

# 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-UOrs --server=https://c109-e.us-east.containers.cloud.ibm.com:31656'  
OC\_LOGIN\_COMMAND='oc login --token=sha256~e8tzZje9glRmHCZEftW7EAXr3FnP\_TpYHTGjRq8p7po --server=https://c109-e.us-east.containers.cloud.ibm.com:31470'

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":true}}' --type=merge

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-1100005cc8-4b30y2-4b4a324f027aea19c5cbc0c3275c4656-0000.us-east.containers.appdomain.cloud  
REGISTRY\_URL=$(oc get route -n openshift-image-registry | grep default-route | awk '{print $2}')  
echo $REGISTRY\_URL

## 2. Preparing Docker images

### 2.1 Loading images locally

In the last part of the of the [prerequisites section](./0_prerequisites.ipynb) 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 paste it in this cell. Please observe the single quotes.  
# 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](#registry_url) 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](#manual_registry_url) 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 it again to be sure  
# Example of expected output: default-route-openshift-image-registry.itzroks-1100005cc8-4b30y2-4b4a324f027aea19c5cbc0c3275c4656-0000.us-east.containers.appdomain.cloud  
REGISTRY\_URL=$(oc get route -n openshift-image-registry | grep default-route | awk '{print $2}')  
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 cluster itself)  
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 Kubernetes terminology  
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 paste it in this cell. Please observe the single quotes.  
# 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 they are not lost  
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 '{print $2}')  
  
# 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/python-//')  
  
# 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](https://github.com/mikefarah/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 ./user-values.yaml .

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](./3_airflow_deploy.ipynb). Previous Section: [Databand deployment](./2_databand_deploy.ipynb)

[Return to main](../README.md)