# Lab 1. Creating OpenShift projects

Learn how to login to an OpenShift cluster and create a new project in Minishift.

## 1. Login to the cluster

Login to the cluster with the output from the command after running `minishift start`.

```
$ oc login -u system:admin
```

If you get an error about the `oc` command not being found, you can source it with the following command:

```
$ eval $(minishift oc-env)
```

As you will be able to see, there are several projects available to be able to switch between different workloads.

## 2. Create a project

You should have a default project setup already but we will create a new project for our new application.

```
$ oc new-project nodejs-echo --display-name="nodejs" --description="Sample
Node.js app"
```

Now you should have a new project with the label `nodejs` and your active project will now point to it. If you want to switch between projects, run:

```
$ oc project <display-name>
```

Congratulations, you have logged into your cluster and have created your first OpenShift project! To learn how to create your first application move on to Lab 2.

# Lab 2: Creating OpenShift applications

After creating a project as instructed in the previous lab (Lab. 1) the next step is to create an OpenShift application in the cluster.

## 1. Creating an app

There are several ways in which you can create an app in OpenShift:

- From source code
- From DockerHub images
- From OpenShift templates

•From the OpenShift UI

## 1.1 Creating an app from source

With `oc new-app` command, you can create an application in OpenShift from some existing source code either locally or with the url to the repository. If a source repository is specified, `new-app` will check to see which build strategy to use (**Docker** or **Source**).

With the former, a runnable image is created, whereas as the latter, `new-app` will try to identify the language by looking at the files in the project's root directory and then use an appropriate builder.

To build from a local Dockerfile:

```
$ oc new-app /path/to/local/or/remote/Dockerfile
```

To build from source:

```
$ oc new-app path/to/local/or/remote/repository.git
```

## 1.2 Creating an app from a DockerHub image

Similar to Docker, OpenShift is also configured to the public image registry DockerHub. If you specify an image that exists in DockerHub, the `new-app` command will create a runnable image directly from this image.

For example, if you wanted to create an app from the official nginx image, you would run:

```
$ oc new-app nginx
```

You are not limited to the DockerHub registry, however - as with Docker, you are able to specify images that are stored in private registries too:

```
$ oc new-app myregistry:8000/example/image
```

## 1.3 Creating an app from an OpenShift template

OpenShift templates are basically starter applications that have been configured ready for OpenShift. These cover frequently used applications deployed in containers e.g. Ruby, Node and MongoDB.

The nodejs template looks as follows:

```
nodejs-ex
├── openshift
│   └── templates
│       ├── nodejs.json
│       ├── nodejs-mongodb.json
│       └── nodejs-mongodb-persistent.json
├── package.json
├── README.md
├── server.js
├── tests
│   └── app_test.js
└── views
    └── index.html
```

To deploy it, you can run:

```
$ oc new-app -f /path/to/nodejs.json
```

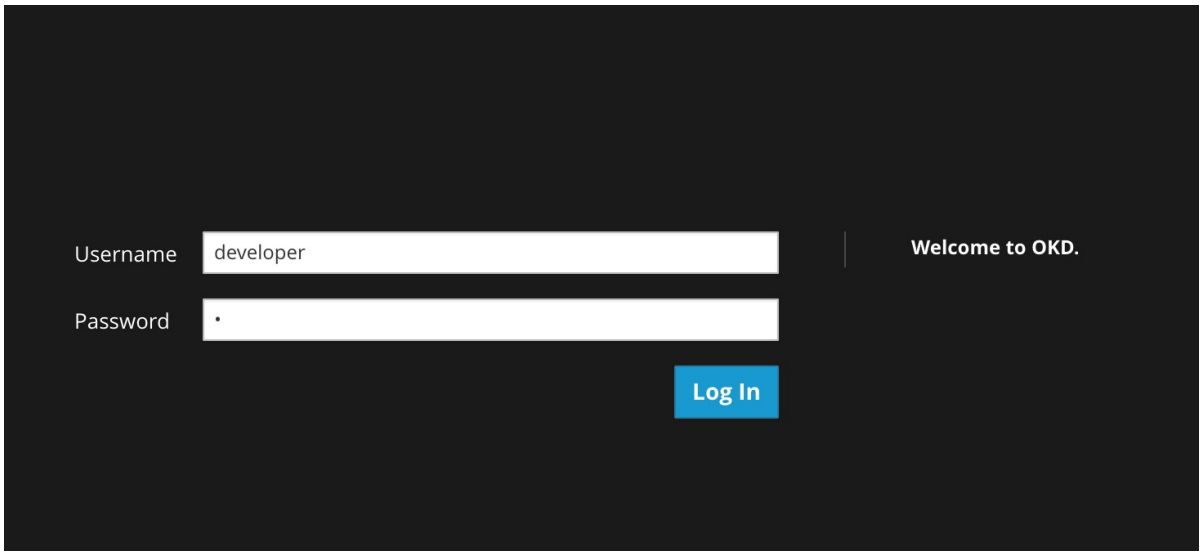
As this template lives in a repo, we could have also run this from source as described in section 1.1

## 1.4 Creating an app from the OpenShift UI

If you're not a fan of the cli and wanted a more visual way of deploying applications in your cluster, you also have the option of using the OpenShift console. This is available locally at the address given after running `minishift start` as we did in the setup:

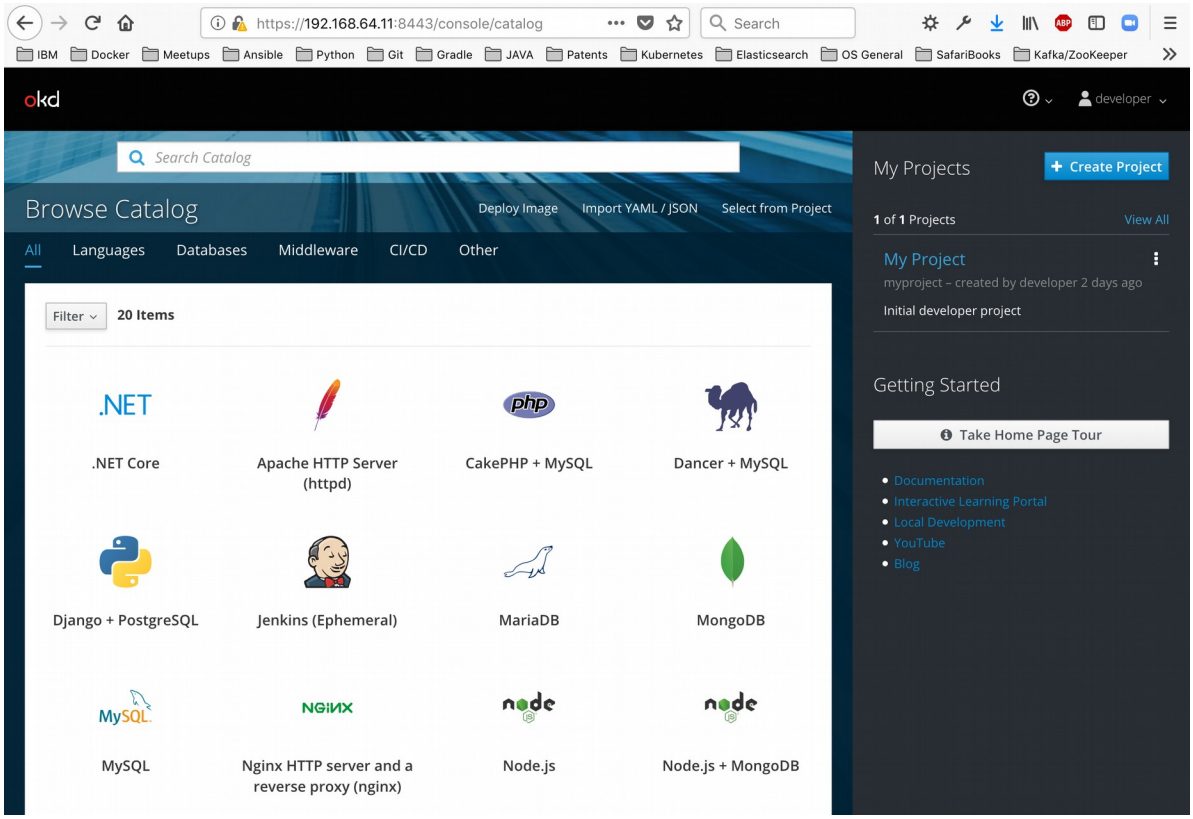
```
$ minishift start
...

The server is accessible via web console at:
    https://192.168.64.11:8443/console

You are logged in as:
    User:    developer
    Password: <any value>
```

Login to the UI:




As mentioned in the `minishift start` output, you can use the user *developer* and password as any string of characters (at least one) and you will be able to access the UI.

Access the catalog:




Once you login, you will be redirected to the browser catalog where there will be a number of sample applications available for you to chose to deploy. This mirrors the OpenShift template steps we saw in section 1.2. We can also create and switch between projects but note, you are limited to the provided sample applications available in the console catalog.

Congratulations! You have learnt several ways to create applications in OpenShift! To see how we can manage our applications in OpenShift, let's continue on to the next Lab (Lab 3).

# Lab 3: Managing OpenShift applications

After creating an OpenShift application as instructed in the previous lab (Lab 2), the next step is ensuring it is up and running and finding out how to access it.

## 1. Monitoring builds

When we run `oc new-app` a build is triggered similar to a `docker build`. Unlike `docker build` however, the build happens in the background and we don't get the output of what is happening.

To see this output, we can run:

```
$ oc status
```

Assuming we were building the ruby template we would run:

```
$ oc new-app https://github.com/sclorg/nodejs-ex
```

Then to check the status:

```
$ oc status
In project nodejs (nodejs-echo) on server https://192.168.64.11:8443

svc/nodejs-ex - 172.30.6.48:8080
  dc/nodejs-ex deploys istag/nodejs-ex:latest <-
    bc/nodejs-ex source builds https://github.com/sclorg/nodejs-ex on openshift/nodejs:10
      build #1 running for 36 hours
    deployment #1 waiting on image or update


2 infos identified, use 'oc status --suggest' to see details.
```

Looking at the output, we can see that we have a service endpoint to access our node app at `172.30.6.48:8080`. We can also see the build type used to create our application: `openshift/nodejs:10`. Finally, we can see in the last line that our application is not actually ready for usage as it is waiting for thr image to be deployed. In the line above, we can use the build number to get a more comprehensive idea of what is happening in the build.

If you wanted to check the build progress of a application, it would follow this format:

```
$ oc logs build/<app-name>-<build-no>
```

In this example, we would use the application name `nodejs-ex` and the build number `1`:

```
$ oc logs build/nodejs-ex-1
```

# 2. Monitoring deployments

In Kubernetes, we are able to the status of all our running applications. OpenShift has the same capability using the `oc` command:

```
$ oc get pod
NAME                READY     STATUS      RESTARTS   AGE
nodejs-ex-1-build   0/1       Completed   0          13m
nodejs-ex-1-hx6v9   1/1       Running     0          12m
```

Here we can see that a pod (`nodejs-ex-1-build`) was created with the sole purpose to run the build and then was cleaned up after completing successfully. We are then left with our running pod (`nodejs-ex-1-hx6v9`) containing our node application.

If we actually just wanted to see the *deployments*, the actual application and not the assoicated pods, we can run:

```
$ oc get dc
NAME        REVISION   DESIRED   CURRENT   TRIGGERED BY
nodejs-ex   1          1         1         config,image(nodejs-ex:latest)
```

Another useful command in Kubernetes is the ability to check the services associated with our applications if any. We do this in Kubernetes by running `kubectl get svc` and can do a similar thing in OpenShift:

```
$ oc get svc
```

```
NAME        TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
nodejs-ex   ClusterIP   172.30.6.48   <none>        8080/TCP   14m
```

As with Kubernetes, each application can be given an internal IP in which we can access our services but unless stated otherwise, this will only be available within the cluster.

Congratulations! You have learnt how to monitor your application builds and deployments within your cluster! To see how we can expose our applications outside of the OpenShift cluster, let's continue on to the final Lab (Lab 4).

# 4. Exposing OpenShift applications

Once it has been verified that the application is up and running as instructed in the previous lab (Lab 3), the next step is to configure access for the application outside of the cluster. There are several ways to do this:

- Node-port services
- Port-forwarding
- Routes

## 4.1 Node port services

This is the cleanest way to access the applications outside of OpenShift environment both locally and publicly. This way essentially makes use of the cluster node's IPs and a port in between the range (30000-32767) and tells OpenShift to proxy to the underlying application via. the port. This is better than the next two solutions for several reasons: we don't have to worry about port clashes, this works for non HTTP based services and finally, does not require a public host name.

To expose our deployment via NodePort, we simply expose the deployment with a *load balancer* type and label it with name *nodejs-ex-ingress*:

```
$ oc expose dc nodejs-ex --type=LoadBalancer --name=nodejs-ex-ingress
service/nodejs-ex-ingress exposed
```

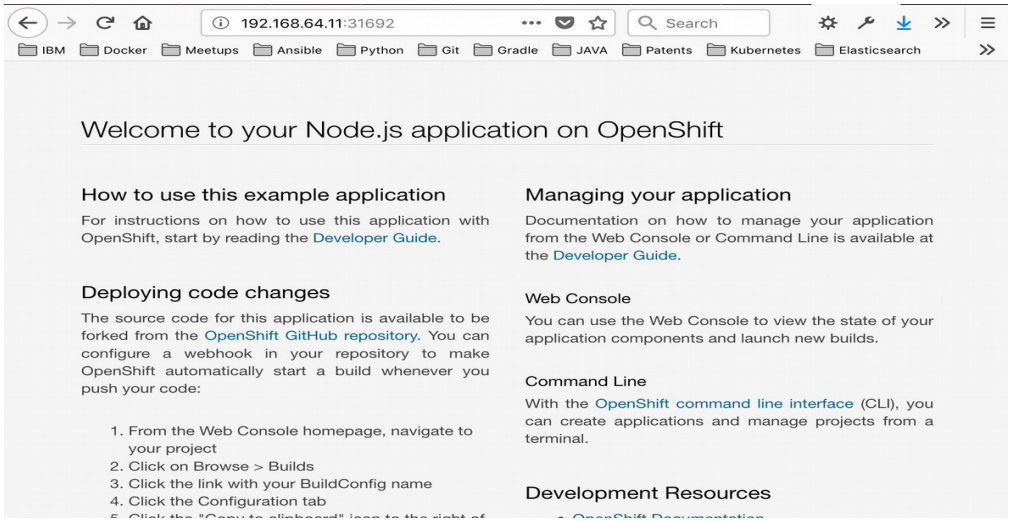
To see the NodePort created, we can run:

```
$ oc get --export svc nodejs-ex-ingress
NAME                TYPE           CLUSTER-IP   EXTERNAL-IP    PORT(S)
AGE
nodejs-ex-ingress   LoadBalancer   <none>       172.29.51.89   8080:31692/TCP
<unknown>
```

We can use the NodePort in conjuction with the cluster's internal or external IP which we can find in the following command:

```
$ oc get node -o wide
NAME        STATUS    ROLES     AGE       VERSION           INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION               CONTAINER-
RUNTIME
localhost   Ready     <none>    13h       v1.11.0+d4cacc0   192.168.64.11
<none>        CentOS Linux 7 (Core)   3.10.0-957.5.1.el7.x86_64
docker://1.13.1
```

We should then be able to access the application in the browser. In this example, we can access the Node application at `192.168.64.11:31692`:




# 4.2 Port-forwarding

Alternatively, if you want to quickly access a port of a specific pod of your cluster, you can also use the oc `port-forward` command:

```
$ oc port-forward POD [LOCAL_PORT:]REMOTE_PORT
```

# 4.3 Routes

For web applications, the most common way to expose it is by a route. A route exposes the service as a host name. You can do this by running the command providing you have a host name available:

```
$ oc expose svc/nodejs-ex --hostname=www.example.com
```

Congratulations! You have completed all labs in this workshop! You have learnt how to:

- Create an OpenShift project
- Create an OpenShift application in various ways
- How to monitor the status of an application
- How to access and expose your application

For more information on how to navigate Minishift, check the Minishift docs.