Red Hat

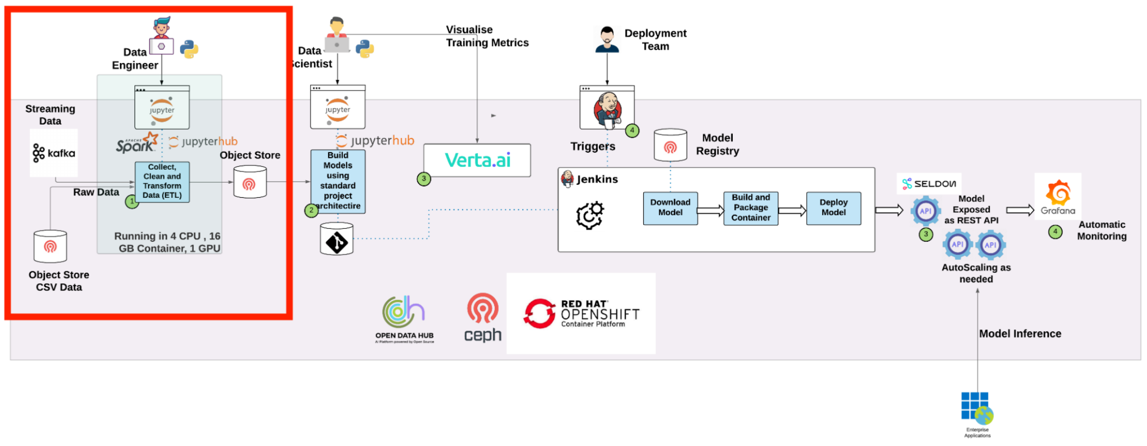
# Lab 2 - Data Engineering

## Introduction

In the previous lab, you explored the Open Data Hub component Superset - which provides easy to create charts and dashboards using the in-memory data engine Trino. Trino and Superset allow data residing anywhere to be accessed using a low latency in memory engine using SQL.

The Open Data Hub exposes a second data focused tool - for Extract Transform Load (ETL) of data originating in multiple data sources, i.e. Apache Spark. Spark allows finer grained ETL control than SQL/Trino does, 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:



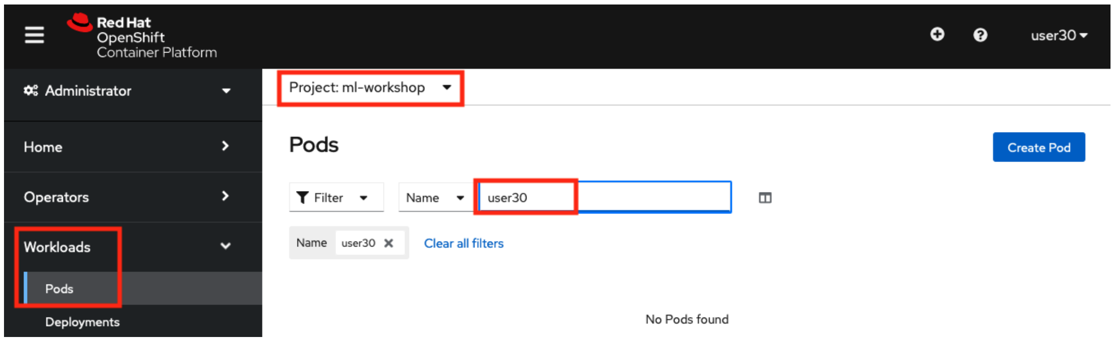
You can see, we source raw data from Kafka and S3 object storage. We use Jupyter notebooks to do some simple data engineering - combining these datasets on customerId using Spark. We then push that prepared data (a CSV file) to another bucket in our S3 object store, called Minio.

Red Hat

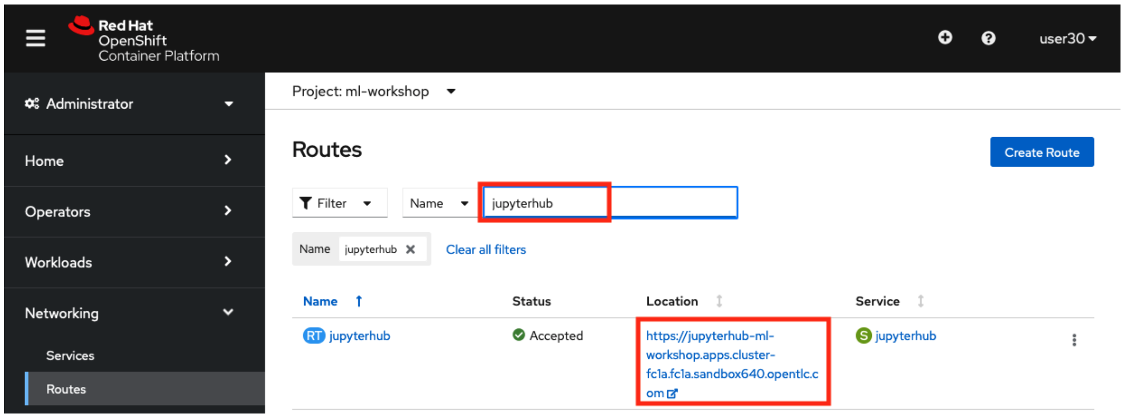
## Instructions for the Spark workshop

Login to OpenShift using the credentials your administrator gave you. Open a second tab on your browser, also logged into OpenShift. Ensure your workshop project ml-workshop is selected.

1. In tab 1, Choose the Administration dropdown , navigate to Workloads -> Pods
   Filter on your username, e.g. in my case user30. There won't be any pods shown - yet.



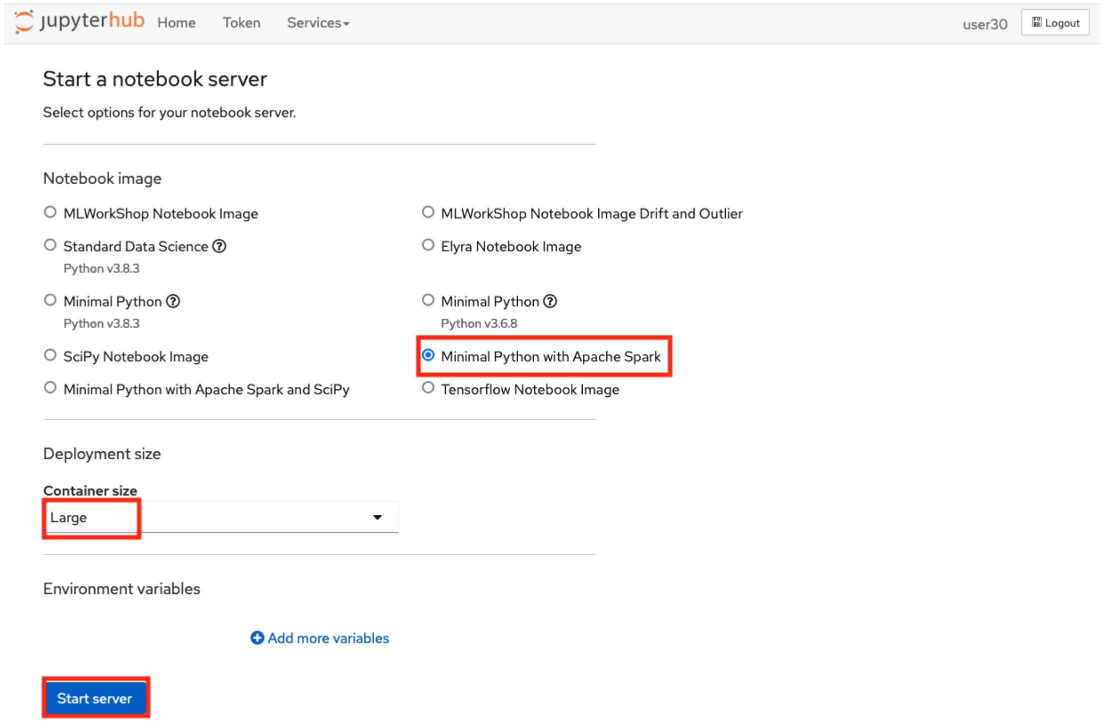
2. In tab 2, Choose the Administration dropdown , navigate to Network -> Routes. Filter on *Jupyterhub* - and open the route.



| Name | Status | Location | Service |
|---|---|---|---|
| RT jupyterhub | Accepted | https://jupyterhub-ml-workshop.apps.cluster-fc1a.fc1a.sandbox640.opentlc.com | S jupyterhub |

Continue with tab 2. In a moment you'll see the following screen. Select

- *Minimal Python with Apache Spark* as shown.
- A *Large* container - allocating the maximum amount of CPU and memory available.



Start a notebook server

Select options for your notebook server.

Notebook image

- ○ MLWorkShop Notebook Image
- ○ MLWorkShop Notebook Image Drift and Outlier
- ○ Standard Data Science (Python v3.8.3)
- ○ Elyra Notebook Image
- ○ Minimal Python (Python v3.8.3)
- ○ Minimal Python (Python v3.6.8)
- ○ SciPy Notebook Image
- [✓ Minimal Python with Apache Spark]
- ○ Minimal Python with Apache Spark and SciPy
- ○ Tensorflow Notebook Image

Deployment size

Container size: Large

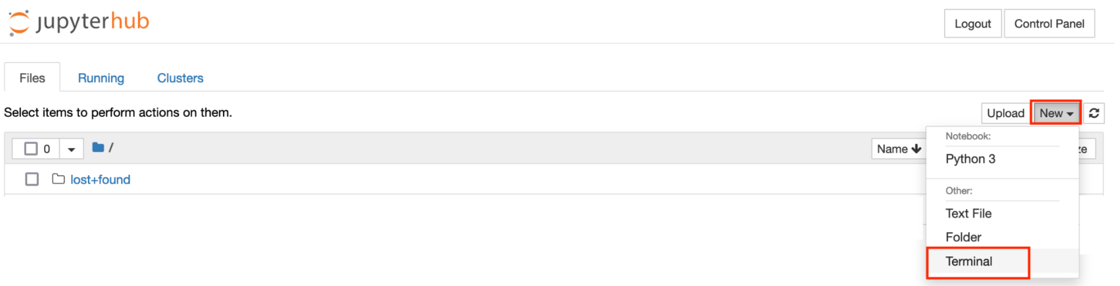
Environment variables

Add more variables

Start server

Start Server. A few moments later the *files* view appears. The first thing we want to do is pull down our notebooks from our repository *https://github.com/masoodfaisal/ml-workshop*.

Choose New Terminal

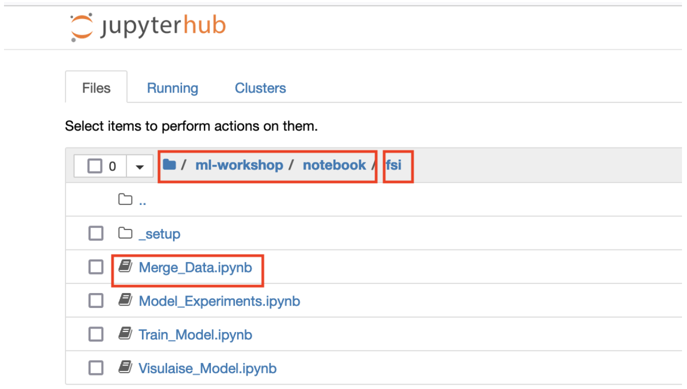


Select items to perform actions on them.

Then paste this command into new Terminal and click enter:

*git clone https://github.com/masoodfaisal/ml-workshop*

Once done you can close that tab. Now refresh the *files* page - you'll see the *ml-workshop* folder. Now drill into ml-workshop -> notebook. Depending on your track ***telco*** or ***fsi***, choose the appropriate subfolder (in this example ***fsi***).

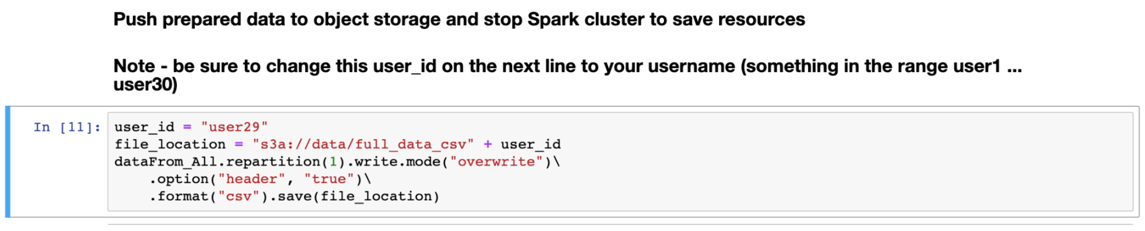


Now open up Merge_Data.ipynb, the file the data engineer uses to prepare their data.

**Before we get going, you need to make a small change to the code.**

Scroll down to the last cell and change the user to match the one provided by your instructor.

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



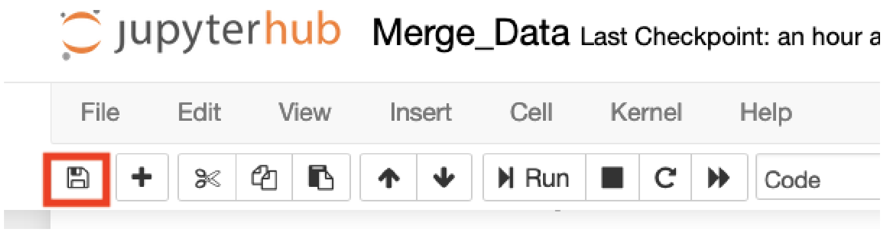
**Push prepared data to object storage and stop Spark cluster to save resources**

**Note - be sure to change this user_id on the next line to your username (something in the range user1 ... user30)**

```
In [11]: user_id = "user29"
         file_location = "s3a://data/full_data_csv" + user_id
         dataFrom_All.repartition(1).write.mode("overwrite")\
             .option("header", "true")\
             .format("csv").save(file_location)
```

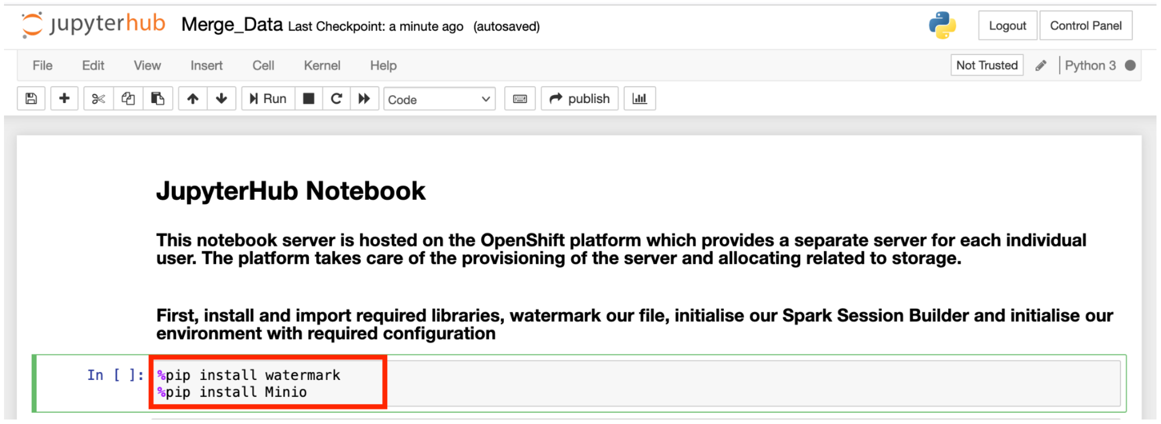
**Change the user_id to be your user_id** and save the notebook - using the save button at the top of the screen.

save the notebook - using the save button at the top of the screen:



You are now good to go!

Place the cursor inside the first cell with the *pip install* commands as shown.



**JupyterHub Notebook**

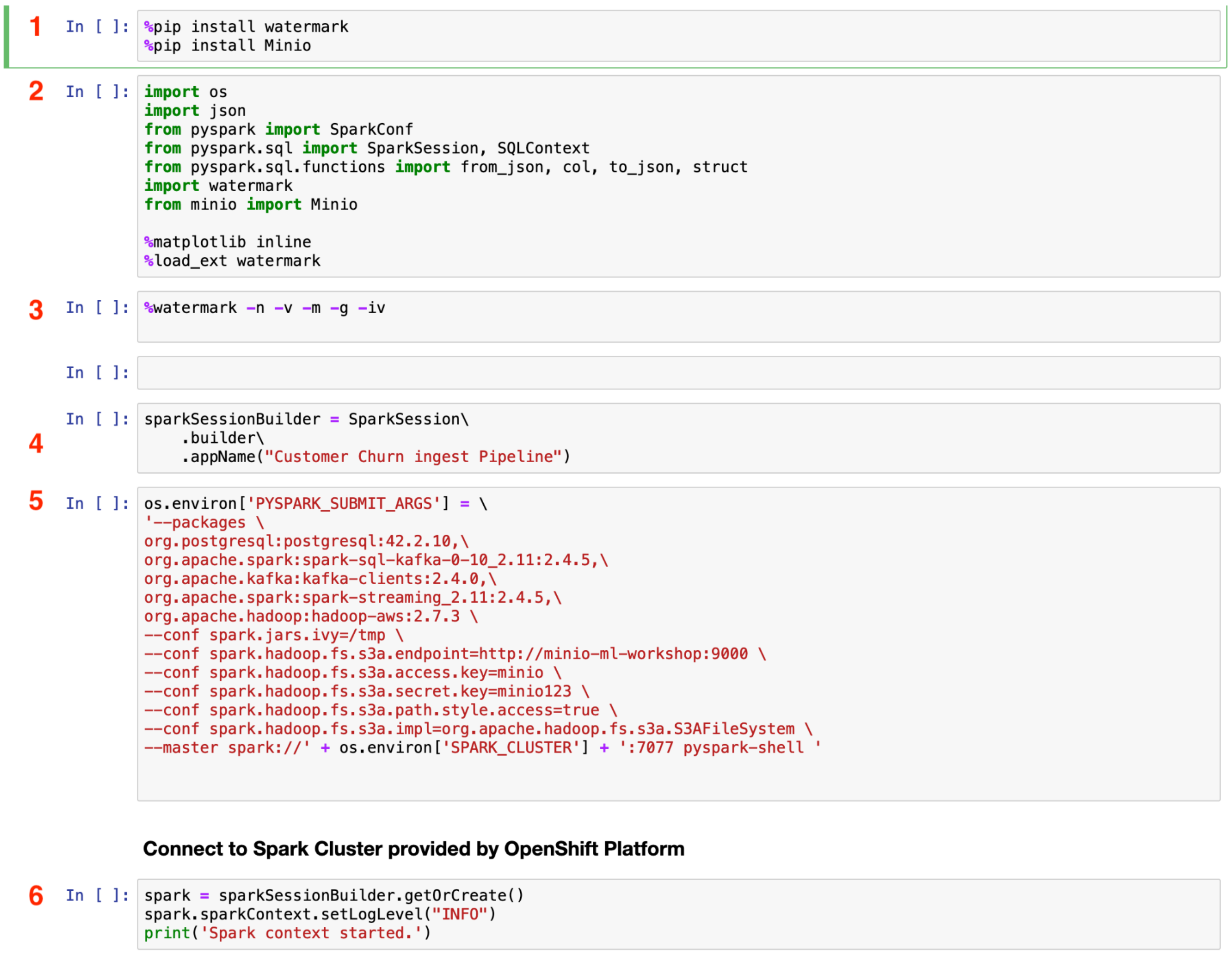
**This notebook server is hosted on the OpenShift platform which provides a separate server for each individual user. The platform takes care of the provisioning of the server and allocating related to storage.**

**First, install and import required libraries, watermark our file, initialise our Spark Session Builder and initialise our environment with required configuration**

```
In [ ]: %pip install watermark
        %pip install Minio
```

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).

Here's a high level description of what's happening in the various cells:



1
```
In [ ]: %pip install watermark
        %pip install Minio
```

2
```
In [ ]: 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
```
In [ ]: %watermark -n -v -m -g -iv
```

```
In [ ]:
```

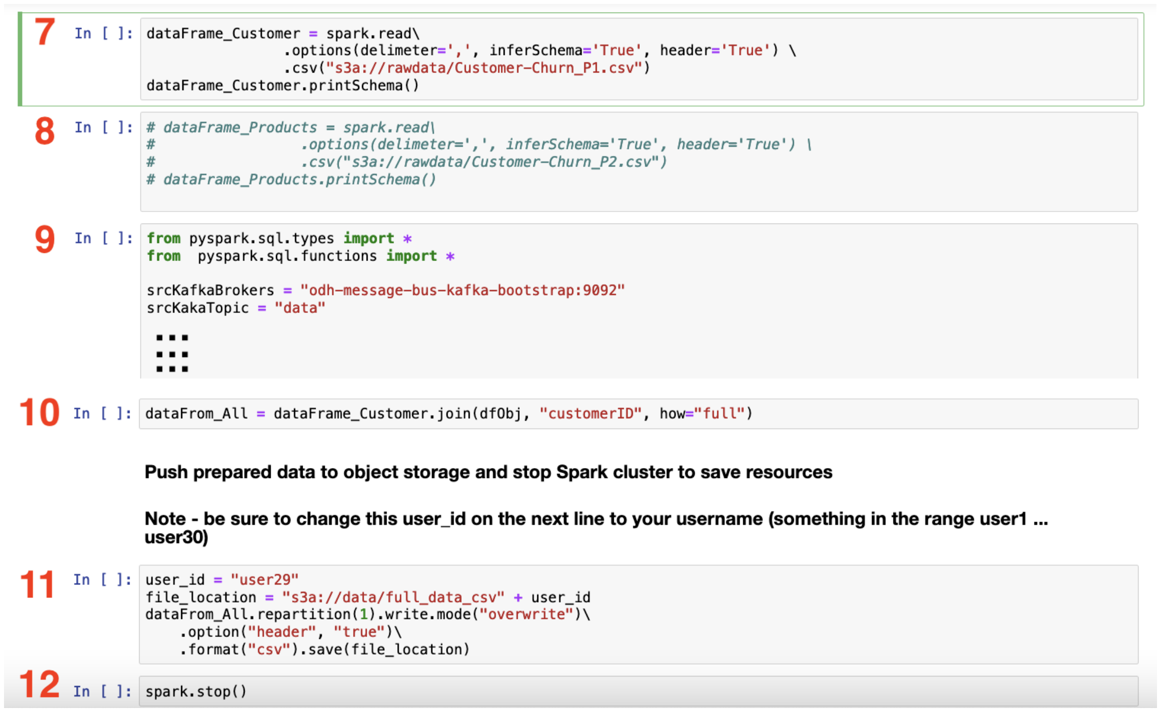
4
```
In [ ]: sparkSessionBuilder = SparkSession\
            .builder\
            .appName("Customer Churn ingest Pipeline")
```

5
```
In [ ]: os.environ['PYSPARK_SUBMIT_ARGS'] = \
        '--packages \
        org.postgresql:postgresql:42.2.10,\
        org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5,\
        org.apache.kafka:kafka-clients:2.4.0,\
        org.apache.spark:spark-streaming_2.11:2.4.5,\
        org.apache.hadoop:hadoop-aws:2.7.3 \
        --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 \
        --master spark://' + os.environ['SPARK_CLUSTER'] + ':7077 pyspark-shell '
```

**Connect to Spark Cluster provided by OpenShift Platform**

6
```
In [ ]: spark = sparkSessionBuilder.getOrCreate()
        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 create a Spark session, a precursor to firing up our own Spark server.
5. 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.
6. Here we actually start our Spark server. This cell can take several minutes to start.



7
```
In [ ]: dataFrame_Customer = spark.read\
                        .options(delimeter=',', inferSchema='True', header='True') \
                        .csv("s3a://rawdata/Customer-Churn_P1.csv")
        dataFrame_Customer.printSchema()
```

8
```
In [ ]: # dataFrame_Products = spark.read\
        #                  .options(delimeter=',', inferSchema='True', header='True') \
        #                  .csv("s3a://rawdata/Customer-Churn_P2.csv")
        # dataFrame_Products.printSchema()
```

9
```
In [ ]: from pyspark.sql.types import *
        from  pyspark.sql.functions import *

        srcKafkaBrokers = "odh-message-bus-kafka-bootstrap:9092"
        srcKakaTopic = "data"

         ⋮
```

10
```
In [ ]: dataFrom_All = dataFrame_Customer.join(dfObj, "customerID", how="full")
```

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

**Note - be sure to change this user_id on the next line to your username (something in the range user1 ... user30)**

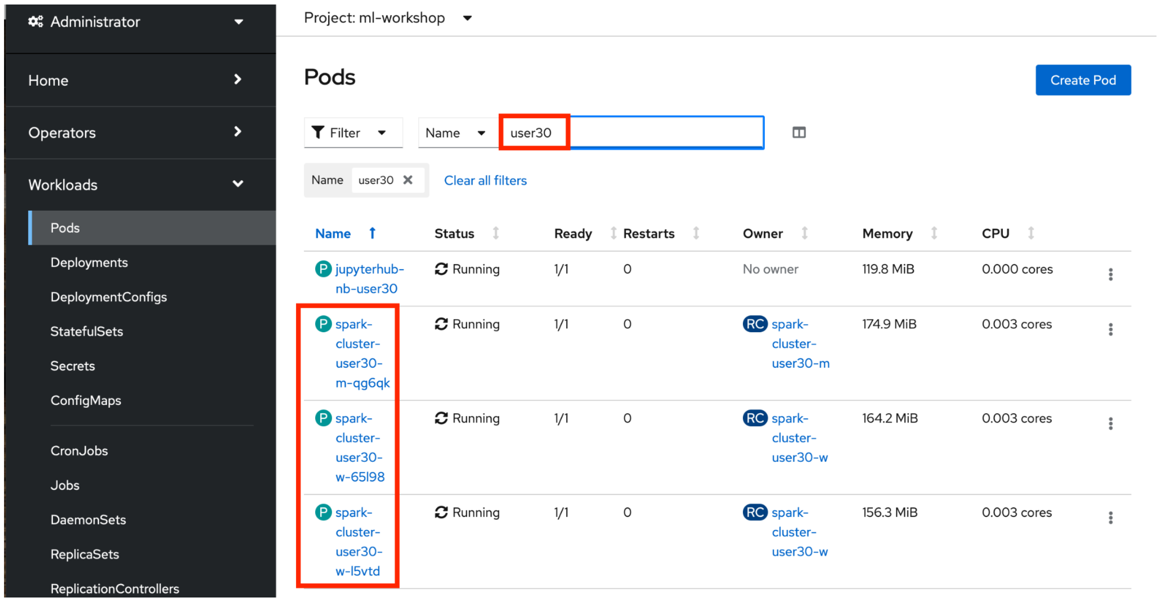
11
```
In [ ]: user_id = "user29"
        file_location = "s3a://data/full_data_csv" + user_id
        dataFrom_All.repartition(1).write.mode("overwrite")\
            .option("header", "true")\
            .format("csv").save(file_location)
```

12
```
In [ ]: spark.stop()
```

7. Here we pull in our data from S3 - our CSV based demographic data for each of our approximately 7000 customers.
8. Commented out - ignore
9. Here we pull in our data from Kafka - our product consumption data for each of our approximately 7000 customers.
10. We join these 2 datasets, on the common column to each: *customerID*.
11. We push our data to our object store - filename contains our username.
12. We are all done now - we stop our Spark server.

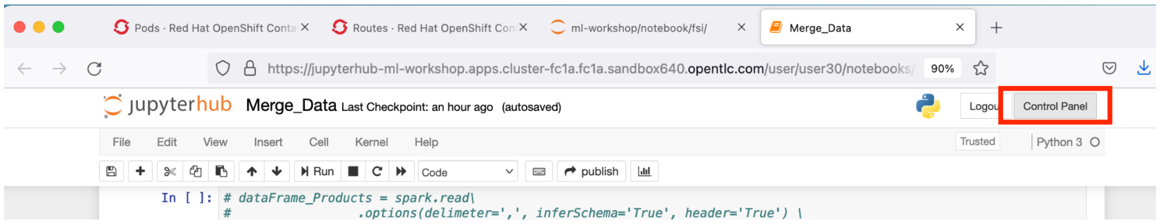
This is our data engineering workshop finished. It's a simple exercise, though the same toolset could be used for much more complex data engineering tasks.

Before we move on, as evidence of this self provisioned cluster, dedicated entirely to you as a user, move back to the Pods view, you saw above, keeping your username as a filter. Notice OpenShift has created a 3-node Spark cluster for us:



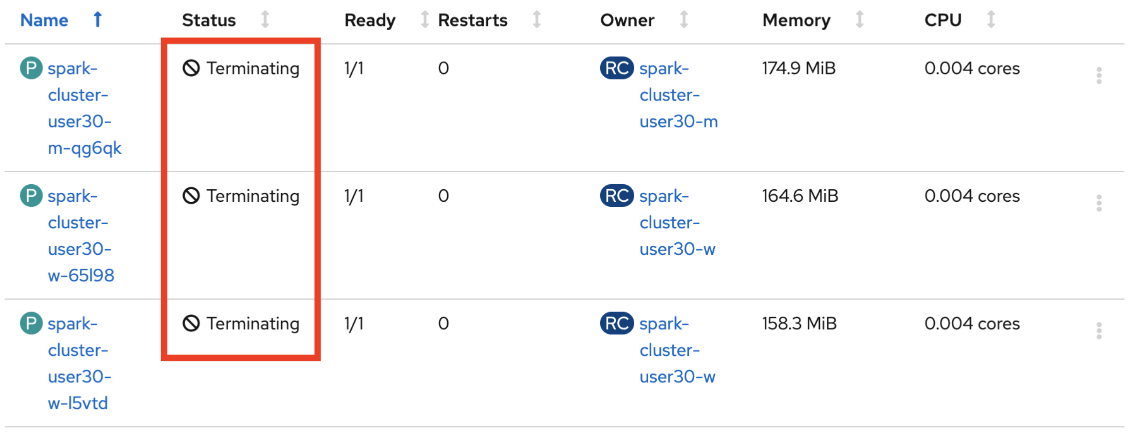
| Name | Status | Ready | Restarts | Owner | Memory | CPU |
|---|---|---|---|---|---|---|
| P jupyterhub-nb-user30 | Running | 1/1 | 0 | No owner | 119.8 MiB | 0.000 cores |
| P spark-cluster-user30-m-qg6qk | Running | 1/1 | 0 | RC spark-cluster-user30-m | 174.9 MiB | 0.003 cores |
| P spark-cluster-user30-w-65l98 | Running | 1/1 | 0 | RC spark-cluster-user30-w | 164.2 MiB | 0.003 cores |
| P spark-cluster-user30-w-l5vtd | Running | 1/1 | 0 | RC spark-cluster-user30-w | 156.3 MiB | 0.003 cores |

Now we need to close down our Jupyter server. Choose *Control Panel* and shown – then on the next screen, choose Stop My Server.


Pods · Red Hat OpenShift Conta
Routes · Red Hat OpenShift Con
ml-workshop/notebook/fsi/
Merge_Data
https://jupyterhub-ml-workshop.apps.cluster-fc1a.fc1a.sandbox640.opentlc.com/user/user30/notebooks/
90%
jupyterhub
Merge_Data Last Checkpoint: an hour ago (autosaved)
Logou
Control Panel
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code
publish

```
In [ ]: # dataFrame_Products = spark.read\
        #                 .options(delimeter=',', inferSchema='True', header='True') \
```

Immediately move back to your *Pods* screen – and 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 |

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.