Red Hat OpenShift

# OpenShift 4.x DevEx Workshop

[YOUR NAME]

Version 1.0.0, December 02, 2019

# Table of Contents

# Introduction

## Attendee details

| | |
|---|---|
| Name: | |
| User ID (userX): | |

This workshop is designed to introduce Developers to **OpenShift 4** and explain the usage and technologies around it from a developer perspective.

### What is Openshift

Red Hat® OpenShift® is a hybrid cloud, enterprise, secure Kubernetes application platform.

OpenShift is a family of containerization software developed by Red Hat. Its flagship product is the OpenShift Container Platform—an on-premises platform as a service built around OCI standard containers orchestrated and managed by Kubernetes on a foundation of Red Hat Enterprise Linux.

### Links

- https://www.openshift.com/learn/what-is-openshift
- https://en.wikipedia.org/wiki/OpenShift
- https://www.openshift.com/products/container-platform
- http://theconsole.blurgh.com (the Web Console URL for the Workshop) [[Application Basics]]

# Application Concepts

## Starting up - logging on and creating a project

Log on to cluster as userx, password openshift

Ensure you are on the Administrator View (top level, select Administrator)

**The Administrator view provides you with an extended functionality interface that allows you to deep dive into the objects available to your user. The Developer view is an opinionated interface designed to ease the use of the system for developers. This workshop will have you swapping between the contexts for different tasks.**

Click on *Create Project*

Name - 'sandboxX' where x is user number

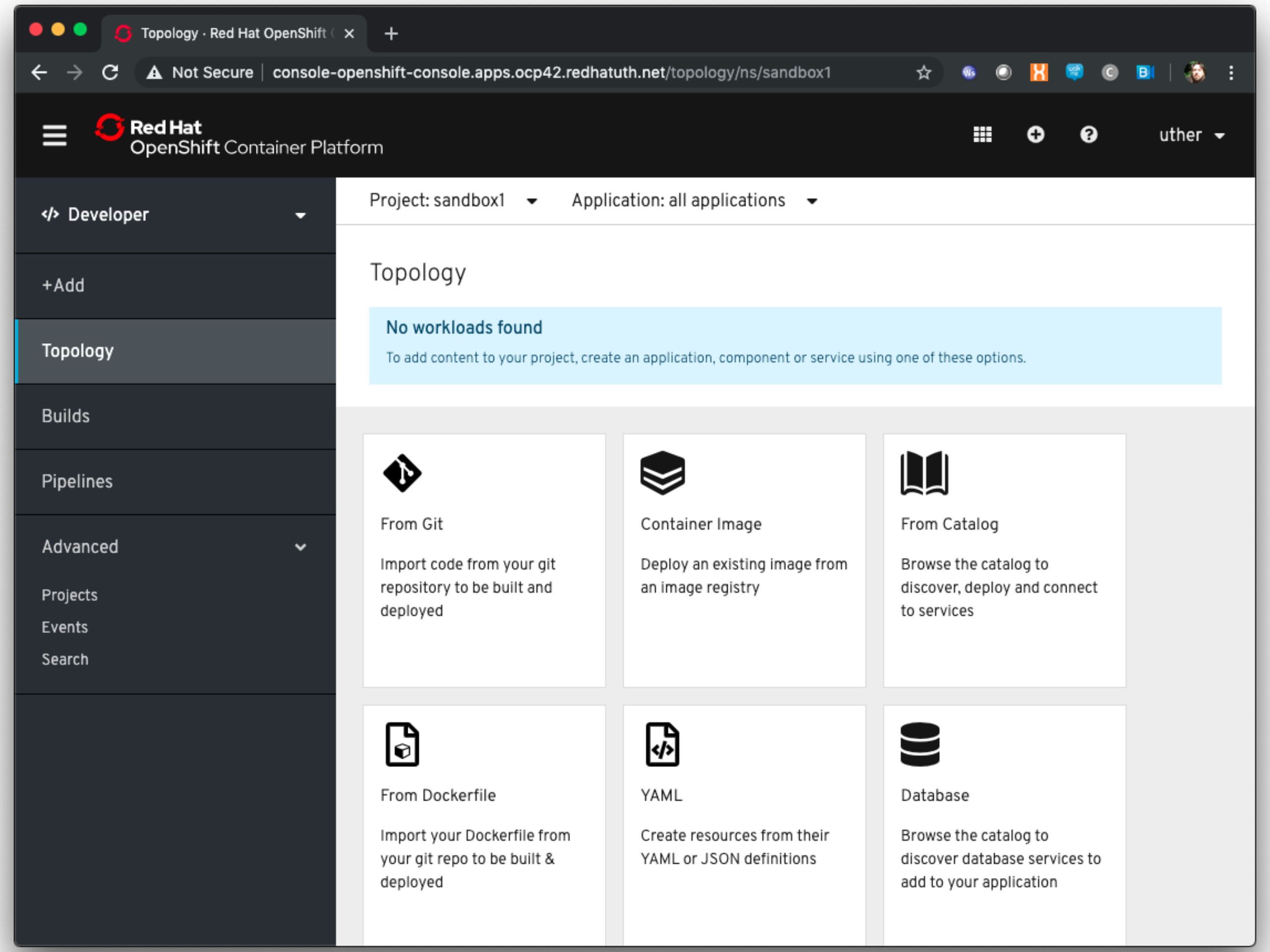
Display Name, Description are labels

When created click on *Role Bindings*

**By default when you create a Project within OpenShift your user is given administration rights. This allows the user to create any objects that they have rights to create and to change the security and access settings for the project itself, i.e. add users as Administrators, Edit Access, Read access or disable other user's abilities to even see the project and the objects within.**

## Creating your first Application

In the top left of the UI, where the label indicates the view mode, change the mode from Administrator to Developer


Topology · Red Hat OpenShift
Not Secure | console-openshift-console.apps.ocp42.redhatuth.net/topology/ns/sandbox1
Red Hat OpenShift Container Platform
uther
Developer
+Add
Topology
Builds
Pipelines
Advanced
Projects
Events
Search
Project: sandbox1
Application: all applications
Topology
No workloads found
To add content to your project, create an application, component or service using one of these options.
From Git
Import code from your git repository to be built and deployed
Container Image
Deploy an existing image from an image registry
From Catalog
Browse the catalog to discover, deploy and connect to services
From Dockerfile
Import your Dockerfile from your git repo to be built & deployed
YAML
Create resources from their YAML or JSON definitions
Database
Browse the catalog to discover database services to add to your application

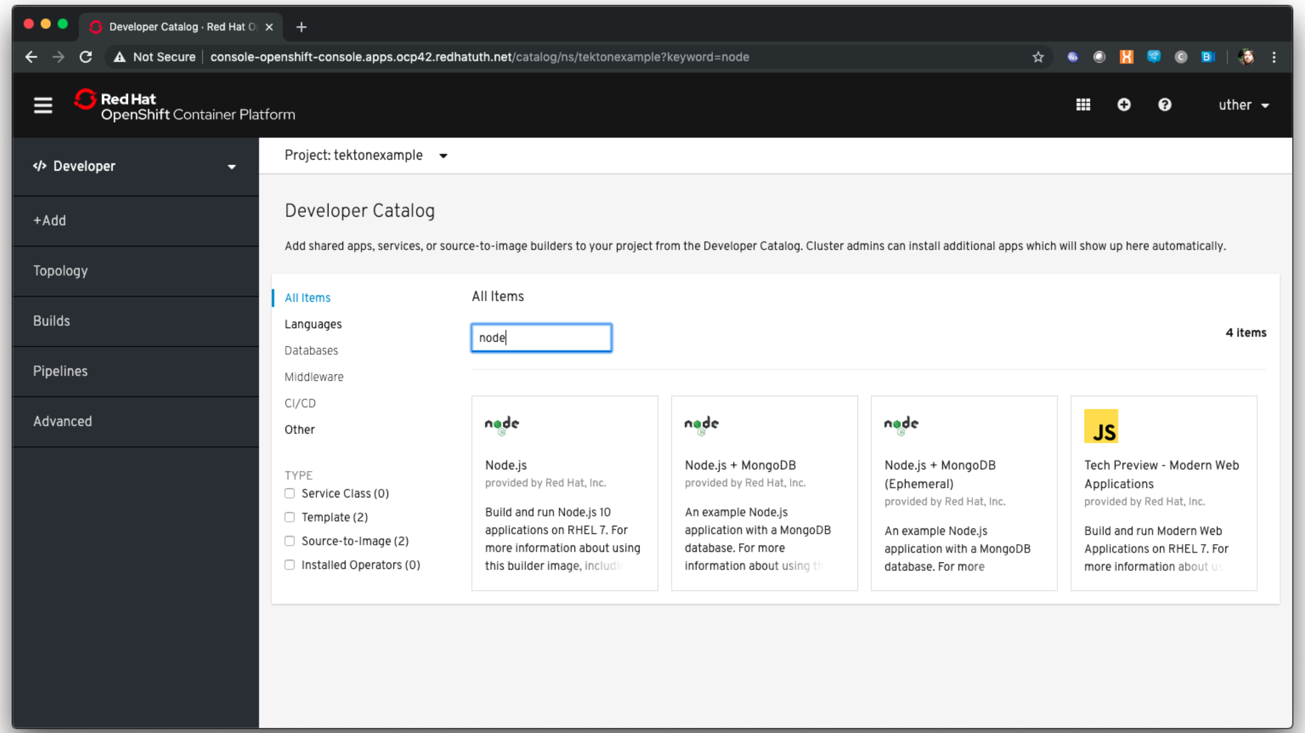
Click *Add*

**The Catalog screen for the developer combines all the ways components can be added into the Project. These are:**

- **From Git - this provides another way to do a Source-2-Image build by first choosing the Git repo and then the builder image to use**
- **Container Image - this provides a way to directly deploy an Image from a repository**
- **From Catalog - this allows the Developer to browse all available templates, which are predefined sets of Objects to create an application**
- **From Dockerfile - this allows the Developer to do a controlled build of an Image from a Dockerfile**
- **YAML - this allows the Developer to provide a set of populated YAML to define the objects to be added to the Project**
- **Database - this allows the Developer to browse pre-created Database services to add to the Project**

Select ‘From Catalog’

Enter ‘node’ in the search box


Developer Catalog · Red Hat O
Not Secure | console-openshift-console.apps.ocp42.redhatuth.net/catalog/ns/tektonexample?keyword=node
Red Hat OpenShift Container Platform
uther
Developer
+Add
Topology
Builds
Pipelines
Advanced
Project: tektonexample

Developer Catalog

Add shared apps, services, or source-to-image builders to your project from the Developer Catalog. Cluster admins can install additional apps which will show up here automatically.

Node.js
provided by Red Hat, Inc.
Build and run Node.js 10 applications on RHEL 7. For more information about using this builder image, includi

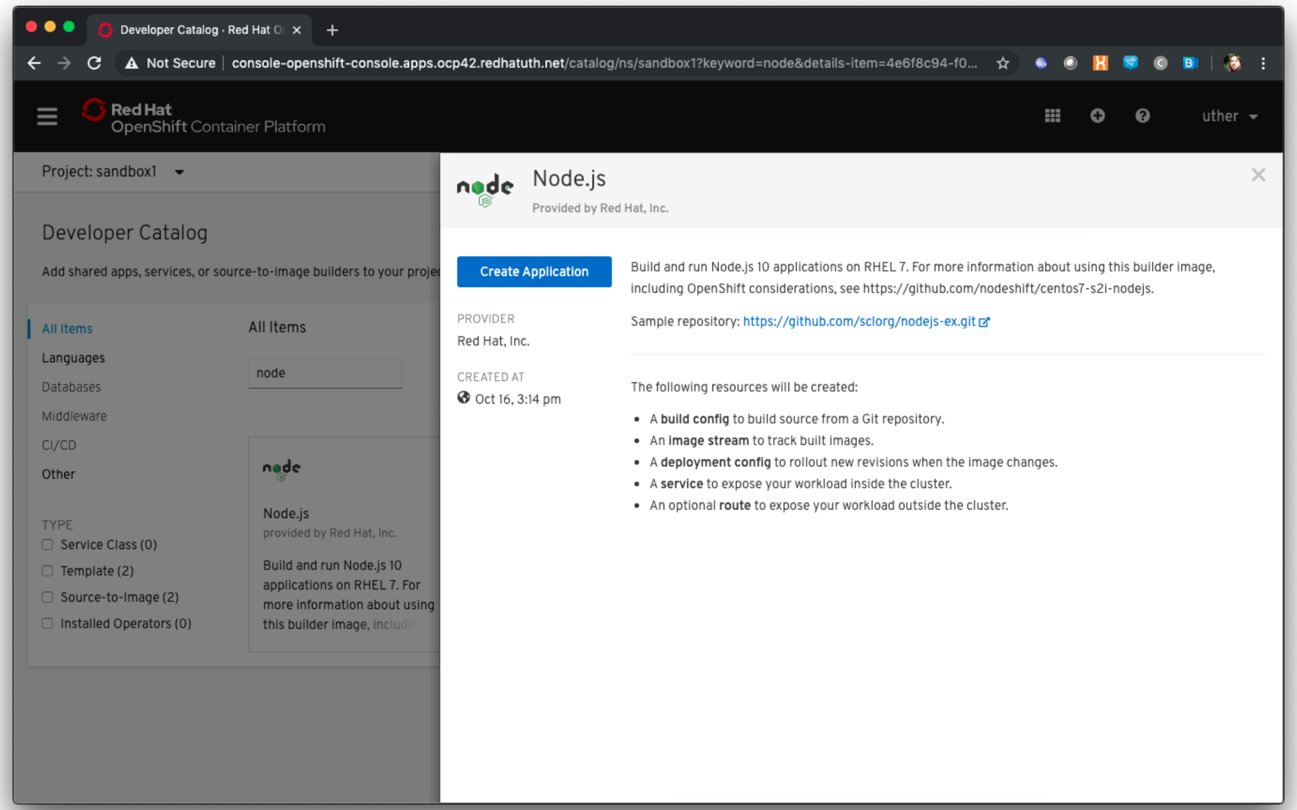
Node.js + MongoDB
provided by Red Hat, Inc.
An example Node.js application with a MongoDB database. For more information about using th

Node.js + MongoDB (Ephemeral)
provided by Red Hat, Inc.
An example Node.js application with a MongoDB database. For more

Tech Preview - Modern Web Applications
provided by Red Hat, Inc.
Build and run Modern Web Applications on RHEL 7. For more information about us

**OpenShift allows for multiple base images to be built upon - the selection of node searches for any images or templates registered into the system with the label *node*. In the screenshot above, and in the catalog you will be presented with, there will be a selection of base images.**

Select ‘Node.js’


Developer Catalog · Red Hat O
Not Secure | console-openshift-console.apps.ocp42.redhatuth.net/catalog/ns/sandbox1?keyword=node&details-item=4e6f8c94-f0...
Red Hat
OpenShift Container Platform
uther
Project: sandbox1
Developer Catalog
Add shared apps, services, or source-to-image builders to your proje
All Items
Languages
Databases
Middleware
CI/CD
Other
All Items
node
TYPE
Service Class (0)
Template (2)
Source-to-Image (2)
Installed Operators (0)
Node.js
provided by Red Hat, Inc.
Build and run Node.js 10 applications on RHEL 7. For more information about using this builder image, includi

Node.js

Provided by Red Hat, Inc.

Create Application

Build and run Node.js 10 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/nodeshift/centos7-s2i-nodejs.

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

PROVIDER
Red Hat, Inc.

CREATED AT
Oct 16, 3:14 pm

The following resources will be created:

- A **build config** to build source from a Git repository.
- An **image stream** to track built images.
- A **deployment config** to rollout new revisions when the image changes.
- A **service** to expose your workload inside the cluster.
- An optional **route** to expose your workload outside the cluster.

**When you select an option, in this case the Node.js builder one, you are presented with a wizard that shows you exactly what components will be created as part of your Project. In this case, with Node.js, the template will create a build config, that will build the Image that will contain your Application, an ImageStream which is the OpenShift representation of an Image, a deployment config, which defines exactly how the image will be deployed as a running Container within the Project, a service which is the internal endpoint for the application within the Project and a route, optionally, which will provide access to the Application for external consumers.**

Click on *Create Application*

**This approach uses the OpenShift '*source-2-image*' concept, which is a controlled mechanism provided by OpenShift that automates the creation of application images using a base image and a source repository.**

Change the Builder Image Version to 8

**The '*source-2-image*' approach allows you to use differing versions of a base image - in this case you can execute the Node build against a number of supported Node versions.**

Git repo - https://github.com/utherp0/nodenews

In a separate browser tab go to https://github.com/utherp0/nodenews

**If you visit the URL you will see there is no OpenShift specific code in the repository at all.**

Close the github tab

Back at the OCP4.2 Ux leave the Application as ‘Create Application’.

**OpenShift 4 introduces the concept of Application Grouping. This allows a visualisation of multiple Applications in the same *group*, making visibility of the Application much simpler.**

Ensure the application name is ‘nodenews-app’

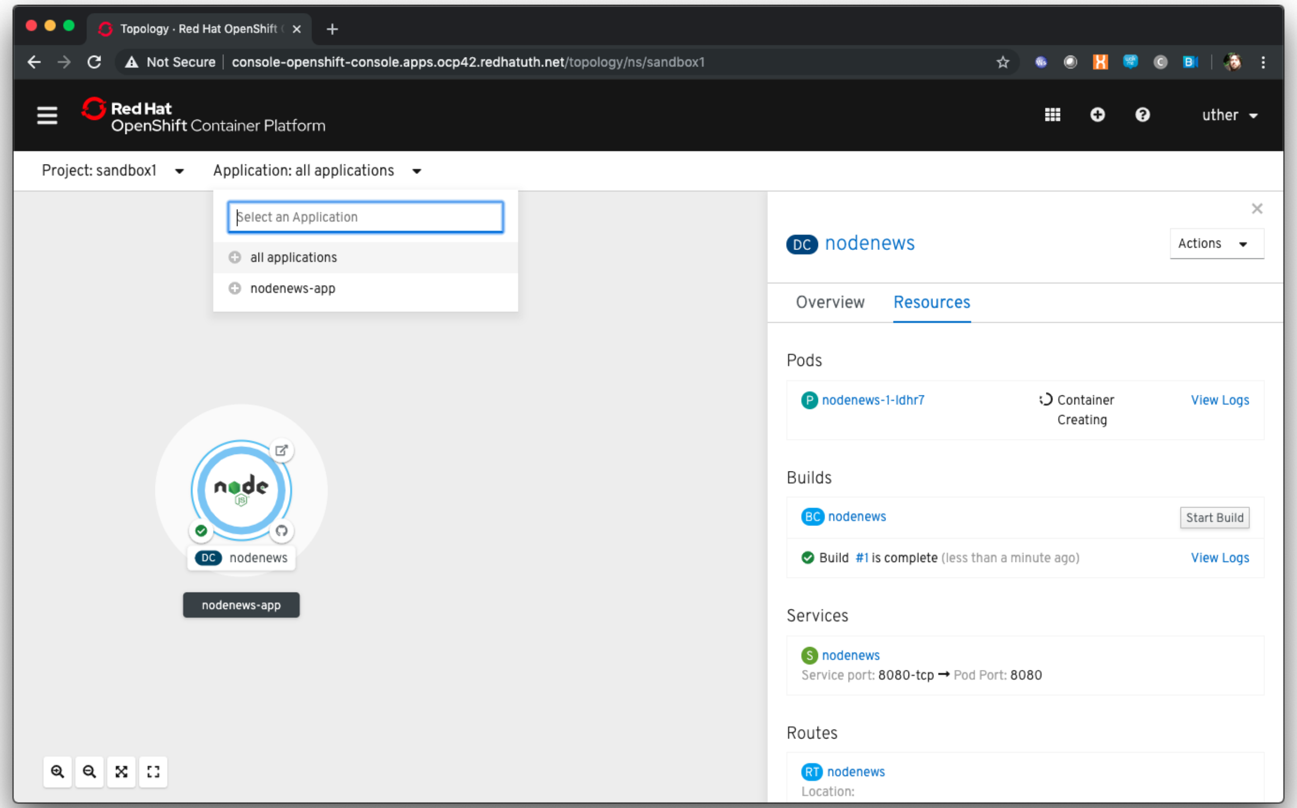
Ensure the Name is ‘nodenews’

Ensure the ‘Create Route is checked’

Click *Create*

**The Topology view is a new feature of OpenShift 4 that provides a dynamic and useful visualisation of all of your Applications in a given Project.**

Click on the Icon marked *Node*



Topology · Red Hat OpenShift
Not Secure | console-openshift-console.apps.ocp42.redhatuth.net/topology/ns/sandbox1
Red Hat OpenShift Container Platform
uther
Project: sandbox1
Application: all applications
Select an Application
all applications
nodenews-app
DC nodenews
nodenews-app
DC nodenews
Actions
Overview
Resources

Pods

P nodenews-1-ldhr7 — Container Creating — View Logs

Builds

BC nodenews — Start Build

Build #1 is complete (less than a minute ago) — View Logs

Services

S nodenews

Service port: 8080-tcp → Pod Port: 8080

Routes

RT nodenews

Location:

**The side-panel contains an overview of the Application you chose. In this case it will cover the build. Once a build has completed this side panel shows the Pods that are running, the builds that have completed, the services exported for the Application and the routes, if the Application has any.**

Wait for the Build to finish, the Pod to change from Container Creating to Running

**When an Application is created the Pod ring will be empty, indicating that an Application will appear once the build has completed. When the build completes the Pod ring will switch to light blue, indicating the Pod is being pulled (the image is being pulled from the registry to the Node where the Pod will land) and is starting (the Pod is physically in place but the Containers within it are not reporting as ready). Once the Pod is placed and running the colour of the Pod ring will change to dark blue.**

Click on the Tick at the bottom left of the Pod

**If you scroll the log of the Build output you will see the steps that the build takes. This includes laying the foundational file layers for the base image, performing the code specific build operations (in this case an *'npm install'*) and then pushing the file layers for the image into the OpenShift integrated registry.**

## Adding additional Applications

Click on *Topology*

Click *Add*

Click *From Catalog*

Search for ‘httpd’

Select the (httpd) template

Click on *Create Application*

Leave Image Version as 2.4

Set the git repo to ‘https://github.com/utherp0/forumstaticassets’

Ensure the Application is ‘nodenews-app’

Ensure the name is forumstaticassets

Ensure the ‘Create a Route’ checkbox is clicked

Click *Create*

**Note that the new Application icon appears within a bounded area on the Topology page labelled with the *Application* chosen above. If you click on the area between the Pods you can move the group as a single action.**

Click on the forumstaticassets Pod

Watch the build complete, the Container Creating and the Running event

Click *Add*

Click *From Catalog*

Search for ‘node’

Select ‘Node.js’

Click *Create Application*

Leave at Builder Image Version 10

Set the git repo to ‘https://github.com/utherp0/ocpnode’

In the ‘Application’ pulldown select ‘Create Application

In the ‘Application Name’ enter ‘ocpnode-app’

Ensure the Name is ‘ocpnode’

Ensure the ‘Create Route’ is checked

Click *Create*

Click on the ‘ocpnode’ Application in the topology - click on the ‘cross’ icon to centralise the topology

**Now we have created a new Application grouping you will see two '*cloud*' groupings, labelled with the appropriate Application name you entered.**

## Interacting with OpenShift through the Command Line

Open another tab in the browser and go to the terminal app route (noted earlier)

Log on with the same user credentials as used for the Ux (i.e. userx and openshift)

When the terminal window renders type:

```
oc whoami
```

Ensure the user listed is the user that was logged on

```
oc version
oc projects
```

User should have access to one project, explain the ability to access multiple projects

```
oc project sandboxX
```

User should now be using the sandboxX project created and configured earlier

```
oc get users
```

**There is a level of permission within the OpenShift system called *'Cluster Admin'*. This permission allows a User to access *any** of the objects on the system regardless of Project. It is effectively a super-user and as such normal users do not normally have this level of access.*

```
oc get pods
```

**If you look carefully at the Pods shown you will notice there are additional Pods above and beyond the ones expected for your Applications. If you look at the state of these Pods they will be marked as Completed. Everything in OpenShift is executed as a Pod, including Builds. These completed Pods are the Builds we have run so far.**

```
oc get pods | grep Completed
```

```
oc get pods | grep Running
```

```
oc get dc
```

**DC is an abbreviation for Deployment Config. These are Objects that directly define how an Application is deployed within OpenShift. This is the *'ops'* side of the OpenShift system. Deployment Configs are different to Kubernetes Deployments in that they are an extension and contain things such as Config Maps, Secrets, Volume Mounts, labelled targetting of Nodes and the like.**

```
oc scale dc/nodenews --replicas=2
```

## A Summary of Application Interactions

Go back to the UI and make sure you are on Developer mode. Click on Topology.

Click on the ‘nodenews’ application

Note the ‘DC’ reference to the application under the icon

In the pop-up panel on the right click on *Resources*

Note that there are two pods running with the application now

Change the mode from Developer to Administrator

Select the sandboxx project in the project list

Note the metrics for the project

Click on *Workloads* and then select Pods.

Change to Developer mode and then select Topology if the Topology page isn’t already shown

Click and hold on the forumstaticassets Pod and drag it onto the ocpnode-app

Select ‘move’ when it prompts whether you want to move it

# Pod Health Probes

## Introduction

Liveness and Readiness probes are Kubernetes capabilities that enable teams to make their containerised applications more reliable and robust. However, if used inappropriately they can result in none of the intended benefits, and can actually make a microservice based application unstable.

The purpose of each probe is quite simple and is described well in the OpenShift documentation here. The use of each probe is best understood by examining the action that will take place if the probe is activated.

**Liveness** : Under what circumstances is it appropriate to restart the pod?

**Readiness** : under what circumstances should we take the pod out of the list of service endpoints so that it no longer responds to requests?

Coupled with the action of the probe is the type of test that can be performed within the pod :

**Http GET request** : For success, a request to a specific http endpoint must result in a response between 200 and 399.

**Execute a command** : For success, the execution of a command within the container must result in a return code of 0.

**TCP socket check** : For success, a specific TCP socket must be successfully opened on the container.

## Creating the project and application

Log on to cluster as userx, password openshift

Ensure you are on the Administrator View (top level, select Administrator)

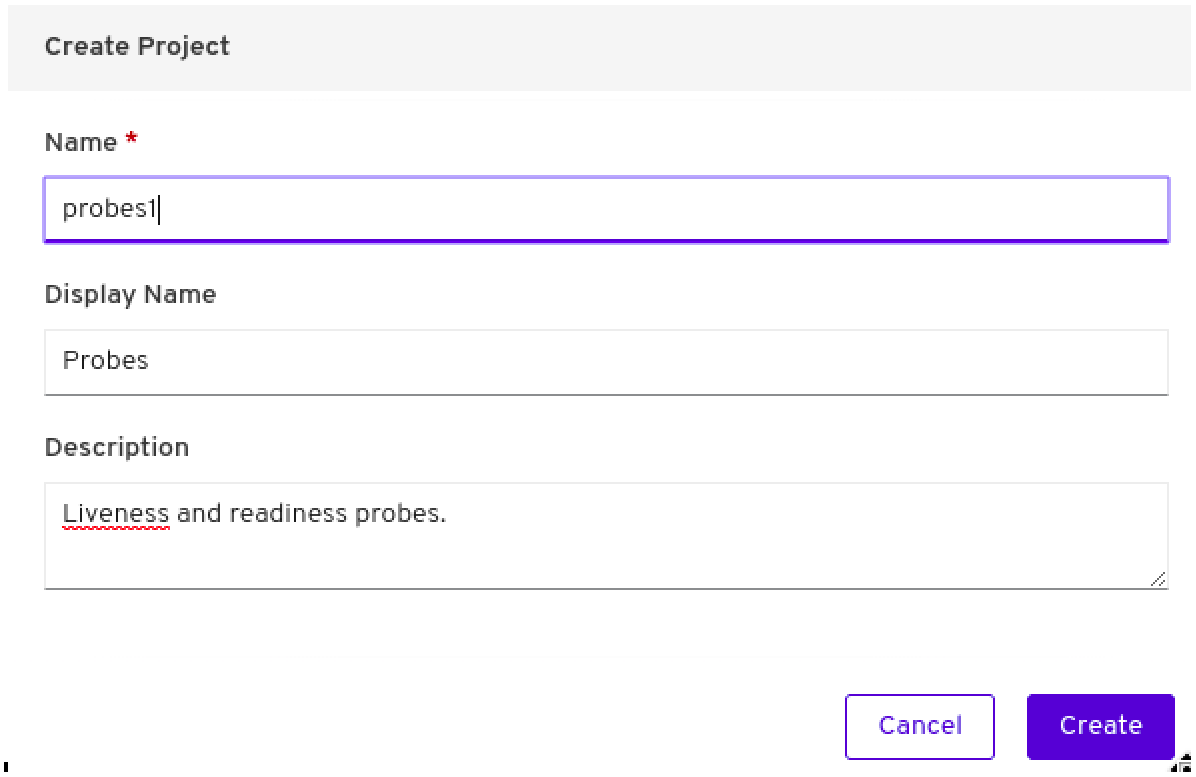
**The Administrator view provides you with an extended functionality interface that allows you to deep dive into the objects available to your user. The Developer view is an opinionated interface designed to ease the use of the system for developers. This workshop will have you swapping between the contexts for different tasks.**

Click on *Create Project*

Name - ‘probesX’ where X is your assigned user number

Display Name - *Probes*

Desciption - *Liveness and readiness probes* as shown in the image below:



Name *

probes1

Display Name

Probes

Description

Liveness and readiness probes.

**By default when you create a Project within OpenShift your user is given administration rights. This allows the user to create any objects that they have rights to create and to change the security and access settings for the project itself, i.e. add users as Administrators, Edit Access, Read access or disable other user's abilities to even see the project and the objects within.**

**In the top left of the UI, where the label indicates the view mode, change the mode from Administrator to Developer and select the Topology view.**

Select 'From Catalog'

Enter 'node' in the search box

**The various catalogue items present different configurations of applications that can be created. For example the node selections include a simple node.js application or node.js with a database platform that is pre-integrated and ready to use within the application. For this workshop you will use a simple node.js application.**

Select 'Node.js'

**A wizard page will then pop up on the right hand side with details of exactly what will be created through the process.**

## Source-2-Image

One of the most exciting features of OpenShift from a developer's perspective is the concept of S2I, or **Source-2-Image** which provides a standardised way of taking a base image, containing, for example, the framework for running a node.js application, a source code repository, containing the code of the application that matches the framework provided in the base image, and constructing and delivering a composite Application image to the Registry. Simply put this enables a developer to create source code for an application and OpenShift will convert that to a running application within a container with minimal effort. The process that you will use below is the Source-2-Image capability.

Click on 'Create Application'

**The wizard process has a number of options that the user may elect to use. These include:**

- Selecting a specific version of Node to use
- Selecting a GIT repository and choosing to use code from a specific directory within the repository.

Select the default offering for the Builder Image Version

For the GIT repository use : https://github.com/marrober/slave-node-app.git

In a separate browser tab go to https://github.com/marrober/slave-node-app.git

**You can see that the GIT repository at this location only has a small number of files specific to the application. There is no content specific to how the application should be built or deployed into a container.**

Close the github tab

Back at the OCP4.2 user interface complete the following information in the section titmed *General*.

For the Application drop down menu select *Create Application*.

For the Application name enter *Slave-application*.

For the Name field enter *node-app-slave*

**The application name field is used to group together multiple items under a single grouping within the topology view. The name field is used to identify the specific containerised application.**

Ensure the 'Create a route to the application' checkbox is checked.

Click Create

**The user interface will then change to display the Topology view. This is a pictorial**

**representation of the various items (applications, databases etc.) that have been created from the catalogue and added to an application grouping.**

**Multiple application groupings can exist within a single project.**

## Viewing the running application

Click on the Icon marked ‘Node’

**A side panel will appear with information on the Deployment Configuration that has just been created. The overview tab shows summary information and allows the user to scale the number of pods up and down. The resources tab shows information on the building process, the pods and the services and route to reach the application (if these have been created).**

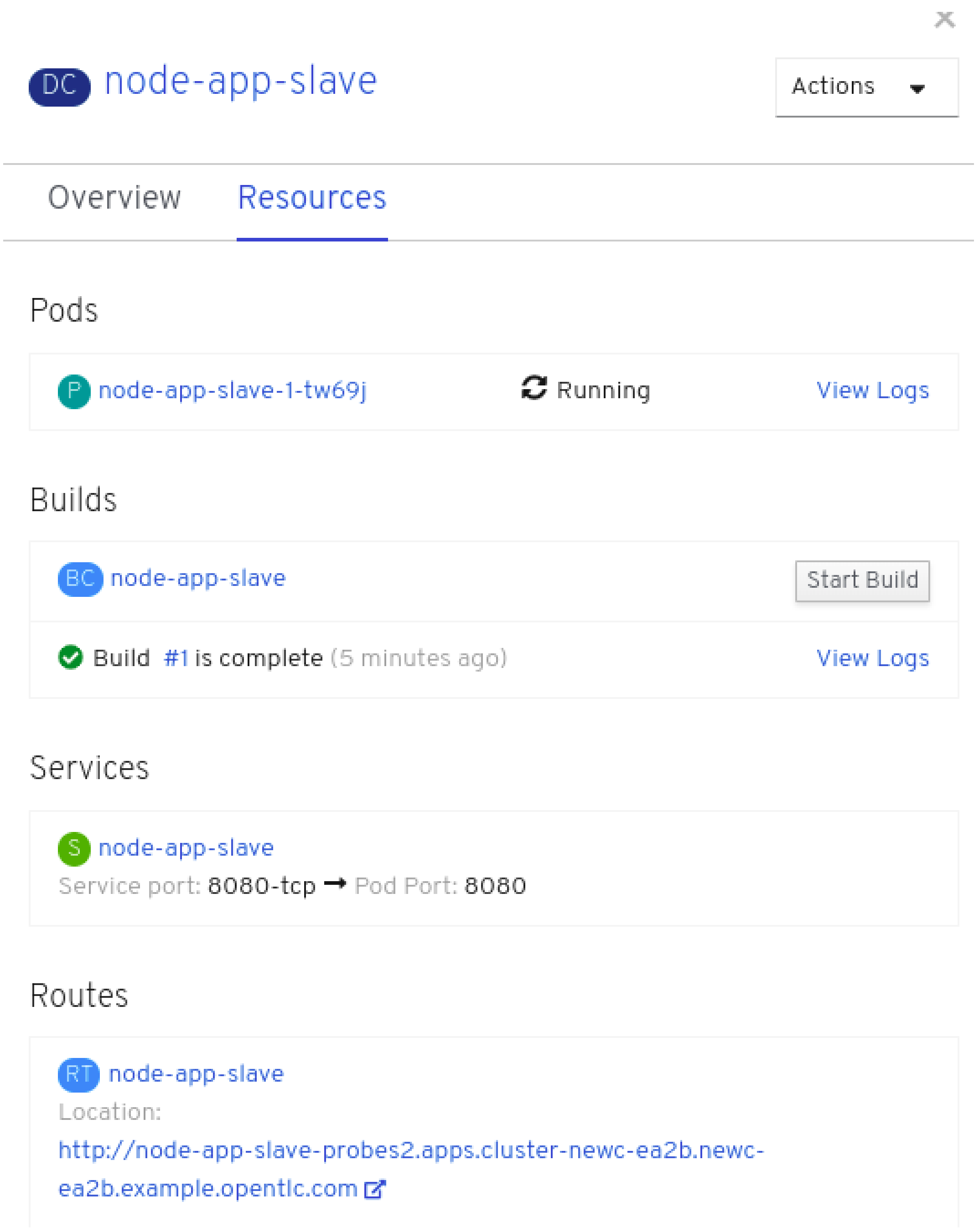
Wait for the Build to finish, the Pod to change from Container Creating to Running.

### Pod Colour Scheme

The colour scheme of the pods is important and conveys information about the pod health and status. The following colours are used :

- Running - Dark blue
- Not Ready - Mid blue
- Warning - Orange
- Failed - Red
- Pending - light blue
- Succeeded - Green
- Terminating - Black
- Unknown - Purple

When the build has completed the right hand side panel will shown something similar to the image below. Note that the route will be different to that which is shown below.


DC node-app-slave

Actions

Overview | Resources

Pods

P node-app-slave-1-tw69j Running View Logs

Builds

BC node-app-slave Start Build

Build #1 is complete (5 minutes ago) View Logs

Services

S node-app-slave

Service port: 8080-tcp → Pod Port: 8080

Routes

RT node-app-slave

Location:

http://node-app-slave-probes2.apps.cluster-newc-ea2b.newc-ea2b.example.opentlc.com

Click on the Tick at the bottom left of the Pod. Note that this display can also be shown by clicking on the ‘View Logs’ section on the right hand side panel.

**The build log will show information on the execution of the source-2-image process.**

Click on the arrow on the top right corner of the Pod, or click on the route URL shown in the right hand side resource details window. The application window will launch in a new browser window and should display text as shown below:

```
Hello - this is the simple slave REST interface
```

## Liveness Probe

**A number of probes will be created to show the different behaviours. The first probe will be a liveness probe that will result in the restart of the pod.**

**To create the probe use the OC command line interface to execute the following command.**

```
oc set probe dc/node-app-slave --liveness --initial-delay-seconds=30 --failure-threshold
=1 --period-seconds=10 --get-url=http://:8080/health
```

**The above probe will create a new liveness probe with the characteristics:**

- Become active after 30 seconds
- Initiated a reboot after 1 instance of a failure to respond
- Probe the application every 10 seconds *Note that ordinarily a gap of 10 seconds between probes would be considered very long, but we use this time delay within the workshop to allow time for observing the behaviour of the probe.*
- Use the URL /health on the application at port 8080. Note that there is no need to specify a URL for the application.

**The command line response should be as shown below.**

```
deploymentconfig.apps.openshift.io/node-app-slave probes updated
```

**Review the liveness probe information by executing the command:**

```
oc describe dc/node-app-slave
```

**The output of this command will include the following section that highlights the new liveness probe**

```
Pod Template:
  Labels:   app=node-app-slave
            deploymentconfig=node-app-slave
  Containers:
   node-app-slave:
    Image:      image-registry.openshift-image-registry.svc:5000/probes2/node-app-
slave@sha256:bf377...241
    Port:           8080/TCP
    Host Port:      0/TCP
    Liveness:       http-get http://:8080/health delay=30s timeout=1s period=10s
#success=1 #failure=1
    Environment:    <none>
    Mounts:         <none>
  Volumes:          <none>
```

**Alternatively to view the probe in a different format use the command below:**

```
oc get dc/node-app-slave -o yaml
```

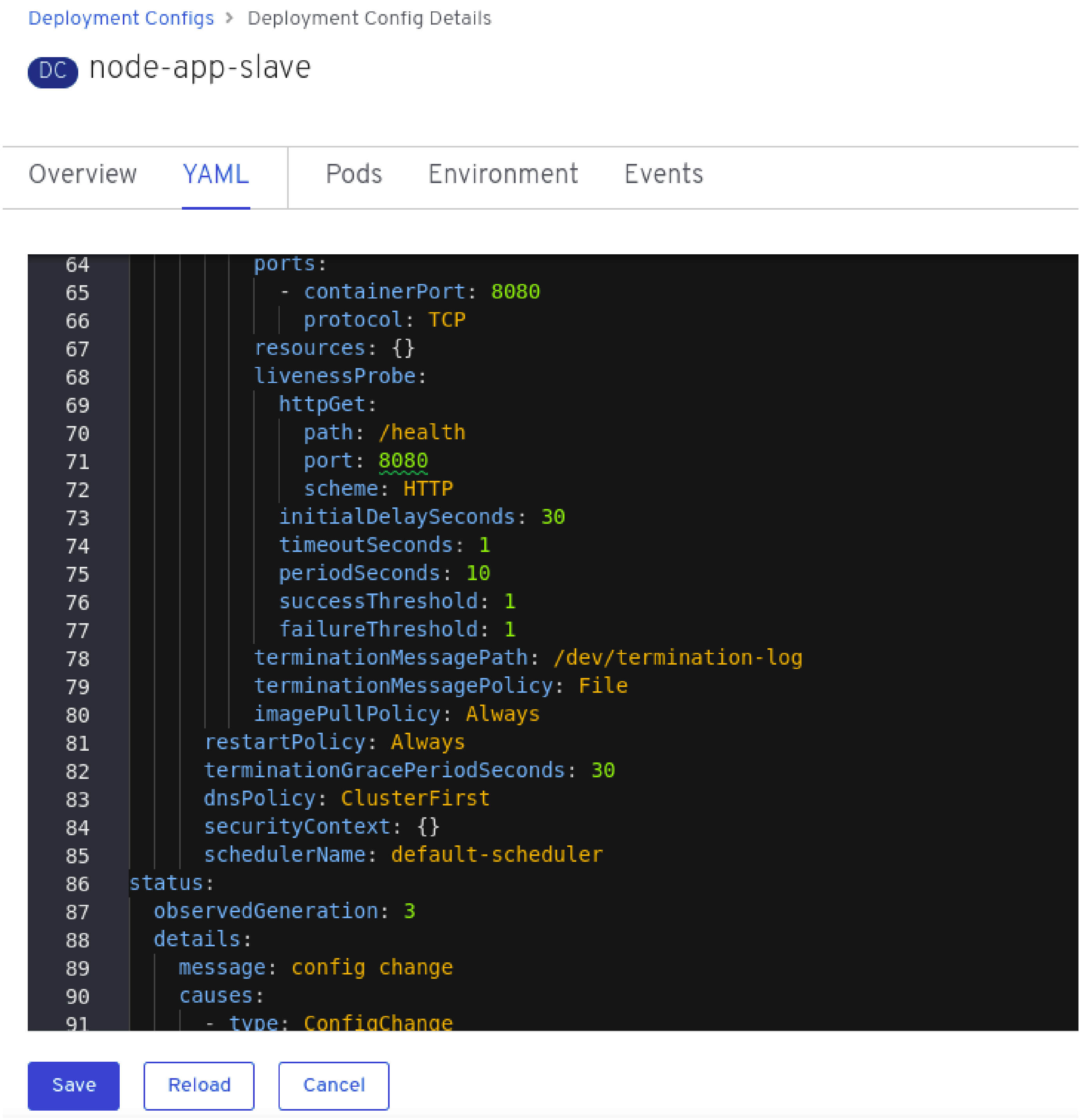
**Part of the output will show:**

```
livenessProbe:
    failureThreshold: 1
    httpGet:
        path: /health
        port: 8080
        scheme: HTTP
    initialDelaySeconds: 30
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 1
```

**To view the above information graphically then use the following steps:**

Select the Topology view of the application.

Click on the pod in the centre of the screen to display the information panel on the right hand side. From the action menu on the right hand side click **Edit Deployment Configuration** as shown in the image below.



```
64            ports:
65              - containerPort: 8080
66                protocol: TCP
67            resources: {}
68            livenessProbe:
69              httpGet:
70                path: /health
71                port: 8080
72                scheme: HTTP
73              initialDelaySeconds: 30
74              timeoutSeconds: 1
75              periodSeconds: 10
76              successThreshold: 1
77              failureThreshold: 1
78            terminationMessagePath: /dev/termination-log
79            terminationMessagePolicy: File
80            imagePullPolicy: Always
81        restartPolicy: Always
82        terminationGracePeriodSeconds: 30
83        dnsPolicy: ClusterFirst
84        securityContext: {}
85        schedulerName: default-scheduler
86  status:
87    observedGeneration: 3
88    details:
89      message: config change
90      causes:
91        - type: ConfigChange
```

**On the Deployment Configuration page that is displayed ensure that the YAML tab is selected and scroll down to aroundline 68 to see the YAML definition for the liveness probe. It is also possible to edit the parameters of the probe from this screen if necessary.**

**In order to execute the probe it is necessary to simulate a pod failure that will stop the application from responding to the health check. A specific REST interface on the application has been created for this purpose called `/ignore`.**

## Activation of the Liveness Prove

**To view the activity of the probe it is necessary to open two windows.**

Select the Topology view of the application.

Click on the arrow on the top right hand corner of the node icon to open the application URL in a new browser tab.

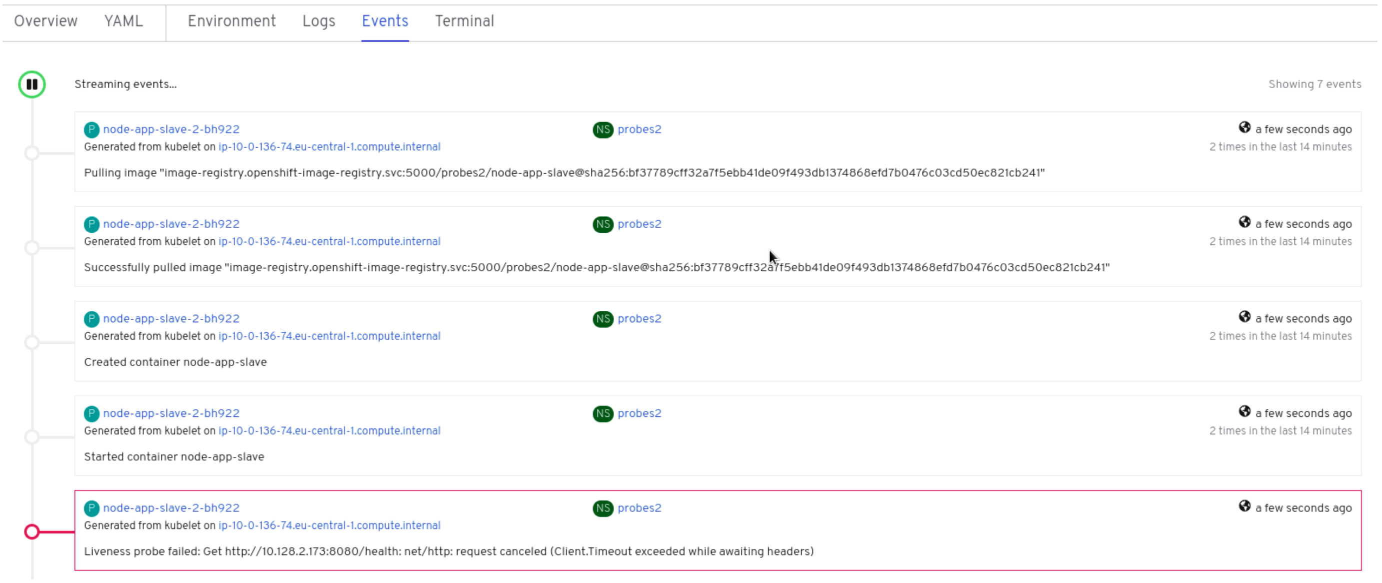
Back on the OpenShift browser tab, Click on the pod to open the details window on the right hand side and then click on the pod link on the resources tab. This will display a multi-tab window with details of the pod, select the events tab.

Switch to the application tab and put /ip on the end of the url and hit return. This will display the ip address of the pod.

Change the url to have /health on the end and hit return. This will display the amount of time that the pod has been running.

Change the url to have /ignore on the end and hit return. Quickly move to the browser tab showing the pod events and watch for the probe event.

The pod will restart after 1 failed probe event which takes up to 10 seconds depending on where the schedule is between the probe cycles. The events for the pod on the details screen will be similar to that shown below.



```
Streaming events...                                                                 Showing 7 events

P node-app-slave-2-bh922                          NS probes2                        a few seconds ago
Generated from kubelet on ip-10-0-136-74.eu-central-1.compute.internal     2 times in the last 14 minutes
Pulling image "image-registry.openshift-image-registry.svc:5000/probes2/node-app-slave@sha256:bf37789cff32a7f5ebb41de09f493db1374868efd7b0476c03cd50ec821cb241"

P node-app-slave-2-bh922                          NS probes2                        a few seconds ago
Generated from kubelet on ip-10-0-136-74.eu-central-1.compute.internal     2 times in the last 14 minutes
Successfully pulled image "image-registry.openshift-image-registry.svc:5000/probes2/node-app-slave@sha256:bf37789cff32a7f5ebb41de09f493db1374868efd7b0476c03cd50ec821cb241"

P node-app-slave-2-bh922                          NS probes2                        a few seconds ago
Generated from kubelet on ip-10-0-136-74.eu-central-1.compute.internal     2 times in the last 14 minutes
Created container node-app-slave

P node-app-slave-2-bh922                          NS probes2                        a few seconds ago
Generated from kubelet on ip-10-0-136-74.eu-central-1.compute.internal     2 times in the last 14 minutes
Started container node-app-slave

P node-app-slave-2-bh922                          NS probes2                        a few seconds ago
Generated from kubelet on ip-10-0-136-74.eu-central-1.compute.internal
Liveness probe failed: Get http://10.128.2.173:8080/health: net/http: request canceled (Client.Timeout exceeded while awaiting headers)
```

**The events after the firing of the liveness probe are the re-pulling and starting of the container image in a new pod.**

Switch to the application tab and put /health on the end of the url and hit return. This will display the amount of time that the new pod has been running, which will understandably be a small number.

**In order to experiment with the readiness probe it is necessary to switch off the liveness probe. This can either be done by changing the deployment configuration YAML definition using the web interface or by executing the following command line:**

```
oc set probe dc/node-app-slave --liveness --remove
```

## Readiness Probe

**To create the probe use the OC command line interface to execute the following command.**

```
oc set probe dc/node-app-slave --readiness --initial-delay-seconds=30 --failure-
threshold=3 --success-threshold=1  --period-seconds=5 --get-url=http://:8080/health
```

**The above command will create a new readiness probe with the characteristics:**

- Become active after 30 seconds
- Remove the pod from the service endpoint after 3 instances of a failure to respond
- Cancel the removal of the pod and add it back to the service endpoint after 1 successful response
- Probe the application every 5 seconds
- Use the URL /health on the application at port 8080. Note that there is no need to specify a URL for the application.

**The command line response should be as shown below**

```
deploymentconfig.apps.openshift.io/node-app-slave probes updated
```

**Review the probe created using the commands above:**

```
oc describe dc/node-app-slave
```

and

```
oc get dc/node-app-slave -o yaml
```

**View the state of the pod within the endpoints using the command below:**

```
oc get ep/node-app-slave -o yaml
```

**The output of the above command will list the details of the service endpoint which will include information on the pod to which the health probe is attached as shown below.**

```
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    endpoints.kubernetes.io/last-change-trigger-time: 2019-11-26T16:04:50Z
  creationTimestamp: 2019-11-26T09:37:12Z
  labels:
    app: node-app-slave
    app.kubernetes.io/component: node-app-slave
    app.kubernetes.io/instance: node-app-slave
    app.kubernetes.io/name: nodejs
    app.kubernetes.io/part-of: master-slave
    app.openshift.io/runtime: nodejs
    app.openshift.io/runtime-version: "10"
  name: node-app-slave
  namespace: probes1
  resourceVersion: "1172051"
  selfLink: /api/v1/namespaces/probes1/endpoints/node-app-slave
  uid: 534139aa-1030-11ea-af1c-024039909e8a
subsets:
- addresses:
  - ip: 10.128.2.145
    nodeName: ip-10-0-136-74.eu-central-1.compute.internal
    targetRef:
      kind: Pod
      name: node-app-slave-5-hwj89
      namespace: probes1
      resourceVersion: "1172049"
      uid: ad6cc0e5-1043-11ea-af1c-024039909e8a
  ports:
  - name: 8080-tcp
    port: 8080
    protocol: TCP
```

The lies of interest above are:

```
subsets:
- addresses:
  - ip: 10.128.2.145
```

This shows that the address is currently part of the endpoint (it will participate in servicing requests) prior to the readiness probe activation.

## Activation of the Readiness Probe

Select the Topology view of the application.

Click on the arrow on the top right hand corner of the node icon to open the application URL in a new browser tab (unless you already have one open).

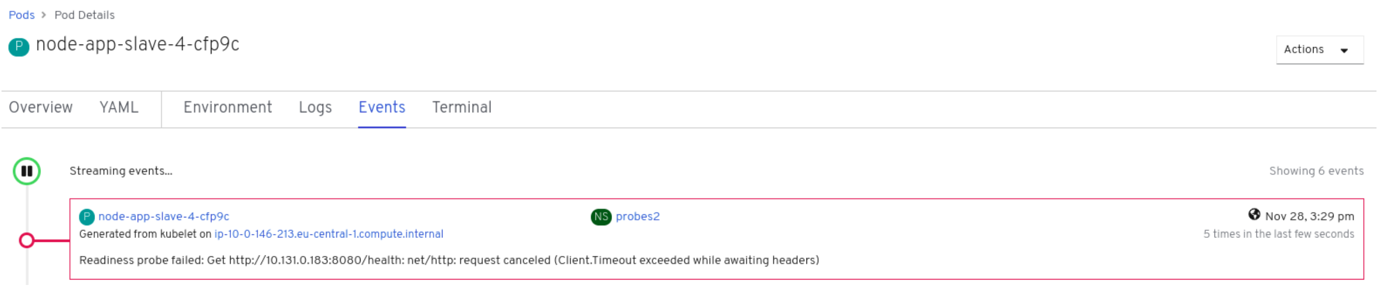
On the OpenShift browser tab, click on the pod to open the details window on the right hand side and then click on the pod link on the resources tab. This will display a multi-tab window with details of the pod, select the events tab.

Switch to the application tab and put /ip on the end of the url and hit return. This will display the ip address of the pod.

Change the url to have /health on the end and hit return. This will display the amount of time that the pod has been running.

Change the url to have /ignore on the end and hit return. Quickly move to the browser tab showing the pod events and watch for the probe event.

The pod events will show a screen similar to that which is shown below.



node-app-slave-4-cfp9c
probes2
Generated from kubelet on ip-10-0-146-213.eu-central-1.compute.internal
Nov 28, 3:29 pm
5 times in the last few seconds

Readiness probe failed: Get http://10.131.0.183:8080/health: net/http: request canceled (Client.Timeout exceeded while awaiting headers)

**Note that you will see the count of the readiness *events* incrementing every 5 seconds.**

**You will also see that the events continue counting up since readiness probes do not stop firing just because the pod has been removed from the endpoint list. It is important that they continue to probe since the conditions may change and it may be appropriate to add the pod back into the endpoint list.**

View the state of the pod within the endpoints using the command below:

```
oc get ep/node-app-slave -o yaml
```

**The output of the above command will list the details of the service endpoint which will include information on the pod to which the health probe is attached as shown below.**

```
subsets:
- notReadyAddresses:
  - ip: 10.128.2.145
```

The subset of the command output shown above indicates that the address is now listed as 'not ready' and is not currently part of the endpoint.

**Under production use conditions for the application may change and the pod may recover from the inability to respond to the readiness probe. If this happens then it will be added back to the endpoint list.**

**To simulate this the Node application has a REST endpoint at /restore. Since the pod is currently not receiving communications from outside the cluster the call to the restore endpoint is done from within the pod command window.**

Switch to the OpenShift browser window that was showing the pod events.

**Note that you will see a large number of pod readiness probe failures while you were reading the notes.**

Select the terminal window tab and enter the command :

```
curl -k localhost:8080/restore
```

**You should see a response similar to that shown below (with a different IP address):**

```
"10.128.2.146 restore switch activated"sh-4.2$
```

View the state of the pod within the endpoints using the command below:

```
oc get ep/node-app-slave -o yaml
```

**You should see that the line of interest, previously shown above, has changed back to that shown below:**

```
subsets:
- addresses:
  - ip: <ip address of the pod>
```

**On the OpenShift browser page switch back to the events tab and you should see that the readiness probe failure count is no longer increasing.**

Finally, switch to the application browser page and change the URL to end in /health. You should see that the application has been running for some time (compared to the liveness probe that showed a restart had taken place) and it should be responding successfully to the health probe.

## Cleaning up

From the OpenShift browser window click on *Advanced* and then *Projects* on the left hand side menu.

In the triple dot menu next to your own project (ProbesX) select ‘Delete Project’ Type ‘ProbesX’ (where X is your user number) such that the Delete button turns red and is active.

Press Delete to remove the project.

# Camel K on Openshift

## Introduction

Apache Camel K is a lightweight integration framework built from Apache Camel that runs natively on Kubernetes and is specifically designed for serverless and microservice architectures.

Users of Camel K can instantly run integration code written in Camel DSL on their preferred cloud (Kubernetes or OpenShift).

We are going to use a new project to create our Camel K integrations

If you are not already logged on go to the UI URL and login as userx (x is the number provided to you) and password openshift. Open another tab in the browser and go to the terminal URL provided and again logon as userx with password openshift

## Create a new project

In the browser based terminal window, create a new project called *camelk-userx* where *x* is your allocated user number

```
oc project camelk-userx
```

## Camel K and the Operator Lifecycle Manager

### Camel K and the Operator Lifecycle Manager

Camel K uses the Operator Lifecycle manager, this means that new Custom Resource Definitions (CRDs) will be added to Openshift. These CRDs will extend the Openshift data model and allow Camel K to be managed using the standard ‘oc’ command. Installing operators requires a higher cluster privilage so the presenter will demonstrate the installation of the operator shortly.

Before installing Camel K make sure that the CRDs do not exist.

In the terminal window type

```
oc whoami
```

If the response from the command is not your user name. Type ‘oc login’ and enter your username (userx) and password ‘openshift’. Rather than using your username and password, you could also use the recommended solution in the web console overview.

In the terminal browser window, type:-

```
oc get integrations
oc get integrationplatforms
```

Both of these commands should return an **error**

***ADMIN only - Presenter demonstrates***

Explain both of the above CRDs and what they are used for in Camel K

Install Camel K:

If you are not already logged on go to the UI URL and login as ‘opentlc-mgr’ (Don’t forget to explain that this is a cluster admin function) and the admin password

Make sure you are on the ‘Administration View’ in the console

Go to *Operators/Operator Hub*

Search for ‘camel k’

Install for all *namespaces* in the cluster

Wait until the ‘Camel K Operator’ is shown as installed on the Operators/Installed Operators window

***ADMIN only - finished***

In the browser terminal window Type:

```
oc get integrations
oc get integrationplatforms
```

This time, both commands should return “no resources found”

Download the *Camel K* examples using the browser terminal :-

## Download and configure your environment

Create a directory to download the examples

type *mkdir camel*

type *cd camel*

Download the Camel K examples

```
wget https://github.com/pprosser-redhat/devex-camelk/raw/master/camelexamples/camel-k-examples-1.0.0-M3.tar.gz
```

Uncompress the file

```
tar -xf camel-k-examples-1.0.0-M3.tar.gz
```

To allow a developer to easily interact with an Openshift cluster, Camel K has it's own command line interface. Download the *Camel K cli* :-

In the same directory, type

```
wget https://github.com/pprosser-redhat/devex-camelk/raw/master/camelkcli/camel-k-client-1.0.0-M3-linux-64bit.tar.gz
```

Uncompress the file

```
tar -xf camel-k-client-1.0.0-M3-linux-64bit.tar.gz
```

Before we can create an integration, we need to add a Camel K *IntegrationPlatform*.

In the terminal window, type

```
oc apply -f   https://raw.githubusercontent.com/pprosser-redhat/devex-camelk/master/camelkconfig/integrationplatform.yaml
```

**You should now have all the pieces you need to start creating and deploying lightweight Camel Integrations to Openshift.**

## Deploy a Camel K Interation

Firstly, let's start by deploying a simple integration, type

```
./kamel run simple.groovy
```

The first time you deploy an Integration, it will take a few minutes. The operator manages all dependencies required by the Integration and downloads these from the Red Hat repository on demand. Once downloaded, the operator caches dependancies therefore subsequent deployments are significantly quicker.

You can see what’s going on :-

If you go to the console UI in the Browser, make sure you are switched to *Developer view*

Click on *Topology*

Click on *Simple* (this is your integration), and you can see what’s going on.

Once the pod has a dark blue ring around it, it is running

Click on *Resources*

Click on *Pods* → *simple-xxxxxxxx* (xxxxxxx is randomly generated)

Click on *Logs* to see the output from the integration

The integration is a simple timer that triggers every 1 second and writes to the log file.

In the Terminal Browser window type

```
oc get integrations
```

You should now see an integration called *simple* in the list.

In the Terminal browser window type

```
 oc describe integration simple
```

You’ll see the “Integration” CRD definition that includes the integration defined as groovy code.

Let’s make a change to the integration

In the browser terminal window

```
vi simple.groovy
```

You will see the text - *Hello Camel K from ${routeId}*

Change the text e.g. *Hello Camel K from ${routeId}. Added some more text*

To change the text, use the arrow gets to move to the text you want to change, press *i*, and type in the text. To save, press *esc* and type *:wq* followed by *enter*

Now, you need to deploy this integration to Openshift again to test

Type

```
./kamel run simple.groovy
```

If you are quick enough (you need to be really quick!), you'll see the integration doing a rolling deployment

Look at the log file again (as above) to see if the change has been deployed

## Deploy Camel K in Developer mode

**While the process of redeploying is simple, it isn't very developer friendly. The *kamel* cli has a developer friendly "hot deploy" mode that makes this experience much better**

Let's try it out :-

Delete the integration :- There are 2 ways you can do this in the Terminal Browser window (your choice) :-

Use the "kamel" cli

```
./kamel delete simple
```

Or

Use the Openshift cli

```
oc delete integration simple
```

This is the great thing about CRDs, you can use the normal Openshift cli to managed the custom data model (integrations in this case)

Lets deploy the integration in developer mode, type

```
./kamel run simple.groovy --dev
```

You will see the deployment phases logged on the screen, followed by the log outputting automatically from the integration pod, useful for a developer to see what’s going on

For the next exercise, you will need 2 terminal windows.

Copy the URL for the “browser terminal” and paste into a new browser tab

It should look something like :- https://xxxxxxxxxxxxx/terminal/session/1

**You will notice that the terminal window is also outputting the log, you need to create a new terminal session**

Change the url to a new session number e.g. https://xxxxxxxxxxxxx/terminal/session/2

In the new terminal, make sure you go back to the camel directory - if you followed the instructions, it will be called *camel*

Make another change to the text in “simple.groovy” by following the same instructions above

Once you have saved the changes, go back to the browser terminal tab outputting the log.

You should see that the changes have been automatically applied to the running integration, without the need to redeploy

**That’s a much better way of round trip development of integrations…**

Go back to the browser terminal that’s outputting the log, press ‘ctrl c’

Look at the Topology view in the Openshift console(or oc get integrations)

The integration should have been deleted -

**Just like a developer would see by pressing *ctrl c* on a Java application running on their laptop**

## Optional labs

Feel free to take a look at some of the other examples

Instructions on how to run the integrations are in the integration source code E.g. ./kamel run --name=rest-with-restlet --dependency=camel-rest --dependency=camel-restlet RestWithRestlet.java

The example above demonstrates Camel K deploying a Java based Camel Route that exposes a Restful service via an Openshift route - the Camel K operator does all the hard work for you

some of the examples might take a little longer to deploy as the operator will need to download more dependencies

To test the rest api, you need to know what the Openshift Route is

In the Developer UI, go back to the Topology view, and click on “rest-with-restlet” application

Click on resources, and you will see the http route at the bottom of the page

Click on the http link

You will get an error as the integration will only response to /hello. Append ‘/hello’ to the URI Now you should see a response