-deploy   1/1     Running       0          16s
[openshift@curso ~]$ oc get pods | grep -e Running -e Terminating
php1-12-hpcb7    0/1     Terminating   0          68m
php1-12-l8zvv    0/1     Terminating   0          68m
php1-12-qmh2c    0/1     Terminating   0          68m
php1-12-rbfrs    0/1     Terminating   0          68m
php1-12-skmg4    0/1     Terminating   0          68m
php1-12-sqgkc    0/1     Terminating   0          68m
php1-12-w5gqv    0/1     Terminating   0          68m
php1-13-deploy   1/1     Running       0          20s
```

```
[openshift@curso ~]$ oc get pods | grep -e Running -e Terminating
php1-13-4m5sb    1/1     Running     0          14s
php1-13-6bq9p    1/1     Running     0          10s
php1-13-7md6l    1/1     Running     0          10s
php1-13-dx5wf    1/1     Running     0          10s
php1-13-g76m7    1/1     Running     0          10s
php1-13-ggxv6    1/1     Running     0          10s
php1-13-k6l5x    1/1     Running     0          10s
php1-13-mfmmg    1/1     Running     0          10s
php1-13-w6mmz    1/1     Running     0          10s
php1-13-z4lnc    1/1     Running     0          10s
```

## Triggers tipo ConfigChange

Si cambiamos una propiedad del DC y tenemos un Trigger de tipo ConfigChange

```
[openshift@curso ~]$ oc edit dc php1
deploymentconfig.apps.openshift.io/php1 edited
```

```
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 40
  test: false
  triggers:
  - type: ConfigChange
  - imageChangeParams:
      containerNames:
      - php1
      from:
        kind: ImageStreamTag
```

Nos va a disparar un nuevo DC automáticamente

```
[openshift@curso ~]$ oc rollout history dc php1
deploymentconfig.apps.openshift.io/php1
REVISION        STATUS          CAUSE
5               Complete        manual change
6               Complete        manual change
7               Complete        manual change
8               Complete        manual change
9               Complete        manual change
10              Complete        manual change
11              Complete        manual change
12              Complete        manual change
13              Complete        manual change
14              Complete        manual change
15              Running         config change
```

## Crear una Template

El fichero nos lo da el curso, creación

```
[openshift@curso plantillas]$ oc apply -f redis-plantilla.yaml
template.template.openshift.io/redis-plantilla created
```

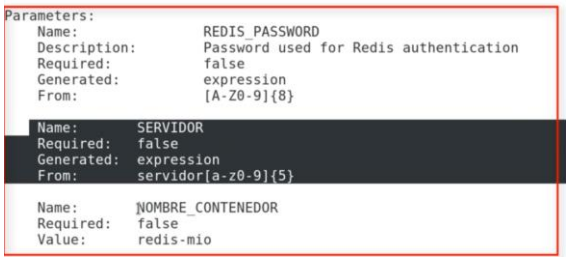
Para ver si se ha generado, podemos ver que es correcto y nos ha generado 2 de los 3 parámetros que tenía y un objeto

```
[openshift@curso plantillas]$ oc get templates
NAME              DESCRIPTION   PARAMETERS        OBJECTS
redis-plantilla   Description   3 (2 generated)   1
```

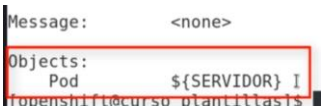
Con el comando oc describe nos da más propiedades

```
[openshift@curso plantillas]$ oc describe template redis-plantilla
Name:           redis-plantilla
Namespace:      pr2
Created:        About a minute ago
Labels:         <none>
Description:    Description
Annotations:    iconClass=icon-redis
                kubectl.kubernetes.io/last-applied-configuration={"apiVers
ls":{"redis":"master"},"metadata":{"annotations":{"description":"Descriptio
name":"redis-plantilla","namespace":"pr2"},"objects":[{"apiVersion":"v1","k
```

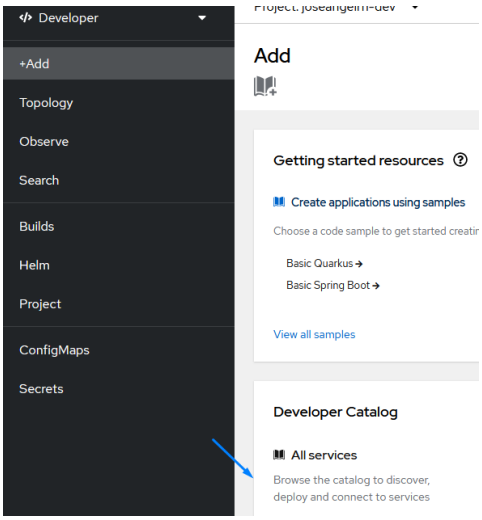
También nos indica los parametros



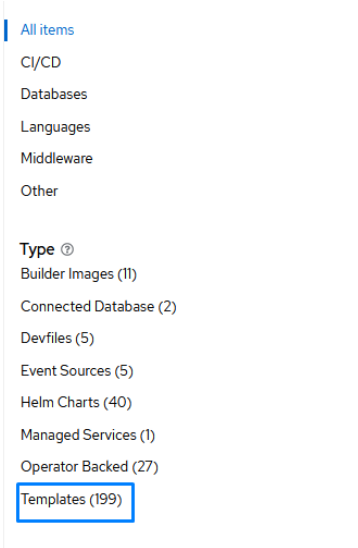

Y los objetos



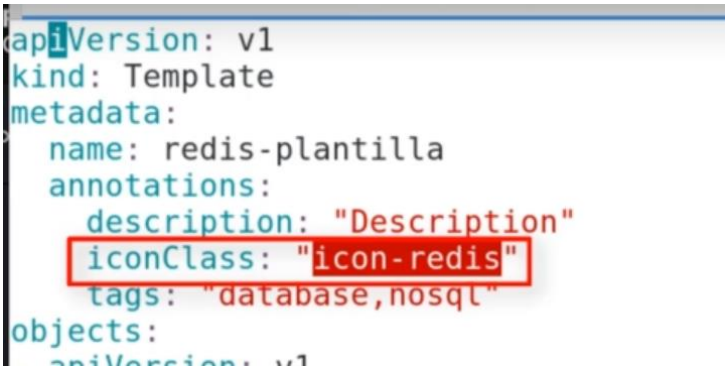

Si vamos a modo web




Nos saldrá la Template que hemos creado



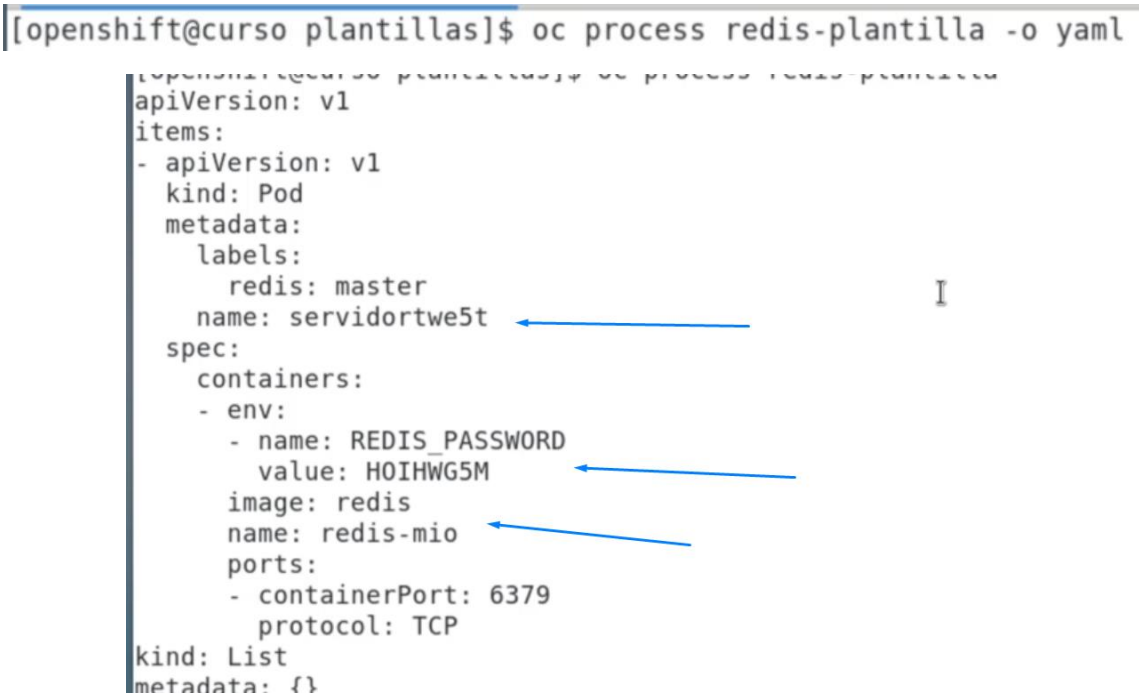
Nos sale el icono de Redis porque se lo hemos indicado en el archivo de configuración




## Comprobar los objetos que genera la plantilla

Con el siguiente comando (Process genera y muestra los objetos de la plantilla), nos devuelve un fichero Json con la estructura y los objetos que se crean


También podemos hacer en YAML



## Crear objetos desde una plantilla

Genérame los objetos de la plantilla y pásarselos al apply

```
[openshift@curso plantillas]$ oc process redis-plantilla | oc apply -f -
pod/servidorg044q created
[openshift@curso plantillas]$ oc get pod
NAME            READY   STATUS    RESTARTS   AGE
servidorg044q   1/1     Running   0          20s
```

Las plantillas cada vez que la ejecuto crear un objeto nuevo, si volvemos a lanzar el comando, vemos que los nombres son distintos

```
[openshift@curso plantillas]$ oc get pod
NAME            READY   STATUS              RESTARTS   AGE
servidorg044q   1/1     Running             0          92s
servidorqhxxy   0/1     ContainerCreating   0          12s
```

## Pasar parámetros a una plantilla

Con el siguiente comando podemos ver los parámetros que le podemos pasar a una plantilla

```
[openshift@curso plantillas]$ oc process redis-plantilla --parameters
NAME                DESCRIPTION                                GENERATOR           VALUE
REDIS_PASSWORD      Password used for Redis authentication     expression          [A-Z0-9]{8}
SERVIDOR                                                       expression          servidor[a-z0
9]{5}
NOMBRE_CONTENEDOR                                                                  redis-mio
```

Para pasarle los parámetros es tan fácil como poner el nombre de la plantilla y con la opción -p pasarle los parámetros

```
[openshift@curso plantillas]$ oc process redis-plantilla -p SERVIDOR=servidor2 | oc apply -f -
pod/servidor2 created
[openshift@curso plantillas]$
```

Si queremos más argumentos

```
[openshift@curso plantillas]$ oc process redis-plantilla -p SERVIDOR=servidor3 -p NOMBRE_CONTENEDOR=co
ntenedor1 | oc apply -f -
pod/servidor3 created
```

Otra forma sería pasarle los parámetros a través de un fichero

```
[openshift@curso plantillas]$ vi parametros.txt
```

Introducimos las propiedades

```
REDIS_PASSWORD=secret
SERVIDOR=servidor4
NOMBRE_CONTENEDOR=contenedor4
```

Y ahora lanzamos el comando pasándole como parámetro el fichero

```
[openshift@curso plantillas]$ oc process redis-plantilla --param-file=parametros.txt | oc apply -f -
pod/servidor4 created
```

## EJEMPLO CON PLANTILLAS (BuildConfig e ImageStream)

La plantilla está en los recursos del curso, la lanzamos

```
[openshift@curso plantilla_build]$ oc apply -f plantilla.yaml
template.template.openshift.io/plantilla-build created
[openshift@curso plantilla_build]$ oc get templates
NAME              DESCRIPTION                                  PARAMETERS    OBJECTS
plantilla-build   Plantilla con BuildConfig para aplicacion PHP   2 (all set)   2
```

Vemos los objetos que tiene

```
[openshift@curso plantilla_build]$ oc describe template plantilla-build
```

```
Objects:
    BuildConfig ${APLICACION}
    ImageStream ${APLICACION}
```

La lanzamos

```
[openshift@curso plantilla_build]$ oc process plantilla-build | oc apply -f -
buildconfig.build.openshift.io/aplicacion created
imagestream.image.openshift.io/aplicacion created
```

Para probarla vamos a crear una aplicación (La plantilla crea una IS que es aplicación)

```
[openshift@curso plantilla_build]$ oc get is
NAME         IMAGE REPOSITORY                                                    TAGS
 UPDATED
aplicacion   default-route-openshift-image-registry.apps-crc.testing/pr2/aplicacion   latest
 38 seconds ago
[openshift@curso plantilla_build]$ oc new-app aplicacion --name=app1
```

Creamos el servicio y listo

```
[openshift@curso plantilla_build]$ oc expose svc app1
route.route.openshift.io/app1 exposed
```

Podemos acceder a ella a través de su url

## Usar plantillas integradas en Openshift (Predefinidas)

Para verlas

```
[openshift@curso ~]$ oc get templates -n openshift
```

Vamos a usar una que es de java, lo primero es describir la plantilla para ver los parámetros que hay que pasarle, importante poner el namespace Openshift

```
[openshift@curso ~]$ oc describe template openjdk-web-basic-s2i -n openshift
```

Creamos la aplicación, a través de la plantilla

```
[openshift@curso ~]$ oc new-app --template=openjdk-web-basic-s2i --name=apli-java
```

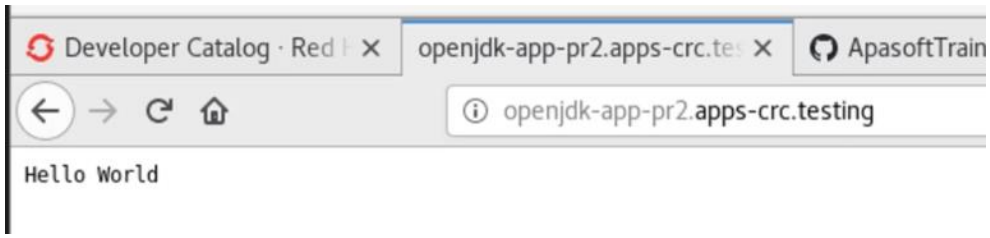
Y crea una aplicación con estos parámetros, en esté caso no ha cambiado el nombre, lo ha creado todo de forma automática

```
 * With parameters:
    * Application Name=openjdk-app
    * Java Version=latest
    * Git Repository URL=https://github.com/jboss-openshift/openshift-quickstarts
    * Git Reference=master
    * Context Directory=undertow-servlet
    * Custom http Route Hostname=
    * Github Webhook Secret=RyemSjt3 # generated
    * Generic Webhook Secret=mNDQBm7J # generated
    * ImageStream Namespace=openshift
```

Copiamos la ruta

```
[openshift@curso ~]$ oc get route
NAME          HOST/PORT                          PATH   SERVICES      PORT       TERMINATI
N   WILDCARD
app1          app1-pr2.apps-crc.testing                 app1          8080-tcp
    None
mlbparks      mlbparks-pr2.apps-crc.testing             mlbparks      <all>
    None
openjdk-app   openjdk-app-pr2.apps-crc.testing          openjdk-app   <all>
```

Y ya se puede acceder ella

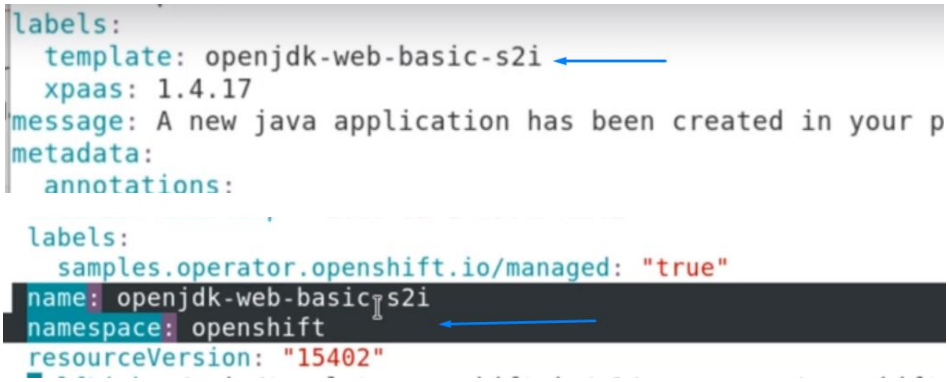



## Crear una plantilla desde otra plantilla

Generamos el YAML y lo metemos en un archivo

```
[openshift@curso ~]$ oc get template openjdk-web-basic-s2i -o yaml -n openshift > jdk.yaml
```

Si editamos el archivo que hemos creado podemos modificar la planilla, hay que modificar estos parámetros, si no lo que haremos será modificar la otra

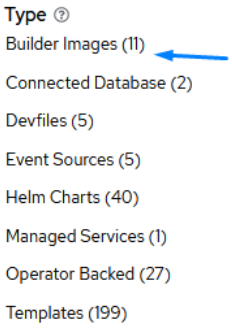


Y ahora la creamos

```
[openshift@curso ~]$ oc apply -f jdk.yaml
template.template.openshift.io/jdk-mia created
```

## Crear plantillas desde objetos existentes

Primero vamos a crear una aplicación desde el modo web en este caso de JavaScript




Lanzamos el comando para ver los objetos y elegir los que nos interesan

```
[openshift@curso ~]$ oc get all -o name
pod/web-1-build
pod/web-1-deploy
pod/web-1-f2jh9
replicationcontroller/web-1
service/web
deploymentconfig.apps.openshift.io/web
buildconfig.build.openshift.io/web
build.build.openshift.io/web-1
imagestream.image.openshift.io/web
route.route.openshift.io/web
```

Creamos la plantilla

```
[openshift@curso ~]$ oc get bc,is,dc,service,route -o yaml > plantilla_web.yaml
```

Y al fichero creado le añadimos lo necesario (Yo he tocado algunas)

```
kind: Template
apiVersion: v1
metadata:
  name: plantilla-web
objects:
- apiVersion: build.openshift.io/v1
  kind: BuildConfig
  metadata:
    name: "${APLICACION}"
  spec:
    failedBuildsHistoryLimit: 5
    nodeSelector: null
    output:
      to:
        kind: ImageStreamTag
```

```
runPolicy: Serial
source:
  git:
    uri: "${REPOSITORIO}"
  type: Git
strategy:
```

Al final del fichero añadiríamos los parámetros

```
parameters:
  -
~
```

## Recuperar información de un PV (Persistent Volume) y PVC (Persistent Volume)

Los PV precreados, podemos ver que algunos tienen su CLAIM (PVC) ya, si no somos administradores puede que no podamos acceder a ellos

```
openshift=>oc get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
          STORAGECLASS   REASON   AGE
example   5Gi        RWO            Retain           Available
          slow                      4m8s
pv0001    100Gi      RWO,ROX,RWX    Recycle          Bound       openshift-image-regi
y-storage                           30d
pv0002    100Gi      RWO,ROX,RWX    Recycle          Available
                                    30d
pv0003    100Gi      RWO,ROX,RWX    Recycle          Available
                                    30d
pv0004    100Gi      RWO,ROX,RWX    Recycle          Available
                                    30d
pv0005    100Gi      RWO,ROX,RWX    Recycle          Available
```

Podemos describir los volúmenes que queramos, podemos ver a que objeto o componente está asociado

```
openshift=>oc describe pv pv0002
Name:            pv0002
Labels:          volume=pv0002
Annotations:     <none>
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:
Status:          Available
Claim:
Reclaim Policy:  Recycle
Access Modes:    RWO,ROX,RWX
VolumeMode:      Filesystem
Capacity:        100Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /mnt/pv-data/pv0002
    HostPathType:
Events:            <none>
```

Para ver los PVC, no tenemos aún. Hay que tener en cuenta que los PVC solo podemos ver los del proyecto en el que estemos

```
openshift=>oc get pvc
No resources found in p3 namespace.
```

## Crear un PV

El fichero está en los recursos del curso

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume
  labels:
    type: local
spec:
  storageClassName: sc-ficheros
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/pv-volumen"
```

Lo lanzamos

```
openshift=>oc apply -f pv.yaml
persistentvolume/pv-volume created
```

Todavía no hay CLAIM, pero porque no lo hemos solicitado aún

```
openshift=>oc get pv | grep volume
pv-volume   20Gi       RWO            Retain           Available
           sc-ficheros                 34s
```

## Crear un PCV y Asociarlo a un PV

Importante que el CLAIM esté vació y saber la STORAGECLASS

```
openshift=>oc get pv pv-volume
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
pv-volume   20Gi       RWO            Retain           Available           sc-ficheros             33s
```

El fichero para crear el CLAIM está en los recursos del curso

Importante que El STORAGECLASS coincida con el STORAGECLASS del PV

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-claim
spec:
  storageClassName: sc-ficheros
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Lo creamos

```
openshift=>oc apply -f pvclaim.yaml
persistentvolumeclaim/pvc-claim created
openshift=>oc get pvc
NAME        STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-claim   Bound    pv-volume   20Gi       RWO            sc-ficheros    4s
```

Vemos que el PV ha cambiado

```
openshift=>oc get pv pv-volume
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM           STORAGECLASS   REASON   AGE
pv-volume   20Gi       RWO            Retain           Bound    p3/pvc-claim    sc-ficheros             3m58
s
```

## Usar un almacenamiento desde un DepoymentConfig

Los recursos los proporciona el curso

Vamos a levantar un Postgres per antes es importante esto

```
      volumes:
      - name: postgres-db-volume
        persistentVolumeClaim:
        claimName: pvc-claim
    est: false
```

```
openshift=>oc get pvc
NAME        STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-claim   Bound    pv-volume   20Gi       RWO            sc-ficheros    3h48m
```

Lo ejecutamos

```
openshift=>oc apply -f postgres.yaml
deploymentconfig.apps.openshift.io/postgres-db created
```

Si nos loggeamos y accedemos a donde hemos guradado la BBDD podemos ver los archivos

```
openshift=>oc rsh postgres-db-1-mfg82
# cd /var/lib
# ls
apt  dpkg  misc  pam  postgresql  systemd  ucf
# cd postgresql/data
# pwd
/var/lib/postgresql/data
# ls
base          pg_hba.conf    pg_notify     pg_stat      pg_twophase  postgresql.auto.conf
global        pg_ident.conf  pg_replslot   pg_stat_tmp  PG_VERSION   postgresql.conf
pg_commit_ts  pg_logical     pg_serial     pg_subtrans  pg_wal       postmaster.opts
pg_dynshmem   pg_multixact   pg_snapshots  pg_tblspc    pg_xact      postmaster.pid
# exit
```

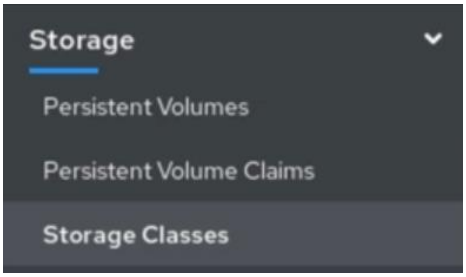
Pero físicamente lo grabamos aquí

```
openshift=>oc describe pv pv-volume
Name:            pv-volume
Labels:          type=local
Annotations:     kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"PersistentVolume","metadata":{"annotations":{}
e":"local"},"name":"pv-volume"},"spec":{"accessMod...
                 pv.kubernetes.io/bound-by-controller: yes
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    sc-ficheros
Status:          Bound
Claim:           p3/pvc-claim
Reclaim Policy:  Retain
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        20Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /mnt/data/pv-volumen
    HostPathType:
```

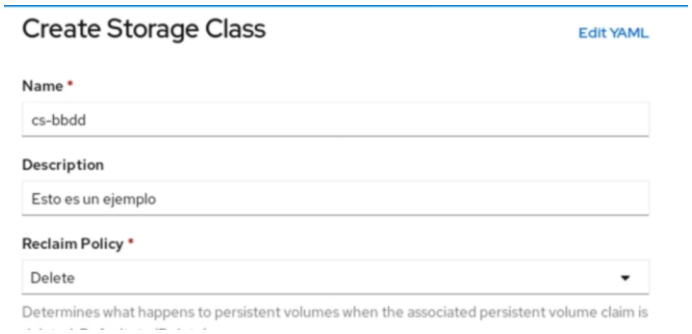
# StorageClass: Almacenamiento dinámico

La utilidad de esto es que podemos tener dentro de un StorageClass varios PV y cuando levantemos aplicaciones en los PVC automáticamente cojan El VP que necesiten depende de su capacidad

Dentro de administrador nos vamos a

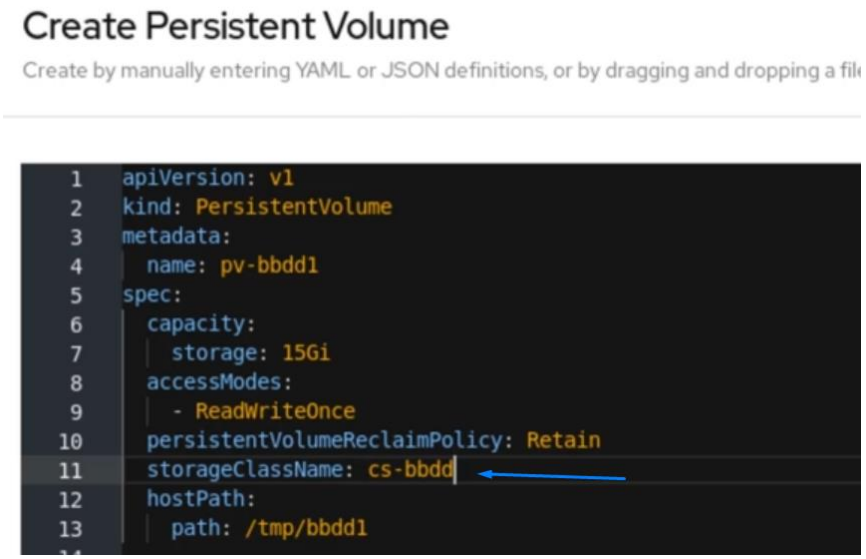



Y creamos uno

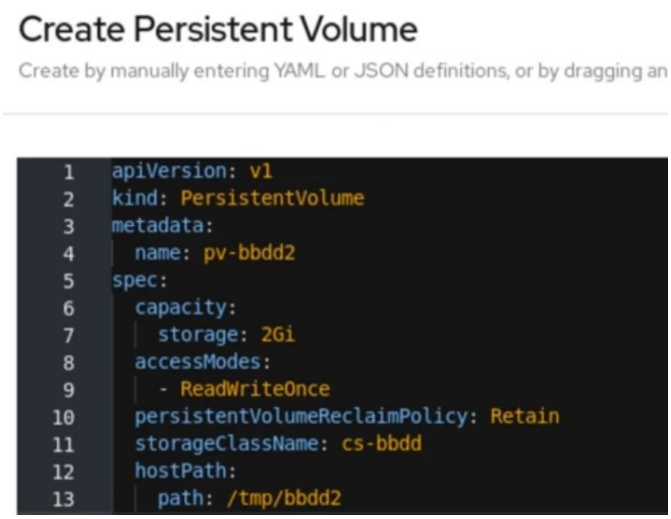



Como trabajamos crc y no tenemos otro proveedor elegimos este que es local

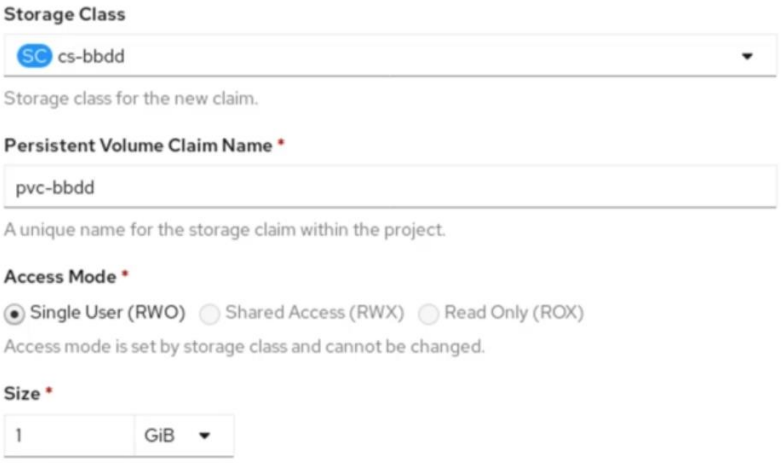

Ahora nos vamos a PV y creamos uno, ponemos el nombre del creado antes



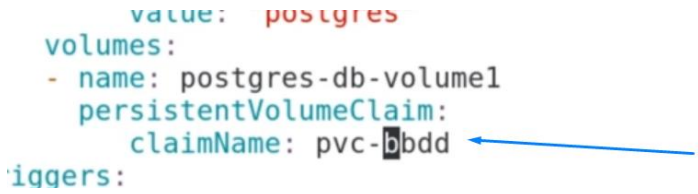
Creamos otro, pero con menos almacenamiento




Ahora creamos el PVC, 9importante seleccionar la StorageClass creada antes




Vamos a levantar un Postgres ahora, lo hemos utilizado en anteriormente ya, pero antes es importante



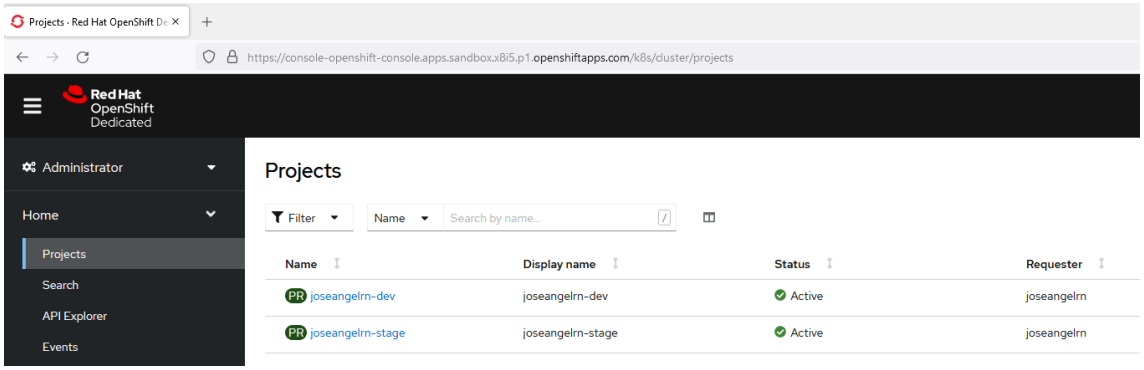

Lanzamos el fichero

```
openshift=>oc apply -f postgres.yaml
deploymentconfig.apps.openshift.io/postgres-db1 created
```

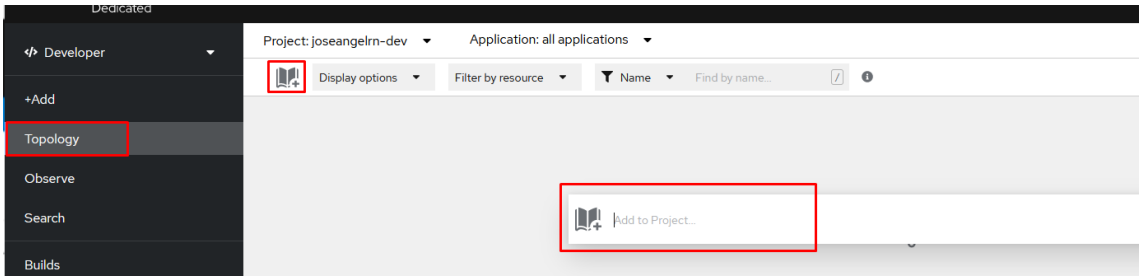
# MODO ONLINE

Accedemos a Openshift online

Creamos una cuenta

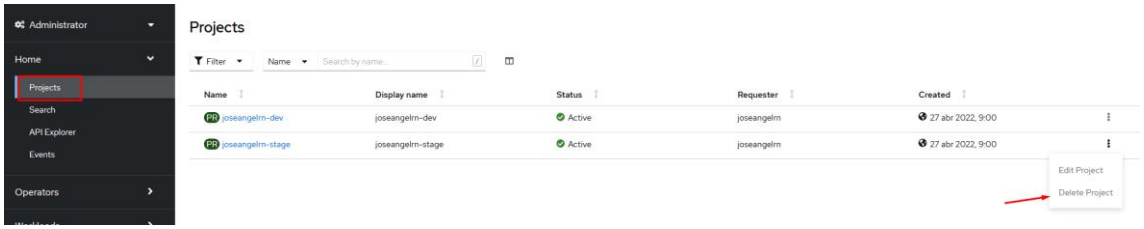



## Crear un proyecto

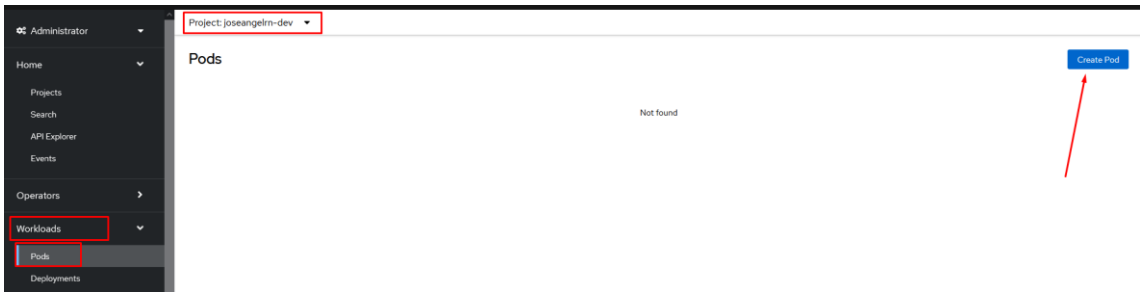



Si creo un namespace también creo un proyecto, si lo borro igual, si creo un proyecto también me crea un namespace

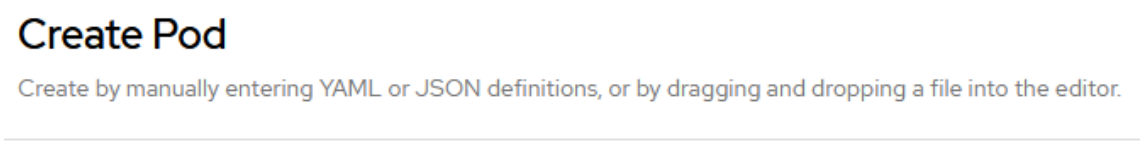
## Borrar un proyecto

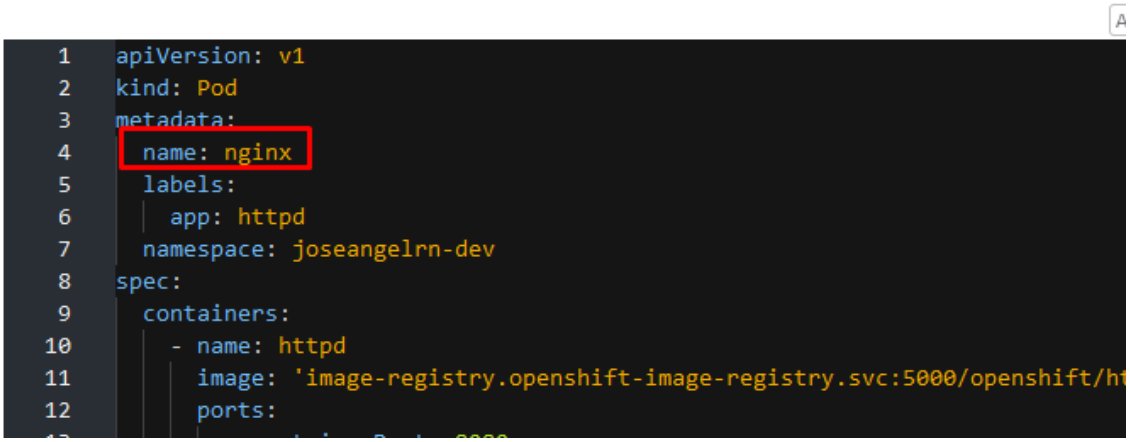



## Crear un POD

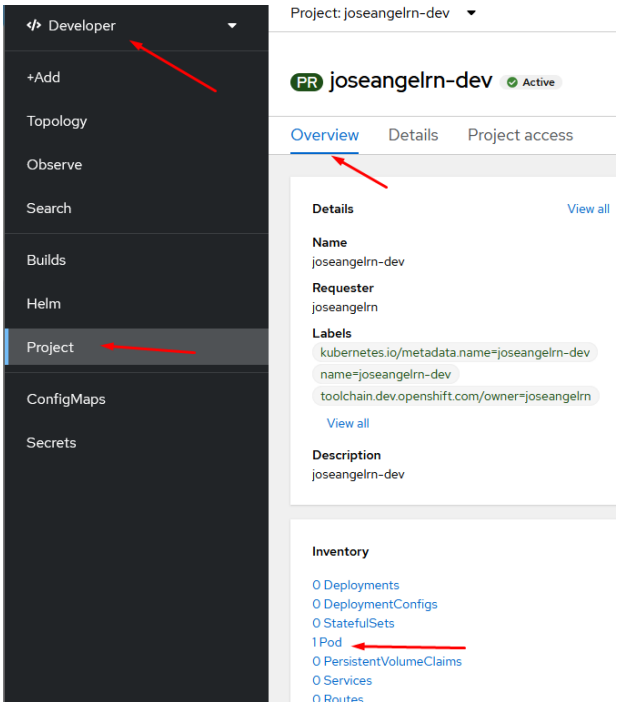



Aquí introducimos el nombre (Se podrá modificar después desde el yaml)



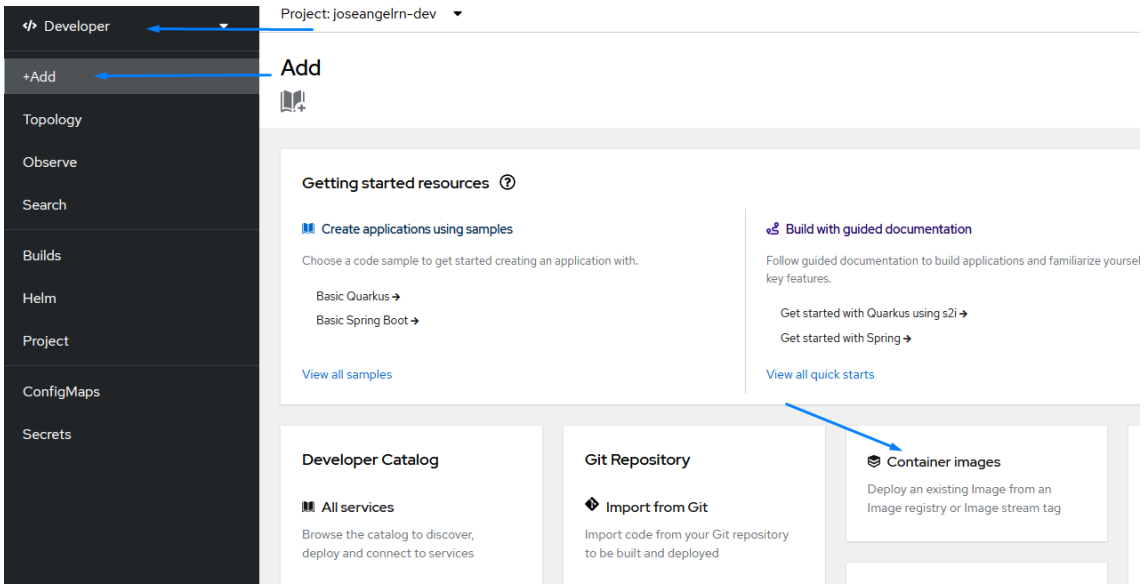




Desde developer podemos ver que se ha creado

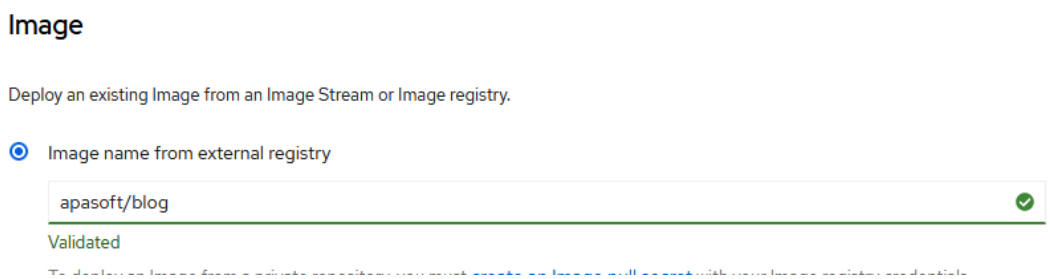



# Despliegue de una imagen

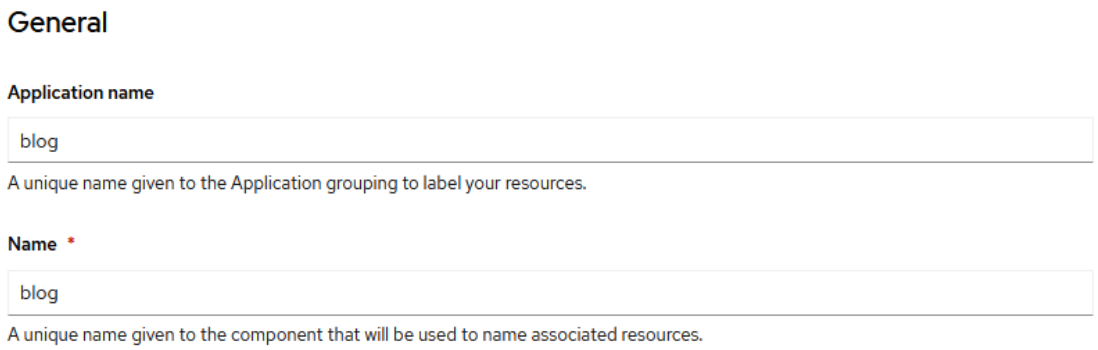
Desde modo developer. Accedemos a lo siguiente.



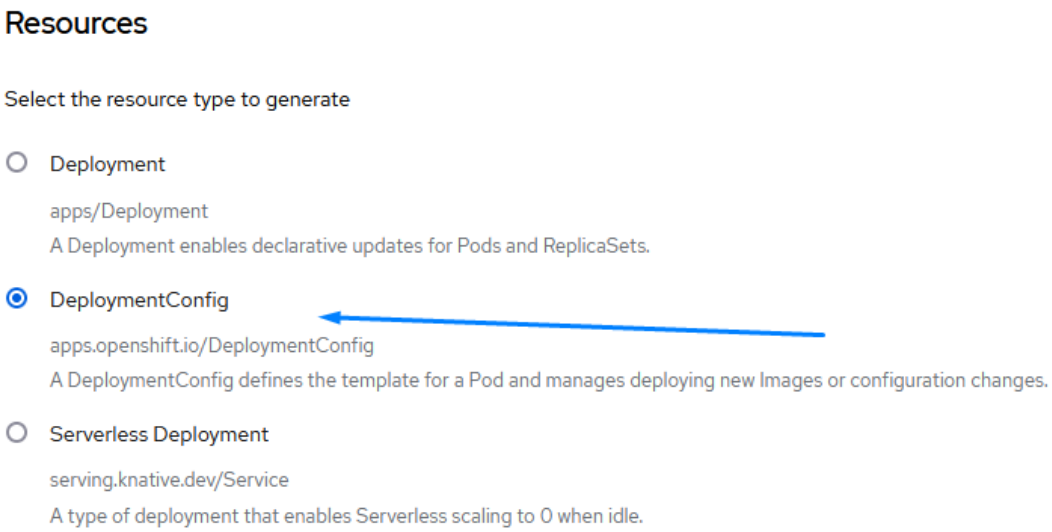

Introducimos la imagen y esperamos que valide si la encuentra



Introducimos el nombre (Se suele poner el mismo)



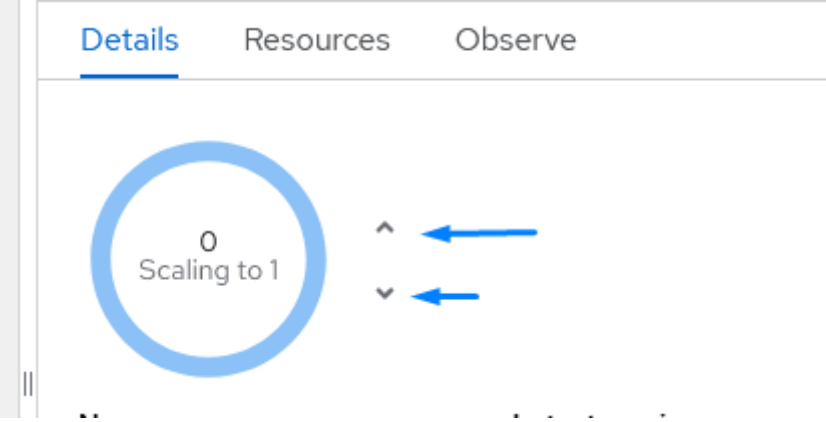

Seleccionamos como lo queremos generar, el Deployment es al estilo kubernetes



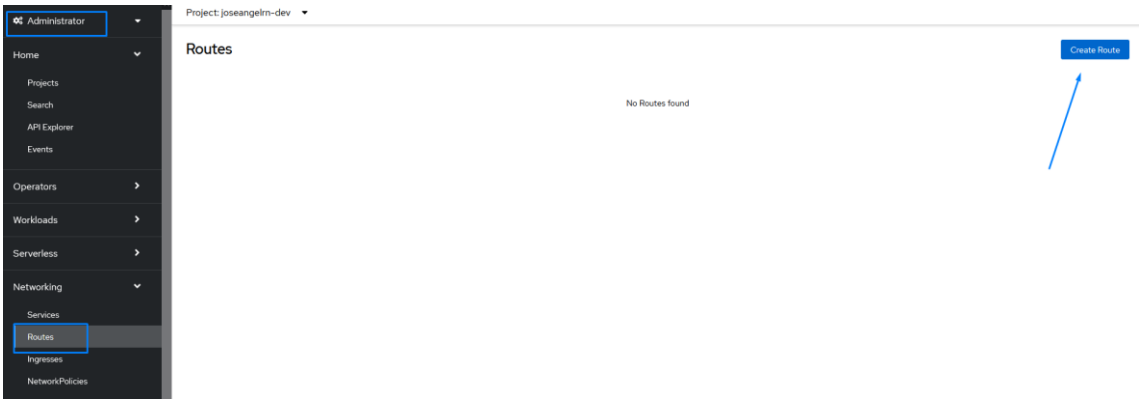

Desmarcamos lo siguiente porque lo haremos posteriormente a mano


Se ha creado, si pinchamos en el podemos ver los detalles

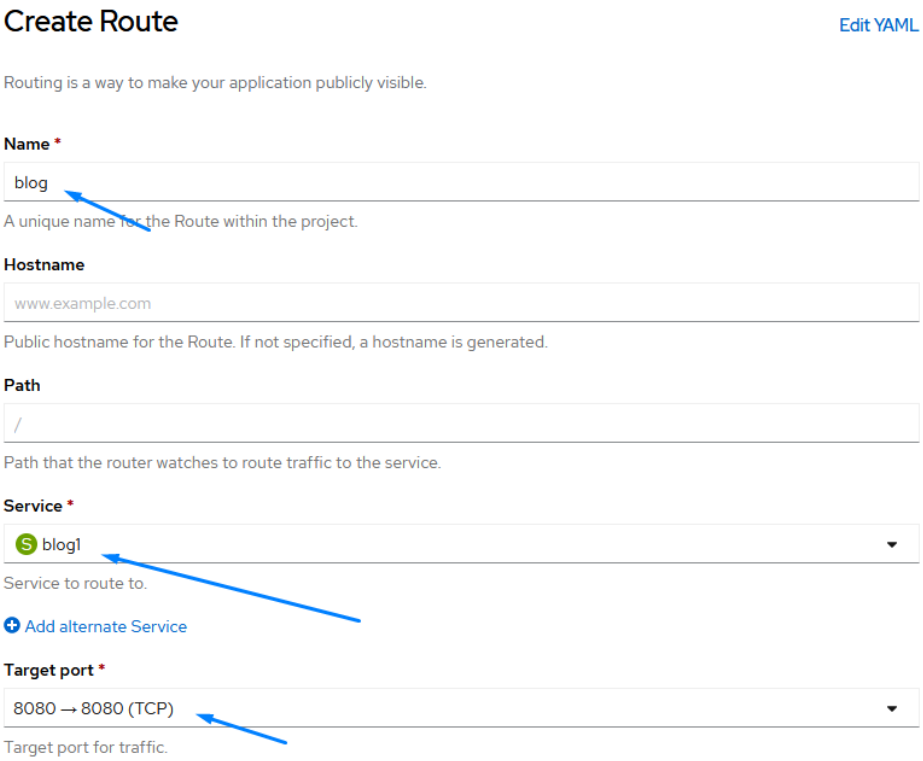

Podemos agregar POD




Ahora creamos el route, desde el administrador

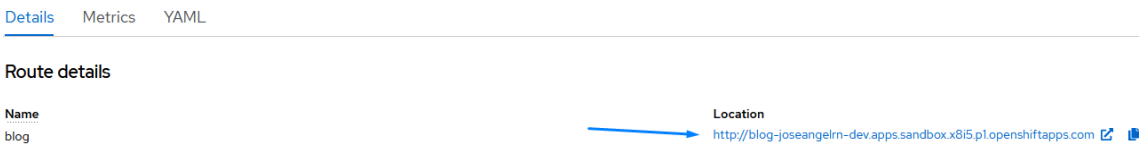



Se llama el servicio blog1 porque he creado otro

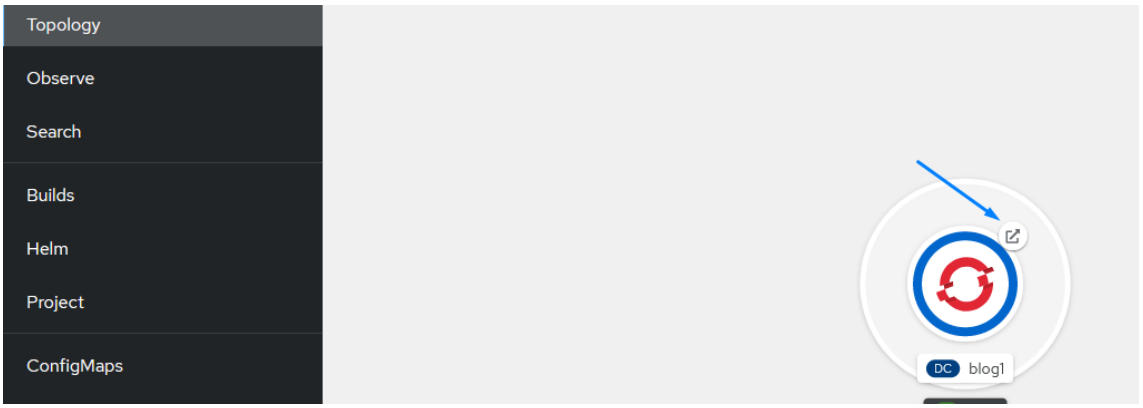



Tras esto nos salen los detalles y demás

Para probarlo desde los detalles nos dará el enlace



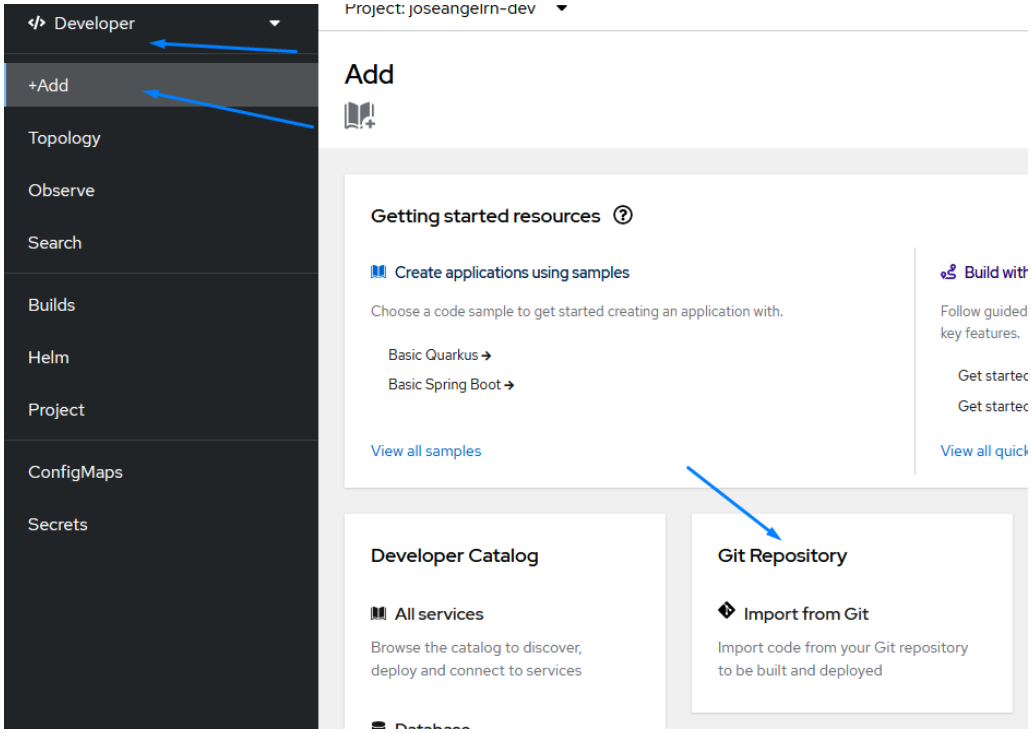

O también desde el modo developer también podriamos




Prueba

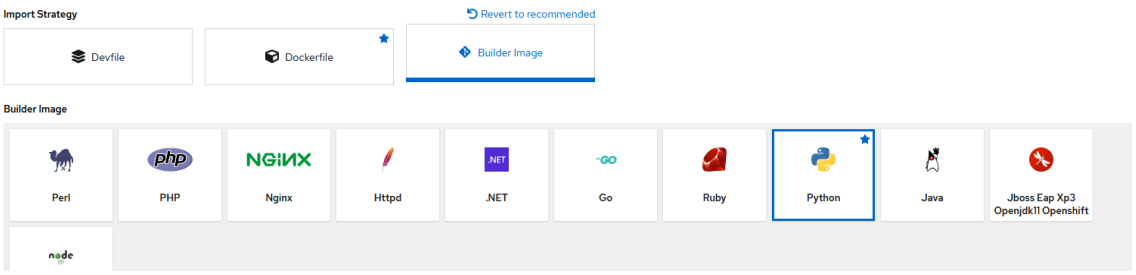

# Despliegue de una aplicación desde código fuente en un repositorio GIT

Desde developer, vamos a add y seleccionamos repositorio Git



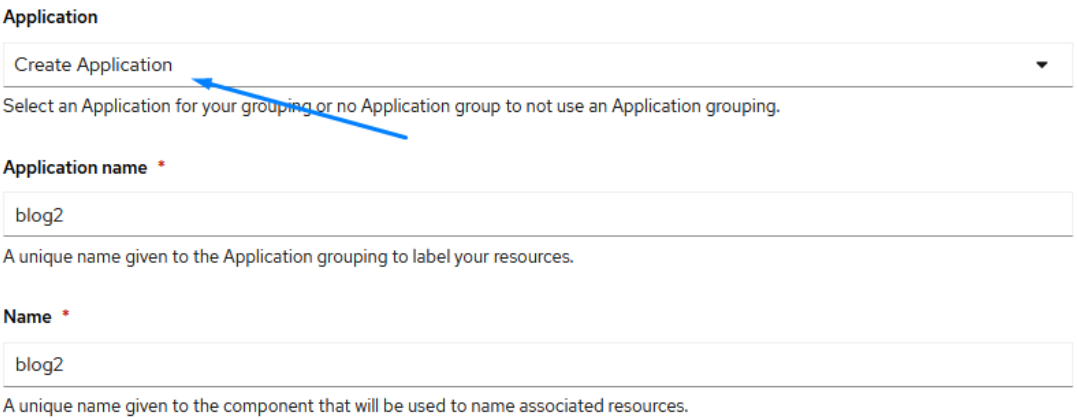

Introducimos la ruta donde está la imagen


Lo siguiente es la estrategia

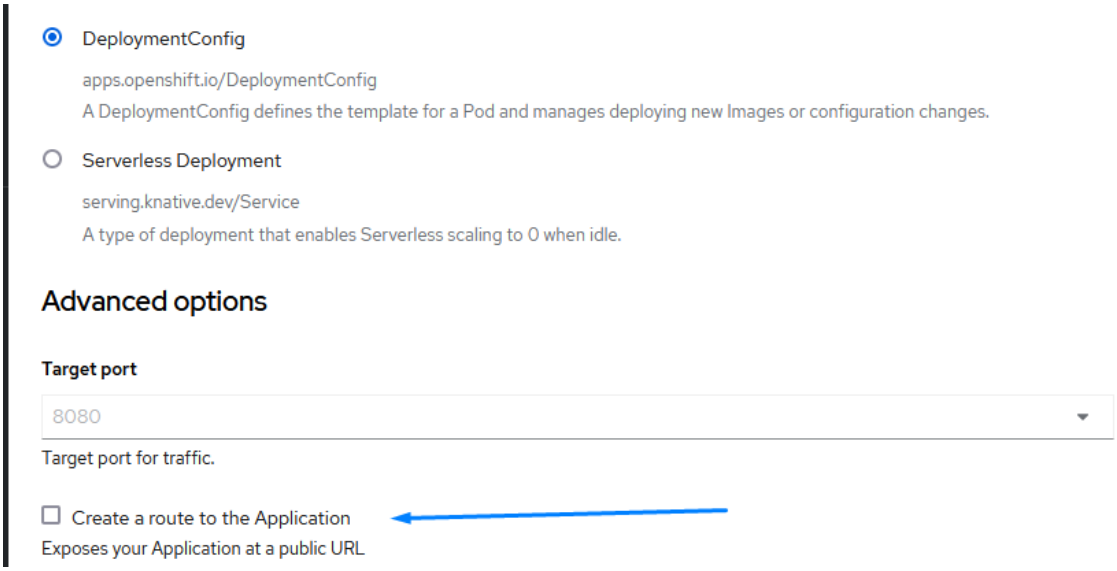


Seleccionamos la versión

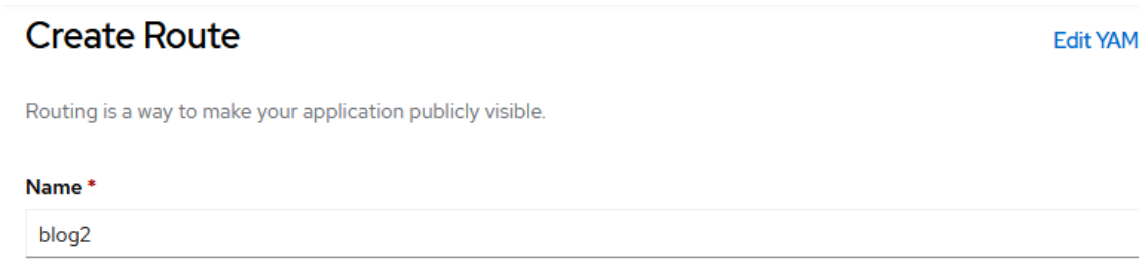

Creamos una aplicación nueva



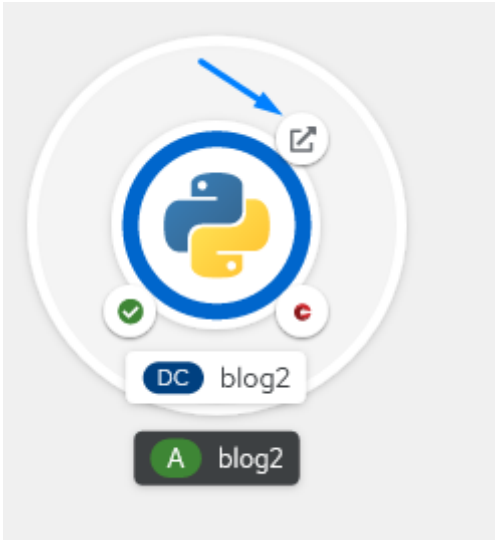

Quitamos el create route y seleccionamos el DeploymentConfig




Creamos un route desde NetWorking en la pestaña de administrador

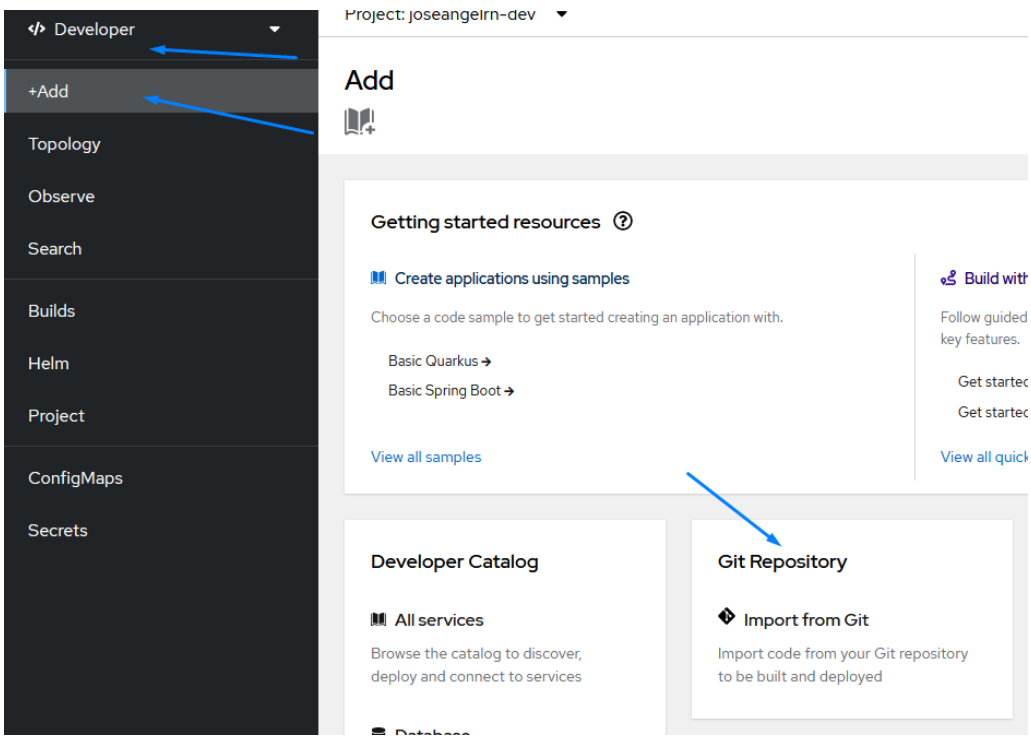



Ya podríamos probarla



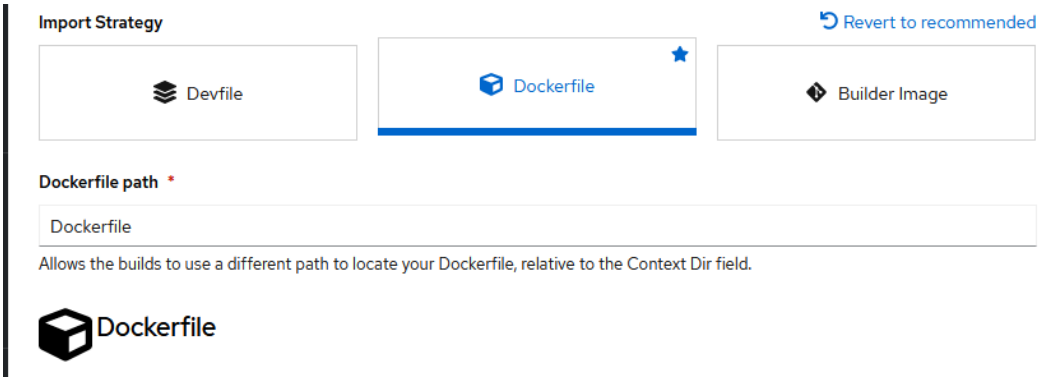

## Despliegue de una aplicación desde Docker file

Haríamos lo mismo que el anterior

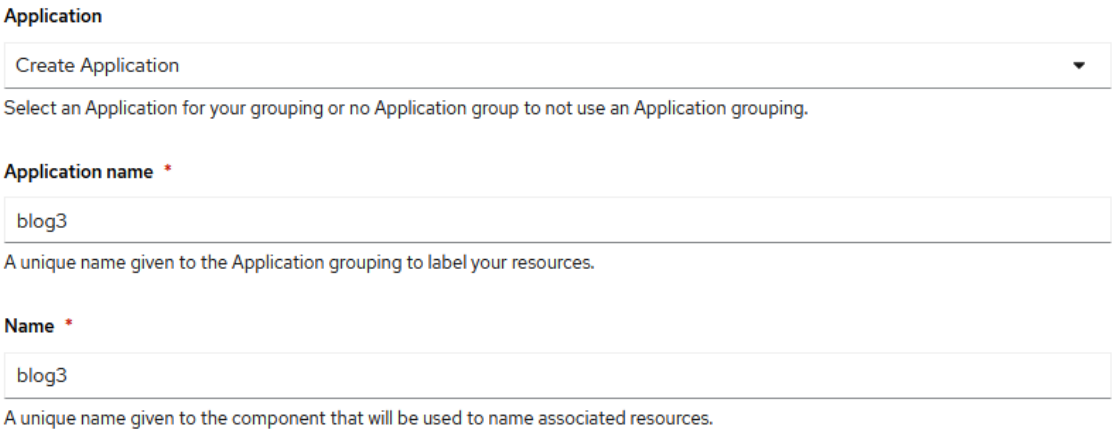


Escribimos el repositorio


Esta vez la estrategia será Docker file




Creamos una aplicación




Elegimos esto

DeploymentConfig

apps.openshift.io/DeploymentConfig

A DeploymentConfig defines the template for a Pod and manages deploying new Images or configuration changes.

Serverless Deployment

serving.knative.dev/Service

A type of deployment that enables Serverless scaling to 0 when idle.

## Advanced options

**Target port**

8080

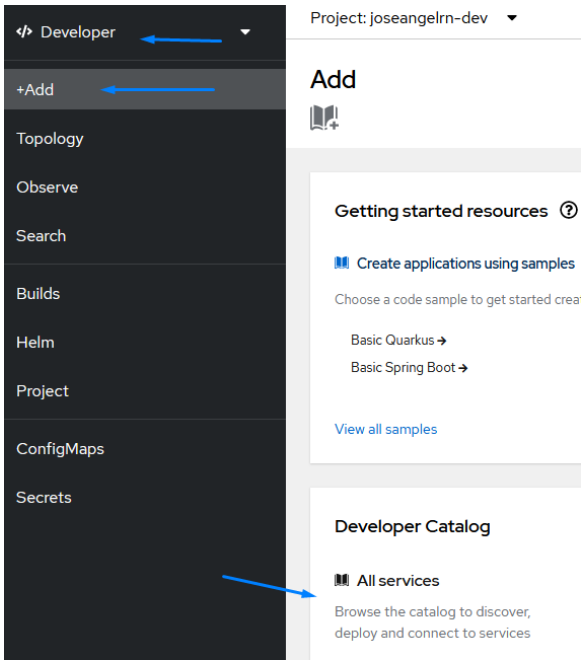
Target port for traffic.

Create a route to the Application

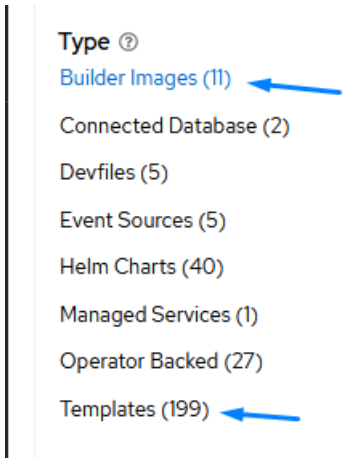
Exposes your Application at a public URL

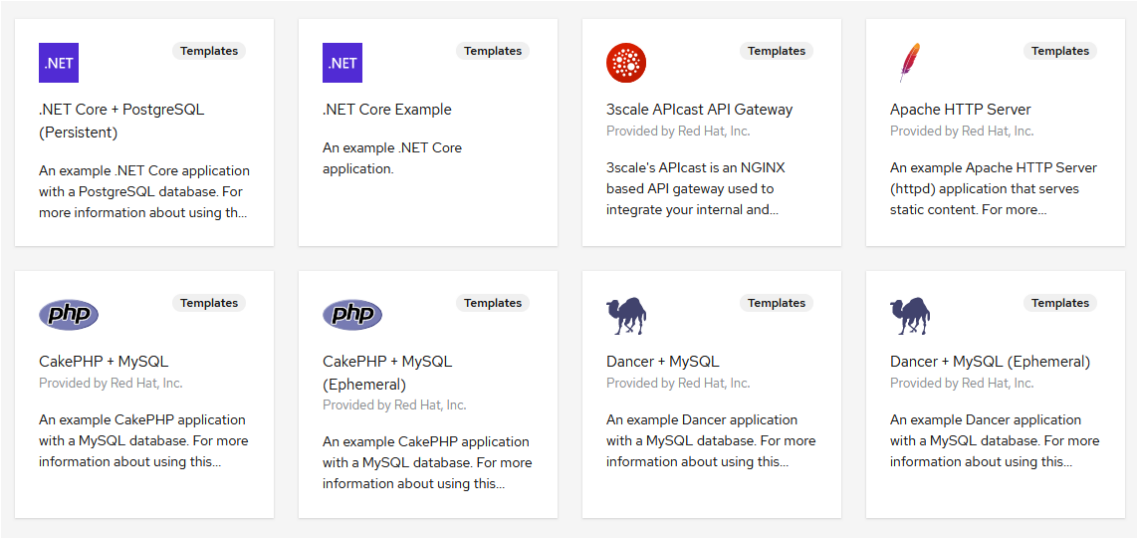
# Despliegue de una aplicación desde el catálogo

Lo primero será elegirlo desde developer

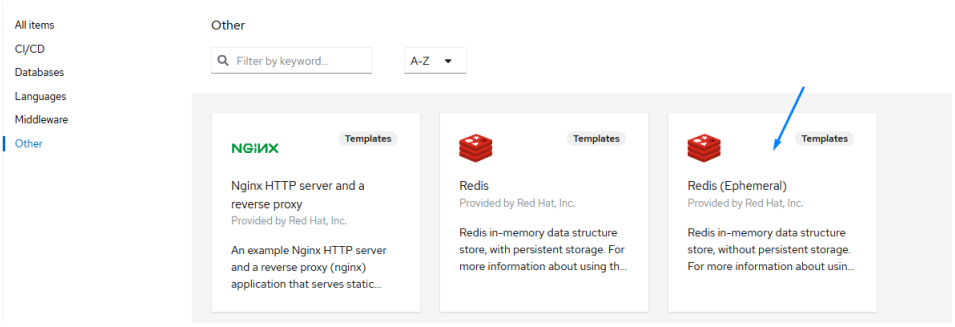


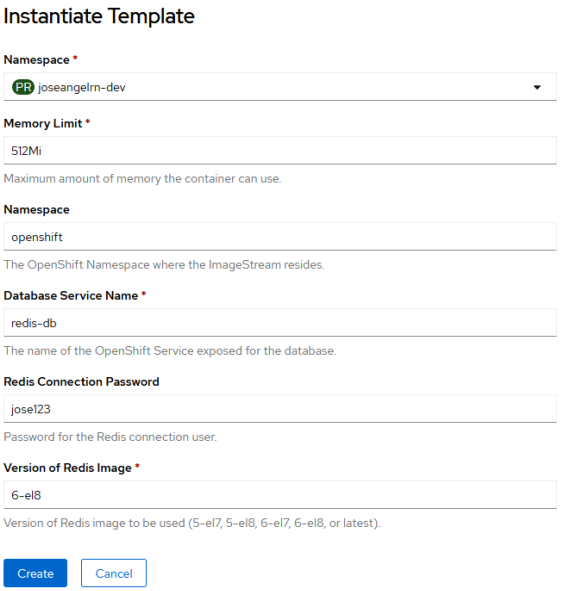

Tenemos dos interesantes, las Builder Images son para lo que hemos hecho anteriormente y las Templates son plantillas ya creadas



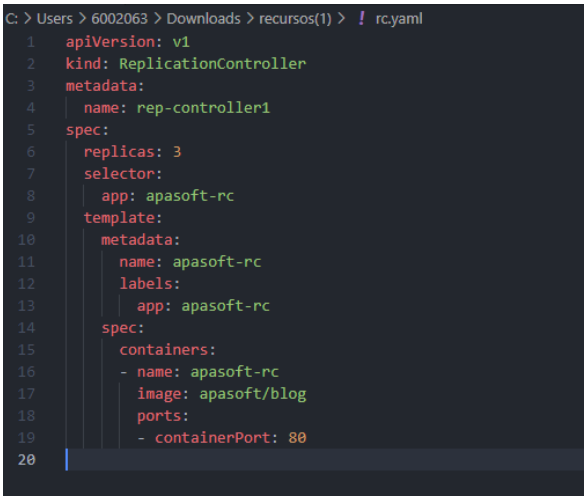

Por ejemplo, crearemos una:



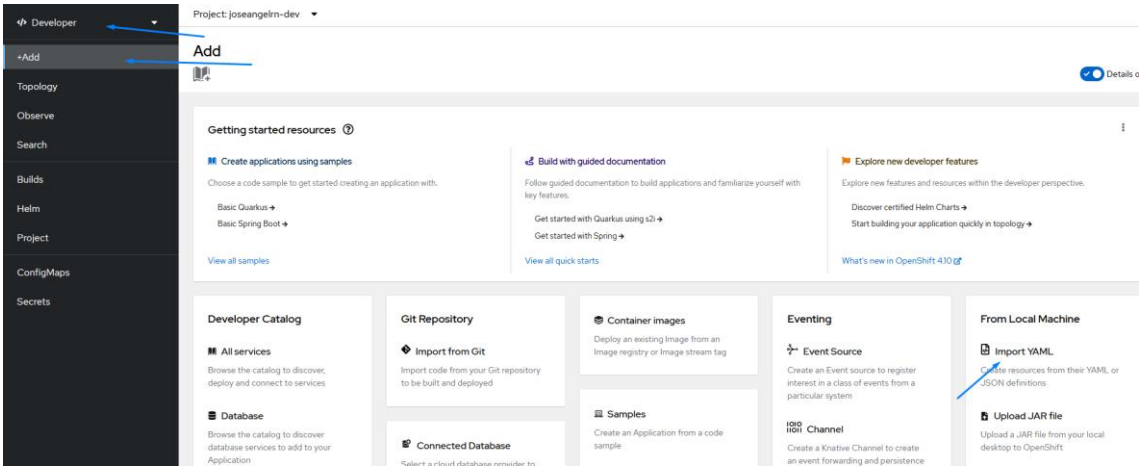




# Crear un recurso YAML

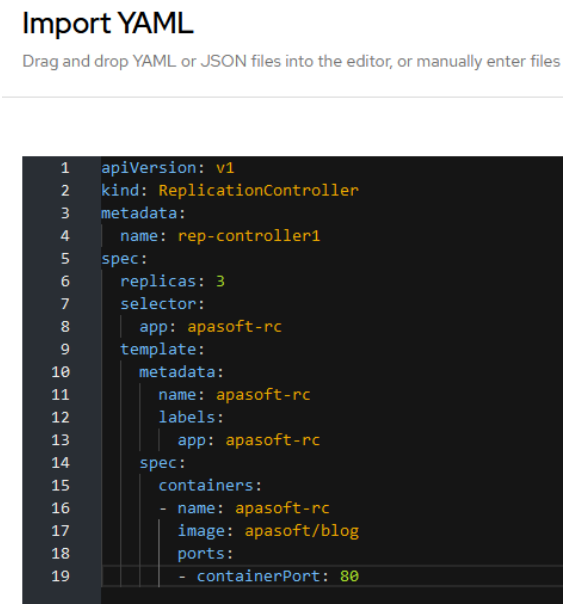
El fichero YAML lo proporciona el curso




Desde el developer importamo un YAML



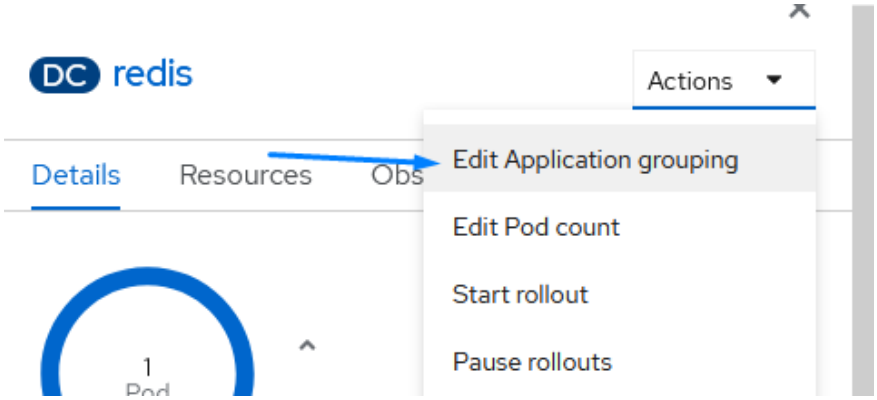
Copiamos y pegamos



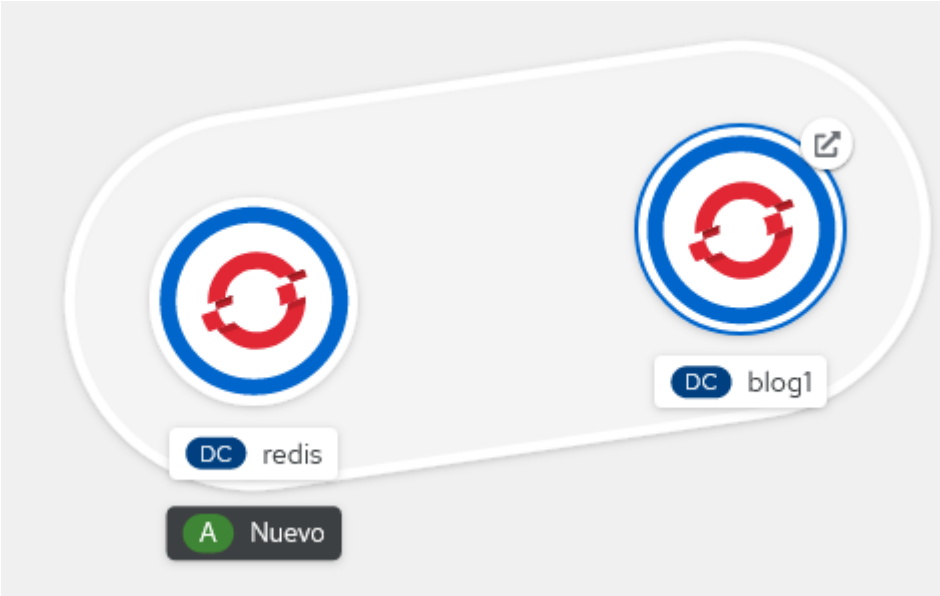

Le damos a crear

## Aplication groupin (Agrupación de aplicaciones)

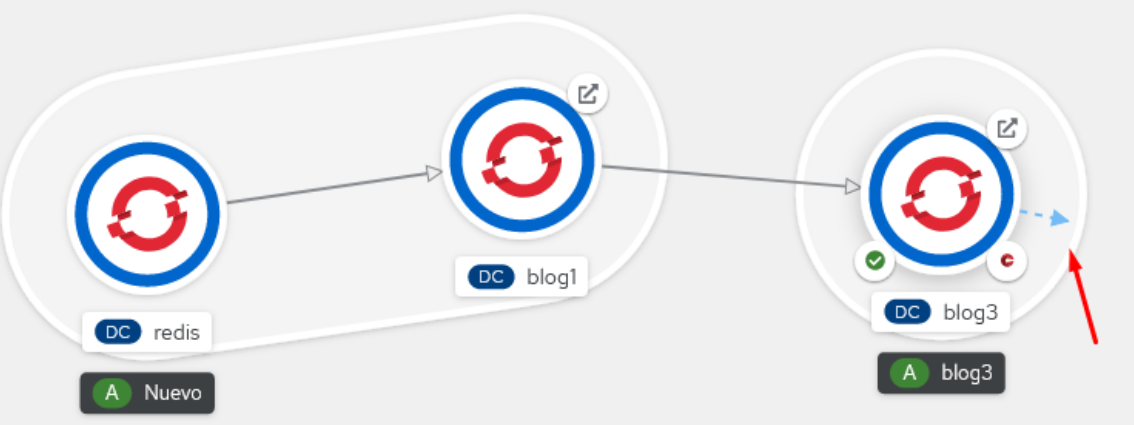
Para ellos desde developer → topología, seleccionamos la aplicación



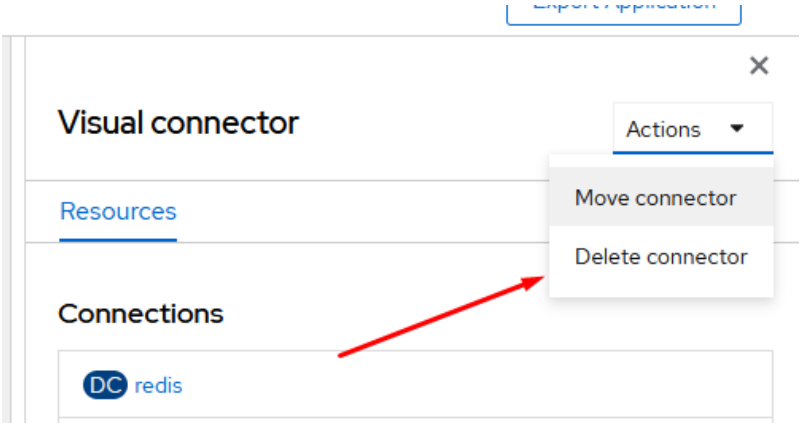

Creamos una nueva y ya estaría, si queremos agregar más aplicaciones al mismo grupo, sería repetir lo mismo y seleccionar el mismo grupo, así queraría




Hay otro tipo que es arrastrando la flecha azul que sale para conectar directamente una aplicación con otra

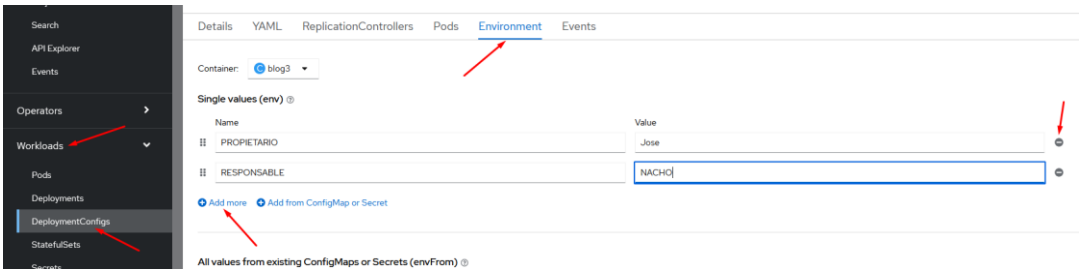



Para borrarlo hacer clic en la flecha gris




## Variables

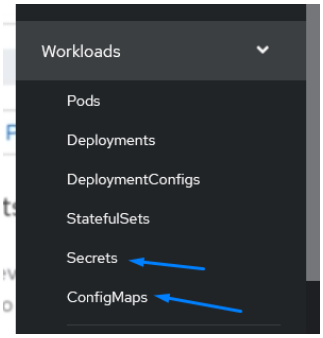
Para añadir, crear, modificar y eliminar variables, vamos a Deployment Config desde el administrador




Importante darle a guardar

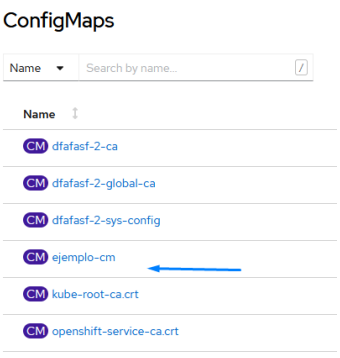
# ConfigMaps y Secrets

Se encuentran en la parte de administrador en workloads

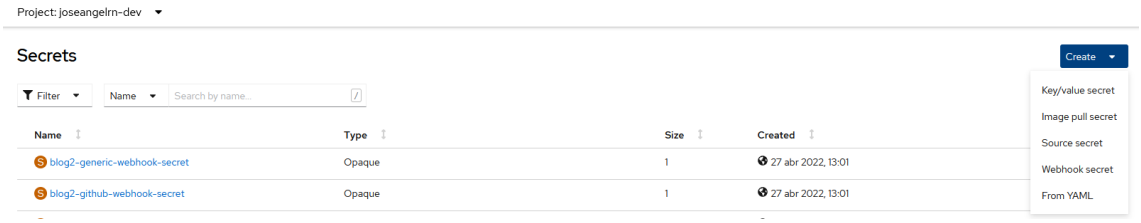



Para crear un ConfigMaps

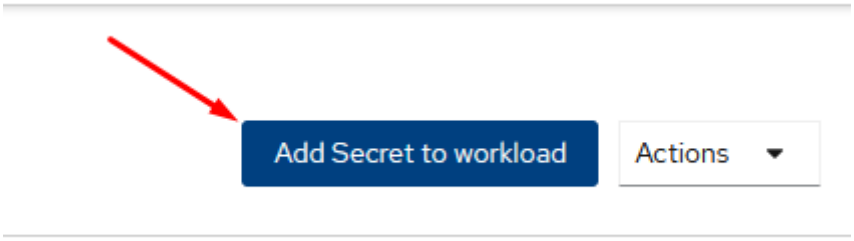

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ejemplo-cm
  namespace: joseangelrn-dev
data:
  example.property.1: hola
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```



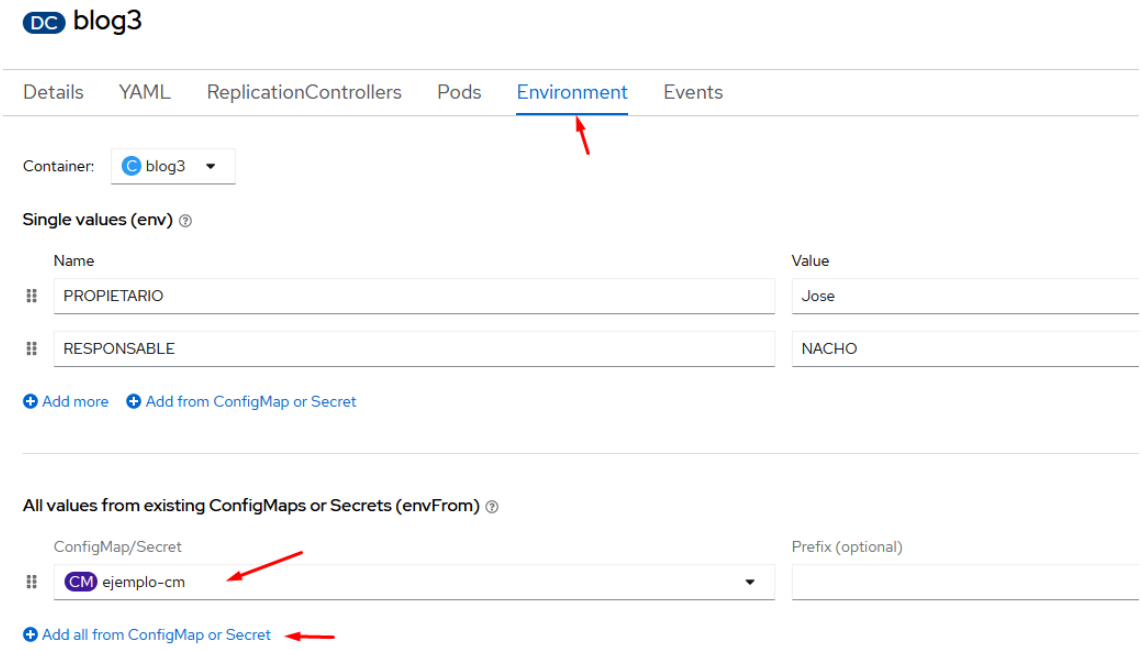
Para crear un secret es igual que los ConfigMaps, pero seleccionando el tipo




Si nos vamos dentro de un secret, podemos añadirlo a un workload

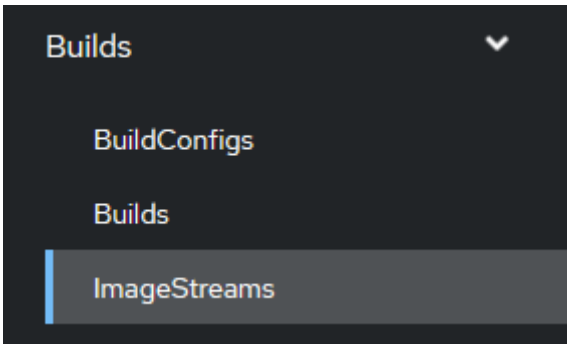



Si vamos dentro de Deployment Config podemos añadir tanto secrets como ConfigMaps

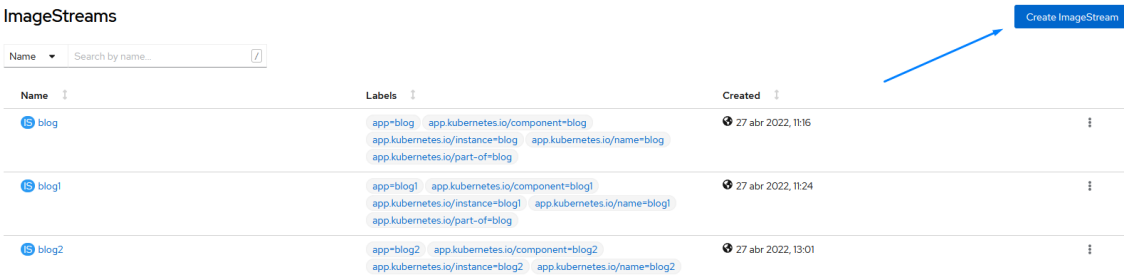



## ImageStream

Desde administrador

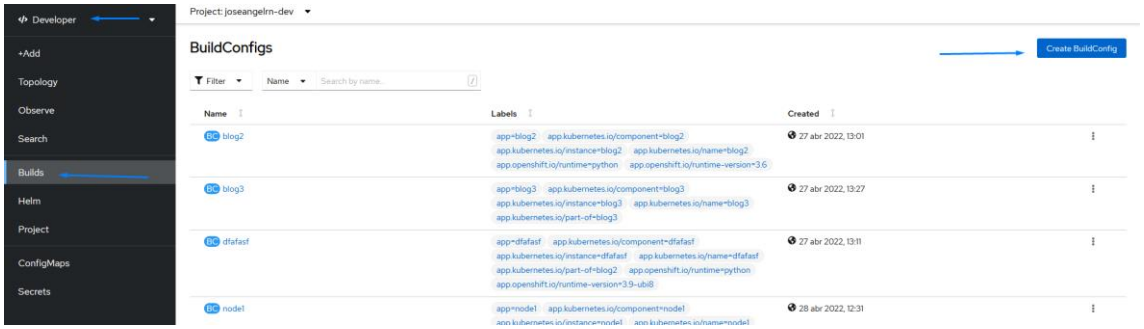



Aquí podríamos ver las que tenemos y crear nuevas

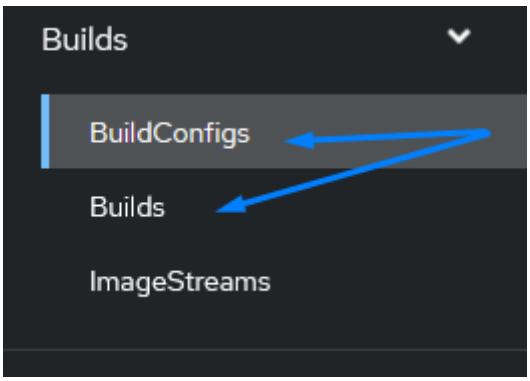



# BuildConfig

Desde developer podemos ver los builds, podemos crear uno

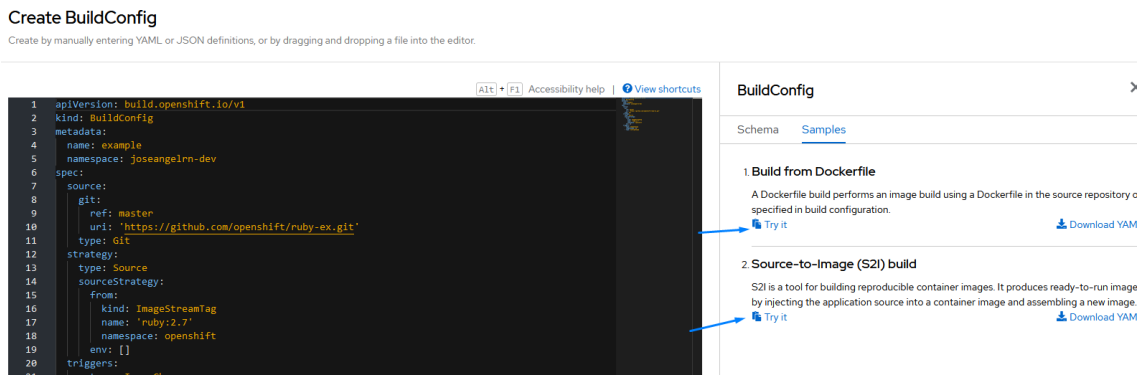



También podemos acceder desde el administrador



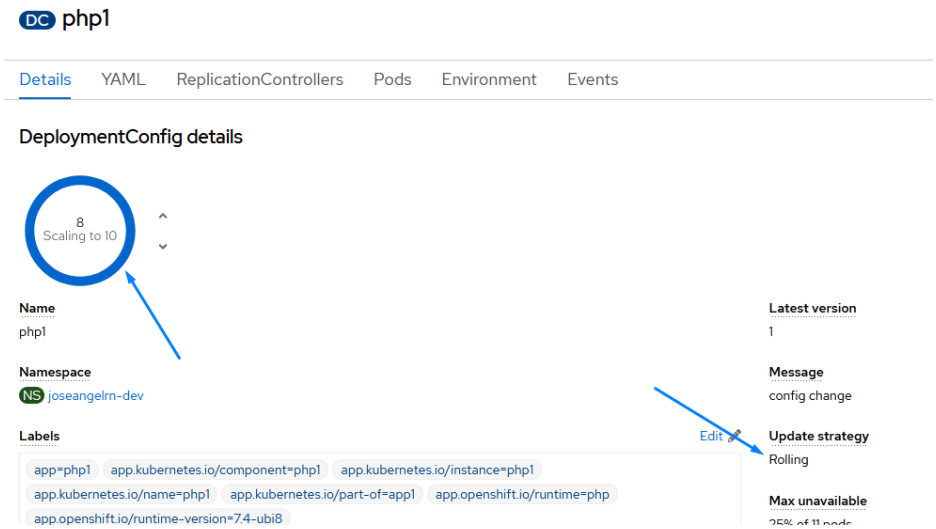

**LOS BUILDS SON LOS HIJOS DE LOS BUILSCONFIG**

Nos proporcionan plantillas




# Rolling Update

Para hacerlo desde modo gráfico es importante que esté escalada y la estrategia de Update será Rolling.

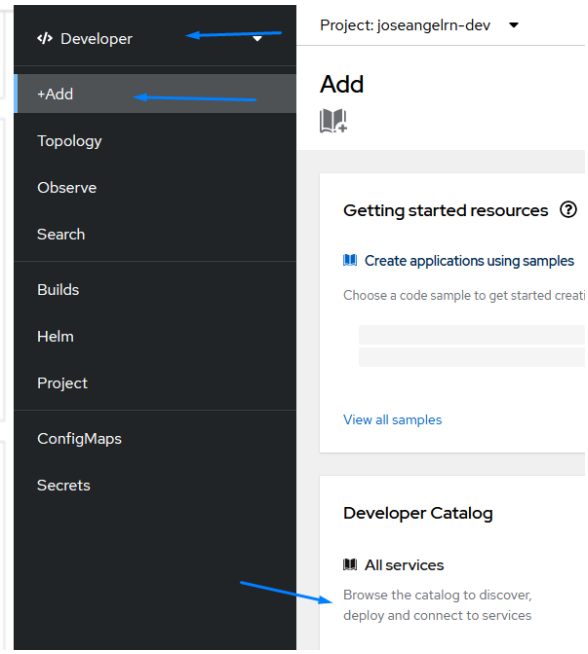



Para hacer el Rollout vamos a start Rollout

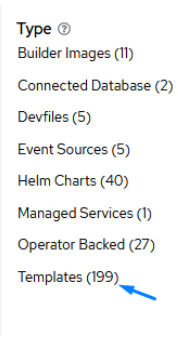

Como observamos a la derecha se están creando los nuevos y a la izquierda terminando los antiguos.

# Crear objetos a través de plantillas

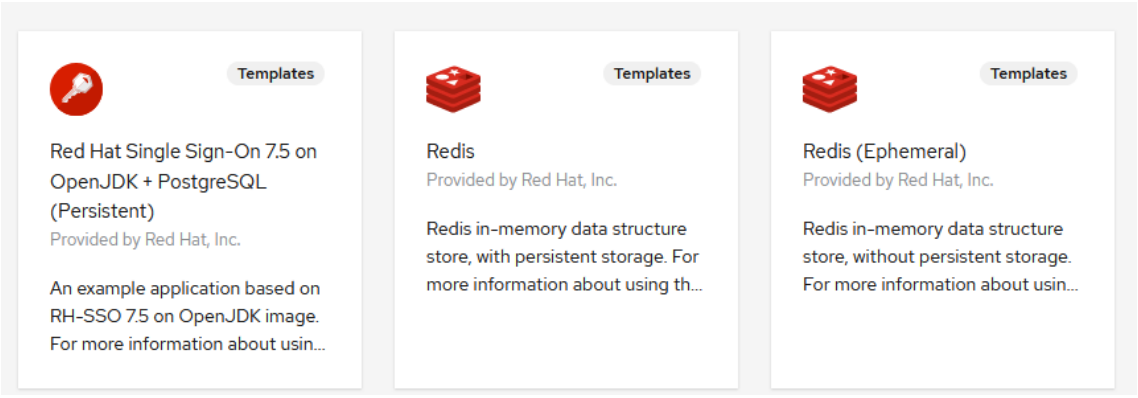
Para ellos desde developer



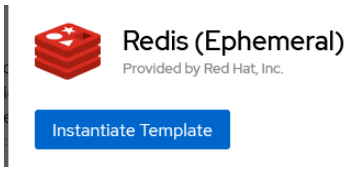

Luego vamos a plantillas



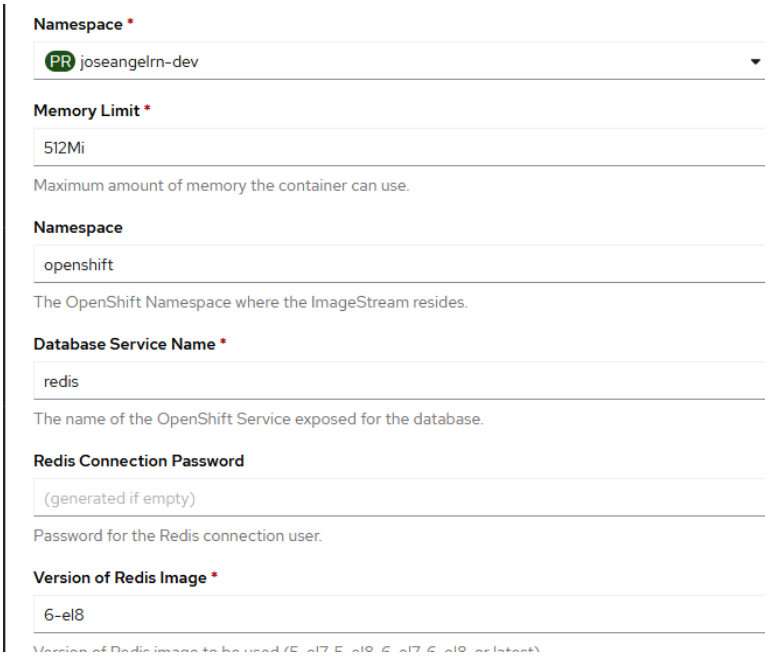

Elegimos la Template




Vamos a crear instancia




Aquí podemos ver los objetos




## PV (Persistent Volume) y PVC (Persistent Volume)

Desde el administrador