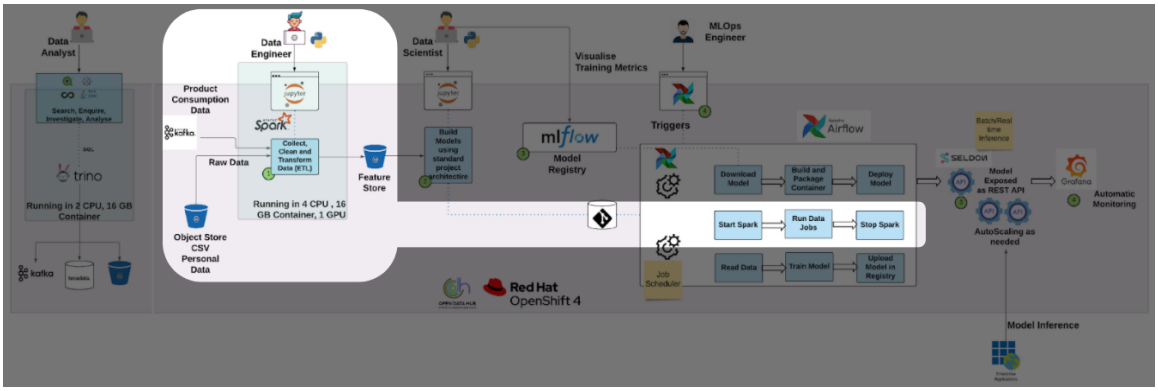
Lab 1 - Data Engineering

1.1 Introduction

The Open Data Hub exposes a data-focused tool - for Extract Transform Load (ETL) of data originating in multiple data sources, i.e. Apache Spark. Spark allows fine-grained ETL control, e.g. using Regex to match data patterns. Spark provides a further toolset to allow data professionals to prepare quality data for consumption by data scientists and AI models.

This diagram illustrates the workflow we're implementing - the beginning part of the overall AI/ML workflow:



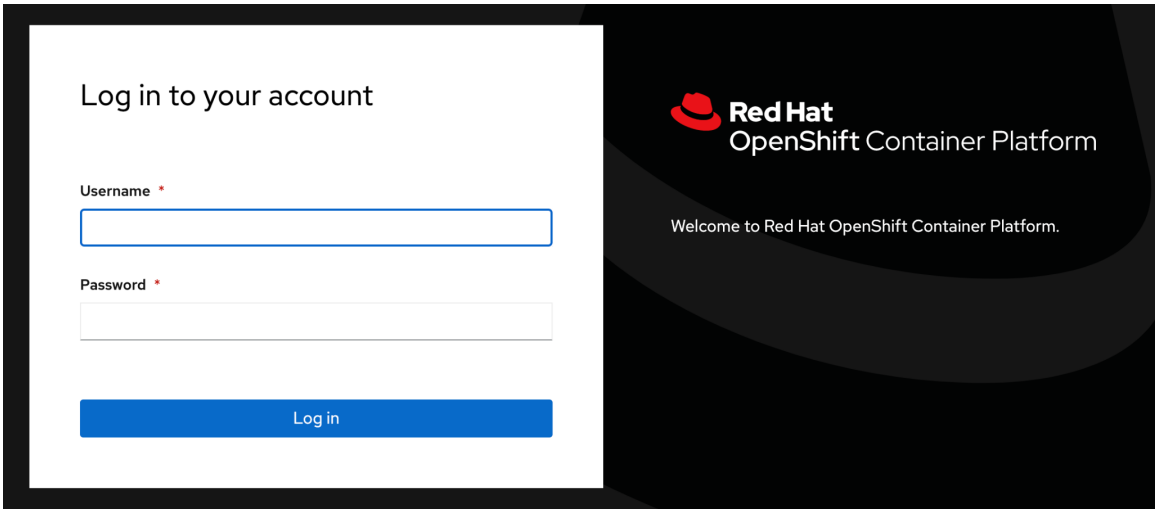
You can see, we source raw data from Kafka and S3 object storage. We use Jupyter notebooks to do some simple data engineering - combining these datasets on customerId using Spark. We then push that prepared data (a CSV file) to another bucket in our S3 object store, called Minio.

We also introduce a new level of autonomy and control - automation and scheduling of this task - using the workflow scheduling and management tool - Apache Airflow. You'll see this is very simple - democratising activities that used to be the realm of DevOPs specialists.

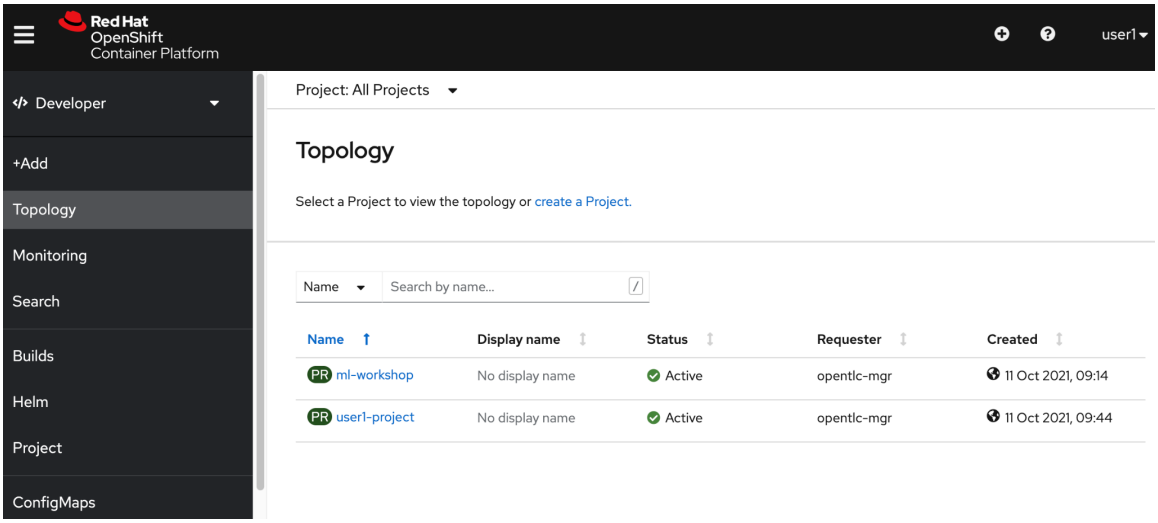
1.2 Access the OpenShift environment

Your instructor will provide you with a username, password, and link to access your environment. Using your browser and the URL provided by the instructor:

1. Open a new web page
2. Enter your username and password
3. Click **Log in**.

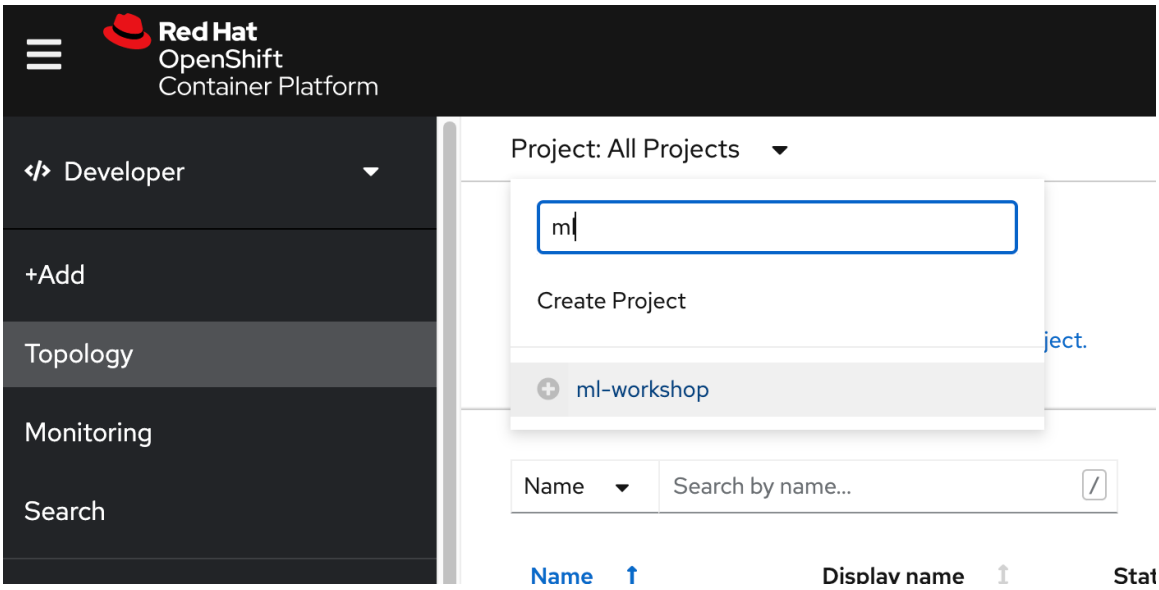


The main OpenShift console will be displayed.

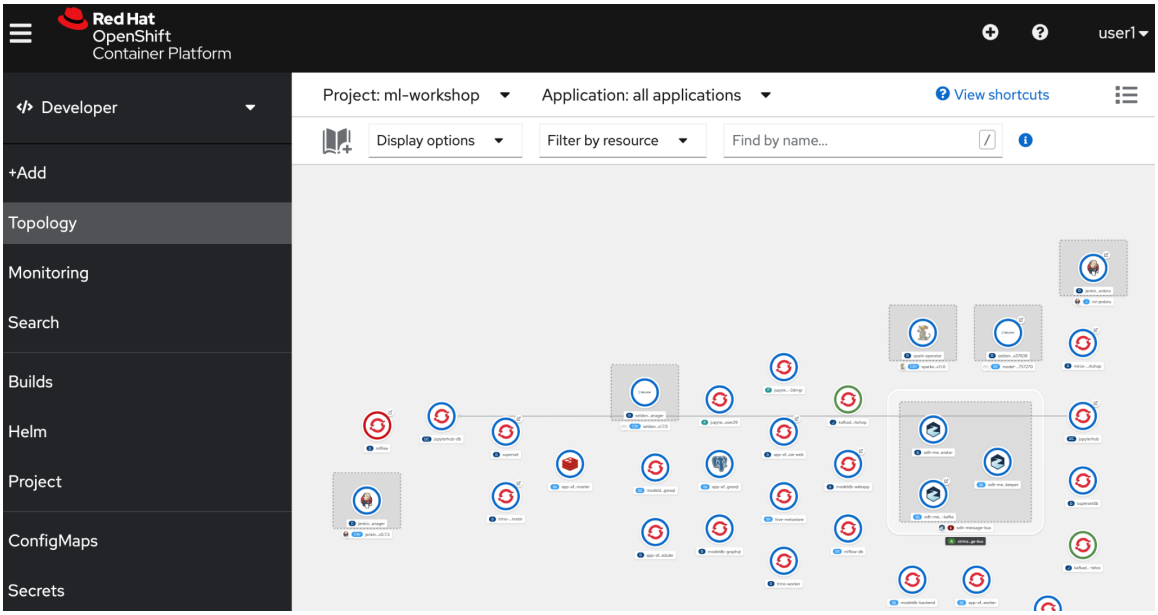


Open the project you will use in the workshop. In the menu bar:

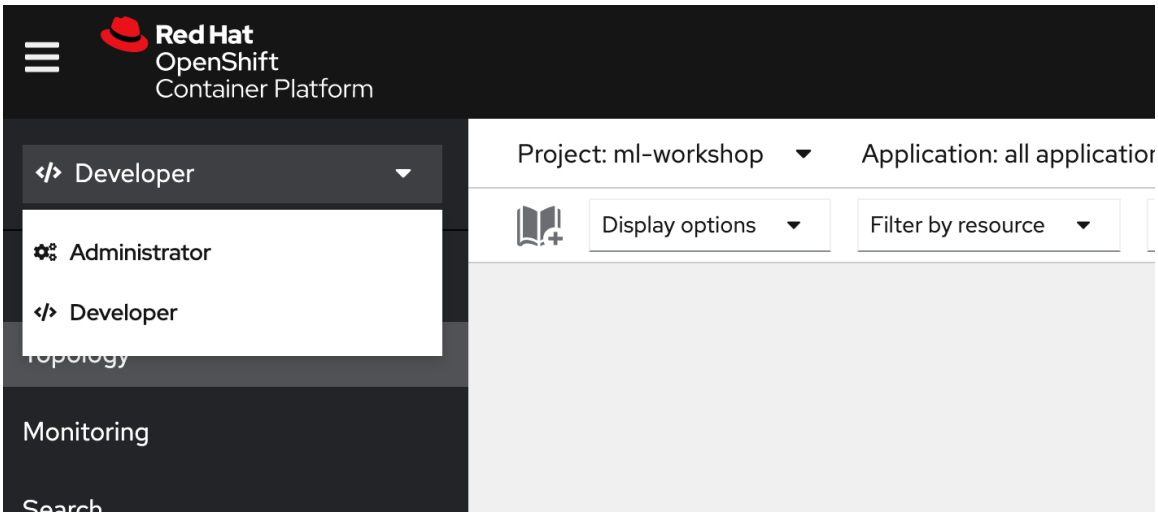
4. Click **All Projects**.
5. Type **ml-workshop** in the project text box.
6. Click **ml-workshop** from the drop-down list.



OpenShift will open the project and display the project Topology.



Note: Throughout these labs, you will be asked to select either the **Developer** or **Administrator** perspectives to perform the lab steps. To change the perspective, click the Perspective dropdown menu in the top left of the console, then click the perspective you need to use.

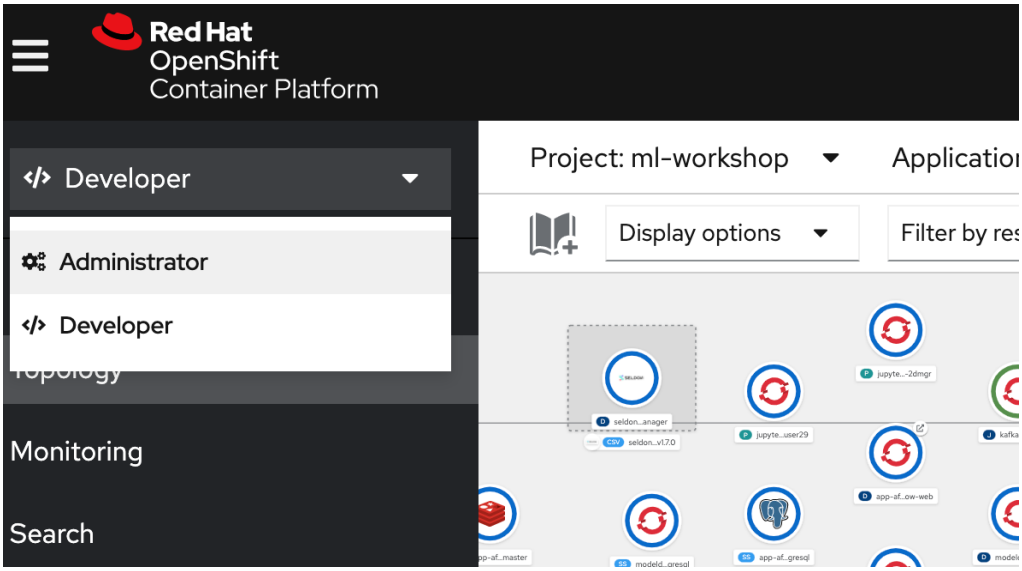


You have now completed accessing the lab environment. Proceed to the next section.

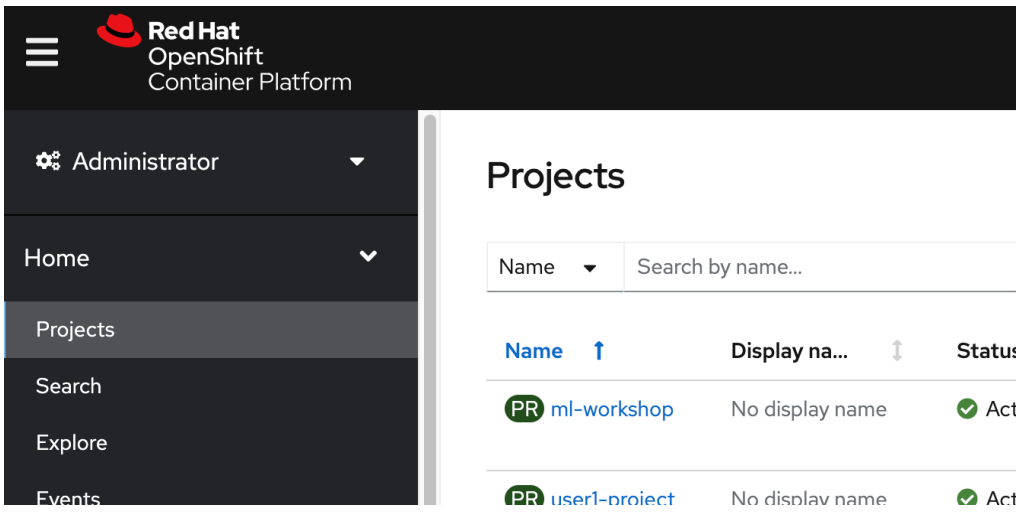
1.3 Instructions for the Spark workshop

Change to the **Administrator** perspective:

- Click the **Perspective** drop-down list in the top left of the console.
- Click **Administrator**.



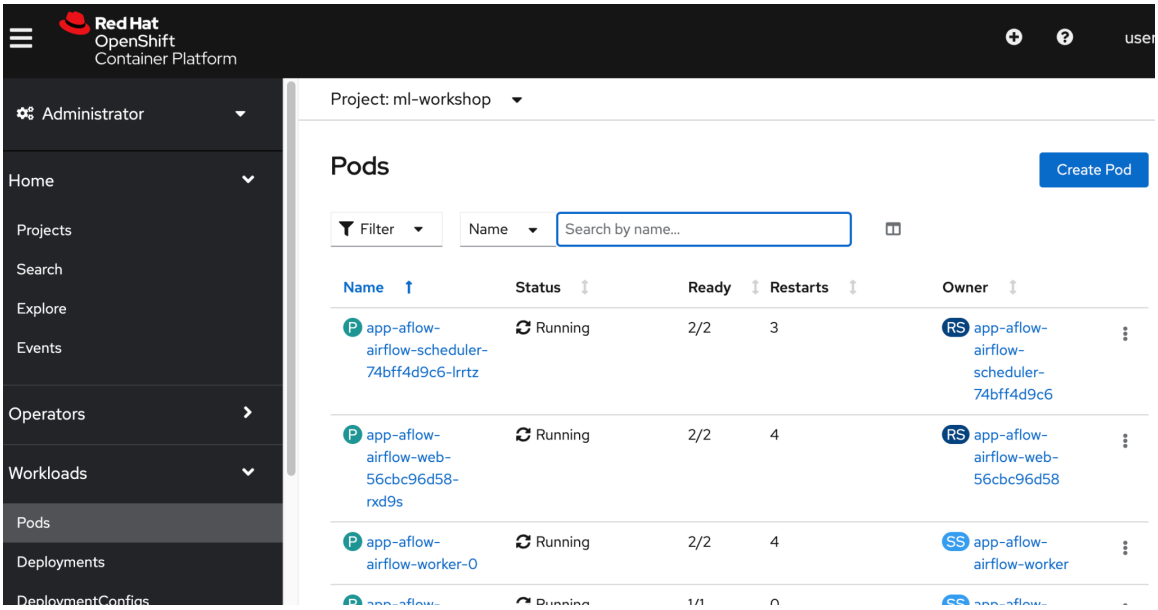
The OpenShift console displays the **Administrator** perspective.



Open a second browser tab with the same URL for the OpenShift console. We will refer to these as **tab 1** and **tab 2** in the instructions below.

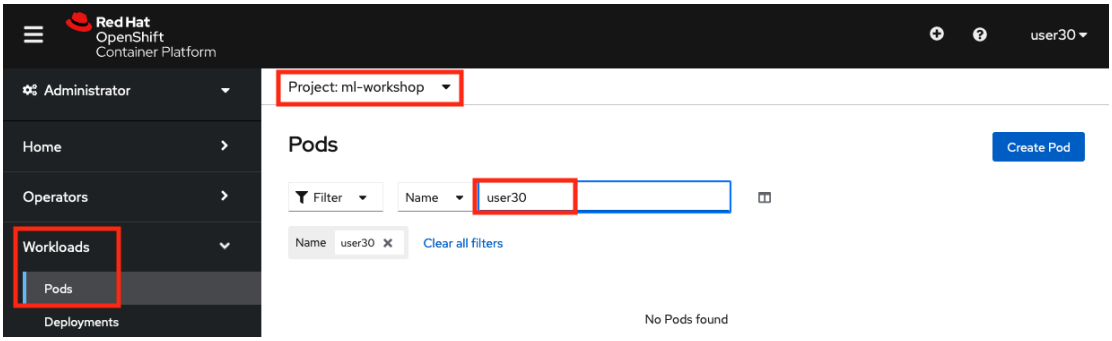
In Tab 1

- **Click Workloads > Pods.**



OpenShift displays a list of all pods running in the ml-workshop project.

- Enter your username in the Filter **Search by Name** text box

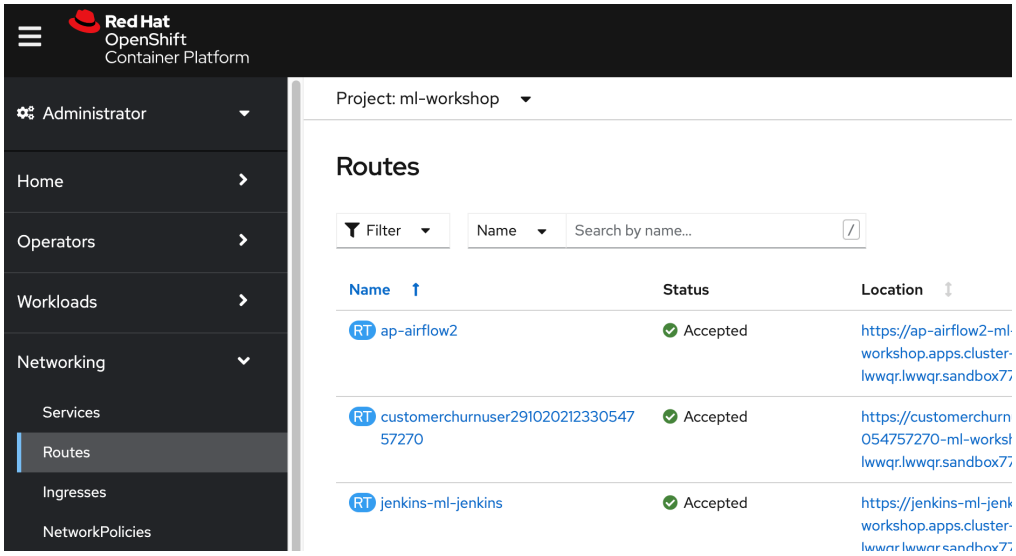


OpenShift will remove all (or almost all) of the pods so you only see the pods related to you.

We will now open JupyterHub using the second tab. Open **Tab 2** in your browser

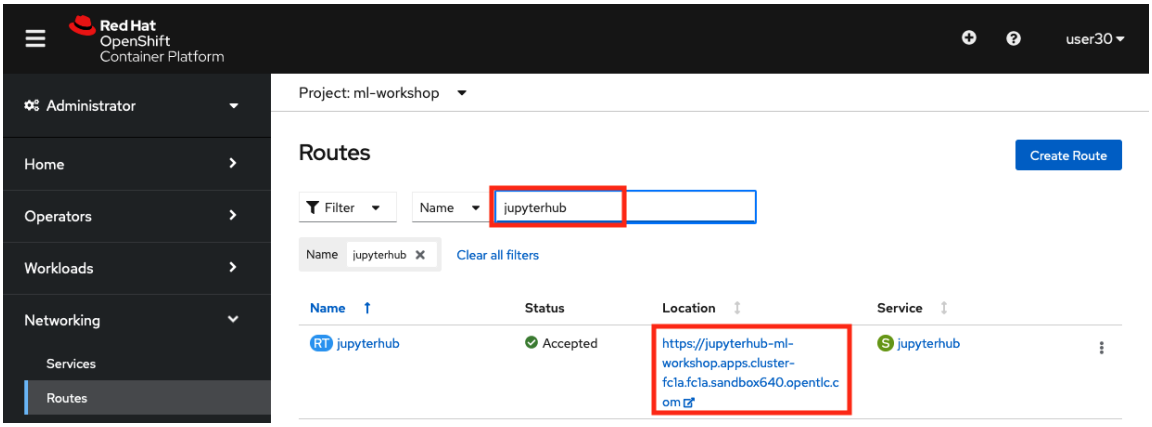
- Make sure you have the Administrator perspective selected.
- Click **Networking > Routes**.

OpenShift displays the external network routes to the applications in the ml-workshop projects. These routes are how you open the console for each application in Open Data Hub.

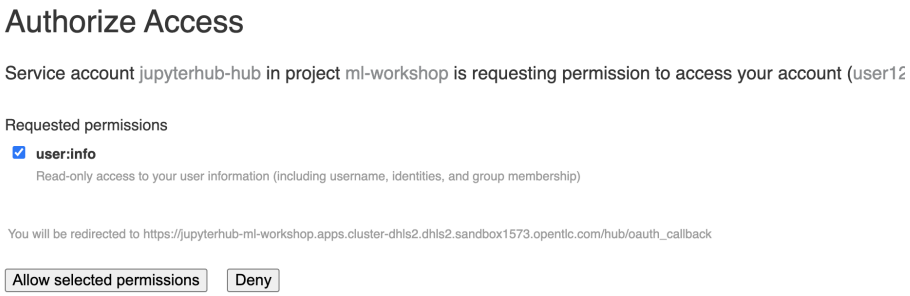


Filter the URL to find Jupyterhub.

- Type *Jupyterhub* in the **Search by name** text box.
- Click the route in the **Location** column

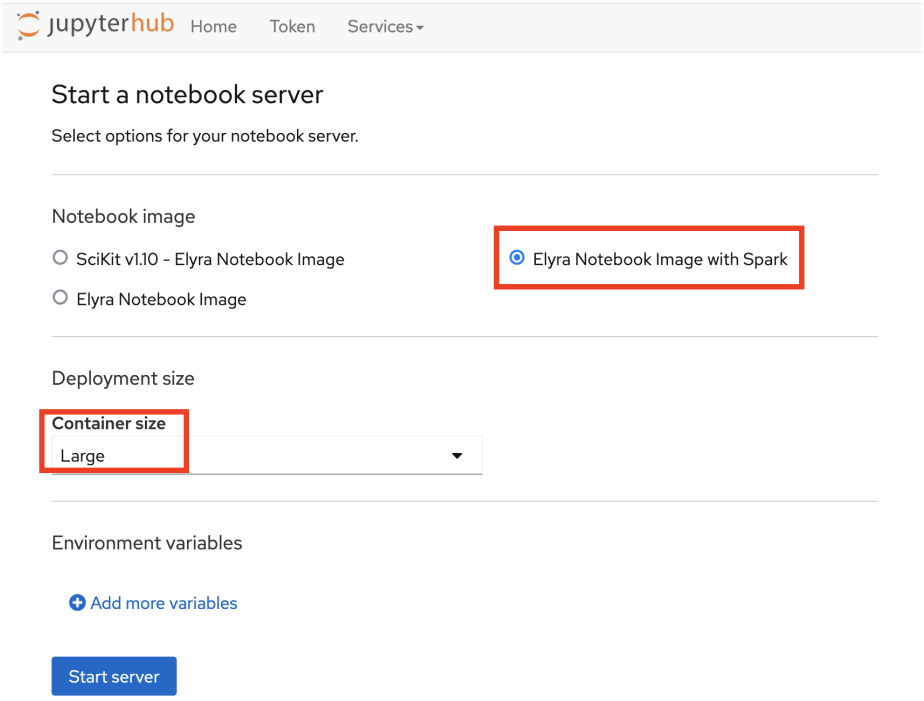


OpenShift will launch a new browser tab and prompt you for a user name and password. Those are the same username and password you used to log in to OpenShift. We now want to authorise the Service account jupyterhub-hub to access your account. Click **Allow selected permissions**.



OpenShift will prompt you to select the type and size of notebook server to start

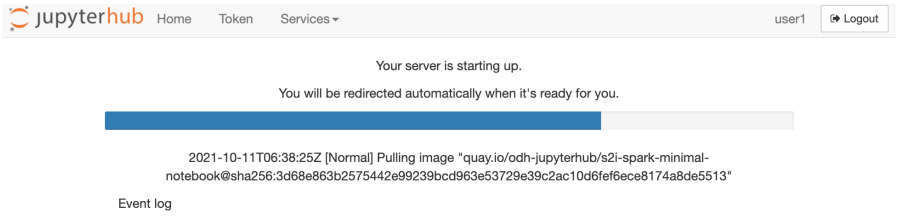
- Click **Elyra Notebook Image with Spark**
- Expand the **Container size** dropdown listbox and click **Large**.



- Click **Start Server**.

Warning: Please select the correct notebook image, otherwise the lab will not work.

OpenShift starts a new Jupyterhub notebook server for your lab environment.

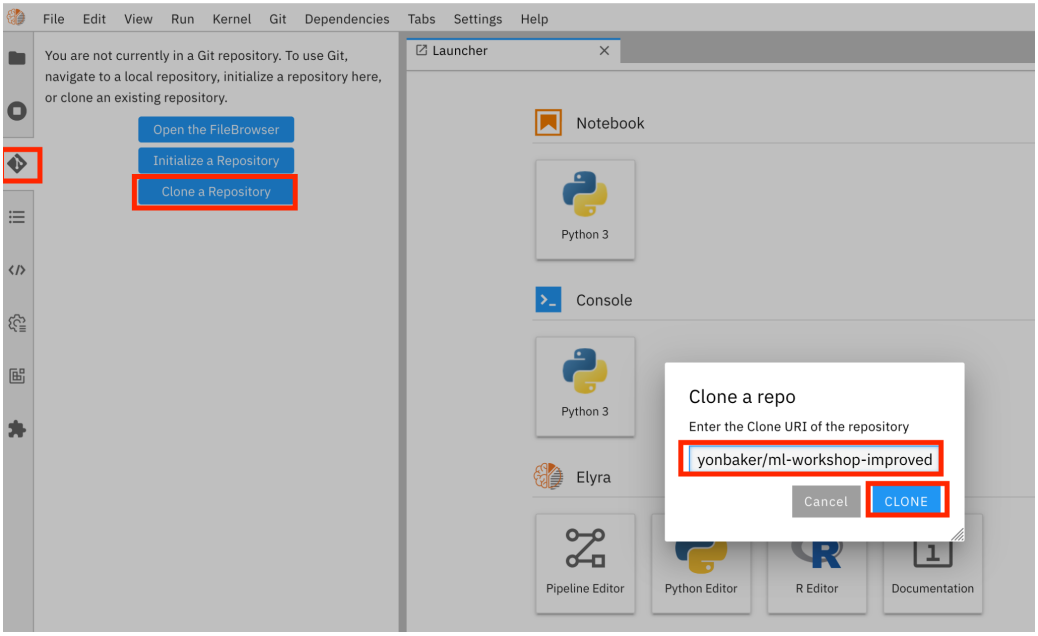


After a few minutes, JupyterHub displays the notebook.

Copy notebooks

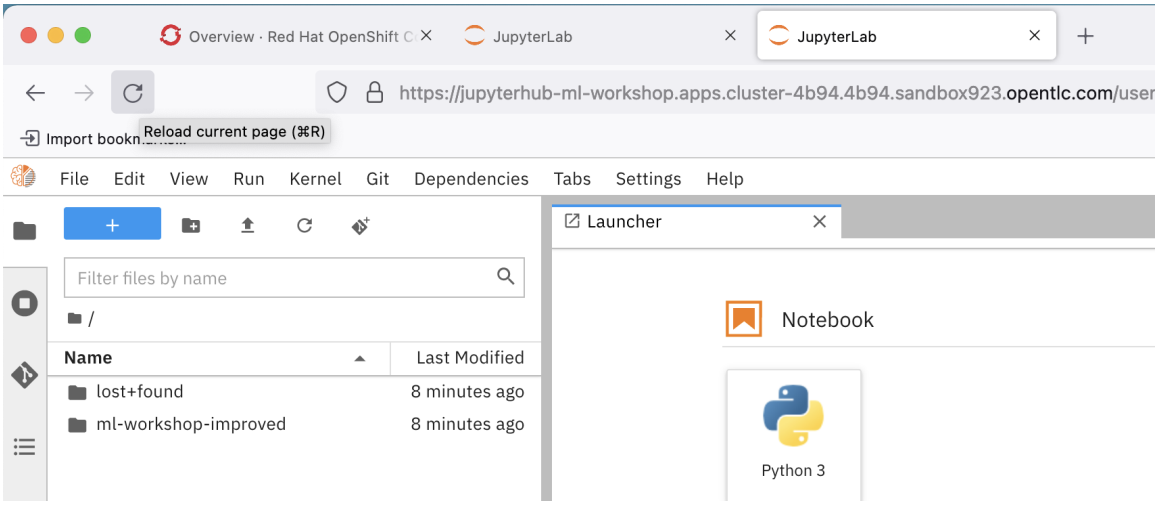
The first thing we want to do is pull down our notebooks from our repository *https://github.com/bryonbaker/ml-workshop-improved*

- Do the following in this order - as shown below
    a. Click the **Git icon** on the left and side of the screen
    b. Click the **Clone a Repository** button
    c. Enter https://github.com/bryonbaker/ml-workshop-improved in the textbox
    d. Click **CLONE**



Jupyterhub clones the git repository that has all the code you will be using for this lab.

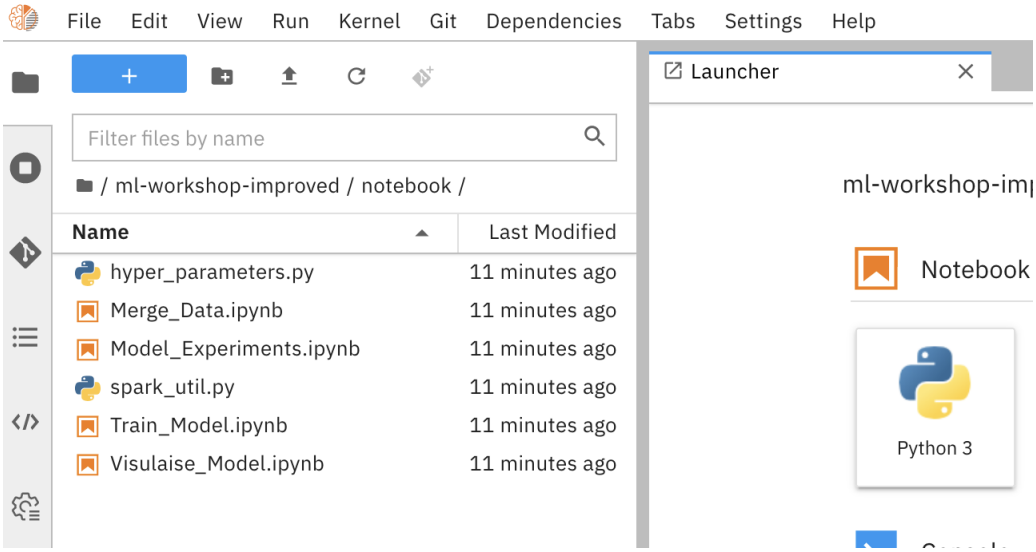
- Click the **Refresh or Reload** tab in your browser to display a Jupyterhub notebook.



Next, we must open the folder containing the notebooks for this workshop.

- Double click **ml-workshop-improved/notebook/**

OpenShift displays all of the notebooks you will be using in this lab.



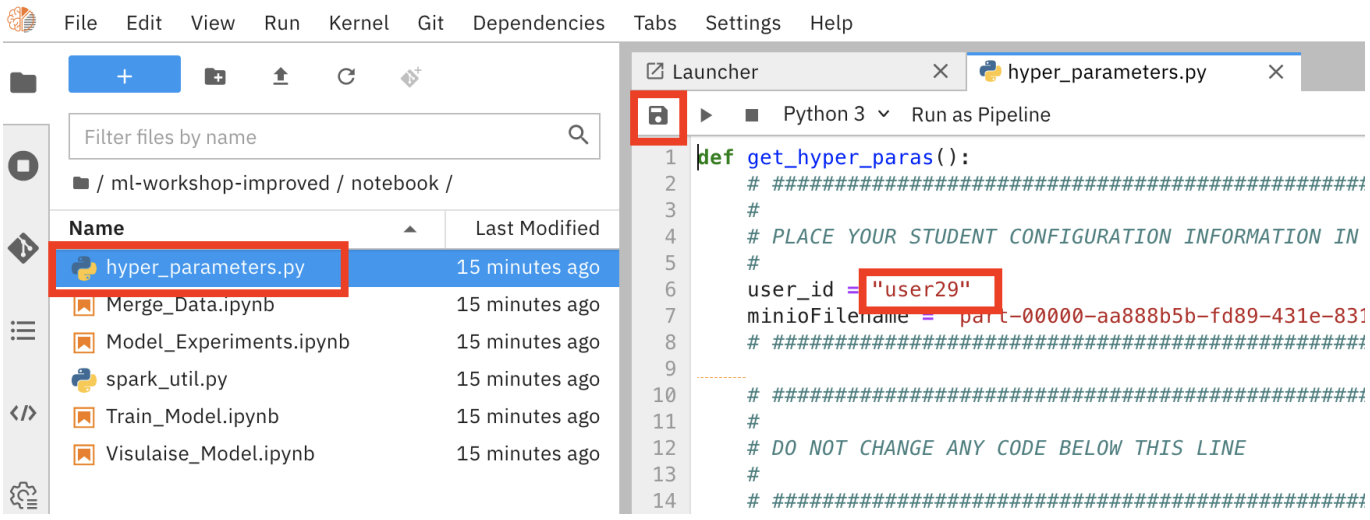
Before we get going, you need to make some small changes to the code.

- Open **hyper_parameters.py**
- Replace **user29** with your username that you are using for this lab (the same one you used to log on to OpenShift).

```
user_id = "<your username>"
```

Save your work

- Click the **Save** icon as shown to save your work.



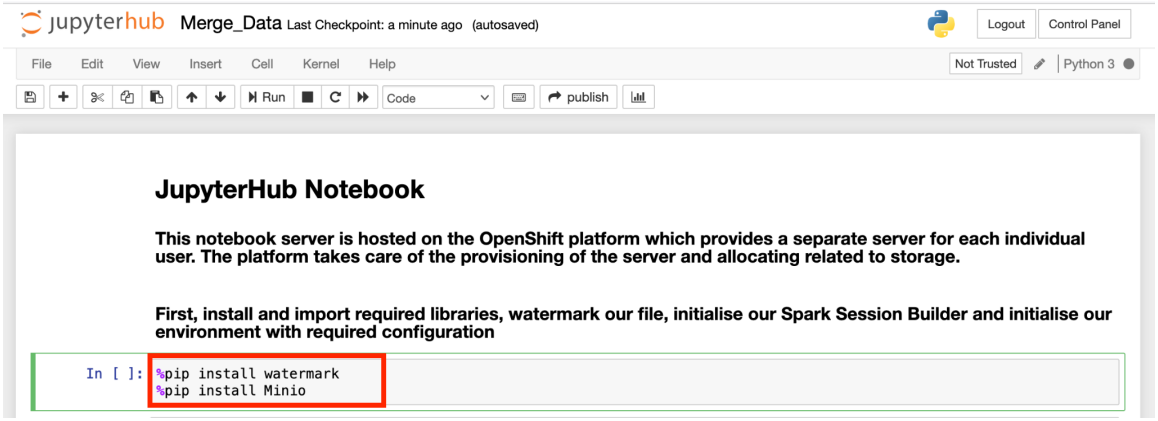
- Click **Merge_Data.ipynb.**

Jupyterhub opens the **Merge_Data** notebook. This notebook is used by the Data Engineer to prepare the data.

Run the notebook

You are now good to go!

- Scroll to the top of the notebook.
- Click inside the first code cell.



We will now step through each cell one by one so you can see what is going on.

- To execute a cell, type the **SHIFT + RETURN** keys together. Walk through the entire file this way, executing as you go. (if there are any blank cells, skip through them using SHIFT + RETURN).

Note: If you want to run the entire notebook click: **Kernel** from the top menu **> Restart & Run All**.

At a high level, this is what is happening in the most important cells:

1
```
%pip install watermark
%pip install Minio
%pip install matplotlib
```

2
```
import os
import json
from pyspark import SparkConf
from pyspark.sql import SparkSession, SQLContext
from pyspark.sql.functions import from_json, col, to_json, struct
import watermark
from minio import Minio

%matplotlib inline
%load_ext watermark
```

3
```
%watermark -n -v -m -g -iv
```

Load Hyper parameters

4
```
from hyper_parameters import get_hyper_paras
user_id,PROJECT_NAME,EXPERIMENT_NAME,experiment_name, s3BucketFullPath = get_hyper_paras()
```

5
```
sparkSessionBuilder = SparkSession\
    .builder\
    .appName("Customer Churn ingest Pipeline")
```

6
```
submit_args = "--conf spark.jars.ivy=/tmp \
--conf spark.hadoop.fs.s3a.endpoint=http://minio-ml-workshop:9000 \
--conf spark.hadoop.fs.s3a.access.key=minio \
--conf spark.hadoop.fs.s3a.secret.key=minio123 \
--conf spark.hadoop.fs.s3a.path.style.access=true \
--conf spark.hadoop.fs.s3a.impl=org.apache.hadoop.fs.s3a.S3AFileSystem \
--packages org.apache.hadoop:hadoop-aws:3.2.0"
```

Connect to Spark Cluster provided by OpenShift Platform

7
```
import spark_util

spark = spark_util.getOrCreateSparkSession("ML Ops Demo", submit_args)
spark.sparkContext.setLogLevel("INFO")
print('Spark context started.')
```

1. *pip install xxxx*, installs various libraries that aren't contained in our case container image
2. import the python libraries we need

3. *watermark* outputs the versions of various components, libraries, operating system attributes etc.
4. Here we load the user specific parameters - based on the username you entered in hyper_parameters.py
5. Here we create a Spark session, a precursor to firing up our own Spark server.
6. Here we set up various environment variables, including connection access to our S3 object store, in our case implemented using the open-source component Minio.
7. Here we actually start our Spark server. This cell can take several minutes to start.

Declare our input data sources, import and combine them

8
```
dataFrame_Customer = spark.read\
                .options(delimeter=',', inferSchema='True', header='True') \
                .csv("s3a://rawdata/customers/Customer-Churn_P1.csv")
dataFrame_Customer.printSchema()
```

9
```
dataFrame_Products = spark.read\
                .options(delimeter=',', inferSchema='True', header='True') \
                .csv("s3a://rawdata/products/Customer-Churn_P2.csv")
dataFrame_Products.printSchema()
```

10
```
# from pyspark.sql.types import *
# from  pyspark.sql.functions import *

# srcKafkaBrokers = "odh-message-bus-kafka-bootstrap:9092"
```

8. Here we pull in our data from S3 - our CSV based demographic data for each of our approximately 7000 customers.
9. Note - we are upgrading our demo and workshop. Kafka will be available shortly for the second half(side) of the data set. For now, both halves of our dataset comes from CSV files in S3 object storage
10. Commented out Kafka connection code - ignore for now

```
11 dataFrom_All = dataFrame_Customer.join(dataFrame_Products, "customerID", how="full")
```

Push prepared data to object storage and stop Spark cluster to save resources

```
12 file_location = "s3a://data/full_data_csv-" + user_id
   dataFrom_All.repartition(1).write.mode("overwrite")\
       .option("header", "true")\
       .format("csv").save(file_location)
```

```
13 spark.stop()
```

```
14 print('Notebook complete!')
```

11. We join these 2 datasets, on the common column to each: *customerID*.
12. We push our data to our object store - filename contains our username.
13. We are all done now - we stop our Spark server.
14. Signal that we're done - and the notebook has run successfully

This is our data engineering notebook finished. It's a simple exercise, though the same toolset could be used for much more complex data engineering tasks. Later on, when we automate this Merge-Data task in a pipeline, you'll see the output file in Minio S3 object storage.

Allow the Data Engineer to automate and schedule their work

Next, we introduce a new level of autonomy and control to the data engineer - automated running and optionally scheduling of the Merge Data notebook. As a data engineer, you've built a data pipeline (merge-data) that merges customer data from 2 data sources. But:

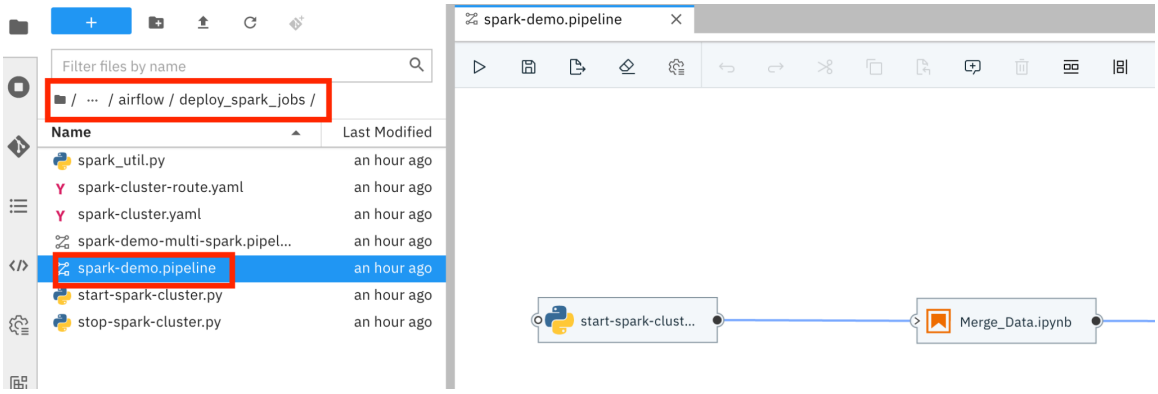
- How do I run it with production workloads?
- How do I schedule it to run - overnight, every day or every week etc.?
- How do I ensure this pipeline works right across the ecosystem?

That's where a tool called Airflow comes in. With Airflow,

- you can automate the running of your notebook - and run it anywhere, or scheduled any time
- the same IDE (Jupyter Hub in OpenShift) can be used to develop and create the automated data pipelines

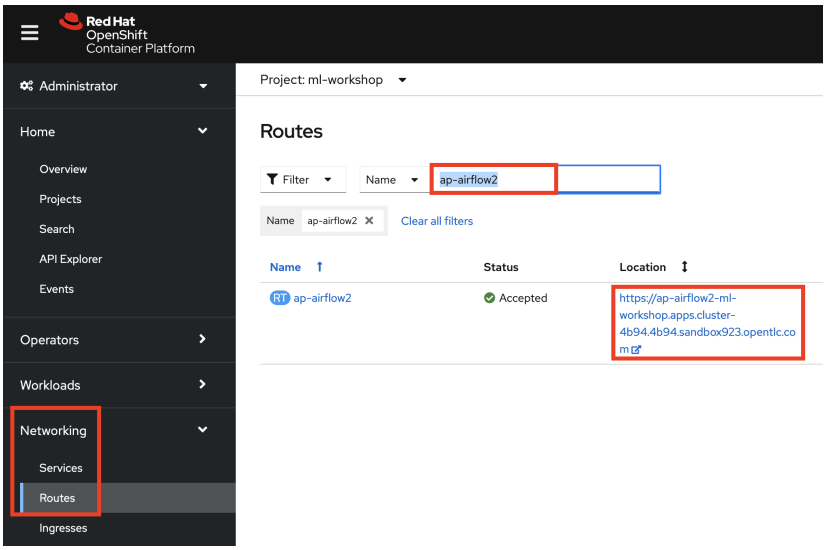
Let's get started

- Navigate to **ml-workshop-improved/airflow/deploy_spark_jobs/**
- Open **spark-demo-multi-spark.pipeline**



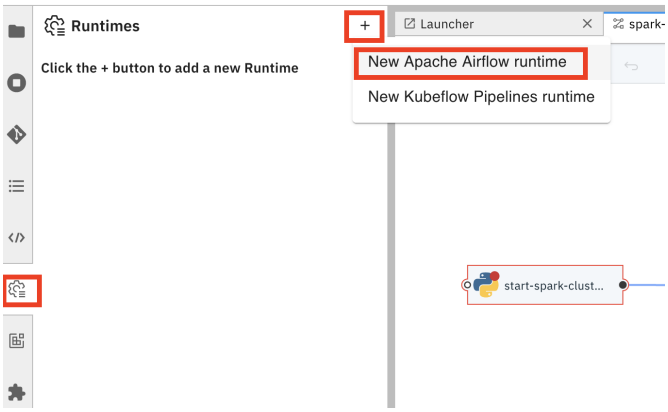
Notice it's a canvas-type GUI component onto which Python files can be dragged. Each component in the pipeline can be configured from this GUI - which we'll do later.

- The next thing we need to do is set up an Apache Airflow runtime. Before we do that we need to get the value of your **Airflow Route URL**.
- On OpenShift, navigate to **Networking > Routes** and filter by the name ***ap-airflow2*** as shown. Copy the URL under Location. This is your **Airflow Route URL.**



Back in Jupyter Hub, do the following:

- Click on the **Runtimes** icon on the left-hand side
- Click on the **+** icon on the top
- Select **New Apache Airflow runtime**



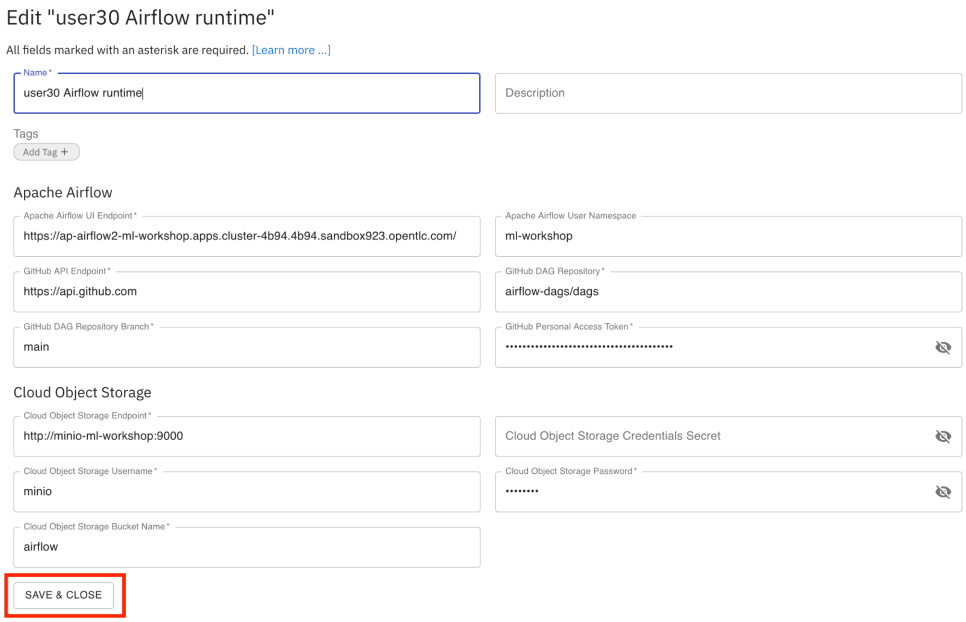
- Fill it in as follows - and click **Save and Close**

Name:	**userXX** Airflow runtime [**footnote** [1]]
Apache Airflow UI endpoint:	your <**Airflow Route URL**> retrieved above
Apache Airflow User Namespace:	ml-workshop
Github API endpoint:	https://api.github.com [**footnote** [2]]
Github DAG Repo:	airflow-dags/dags [**footnote** [3]]
Branch:	main
Github Personal Access Token:	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
Cloud Object Storage Endpoint	http://minio-ml-workshop:9000 [**footnote** [4]]
The Cloud Object Storage Username	minio
The Cloud Object Storage Password	minio123
Cloud Object Storage bucket name	airflow

[1] Replacing **userXX** with you username
[2] Note: when you create the pipeline, it doesn't get saved to Airflow, rather it is saved to Git - to give you versioning etc.
Then Airflow downloads from Git. Means you need to use a Git repo.

[3] For convenience, use our temporary repo setup for workshops:
https://github.com/airflow-dags/dags
[4] Airflow needs S3 to push logs to

As shown here:

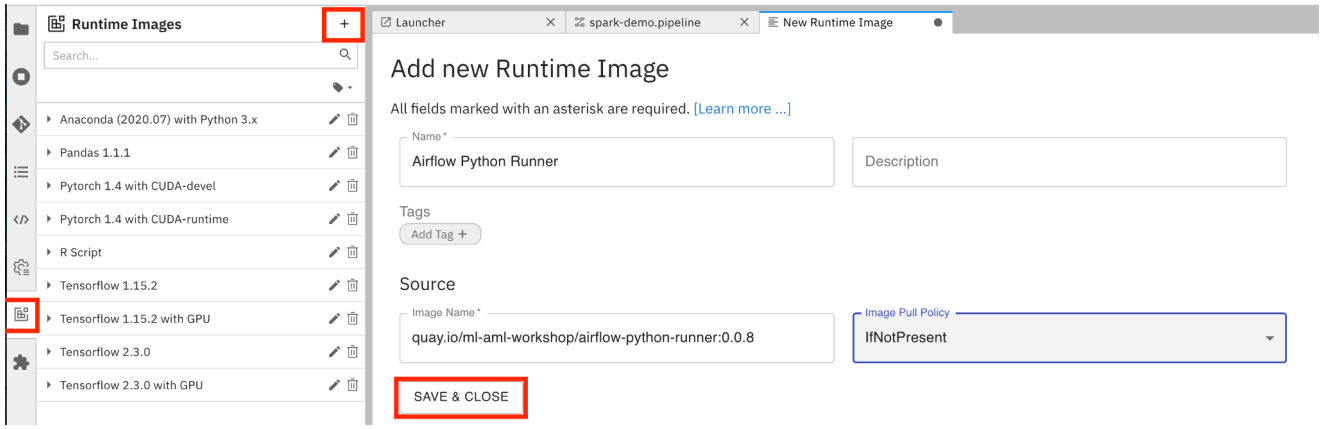


Next, we need to add a Python Runner.

- Choose **Runtime Images > +** and fill it in as follows - and click **Save and Close**

Name: Airflow Python Runner
Image Name: quay.io/ml-aml-workshop/airflow-python-runner:0.0.8
Image Pull Policy: IfNotPresent

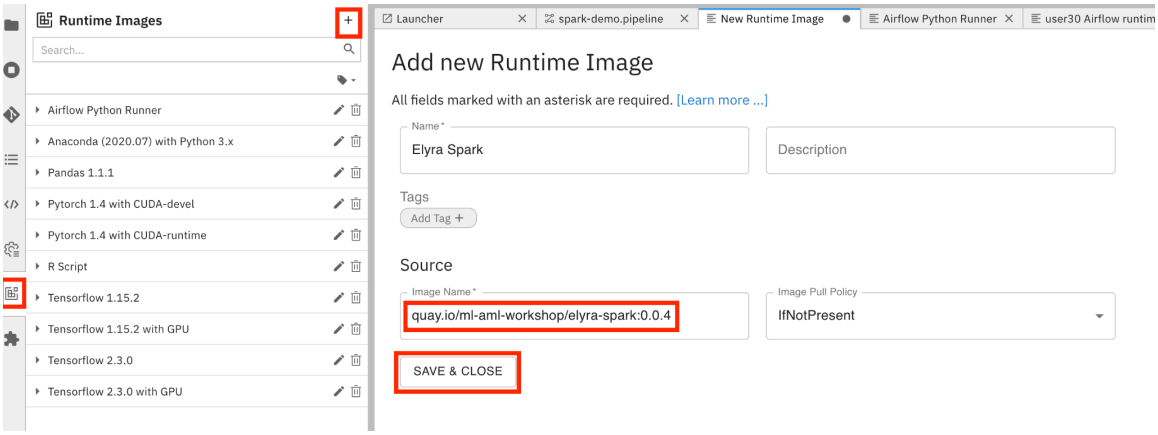
as shown here:



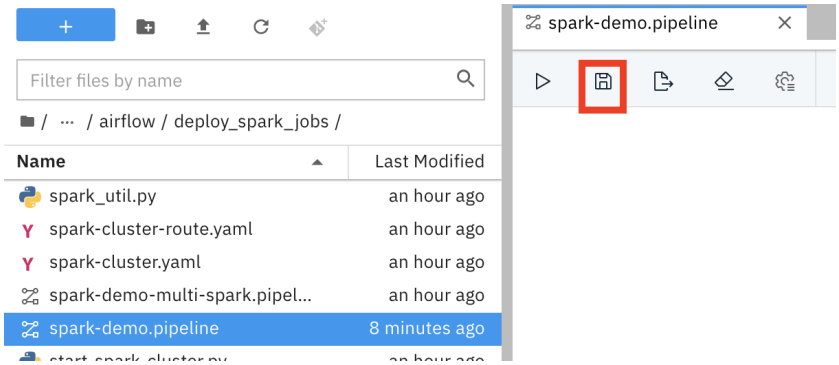
- Again, choose **Runtime Images > +** and fill it in as follows - and click **Save and Close**

Name: Elyra Spark
Image Name: quay.io/ml-aml-workshop/elyra-spark:0.0.4
Image Pull Policy: IfNotPresent

as shown here:

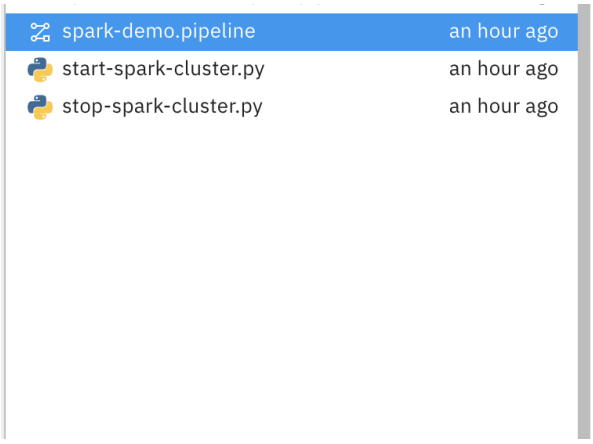


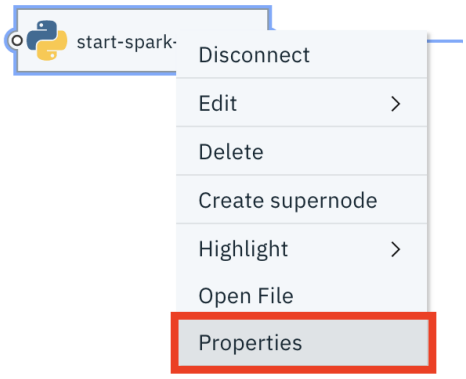
- Save your work by clicking on the **Save** button



Each component in the pipeline can be configured from the pipeline GUI - which we'll do now beginning with the start-spark-cluster element of the pipeline.

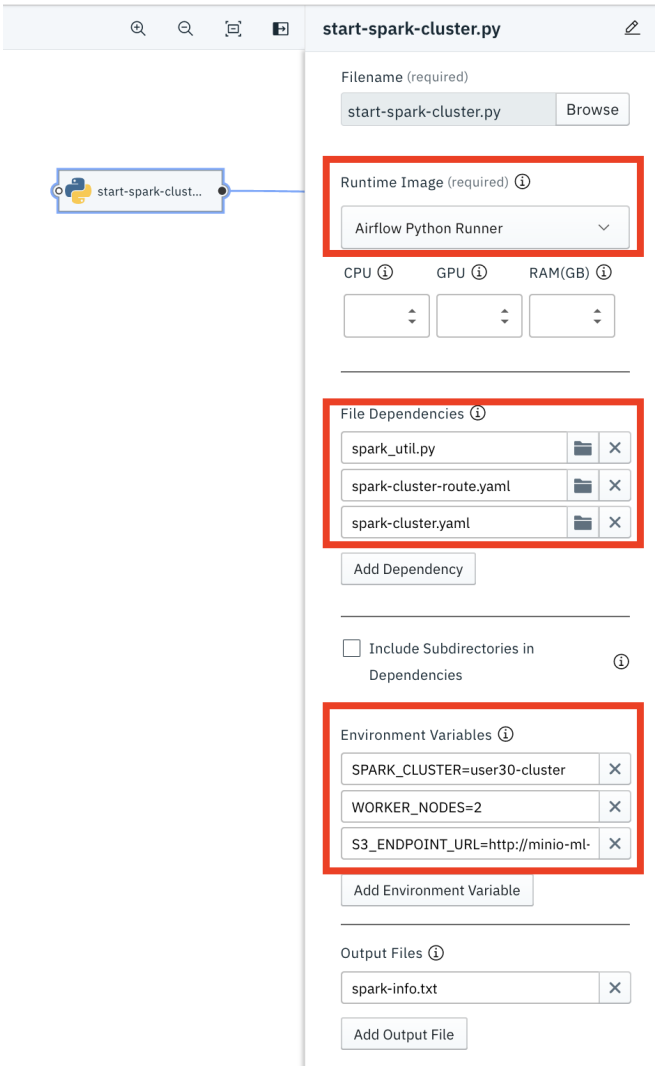
- Right-click on **start-spark-cluster** and choose **Properties**



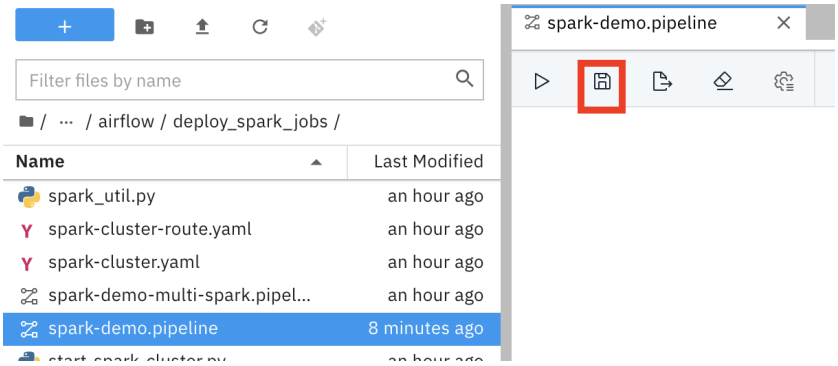


- Select these values as shown in the screenshot below
  - **Runtime Image:** Airflow Python Runner
  - File Dependencies
    - spark_util.py
    - spark-cluster-route.yaml
    - spark-cluster.yaml
  - Environment Variables
    - S3_ENDPOINT_URL=http://minio-ml-workshop:9000
    - SPARK_CLUSTER=**userXX**-cluster
      replacing **userXX** with your username and ensuring your cluster has 2 worker nodes
      (if a second SPARK_CLUSTER is there, **delete it**).
    - WORKER_NODES=2
  - Output Files
    - spark-info.txt

As shown here:


start-spark-cluster.py
start-spark-clust...
Filename (required)
start-spark-cluster.py
Browse
Runtime Image (required)
Airflow Python Runner
CPU
GPU
RAM(GB)
File Dependencies
spark_util.py
spark-cluster-route.yaml
spark-cluster.yaml
Add Dependency
Include Subdirectories in Dependencies
Environment Variables
SPARK_CLUSTER=user30-cluster
WORKER_NODES=2
S3_ENDPOINT_URL=http://minio-ml-
Add Environment Variable
Output Files
spark-info.txt
Add Output File

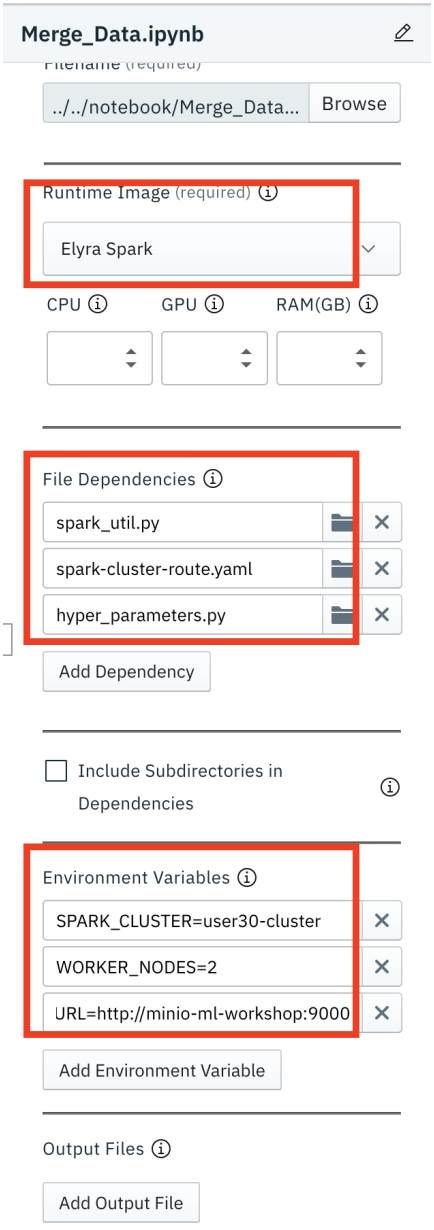
- Save your work by clicking on the **Save** button



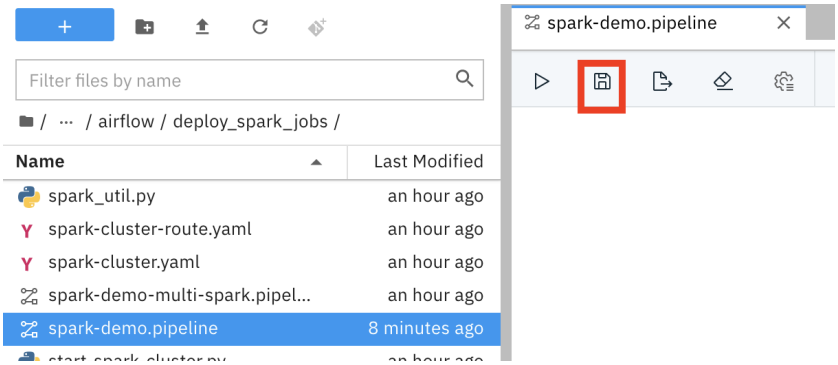
- Do the same with Merge_Data, right-clicking and filling as follows:
- Select these values as shown in the screenshot below
    - **Runtime Image: Elyra Spark**
    - File Dependencies
        - spark_util.py
        - spark-cluster-route.yaml
        - hyper_parameters.py
    - Environment Variables
        - S3_ENDPOINT_URL=http://minio-ml-workshop:9000
        - SPARK_CLUSTER=**userXX**-cluster
        - WORKER_NODES=2
          replacing **userXX** with your username and ensuring your cluster has 2 worker nodes

  As shown here:

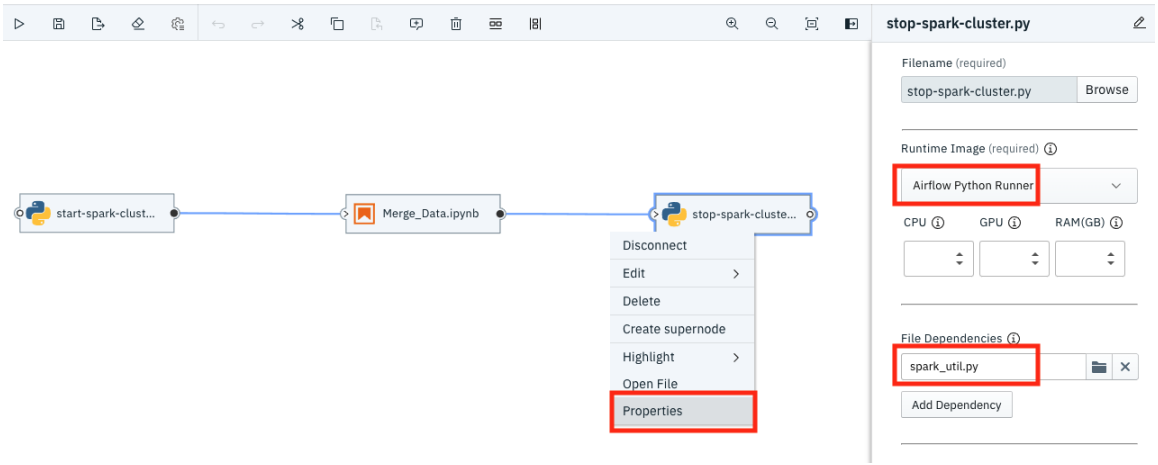
- **hyper_parameters.py** as a *File Dependency*



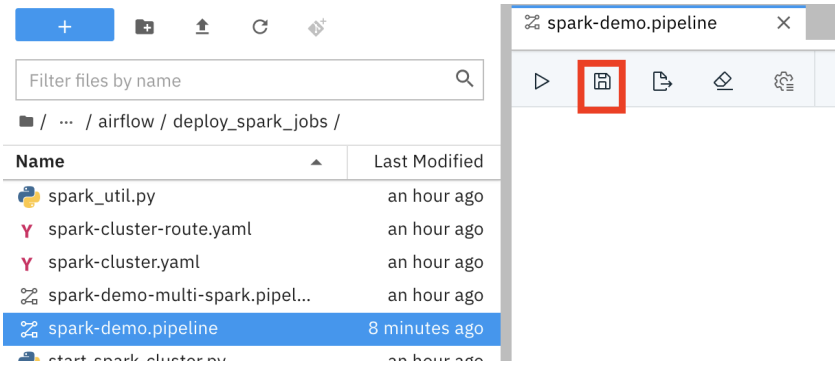
- Save your work by clicking on the **Save** button



- Finally, do the same with stop-spark-cluster - which has fewer still values



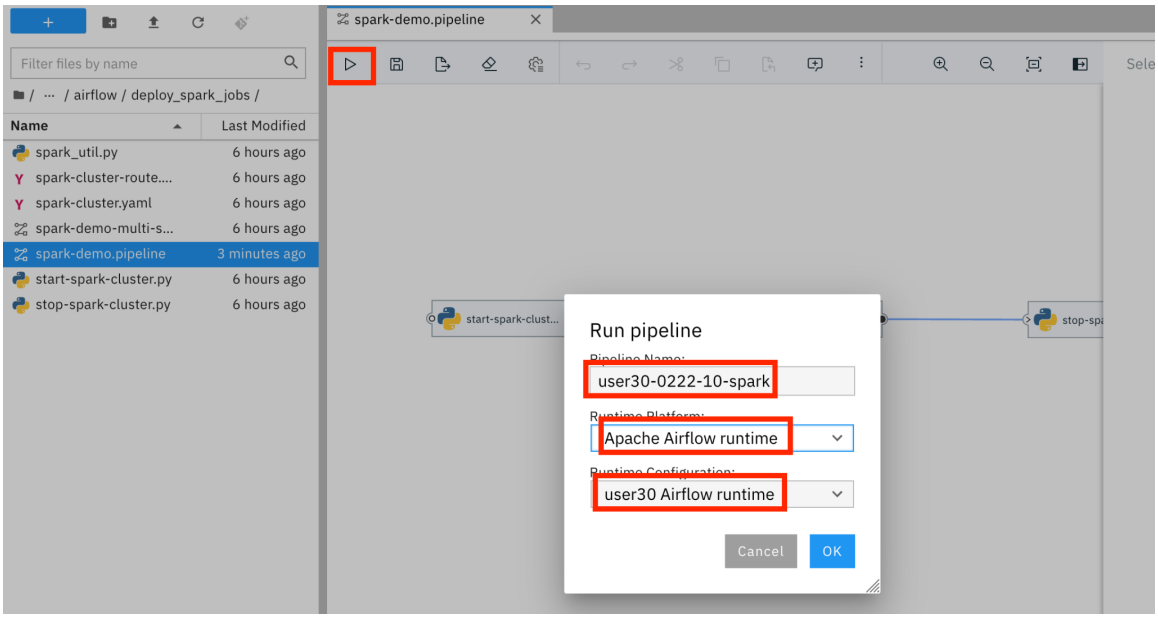
- Save your work by clicking on the **Save** button



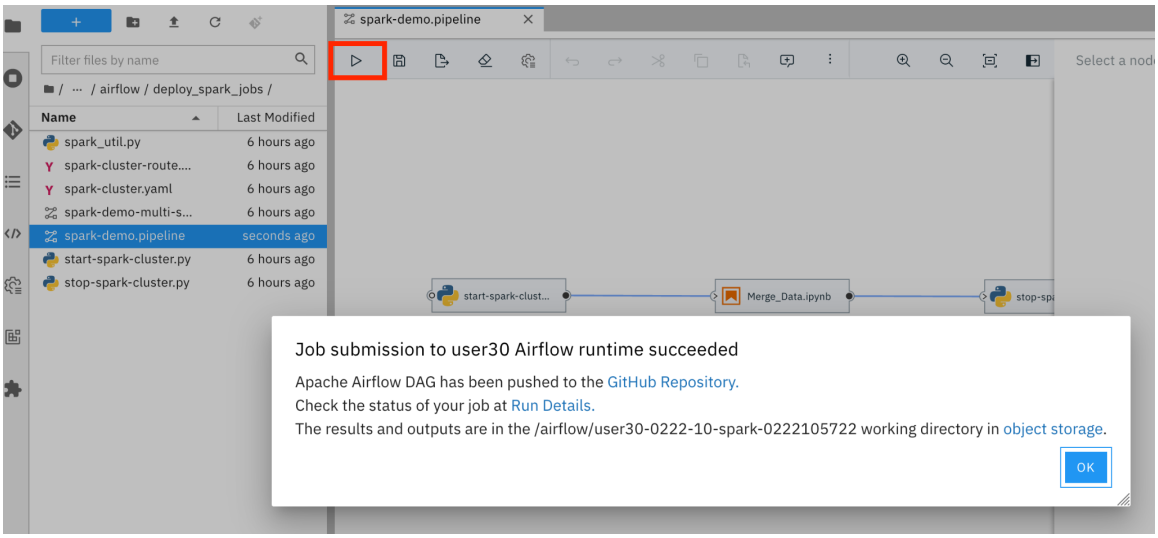
Run Pipeline

You can now run your pipeline. Click the Play button as shown. A useful naming convention is to enter a name in a format beginning with your username - followed by the month and day and the count of runs that day. This sends each run to the bottom of the list on the Airflow GUI.

userXX-**MMDD**-01-spark, in my case ***user30-0223-01-spark***. Also choose your Airflow Runtime and Runner as shown:

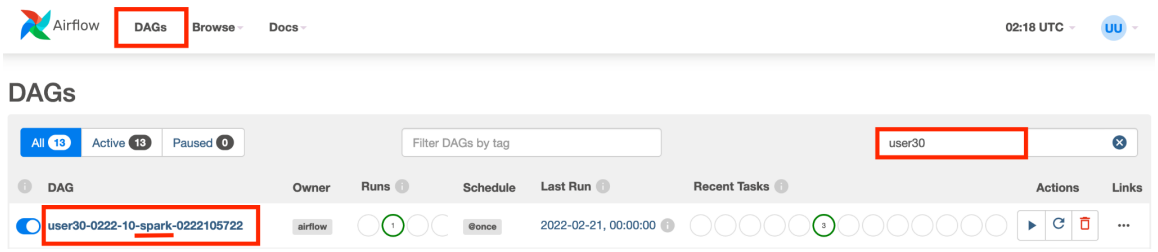


Two pop-ups will show - including this one. OK them



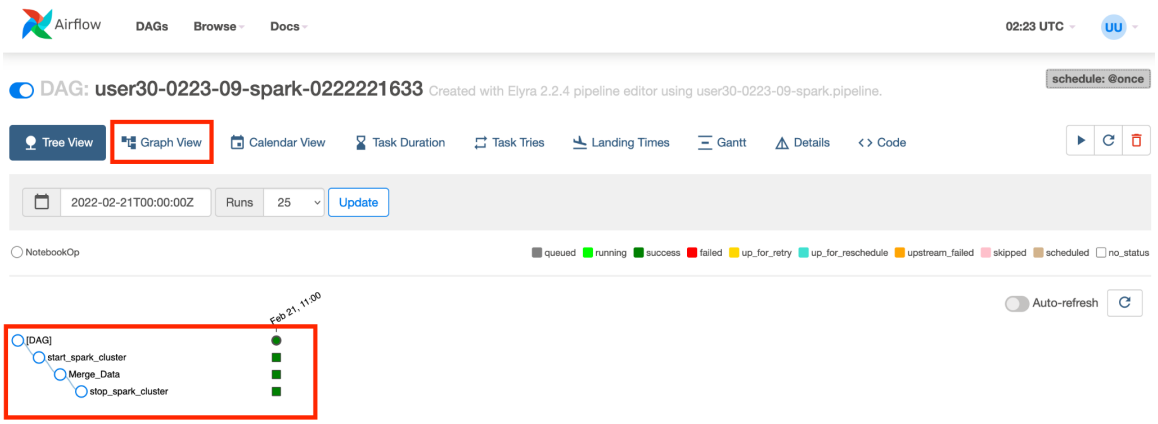
Now let's take a look at the Spark job we pushed to our workflow engine Airflow.

In a browser open your **Airflow Route URL** as described above in step 30. After logging in with your OpenShift credentials. click DAGs and you'll be presented with the a page like this:

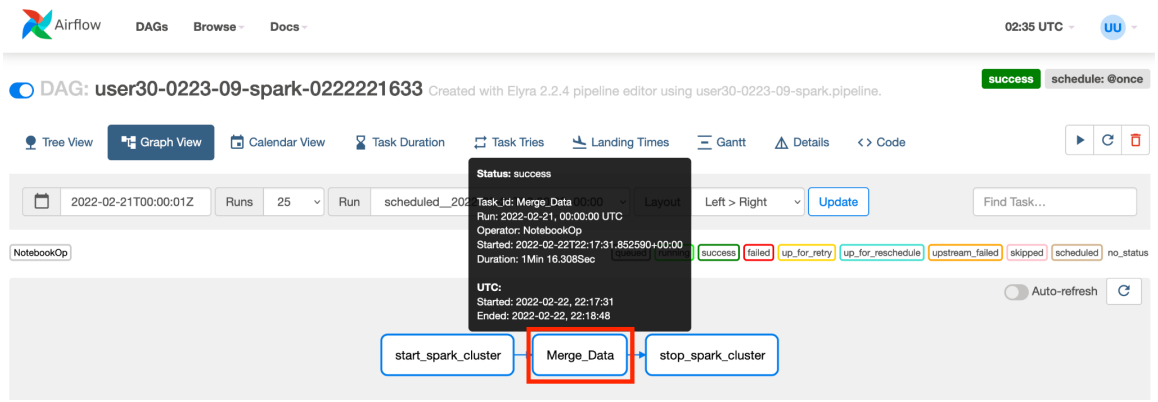


You will see many DAGs, representing previous pipeline runs. Filter on your username in the right hand side box to see only yours.

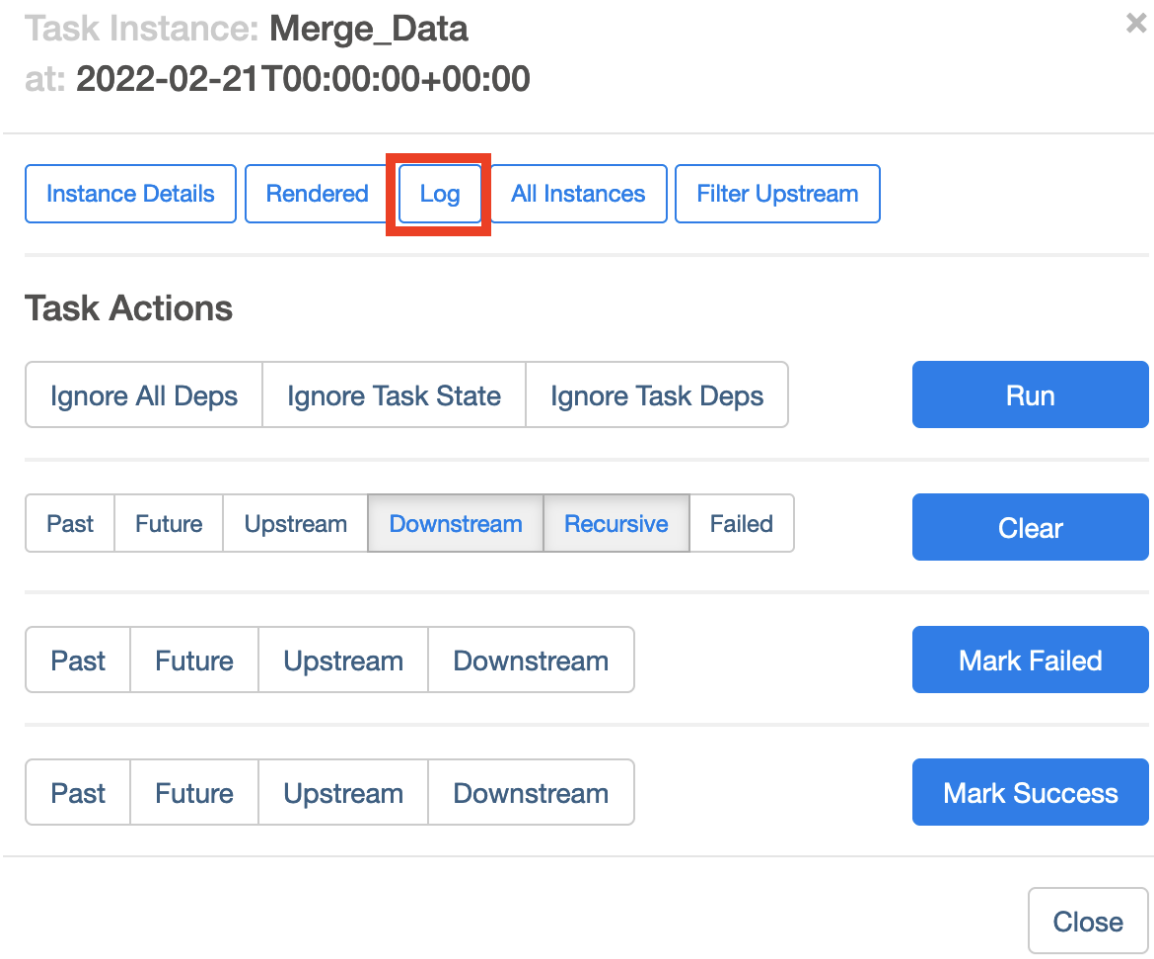
Drill into your Spark job DAG by clicking on it - the one you just ran. You can see the stages and the success or otherwise of the various stages in your pipeline.Click on Graph View for a more detailed view



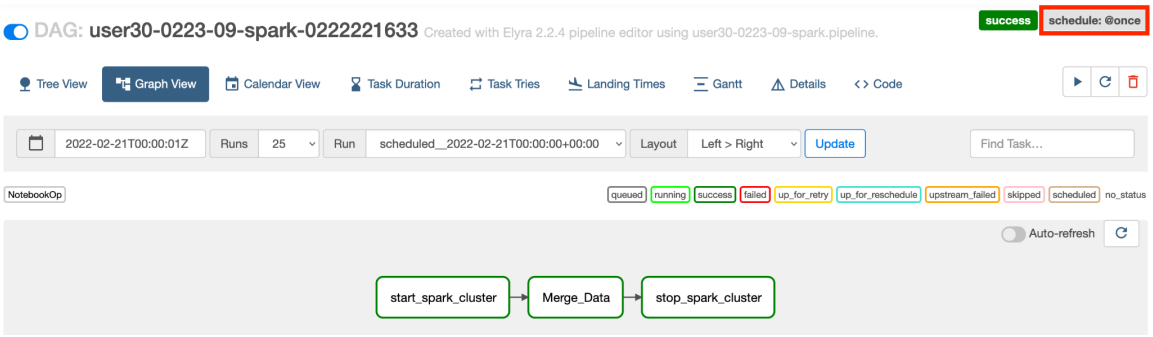
Click on the Merge Data stage



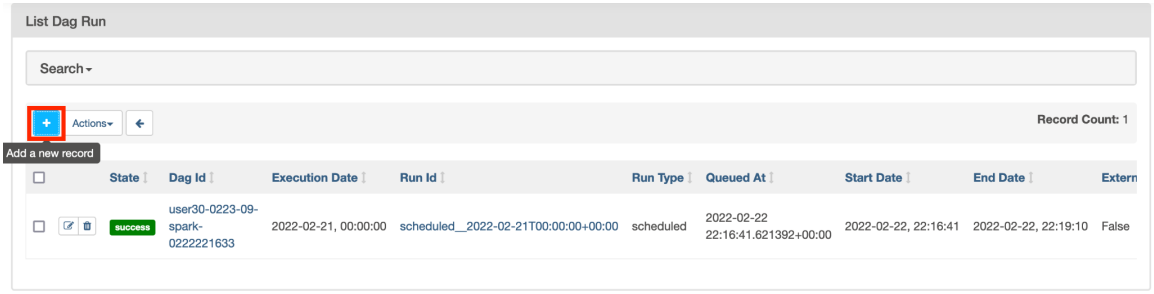
You have many options to control and re-run stages for example. You can also view the logs for the stage from here as well



Finally, by clicking **schedule@once** you can schedule this job to run on a periodic basis - fully automated through cron.



Click the blue + icon to do this. We'll leave this as an exercise for you, the user.

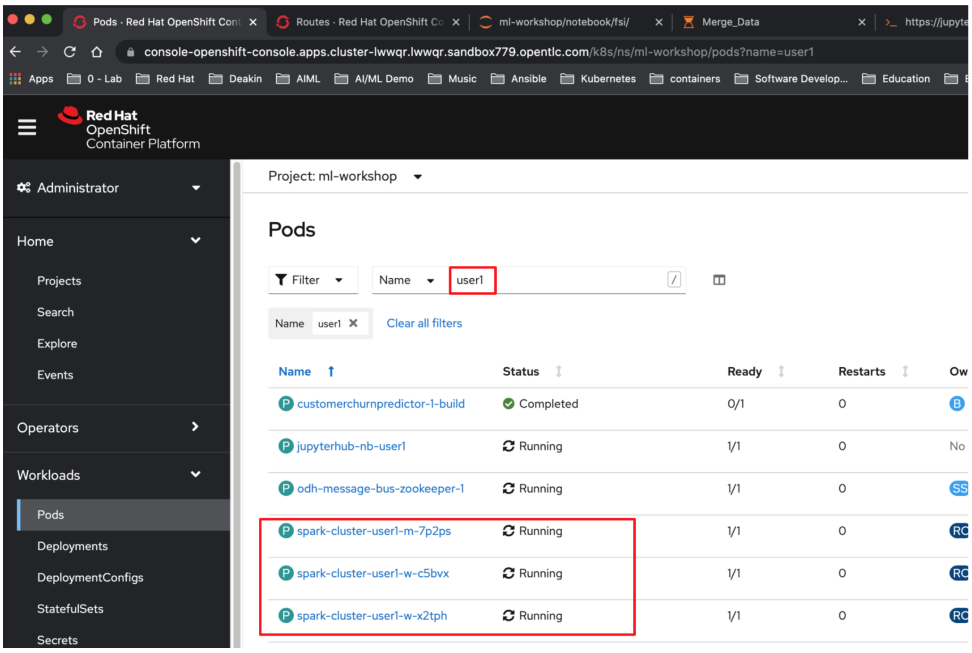


That concludes our Airflow pipeline and workflow section. Hopefully you agree this is a very powerful tool - bringing agility and autonomy to the data engineer!

Recap and Cleanup

Before we move on, let's have a look at how OpenShift provisioned a Spark cluster for you to do your work.

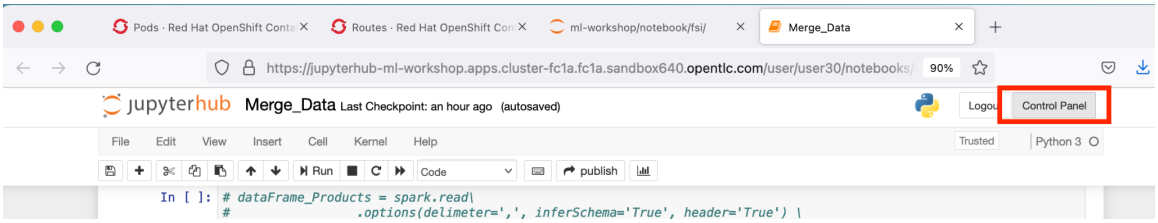
- Open **Tab 1**.



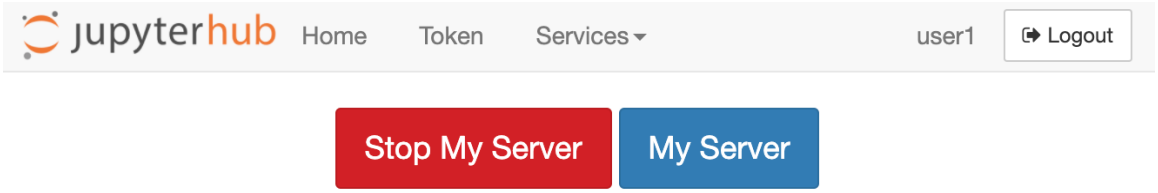
OpenSHift displays all of the pods for your username. Here you can observe that a Spark cluster was started for the Data Engineering work. This is an example of the self-service capabilities OpenShift brings to the table and unlocks the productivity of your team.

We don't need this notebook any more so we will shut it down - along with the Spark cluster that was created with it. This is another example of how OpenShift provides efficient use of resources by returning them to the pool when they are no longer needed.

- Open the Jupyterhub notebook tab.
- Click **Control Panel**



Jupyterhub displays the server-control page.



- Click **Stop My Server**.

  Jupyterhub commences releasing all of the OpenShift resources.

  To watch this in action:

- Open **Tab 1**

  Observe your Spark pods being destroyed.

Name	Status	Ready	Restarts	Owner	Memory	CPU
P spark-cluster-user30-m-qg6qk	Terminating	1/1	0	RC spark-cluster-user30-m	174.9 MiB	0.004 cores
P spark-cluster-user30-w-65l98	Terminating	1/1	0	RC spark-cluster-user30-w	164.6 MiB	0.004 cores
P spark-cluster-user30-w-l5vtd	Terminating	1/1	0	RC spark-cluster-user30-w	158.3 MiB	0.004 cores

You can see each of the pods terminating. This is a powerful demonstration of OpenShift's self service capabilities. No waiting for IT to provision you a server, no waiting around for access to a scarce Spark server. All self service, on demand, and those resources returned back to the central pool when finished.

Congratulations - you've completed the first lab. Now move to the next one: **AI/ML on OpenShift Workshop - Lab 2 - Data Science.**