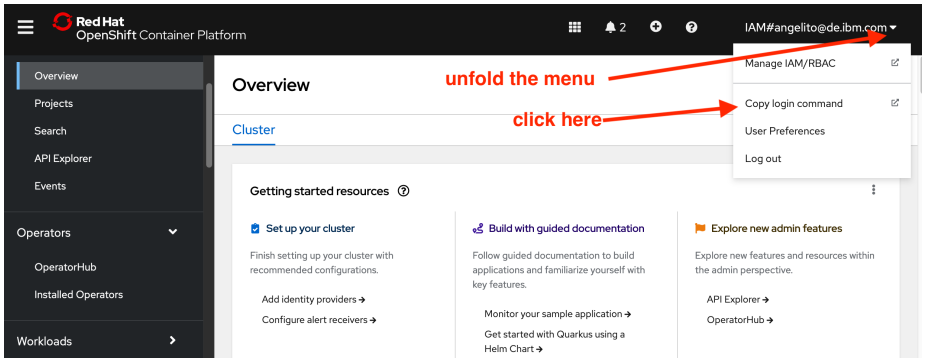

# Deploying Databand

## 1. Preparing the environment

### 1.1 Logging to the cluster

First of all, we need a way to login to the custer from the command line. Open the Openshift Console and proceed as follows:




After clicking on `Copy login command` a new tab displays the contents of the command you need to copy to the clipboard


You need to paste the contents of the clipboard into the next cell

```
# Replace the command with your own one inside the single quotes and run the cell
# Example OC_LOGIN_COMMAND='oc login --token=sha256~3bR5KXgwiUoaQiph2_kIXCDQnVfm_HQy3YwU2m-
OC_LOGIN_COMMAND='oc login --token=sha256~e8tzZje9glRmHCZEftW7EAXr3FnP_TpYHTGjRq8p7po --serv
```

Now, you can test that it works by executing the next cell:

```
echo $OC_LOGIN_COMMAND
$OC_LOGIN_COMMAND
```

Please remember this command as you may need to issue it again if, for example, you leave the jupyter session.

**Warning:** The token will expire and changes after some time. Don't be surprised if you need to repeat the same thing tomorrow for logging into the cluster again

## 1.2 Expose the registry

We will need to upload some docker images to the cluster image registry but, by default, it is not accesible after the provisioning. So, we will expose it with this command:

```
# This command exposes the registry
oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec":{"defaultRoute
```

To verify if the registry is really exposed now, run the following command:

```
# This command check if there is a network route associated the the image registry
oc get route -n openshift-image-registry
```

If something like "No resources found" is displayed, the patch command did not work. Stop here and fix it before continuing.

Now, we need to select the actual route out of this complicated output. Just run this:

```
# This command extracts the portion of the registry route
# Example: default-route-openshift-image-registry.itzroks-1100005cc8-4b30y2-4b4a324f027aea1
REGISTRY_URL=$(oc get route -n openshift-image-registry | grep default-route | awk '{print $
echo $REGISTRY_URL
```

## 2. Preparing Docker images

### 2.1 Loading images locally

In the last part of the of the prerequisites section we downloaded the file `databand-1.0.19-helm-chart.tar.gz` (960MB). After decompressing it, we see something like this:

```
(base) Angel@AngelMac databand-1.0.19-helm-chart % ls -l
total 5050480
-rw-r--r--@ 1 Angel  staff      318291 13 feb 14:07 databand-1.0.19-7.tgz
-rw-------@ 1 Angel  staff  2579398656 13 feb 14:09 databand-v1.0.19.7-images.tar`
```

Our task now is importing the databand images in the reqistry of our local computer. First, let's define the directory where these two files are located after decompressing the databand package file:

```
# Copy the directory where you decompressed the databand package into the clipboard and pas
# Example: DATABAND_UNCOMPRESSED_DIR='/Users/Angel/Downloads/databand-1.0.19-helm-chart'
# Then, run this cell:
DATABAND_UNCOMPRESSED_DIR='paste_here_the_clipboard'
echo $DATABAND_UNCOMPRESSED_DIR
```

```
# Run this cell. It will produce a very long output
cd $DATABAND_UNCOMPRESSED_DIR
ls -l
DATABAND_IMAGES=$(ls databand*images.tar)
docker load -i $DATABAND_IMAGES
```

Run this command to verify that the load went well:

```
docker images
```

### 2.2. Tagging images

Now, we need to issue very cumbersome commands. Pay attention to the following steps.

First, we need to ensure that we know the route to the registry. We set the variable `REGISTRY_URL` in a previous cell where we extracted just the route of the registry. We can run it once again to be sure that it is set and remember the manual circumvention as explained here in case that the output is different from the expected.

```
# This command extracts the portion of the registry route. We did it before but you can run
# Example of expected output: default-route-openshift-image-registry.itzroks-1100005cc8-4b3
REGISTRY_URL=$(oc get route -n openshift-image-registry | grep default-route | awk '{print $
echo $REGISTRY_URL
```

Then, we need to retrieve our actual userid and password in the OpenShift cluster and log into the registry

```
# Run this cell. It is necessary to login to the registry (apart from being logged into the
REGISTRY_USER=$(oc whoami)
echo $REGISTRY_USER
REGISTRY_PASS=$(oc whoami -t)
echo $REGISTRY_PASS

docker login -u $REGISTRY_USER -p $REGISTRY_PASS $REGISTRY_URL
```

Now, we create a project in the cluster. Name it `databand` just for simplicity

```
# Run this cell. It will create an OpenShift project, which is equivalent to a namespace in
DATABAND_PROJECT=databand
oc new-project $DATABAND_PROJECT
```

Finally, we have everything to tag our container images. Please review the output of this command:

```
# Run this command to check that we will produce correct tags
# it only echoes the commands but does not issue them
docker images | grep 'dbnd-' | while read a b c
do
   my_image=$(echo ${a}:${b} | awk -F"/" '{print $NF}')

   echo docker tag ${a}:${b} $REGISTRY_URL/$DATABAND_PROJECT/$my_image
done
```

If you see a series of commands with this format...

```
docker tag _image-in-local-registry_  _registry-route_/databand/_image_name_
```

...you can then run safely this cell:

```
# Run this cell to tag the databand images
docker images | grep 'dbnd-' | while read a b c
do
   my_image=$(echo ${a}:${b} | awk -F"/" '{print $NF}')

   docker tag ${a}:${b} $REGISTRY_URL/$DATABAND_PROJECT/$my_image
done
```

### 2.3 Pushing images

If the last command worked fine, run this one as well but now be more patient. It will upload the local images to the OpenShift cluster

```
# Run this cell to push the databand images. Expect a long output
docker images | grep 'dbnd-' | while read a b c
do
   my_image=$(echo ${a}:${b} | awk -F"/" '{print $NF}')

   docker push $REGISTRY_URL/$DATABAND_PROJECT/$my_image
done
```

## 3. Helm Chart deployment

We unpacked the databand media package in a previous step to load the container images (section 2.1). Let's go to that working directory again:

```
# Copy the directory where you decompressed the databand package into the clipboard and pas
# Example: DATABAND_UNCOMPRESSED_DIR='/Users/Angel/Downloads/databand-1.0.19-helm-chart'
# Then, run this cell:
DATABAND_UNCOMPRESSED_DIR='paste_here_the_clipboard'
echo $DATABAND_UNCOMPRESSED_DIR
```

You can verify that there is a big file with the docker images and a smaller one with the helm charts

```
cd $DATABAND_UNCOMPRESSED_DIR
pwd
ls -l
```

Now, lets unpack the helm charts and we will see that the new subdirectory `databand`

```
tar -zxf databand-1.0.19-7.tgz
ls -l
```

Go to this directory and have a look at the list of files.

```
cd databand
pwd
ls -l
```

We will need to manipulate just one of them but, before that, let's generate some secrets:

### 3.1 Generate secrets

We need to generate two internal passwords (secrets) for our deployment and we will ensure that the format will comply with the expectations. Additionally, we will retrieve other variables that we will use in the next steps. Run the

following cell and take a look at the output values because you may need to copy-and-paste them later.

```
# Run this cell

# these are the secrets
export FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | openssl base64)
export WEBSERVER_KEY=$(head -c 32 /dev/urandom | base64 | tr -d =)

# these are other values we retrieved before but we get them once again to be sure that the
export DATABAND_PROJECT=databand
export REGISTRY_USER=$(oc whoami)
export REGISTRY_PASS=$(oc whoami -t)
export REGISTRY_URL=$(oc get route -n openshift-image-registry | grep default-route | awk '

# this is simply to retrieve the version number (image tag)
export IMAGE_TAG=$(docker images | grep dbnd-webserver | head -1 | awk '{print $2}' | sed 's

# this is just to verify that the variables are not empty. You may need these values later
echo DATABAND_PROJECT=$DATABAND_PROJECT
echo REGISTRY_USER=$REGISTRY_USER
echo REGISTRY_PASS=$REGISTRY_PASS
echo REGISTRY_URL=$REGISTRY_URL
echo IMAGE_TAG=$IMAGE_TAG
echo FERNET_KEY=$FERNET_KEY
echo WEBSERVER_KEY=$WEBSERVER_KEY
```

### 3.2 Customize the deployment file

Now, we will change the file `user-values.yaml` to customize our deployment. You need to choose one of two options:

- a. manual edit
- b. automatic edit

It may be a good idea to start with the option a. but if you find it too complicated try b.

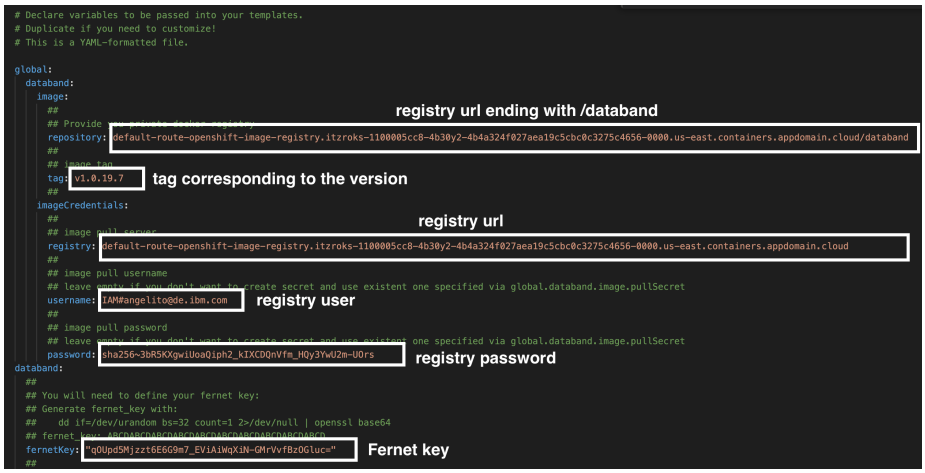
**3.2.a Manual edit of `user-values.yaml`** Check that you are in the right directory, i.e. containing the contents of the helm charts

```
cd $DATABAND_UNCOMPRESSED_DIR/databand
pwd
ls -l
```

Create the file `user-values.yaml` by copying `user-values.yaml.example`

```
cp user-values.yaml.example user-values.yaml
ls -l user-values*
```

Edit the file `user-values.yaml` end ensure that it looks like this (use the prover values for your environment that we retrieved before):




Save the file and you are done. Skip 3.2.b if you are fine with the result.

**3.2.b Automatic edit of user-values.yaml** If you don't want to edit the `user-values.yaml` file manually, the following cells will do the work for you.

First, you need to install a small utility that changes yaml files from the command line. It is called `yq`

```
# Install yq if you want to edit the user-values.yaml automatically
# use brew on MacOS
# alternatively, you may need to use apt, yum, snap or simply download the binary.
# go here for instructions https://github.com/mikefarah/yq
```

```
brew install yq
```

Ensure that you are in the directory where the helm charts were unpacked and create the file `user-values.yaml` from the example

```
# Run this cell to create a user-values.yaml file
```

```
cd $DATABAND_UNCOMPRESSED_DIR
cd databand
cp user-values.yaml.example user-values.yaml
pwd
ls -l user-values.yaml
```

This file contains some default values that we will need to modify. This is how they look like now:

```
# These commands display the values that we will edit automatically
```

```
echo repository=$(yq '.global.databand.image.repository' user-values.yaml)
echo tag=$(yq '.global.databand.image.tag' user-values.yaml)
echo registry=$(yq '.global.databand.imageCredentials.registry' user-values.yaml)
echo username=$(yq '.global.databand.imageCredentials.username' user-values.yaml)
echo password=$(yq '.global.databand.imageCredentials.password' user-values.yaml)
echo fernetKey=$(yq '.databand.fernetKey' user-values.yaml)
echo webKey=$(yq '.web.secret_key' user-values.yaml)
echo datastage=$(yq '.dbnd-datastage-monitor.enabled' user-values.yaml)
```

The following cell perfoms the actual edit. Now, you understand why we retrieved and exported some variables above.

```
# Change the file user-values.yaml

yq -i '.global.databand.image.tag = strenv(IMAGE_TAG)' user-values.yaml
yq -i '.global.databand.imageCredentials.registry = strenv(REGISTRY_URL)' user-values.yaml
yq -i '.global.databand.imageCredentials.username = strenv(REGISTRY_USER)' user-values.yaml
yq -i '.global.databand.imageCredentials.password = strenv(REGISTRY_PASS)' user-values.yaml
export YAML_REGISTRY=$REGISTRY_URL/$DATABAND_PROJECT
yq -i '.global.databand.image.repository = strenv(YAML_REGISTRY)' user-values.yaml
yq -i '.databand.fernetKey = strenv(FERNET_KEY)' user-values.yaml
yq -i '.web.secret_key = strenv(WEBSERVER_KEY)' user-values.yaml
yq -i '.dbnd-datastage-monitor.enabled = true ' user-values.yaml
```

If you wish to review the changes, run the following cell and compare the values.

```
echo repository=$(yq '.global.databand.image.repository' user-values.yaml)
echo tag=$(yq '.global.databand.image.tag' user-values.yaml)
echo registry=$(yq '.global.databand.imageCredentials.registry' user-values.yaml)
echo username=$(yq '.global.databand.imageCredentials.username' user-values.yaml)
echo password=$(yq '.global.databand.imageCredentials.password' user-values.yaml)
echo fernetKey=$(yq '.databand.fernetKey' user-values.yaml)
echo webKey=$(yq '.web.secret_key' user-values.yaml)
echo datastage=$(yq '.dbnd-datastage-monitor.enabled' user-values.yaml)
```

Done! Optionally, you may want to display the file `user-values.yaml` and verify that is similar to the picure of the previous section

### 3.3 Run the helm deployment

The actual deployment of databand is done with a single command from the helm charts directory (the one containing the file that we've just edited)

```
cd $DATABAND_UNCOMPRESSED_DIR
cd databand
helm upgrade databand --install --namespace databand --values ./values-ocp.yaml --values ./u
```

The output of the deplyoment command will display the way to access databand with port-forwarding, but there is another way. We just need to create the route

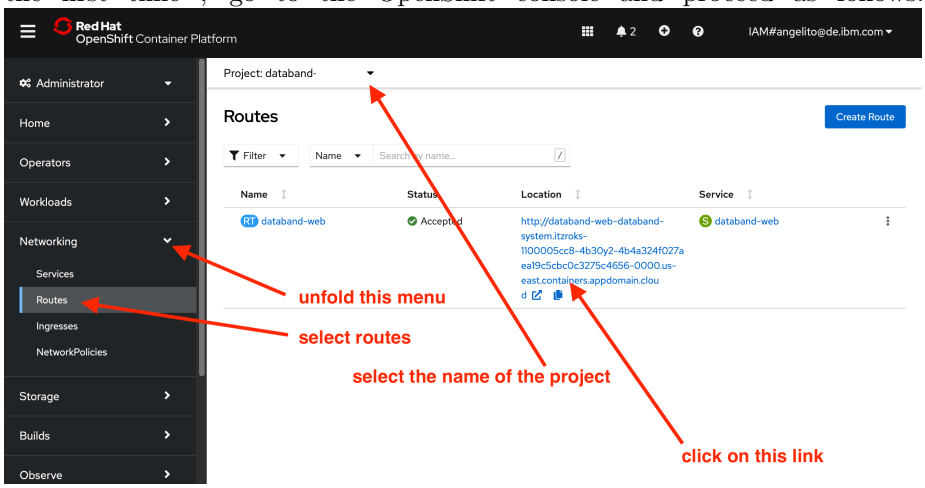
that exposes the service.

```
oc project databand
oc get svc
oc expose svc databand-web
```

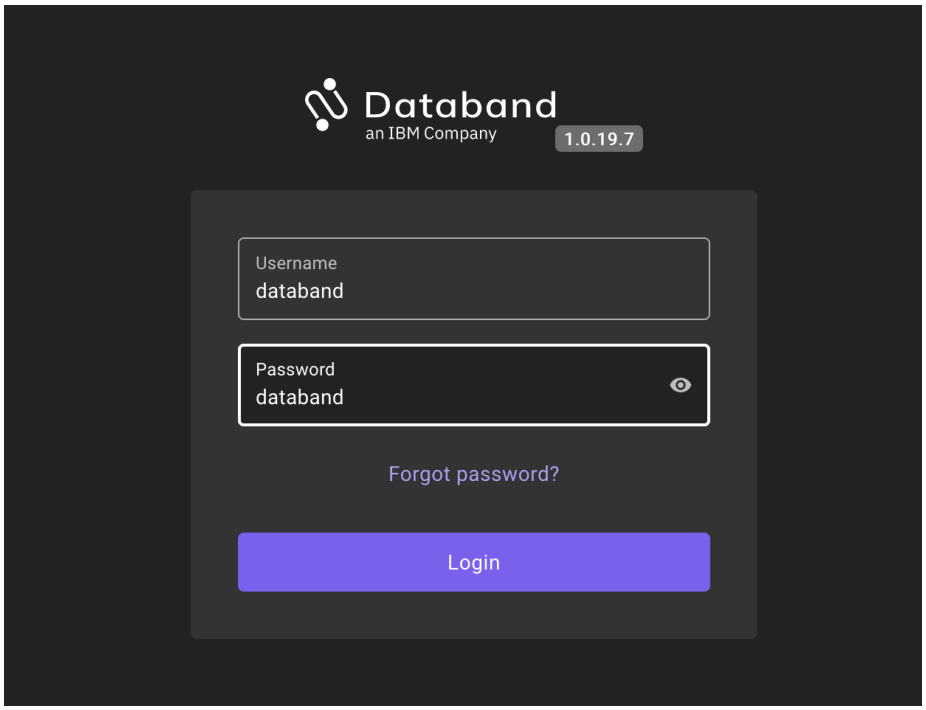
In the next section, you will see how to verify that databand is running

## 4. Test Databand

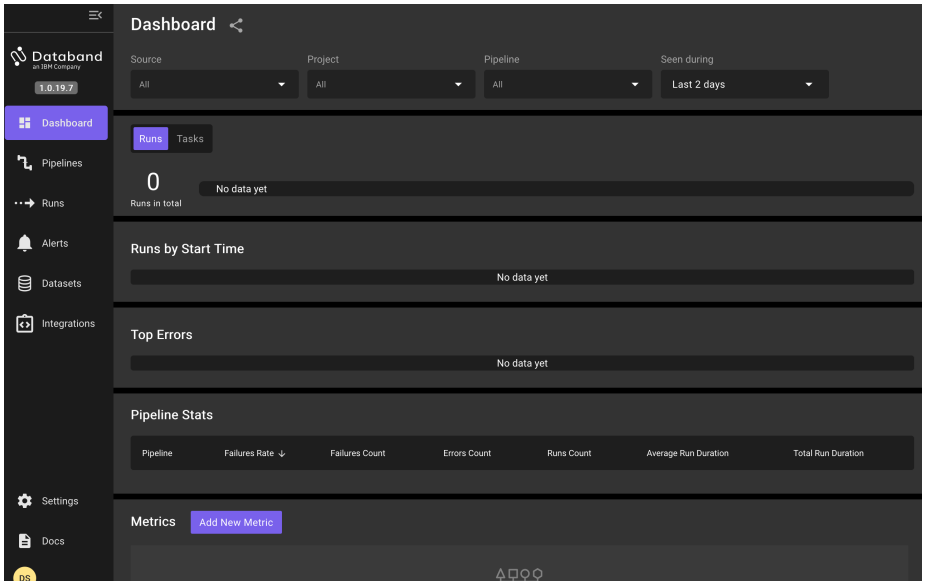
We have just created a databand instance with the default user `databand` and password `databand`. If you want to test the deployment for the first time , go to the OpenShift console and proceed as follows:




Type the defaul credentials




And the main dashboard of databand will start. Note that it is empty after the deployment.



Next Section: Airflow deployment. Previous Section: Databand deployment
Return to main