Workshop Introducción a Openshift Container Platform

Laboratorio Openshift

Los siguientes ejercicios se van a ejecutar en cada una de las PC de los asistentes, conectándose directamente a un ambiente que se les proveerá para el curso.

Para el laboratorio se va a utilizar los siguientes recursos:

- Acceso a la interfaz gráfica de Openshift
- Acceso al CLI para la ejecución de comandos desde la terminal de Openshift

Nota: El presentador les dará acceso a las URL necesarias.

Prerequisitos

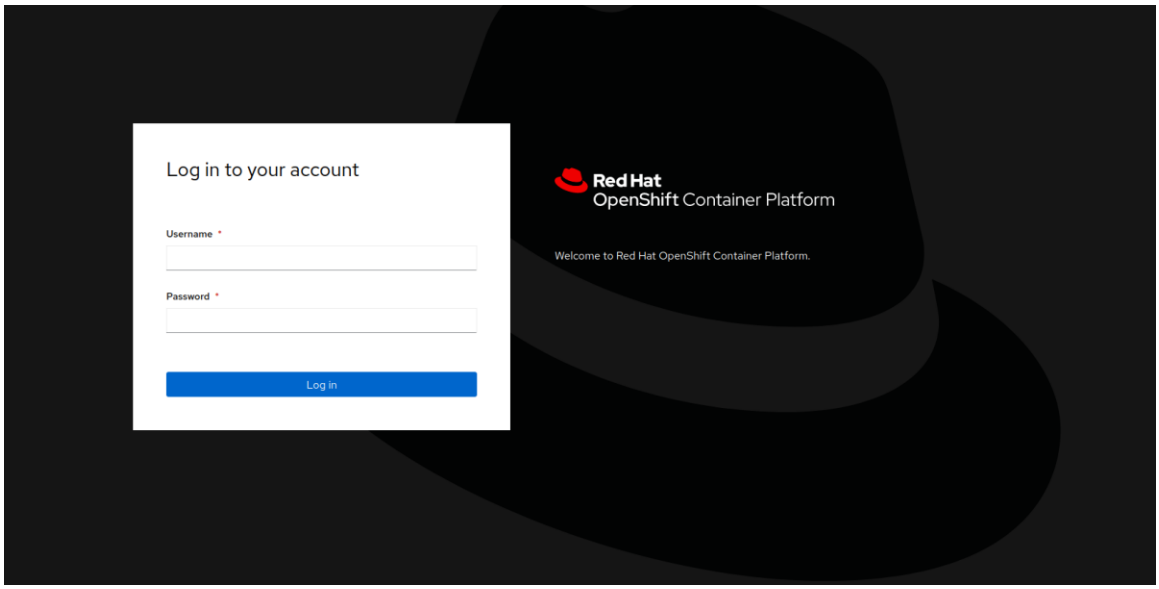
a) Se le brindará una URL para accesar al ambiente del curso.

Importante: Dentro de los ejercicios de los laboratorios deberá sustituir los valores de color rojo por el número de usuario asignado previamente por el presentador.

Acceso a la interfaz Openshift

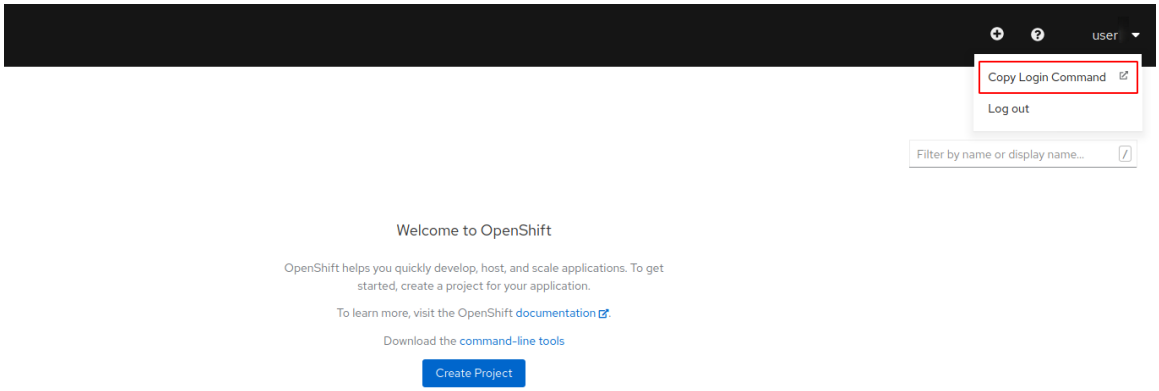
Una vez con acceso a las URL, loguearse en la consola web de Openshift Container Platform, para ello ingresaremos los siguientes datos.

- Usuario: userX
- Password**:** openshift

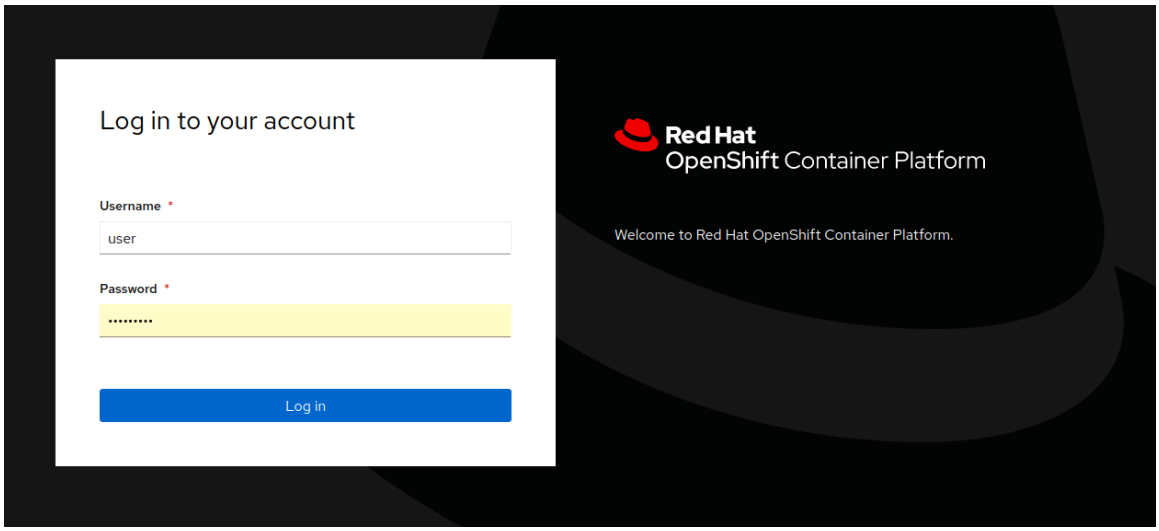


b) Al ingresar a la consola, nos vamos a la esquina derecha.

Y copiamos el login command. Para ellos damos clic en **"Copy Login Command"** para poder tener acceso a nuestros laboratorios desde una terminal.

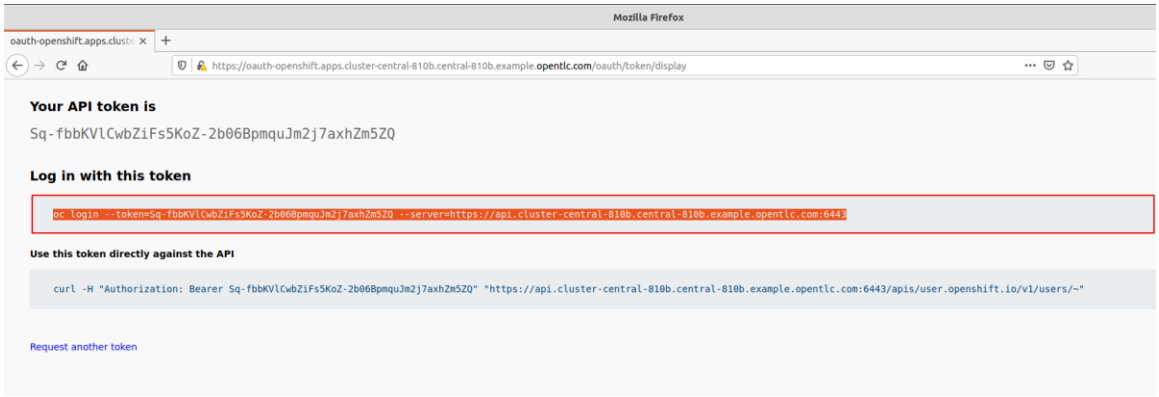


Indicamos nuevamente nuestro usuario y contraseña y luego Login



En la siguiente pantalla damos clic en "**Display Token**"

Y copiamos el valor contenido en la leyenda **"Log in with this Token"** y ese contenido lo pegamos en la consola del CLI que veremos a continuación.

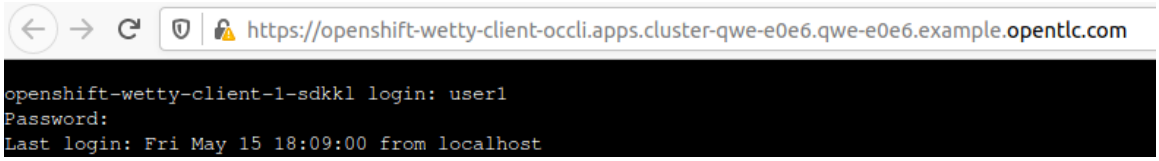


Acceso a la terminal Openshift CLI

Se ingresa a la URL de la terminal que se le brindó al inicio del laboratorio prevista con el usuario de la consola del CLI que tiene el siguiente formato:

- Usuario: userx
- Password: Passwordx

Donde el valor "x" es el número que se le asignó para el usuario del curso.



Y luego se ingresa el token obtenido desde la interfaz de Openshift en el paso anterior, para loguearnos dentro de nuestro ambiente y poder ejecutar el laboratorio.

Ya luego de esto tenemos nuestro ambiente listo para iniciar con los laboratorios

1. Creación de proyecto desde línea de comandos OC.

1.1 Una vez logueados en la consola de OC, creamos el nuevo proyecto con el nombre “user**x**-lab1”, de la siguiente forma, debería dar el siguiente output.

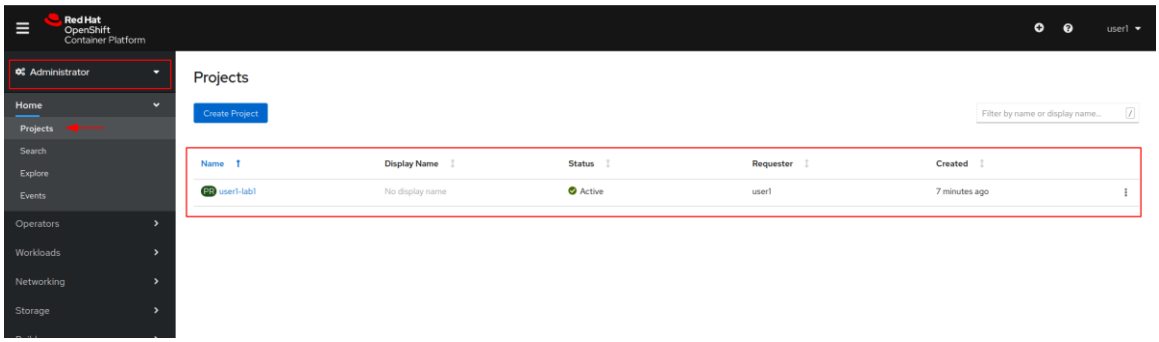
```
$ oc new-project userx-lab1
Now using project "userx-lab1" on server "https://api.cluster-central-810b.central-810b.example.opentlc.com:6443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
```

1.2 Verificamos el nuevo projecto “**oc new-project userx-lab1**” creado desde la linea de commandos oc.

```
$ oc get projects
NAME        DISPLAY NAME   STATUS
userx-lab1                 Active
```

1.3 También lo podemos verificar en la consola web.

En la vista Administrator > Home > Projects



2. Creación de una aplicación desde línea de comandos OC.

2.1 Creamos un nuevo proyecto

```
$ oc new-project userx-lab2
Now using project " oc new-project userx-lab2" on server
"https://master.na311.openshift.opentlc.com:443".
You can add applications to this project with the 'new-app' command. For example, try:

oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
```

2.2 Crear una nueva aplicación utilizando la imagen latest de wildfly disponible en Docker Hub, para ellos vamos a ejecutar el siguiente comando

```
$ oc new-app docker.io/jboss/wildfly:latest

--> Found Docker image 254d174 (2 weeks old) from docker.io for
"docker.io/jboss/wildfly:latest"

    * An image stream tag will be created as "wildfly:latest" that will track this image
    * This image will be deployed in deployment config "wildfly"
    * Port 8080/tcp will be load balanced by service "wildfly"
      * Other containers can access this service through the hostname "wildfly"

--> Creating resources ...
    imagestream.image.openshift.io "wildfly" created
    deploymentconfig.apps.openshift.io "wildfly" created
    service "wildfly" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/wildfly'
    Run 'oc status' to view your app.
```

2.3 Verificamos que se hayan creado los objetos de Openshift, entre ellos: **imagestream, deploymentconfig**, **pod**, **replicationcontroller** y **service**.

El **replication controller,** debe estar en el mismo valor: **Desired, Current** y **Ready**, esto indica que ya la aplicación inició de forma correcta en la cantidad de réplicas deseadas.

```
$ oc get all
NAME          READY   STATUS    RESTARTS   AGE
pod/wildfly-1-lfmv9   1/1     Running   0          1m

NAME                    DESIRED   CURRENT   READY   AGE
replicationcontroller/wildfly-1   1         1         1       1m

NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/wildfly   ClusterIP   172.30.74.215   <none>        8080/TCP   1m

NAME                                 REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfig.apps.openshift.io/wildfly   1          1         1
config,image(wildfly:latest)

NAME                          DOCKER REPO                                TAGS      UPDATED
imagestream.image.openshift.io/wildfly   docker-registry.default.svc:5000/jjl-new-
apps/wildfly   latest    About a minute ago
```

2.4 Los servicios se utilizan para la comunicación interna de OpenShift. De esta manera, otra aplicación no necesita saber la dirección IP real del pod o cuántos pods se están ejecutando para una aplicación determinada. Pero dado que los servicios no son accesibles fuera del clúster de OpenShift, debe crear una ruta para exponer el servicio al mundo exterior:

```
$ oc expose svc wildfly
route.route.openshift.io/wildfly  exposed
```

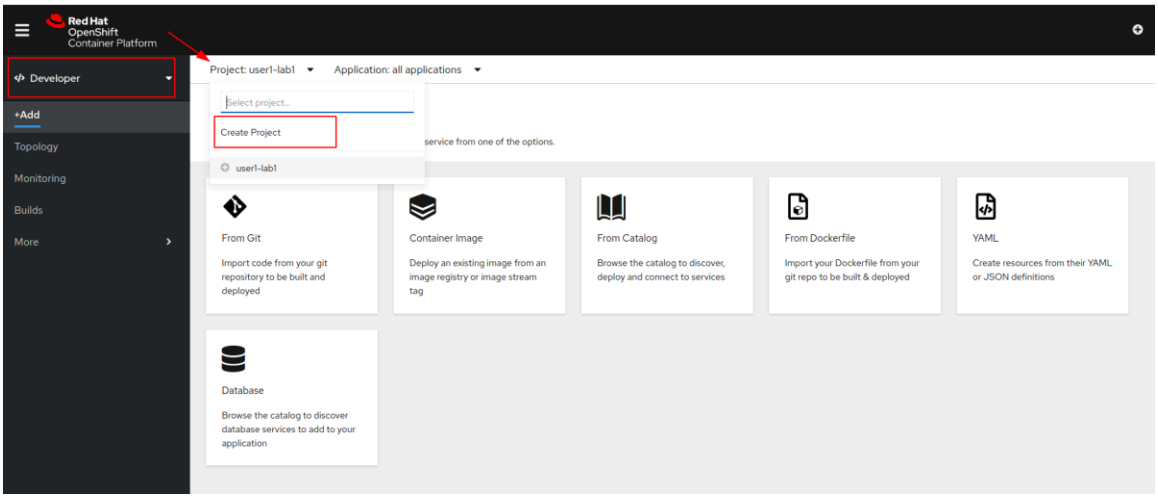
2.5 Verificar la ruta creada:

```
$ oc get route
NAME      HOST/PORT                                                PATH      SERVICES   PORT
TERMINATION   WILDCARD
wildfly   wildfly-user1-lab1.apps.cluster-qwe-e0e6.qwe-e0e6.example.opentlc.com
wildfly    8080-tcp              None
```

2.6 Copiamos la ruta que se creó con el **expose** en un browser, en este caso es:
wildfly-user1-lab1.apps.cluster-qwe-e0e6.qwe-e0e6.example.opentlc.com

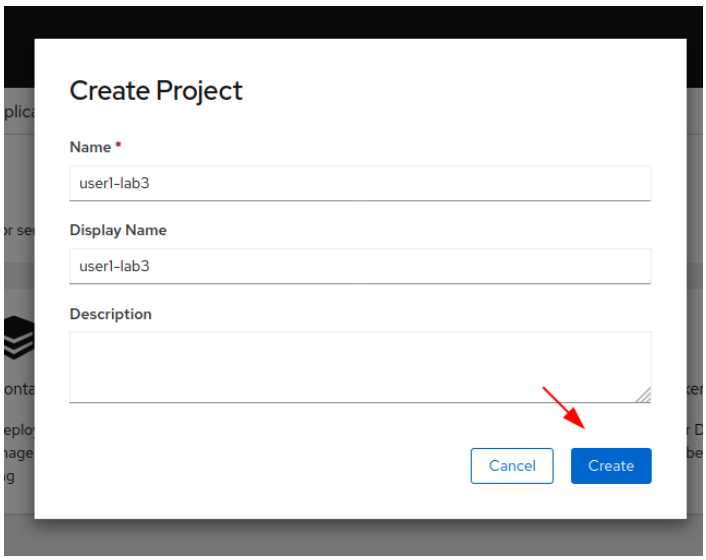
3. Creación de aplicación PHP desde un repositorio GitHub.

3.1 Para ello vamos a la vista **Developer** y agregamos un proyecto desde la lista desplegable llamada **Project** y damos clic en **Create Project** como se muestra la figura adjunta.

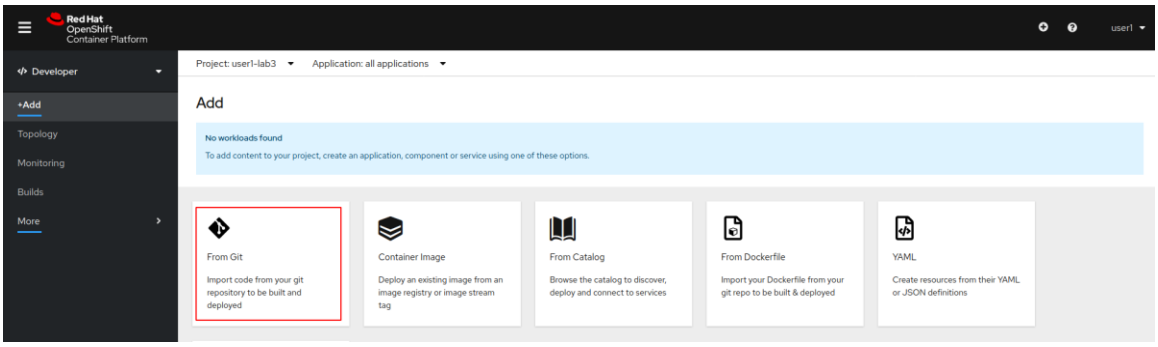


3.2 Le indicamos el nombre user**x**-lab3

- Name: **user**x**-lab3**
- Display Name: **user**x**-lab3**
- Description: **Aplicación PHP**



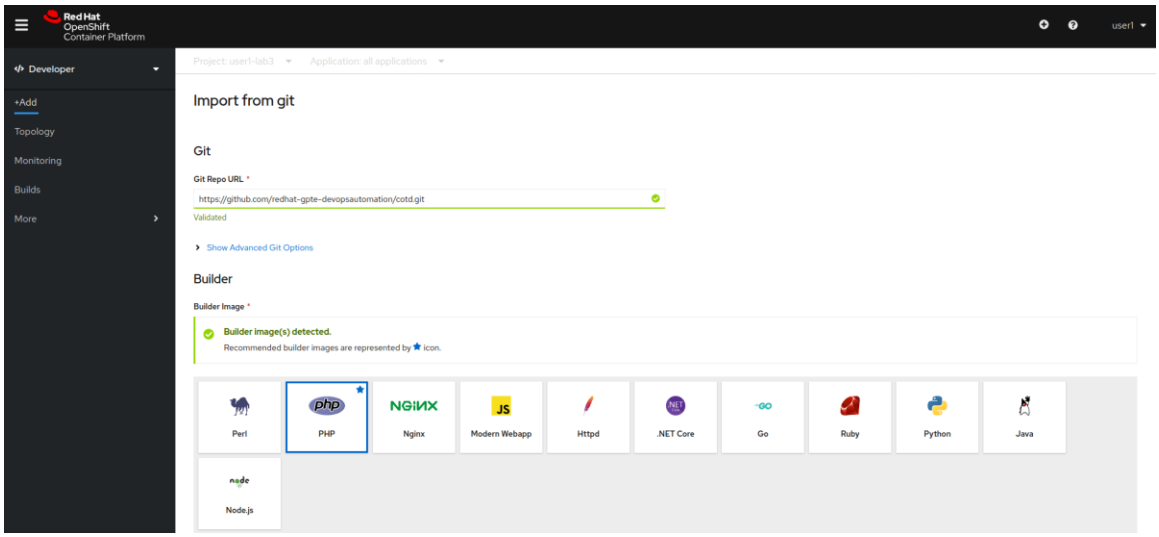
3.3 Procedemos a irnos a la opción **From Git** para crear una aplicación desde un repositorio

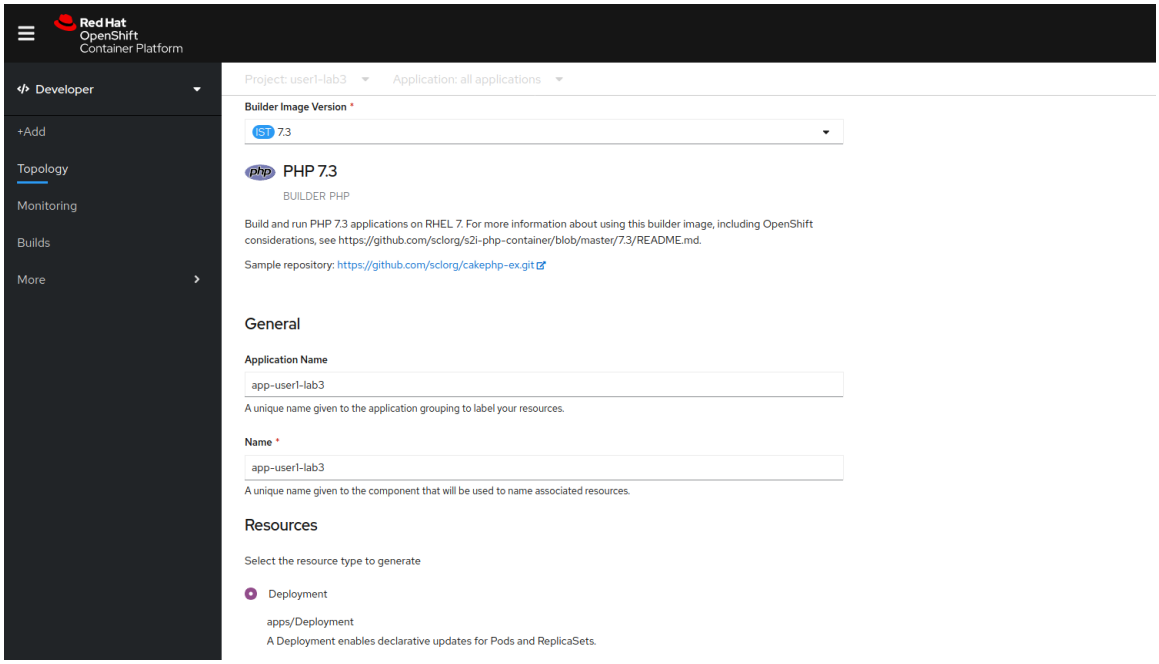


3.4 Procedemos a completar la información con los siguientes datos:

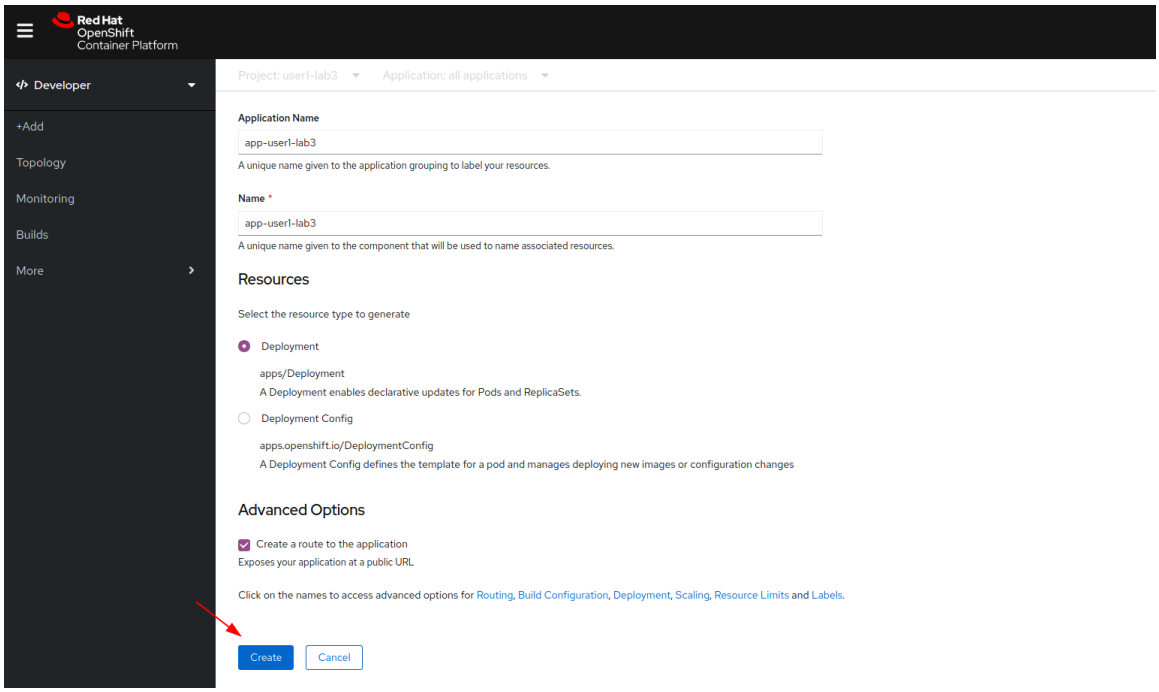
- **Git Repo URL:** https://github.com/redhat-gpte-devopsautomation/cotd.git
- Application Name: app-userx-lab3
- Name: app-userx-lab3

Quedando de la siguiente manera:

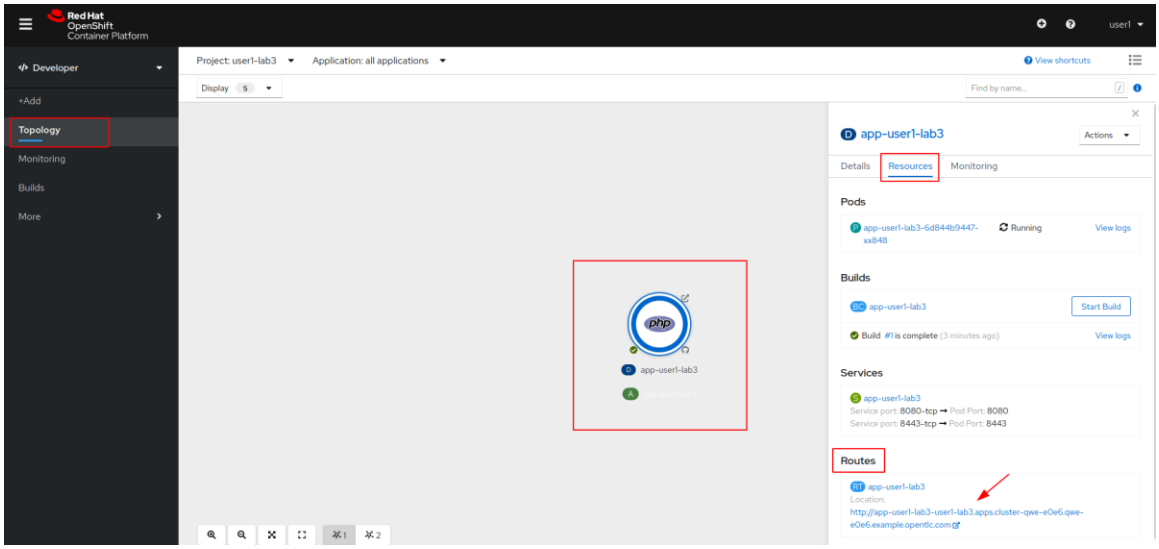




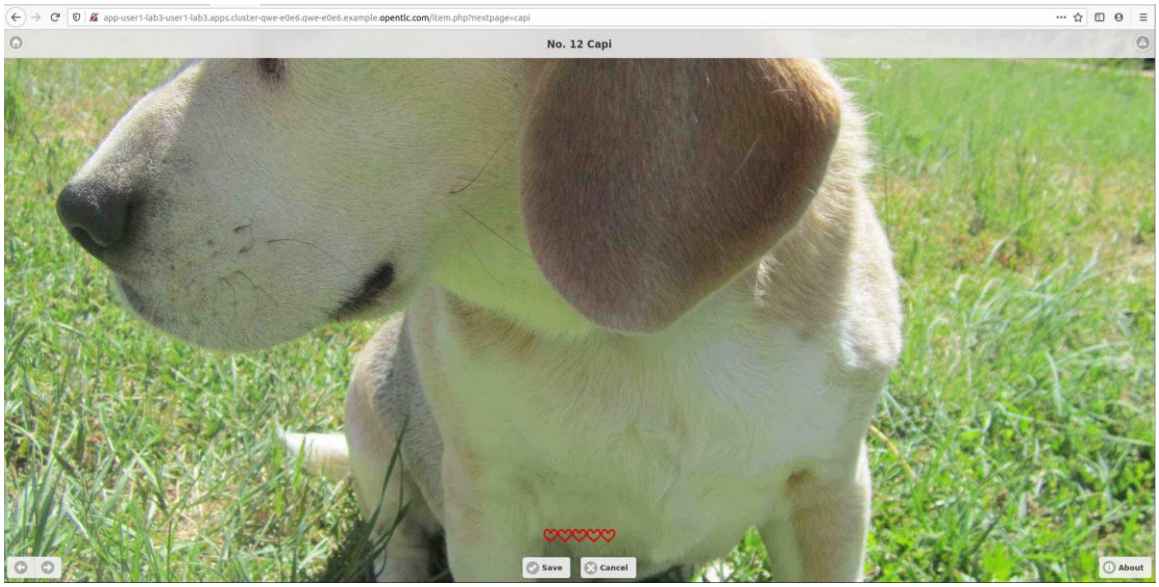
3.5 Procedemos a crear nuestra aplicación, luego de estos pasos tan sencillos



3.6 Desde la vista **Topology** podemos ver nuestro pod y dándole clic vemos todos atributos configurados para nuestra aplicación.
Desde la pestaña **Resources,** podremos ver el link para accesar a nuestra aplicación.



3.7 Luego de dar clic en la ruta nuestra aplicación ya estará expuesta y será gestionada totalmente por Openshift



4. Creación de aplicación PHP (Animales y cuidades) desde la línea de comando CLI.

Explora los detalles de los diversos objetos y observa los eventos asociados. Finalmente, crea una segunda versión de la aplicación y una ruta A / B para la aplicación, y luego observa cómo se enruta el tráfico a las dos versiones de la aplicación.

4.1 Crear un **proyecto** con el nombre "**user**x**-lab4**".

```
$ oc new-project userx-lab4
Now using project "oc new-project user1-lab4" on server
"https://master.na311.openshift.opentlc.com:443".

You can add applications to this project with the 'new-app' command. For example,
try:

   oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
```

4.2 Crear una nueva aplicación desde el repositorio de git: **https://github.com/redhat-gpte-devopsautomation/cotd.git**

```
$ oc new-app https://github.com/redhat-gpte-devopsautomation/cotd.git
--> Found image 6eeec1d (11 months old) in image stream "openshift/php" under tag
"7.1" for "php"

    Apache 2.4 with PHP 7.1
    ----------------------
    PHP 7.1 available as container is a base platform for building and running various PHP
7.1 applications and frameworks. PHP is an HTML-embedded scripting language. PHP
attempts to make it easy for developers to write dynamically generated web pages. PHP
also offers built-in database integration for several commercial and non-commercial
database management systems, so writing a database-enabled webpage with PHP is
fairly simple. The most common use of PHP coding is probably as a replacement for CGI
scripts.

    Tags: builder, php, php71, rh-php71

    * The source repository appears to match: php
    * A source build using source code from https://github.com/redhat-gpte-
devopsautomation/cotd.git will be created
      * The resulting image will be pushed to image stream tag "cotd:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "cotd"
    * Ports 8080/tcp, 8443/tcp will be load balanced by service "cotd"
      * Other containers can access this service through the hostname "cotd"

--> Creating resources ...
    imagestream.image.openshift.io "cotd" created
    buildconfig.build.openshift.io "cotd" created
    deploymentconfig.apps.openshift.io "cotd" created
    service "cotd" created
--> Success
    Build scheduled, use 'oc logs -f bc/cotd' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/cotd'
    Run 'oc status' to view your app.
```

4.3 Creamos la ruta para poder accesar la aplicación desde fuera del clúster de Openshift.

```
$ oc expose svc/cotd
route.route.openshift.io/cotd  exposed
```

4.4 Revisamos los logs del build.

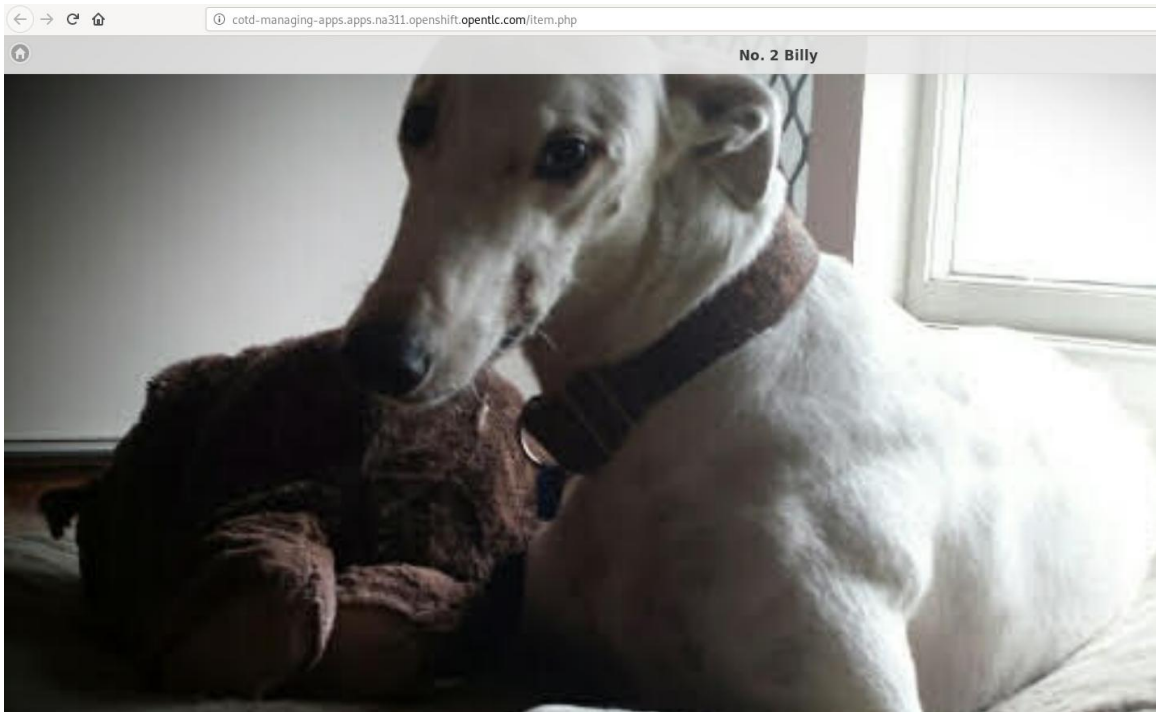
```
$ oc logs -f bc/cotd
Cloning "https://github.com/redhat-gpte-devopsautomation/cotd.git"  ...
        Commit:         b53da24b6cfad8e31f0706f9ef8936761cba97e0 (Merge pull
request #1 from StefanoPicozzi/master)
        Author:         Wolfgang Kulhanek <wkulhanek@users.noreply.github.com>
        Date:   Thu Jan 11 07:38:09 2018 -0500
Using docker-
registry.default.svc:5000/openshift/php@sha256:7a8b79ebc15ebf8e79f6b8624cd028c7
87d7d23d1b36677d7ee1c6e0ef1f3349  as the s2i builder image
---> Installing application source...
=> sourcing 20-copy-config.sh ...
---> 18:30:15     Processing additional arbitrary httpd configuration provided by s2i ...
=> sourcing 00-documentroot.conf  ...
=> sourcing 50-mpm-tuning.conf ...
=> sourcing 40-ssl-certs.sh ...

Pushing image docker-registry.default.svc:5000/managing-apps/cotd:latest ...
Pushed 5/6 layers, 87% complete
Pushed 6/6 layers, 100% complete
Push successful
```

4.5 Verificamos que se hayan creado los objetos de Openshift, entre ellos: **imagestream**, **deploymentconfig**, **pod, replicationcontroller** y **service**.
El **replication controller**, debe estar en el mismo valor: **Desired**, **Current** y **Ready**, esto indica que ya la aplicación inició de forma correcta en la cantidad de réplicas deseadas.

```
$ oc get all
NAME          READY    STATUS     RESTARTS   AGE
pod/cotd-1-build   0/1      Completed   0        1h
pod/cotd-1-zj97x   1/1      Running     0        1h

NAME                  DESIRED   CURRENT   READY    AGE
replicationcontroller/cotd-1   1         1         1        1h

NAME        TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/cotd   ClusterIP   172.30.43.124   <none>      8080/TCP,8443/TCP   1h

NAME                          REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfig.apps.openshift.io/cotd    1          1         1         config,image(cotd:latest)

NAME                        TYPE     FROM     LATEST
buildconfig.build.openshift.io/cotd    Source   Git      1

NAME                    TYPE     FROM       STATUS     STARTED          DURATION
build.build.openshift.io/cotd-1    Source   Git@b53da24   Complete   About an hour ago
23s

NAME                        DOCKER REPO                                TAGS     UPDATED
imagestream.image.openshift.io/cotd   docker-registry.default.svc:5000/managing-
apps/cotd    latest    About an hour ago

NAME                  HOST/PORT                                PATH     SERVICES   PORT
TERMINATION   WILDCARD
route.route.openshift.io/cotd    cotd-user1-lab4.apps.cluster-qwe-e0e6.qwe-
e0e6.example.opentlc.com          cotd       8080-tcp             None
```

Lo marcado en rojo es la ruta que debemos copiar en nuestro navegador para acceder a la aplicación.

4.6 Verificamos el acceso a la aplicación PHP, es una aplicación que en cada refrescamiento muestra imágenes de mascotas.



4.7 OpenShift hace que sea extremadamente fácil escalar una aplicación simplemente escalando el controlador de replicación o la configuración de implementación. Solo escalaría el controlador de replicación directamente en casos excepcionales cuando no está controlado por una configuración de implementación.

Cuando cambia el número de réplicas de pod solicitadas, OpenShift hace girar nuevos pods o los elimina. Puede usar el comando oc scale para cambiar el número de pods solicitados para una aplicación.
Verificamos la cantidad de pods que existen antes de crear más réplicas y para verificar la cantidad de réplicas existentes.

```
$ oc get pods
NAME          READY    STATUS      RESTARTS   AGE
cotd-1-build  0/1      Completed   0          1h
cotd-1-zj97x  1/1      Running     0          1h
```

4.8 Hacemos la escalación directamente en el **deployment config** "**cotd**" a **3 réplicas**.

```
$ oc scale dc cotd --replicas=3
deploymentconfig.apps.openshift.io/cotd  scaled
```

4.9 Verificamos la cantidad de réplicas existentes después de la escalación.

```
$ oc get pods
NAME        READY    STATUS     RESTARTS   AGE
cotd-1-build   0/1      Completed   0         1h
cotd-1-jkttb   1/1      Running     0         35s
cotd-1-sqtw7   1/1      Running     0         35s
cotd-1-zj97x   1/1      Running     0         1h
```

4.10 Espere ver seis **endpoints** para la ruta, 3 para el puerto 8443 y 3 para el puerto 8080. OpenShift enruta el tráfico de manera transparente a cada uno de estos pods.

```
$ oc describe route cotd
Name:                   cotd
Namespace:              managing-apps
Created:                About an hour ago
Labels:                 app=cotd
Annotations:            openshift.io/host.generated=true
Requested Host:                 cotd-user1-lab4.apps.cluster-qwe-e0e6.qwe-
e0e6.example.opentlc.com
                          exposed on router router about an hour ago
Path:                   <none>
TLS Termination:        <none>
Insecure Policy:        <none>
Endpoint Port:          8080-tcp

Service:        cotd
Weight:                 100 (100%)
Endpoints:      10.1.11.230:8443, 10.1.12.92:8443, 10.1.8.157:8443 + 3 more...
```

4.11 Ahora escalamos de nuevo a un pod.

```
$ oc scale dc cotd --replicas=1
deploymentconfig.apps.openshift.io/cotd  scaled
```

4.12 Verificamos de nuevo los pods y ahora solo hay 1.

```
$ oc get pods
NAME        READY    STATUS     RESTARTS   AGE
cotd-1-build   0/1      Completed   0         1h
cotd-1-zj97x   1/1      Running     0         1h
```

OpenShift incluye la capacidad de establecer **dos o más back-end** para cualquier ruta dada. Esto se puede usar para las **pruebas A / B** donde se implementan dos versiones de la aplicación al mismo tiempo y los clientes se envían aleatoriamente a una de las dos versiones. El enrutamiento A / B se usa con mayor frecuencia para las pruebas de la interfaz de usuario, por ejemplo, para determinar qué versión de un sitio web genera más conversiones de ventas.

4.13 Realice una segunda versión de la aplicación utilizando **cities** como **selector**. Esto le indicará a la aplicación que muestre imágenes de ciudades en lugar del conjunto predeterminado de imágenes de animales.

```
$ oc new-app --name='cotd2' -l name='cotd2' https://github.com/redhat-gpte-devopsautomation/cotd.git -e SELECTOR=cities

--> Found image 6eeec1d (11 months old) in image stream "openshift/php" under tag "7.1" for "php"

    Apache 2.4 with PHP 7.1
    -----------------------
    PHP 7.1 available as container is a base platform for building and running various PHP 7.1 applications and frameworks. PHP is an HTML-embedded scripting language. PHP attempts to make it easy for developers to write dynamically generated web pages. PHP also offers built-in database integration for several commercial and non-commercial database management systems, so writing a database-enabled webpage with PHP is fairly simple. The most common use of PHP coding is probably as a replacement for CGI scripts.

    Tags: builder, php, php71, rh-php71

    * The source repository appears to match: php
    * A source build using source code from https://github.com/redhat-gpte-devopsautomation/cotd.git will be created
      * The resulting image will be pushed to image stream tag "cotd2:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "cotd2"
    * Ports 8080/tcp, 8443/tcp will be load balanced by service "cotd2"
      * Other containers can access this service through the hostname "cotd2"
```

Ahora ha creado una segunda aplicación**, cotd2**. Debido a que no especificó un selector para la primera aplicación, esa aplicación utilizó las mascotas predeterminadas. Ahora puede determinar mirando la imagen de qué versión de la aplicación es.

4.14 Esta vez no expone el servicio **cotd2** como una **ruta**, pero lo agrega a su ruta anterior como otro **back-end** de ruta.
Con esto **50%** del tráfico se irá a la aplicación **cotd y 50%** del tráfico se irá al **cotd2.**

```
$ oc set route-backends cotd cotd=50 cotd2=50
route.route.openshift.io/cotd  backends updated
```

4.15 Verificamos que el cambio en la ruta se ha realizado de forma correcta.

```
$ oc describe route cotd
Name:                   cotd
Namespace:              managing-apps
Created:                2 hours ago
Labels:                 app=cotd
Annotations:            openshift.io/host.generated=true
Requested Host:                 cotd-user1-lab4.apps.cluster-qwe-e0e6.qwe-
e0e6.example.opentlc.com
                          exposed on router router 2 hours ago
Path:                   <none>
TLS Termination:        <none>
Insecure Policy:        <none>
Endpoint Port:          8080-tcp

Service:        cotd
Weight:                 50 (50%)
Endpoints:      10.1.12.92:8443, 10.1.12.92:8080

Service:        cotd2
Weight:                 50 (50%)
Endpoints:      10.1.8.171:8443, 10.1.8.171:8080
```

4.16 Opcional use **curl** para conectarse a la aplicación cada segundo usando la ruta:

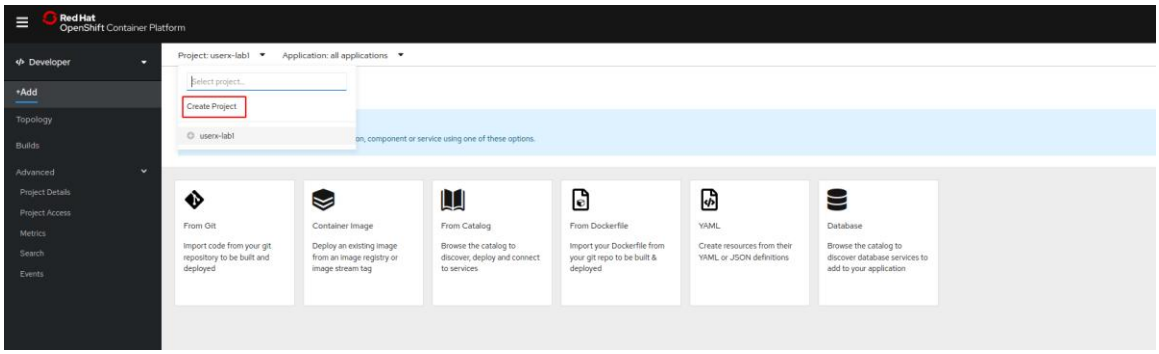
Debe usar **curl** desde la línea de comandos porque todos los navegadores modernos usan cookies para identificar su sesión actual. Y cada vez que actualiza el navegador, se lo envía al mismo servicio al que se lo envió antes, lo cual tiene sentido, porque en el mundo real cuando realiza **pruebas A / B**, desea que una sesión de navegador determinada tenga afinidad con el servicio al que fue enrutado cuando se conectó por primera vez.

```
$ while true; do curl -s http://$(oc get route cotd --template='{{ .spec.host
}}')/item.php | grep "data/images" | awk '{print $5}'; sleep 1; done
data/images/pets/billie.jpg
data/images/cities/adelaide.jpg
data/images/pets/taffy.jpg
data/images/cities/canberra.jpg
data/images/pets/milo.jpg
data/images/cities/perth.jpg
data/images/pets/billy_2.jpg
data/images/pets/deedee.jpg
data/images/cities/wellington.jpg
data/images/pets/neo.jpg
data/images/cities/adelaide.jpg
```

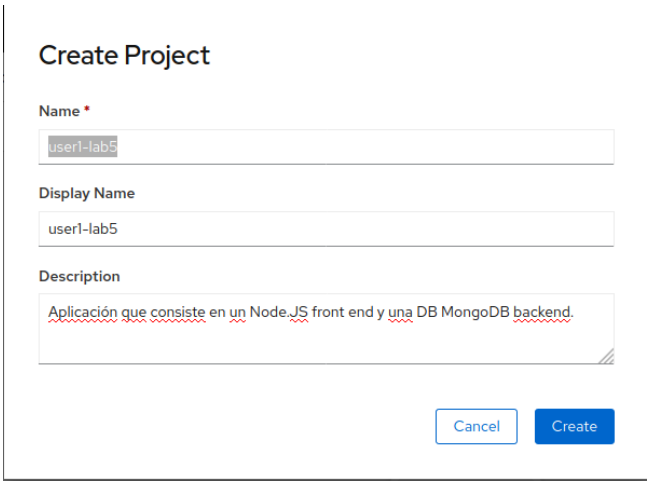
Podemos ver como el **50%** del **tráfico** va **hacia cities** y el **50% hacia pets**.

5. Desplegar aplicación Node.JS con conexión a una base de datos MongoDB, desde Openshift.

5.1 Creamos un proyecto. Para ello vamos a la vista **Developer** y agregamos un proyecto desde la lista desplegable llamada **Project** y damos clic en **Create Project** como se muestra la figura adjunta.

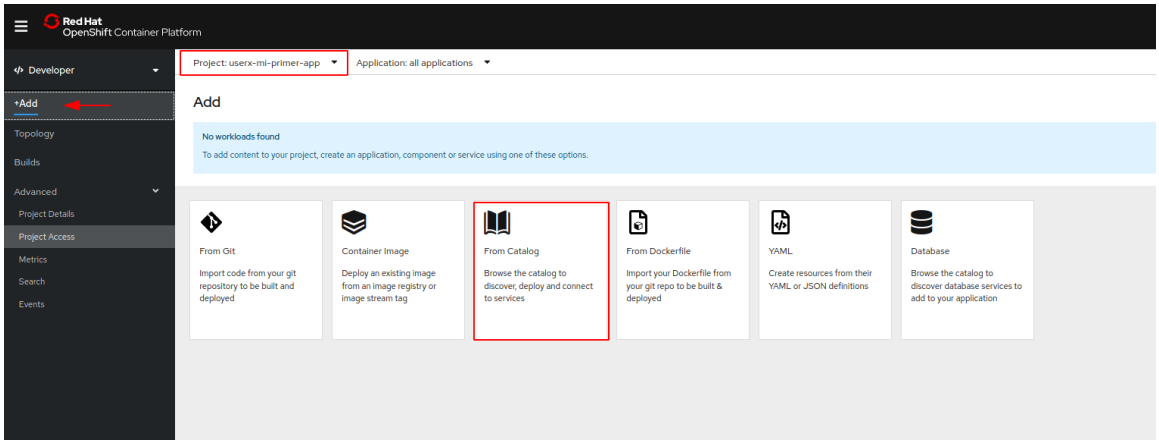


- Name: **userx-lab5**
- Display Name: **userx-lab5**
- Description: **Aplicación que consiste en un Node.JS front end y una DB MongoDB backend.**

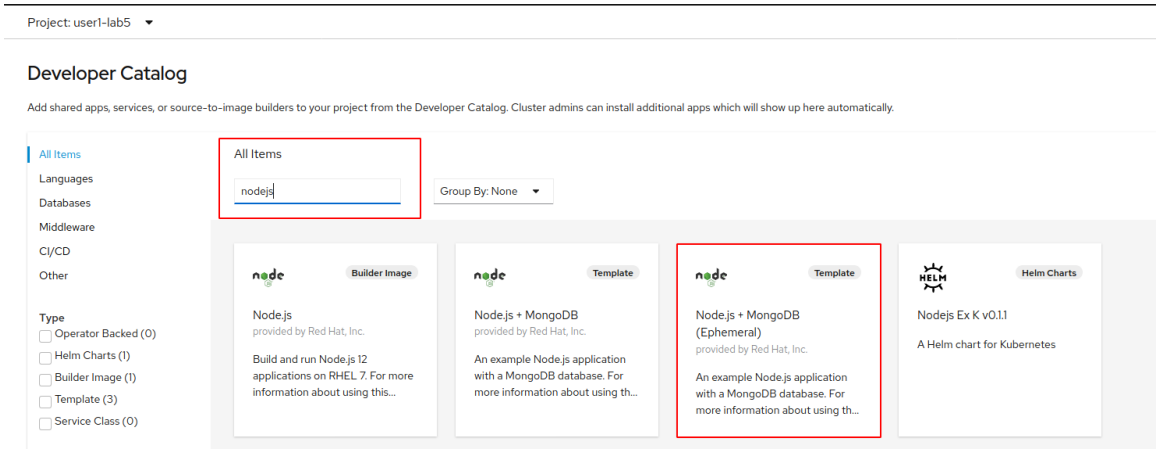


Y damos clic en el botón "**Create**"

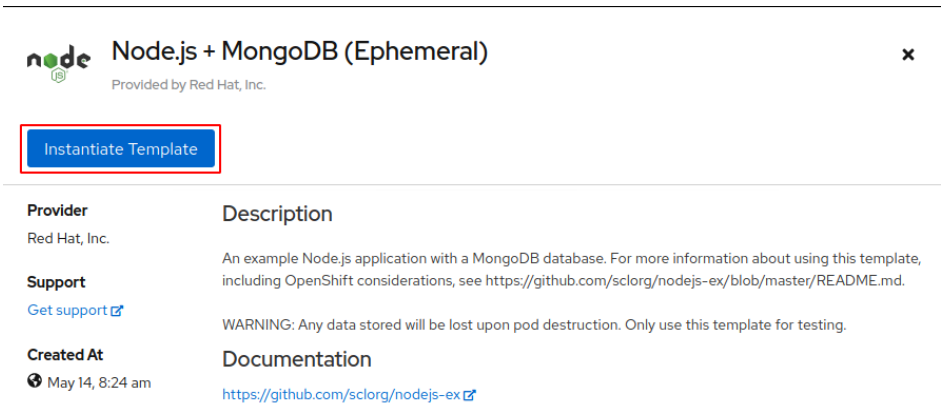
5.2 Seleccionamos la opción From Catalog



5.3 Seleccionamos la opción nodejs+mongoDB para ello filtramos por la palabra "nodejs"



Y le damos al botón "Instantiate Template"

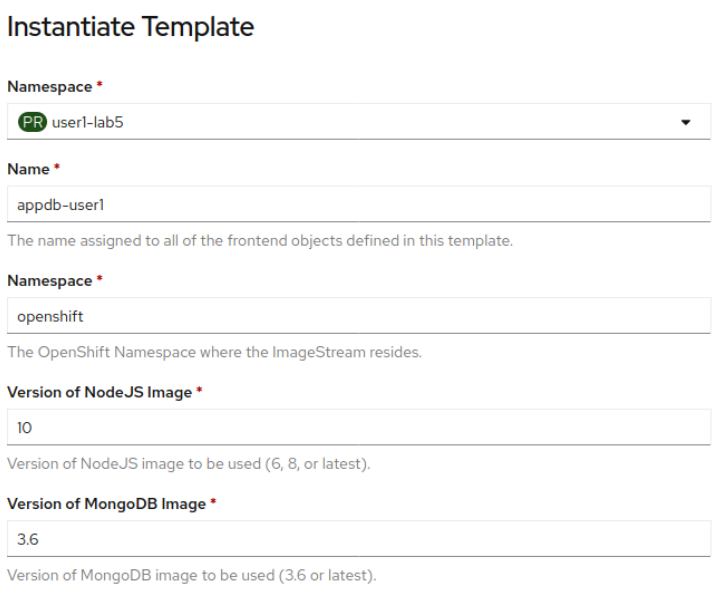


5.4 Para efectos del laboratorio usarmos los repositorios de aplicaciones y base de datos provistos por Red Hat para facilitar su configuración.

Cambiamos los siguientes valores:

- Name: appdb-userx
- Git Repository URL: https://github.com/sclorg/nodejs-ex.git

Quedando la configuración de la siguiente manera:



Memory Limit *

512Mi

Maximum amount of memory the Node.js container can use.

Memory Limit (MongoDB) *

512Mi

Maximum amount of memory the MongoDB container can use.

Volume Capacity *

1Gi

Volume space available for data, e.g. 512Mi, 2Gi

Git Repository URL *

https://github.com/sclorg/nodejs-ex.git

The URL of the repository with your application source code.

Git Reference

Set this to a branch name, tag or other ref of your repository if you are not using the default branch.

Context Directory

Set this to the relative path to your project if it is not in the root of your repository.

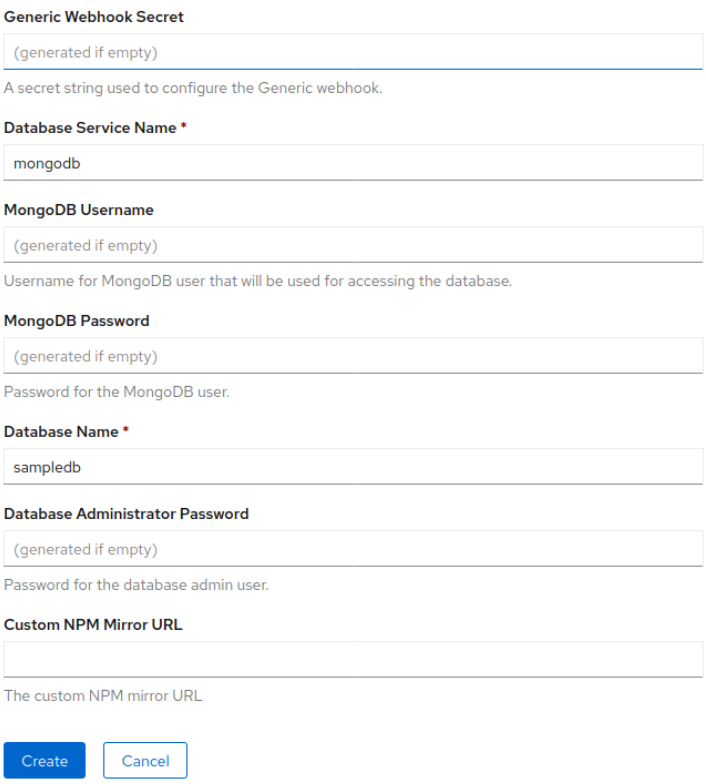
Application Hostname

The exposed hostname that will route to the Node.js service, if left blank a value will be defaulted.

GitHub Webhook Secret

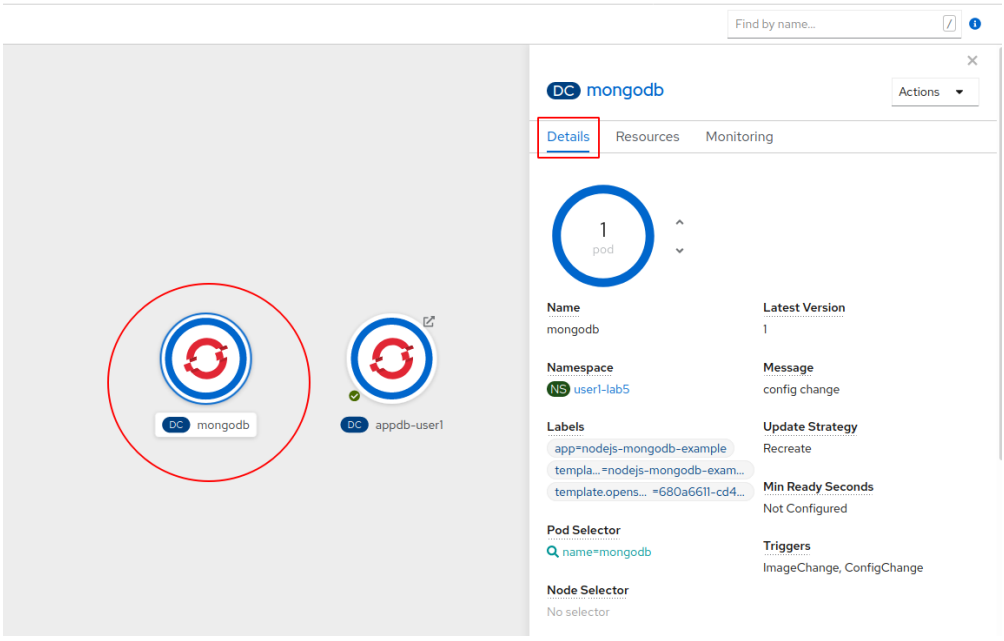
(generated if empty)

Github trigger secret. A difficult to guess string encoded as part of the webhook URL. Not encrypted.

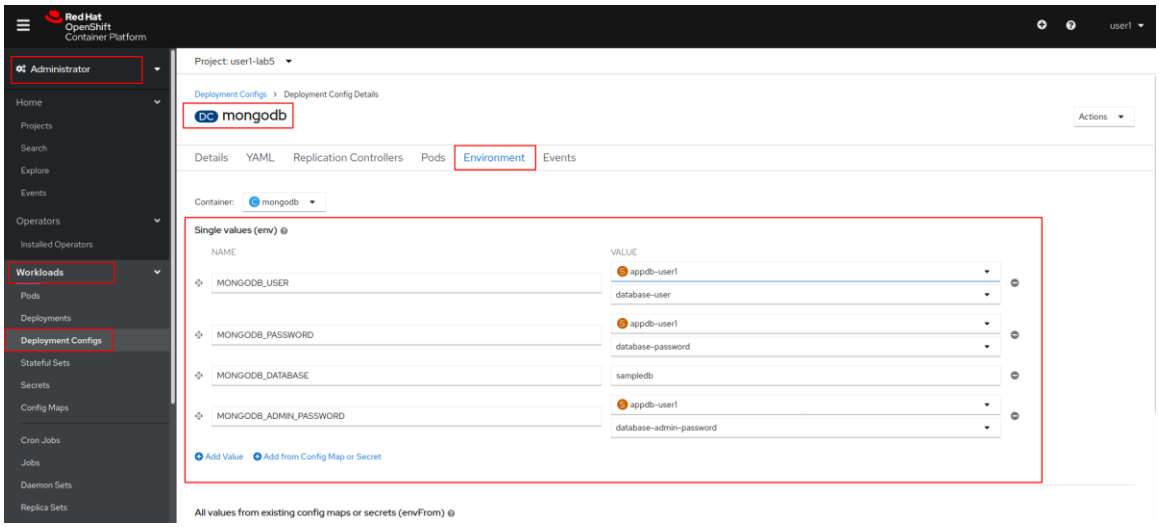


Una vez creado nuestra configuración para nuestro nodejs y mongodb podremos ver desde la vista Topology, ambos componentes y sus recursos.

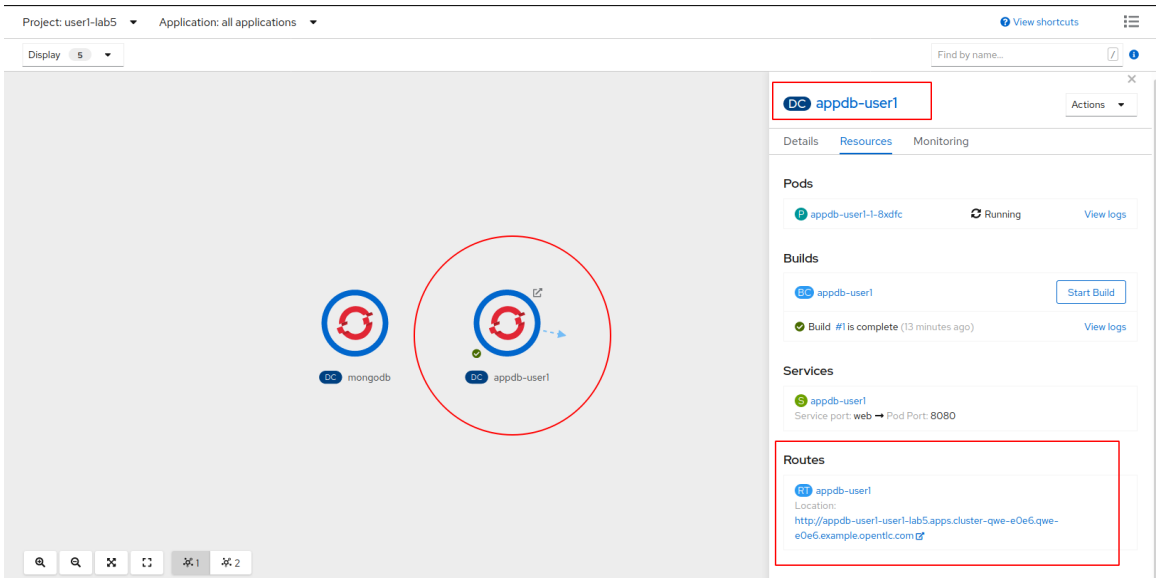
En el caso de MongoDB, podremos ver su pod, su deploy config y los valores de configuración. Para ello seleccionamos el Deploy config llamado “mongodb”



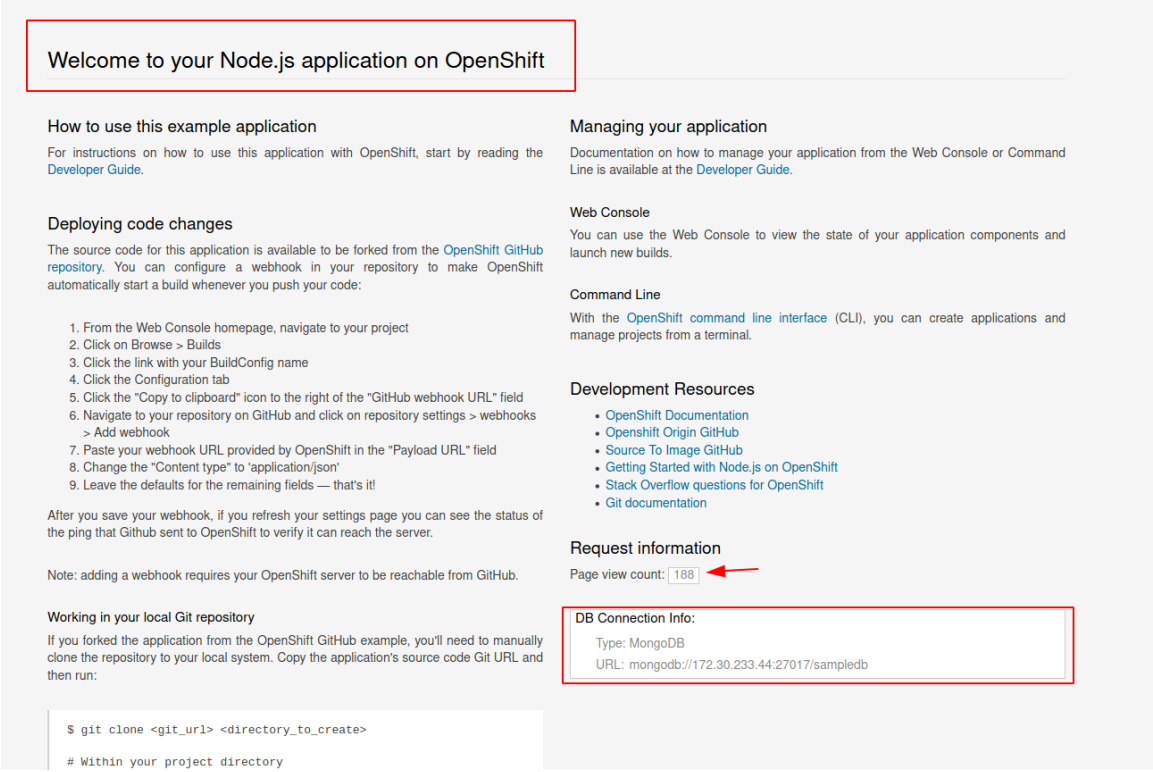
Para validar los detalles y valores como usuario y password de la base de datos podemos dirigirnos a la vista **Administrator > Workloads > Deployments Configs > Seleccionar el deploy config llamado mongodb y nos dirigimos a la pestaña "Environment"**



Ahora en el caso de nuestra aplicación nodejs la cual está configurada por medio del repositorio https://github.com/sclorg/nodejs-ex para ver su configuración, pod, rutas nos dirigimos a la vista **Developer > Topology y seleccionamos del deployment config appdb-user**x



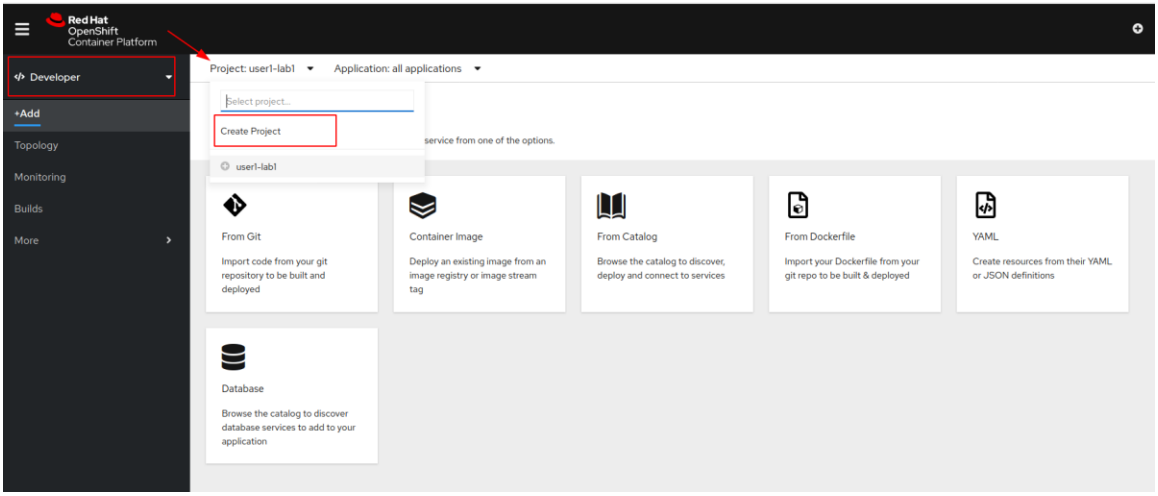
Ver el apartado rutas y damos clic en la ruta expuesta por la aplicación



Para verificar que la aplicación se unió a la base de datos, ver el label "**DB Connection Info**", esto con detalles sobre la conexión a la base de datos. También se puede ver que cada vez que se refresca la página, el "**Page View Count**" se incrementa.

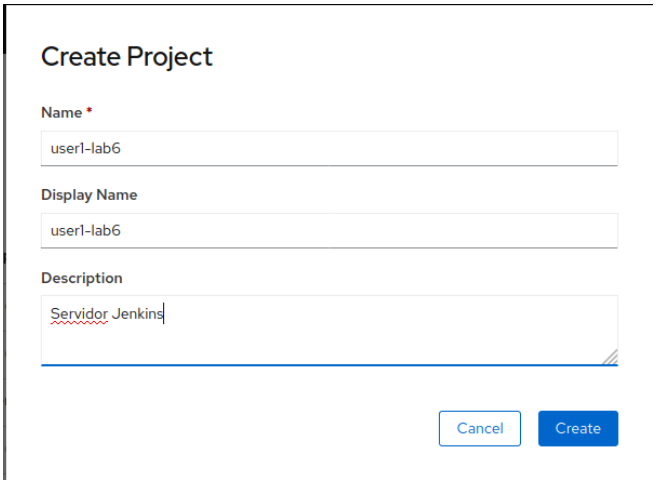
6. Laboratorio Jenkins: Creando un entorno de CI / CD utilizando Jenkins con una aplicación Java EE

6.1 Se debe crear un nuevo proyecto. Para ello vamos a la vista **Developer** y agregamos un proyecto desde la lista desplegable llamada **Project** y damos clic en **Create Project** como se muestra la figura adjunta.

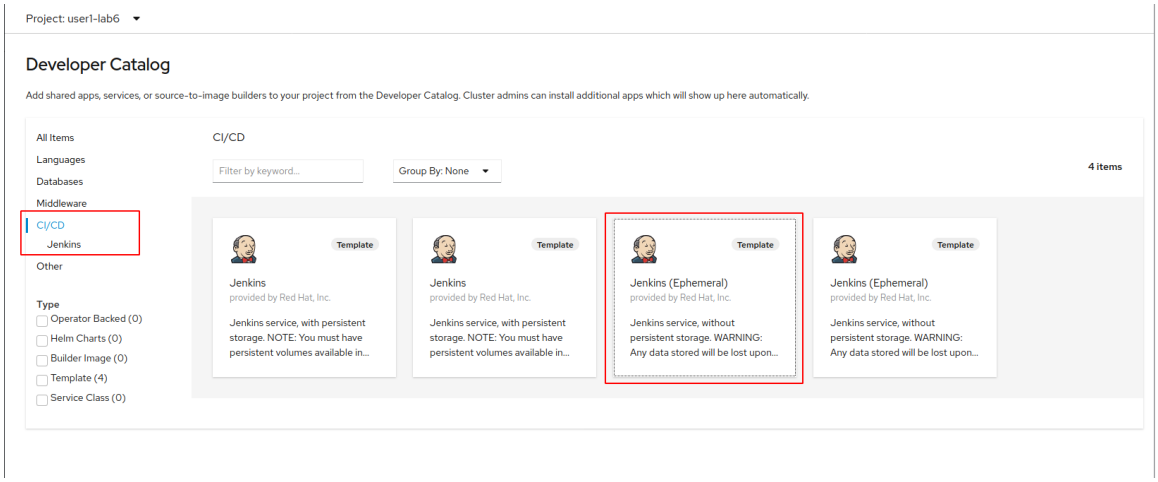


6.2 Le indicamos el nombre userx-lab6

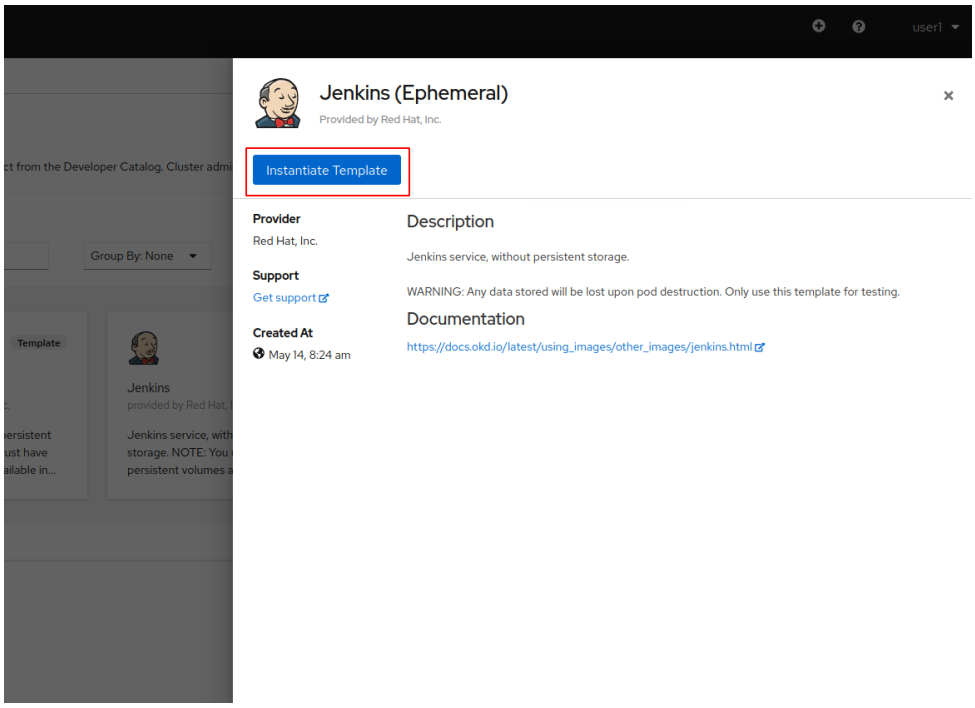
- Name: **userx-lab6**
- Display Name: **userx-lab6**
- Description: **Servidor Jenkins**



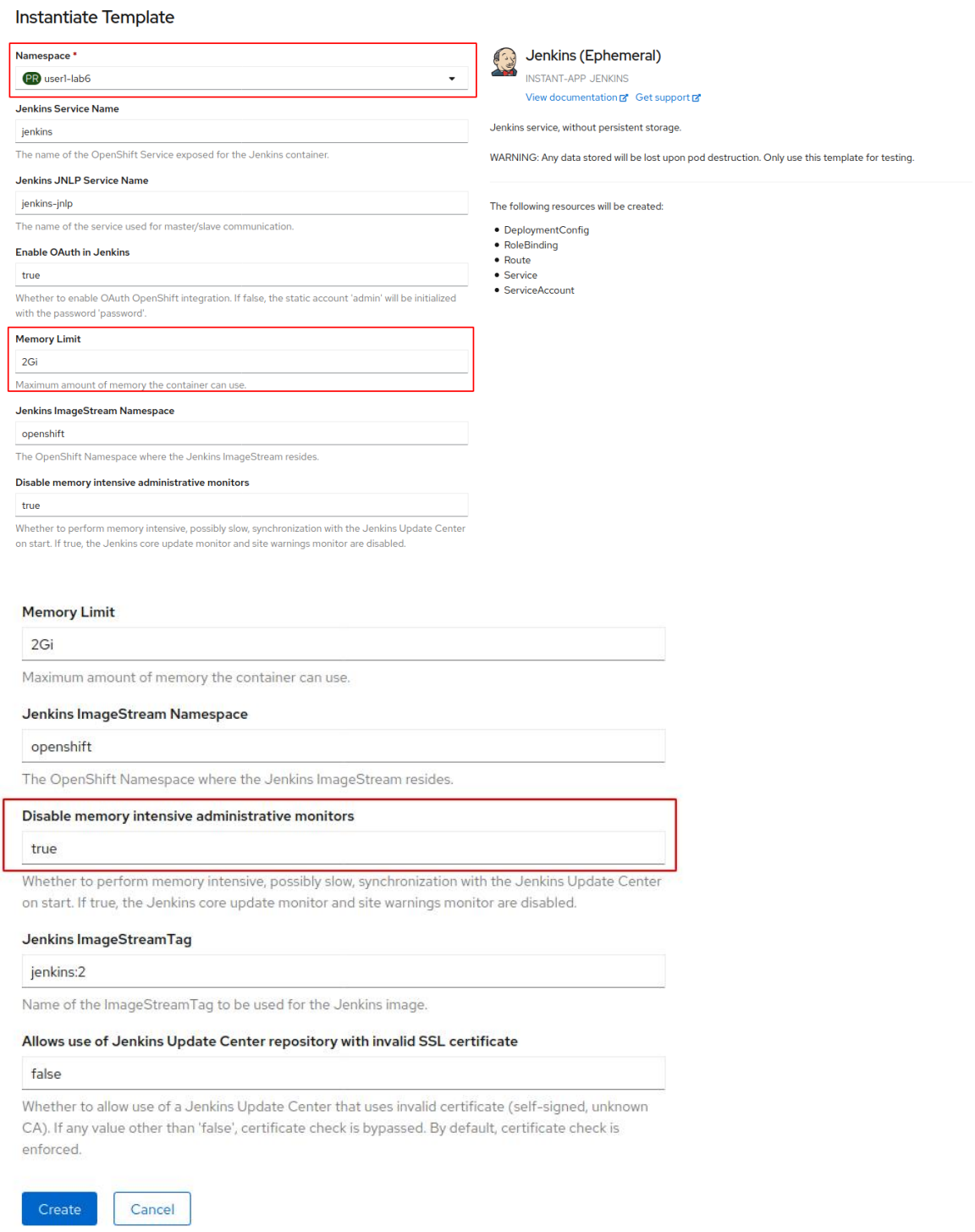
6.3 Ir al catálogo, click en CI/CD, y click en Jenkins (Ephemeral).



6.4 Lo selccionamos y damos click en “Instantiate Template”.

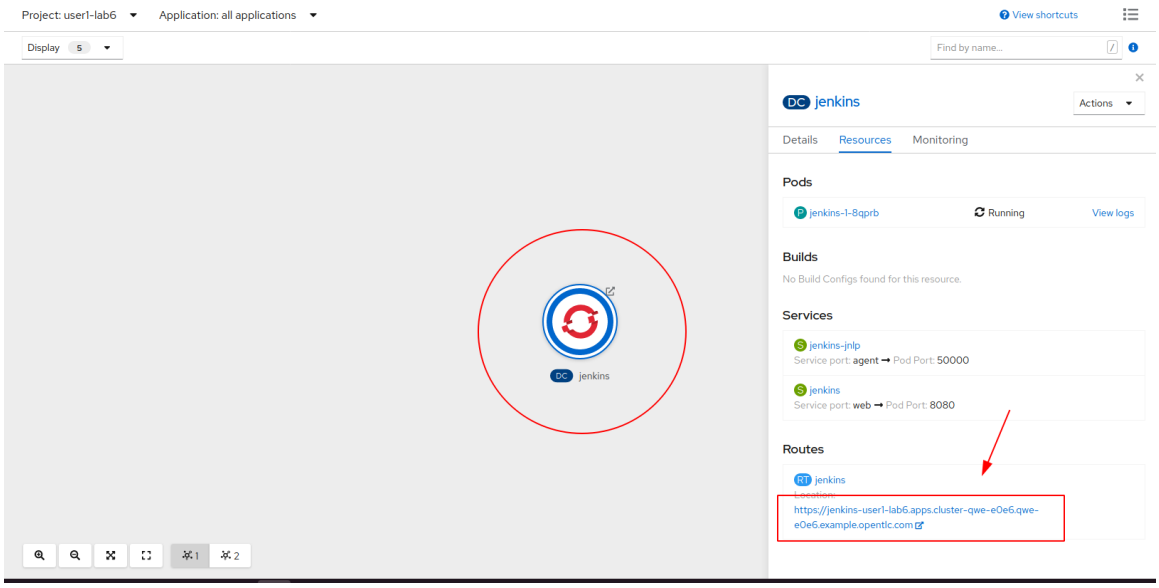


6.5 Se configura la aplicación de Jenkins con los siguientes valores.

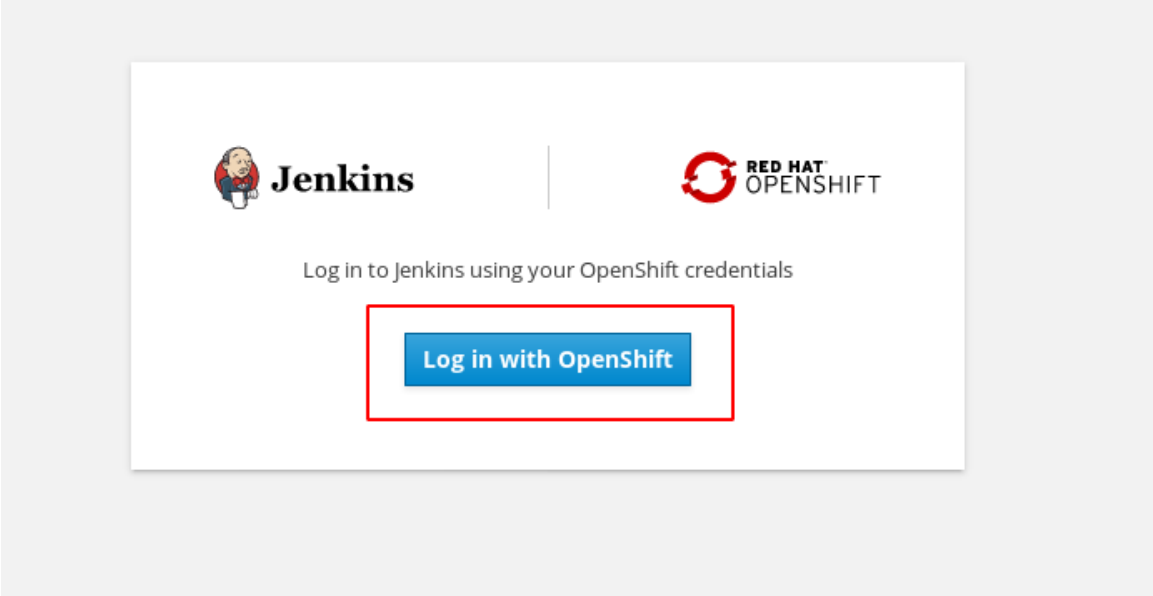


Y damos clic al botón “Create”

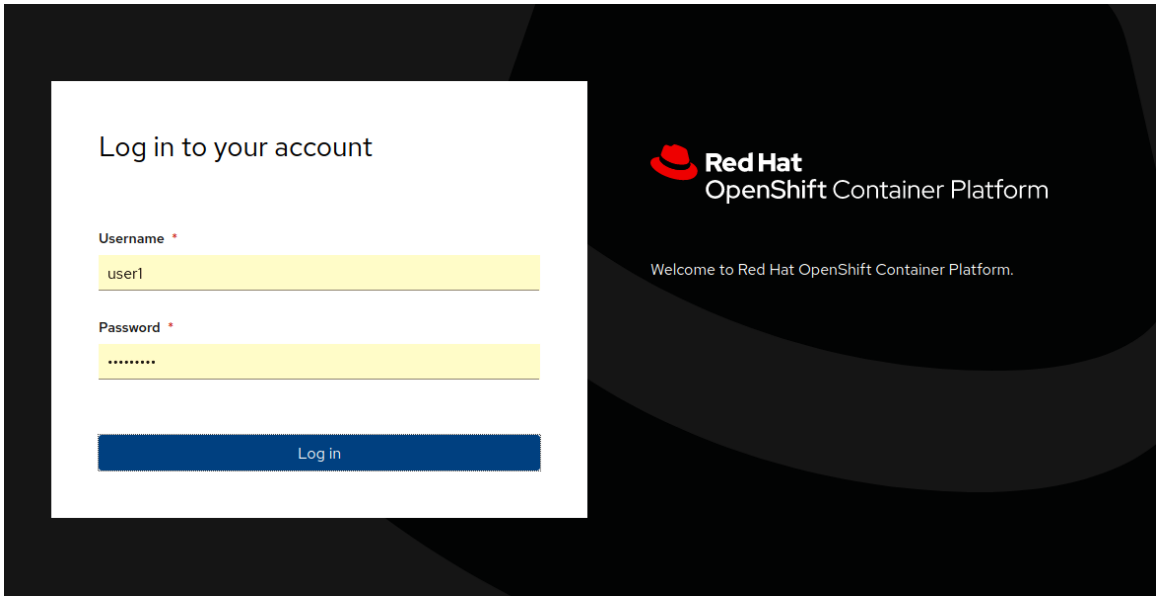
6.6 Una vez creado nuestro pod de Jenkins, desde la vista Topology podemos ver los parámetros de su configuración, la ruta de acceso y demás. Verificamos el estado del pod, damos unos minutos para que se complete la configuración. Para ello damos clic en la ruta mostrada en el apartado "Routes" para acceder a nuestro servidor Jenkins.



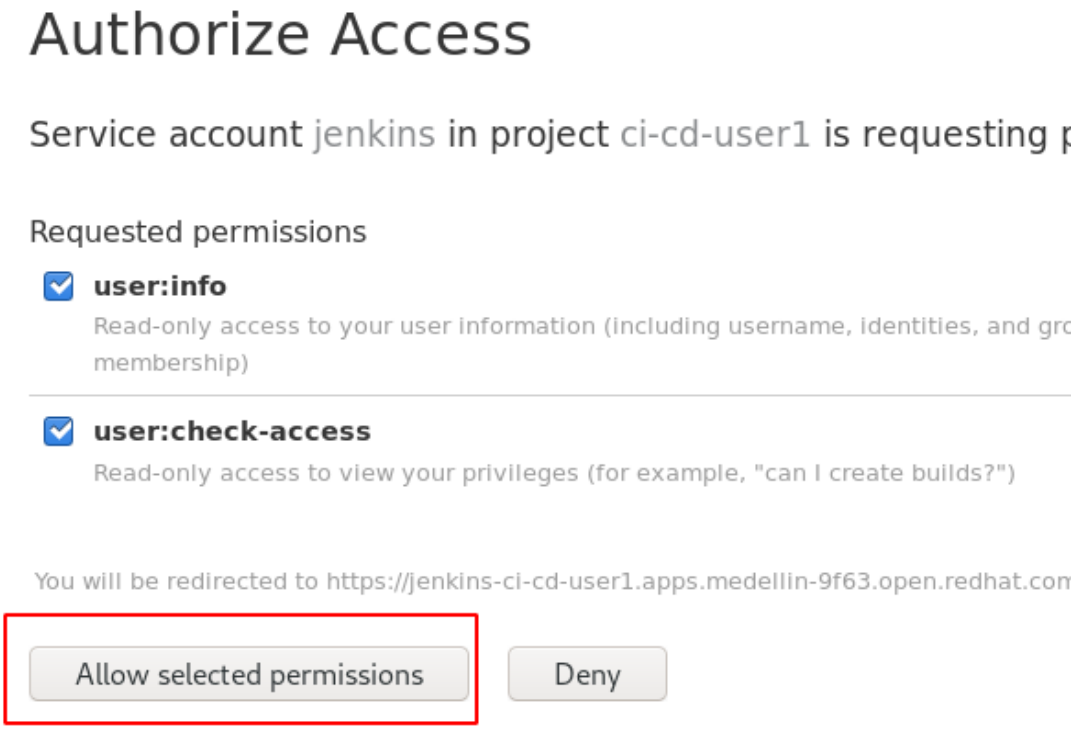
6.7 Al abrir la ruta de Jenkins, debemos aceptar el certificado y darle click en "**Login with Openshift**", para abrir el Jenkins.



Y nos pedirá nuevamente ingresar nuestro usuario y contraseña de openshift

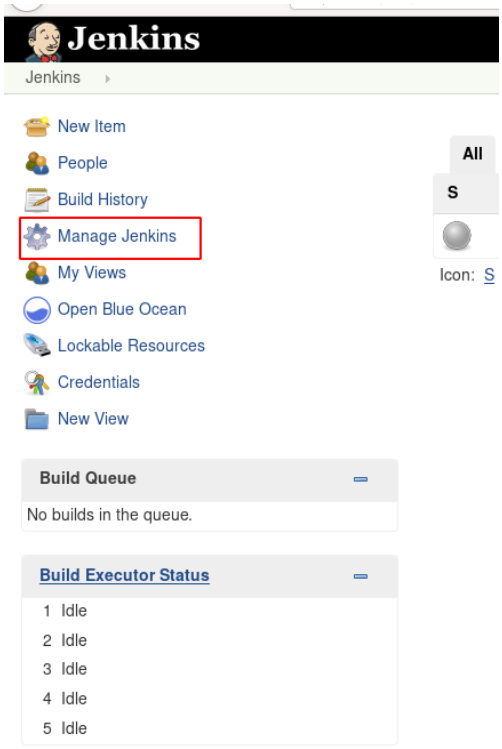


Nos pedirá aceptar autorizaciones de acceso y daremos clic al botón "Allow selected permissions"

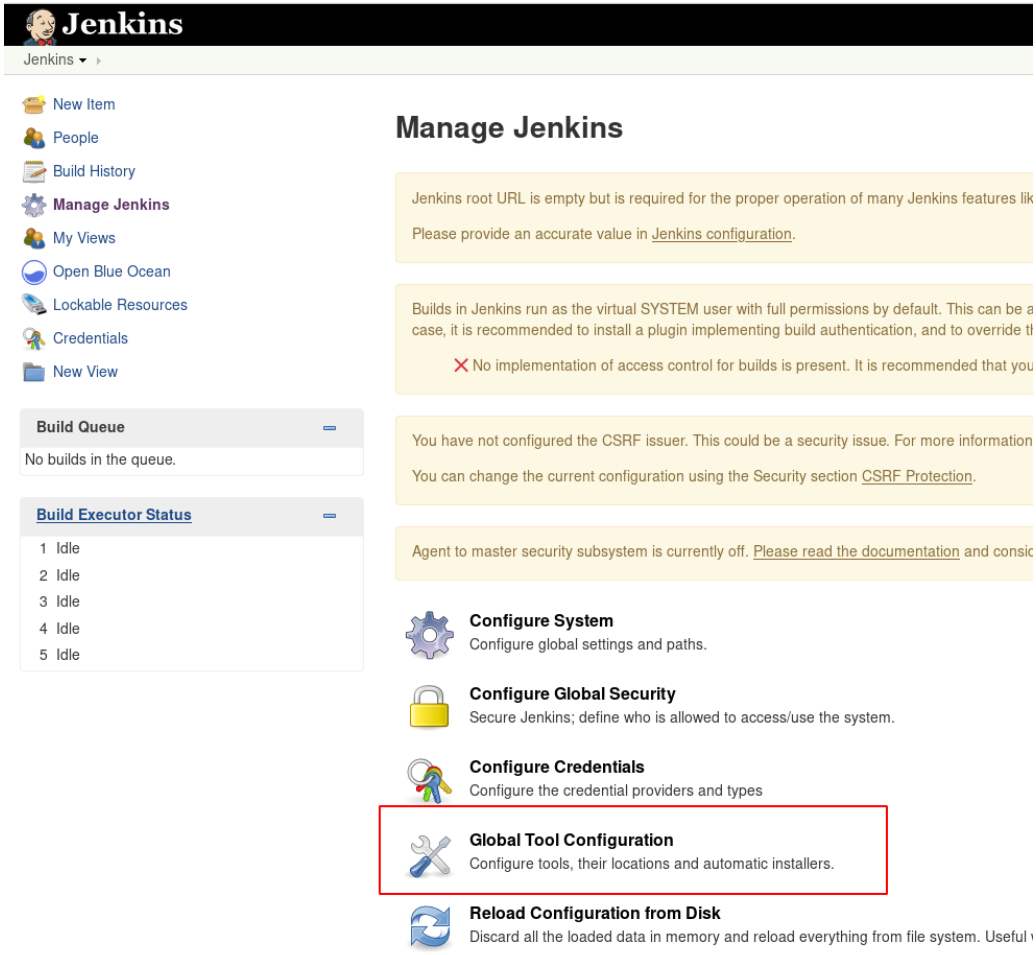


Luego de esto ya tendremos acceso a nuestro servidor Jenkins

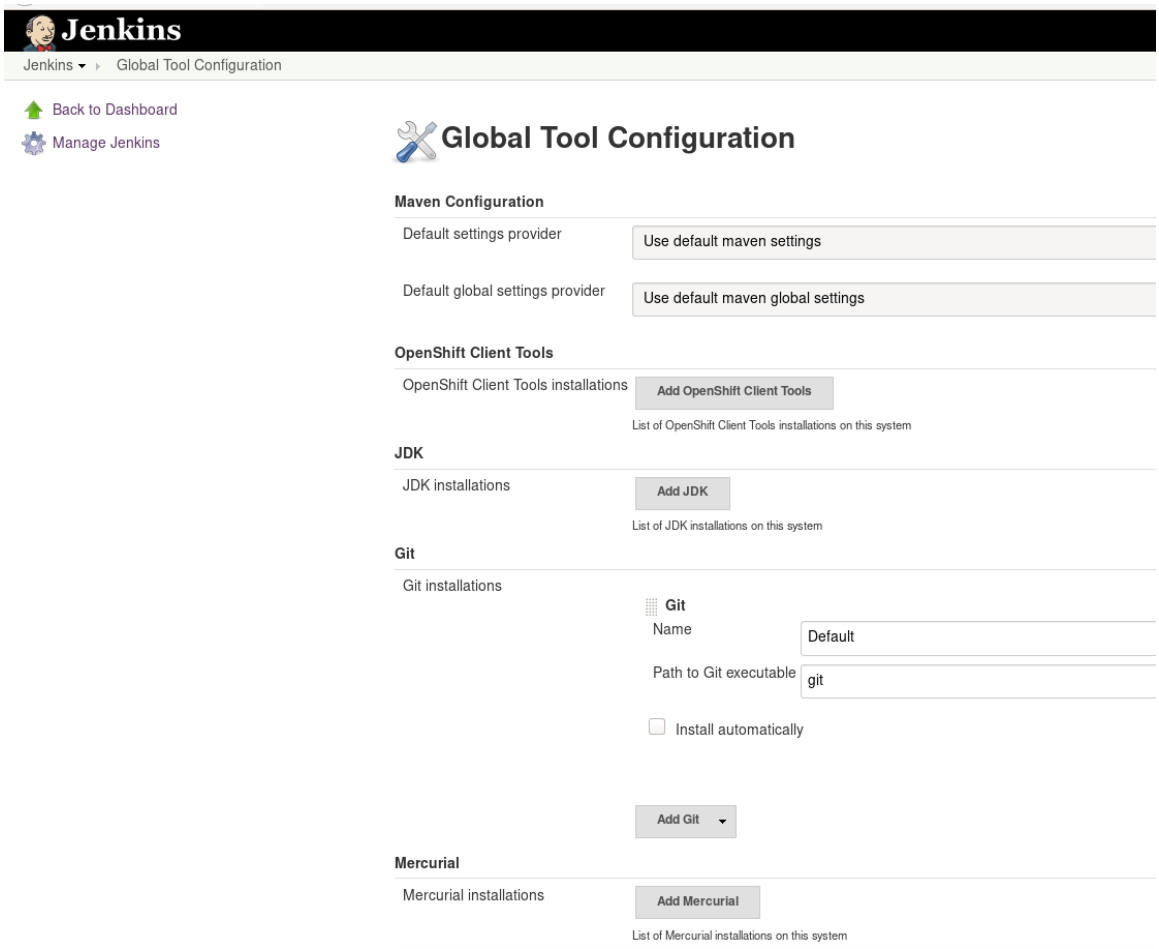
6.8 Debemos activar el **maven** en el Jenkins esto para poder hacer despliegue de aplicaciones Java.
Para esto le damos **click** desde el Jenkins donde dice "**Manage Jenkins**".



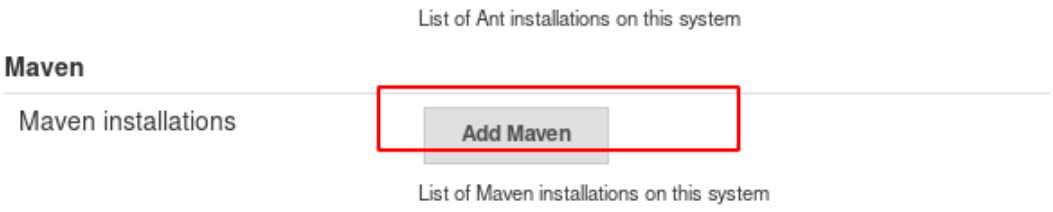
Le damos click en “**Global Tool Configuration**”



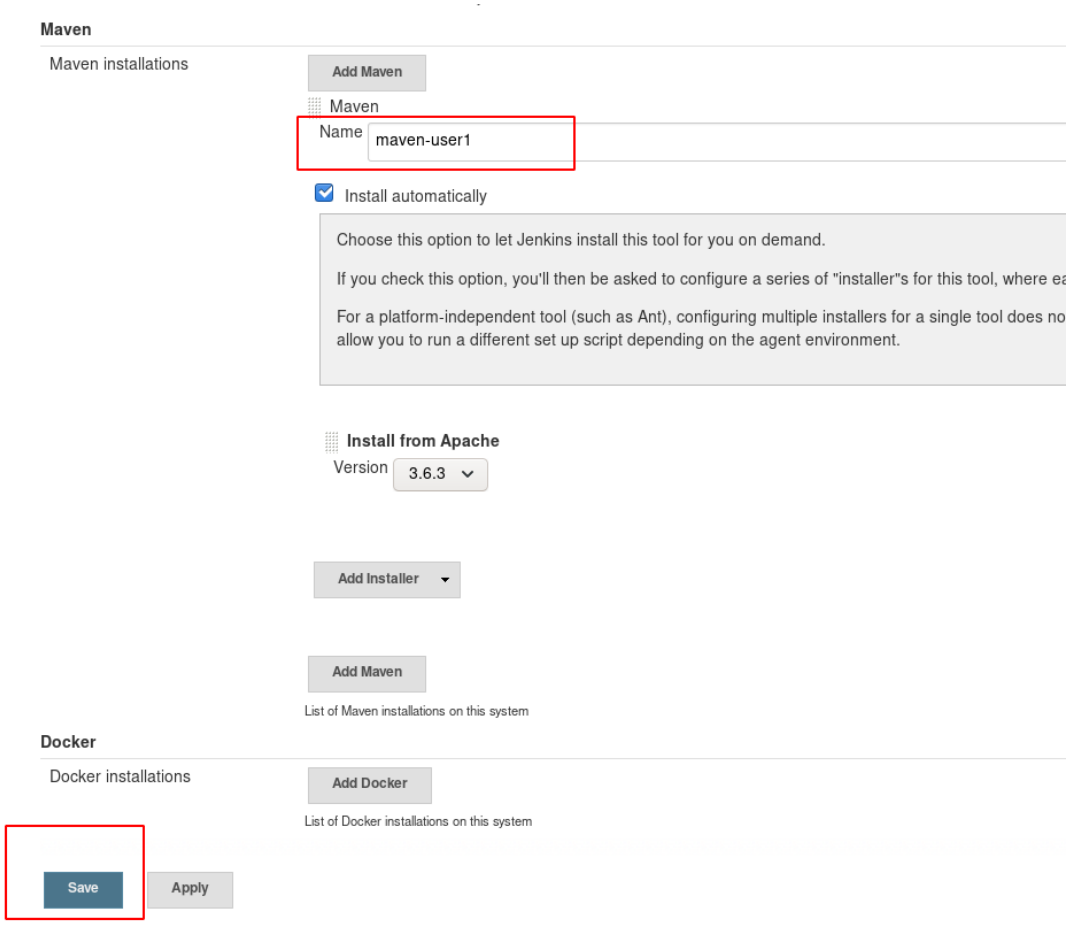
Vamos a ver una imagen como la siguiente.



Buscamos **Maven** y lo agregamos:



Agregamos el nombre "**maven-userx**", como se muestra en la imagen y le damos **click** en "**Save**".



Ahora que Jenkins está listo, configura un proyecto en OpenShift para mantener la aplicación que se creará utilizando la línea de comando.

6.9 Desde la línea de comandos de **oc**, y **logueado**s a la misma, creamos el proyecto: "**user1-cicd-tareas**"

```
$ oc new-project user1-cicd-tareas
Now using project "user1-cicd-tareas" on server
"https://api.cluster-qwe-e0e6.qwe-e0e6.example.opentlc.com:6443".

You can add applications to this project with the 'new-app' command. For example, try:

 oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.[default@wetty-1-9nclg ~]$
```

6.10 Creamos la aplicación que vamos a utilizar con Jenkins con el siguiente comando: "**oc new-app jboss-eap71-openshift:1.3 https://github.com/redhat-gpte-devopsautomation/openshift-tasks**"

```
$ oc new-app jboss-eap71-openshift:1.3 https://github.com/redhat-gpte-devopsautomation/openshift-tasks
```

6.11 Procedemos a **exponer** el **servicio** para poder **accesar** a la aplicación fuera del cluster de Openshift.

```
$ oc expose svc openshift-tasks
```

6.12 **Apagar** todos los **triggers automáticos**.

Debido a que está creando esta aplicación utilizando un **pipeline de Jenkins**, es **Jenkins quien debe tener control total sobre lo que sucede en este proyecto**. Por defecto, la aplicación se vuelve a implementar cada vez que hay una nueva imagen disponible. Sin embargo, si reconstruye la imagen a través de Jenkins, es posible que desee ejecutar algunas pruebas antes de volver a implementar la aplicación.

```
$ oc set triggers dc openshift-tasks --manual
```

6.13 Otorgue a la cuenta de servicio los permisos correctos para editar objetos en este proyecto para permitir que Jenkins construya e implemente la aplicación.

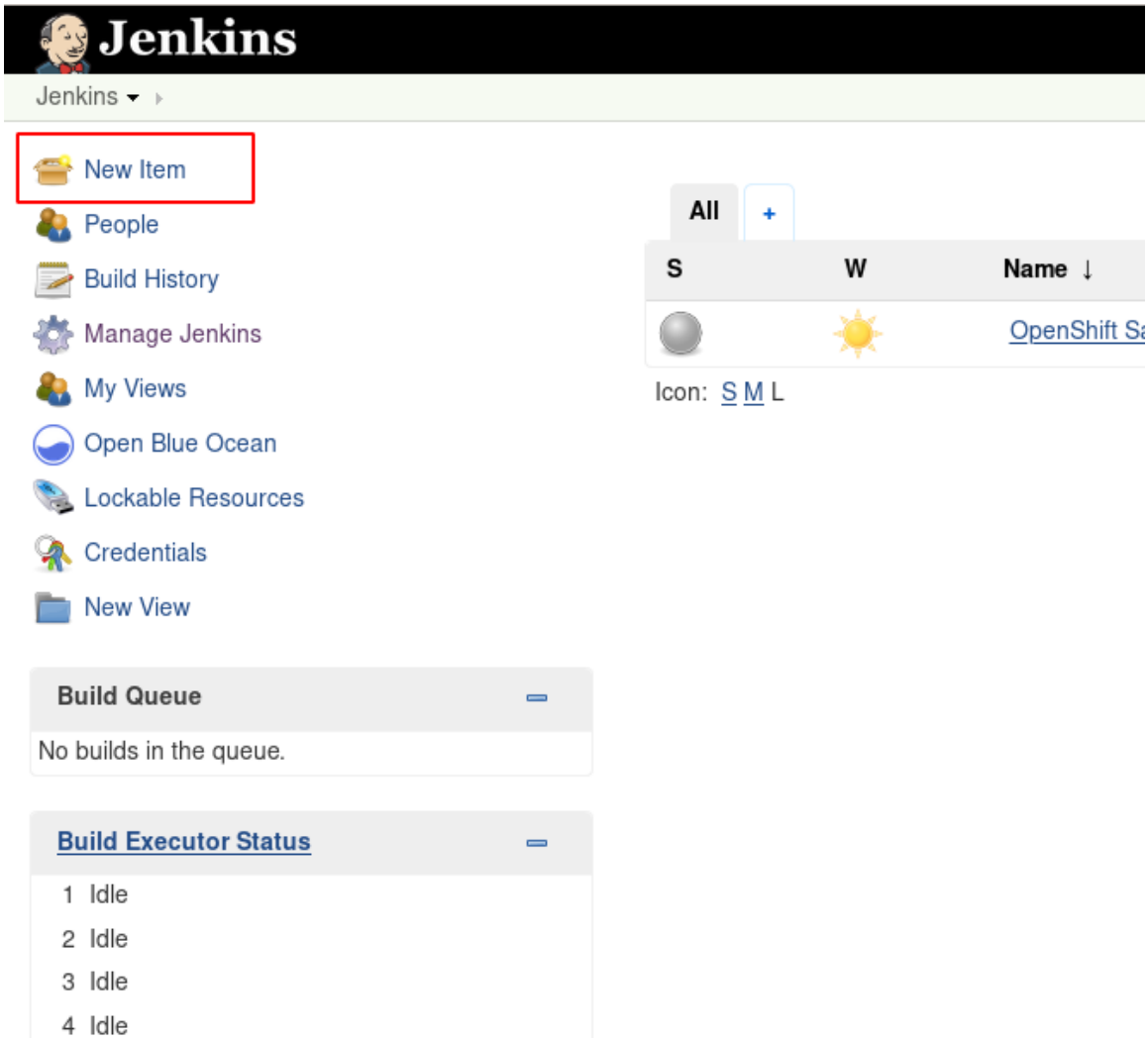
Para esto ejecutamos el siguiente comando: verificar que estemos utilizando el proyecto correspondiente, se subrayan en negro para su atención.

oc policy add-role-to-user edit system:serviceaccount:userx-lab6:jenkins -n userx-cicd-tareas

```
$ oc policy add-role-to-user edit system:serviceaccount:userx-lab6:jenkins -n userx-cicd-tareas
```

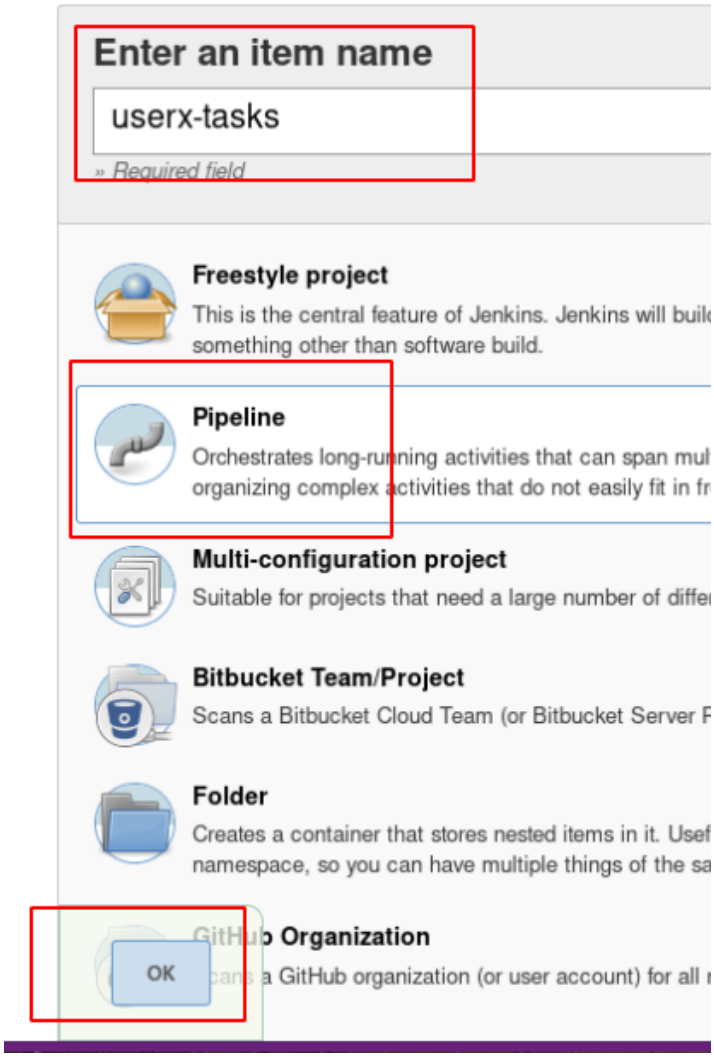
6.14 Crear pipeline en Jenkins.

Loguearse a Jenkins, en el **navigator** en la izquierda, **click** en "**New Item**".



6.15 En la página de "**New Item**", lo llenamos de la siguiente forma.

Enter an Item name: userx**-tasks**
Seleccionar "**Pipeline**" para el tipo de job.
Y le damos **click** en **"Ok"**



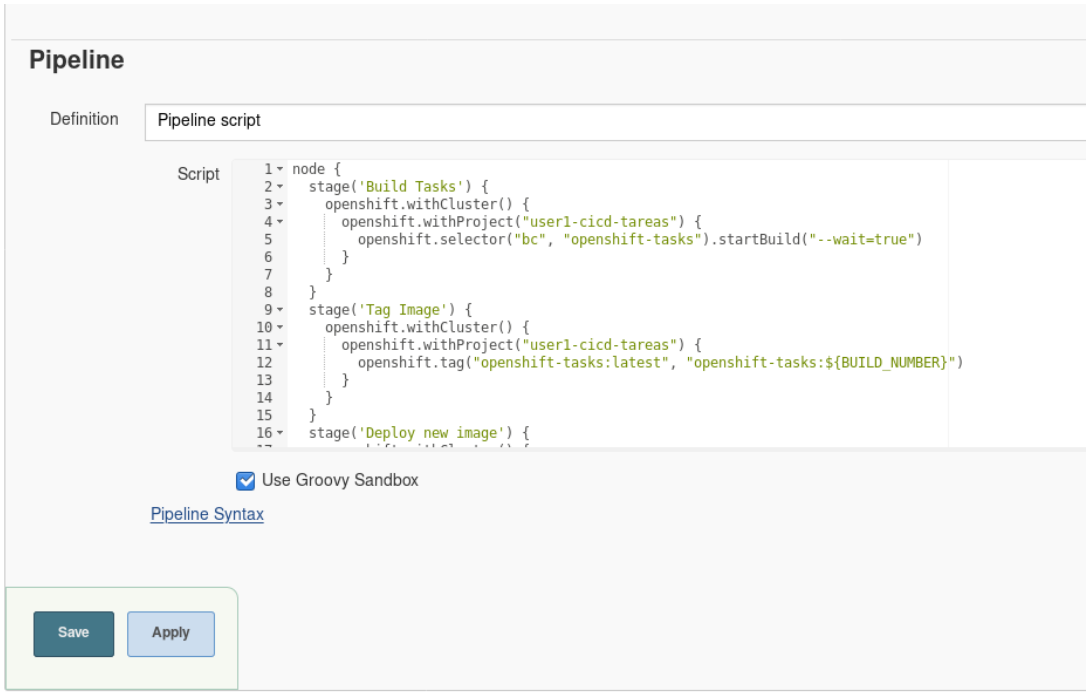
6.16 En la carpeta donde están los documentos del workshop, verificamos el archivo "**pipeline-openshift-tasks.yaml**", lo abrimos y modificamos el valor que vemos en color rojo en el texto a continuación.

```
node {
  stage('Build Tasks') {
    openshift.withCluster() {
      openshift.withProject("userx-cicd-tareas") {
        openshift.selector("bc", "openshift-tasks").startBuild("--wait=true")
      }
    }
  }
  stage('Tag Image') {
    openshift.withCluster() {
      openshift.withProject("userx-cicd-tareas") {
        openshift.tag("openshift-tasks:latest", "openshift-tasks:${BUILD_NUMBER}")
      }
    }
  }
  stage('Deploy new image') {
    openshift.withCluster() {
      openshift.withProject("userx-cicd-tareas") {
        openshift.selector("dc", "openshift-tasks").rollout().latest();
      }
    }
  }
}
```

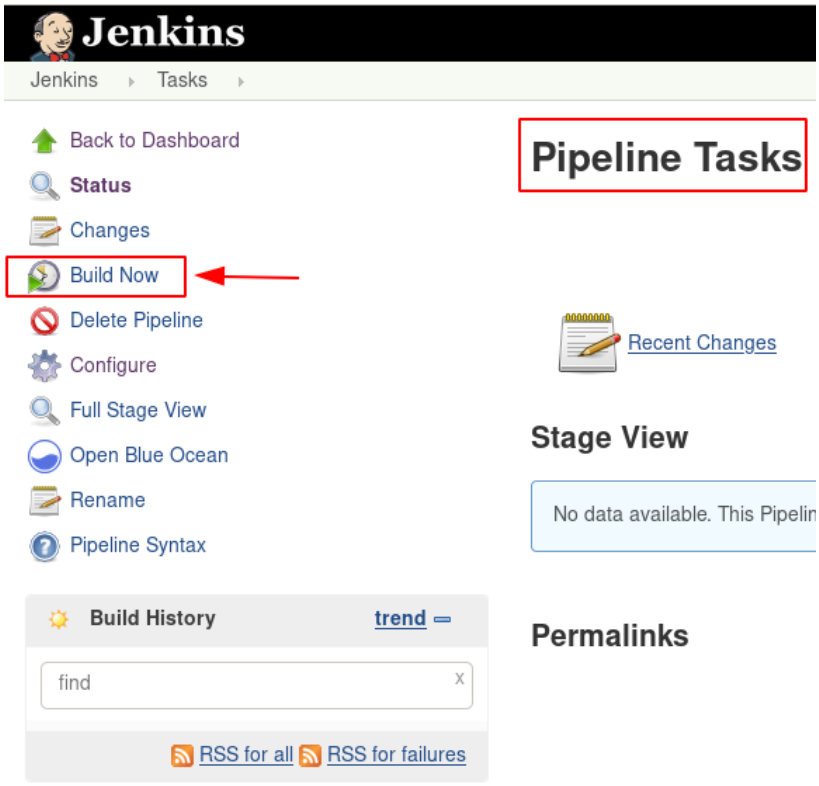
Asegúrese de que el project/namespace "**userx-cicd-tareas**" apunta al nombre real del Proyecto en la opción "**openshift.withProject**" con el que nosotros creamos en pasos anteriores, debe coincidir con nuestro usuario.

Este texto lo debemos pegar en el campo de **Pipeline,** después de haberlo modificado, este campo está en la última parte de la configuración del pipeline, para ser específicos en la **Definition Pipeline script.**

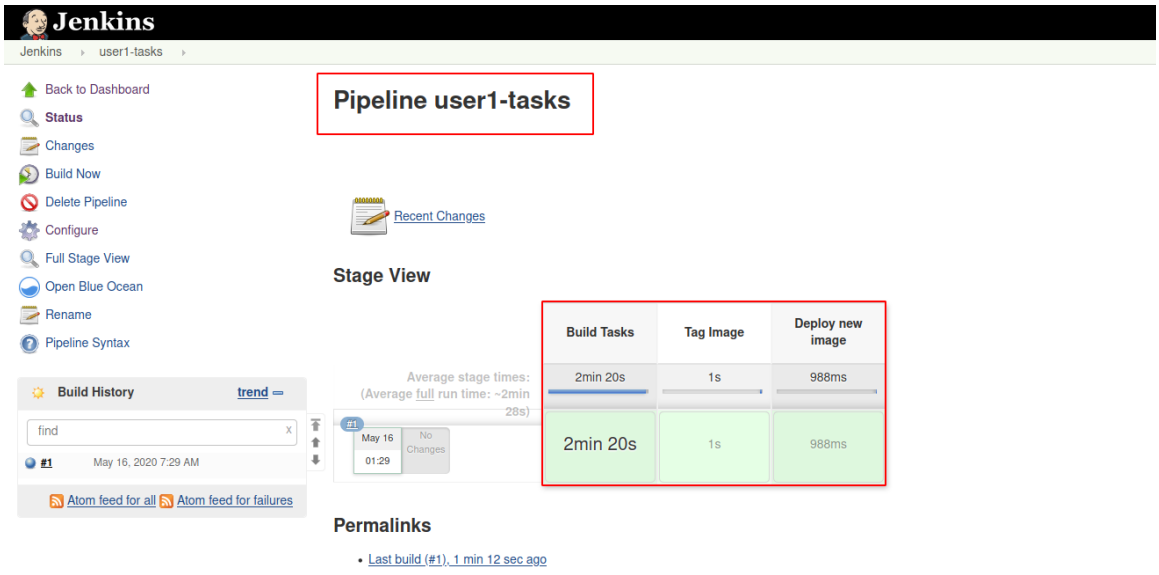
Y hay que darle **click** en "**Save**" a este **pipeline job.**



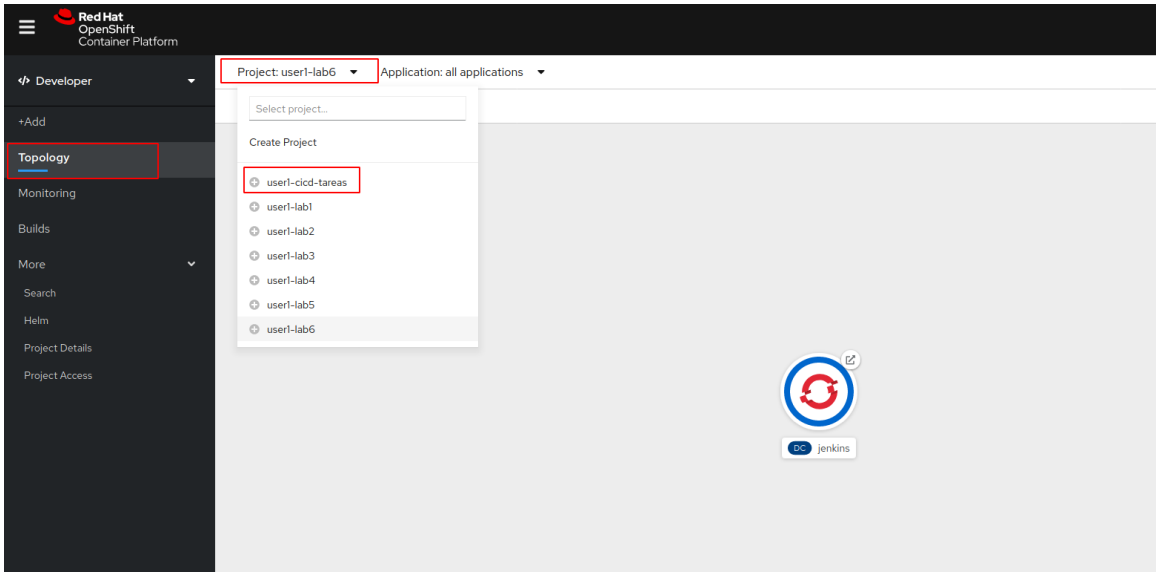
6.17 En la página de Jenkins, **click** en "**Build Now**" .



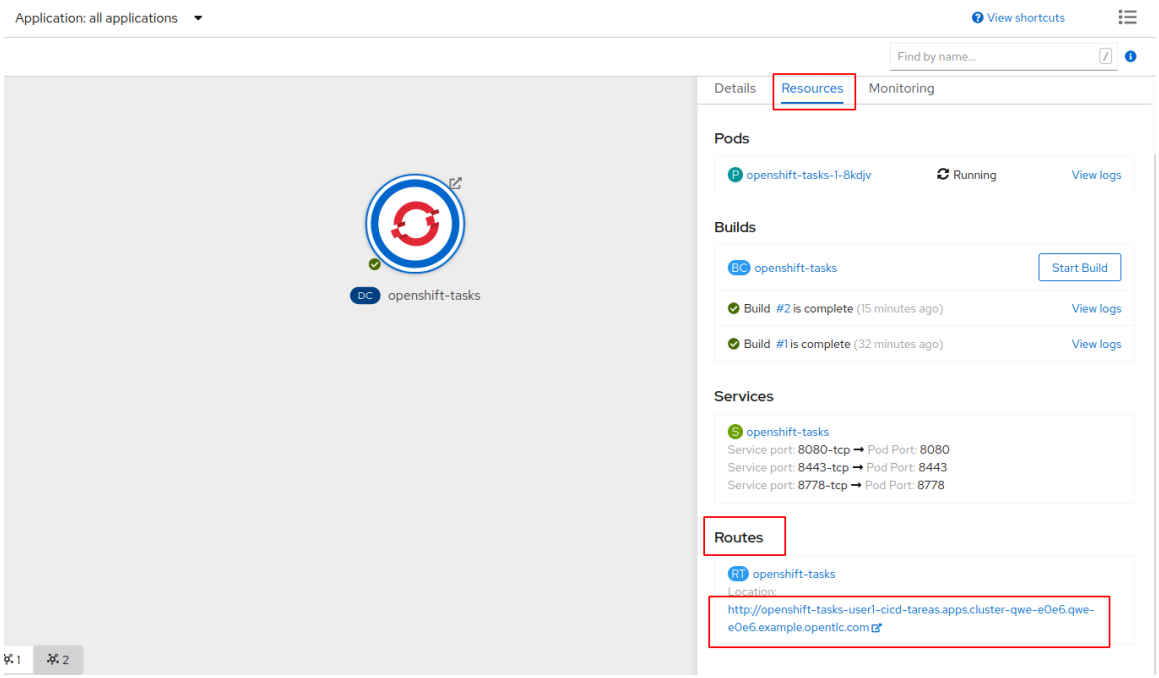
Nos esperamos a que se realice el **build** de forma correcta, el mismo **dura alrededor de 3 minutos**.



6.18 Ya con el **build** realizado de forma correcta, ingresamos a la consola web de Openshift, abrimos el **proyecto donde hicimos el deploy**, en este caso: "**userx-cicd -tareas**". Para ellos desde la vista Topology desplegamos la lista de proyectos y buscamos el proyecto llamado "**userx-cicd -tareas**"



Para ingresar al URL, dentro de la parte de **Resources**, hay una parte de rutas, en esa debe estar el URL de acceso a la aplicación. Para ello damos clic en nuestro Deploy Config y nos vamos a la sección de rutas.



6.19 Abrimos la página, desde la ruta que se muestra en el proyecto "**user1-cicd-tareas**", en este caso sería: http://openshift-tasks-user1-cicd-tareas.apps.cluster-qwe-e0e6.qwe-e0e6.example.opentlc.com/