Red Hat
OpenShift

# Openshift Introduction

***Workshop for developers***

Ian Uther Lawson

Version 1.0.0, July 31, 2019

# Table of Contents

# Introduction

## Attendee details

This workshop is designed to give developers an introduction to Openshift

❗ **TODO**

Add basic *What is Openshift* section ?

Add more info in the intended audience.

Add relevant links

# Setup

## Web console

You will be assigned an ID by the facilitator. At any time that the content refer to userX, (or *anything_X) you need to replace X with your ID. Your userId to log onto the web console will be _userX* and the password will be *openshift*

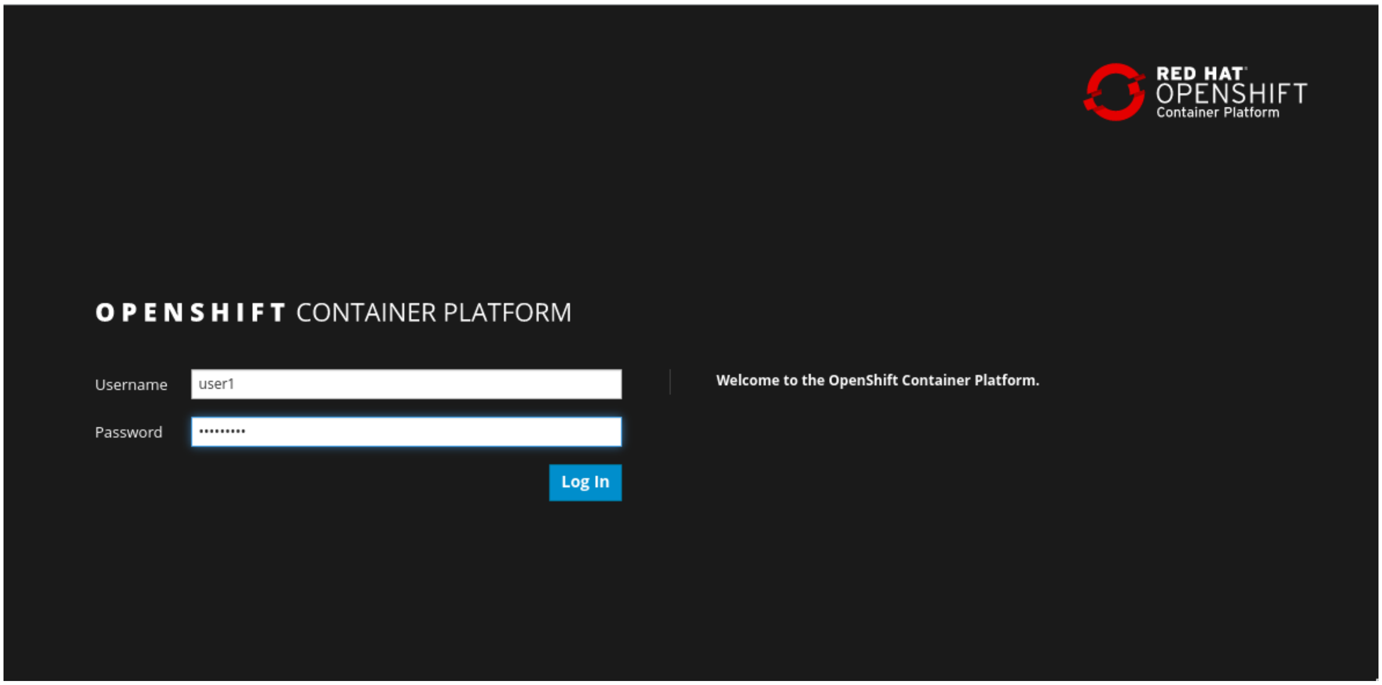
You can access the web console using your browser:

http://myopenshift.com

**Example:** If your ID is 1:

- **Username:** user1
- **Password:** openshift

Click **[ Log In ]**



OPENSHIFT CONTAINER PLATFORM

Username: user1

Password: •••••••••

Welcome to the OpenShift Container Platform.

## CLI (oc)

With the OpenShift Enterprise command line interface (CLI), you can create applications and manage OpenShift projects from a terminal. The CLI is ideal in situations where you are:

- Working directly with project source code.
- Scripting OpenShift Enterprise operations.

- Restricted by bandwidth resources and cannot use the web console.

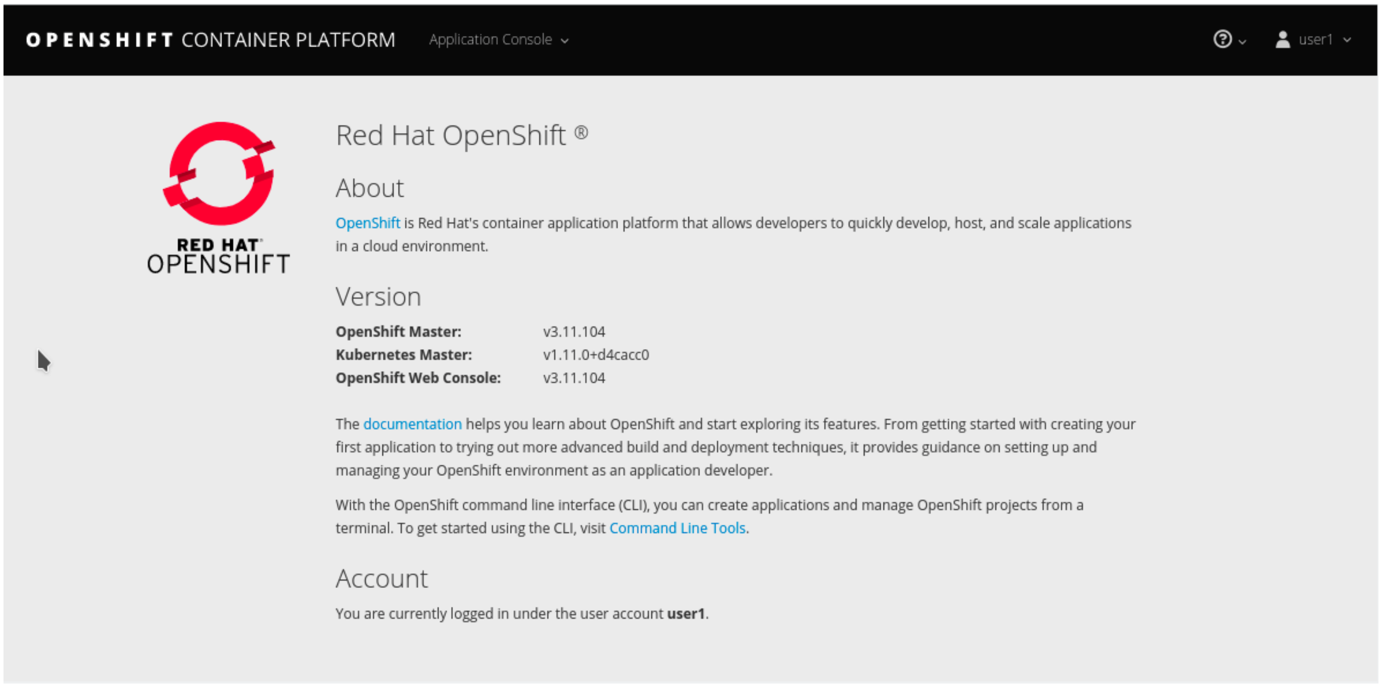
The CLI is available using the oc command:

```
oc {command}
```

You have two options to get a working CLI. Either install `oc` on your localhost or use a docker image in OpenShift:

## Option 1: Installing the CLI on localhost

The easiest way to download the CLI is by accessing the About page on the web console [ (?)→ About ]



Red Hat OpenShift ®

About

OpenShift is Red Hat's container application platform that allows developers to quickly develop, host, and scale applications in a cloud environment.

Version

**OpenShift Master:** v3.11.104
**Kubernetes Master:** v1.11.0+d4cacc0
**OpenShift Web Console:** v3.11.104

The documentation helps you learn about OpenShift and start exploring its features. From getting started with creating your first application to trying out more advanced build and deployment techniques, it provides guidance on setting up and managing your OpenShift environment as an application developer.

With the OpenShift command line interface (CLI), you can create applications and manage OpenShift projects from a terminal. To get started using the CLI, visit Command Line Tools.

Account

You are currently logged in under the user account **user1**.

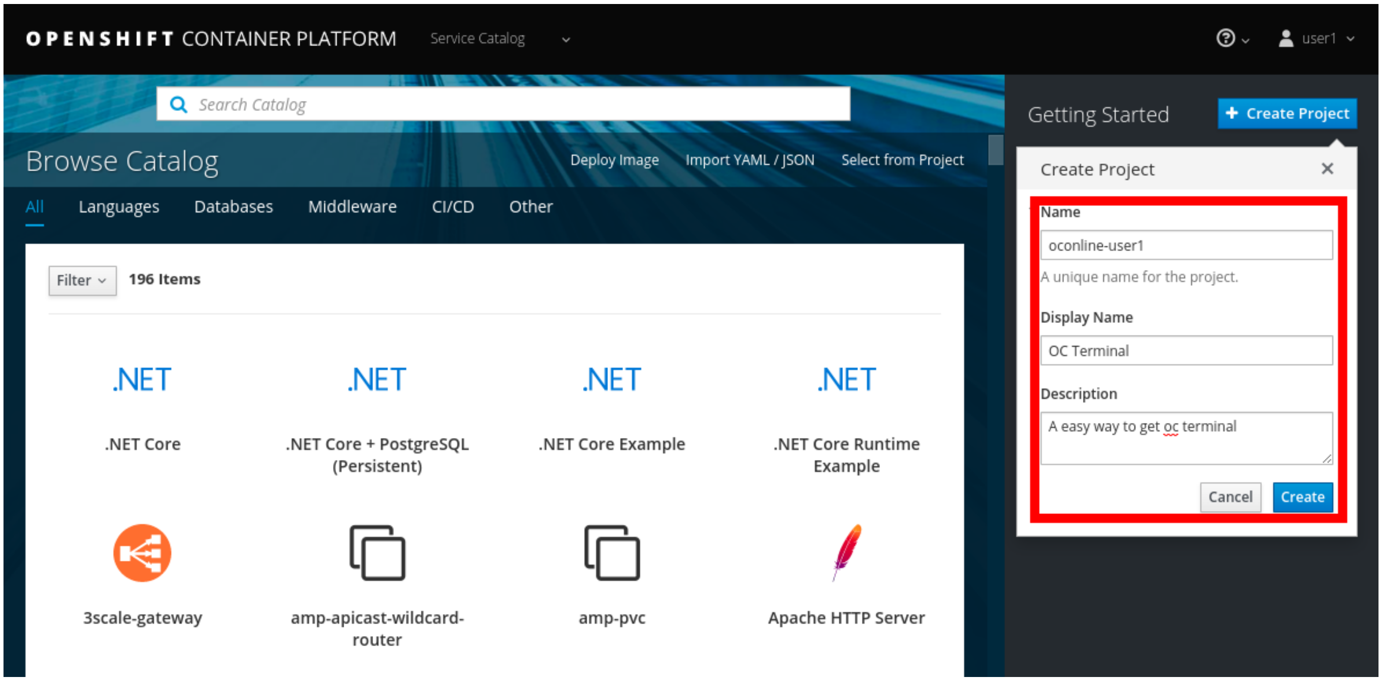
or you can follow the instructions from the official documentation

## Option 2: Use a pre-configured docker image on OpenShift

This option install and run a docker image inside OpenShift that already has oc installed and configured. To install this image, do the following:

- Login to the OCP system through the UI. (userX / openshift)
- Create a new project:
  - **Name:** oconline-userX
  - **Display Name:** OC Terminal

Click **[ Create ]**



Create Project

Name

oconline-user1

A unique name for the project.

Display Name

OC Terminal

Description

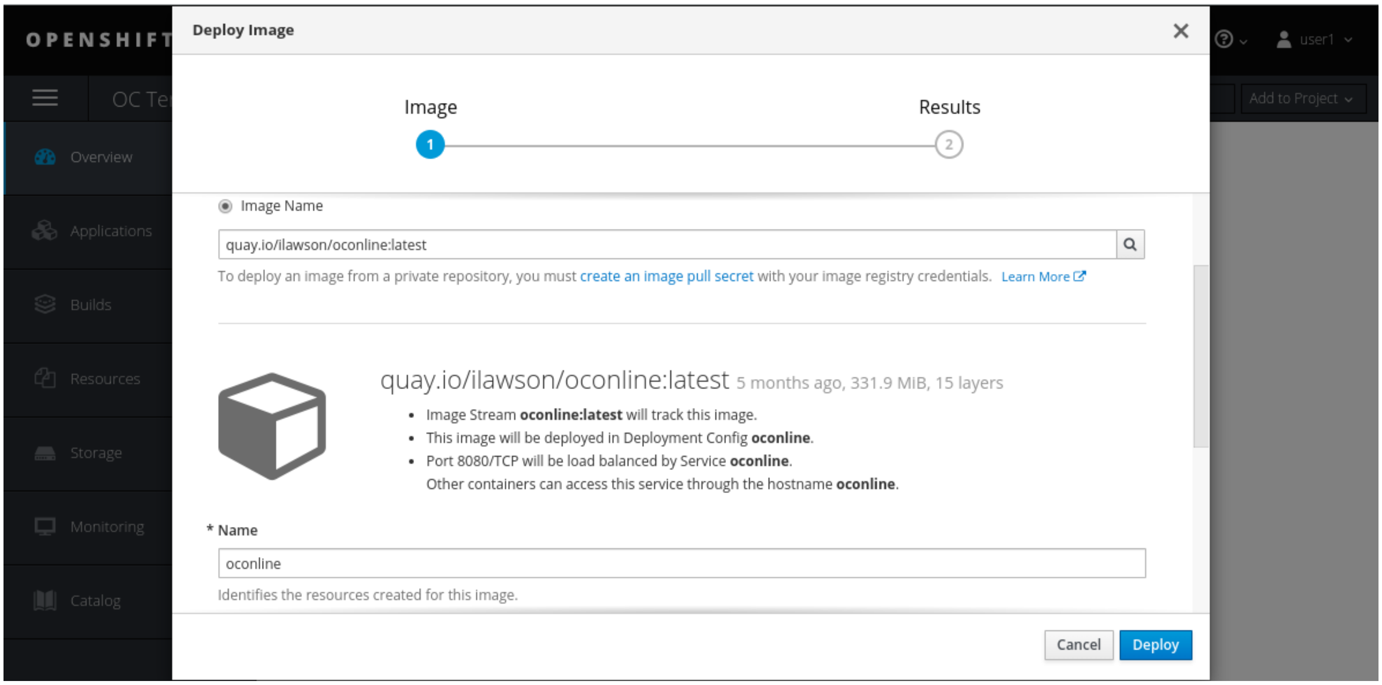
A easy way to get oc terminal

Select the newly created `OC Terminal` project and then select `Deploy Image`

Use the Image Name option and add the following name:

```
quay.io/ilawson/oconline:latest
```

Click **[ Search ]** and then leave the default values in the metadata form and click **[ Deploy ]**



Deploy Image

● Image Name

quay.io/ilawson/oconline:latest

To deploy an image from a private repository, you must create an image pull secret with your image registry credentials. Learn More

quay.io/ilawson/oconline:latest 5 months ago, 331.9 MiB, 15 layers

- Image Stream **oconline:latest** will track this image.
- This image will be deployed in Deployment Config **oconline**.
- Port 8080/TCP will be load balanced by Service **oconline**.
  Other containers can access this service through the hostname **oconline**.

* Name

oconline

Identifies the resources created for this image.

Once the image has deployed and a Pod will appear

- Click on the Pod ring (blue)

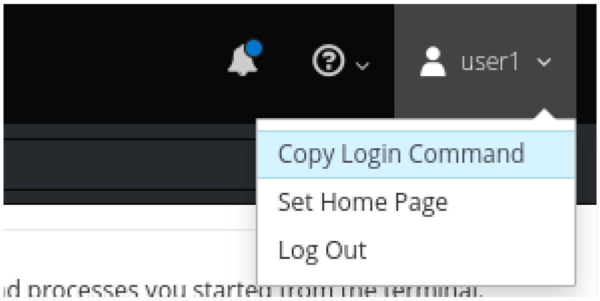
- Select `Terminal` from the options (This will be *your terminal* you can use to do `oc` commands)

> It might be easier to keep this project with the terminal view open in a separate browser tab, so that you can switch back and forth to it easy

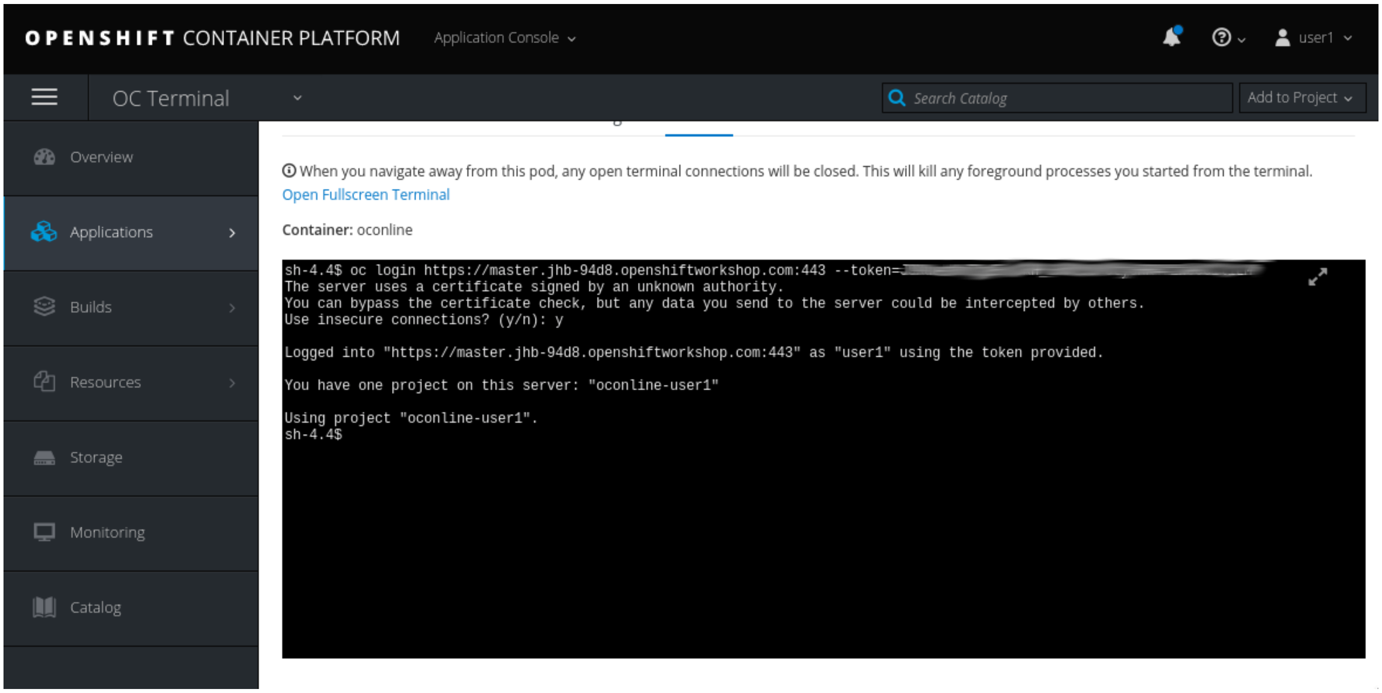
# Using your terminal (Both options)

If this document refer to *your terminal* it will either be on your localhost or the docker install terminal depending your option above.

- In the Web Console, select the top right pulldown, choose **[ Copy Login Command ]**



- Paste that command into *your terminal*, hit return - hit 'y' for insecure access



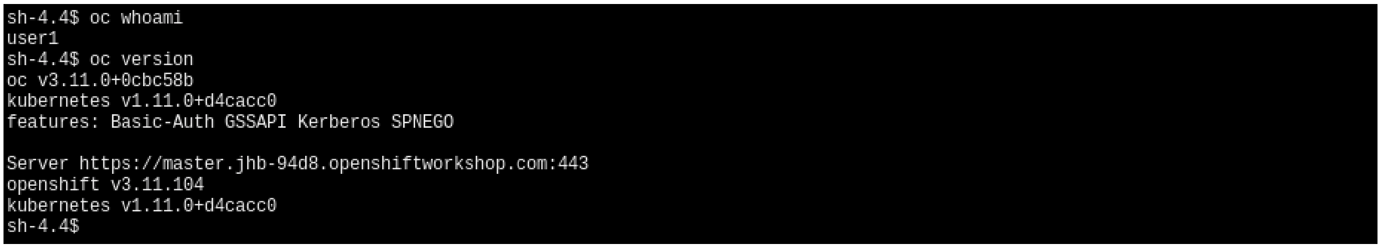
When you navigate away from this pod, any open terminal connections will be closed. This will kill any foreground processes you started from the terminal.

**Container:** oconline

```
sh-4.4$ oc login https://master.jhb-94d8.openshiftworkshop.com:443 --token=[redacted]
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be intercepted by others.
Use insecure connections? (y/n): y

Logged into "https://master.jhb-94d8.openshiftworkshop.com:443" as "user1" using the token provided.

You have one project on this server: "oconline-user1"

Using project "oconline-user1".
sh-4.4$
```

Make sure `oc` is working, type:

```
oc whoami
oc version
```



```
sh-4.4$ oc whoami
user1
sh-4.4$ oc version
oc v3.11.0+0cbc58b
kubernetes v1.11.0+d4cacc0
features: Basic-Auth GSSAPI Kerberos SPNEGO

Server https://master.jhb-94d8.openshiftworkshop.com:443
openshift v3.11.104
kubernetes v1.11.0+d4cacc0
sh-4.4$
```

Also see the **Command-Line Walkthrough**: https://docs.openshift.com/container-platform/3.11/getting_started/developers_cli.html

# Web Console overview

## Sandbox project

So you have already created your first project using the web console (`oconline-userX`).

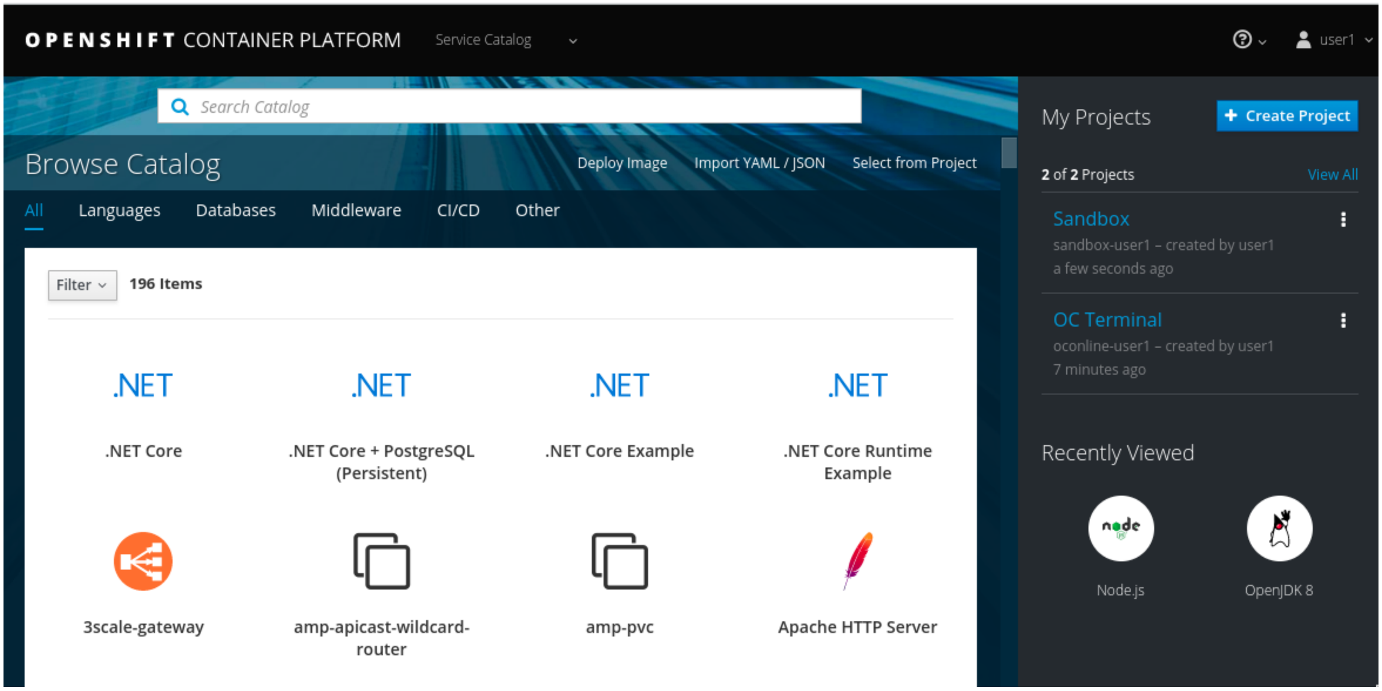
Let’s create another project where we can play in:

**Create a new project:**

- **Name:** sandbox-userX
- **Display Name:** Sandbox

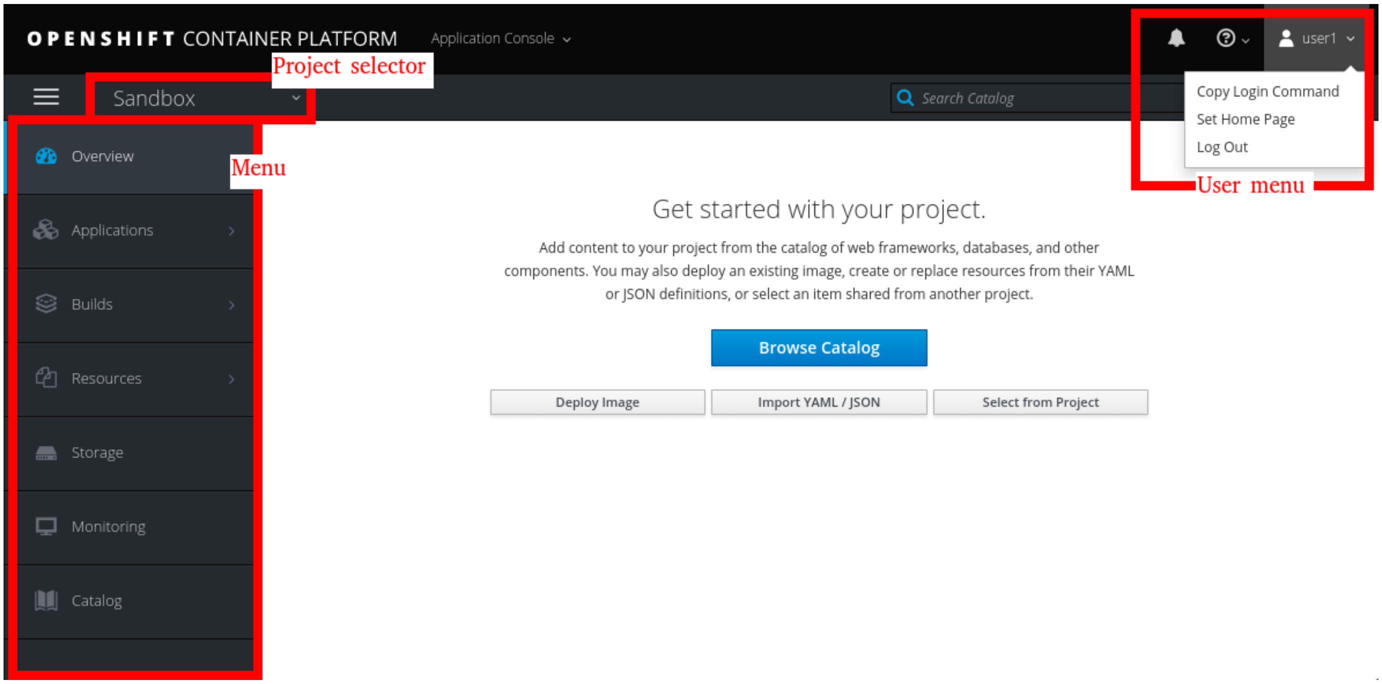
Click **[ Create ]**

You should now have 2 projects like this:



Click the **Sandbox** project to go to the *Application Console* for that project, you will see:

- **Project selector** (top left): Here you can switch between your projects
- **Menu** (left): This is all the sections available.
- **Context area** (center): This will display content based on the menu item selected.
- **User menu** (top right): Here you can get the cli login command.



Get started with your project.

Add content to your project from the catalog of web frameworks, databases, and other components. You may also deploy an existing image, create or replace resources from their YAML or JSON definitions, or select an item shared from another project.

# Build options

A build is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A BuildConfig object is the definition of the entire build process. OpenShift Container Platform leverages Kubernetes by creating Docker-formatted containers from build images and pushing them to a container image registry. Build objects share common characteristics: inputs for a build, the need to complete a build process, logging the build process, publishing resources from successful builds, and publishing the final status of the build. Builds take advantage of resource restrictions, specifying limitations on resources such as CPU usage, memory usage, and build or pod execution time.

The OpenShift Container Platform build system provides extensible support for build strategies that are based on selectable types specified in the build API. There are three primary build strategies available:

- Docker build
- Source-to-Image (S2I) build
- Custom build

By default, Docker builds and S2I builds are supported. The resulting object of a build depends on the builder used to create it. For Docker and S2I builds, the resulting objects are runnable images. For Custom builds, the resulting objects are whatever the builder image author has specified.

Additionally, the Pipeline build strategy can be used to implement sophisticated workflows:

- continuous integration
- continuous deployment

Also see the **Builds and Image Streams**: https://docs.openshift.com/container-platform/3.11/architecture/core_concepts/builds_and_image_streams.html#builds

We have already done a docker build with `oconline`.

## The catalog

Click **[ Browse Catalog ]**.

OpenShift Container Platform provides out of the box a set of languages and databases for developers with corresponding implementations and tutorials that allow you to kickstart your application development. Language support centers around the Quickstart templates, which in turn leverage builder images.

**Openshift** is effectively **Kubernetes**, and as such it works with Objects that are part of the Kubernetes/Openshift Object model. When interacting with this model through the interfaces, be it CLI via *oc* or the web interface, a User is creating, deleting and changing Objects in this model.

The **Catalog** is designed as a quickstart way to interact with the Openshift instance by pre-creating the objects you need for a specific Application. It does this by providing Templates, which are pre-created sets of objects with parameterised components that have to be provided to create the application - a good example of this is the node.js Template, which requires only three initial parameters (with a lot of advanced and optional parameters for deeper configuration) to create a full Application within Openshift.

These parameters, in the case of the node.js Template, are the base image to build upon, the name to assign to all the Objects created as part of the Template and the git repo containing the node.js source to build into the Application image. When you choose this Template from the catalog you are provided with a Wizard that prompts for those three essential parameters (plus provides advanced inputs for the additional ones).

There is another type of Template supported by Openshift called a *Quickstart*. This is identical in theory to the Template but instead has all the required Parameters pre-defined so a User can simply create an instance of the Quickstart without having to provide any information other than the name of the Application to create.

In Openshift 3 the Catalog also supports **Services**. This is an implementation of the Service Broker concept and allows external and pre-created Services, where Service in this case refers to an external Service providing functionality as opposed to the services internally which are Application endpoints within Openshift. Services provided via the Service Broker are external interfaces with a provided Service definition that defines how to call them and what they contain. A User can add a Service to his namespace/project, and when he does that Service is callable vie the defined interface from his Applications in the namespace/project. This functionality is being wound down as part of Openshift 4 with the functionality being instead provided by **Operators**.

**Operators** are a fantastic new concept that simplifies the administration and on-going maintenance of Kubernetes applications. How they work is very elegant - an Operator maintains the state of a set of Objects within the Openshift/Kubernetes object model **but** can also extend it. An Operator is itself a running Pod, and this Pod handles installation of Objects, updates of Objects and monitoring and response to change events in the Object state and behaviour.

An easier explanation is this - imagine you are writing an Application that needs to be deployed in Openshift. This Application is complex, consisting of multiple Pods, configuration options, and you want to be able to update it in real time as fast as you can. By producing an Operator, which installs the Object model and then can act on changes and events concerning it (which it does by extending the Openshift/Kubernetes Object model with Custom Resource Definitions, which can then be triggered to be updated by simply providing updated YAML defintitions of the type defined), the Application can be deployed simply by passing the appropriate YAML for the Custom Resource, which is specific and contextual *only* to the Application, to the Operator.

Red Hat has been instrumental in creating the operatorhub.io site, where people can publish Operators than can be easily consumed by Openshift 4. In fact the functionality for Operators has been back ported into Openshift 3.11 because it is so powerful and essential. As of Openshift 4 the Catalog has direct linkage to the operatorhub and allows Users to request Operators to consume.

Interestingly these are handled in a much more secure way than with vanilla Kubernetes. The Operator, when installed, needs to alter the core Object model of the Cluster, which is very dangerous. Openshift prohibits the installation of Operators (you need Cluster Admin anyway) by implementing an Operator itself, called the Operator Lifecycle Manager.

This Operator (shortened to **OLM**) allows Admins to install Operators as *Subscriptions* which can then be used by named Users, Projects or globally depending on the security required. The OLM handles the update of the Operators, and the Users can consume them simply by creating Objects of the appropriate Custom Resource. This way Users do not need to install the Operators themselves - installation and Object model update is controlled centrally by the OLM and Administrators.

**TODO**

Explain how the catalog works in terms of templates, quickstarts, services, (optional) operators.

Explain in detail - templates are object model definitions that require additional parameters.

Explain in detail - quickstarts are object models definitions that contain predefined parameter values.

Explain the oauth token (derived from the *copy login command*)

Also see the **Web Console Walkthrough**: https://docs.openshift.com/container-platform/3.11/getting_started/developers_console.html

# Simple application lifecycle

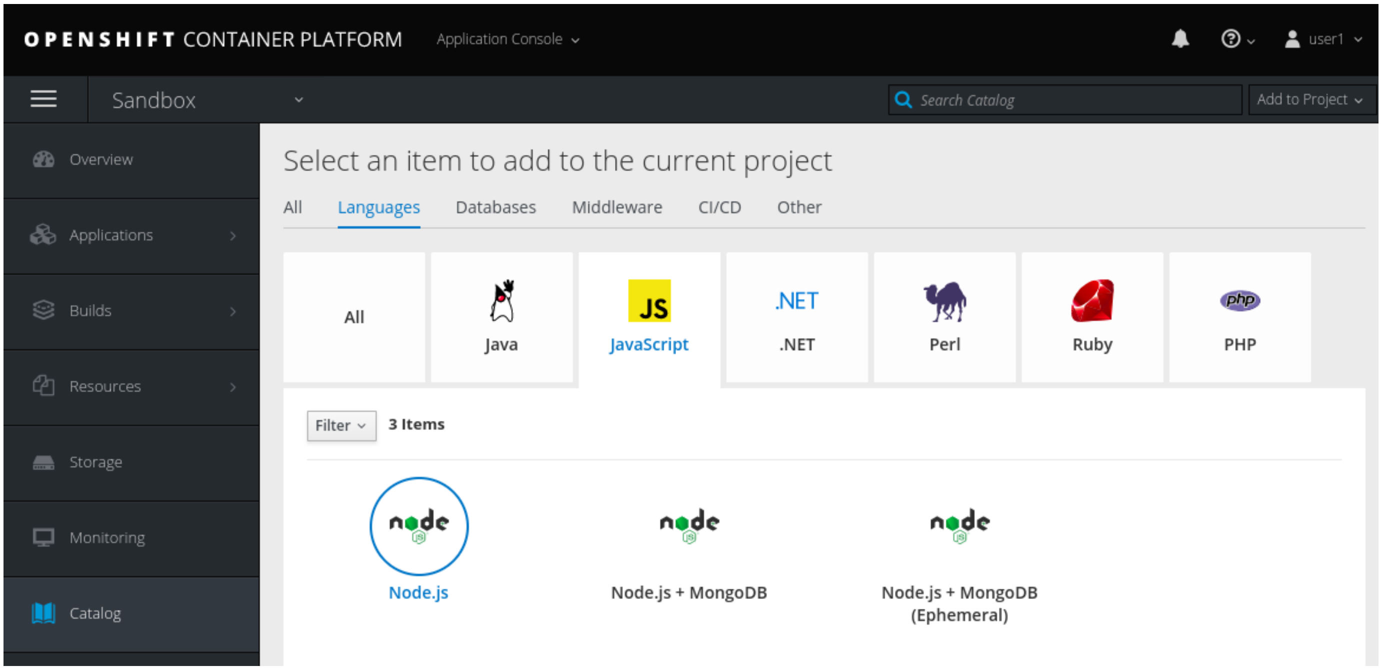
**TODO**

Describe the scripts within an S2I build (compile, assemble, execute)

## Deploying an application using S2I

In the Catalog of your sandbox project:

- Filter (top tab bar) **Languages**
- Select **JavaScript**
- Choose **Node.js**



You will now go though a wizard to gather all data needed for this S2I build config:

**TODO**

Explain the nature of the wizards

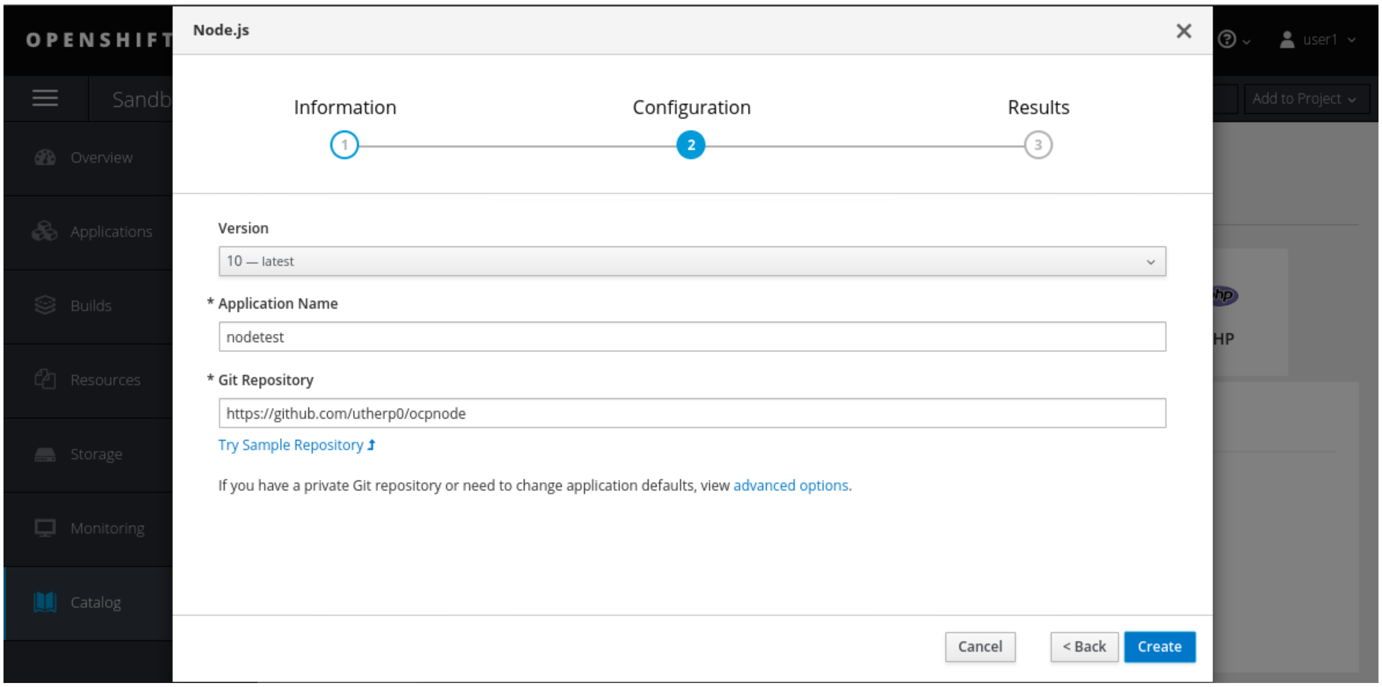
- Click **[ Next ]** on first page of Wizard.

**TODO**

Explain the parameter fields as pertains to templates

- Select node image version **10**
- Enter name as `nodetest`
- Enter the following url as the github repo https://github.com/utherp0/ocpnode

Click **[ Create ]**, then **[ Close ]** to close the wizard



Node.js

Information — Configuration — Results

Version: 10 — latest

* Application Name: nodetest

* Git Repository: https://github.com/utherp0/ocpnode

Try Sample Repository

If you have a private Git repository or need to change application defaults, view advanced options.

Now go back to the **Overview** page.

**TODO**

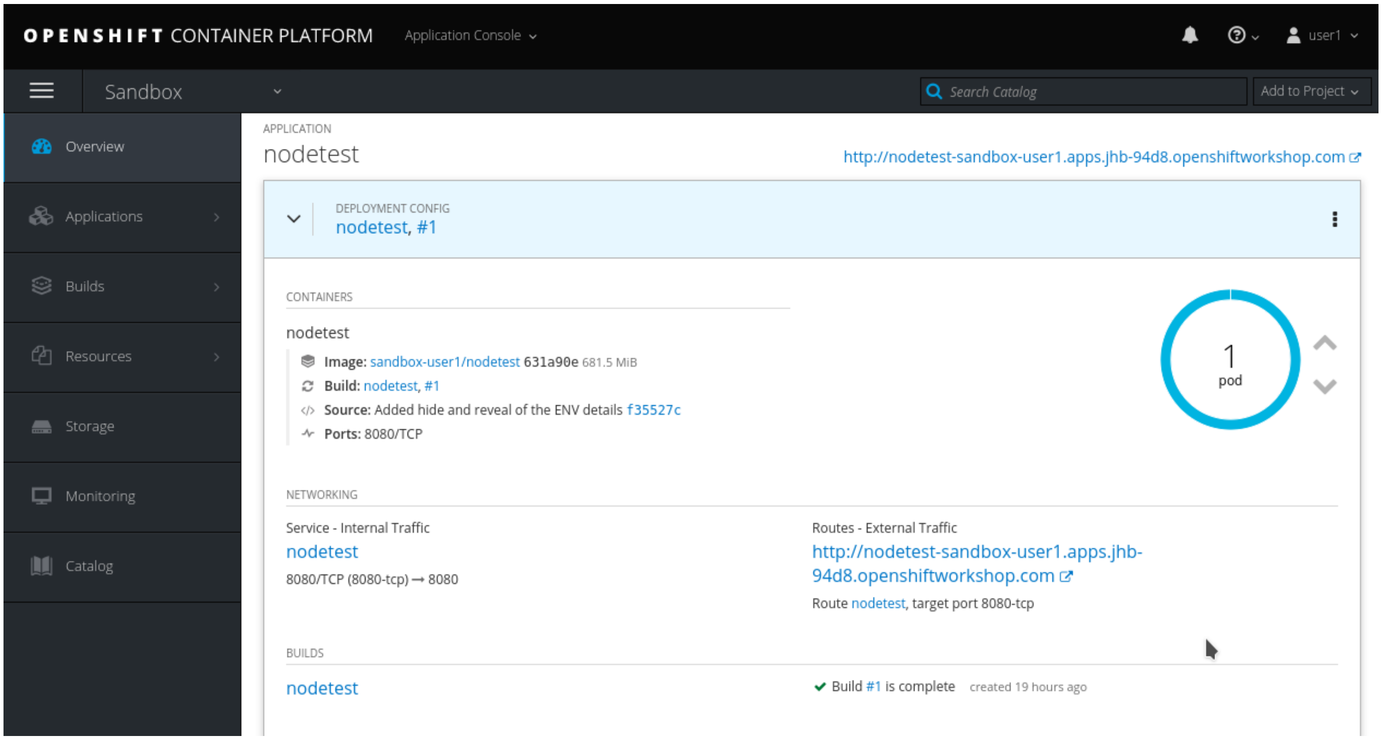
Explain the behaviour that is happening, the creation of the objects from the template, the build-config being used by the build, the build delivering the image into the registry, the deployment-config waiting on the image arrival, the default single Pod deployment, the creation of the route

# The running application

When in the Overview page, you will see all running applications. Expand the `nodetest` application we just deployed. You will see an overview of the running application:

- Information on the running container

- Number of pods and the status (a.k.a `Pod ring`) of the pods
- Networking information including internal port mapping and external routes
- Build history and information



APPLICATION
nodetest
http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

DEPLOYMENT CONFIG
nodetest, #1

CONTAINERS
nodetest
Image: sandbox-user1/nodetest 631a90e 681.5 MiB
Build: nodetest, #1
Source: Added hide and reveal of the ENV details f35527c
Ports: 8080/TCP

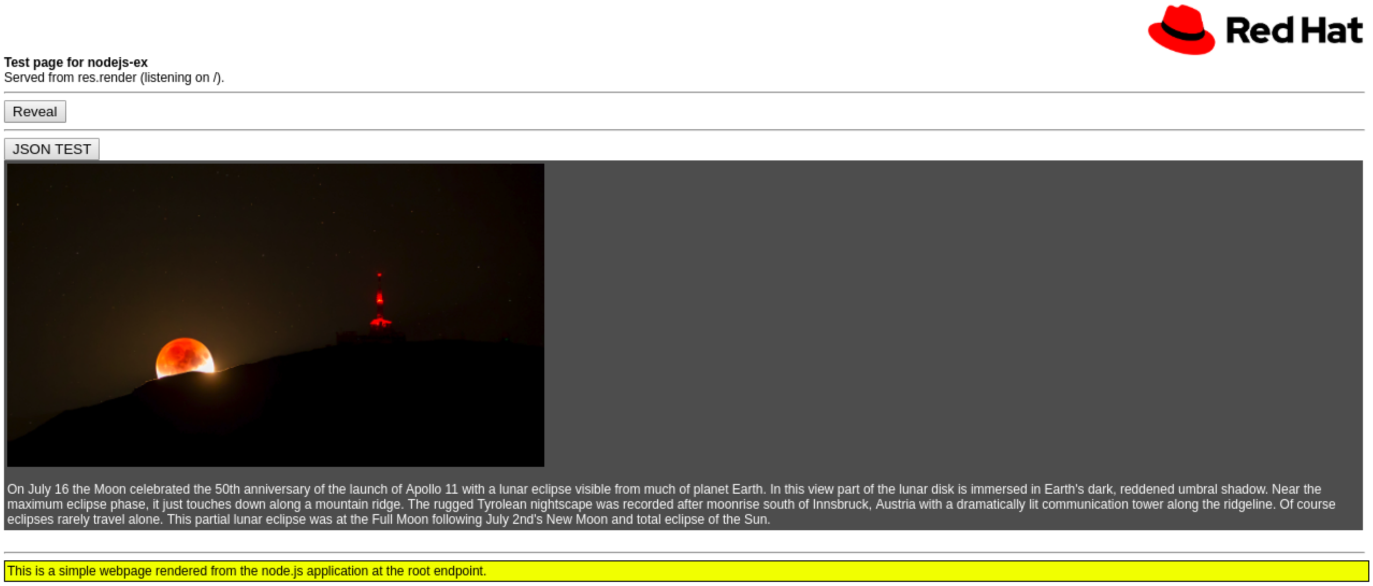
1 pod

NETWORKING
Service - Internal Traffic
nodetest
8080/TCP (8080-tcp) → 8080

Routes - External Traffic
http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com
Route nodetest, target port 8080-tcp

BUILDS
nodetest
✓ Build #1 is complete created 19 hours ago

To see the application in action, click on the link in the external route. This will open the basic node.js application:



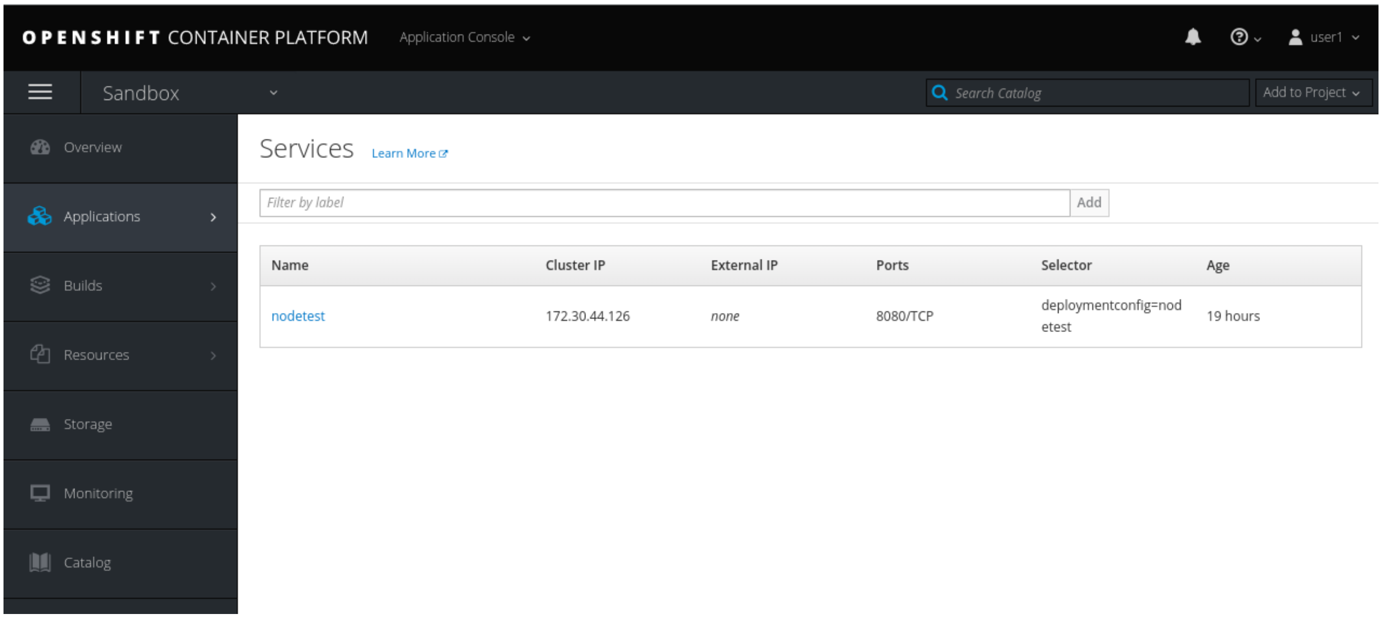
**Test page for nodejs-ex**
Served from res.render (listening on /).

On July 16 the Moon celebrated the 50th anniversary of the launch of Apollo 11 with a lunar eclipse visible from much of planet Earth. In this view part of the lunar disk is immersed in Earth's dark, reddened umbral shadow. Near the maximum eclipse phase, it just touches down along a mountain ridge. The rugged Tyrolean nightscape was recorded after moonrise south of Innsbruck, Austria with a dramatically lit communication tower along the ridgeline. Of course eclipses rarely travel alone. This partial lunar eclipse was at the Full Moon following July 2nd's New Moon and total eclipse of the Sun.

This is a simple webpage rendered from the node.js application at the root endpoint.

## Application Services

Using the menu on the left go to the **Applications › Services** page.



Services Learn More

Filter by label Add

| Name | Cluster IP | External IP | Ports | Selector | Age |
|---|---|---|---|---|---|
| nodetest | 172.30.44.126 | *none* | 8080/TCP | deploymentconfig=nodetest | 19 hours |

**TODO**

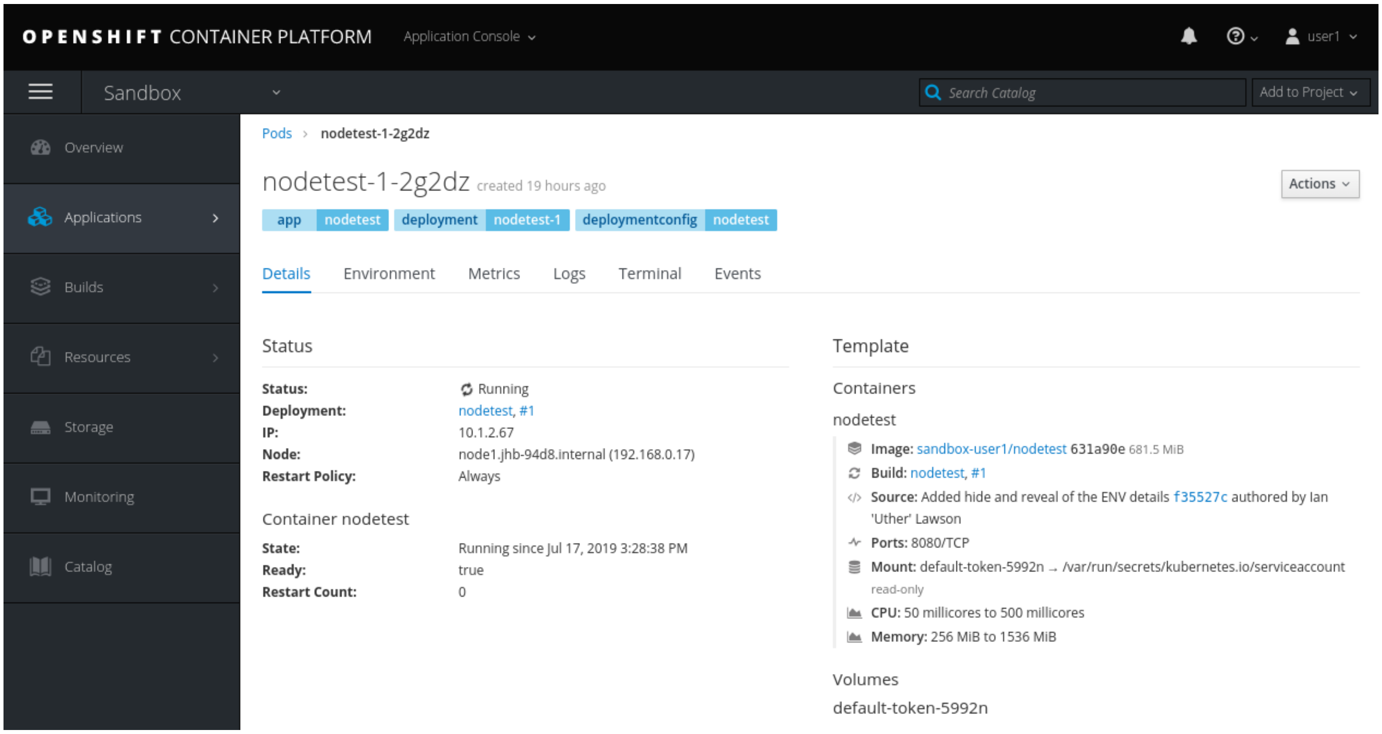
Explain the nature of the single service endpoint for the application - note the cluster IP address

More info here: https://docs.openshift.com/container-platform/3.11/architecture/core_concepts/pods_and_services.html#services

Go back to the **Overview** page.

## Application Pods

Click on the `Pod ring`, or alternatively use the menu **Applications › Pods › nodetest-****



nodetest-1-2g2dz created 19 hours ago

Status

**Status:** Running
**Deployment:** nodetest, #1
**IP:** 10.1.2.67
**Node:** node1.jhb-94d8.internal (192.168.0.17)
**Restart Policy:** Always

Container nodetest

**State:** Running since Jul 17, 2019 3:28:38 PM
**Ready:** true
**Restart Count:** 0

Template

Containers

nodetest

**Image:** sandbox-user1/nodetest 631a90e 681.5 MiB
**Build:** nodetest, #1
**Source:** Added hide and reveal of the ENV details f35527c authored by Ian 'Uther' Lawson
**Ports:** 8080/TCP
**Mount:** default-token-5992n → /var/run/secrets/kubernetes.io/serviceaccount read-only
**CPU:** 50 millicores to 500 millicores
**Memory:** 256 MiB to 1536 MiB

Volumes

default-token-5992n

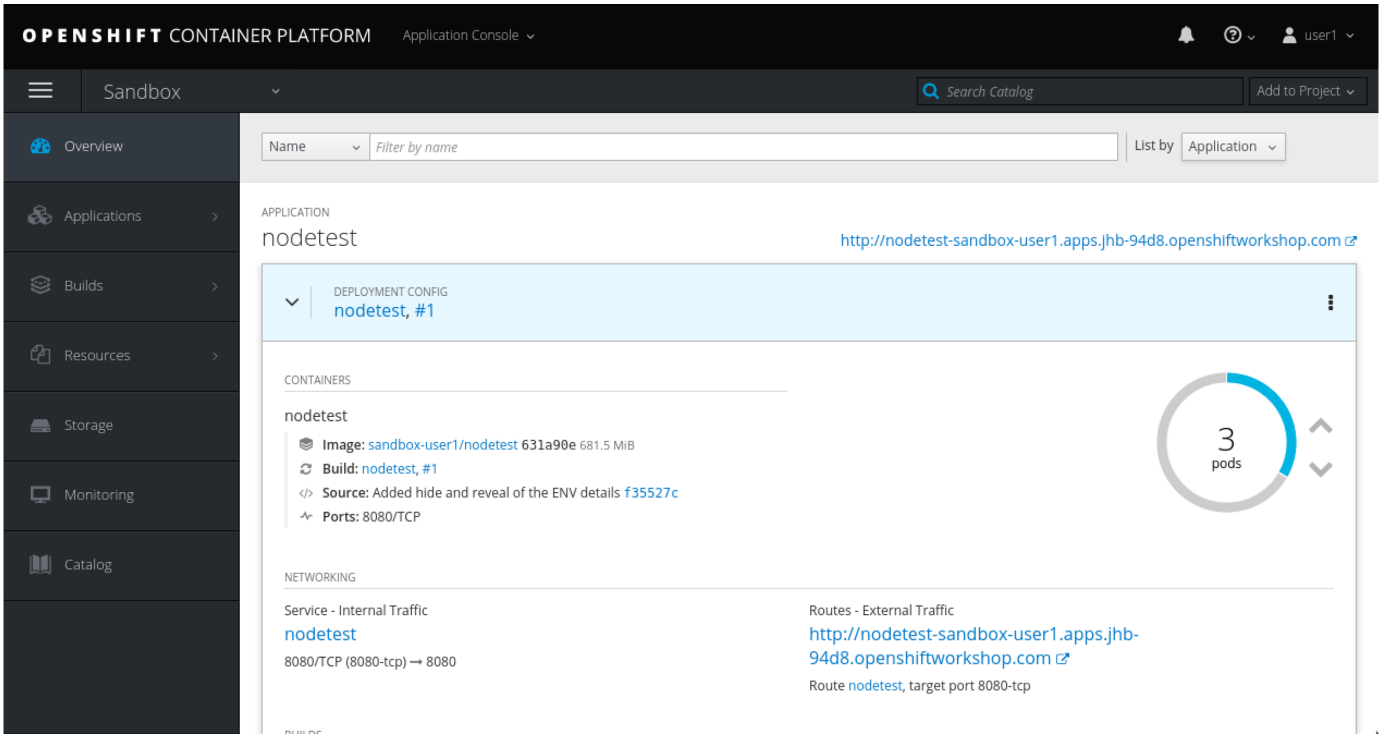
See the differing IP address for the Pod compared to the cluster IP

Go back to the **Overview** page.

## Application Scaling

Let’s pretend that this app is suddenly getting many requests from many users (so there is a load increase on the app). So we need to scale the application to 3 instances.

Click the **Up arrow** (^) until there are 3 replicas.



Name: Filter by name — List by: Application

APPLICATION
nodetest
http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

DEPLOYMENT CONFIG
nodetest, #1

CONTAINERS

nodetest

- **Image:** sandbox-user1/nodetest `631a90e` 681.5 MiB
- **Build:** nodetest, #1
- **Source:** Added hide and reveal of the ENV details `f35527c`
- **Ports:** 8080/TCP

NETWORKING

Service - Internal Traffic
nodetest
8080/TCP (8080-tcp) → 8080

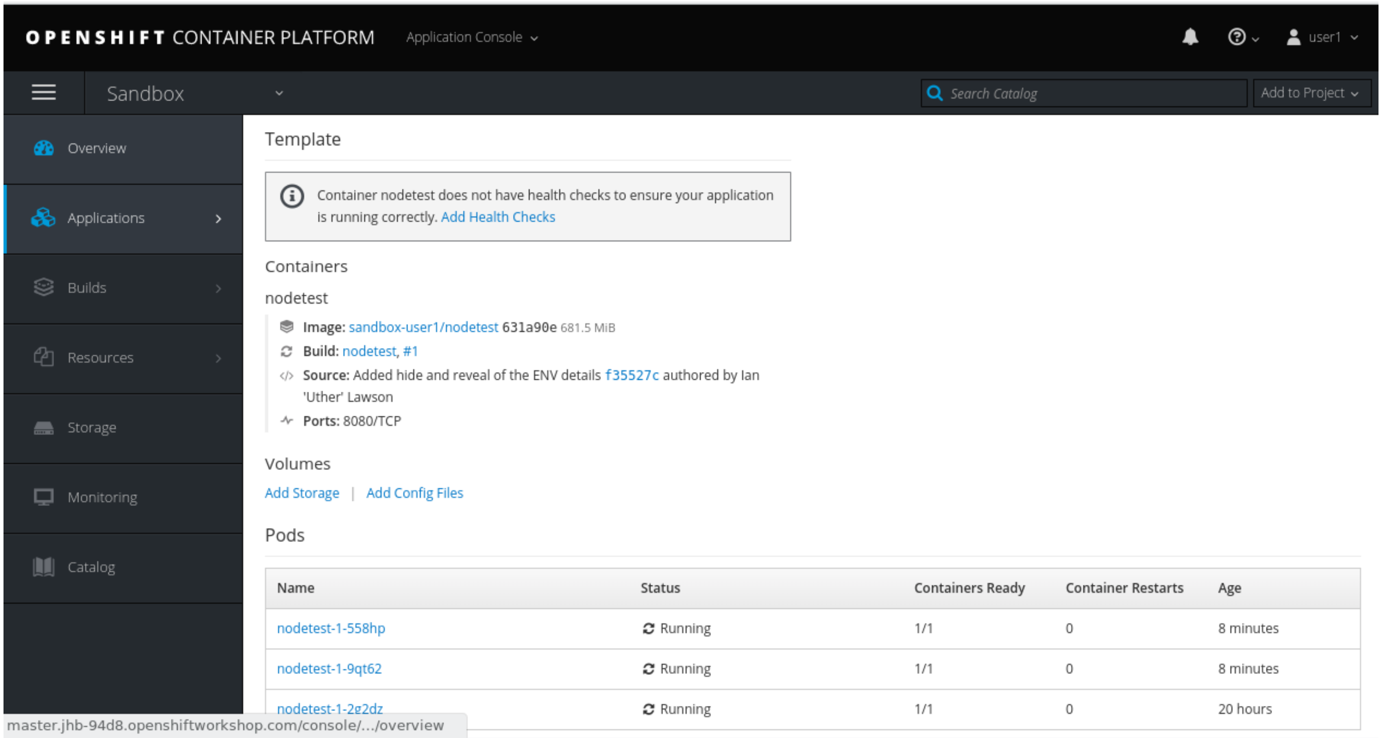
Routes - External Traffic
http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com
Route nodetest, target port 8080-tcp

**TODO**

Explain the blue, gray, dark blue, red colour schemes for the Pod behaviors

Click on the `Pod ring`, or alternatively use the menu **Applications › Deployments › nodetest › #1 (latest)**.

Scroll down to where the Pods are listed:



Template

Container nodetest does not have health checks to ensure your application is running correctly. Add Health Checks

Containers

nodetest

- Image: sandbox-user1/nodetest 631a90e 681.5 MiB
- Build: nodetest, #1
- Source: Added hide and reveal of the ENV details f35527c authored by Ian 'Uther' Lawson
- Ports: 8080/TCP

Volumes

Add Storage | Add Config Files

Pods

| Name | Status | Containers Ready | Container Restarts | Age |
|---|---|---|---|---|
| nodetest-1-558hp | Running | 1/1 | 0 | 8 minutes |
| nodetest-1-9qt62 | Running | 1/1 | 0 | 8 minutes |
| nodetest-1-2g2dz | Running | 1/1 | 0 | 20 hours |

See the difference in age between the initial pod and the 2 recent scaled pods.

Select on of the recent (younger) pods.

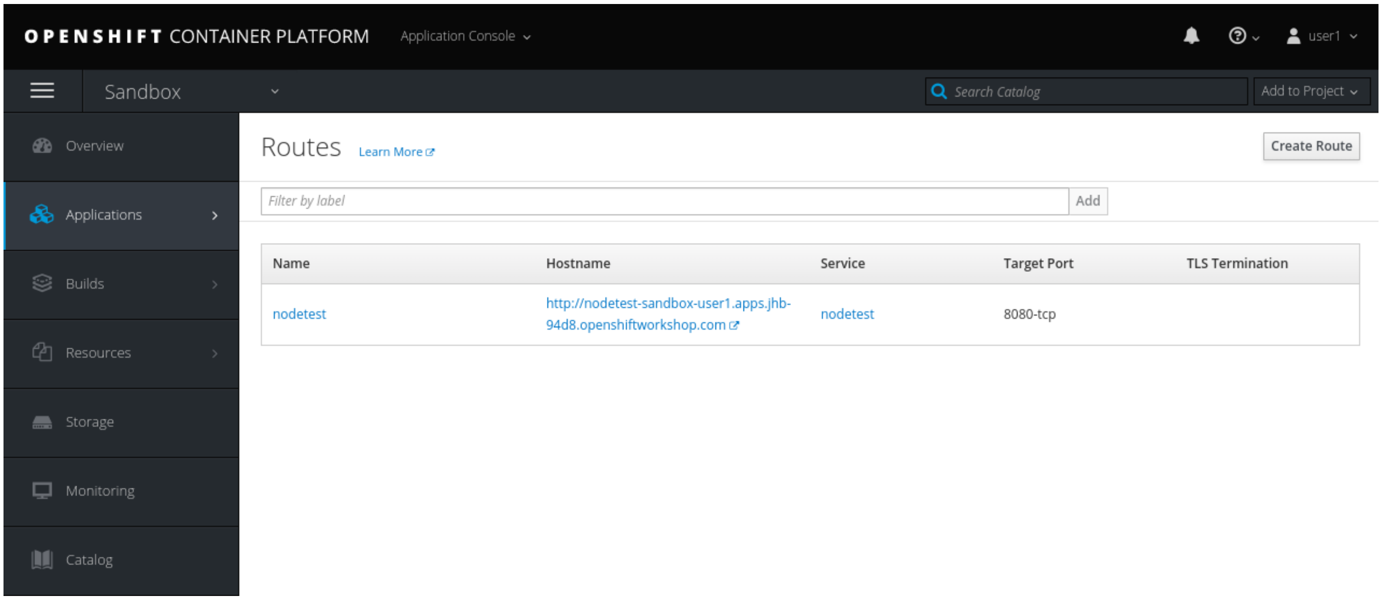
Note the IP difference compared to the initial pod.

**TODO**

Explain the load-balancing of Pod IP endpoints from the singular cluster IP and how that abstracts from the Route.

## Application Route

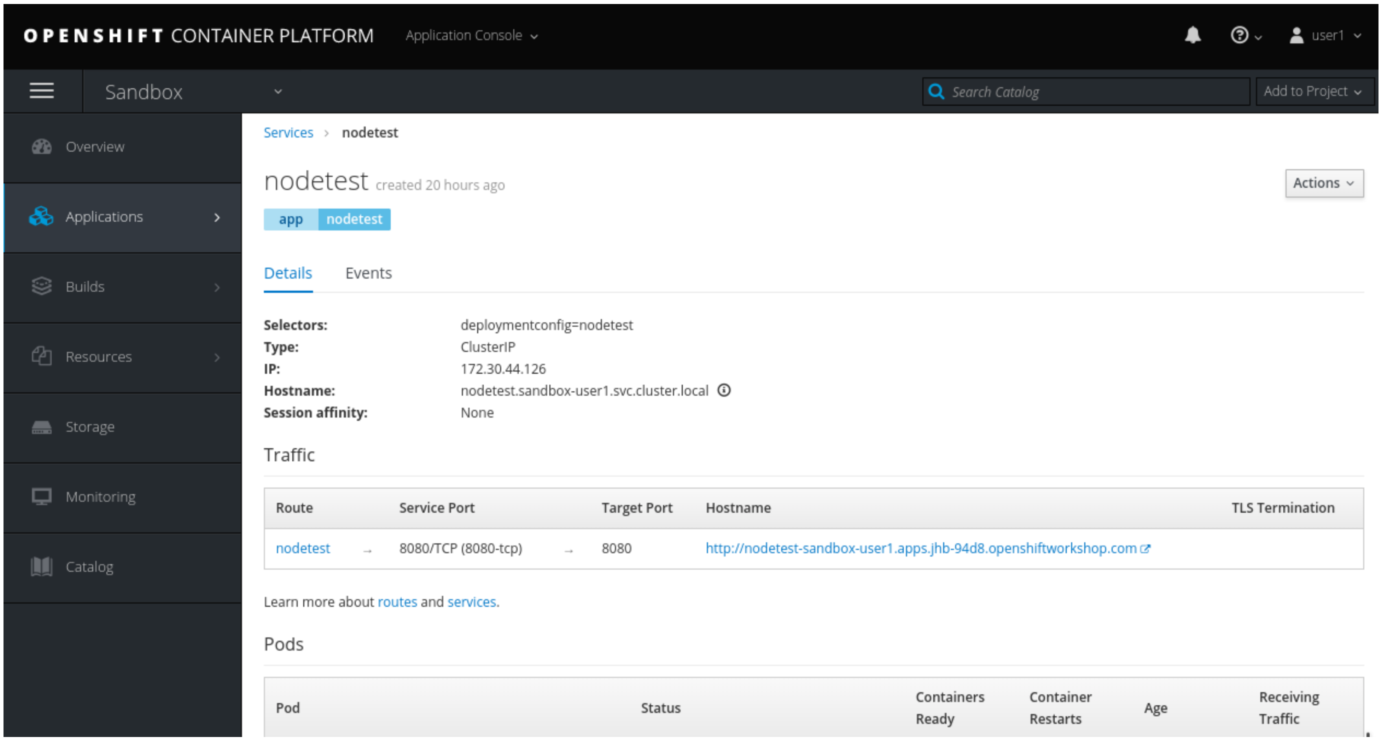
Using the menu on the left go to the **Applications › Routes** page.



| Name | Hostname | Service | Target Port | TLS Termination |
|---|---|---|---|---|
| nodetest | http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com | nodetest | 8080-tcp | |

Note the mapping of the fully qualified domain name to the cluster IP via the service name

Select the nodetest link in the service column.



**Selectors:** deploymentconfig=nodetest
**Type:** ClusterIP
**IP:** 172.30.44.126
**Hostname:** nodetest.sandbox-user1.svc.cluster.local
**Session affinity:** None

Traffic

| Route | | Service Port | | Target Port | Hostname | TLS Termination |
|---|---|---|---|---|---|---|
| nodetest | → | 8080/TCP (8080-tcp) | → | 8080 | http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com | |

Learn more about routes and services.

Pods

| Pod | Status | Containers Ready | Container Restarts | Age | Receiving Traffic |
|---|---|---|---|---|---|

Note that the route maps to the cluster IP

## Application from CLI

Now let’s go to the console (either using `localhost` or `oconline` as explained in the CLI (oc) section)

Make sure you are still logged in:

```
oc whoami
```

(if not, log in again as explained in the Using your terminal (Both options) section)

Make sure we are using our sandbox project:

```
oc project sandbox-userX
```

This will print:

```
Now using project "sandbox-userX" on server "http://myopenshift.com:443".
```

You can find all `objects` that you can interact with in this namespace/project:

```
oc get all
```

Get all `pods`:

```
oc get pods -o wide
```

This will output something similar to this:

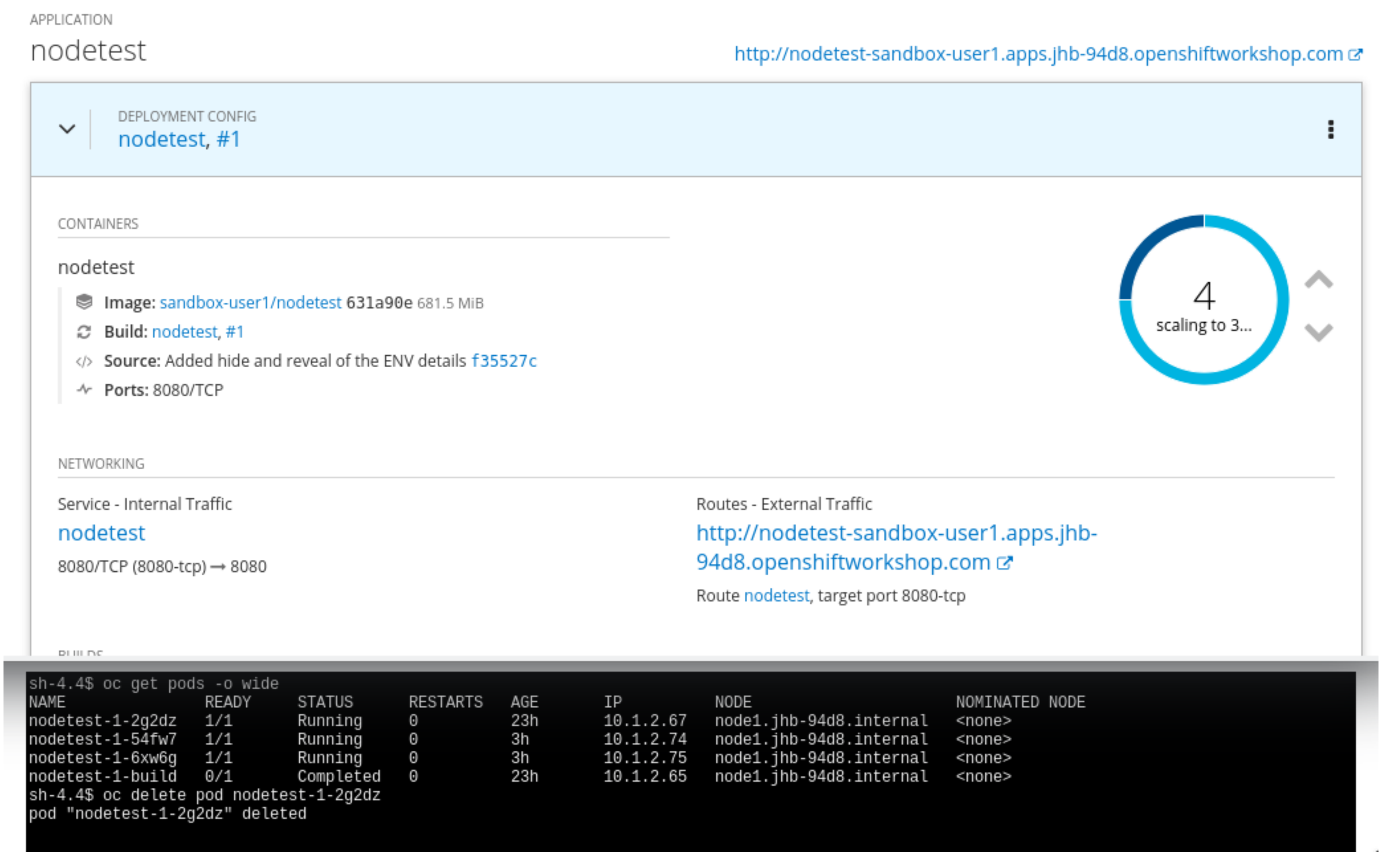
```
NAME                READY     STATUS      RESTARTS   AGE       IP          NODE
NOMINATED NODE
nodetest-1-2g2dz    1/1       Running     0          23h       10.1.2.67   node1.jhb-
94d8.internal    <none>
nodetest-1-54fw7    1/1       Running     0          3h        10.1.2.74   node1.jhb-
94d8.internal    <none>
nodetest-1-6xw6g    1/1       Running     0          3h        10.1.2.75   node1.jhb-
94d8.internal    <none>
nodetest-1-build    0/1       Completed   0          23h       10.1.2.65   node1.jhb-
94d8.internal    <none>
```

Note the pod used to build the project is there, just inactive.
Also note the differing IPs for the individual Pods and the NODE information.

In the Web Console, make sure you are on the **[ Overview ]** page, then do the following in CLI while watching the page:

```
oc delete pod nodetest-****
```

(Replace ** with once of the running pods)



APPLICATION

nodetest

http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

DEPLOYMENT CONFIG

nodetest, #1

CONTAINERS

nodetest

- **Image:** sandbox-user1/nodetest 631a90e 681.5 MiB
- **Build:** nodetest, #1
- **Source:** Added hide and reveal of the ENV details f35527c
- **Ports:** 8080/TCP

NETWORKING

Service - Internal Traffic

nodetest

8080/TCP (8080-tcp) → 8080

Routes - External Traffic

http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

Route nodetest, target port 8080-tcp

```
sh-4.4$ oc get pods -o wide
NAME               READY    STATUS      RESTARTS   AGE      IP          NODE                     NOMINATED NODE
nodetest-1-2g2dz   1/1      Running     0          23h      10.1.2.67   node1.jhb-94d8.internal   <none>
nodetest-1-54fw7   1/1      Running     0          3h       10.1.2.74   node1.jhb-94d8.internal   <none>
nodetest-1-6xw6g   1/1      Running     0          3h       10.1.2.75   node1.jhb-94d8.internal   <none>
nodetest-1-build   0/1      Completed   0          23h      10.1.2.65   node1.jhb-94d8.internal   <none>
sh-4.4$ oc delete pod nodetest-1-2g2dz
pod "nodetest-1-2g2dz" deleted
```

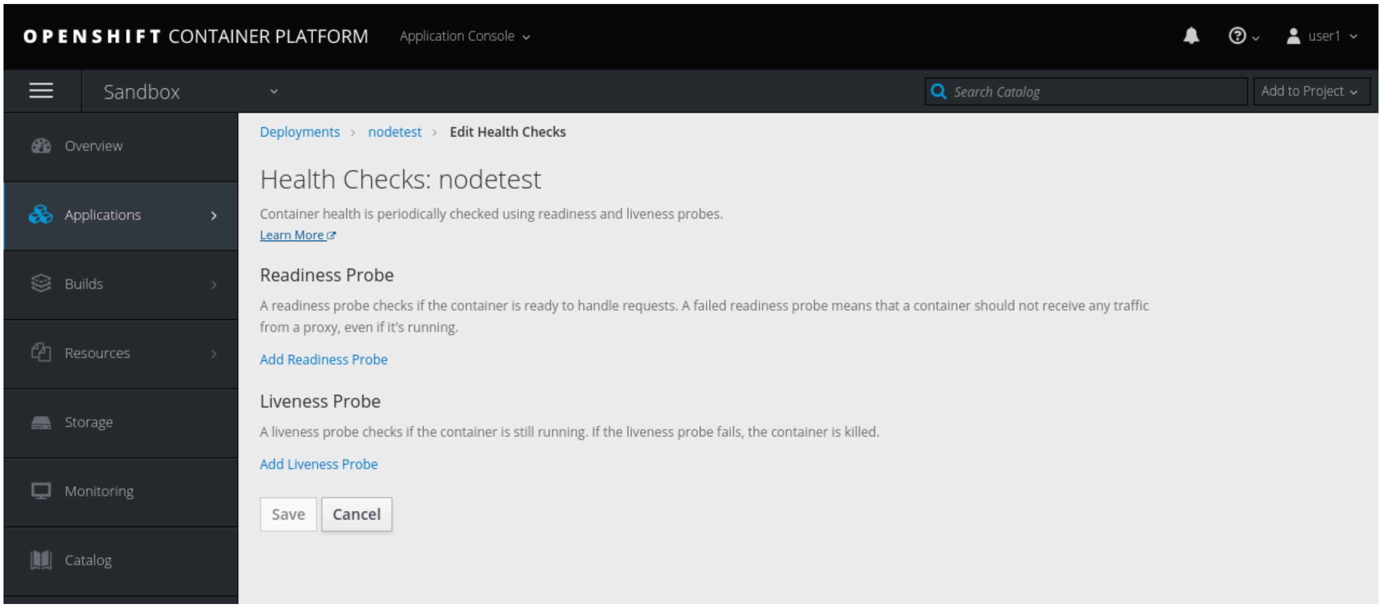
**TODO**

Explain the nature of Liveness (kill/restart) and Readiness (if not ready Pod IP is removed from the round-robin HAProxy)

## Health Checks

In the Web Console, go to **Applications › Deployments › nodetest › Configuration**.

Under Template, click `Add Health Checks`:



Health Checks: nodetest

Container health is periodically checked using readiness and liveness probes.

Readiness Probe

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

Liveness Probe

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

TIP: Click on the `Learn More` link or here: https://docs.openshift.com/container-platform/3.11/dev_guide/application_health.html to read more about Health probes

**TODO**

Explain the concepts of the readiness and health probes

## Application Deployment Strategies

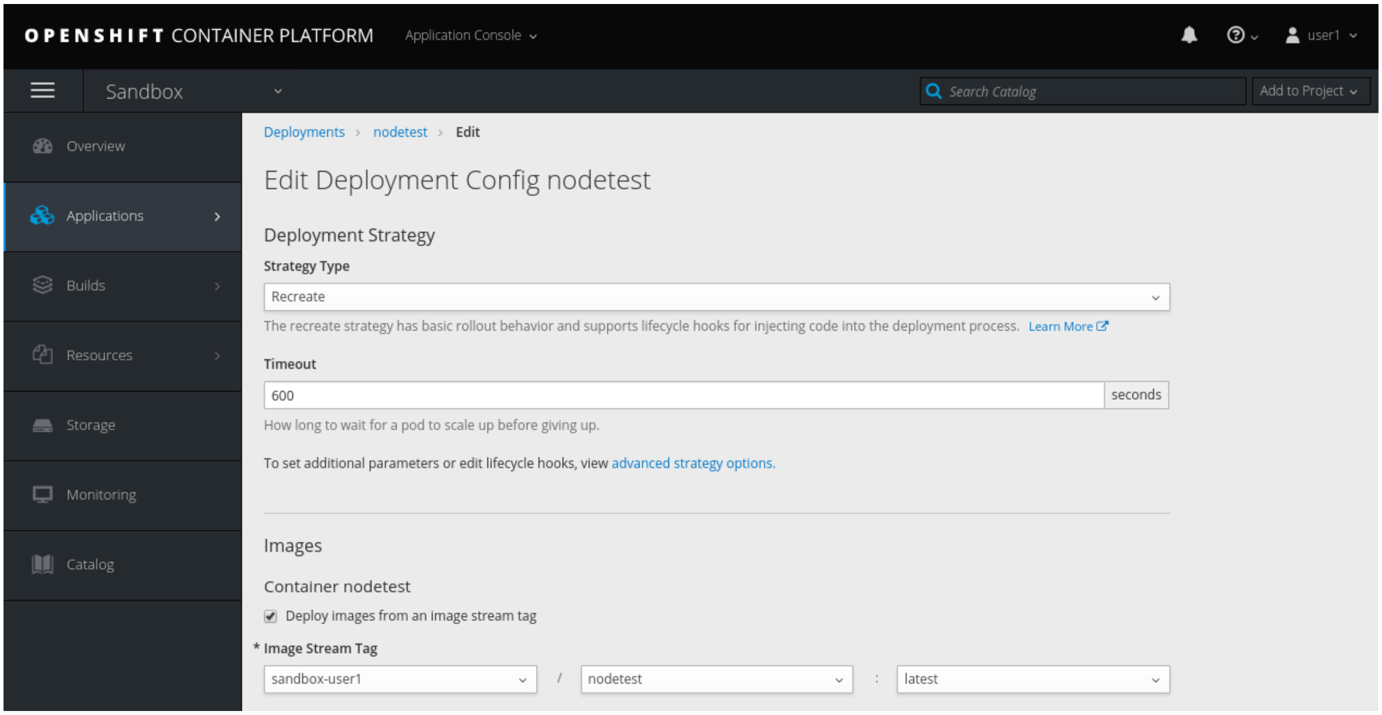
From the menu: **Applications › Deployment › nodetest › Configuration**

**TODO**

Explain rolling and recreate - explain deployment triggers (image and config)

In the top right corner, click the **[ Actions > Edit ]** button.

Change the **[ Strategy Type ]** to `Recreate` and click **[ Save ]**



Deployments › nodetest › Edit

Edit Deployment Config nodetest

Deployment Strategy

Strategy Type: Recreate

The recreate strategy has basic rollout behavior and supports lifecycle hooks for injecting code into the deployment process. Learn More

Timeout: 600 seconds

How long to wait for a pod to scale up before giving up.

To set additional parameters or edit lifecycle hooks, view advanced strategy options.

Images

Container nodetest

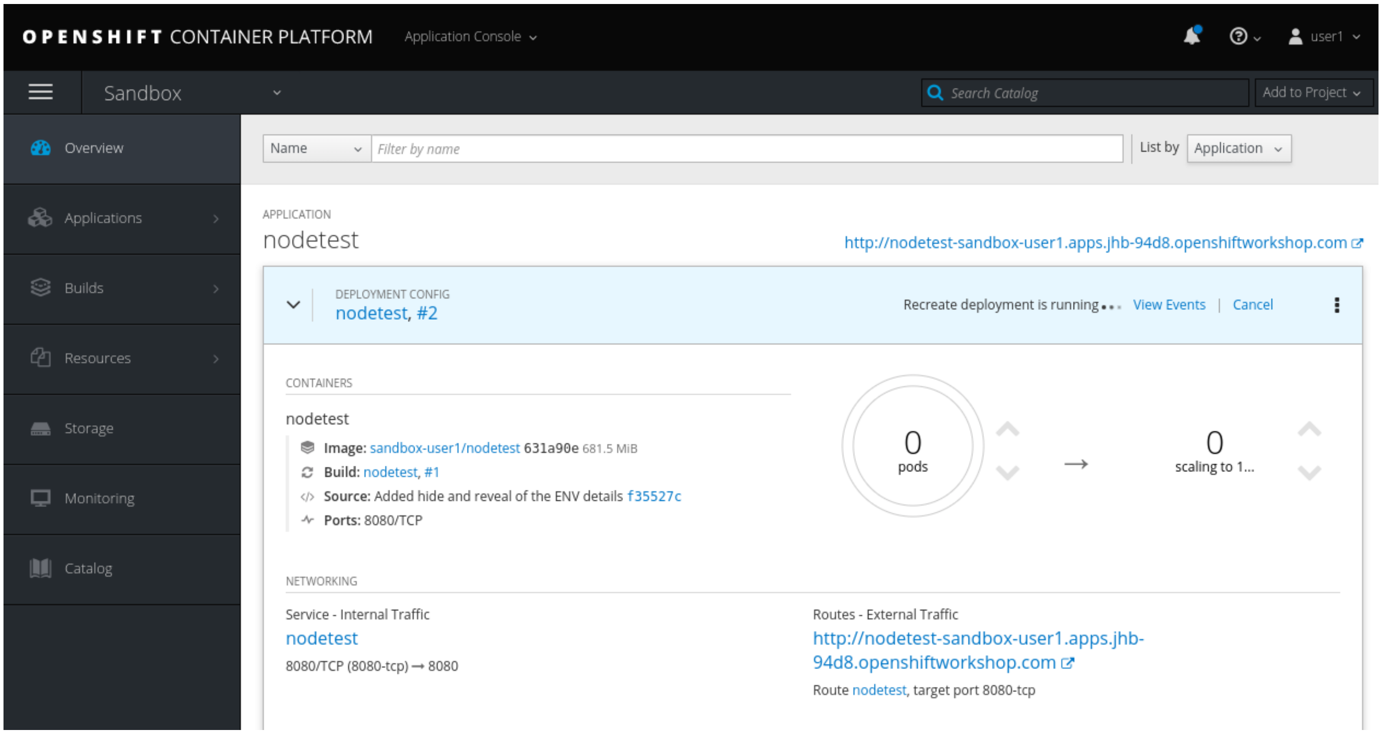
☑ Deploy images from an image stream tag

* Image Stream Tag: sandbox-user1 / nodetest : latest

Now go to **Applications › Deployments › notetest**

Note that Deployment \#1 is active.

Click the **[ Deploy ]** button (top right) and the quickly go back to the **Overview** page.



APPLICATION

nodetest

http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

DEPLOYMENT CONFIG

nodetest, #2

Recreate deployment is running... View Events | Cancel

CONTAINERS

nodetest

- Image: sandbox-user1/nodetest 631a90e 681.5 MiB
- Build: nodetest, #1
- Source: Added hide and reveal of the ENV details f35527c
- Ports: 8080/TCP

0 pods → 0 scaling to 1...

NETWORKING

Service - Internal Traffic

nodetest

8080/TCP (8080-tcp) → 8080

Routes - External Traffic

http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

Route nodetest, target port 8080-tcp

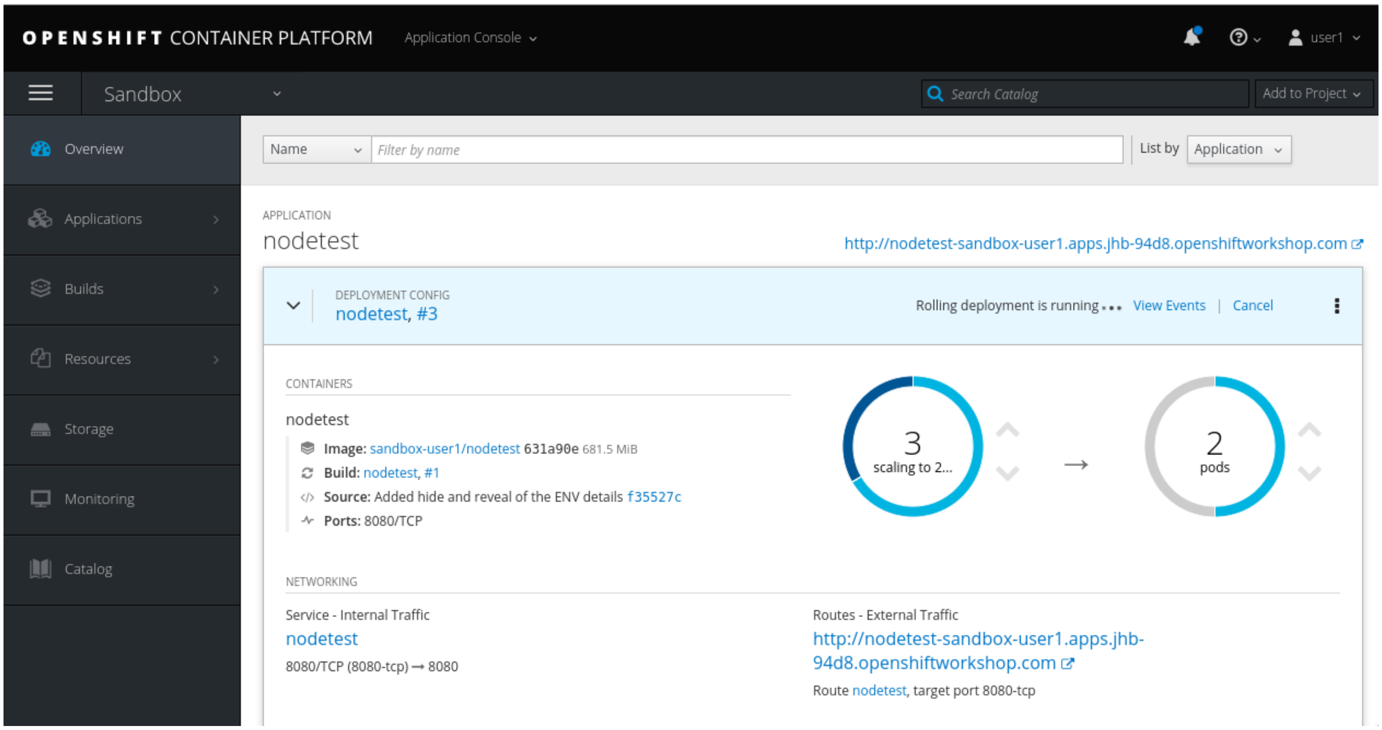
Note that all instances is being recreate and there is zero instances available above.

Go back to **Applications › Deployments › notetest**

Note that Deployment \#2 is active.

Change back to Rolling Strategy: **[ Actions** > **Edit ]** then change the **[ Strategy Type ]** to `Rolling` and click **[ Save ]**

Now again click the **[ Deploy ]** and quickly go back to the **Overview** page.



APPLICATION

nodetest

http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

DEPLOYMENT CONFIG

nodetest, #3

Rolling deployment is running... View Events | Cancel

CONTAINERS

nodetest

- **Image:** sandbox-user1/nodetest **631a90e** 681.5 MiB
- **Build:** nodetest, #1
- **Source:** Added hide and reveal of the ENV details f35527c
- **Ports:** 8080/TCP

NETWORKING

Service - Internal Traffic

nodetest

8080/TCP (8080-tcp) → 8080

Routes - External Traffic

http://nodetest-sandbox-user1.apps.jhb-94d8.openshiftworkshop.com

Route nodetest, target port 8080-tcp

Note that the number of available pods never drops beneath the required number of replicas

Read more about deployment strategies here: https://docs.openshift.com/container-platform/3.11/dev_guide/deployments/deployment_strategies.html

## Storage

Go to **Storage › ] page and select [ Create Storage** :

- **Name:** test
- **Access Mode:** RWO
- **Size:** 1 GiB

Click **[ Create ]**

**TODO**

Explain the nature of PVs, how they are exported to all nodes, how the Container Runtime maps them into the Container at deployment as an additional file system

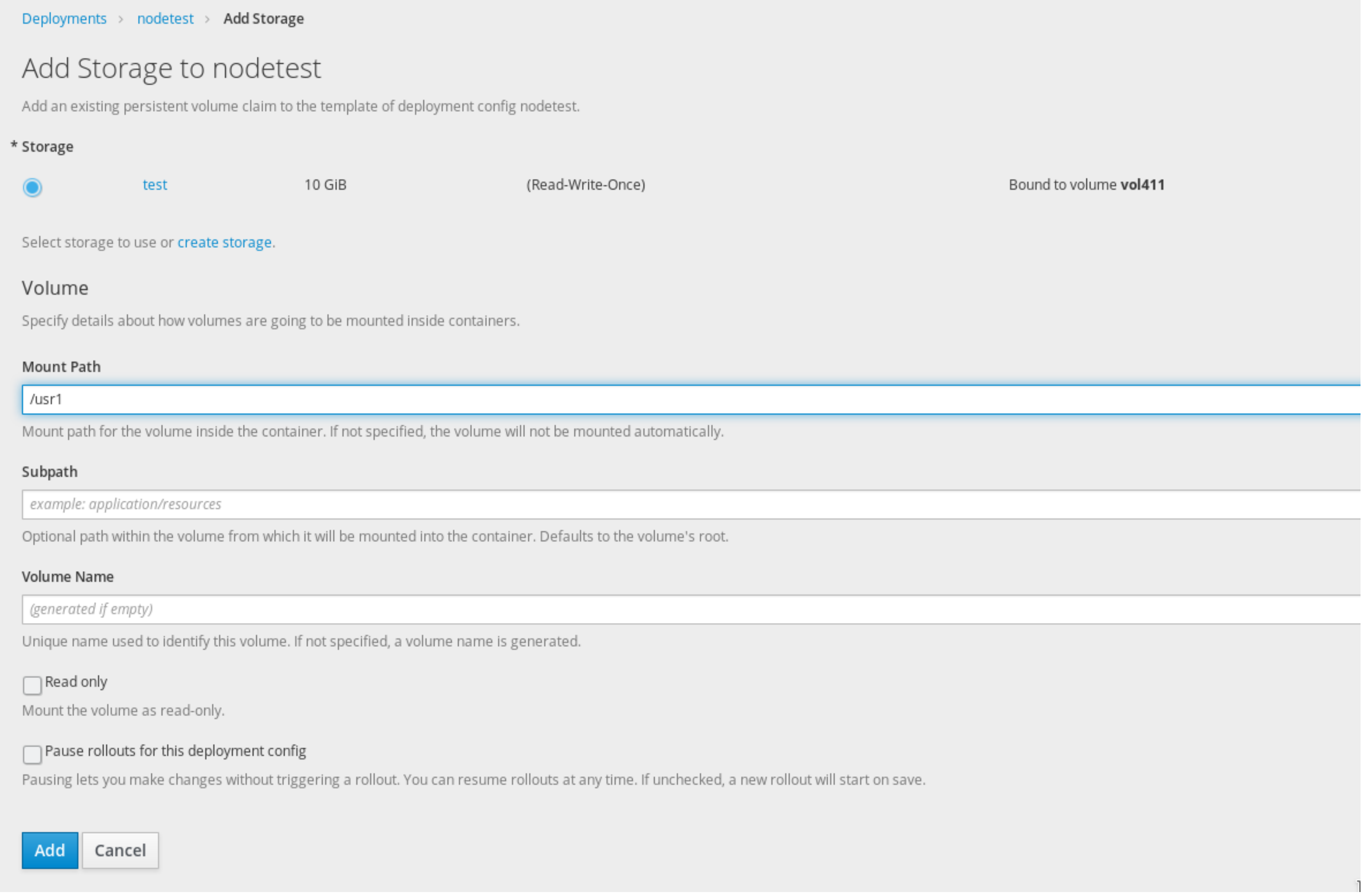
Now we will assign this storage to our application. Go to **Applications › Deployments** and select `nodetest` and select the `Configuration` tab. Under the Volumes section, click `Add Storage`

**TODO**

Explain the nature of a PVC which locks the storage into the container

Select the `test` storage option (This is the one we just created)

In the **Mount Path** make sure that the path is unique to you, so make it `/usrX` (Where X is your assigned ID). Click **[ Add ]**



Add Storage to nodetest

Add an existing persistent volume claim to the template of deployment config nodetest.

* Storage

● test 10 GiB (Read-Write-Once) Bound to volume **vol411**

Select storage to use or create storage.

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path: /usr1

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Subpath: *example: application/resources*

Optional path within the volume from which it will be mounted into the container. Defaults to the volume's root.

Volume Name: *(generated if empty)*

Unique name used to identify this volume. If not specified, a volume name is generated.

□ Read only

Mount the volume as read-only.

□ Pause rollouts for this deployment config

Pausing lets you make changes without triggering a rollout. You can resume rollouts at any time. If unchecked, a new rollout will start on save.

Go back to the **Overview** page.

Note the redeployment, this is because above is a config change and a new image needs to be build to make this mount point available

Click on the `Pod ring` and select the first (top) pod in the `Pods` section. Select the `Terminal` tab and then type the following:

```
id
```

This will print the unique id for this pod, example:

```
uid=1000360000 gid=0(root) groups=0(root),1000360000
```

Now type the following in the terminal:

```
df -h
```

This will report information on the disk space for this pod, example:

```
Filesystem                                                  Size  Used Avail Use% Mounted
on
overlay                                                      50G  7.3G   43G  15% /
tmpfs                                                        32G     0   32G   0% /dev
tmpfs                                                        32G     0   32G   0%
/sys/fs/cgroup
support1.fourways-3631.internal:/srv/nfs/user-vols/vol411  197G  498M  187G   1% /usr1
/dev/xvda2                                                   50G  7.3G   43G  15%
/etc/hosts
shm                                                          64M     0   64M   0% /dev/shm
tmpfs                                                        32G   16K   32G   1%
/run/secrets/kubernetes.io/serviceaccount
tmpfs                                                        32G     0   32G   0%
/proc/acpi
tmpfs                                                        32G     0   32G   0%
/proc/scsi
tmpfs                                                        32G     0   32G   0%
/sys/firmware
```

You will see the volume create earlier mounted under `/usrX`.

Type the following:

```
ps -ef
```

**TODO**

Explain the ‘it thinks it is an OS’ concept, explain SELinux constraints around the ‘ps -ef’, note the addition of the new disk at the mount point provided at the PV creation

Now let’s go to the volume mount point and create a file in the root:

```
cd /usrX
touch test.txt
```

List the contents of the folder:

```
ls -alZ
```

You will see a list of directories and files, example:

```
drwxrwxrwx. root       root  system_u:object_r:nfs_t:s0                    .
drwxr-xr-x. root       root  system_u:object_r:container_file_t:s0:c9,c19  ..
-rw-r--r--. 1000360000 65534 system_u:object_r:nfs_t:s0                    test1.txt
```

**TODO**

Explain the selinux constraints

Now, in the CLI (NOT the terminal we have been using just now), do the following:

```
oc get pods -o wide
```

You will see a list of all pods, example:

```
NAME                READY   STATUS      RESTARTS   AGE   IP          NODE
NOMINATED NODE
nodetest-1-build    0/1     Completed   0          1h    10.1.4.8    node1.jhb-
94d8.internal   <none>
nodetest-2-5lcdq    1/1     Running     0          15m   10.1.4.12   node1.jhb-
94d8.internal   <none>
nodetest-2-7dnjv    1/1     Running     0          12m   10.1.4.14   node1.jhb-
94d8.internal   <none>
nodetest-2-nfnlf    1/1     Running     0          12m   10.1.4.13   node1.jhb-
94d8.internal   <none>
```

**TODO**

Find two Pods on physically separate Nodes - take note of the Pod names - explain the format, name-(x)-(randomchars)

Go back to the **Overview** page.

Click on the `Pod ring` and select the first (top) pod in the `Pods` section. Select the `Terminal` tab and then type the following:

```
cd /usrX
vi test.txt
```

Once in `vi`, press **i** to enter **insert** mode.

Now type something, example: `Hello World`.

Then press **Esc** (to exit the insert mode) and then **:wq** to write and quit vi. You can do a `cat` to make sure the contents is saved in the file:

```
cat test.txt
Hello world
```

Now go back to the **Overview** page.

Click on the `Pod ring` and select any pod except the first (top) one in the `Pods` section. Select the `Terminal` tab and then type the following:

```
cd /usrX
cat test.txt
Hello world
```

As you can see the file is available on all pods.

**TODO**

Explain the nature of the single file in persisted storage across multiple physical nodes

## Config Maps

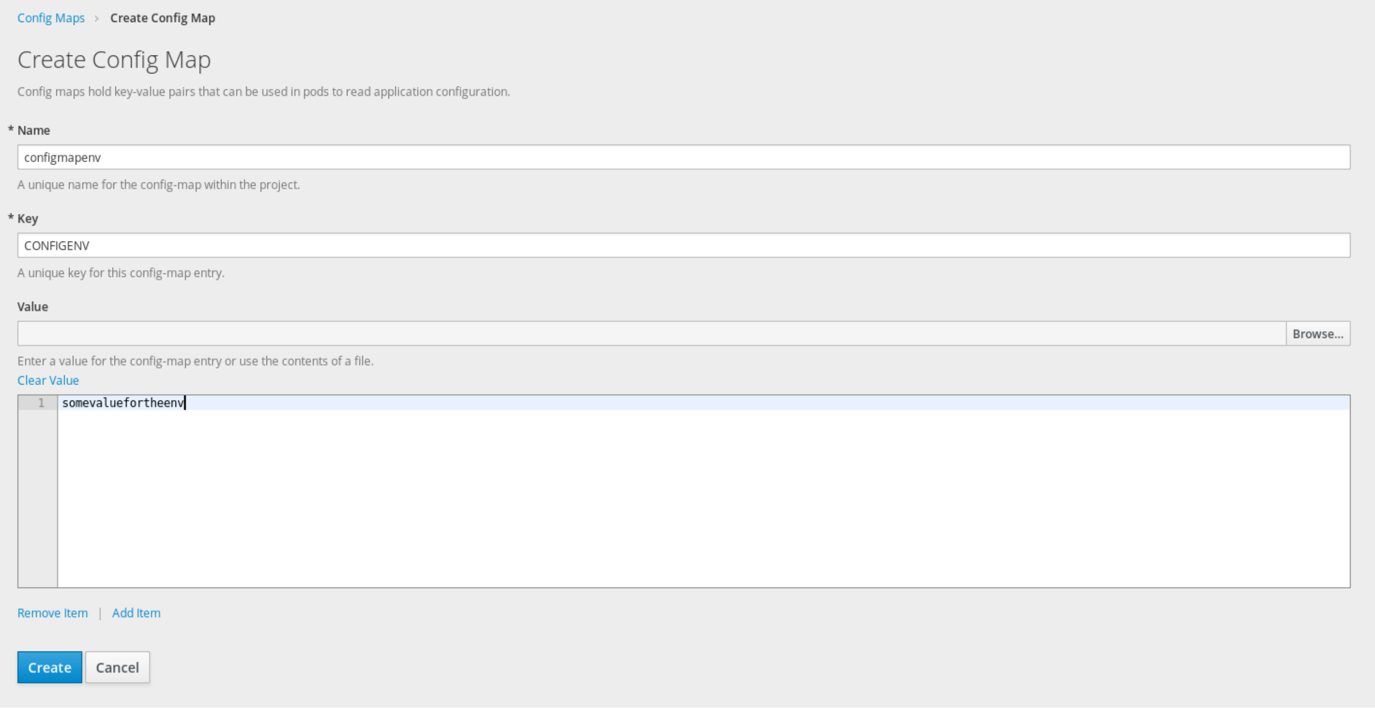
Navigate to **Resources › Config Maps** and then click **[ Create Config Map ]**

**TODO**

Discuss the nature of config maps as environment vars

Enter the following in the fields: * **Name:** configmapenv * **Key:** CONFIGENV * **Value:** somevaluefortheenv

Then click **[ Create ]**



Create Config Map

Config maps hold key-value pairs that can be used in pods to read application configuration.

* Name: configmapenv

A unique name for the config-map within the project.

* Key: CONFIGENV

A unique key for this config-map entry.

Value

Enter a value for the config-map entry or use the contents of a file.

```
somevaluefortheenv
```

Navigate to **Applications › Deployments** select `nodetest` and then the `Environment` Tab.

In the `Environment From` section, select the `configmapenv` we just created.

Click the `Add ALL Values from Config Map or Secret` link and then **[ Save ]**.

Now go back to the **Overview** page, and watch the deployment finish.

Click on the `Pod ring` and select the first (top) pod in the `Pods` section. Select the `Terminal` tab and then type the following:

```
env | grep CONFIGENV
```

You will see the key/value we just created.

> **TODO**
>
> Explain the relevance of the environment variable - not part of the deployment, applied at the container level

Now let's create another config map. Navigate back to **Resources › Config Maps** and then click **[ Create Config Map ]**

> **TODO**
>
> Discuss the nature of config maps as an embedded overlay file (overwriting image contents)

Enter the following in the fields: * **Name:** configmapfile * **Key:** myapp.conf * **Value:** hello!

Then click **[ Create ]**

Navigate to **Applications › Deployments** select `nodetest` and then the `Configuration` Tab.

In the `Volumes` section, select `Add Config Files`:

- **Source:** configmapfile
- **Mount Path:** /config/app

Click **[ Add ]**

Now go back to the **Overview** page, and watch the deployment finish.

Click on the `Pod ring` and select the first (top) pod in the `Pods` section. Select the `Terminal` tab and then type the following:

```
cd /config/app
cat myapp.conf
```

You will see the value we just created.

**TODO**

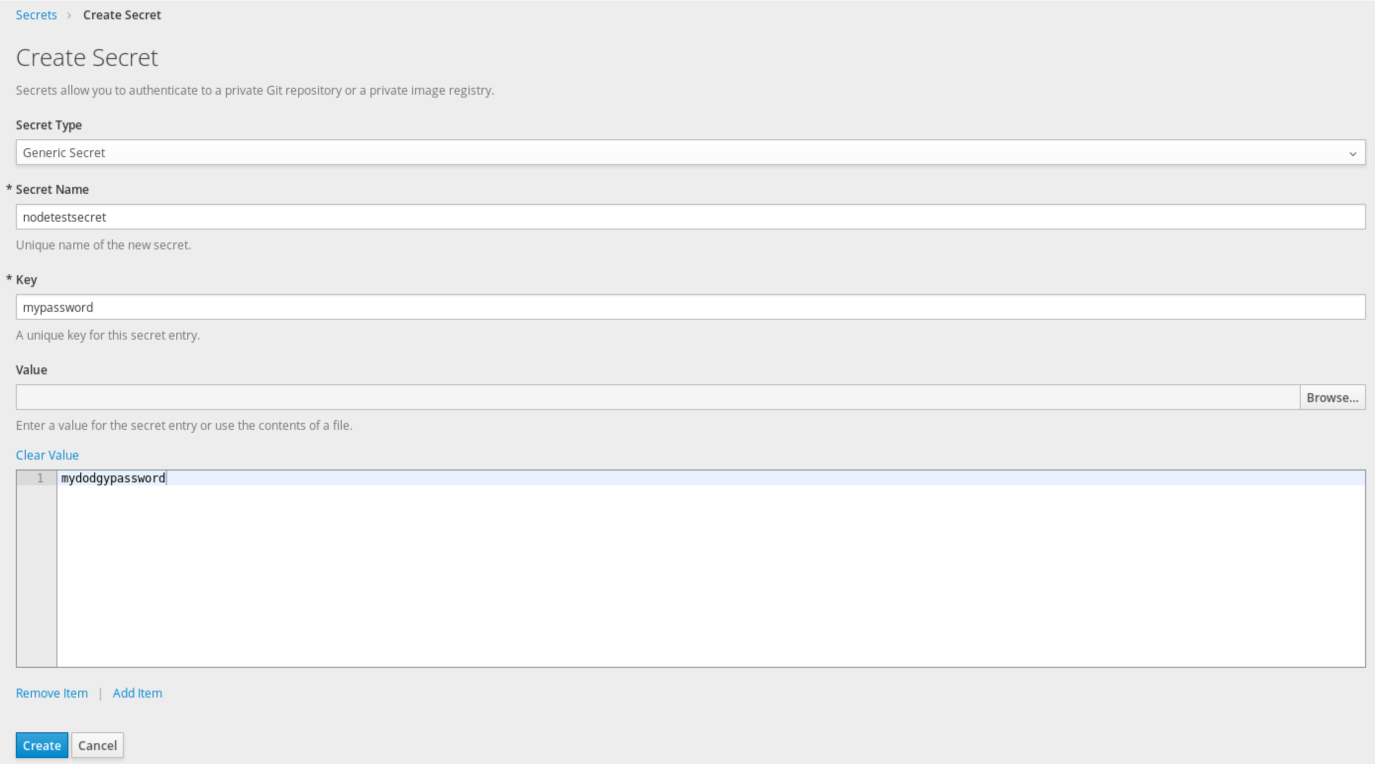
Explain the nature of the config map being written as a file into the container file system - external to image Discuss the difference between configmaps and secrets

## Secrets

Navigate to **Resources › Secrets** and then click **[ Create Secrets ]**. Enter the following:

- **Secret Type:** Generic Secret
- **Secret Name:** nodetestsecret
- **Key:** mypassword
- **Value:** mydodgypassword

Click **[ Create ]**



Secrets › Create Secret

Create Secret

Secrets allow you to authenticate to a private Git repository or a private image registry.

Secret Type: Generic Secret

* Secret Name: nodetestsecret

Unique name of the new secret.

* Key: mypassword

A unique key for this secret entry.

Value

Enter a value for the secret entry or use the contents of a file.

```
mydodgypassword
```

Now select the newly created secret `nodetestsecret` and then click **[ Add to Application ]**.

Select the `nodetest` application and click **[ Save ]**.

Now go back to the **Overview** page, and watch the deployment finish.

Click on the `Pod ring` and select the first (top) pod in the `Pods` section. Select the `Terminal` tab and then type the following:

```
env | grep password
```

**TODO**

Explain the encrypted nature of the secret outside of the Pods

Now, in the CLI (NOT the terminal we have been using just now), do the following:

```
oc describe secret nodetestsecret
```

This will show the secret, example:

```
Name:         nodetestsecret
Namespace:    sandbox-user1
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
mypassword:  15 bytes
```

Now look at the secret in the object:

```
oc edit secret nodetestsecret
```

Here you can see the secret is encrypted:

```
apiVersion: v1
data:
  mypassword: bXlkb2RneXBhc3N3b3Jk
kind: Secret
metadata:
  creationTimestamp: "2019-07-31T05:09:01Z"
  name: nodetestsecret
  namespace: sandbox-user1
  resourceVersion: "167161"
  selfLink: /api/v1/namespaces/sandbox-user1/secrets/nodetestsecret
  uid: 4f06e44d-b351-11e9-b116-16c647cb1fdc
type: Opaque
```

**TODO**

Explain the encryption of the secret at the object level

## Clean up

Now let’s clean up everything we did in the Simple application lifecycle section:

```
oc describe bc nodetest
oc delete all -l "app=nodetest"
```

**TODO**

Point out the Label (app=nodetest), explain its relevance, explain the nature of the extensible object model Explain the clean-up process

# Software Defined Networking

**TODO**

Explain the mechanisms of SDN, routes, singular service endpoints and actual service endpoints.

If you have admin rights (you don't), you can get the cluster network:

```
oc get clusternetwork
NAME      CLUSTER NETWORKS   SERVICE NETWORK   PLUGIN NAME
default   10.1.0.0/16:9      172.30.0.0/16     redhat/openshift-ovs-subnet
```

**TODO**

Explain the differences - RHPDS instance will be subnet, which is the flat networking

Let's create a new test project using the CLI:

```
oc new-project sdntest-userX --description="SDN Test" --display-name="SDN Test"
```

You can make sure you are using the newly created project:

```
oc get project sdntest-user1
```

Now let's ad Applications to our new project:

```
oc new-app registry.access.redhat.com/rhscl/nodejs-8-rhel7~https://github.com/utherp0/nodejs-ex --name="nodetest1"
oc new-app registry.access.redhat.com/rhscl/nodejs-8-rhel7~https://github.com/utherp0/nodejs-ex --name="nodetest2"
```

**TODO**

Explain the lack of a route

Back in the Web Console. Make sure that you select the `SDN Test` Project in the project dropdown.

In the **Overview** page you will see the two nodetest application we just created.

Click on the `Pod ring` for nodetest1, under the `Status` section you will see the IP address for this Pod.

Navigate to **Applications › Services**. Note the Cluster IP’s for nodetest1 and nodetest2.

Now go back to **Overview** and select the `Pod ring` for nodetest1.

Select the `Terminal` tab and type the following:

```
curl http://localhost:8080
```

This will return the basic HTML for the node.js application, example:

```
<html>
<head>
  <title>Test Page for nodejs-ex app</title>
</head>

<body>
<b>Test page for nodejs-ex</b> - served from res.render.

This is a simple webpage rendered from the node.js application at the root endpoint.<p/>

</body>
</html>
```

**TODO**

Explain the resolution of localhost and the output returned

Now do the same, but using the pod name:

```
curl http://nodetest1:8080
```

You will see the same response.

**TODO**

Explain the resolution of the short-name for the service

Let’s see if we can see the other Pod from here:

```
curl http://nodetest2:8080
```

As you can see the name is resolved to the correct IP for a Pod.

> **TODO**
>
> Explain the namespace resolution

Let’s now do a full name resolution:

```
getent hosts nodetest1
```

You will see the Cluster IP and the full URL for nodetest1, example:

```
172.30.179.70   nodetest1.sdntest-user1.svc.cluster.local
```

> **TODO**
>
> Explain the full name resolution and the Cluster IP of the service

Now curl the full URL:

```
curl http://nodetest1.sdntest-userX.svc.cluster.local:8080
```

You will again get the test page as a response.

> **TODO**
>
> Talk about the conversion between the Cluster IP and the node endpoint

Let’s try the person next to you:

```
curl http://nodetest1.sdntest-userX+1.svc.cluster.local:8080
```

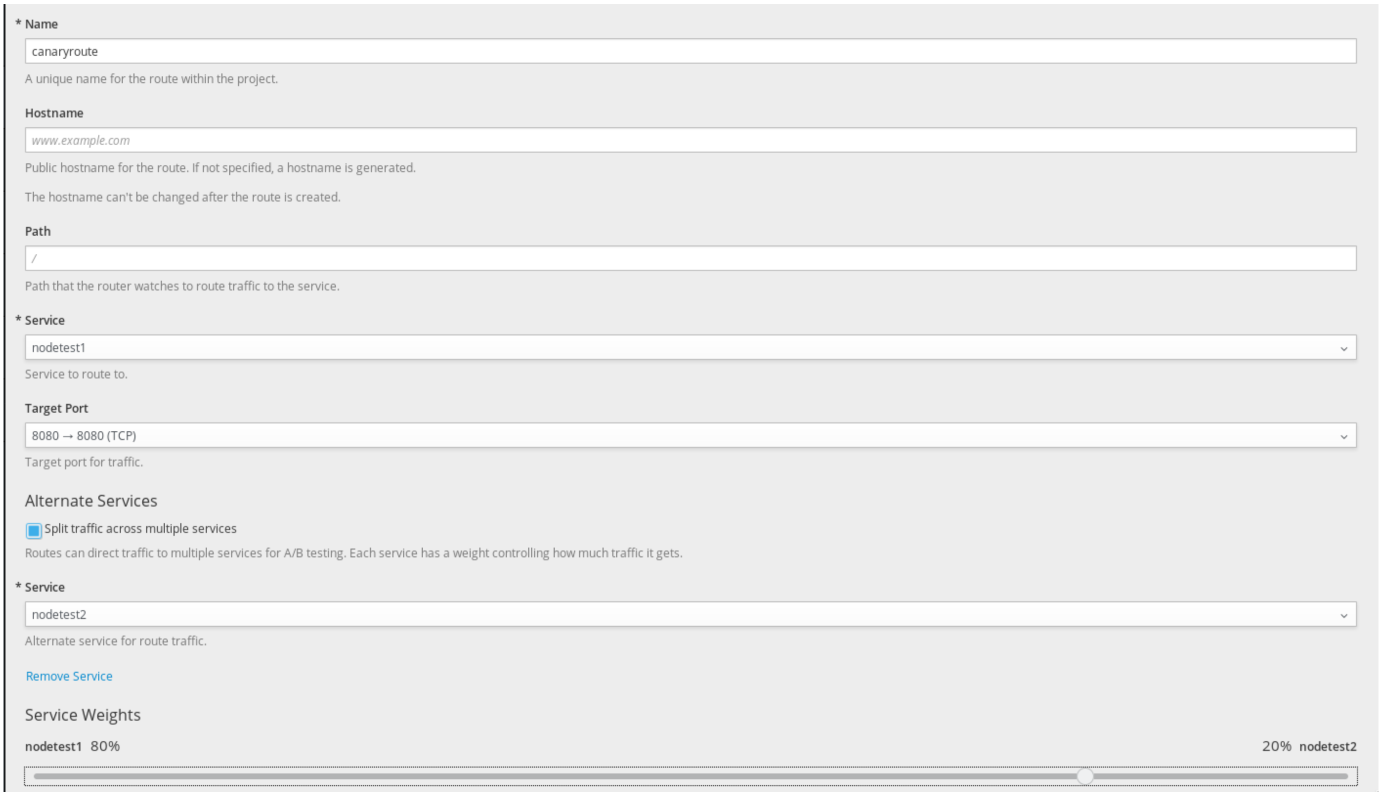
**TODO**

Explain the resolution on a flat SDN. Explain the difference with multi-tenant. Explain network-policy above that

Now let’s create a route, navigate to **Applications › Route** and click **[ Create Route ]**:

- **Name:** canaryroute
- **Service:** nodetest1
- **Alternate Services:** select *Split Traffic Across Multiple Services*
    - **Service:** nodetest2
    - **Service Weights:** 80% nodetest1, 20% nodetest2

Click **[ Create ]**



* Name

canaryroute

A unique name for the route within the project.

Hostname

www.example.com

Public hostname for the route. If not specified, a hostname is generated.

The hostname can't be changed after the route is created.

Path

/

Path that the router watches to route traffic to the service.

* Service

nodetest1

Service to route to.

Target Port

8080 → 8080 (TCP)

Target port for traffic.

Alternate Services

☑ Split traffic across multiple services

Routes can direct traffic to multiple services for A/B testing. Each service has a weight controlling how much traffic it gets.

* Service

nodetest2

Alternate service for route traffic.

Remove Service

Service Weights

nodetest1 80% 20% nodetest2

Now select the newly created `canaryroute`.

**TODO**

Explain the nature of the load-balancing

If you go back to the **Overview** page, note that both applications has got the same external route.

Click on the `Pod ring` of nodetest1, select the `Terminal` tab and then type the following:

```
getent hosts canaryroute-sdntest-userX.(domain)
```

This will return the IP of the OCP router, example:

```
3.219.175.39    canaryroute-sdntest-user1.apps.jhb-94d8.openshiftworkshop.com
```

**TODO**

Explain the IP returned is the IP of the OCP router, and why that is

# User Role based access control

**TODO**

Explain the nature of the user model in OCP compared to the visibility of objects. Explain admin, edit, view and **no** access models

Using the project dropdown, go back to the `Sandbox` project we create earlier.

Go to the **Resources › Membership** page. (using the left side menu)

You will see 3 tabs:

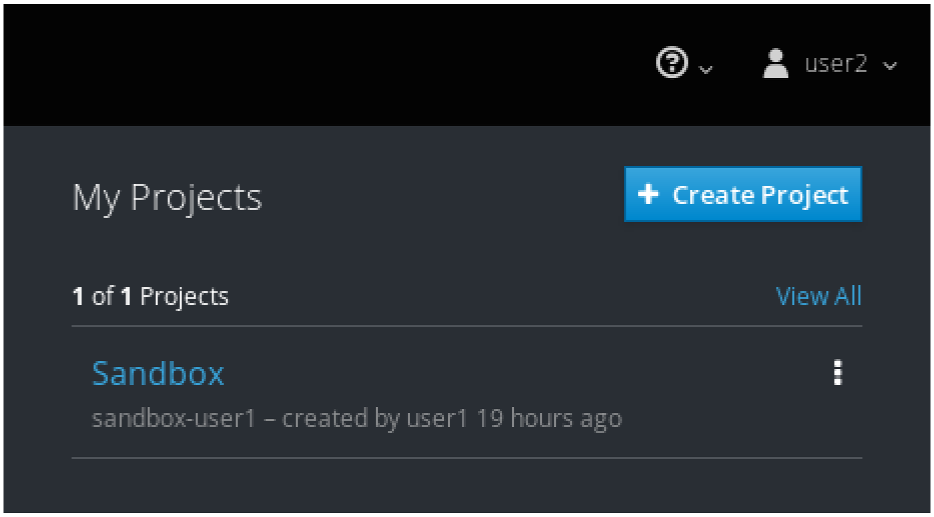
- **Users:** This should list your user (userX).
- **Groups:** - TODO: explain the difference between the full name system:serviceaccount:sandbox-userX and the *all* groups system:serviceaccounts:projectName
- **Service Accounts** - TODO: explain the concept of a *virtual* user within the namespace.

In the `Users` Tab, click **[ Edit Membership ]** and add the attendee next to you as a user:

- **Name:** UserX+1
- **Roles:** admin

Click **[ Add ]**

You will now see the project assigned to you in your Project list, example `User2` can see `sandbox-user-1` project:



(click on the OPENSHIFT CONTAINER PLATFORM logo)

Now do a build of the application in that project.

Next, let’s remove the access from that user using CLI.

First make sure you are using the correct project:

```
oc project sandbox-userX
```

Next remove the access:

```
oc adm policy remove-role-from-user admin userX+1
```

Now go back to your project list and note that your access has been removed.

# CICD Pipeline

For this exercise we will be using some downloadable information - explain the mechanics to the attendees as it is done Show the https://github.com/utherp0/workshop_material repo - explain the need for it (the groovy_pipeline.sh script will pre-install the application, pre-install the pipeline and kick-off the download of the Jenkins container Explain (as with the slides) the role that the 'pipeline' type of build plays - an extended mechanism to allow the whole CICD chain to be encompassed in a single, repeatable OCP build [oc] In the appropriate oc window create a directory for the git source - suggestion is mkdir ~/git [oc] 'cd ~/git' [oc] 'git clone https://github.com/utherp0/workshop_material' [oc] 'cd workshop_material' [oc] 'cd attendee' [oc] 'ls' Explain what the script does. Explain the need for the pipeline_complex_bc.yaml file (and the alternative ways of injecting the pipeline script into the buildconfig) [oc] 'cat groovy_pipeline.sh' Explain the need for changing the namespace - the fact namespaces are unique within a cluster. [oc] sed -i 's/PROJECT.NAME/{uniquenameforattendee}/g' groovy_pipeline.sh [oc] 'cat groovy_pipeline.sh' to make sure the project name is correct [oc] sed -i 's/PROJECT.NAME/{uniquenameforattendee}/g' pipeline_complex_bc.yaml [oc] oc whoami Make sure the attendee is logged on to the OCP cluster [oc] ./groovy_pipeline.sh Explain what the script is doing as it runs - the script installs the nodetest app then waits for it to appear, and then installs and starts the pipeline process Once the process has started, direct the attendees back to the UI [UI] Go to the project/Overview - Watch the Jenkins Ephemeral Pod startup - explain why it takes so long and what it is doing - the creation of a buildconfig that had a pipeline type automatically creates the Jenkins Pod if one doesn't exist [UI] Once the Pod turns blue, click on the Route - THIS MAY TAKE A WHILE - walk the attendees through logging on to the Jenkins instance [UI] Builds/Pipelines - show the stages running, explain what each is doing, refer to the Groovy script [UI] When the pipeline completes (address issues if not - usually a naming convention of the project, if so correct the pipeline_complex_bc.yaml and show how to a: oc delete bc x, and b: oc create -f pipeline_complex_bc.yaml [UI] Overview - show the six pods running, point out the deployment number, Discuss the removal of the triggers on the deployment and why (with automation in place the creation of an image would force a deployment which would interfere with the pipeline, we disable the deployment on image and config triggers so the pipeline can control it all) Discuss complexities of pipelines, the addition of Jenkins commands into the Groovy etc [UI] Go back to the Jenkins Route - select Open Blue Ocean. Choose the Pipeline, click on the build, show the prettier Jenkins UI [UI] Builds/Pipelines - click on complexpipeline - Actions/Edit [UI] Replace the line 'dc,scale("--replicas=6")' with 'dc.scale("--replicas=4")' [UI] Start Pipeline, then switch to Overview Explain what is happening, show the Pod count changing

# Wrap-up

Allow for a Q/A session on the previous chapters. Use the system to demonstrate behaviour asked for by attendees OPTIONAL - show the RHOAR launchpad and walk them through an example OPTIONAL - talk about futures/Operators/OCP4 - demo operators on 3.11 if possible OPTIONAL - walk the attendees, **if** they are .NET devs, through the https://github.com/utherp0/meowworldforum app (currently .NET Core 1.0 but will be updated soon)

Ian Uther Lawson

Principal Domain Solution Architect

uther@redhat.com

Original workshop done by Ian *Uther* Lawson