# How to access the Lab Environment

This Hands-on lab is designed to give you a short introduction into creating Apps on OpenShift.

## A. How to access the Lab Environment by cli

1. Each student is given access to an online terminal the link is available in the etherpad link.

   https://terminal-terminal.apps.cluster-mum-215c.mum-215c.example.opentlc.com/

2. The user id and password to login to the terminal is same as shared for the labs

3. We will be oc client to execute labs and same is installed as part of the online terminal.

## B. Accessing the OpenShift Web Console

4. You can connect to the OpenShift web UI by clicking the following link:

   http://console-openshift-console.apps.cluster-mum-215c.mum-215c.example.opentlc.com

5. Please accept the self-signed certificate if asked for

6. If you want to know if your Web browser is supported by OpenShift, please have a look at OpenShift Container Platform 4.x Tested Integrations

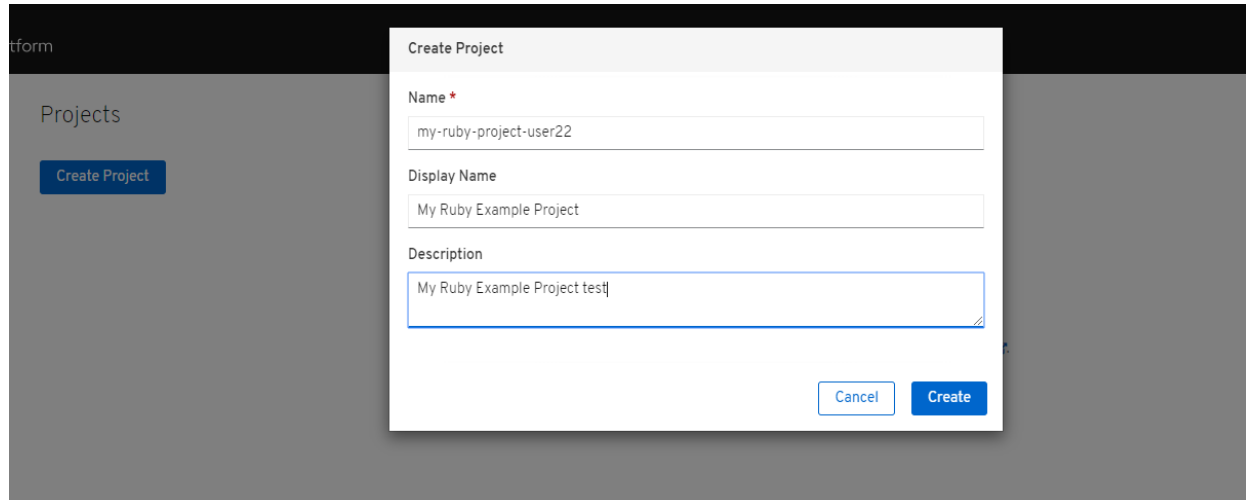# Lab 1. Creating Applications Lab

Deploy Application on Web Console
Here, you connect to and become familiar with the web console

Connect To and Explore Web Console

1. Use your browser to go to the OpenShift Web UI given on Etherpad

2. Log in as `userXX` with the password `openshift`.

# A. Create New Application From Source Code

1. Click the blue Create Project button in the top right corner.

2. Give the new project a name, display name, and description:

   a. Name: my-ruby-project-userXX

   b. Display Name: My Ruby Example Project

   c. Description: An explanation of your choice
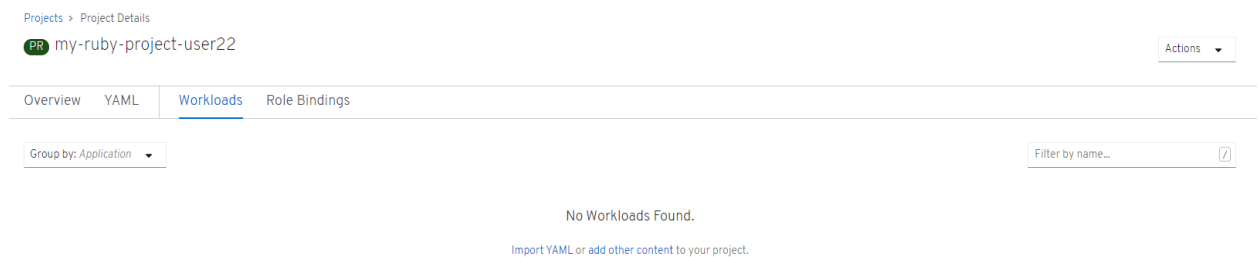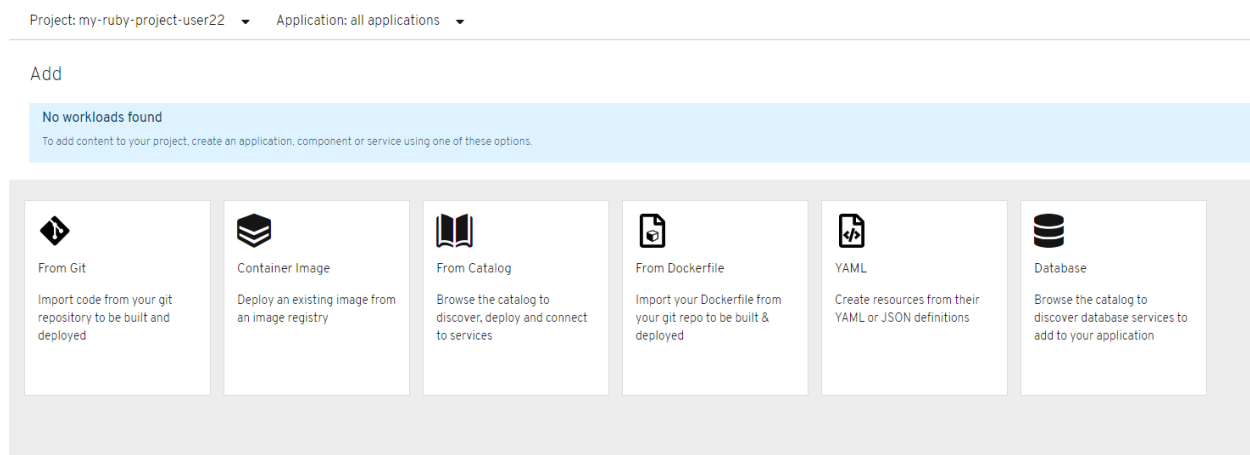


3. Click on the blue Create button.

Once the project is in place, click on your new project `My Ruby Example Project`

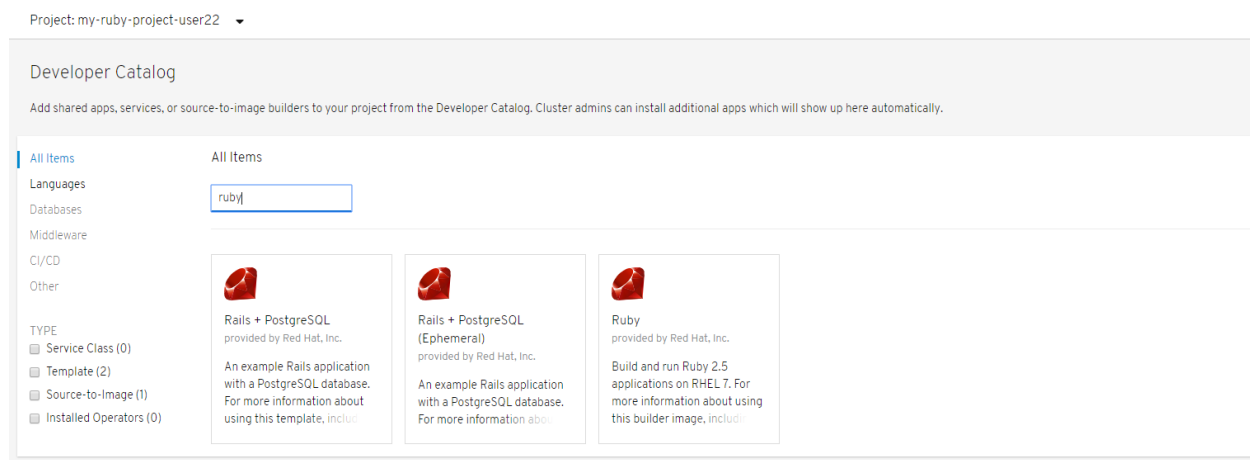4. In the **Project** screen, click on newly created project-`my-ruby-project. Click`
`on` **`Workloads`**

Projects > Project Details
PR my-ruby-project-user22                                                                          Actions ▾

Overview    YAML    Workloads    Role Bindings

Group by: Application ▾                                                          Filter by name... 🔍

No Workloads Found.

Import YAML or add other content to your project.

5. -> click on **add other content** hyperlink present at the bottom of the page .Select
**From Catalog** button to get the overview of all possible builder images.

Project: my-ruby-project-user22 ▾    Application: all applications ▾

Add

No workloads found
To add content to your project, create an application, component or service using one of these options.

| ◈ From Git | ⬢ Container Image | 📖 From Catalog | 📄 From Dockerfile | </> YAML | 🗄 Database |
|---|---|---|---|---|---|
| Import code from your git repository to be built and deployed | Deploy an existing image from an image registry | Browse the catalog to discover, deploy and connect to services | Import your Dockerfile from your git repo to be built & deployed | Create resources from their YAML or JSON definitions | Browse the catalog to discover database services to add to your application |

6. We search `Ruby` in the search box ->click on it  and

Project: my-ruby-project-user22 ▾

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.

All Items        All Items
Languages
Databases        ruby
Middleware
CI/CD
Other

TYPE
☐ Service Class (0)
☐ Template (2)
☐ Source-to-Image (1)
☐ Installed Operators (0)

| Rails + PostgreSQL | Rails + PostgreSQL (Ephemeral) | Ruby |
|---|---|---|
| provided by Red Hat, Inc. | provided by Red Hat, Inc. | provided by Red Hat, Inc. |
| An example Rails application with a PostgreSQL database. For more information about using this template, includ... | An example Rails application with a PostgreSQL database. For more information abo... | Build and run Ruby 2.5 applications on RHEL 7. For more information about using this builder image, includ... |

7. click on **Create Application**.

Ruby
Provided by Red Hat, Inc.

×

**Create Application**

Build and run Ruby 2.5 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-ruby-container/blob/master/2.5/README.md.

PROVIDER
Red Hat, Inc.

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

CREATED AT
🌐 Nov 10, 9:28 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.

8. Set the Builder Image Version* to `2.5`

9. Specify the name and Git repository URL:

    a. **Application Name**: `my-ruby-hello-world`

    b. **Git Repository URL**: https://github.com/openshift/ruby-hello-world

10. Click on **advanced options** and select the following options:

    a. Make sure the following option is checked(ticked): Create a route to the application. Exposes your application at a public URL.

    b. Change the scaling parameter to 3 replicas.

    c. Add a label with the name `environment` and the value of `dev`.

11. Click the blue **Create** button to create the application.

12. Click **Overview** to go to the application's **Overview** screen.

13. Go on **Workloads (on left panel)-> Pods->**Select Pod ->**Logs** to get logs to verify that a build is in progress.

14. Review the log as the build progresses.

15. Wait for the build to complete and go back to the overview page and click the blue external route link on the right side. It should look like `http://my-ruby-hello-world-my-ruby-project.apps-REPL.generic.opentlc.com`

    a. The database for our application isn't running, so expect to see the webpage mention that.

    You can also use the command line to create a new application: `oc new-app` `https://github.com/openshift/ruby-hello-world` `-l environment=dev`.

    To change scaling from the command line, use `oc scale`.

# B. Create New Application through an Image

1. Go on **Home** ->Sub menu **Projects**

2. Click the blue **Create Project** button in the top left corner.

3. Give the new project a name, display name, and description:

   ○ **Name**: `parksmap-userXX`

   ○ **Display Name**: `My Parksmap`

   ○ **Description**: An interactive map.

4. Click on the blue **Create** button.

Once the project is in place, navigate to your new project on **Home->Projects-> `Parksmap.`**

7. Select Workloads

8. Click on **add other content** hyperlink at the bottom of the page.

9. Select **Container Images** from the list of options .

10. Use **openshiftroadshow/parksmap-katacoda:1.0.0** as image name and click on the magnifying glass.

11. Select the following checkbox in the Advanced Options in the form - Create a route to the application.Exposes your application at a public URL

12. Just ensure Application Name and Name fields are auto filled and not left empty.

13. Click on Create.

14. After that you can click on the link to our new app. It looks like http://parksmap-katacoda-parksmap.apps-REPL.generic.opentlc.com.

These are all the steps you need to run to get a "vanilla" Docker-formatted image deployed on OpenShift. This should work with any Docker-formatted image that follows best practices, such as defining the port any service is exposed on, not needing to run specifically as the root user or other dedicated user, and which embeds a default command for running the application.

# C. Create New Application from binary

1. Make sure you are logged into Openshift with your cli oc client

```
$ oc login -u userX -p openshift  https://api.cluster-mum-215c.mum-
215c.example.opentlc.com:6443
```

2. Create a project for your user named "pipelineproject" in cli command line

```
 $ oc new-project binary-userXX
```

3. Create  build config for binary build

```
$ oc new-build java --name=java-binary-build --binary=true
```

4. Download the binary file to local directory

```
$  wget https://github.com/a-machado/labguide/blob/master/gs-spring-boot-0.1.0.jar
```

5. Trigger the build of the file using the local file

```
$ oc start-build bc/java-binary-build --from-file=gs-spring-boot-0.1.0.jar --follow
```

6. Trigger the actual pod deployment

```
$ oc new-app java-binary-build
```

7. Expose the route of the application to access from external

```
$ oc expose svc/java-binary-build
```

8. Now get the route of the application accesable outside platform

```
oc get route
```

9. Copy the route for the command and paste in a browser and check

# D. Templates Lab

You can create a list of objects from a template using the CLI or, if a template has been uploaded to your project or the global template library, using the web console. For a curated set of templates, see the OpenShift Image Streams and Templates library.

This lab includes the following sections:

- **Create and Upload Template**

  In this section, you create a template for a two-tier application (front end and database), upload it into the shared namespace (the `openshift` project), and ensure that users can deploy it from the web console.

- **Use Templates and Template Parameters**

  In this section, you create two separate template instances in two separate projects and establish a front-end-to-database-back-end connection by means of template parameters.

*Templates are a complex*

Templates allow an easy way to define all the required objects of a complex to be specified together and made available in Catalogs. Please see our OpenShift Documentation on Templates for more information.

## Create and Upload Template

### Install Template

The example in this section shows an application and a service with two pods: a front-end web tier and a back-end database tier. This application uses auto-generated parameters and other sleek features of OpenShift Container Platform. Note that this application contains predefined connectivity between the front-end and back-end components as part of its YAML definition.

This example is, in effect, a "quick start" — a predefined application that comes in a template and that you can immediately use or customize.

1. If not already, please login with `oc` as `admin` and change into the `default` project.

```
oc login https://master.example.com:8443 -u admin
```

2. Now we create a new project for the template based instant-app with appropriate node-selector label:

```
oc adm new-project instant-app-userXX --display-name="instant app example
project" --description='A demonstration of an instant-app/template' --
admin=userXX
```

```
Created project instant-app
```

3. Download the template's definition file:
   ```
   $ wget http://www.rhpet.de/ocp-workshop/Template_Example.yml
   ```
4. Change the "node-selector" label in DeploymentConfig section of front-end and database of Template_Example as mentioned in the last command.
5. Have a look into the template file:
   ```
   $ cat Template_Example.yml
   ```

```
---
kind: Template
apiVersion: v1
metadata:
 name: a-quickstart-keyvalue-application
 creationTimestamp:
 annotations:
   description: This is an example of a Ruby and MySQL application on
OpenShift 3
   iconClass: icon-ruby
   tags: instant-app,ruby,mysql
objects:
- kind: Service

...
```

6. Create the template object in the instant-app project. This is also referred to as *uploading* the template.

```
$ oc create -f Template_Example.yml -n instant-app
```

```
template "a-quickstart-keyvalue-application" created
```

> The `Template_Example.yml` file defines a template. You just added it to the instant-app project. If you add the template to the `openshift` project, it made your template available throughout your OpenShift cluster for all projects.

7. The OpenShift Container Platform comes with a long list of preconfigured templates available for usage. You can take a look at the installed list with the following `oc` command. This list over 100 entries, that is why we did not include the output here.

```
$ oc get templates -n openshift
```

```
NAME                                                DESCRIPTION
PARAMETERS        OBJECTS
3scale-gateway                                      3scale API Gateway
15 (6 blank)        2

... <many lines> ...

sso71-postgresql-persistent                         Application template
for SSO 7.0 PostgreSQL applications with persistent storage    33 (17
blank)        9
```

8. Look for your generated template in your `instant-app` project:

```
# oc get templates -n instant-app
```

```
NAME                                       DESCRIPTION
PARAMETERS          OBJECTS

a-quickstart-keyvalue-application    This is an example of a Ruby
and MySQL application on OpenShift 3    5 (4 generated)    8
```

Do not be alarmed by the complexity of Templates. You can even create templates from existing Objects. Please see our Documentation on How to Create a Template from existing Objects.

## Create Instant App from Template

1. On your browser, connect to the OpenShift web console as `student` with the password `openshift`, if not already.

2. If you are not on the welcome screen, please click on the `OpenShift Container Platform` banner at the top left side.

3. Click on the link for the `instant app example project` at the top right.

4. From the `instant app example project` project's **Overview** screen, click **Browse Catalog**.

   Here you find the instant applications, a special kind of template with the `instant-app` tag. The idea behind an instant application is that, when you create a template instance, you already have a fully functional application. In this example, your instant application is just a simple web page for key-value storage and retrieval.

5. Select **a-quickstart-keyvalue-application** and click on `Next`.

   The template configuration screen is displayed. Here, you can specify certain options for instantiating the application components:

   ○ Set the `ADMIN_USERNAME` to `admin`.

   ○ Set the `ADMIN_PASSWORD` parameter to your favorite password.

6. Click **Create** to instantiate the services, pods, replication controllers, etc. and then click on the `Continue to the project overview` link.

   ○ The build starts immediately.

7. Wait for the build to finish. You can browse the build logs to follow the progress.

   Our Application is currently still missing health checks for all containers. If you are an experienced OpenShift User feel free to build a template with health checks included.

## Use the Application

After the build is complete and both frontend and database are up and running, visit your application at the `Routes - External Traffic` from the frontend deployment config. It should look like: http://example-route-instant-app.apps-REPL.generic.opentlc.com

> Be sure to use HTTP and *not* HTTPS. HTTPS does not work for this example because the form submission was coded with HTTP links.
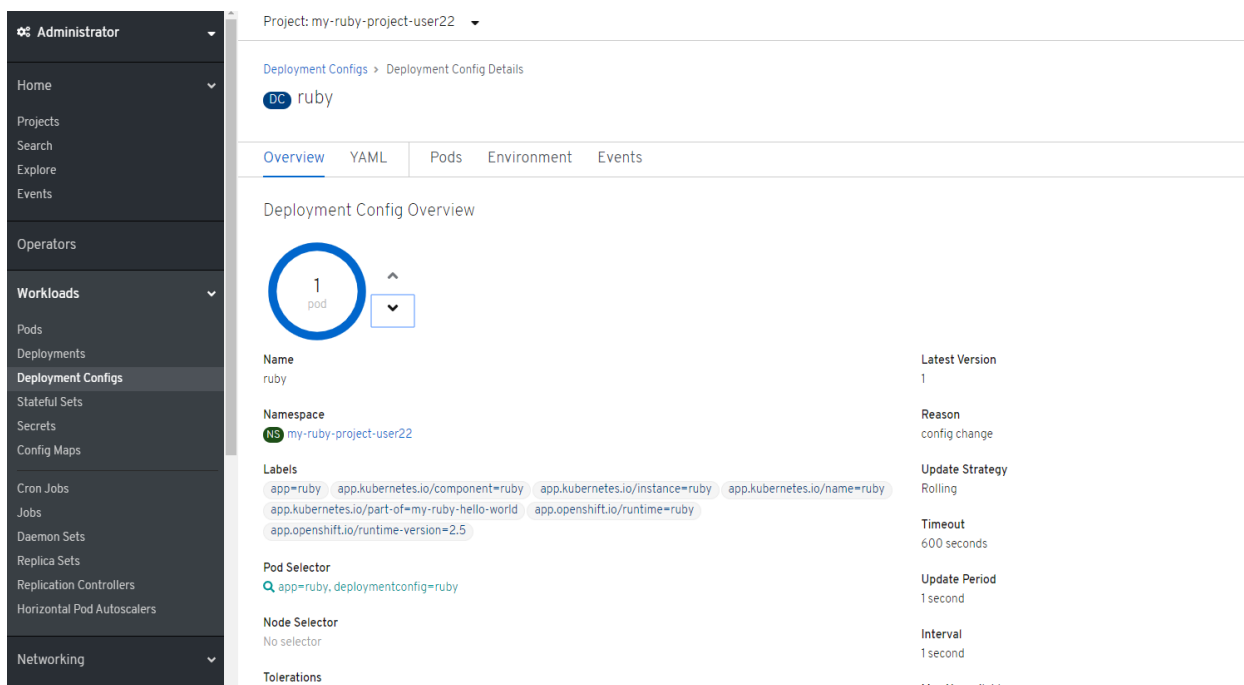
You see the same Frontend like in one of the labs before, but now with a running database :-)

For more informations about the powerful template function, please look into the template documentation.

# Lab 2. Service resilience and fault tolerance

## A. Scale Deployment

1. Go to Workloads->Deployment Configs ->Select my-ruby-hello-world-userXX

2. You would see a circle under Deployment Config Overview that shows the current number of pods, which is 1. You can increase that number by clicking the ^ button next to it.



3. Click the ^ button twice to increase the number of replicas to 5.

4. Click on the name of one of your running Pods.

5. Click on Logs to see the log for this Pod.

6. Click on Terminal to open a shell console inside the Pod right in your web browser.

7. Click on Events to see the last actions of this Pod.

8. Go back to your application's Overview screen by clicking Overview again.

## B. Service resilience and fault tolerance

1. Go to cli terminal

2. Login to oc using your userx credentials

   $oc login -u userXX -p openshift https://api.cluster-mum-215c.mum-215c.example.opentlc.com:6443

3. Go to the project from earlier example

   $ oc projects my-ruby-hello-world-userXX
4. Get the list of pods in the project

   $ oc get pods
5. Delete one of the Pods in the list

   $ oc delete pod < name of the pod >
6. Go to the web console

7. GO to Workloads -> Pods and see the new pods getting deployed

8. The Same exercise can be also performed from the Web console by selecting delete

   against the Pods listed

Pods

| | Create Pod | | | | | | Filter by name... | |
|---|---|---|---|---|---|---|---|---|

| 3 Running | 0 Pending | 0 Terminating | 0 CrashLoopBackOff | 5 Completed | 0 Failed | 0 Unknown | Select All Filters | 3 of 8 Items |
|---|---|---|---|---|---|---|---|---|

| Name ↑ | Namespace | Pod Labels | Node | Status | Readiness | |
|---|---|---|---|---|---|---|
| ⓟ java-binary-build-3-gdw8v | ⓃⓈ binary-user22 | app=java-binary-build deploym... =java-binary-buil... deploymentc... =java-binary-... | ip-10-0-160-96.ap-southeast-1.compute.internal | ⟳ Running | Ready | ⋮ |
| ⓟ java-binary-build-3-gvg52 | ⓃⓈ binary-user22 | app=java-binary-build deploym... =java-binary-buil... deploymentc... =java-binary-... | ip-10-0-139-119.ap-southeast-1.compute.internal | ⟳ Running | Ready | |
| ⓟ java-binary-build-3-x8q9g | ⓃⓈ binary-user22 | app=java-binary-build deploym... =java-binary-buil... deploymentc... =java-binary-... | ip-10-0-154-243.ap-southeast-1.compute.internal | ⟳ Running | Ready | |

Edit Labels
Edit Annotations
Edit Pod
Delete Pod

# Lab 3. Multi Pod Applications and Managing Bindings

We will create a NodeJS Application supported by a MongoDB Database. Although all these steps can be accomplished through the CLI we will use the Web Console this time.

The Service Catalog in Webconsole provides a fast an easy way to start new preconfigured Application Projects. First, we will provision a MongoDB Service consisting of the MongoDB Pod as well as a Secret. The Secret will contain the password for the database. Then we will provision a NodeJS Service and we will bind the Secret to the NodeJS Pod which will allow the application to access the database.

## 1. Provision MongoDB Database

*Provisioning Steps*

a.    Log as student into the Webconsole

b. Change to Developer mode from Administrator mode

c. Click For Catalog from +Add section  select Databases > MongoDB (Ephemeral) to

   provision a MongoDB without persistent storage.

d.  On the configuration Page fill out the following values:

e.  Project Name: multi-pod-project-userXX

f.  All other settings: Leave the default values

g.   Click Create

h.    Change to Administrator mode from Developer mode

Observe the deployment of the MongoDB Pod on the **Deployment Config** section.


*Verify the created Secret*

a.  From the **Workloads** section, navigate to **Secrets** tab.

b.  Select **mongodb** from the list.

c.  Click **View Secret** to show the Secret data

d.  Click **Reveal Values** to show the database-password
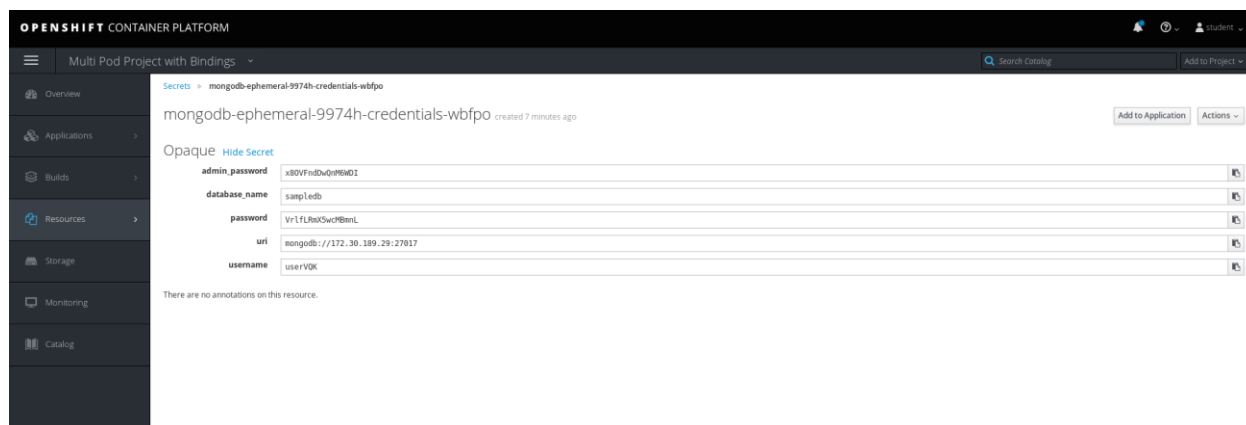
## 2. Provision NodeJS App

*Provisioning Steps*

a. Change to **Developer** mode from **Administrator** mode

b. Click **For Catalog** from **+Add** section  select Application > **Node.js** to provision a Node.js application.

c. Apply the following configuration:

    a. Builder Image Version: `10`

    b. Application Name: `multi-pod-app`

    c. Git Repository: `https://github.com/openshift/nodejs-ex.git`

d. Click **Create** & **Close**

**e.** Change to **Administrator** mode from **Developer** mode

**f.** Verify that the application pod for **multi-pod-app** is being created and is available by displaying a blue ring in **Deployment Configs**

**g.** Navigate to **Routes** tab to **Networking** section, click the URL entry of **location** column: `http://multi-pod-app-multi-pod-project.apps-REPL.generic.opentlc.com` which will take you to the application page

You should see a Message **No Database configured** under **Request Information section** since the application does not know how to access the MongoDB Pod

Look at the bottom right:

# Welcome to your Node.js application on OpenShift

## How to use this example application

For instructions on how to use this application with OpenShift, start by reading the Developer Guide.

## Deploying code changes

The source code for this application is available to be forked from the OpenShift GitHub repository. You can configure a webhook in your repository to make OpenShift automatically start a build whenever you push your code:

1. From the Web Console homepage, navigate to your project
2. Click on Browse > Builds
3. Click the link with your BuildConfig name
4. Click the Configuration tab
5. Click the "Copy to clipboard" icon to the right of the "GitHub webhook URL" field
6. Navigate to your repository on GitHub and click on repository settings > webhooks > Add webhook
7. Paste your webhook URL provided by OpenShift in the "Payload URL" field
8. Change the "Content type" to 'application/json'
9. Leave the defaults for the remaining fields — that's it!

After you save your webhook, if you refresh your settings page you can see the status of the ping that Github sent to OpenShift to verify it can reach the server.

Note: adding a webhook requires your OpenShift server to be reachable from GitHub.

Working in your local Git repository

## Managing your application

Documentation on how to manage your application from the Web Console or Command Line is available at the Developer Guide.

### Web Console

You can use the Web Console to view the state of your application components and launch new builds.

### Command Line

With the OpenShift command line interface (CLI), you can create applications and manage projects from a terminal.

## Development Resources

- OpenShift Documentation
- Openshift Origin GitHub
- Source To Image GitHub
- Getting Started with Node.js on OpenShift
- Stack Overflow questions for OpenShift
- Git documentation

## Request information

Page view count: No database configured

## 3. Add Database Binding

*Steps*

   a. Under **Deployment Config** click on **multi-pod-app**

   b. Click the **Environment** tab

   c. Click **Add Value from Config Map or Secret**

   d. Create four environment variables, which will map Environment Variables expected by the Node.js Pod to the Keys that have been created in the secret. As **resource** always select the **mongodb** & **mongodb- Secret** that you observed before.

| Name | Secret Key |
|---|---|
| MONGODB_USER | database-user |
| MONGODB_DATABASE | database-name |
| MONGODB_PASSWORD | database-password |
| MONGODB_ADMIN_PASSWORD | database-admin-password |

   e. Click Add Value to create a normal environment variable with Name : DATABASE_SERVICE_NAME and Value : mongodb . This will tell the App to retrieve the Database URL from the mongodb service resource.

   f. Click Save

Observe how the configuration change triggers a new deployment and how the apps page will now show a valid database connection.

Look at the bottom right again:

## Welcome to your Node.js application on OpenShift

### How to use this example application

For instructions on how to use this application with OpenShift, start by reading the Developer Guide.

### Deploying code changes

The source code for this application is available to be forked from the OpenShift GitHub repository. You can configure a webhook in your repository to make OpenShift automatically start a build whenever you push your code:

1. From the Web Console homepage, navigate to your project
2. Click on Browse > Builds
3. Click the link with your BuildConfig name
4. Click the Configuration tab
5. Click the "Copy to clipboard" icon to the right of the "GitHub webhook URL" field
6. Navigate to your repository on GitHub and click on repository settings > webhooks > Add webhook
7. Paste your webhook URL provided by OpenShift in the "Payload URL" field
8. Change the "Content type" to 'application/json'
9. Leave the defaults for the remaining fields — that's it!

After you save your webhook, if you refresh your settings page you can see the status of the ping that Github sent to OpenShift to verify it can reach the server.

Note: adding a webhook requires your OpenShift server to be reachable from GitHub.

#### Working in your local Git repository

If you forked the application from the OpenShift GitHub example, you'll need to manually clone the repository to your local system. Copy the application's source code Git URL and then run:

```
$ git clone <git_url> <directory_to_create>
```

### Managing your application

Documentation on how to manage your application from the Web Console or Command Line is available at the Developer Guide.

#### Web Console

You can use the Web Console to view the state of your application components and launch new builds.

#### Command Line

With the OpenShift command line interface (CLI), you can create applications and manage projects from a terminal.

### Development Resources

- OpenShift Documentation
- Openshift Origin GitHub
- Source To Image GitHub
- Getting Started with Node.js on OpenShift
- Stack Overflow questions for OpenShift
- Git documentation

### Request information

Page view count: 1

DB Connection Info:

    Type: MongoDB
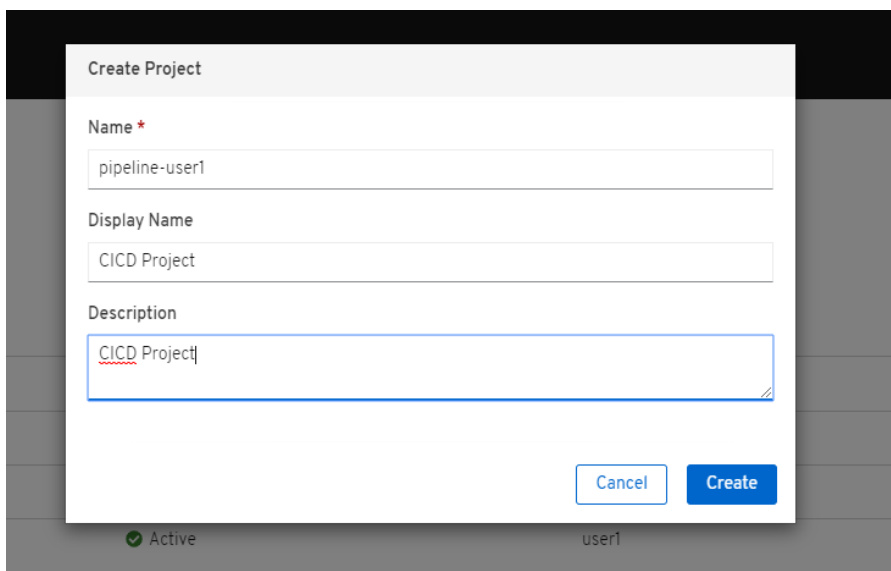    URL: mongodb://172.30.189.29:27017/sampledb

# Lab 4 : CI/CD Setup with Pipelines

CI/CD on OpenShift can be accomplished either by an internal or external CI-server such as Jenkins. In this example we will utilize the Jenkins image that is provided in the service catalog as well as the OpenShift Integration.

You will find an quick overview about CICD with OpenShift in our official OpenShift blog. But you do all needed steps here in the lab.

## A. Deploy and Configure the Internal Jenkins

9.  Login into the webinterface as `userXX`

10. Go on Home ->Sub menu  Projects

11. Click the blue Create Project button in the top left  corner.

12. Give the new project a name, display name, and description:

13. Enter configuration :

   a.  Add to Project: Select **Create Project**

   b.  Project Name: `pipelineproject-userXX`

   c.  Project Display Name: `CICD Project`

## 14. Click Workloads tab beside Role Bindings

Projects > Project Details

**PR** pipeline-user1

Overview    YAML    **Workloads**    Role Bindings

## 15. Click add other content

Projects > Project Details
**PR** pipeline-user1

Actions ▾

Overview    YAML    Workloads    Role Bindings

Group by: Application ▾

Filter by name...

No Workloads Found.

Import YAML or add other content to your project.

## 16. Click Select from catalog

Project: pipeline-user1 ▾    Application: all applications ▾

Add

**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

## 17. In the Service Catalog select CI/CD > Jenkins (Ephemeral)

Project: pipeline-user1 ▾

**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.

All Items
Languages
Databases
Middleware
| CI/CD
  Jenkins
Other

TYPE
☐ Service Class (0)
☐ Template (2)
☐ Source-to-Image (0)
☐ Installed Operators (0)

CI/CD

Filter by keyword...

**Jenkins**
provided by Red Hat, Inc.

Jenkins service, with persistent storage. NOTE: You must have persistent volumes available in your cluster to
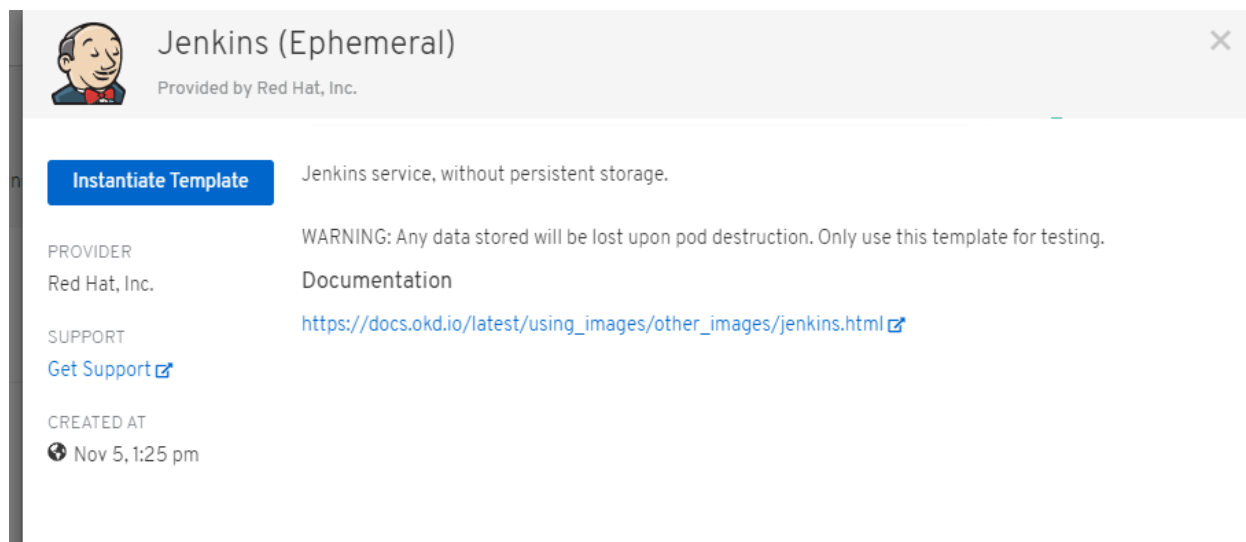
**Jenkins (Ephemeral)**
provided by Red Hat, Inc.

Jenkins service, without persistent storage. WARNING: Any data stored will be lost upon pod destruction. On

18. Click Instantiate Template



19. Click Create

20. On the navigation tab go to Deployment Configs under Workloads,



21. Open the Jenkins deployment configs and wait until Jenkins is fully up and running

22. Once Jenkins is up and running, go back to your project overview.

23. Click the route above the blue circle to open Jenkins.

24. You may need to accept the certificate to proceed. You are then presented with the Jenkins login screen.

25. Jenkins is OAuth enabled, simply click the Login with OpenShift button.

26. Login as userxx and password openshift.

27. Grant Jenkins to access to OpenShift resources by clicking Allow selected permissions.

28. After a short while you should see the main Jenkins Page.

# 29.     Setup Project

1. Make sure you are logged into Openshift with your cli oc client

```
a. $ oc login -u userX -p openshift  https://api.cluster-mum-215c.mum-
   215c.example.opentlc.com:6443
```

2. Create a project for your user named "pipelineproject" in cli command line

```
a.  $ oc new-project pipelineproject
```

3. Run this command to instantiate the template which will create a pipeline buildconfig and

   some other resources in your project:

```
a. oc new-app -f
   https://raw.githubusercontent.com/openshift/origin/master/examples/jenki
   ns/pipeline/samplepipeline.yaml
```

## 30.     Setup Build Pipeline

1.  Log in to your Jenkins

2.  On the left, click **New Item**

3.  Type **Tasks** for the item name, select **Pipeline** for the job type, and click **OK**

4.  At the bottom of the screen expect to see a multiline text box to write your pipeline in

5.  Paste the pipeline shown below into the text box:

```
try {
   timeout(time: 20, unit: 'MINUTES') {
      node('nodejs') {
         stage('build') {
           openshift.withCluster() {
              openshift.withProject() {
                 def bld = openshift.startBuild('nodejs-mongodb-example')
                 bld.untilEach {
                   return it.object().status.phase == "Running"
                 }
                 bld.logs('-f')
              }
           }
         }
         stage('deploy') {
           openshift.withCluster() {
             openshift.withProject() {
               def dc = openshift.selector('dc', 'nodejs-mongodb-example')
               dc.rollout().latest()
             }
           }
         }
      }
   }
} catch (err) {
   echo "in catch block"
   echo "Caught: ${err}"
   currentBuild.result = 'FAILURE'
   throw err
}
```

If you want to create or edit pipelines use the **Pipeline Syntax** generator accessible by the link below the textbox.

6. Click Save

7. Start a new Pipeline Build from Jenkins by clicking Build Now in the menu in the left

8. You can view and follow the build either from Jenkins or in OpenShift in the under `Builds` > `Builds`. After the Pipeline has completed you should have a new version of the tasks application up and running in the tasks-project.

# Lab 5 : Advanced Deployment Techniques

Depending on the type of application that you are deploying and the Quality of Service requirements in place, it is desirable to deploy frequent updates without any downtime. There are several deployments patterns that cater to these needs and we will show you how to execute these in OpenShift.

## A. Blue-Green Deployment

A Blue-Green deployment runs two different versions of an applications with a load balancer in front that can be used to switch traffic between the versions.

1. Create a new project

```
$ oc new-project strategies-userXX
```

2. Deploy 3 replicas of the hello-word app

```
$ oc run blue --image=openshift/hello-openshift --replicas=3
```

3. Set the environment variable inside the deployment config

```
$ oc set env dc/blue RESPONSE="Hello from Blue"
```

4. Expose the deployment internally in the cluster

```
$ oc expose dc/blue --port=8080
```

5. Exposes the application to be available outside the cluster under hello route

```
$ oc expose svc/blue --name=bluegreen
```

6. Create Green application

```
$ oc run green --image=openshift/hello-openshift --replicas=3
```

7. Set the environment variable inside the deployment config

```
$ oc set env dc/green RESPONSE="Hello from Green"
```

8. Expose the deployment internally in the cluster

```
$ oc expose dc/green --port=8080
```

9. Switch traffic to blue

```
$ oc set route-backends bluegreen blue=100 green=0
```

10. Test traffic distribution in a **new** shell (open a second ssh connection to the workstation host):

```
$ while true; do curl http://bluegreen-strategies.apps.cluster-demo-d92a.demo-d92a.example.opentlc.com/; sleep .2; done
Hello from Blue
Hello from Blue
Hello from Blue
Hello from Blue
Hello from Blue
Hello from Blue
...
```

11. Switch traffic to green and watch the other shell outputs:

```
$  oc patch route/bluegreen -p '{"spec":{"to":{"name":"green"}}}'
```

Now it should be all green.

12. You can play around with the numbers:

```
$ oc set route-backends bluegreen blue=50 green=50
```

If you want to know more about Blue-Green deployments on OpenShift, look [here](here).

## B. Canary Deployment

The canary deployment strategy means rolling out a new application version in small incremental steps.

1. Create a new project

```
$ oc new-project canary-userXX
```

2. Deploy 3 replicas of the hello-word app

```
$ oc run prod --image=openshift/hello-openshift --replicas=3
```

3. Set the environment variable inside the deployment config

```
 oc set env dc/prod RESPONSE="Hello from Prod"
```

4.  Expose the deployment internally in the cluster

```
oc expose dc/prod --port=8080
```

5. Exposes the application to be available outside the cluster under hello route
```
   $ oc expose svc/prod
```
6. Deploy the new version as canary

```
# oc run canary --image=openshift/hello-openshift
```

7. Set the environment variable inside the deployment config

```
# oc set env dc/canary RESPONSE="Hello from Canary"
```

8. Expose the deployment internally in the cluster

```
 $ oc expose dc/canary --port=8080
```

9. Switch traffic to prod

```
oc set route-backends prod prod=100 canary=0
```

10. Send 10 % of traffic to the new application ( canary )

```
 oc set route-backends prod prod=90 canary=10
```

11. Test traffic distribution in a **new** shell :

```
# while true; do curl http://rolling-strategies.apps.cluster-
demo-d92a.demo-d92a.example.opentlc.com/; sleep .2; done

Hello from Canary
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Canary
Hello from Prod
Hello from Prod
Hello from Prod
Hello from Prod


...
```

## C. Rolling Deployment

Rolling deployment is the default strategy which scales the old application version down while scaling the new application version up

1. Deploy the new version.

```
oc run rolling --image=openshift/hello-openshift --replicas=3
```

2. Expose the deployment internally in the cluster

```
# oc expose dc/rolling --port 8080
```
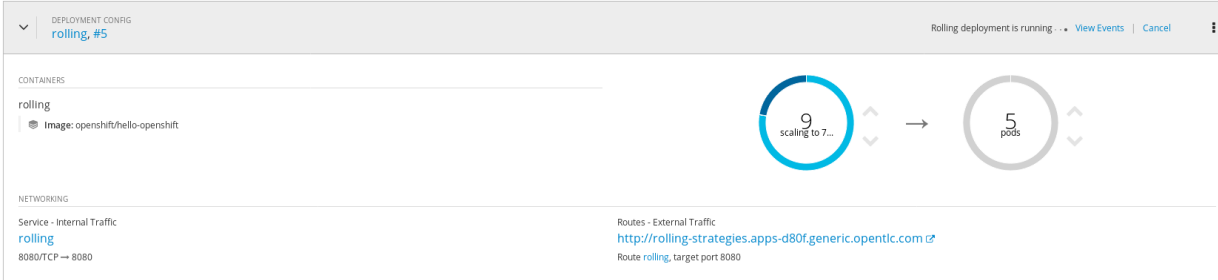
3. Exposes the application to be available outside the cluster

```
$oc expose svc/rolling
```

4. Trigger new deployment.

```
$ oc set env dc/rolling RESPONSE="Hello from new roll"
```

5. Check the rollout in the OpenShift Webconsole in the strategies project in the `Deployment Config` section from `rolling`.



6. Test traffic distribution in a **new** shell :

```
$ while true; do curl http://rolling-strategies.apps.cluster-
demo-d92a.demo-d92a.example.opentlc.com/; sleep .2; done
```

7.  Change the variable and test it again:

```
$ oc set env dc/rolling RESPONSE="Hello from second roll"
```

8.  Test traffic distribution in a new shell :

```
# while true; do curl http://rolling-strategies.apps.cluster-
demo-d92a.demo-d92a.example.opentlc.com/; sleep .2; done
```

If you want to know more about Advanced Deployment Strategies on OpenShift, have a look into the [here](#)

# The END

## Thanks for attending the OpenShift Hands-on lab

If you want more Information's about OpenShift, please see our YouTube Channel.

Please see OpenShift Introduction if you want more details of the architecture and components of OpenShift.