Red Hat
OpenShift
Data Science

Anomaly Detection and Failure Prediction workshop

Red Hat Summit 2023

Audrey Reznik Guidera	*Sr. Principal Software Engineer*
Christina Xu	*Software Engineering Intern*
Eli Guidera	*Sr. Architect (Retired)*
Guillaume Moutier	*Sr. Principal Product Marketing Manager*

Acknowledgements

This workshop is a result of a group of talented people coming together to share their expertise and knowledge. From Data Scientists to MLOPs Engineers, there has been a great deal of input from many role types.

Thank you everyone for your time, effort and Red Hatter passion - Y'all Rock!

Cameron Garrison - RHODS Engineering Intern

Christina Xu - RHODS Engineering Intern

Troy Nelson - RHODS Engineering Intern

Cory Latschkowski - Sr. Specialist Solutions Architect

Trevor Royer - Architect

Mario Bogoevici - Chief Solutions Architect - Financial Services

Hema Veeradhi - Sr. Software Engineer

Andy Block - Distinguished Architect - Services

Eli Guidera - Independent Consultant (Sr Architect)

This workshop will teach you how to use OpenShift and the Red Hat OpenShift Data Science platform to examine and understand anomalies and predict failures in water pumps. We will work through the following agenda:

Agenda:

1 Introduction 2:00-2:15pm

OpenShift Data Science is a cloud service that gives data scientists and developers a powerful AI/ML platform for building intelligent applications. Data scientists and developers can collaborate to quickly move from experiment to production in a consistent environment.

Available as an add-on cloud service to Red Hat OpenShift Dedicated and Red Hat OpenShift Service on AWS or as a self-managed software product, OpenShift Data Science allows data scientists to quickly develop, train, and test machine learning models using the JupyterLab interface. After models are developed, data scientists can use GitHub integration to trigger updated builds of OpenShift applications created by developers.

In this introductory workshop, you'll learn how to:

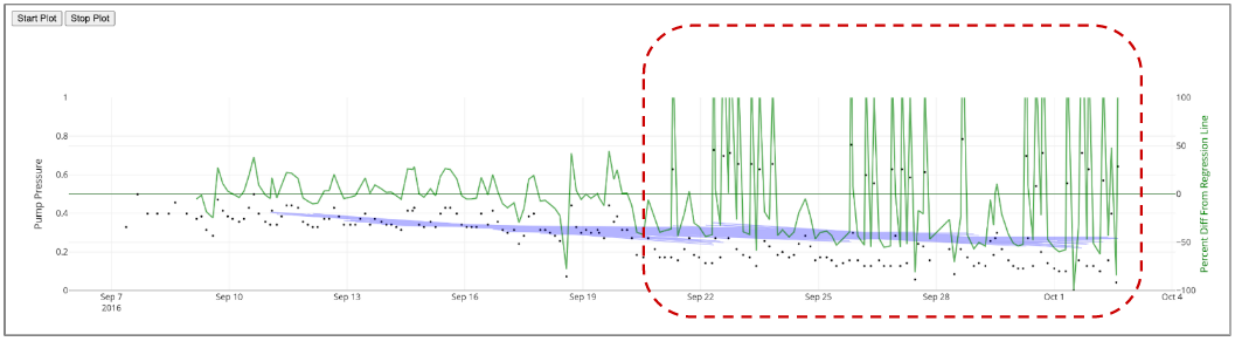
Use the Red Hat OpenShift Data Science platform to:

1. Deploy an Anomaly Detection 'containerized web application' (in the Red Hat OpenShift platform) to display time series data which contains an anomaly.
2. Generate Synthetic sensor data to mimic real-time data generated by an Edge device. You will use this data in the next step.
3. Deploy a Failure Prediction 'containerized web application' (in the Red Hat Openshift platform) and use the synthetic sensor data to predict the failure of a mechanical device.

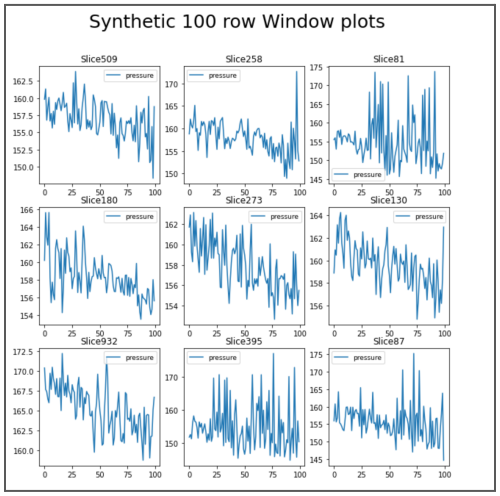
We'll start from a sensor dataset like this:

timestamp	sensor_25	sensor_11	sensor_36	sensor_34
2018-04-10 09:55:00	0.886128288849588	0.6936344767141810	0.21762059617623900	0.25318778143401500
2018-04-10 09:56:00	0.9204208188441080	0.7000403380548420	0.0968758054521226	0.2539990089674020
2018-04-10 09:57:00	0.9202596186121920	0.7037891972869380	0.09844494043538750	0.2719518531347420
2018-04-10 09:58:00	0.8767057041000580	0.7074345247993290	0.11346761648322900	0.26238610341377100
2018-04-10 09:59:00	0.907298589265789	0.7092217106877420	0.11872560793926700	0.24854953238656100
2018-04-10 10:00:00	0.9458851484833250	0.7088721741223050	0.12288425435926700	0.25818383654697300
2018-04-10 10:01:00	0.9458851484833250	0.7088721741223050	0.12288425435926700	0.25818383654697300
2018-04-10 10:02:00	0.8725669047299990	0.7112236359658490	0.12616730006050600	0.26549390466843700
2018-04-10 10:03:00	0.8795137723375100	0.7087605920157160	0.11858339137158600	0.2587340761266980
2018-04-10 10:04:00	0.8923600377327950	0.7036476602524060	0.126257236501765	0.27057234537967100

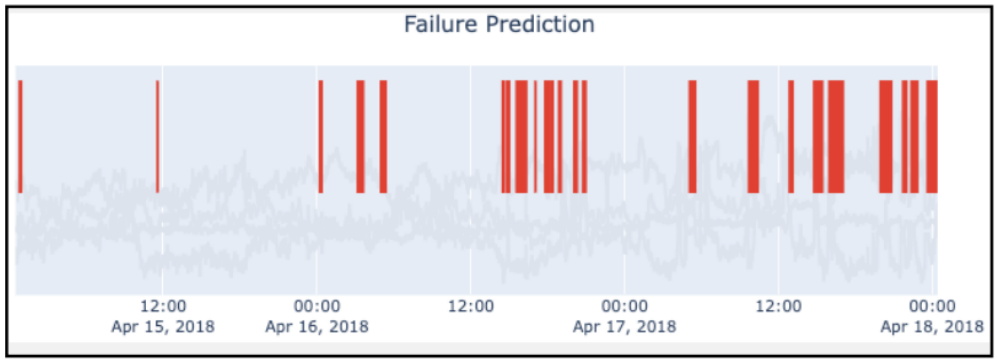
We will learn how to recognize anomalies in our data like this:



We will then generate synthetic data to mimic ‘live’ sensor data streaming from an Edge device connected to a Mechanical Pump. Plotted synthetic sensor data will look like this:



From this synthetic sensor data we will use our failure prediction model to predict when our mechanical device will fail based on sensor readings like this:



The failure prediction model will be accessible, via a Failure Prediction web application which we will deploy on the OpenShift (kubernetes) platform. We can do all of this without having to install anything on your computer, thanks to Red Hat OpenShift Data Science!

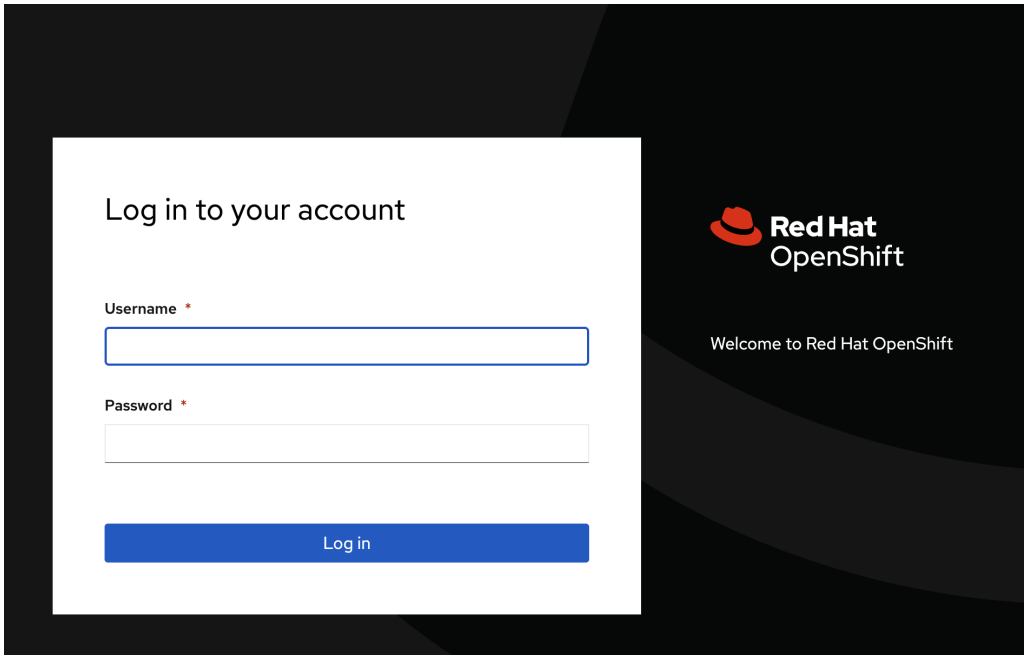
If you're ready, let's start!

2 Log into the Red Hat OpenShift Data Science Platform 2:15-2:30

Use the url, from your instructor, to access the Red Hat OpenShift Data Science platform. The url will look similar to the following example:

https://console-openshift-console.apps.ieee.d7se.p1.openshiftapps.com

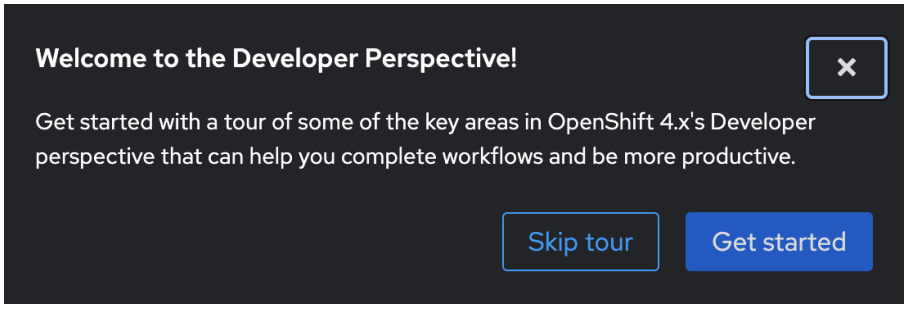
You will see the following login page appear.



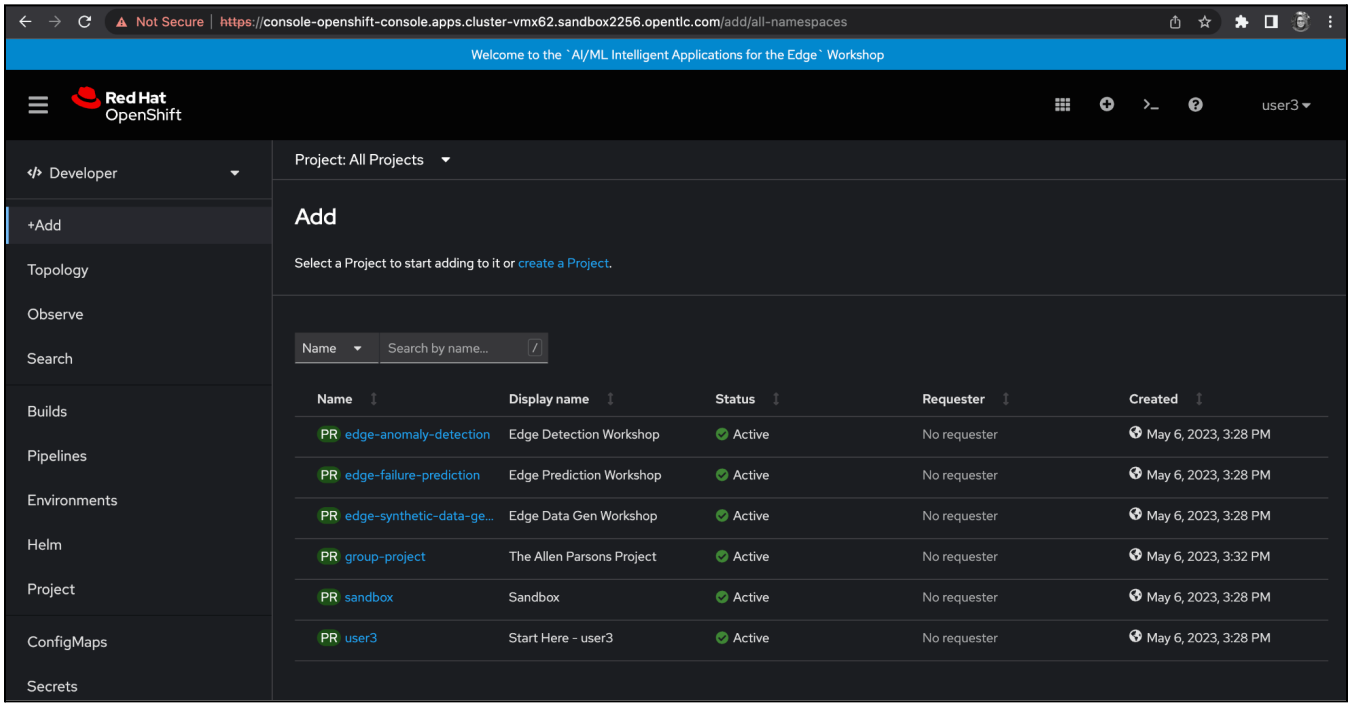
Login in with your **username**: user<#> & **password**: openshift.

Note: your instructor will hand out usernames once the workshop starts.

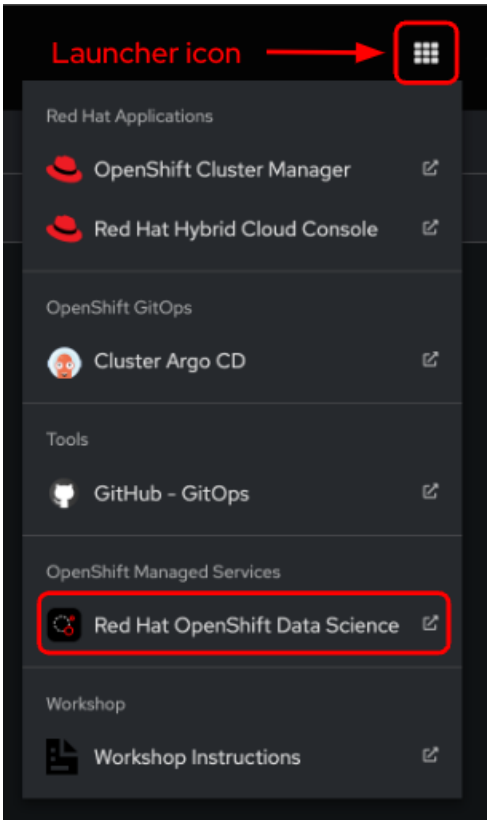
Upon successful login, you will see the Red Hat OpenShift Dedicated environment. You may also see a “Welcome to the Developer Perspective!” dialog box. Click “Skip tour” to close this box.



The **Red Hat OpenShift Dedicated platform** will be displayed. This is a kubernetes platform from which you can create containerized applications. We will come back to using OpenShift. We are going to first look at and use the **Red Hat OpenShift Data Science platform**.

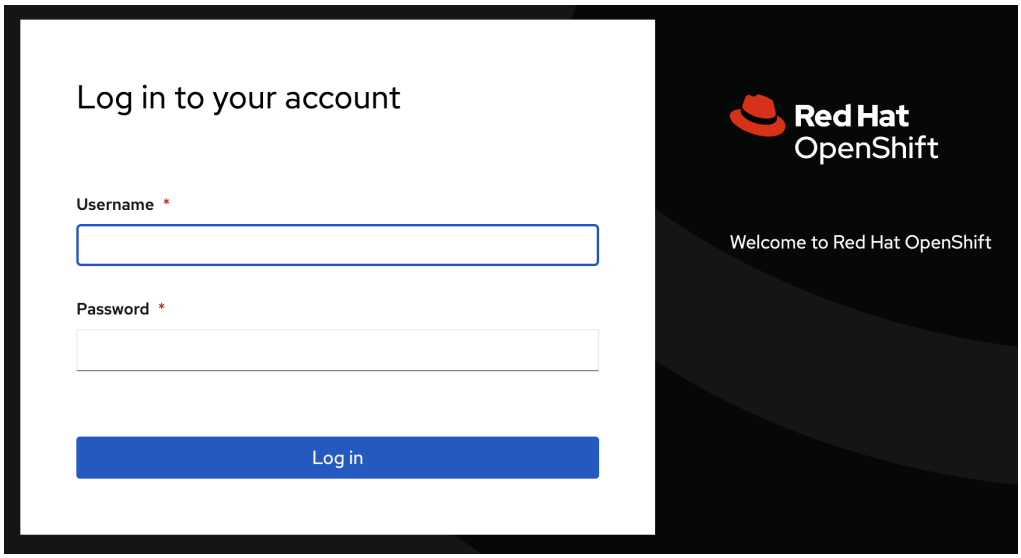


The Red Hat OpenShift Data Science platform (dashboard) is accessed by **selecting** the '*launcher icon*' in the upper right corner. **Select the option** *'Red Hat OpenShift Data Science'.* See the following screenshot for the location of the Launcher icon.

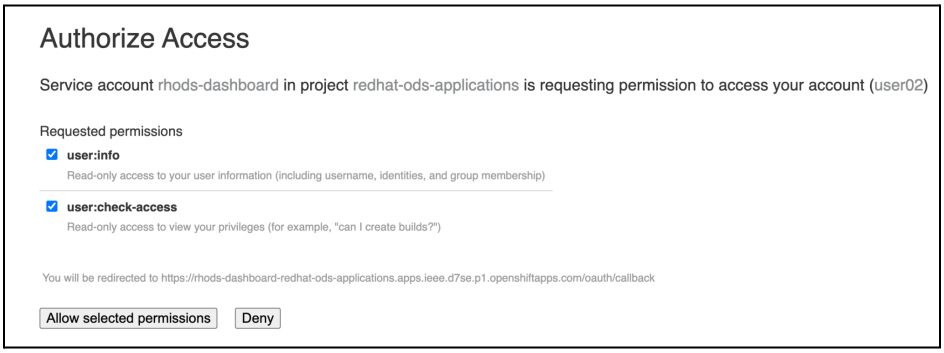


After selecting the '**Red Hat OpenShift Data Science**' option, a dialog box with a "**Log in with OpenShift**" button will appear. Click this button to proceed with logging into the Red Hat OpenShift Data Science (RHODS) dashboard.

You will encounter another Login page. Enter the username and password you were given at the beginning of the workshop, then click the ‘Login’ button.

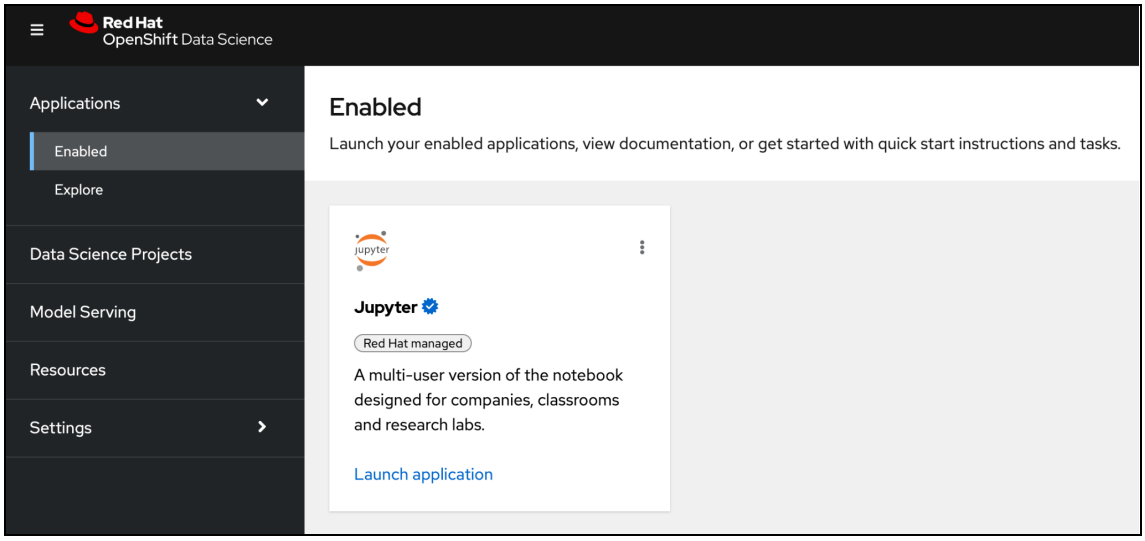


You may see an ‘Authorize Access’ screen. Click the ‘Allow selected permissions’ button.

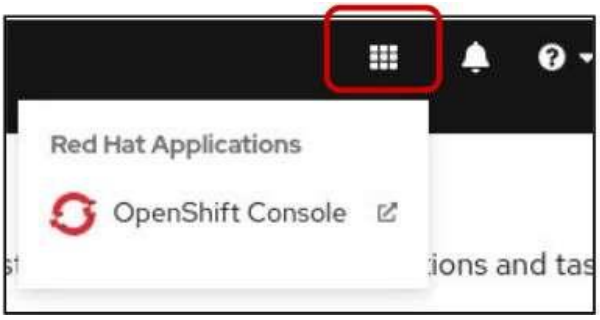


You will now be logged into the Red Hat OpenShift Data Science platform.

The platform, by default, displays all currently enabled applications. In this case, Jupyter is enabled.



Note: you can always return to the **OpenShift Console** by using the launcher button (the black-and-white icon that looks like a grid), and choosing **OpenShift Console**.



Let's look at some of the menu items available in the Red Hat OpenShift Data Science platform:

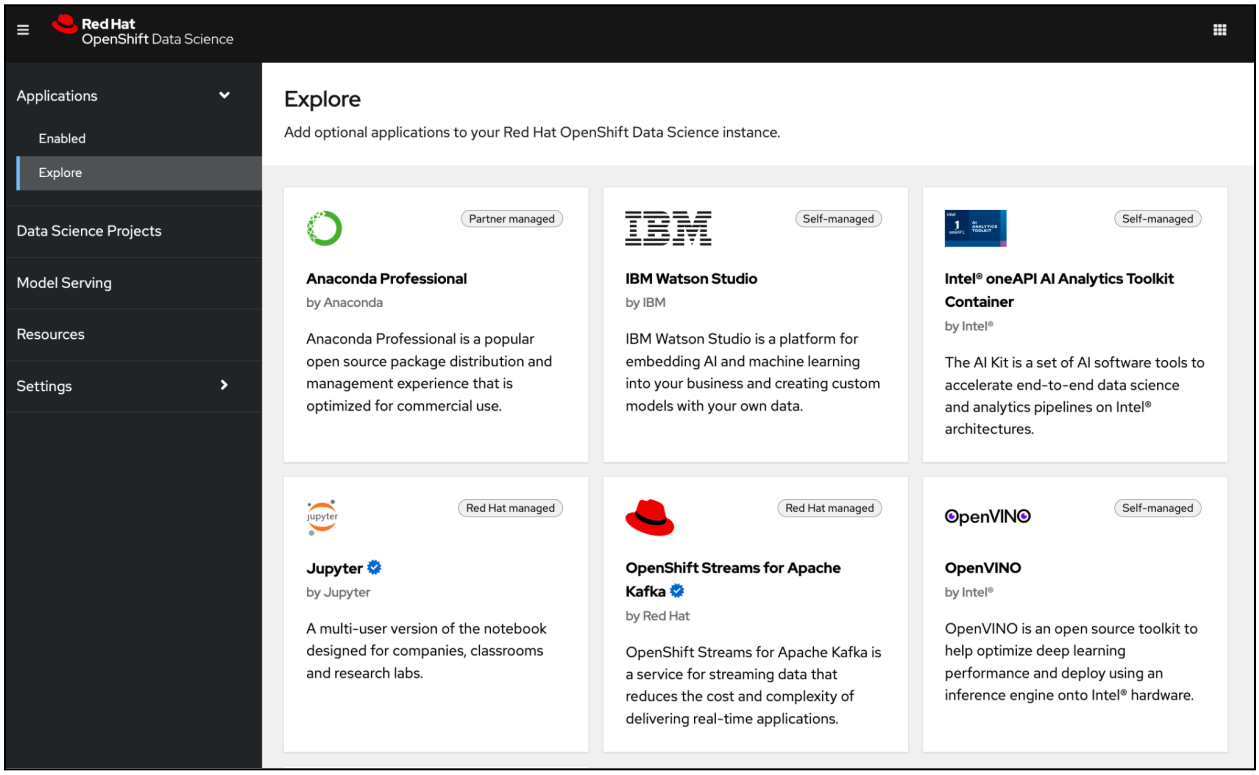
1. Applications
2. Data Science Projects
3. Model Serving
4. Resources
5. Settings (note: this option is only seen and available to administrators)

1. The **Applications** menu option takes you to a variety of available applications. These include:

- Red Hat managed cloud services: Applications that have been developed by Red Hat, such as OpenShift Streams for Apache Kafka.
- Partner managed services: Services created by one of Red Hat's external partners, such as Intel oneAPI AI Analytics Toolkit Container. These services are available to you in the OpenShift Data Science dashboard.
- Self-managed software: Other types of software developed by one of our external partners, such as OpenVino. This software is available to you in the Openshift Data Science dashboard.

The **Applications** menu items allows you to see already **'Enabled'** applications. You can launch your enabled applications, view documentation, or get started with quick instructions and tasks.

The **Applications** menu item also allows you to **'Explore'** all available applications by choosing the Explore menu item. In the Explore menu, you can add optional applications to your Red Hat OpenShift Data Science instance.



2. The **Data Science Projects** menu option allows you to view your existing projects or create new projects. When you create a Data Science project you are able to add Workbenches, Cluster Storage, Data Connections, Models and Servers. This enables you to quickly access your work for various projects.

3. The **Model Serving** menu option allows you to view the health and performance of your deployed models.

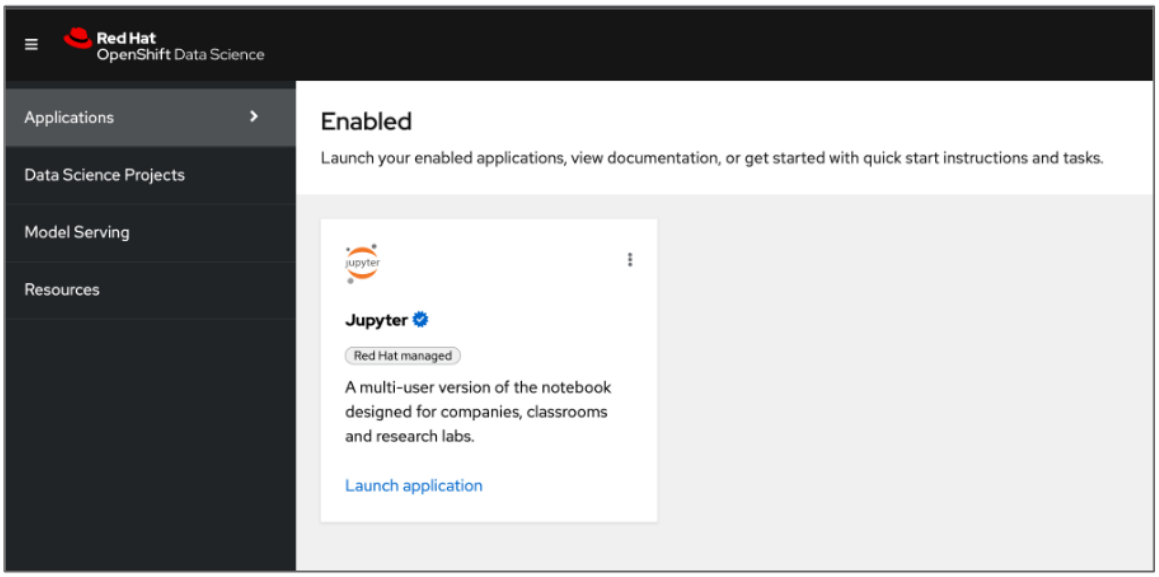
4. The **Resources** menu option allows you to access all the learning resources for Red Hat OpenShift Data Science and supported applications.

5. *If you login as an admin,* you will see an additional **Settings** menu option. (The above screen capture comes from an admin user dashboard view.) The Settings option allows admins to:
    a. Import, delete and modify Notebook Images
    b. Update Cluster Settings (global settings for all users)
    c. Manage Users by defining OpenShift group membership for Data Science administrators and users.

Let's start the lab by launching our Jupyter notebook server!

3 Start your Jupyter Image Notebook server

Let's launch the Jupyter Notebook server and customize options for your Jupyter Notebook server! Click the Launch application url at the bottom of the 'Jupyter' card..

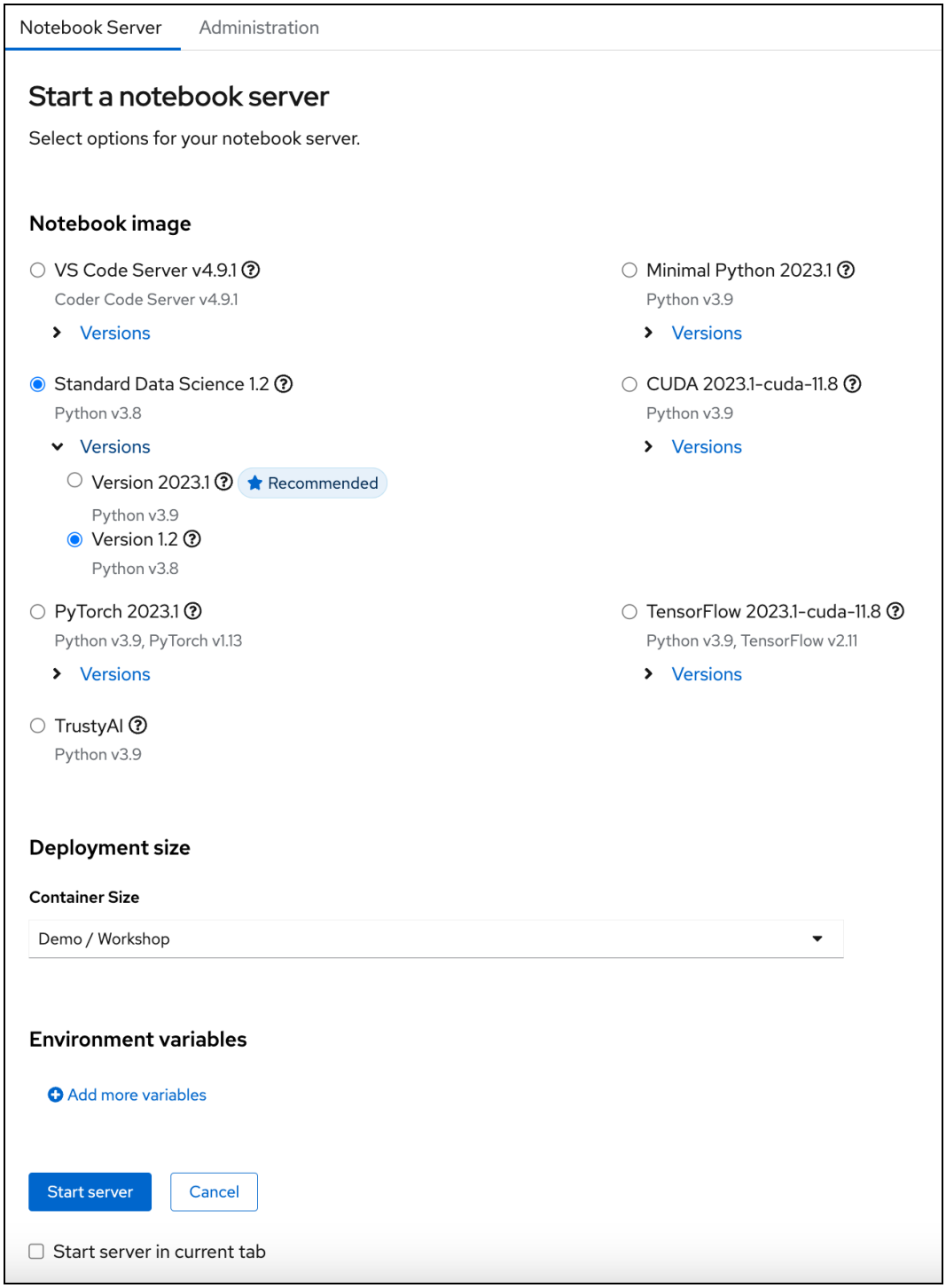


Start a notebook server.

When you first gain access to Jupyter, a configuration screen gives you the opportunity to select a notebook server image and configure the deployment size and environment variables for your data science project.

You can customize the following options:

- Notebook image
- Deployment size
- Environment variables



The following subsections list which prerequisite(s) you need to choose for this activity.

Notebook image

You can choose from a number of predefined images. When you choose a predefined image, your JupyterLab instance will then contain the associated libraries and packages that you need to do your work.

Available notebook images include:

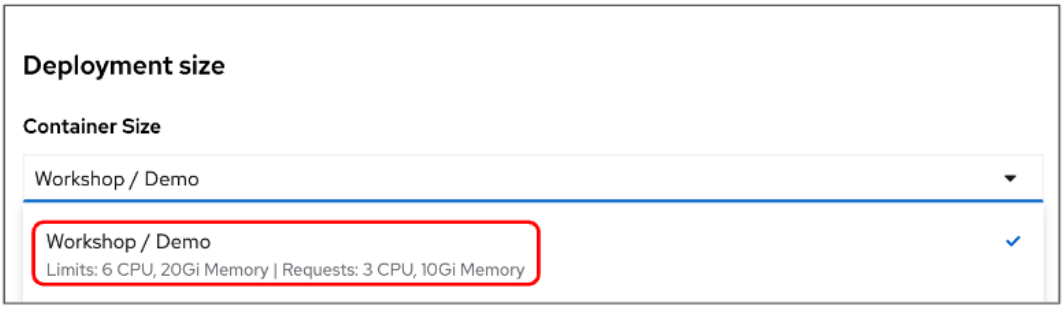
- VS Code Server
- **Standard Data Science**
- PyTorch
- TrustyAI
- Minimal Python
- CUDA
- Tensorflow

For this learning path, choose the **Standard Data Science** notebook image. Make certain you choose Version 1.2 Python v3.8.

> **Note:** not sure what Notebook images contain? Click on the "?" beside each image to learn what packages/libraries are included within each image.

Deployment size

You can choose different deployment sizes (resource settings) based on the type of data analysis and machine learning code you are working on. Each deployment size is pre-configured with specific CPU and memory resources.



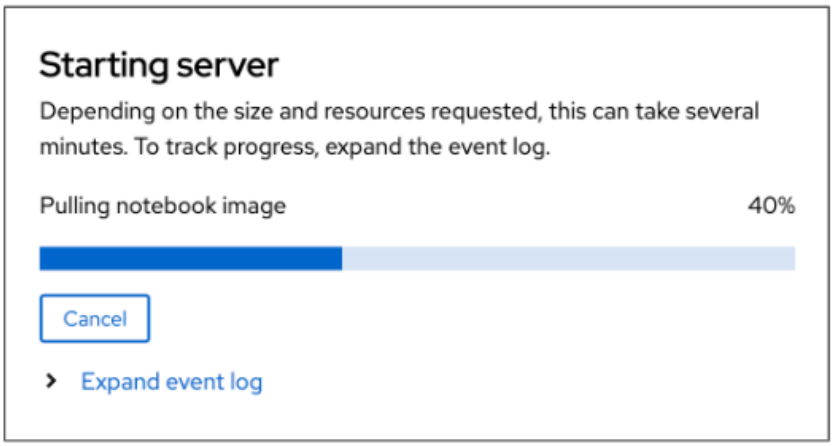
For this learning path, the container size has been pre-configured to 6 CPU, 20 Gi Memory

Environment variables

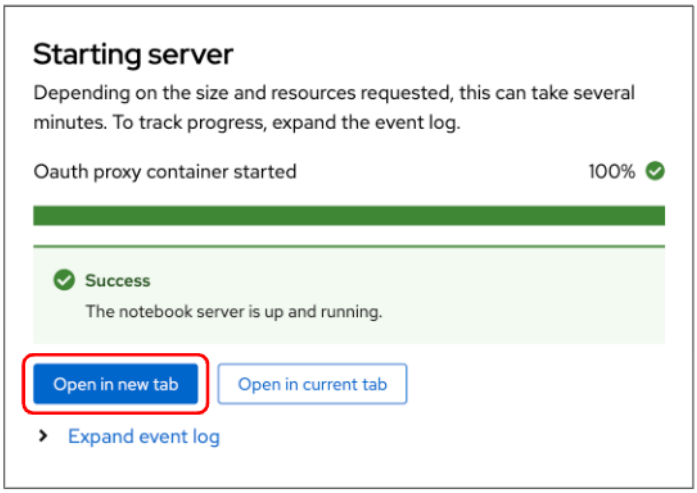
The environment variables section is useful for injecting dynamic information that you don't want to save in your notebook.

> **Note**: This learning path does not use any environment variables.

If you are satisfied with your notebook server selections, click the **Start Server** button to start the notebook server. The following “Starting server” dialog box will appear.

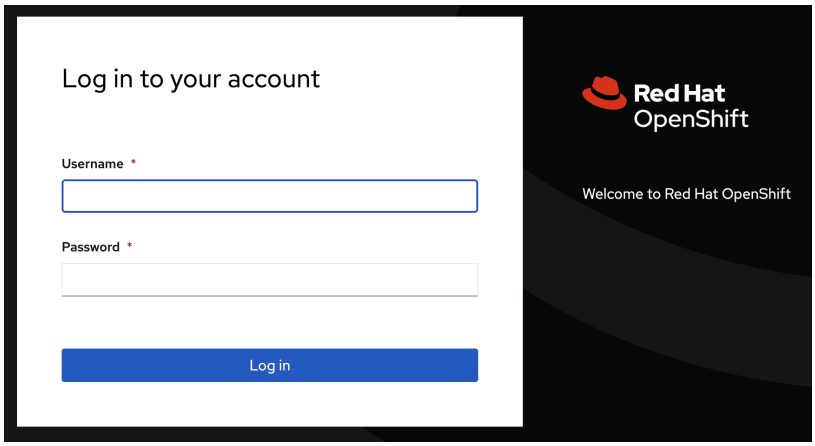


Your Notebook server may take a minute to spin up. If you are interested, you can click the “Expand event log” to see where in the process your Notebook Server is. Once your server is ready, you will see the progress bar turn green and reach 100%.

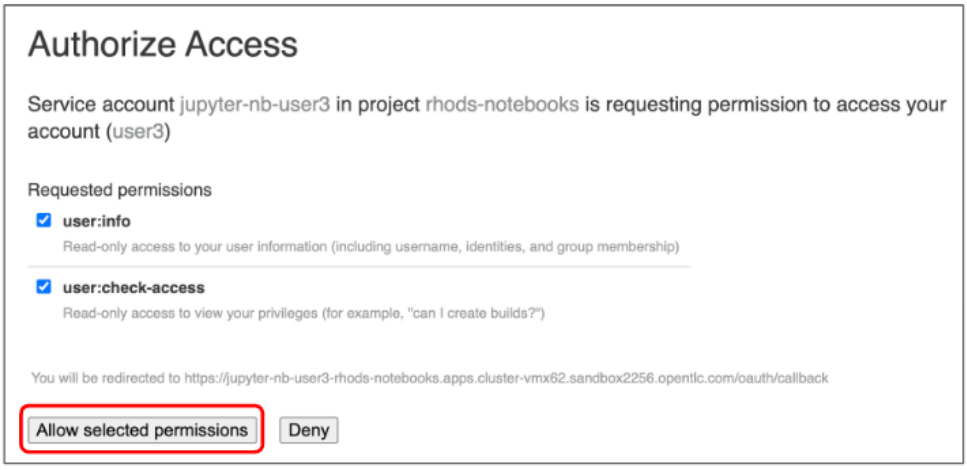


You can now choose to start your Jupyter Lab environment in the current tab or open the environment in a new tab. Click the '*Open in new tab*' **button** to launch Jupyter Lab in a new tab.

You will see another Login page. Enter the username and password you were given at the beginning of the workshop, then click the 'Login' button.



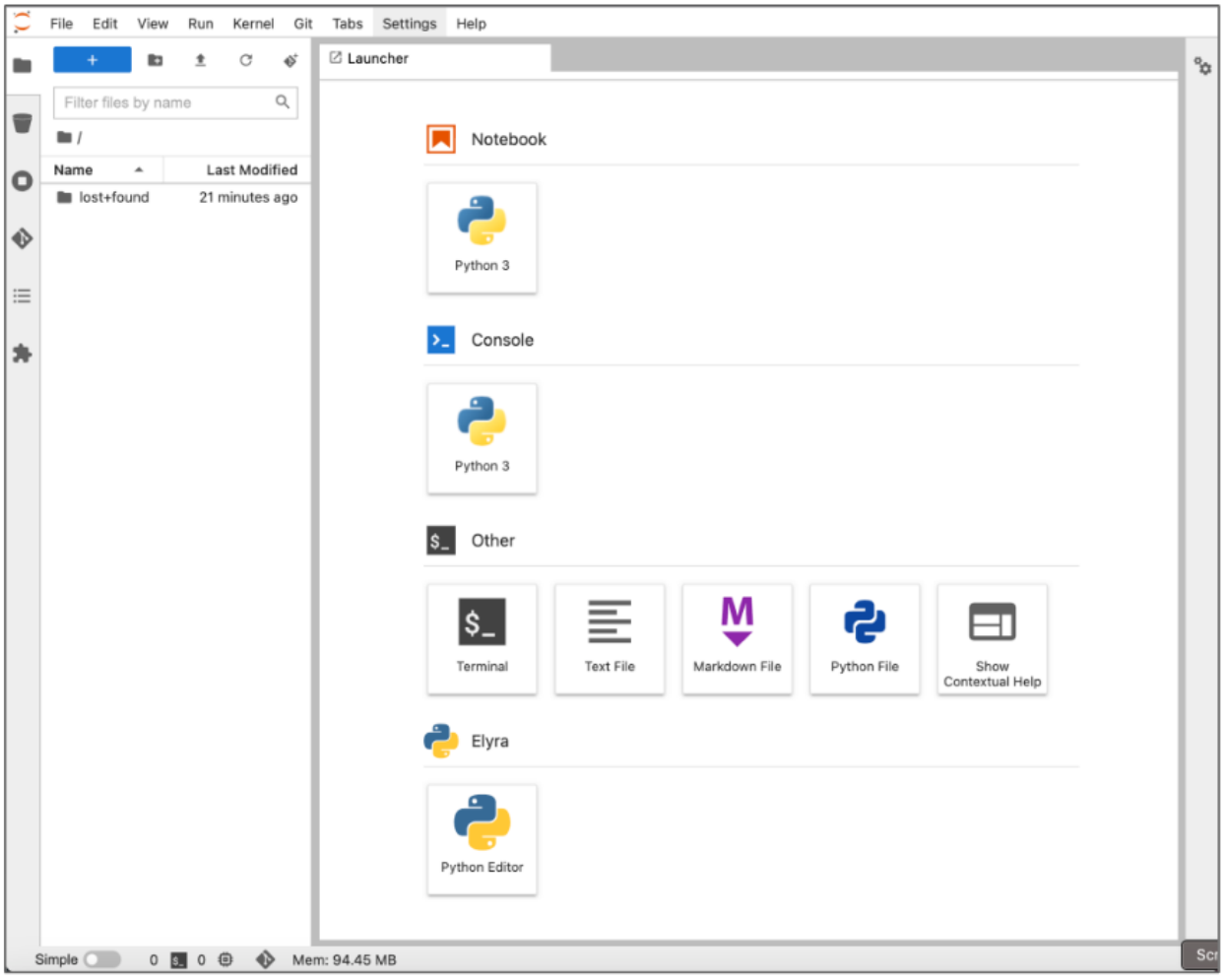
After you log in, you may see an '*Authorize Access*' screen. Click the 'Allow selected permissions' button to continue.



If your login is successful, you will have started your Jupyter notebook server and will be taken into the Jupyter Lab Environment.

3a The Jupyter Lab Environment

You are now inside your Jupyter Lab environment. As you can see, it's a web-based environment, but everything you'll do here is in fact happening on the Red Hat OpenShift Data Science cluster. This means that without having to install and maintain anything on your own computer, and without disposing of lots of local resources like CPU and RAM, you can still conduct your Data Science work in this powerful and stable managed environment.



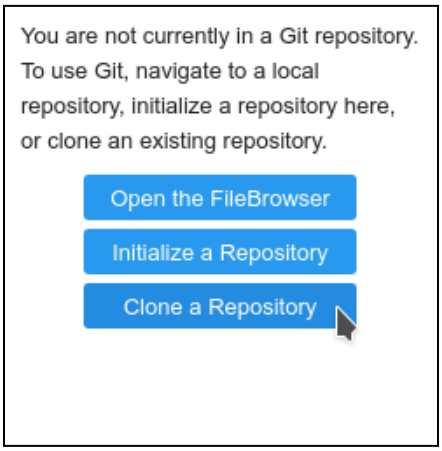
In the "file-browser" like window you're in right now, you'll find the files and folders that are saved inside your own personal space inside Red Hat OpenShift Data Science. It's

pretty empty right now though… So the first thing we will do is to bring the content of the workshop inside this environment.

3b Clone a github repository

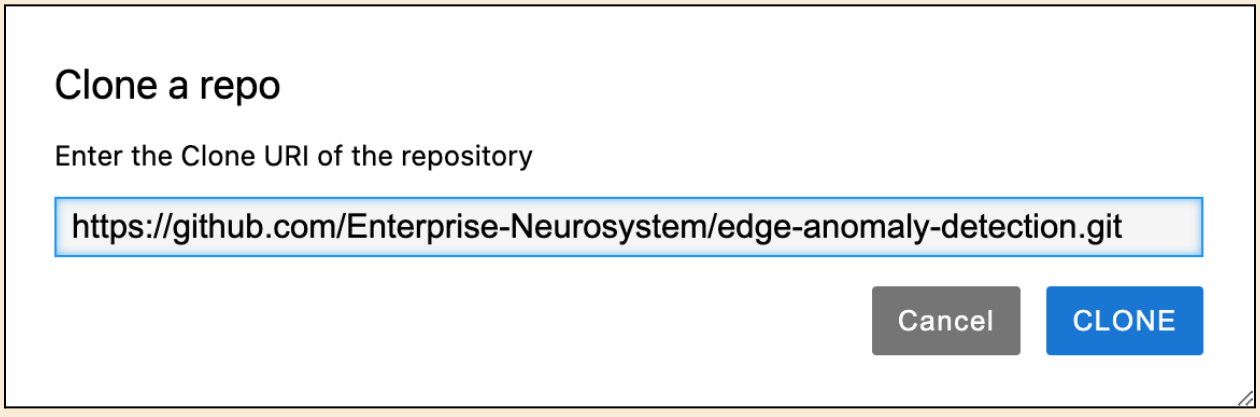
- On the left toolbar, click on the Git icon:

- Then click on Clone a Repository:



We are going to first look at Anomaly Detection. Therefore, enter the following git repo for Anomaly Detection.

- Enter Clone URI of the repository, `https://github.com/Enterprise-Neurosystem/edge-anomaly-detection.git`, then **click the CLONE button**.

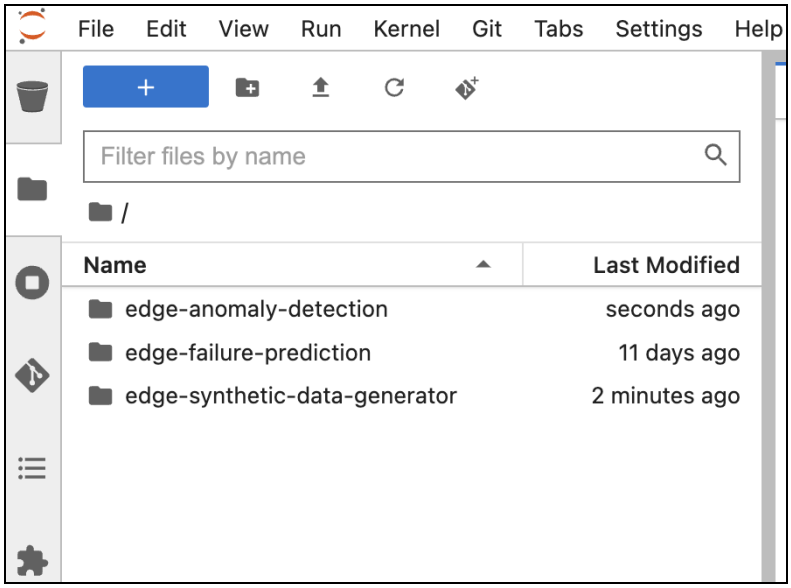


- It takes a few seconds, after which you can view the newly-created folder, `edge-anomaly-detection`

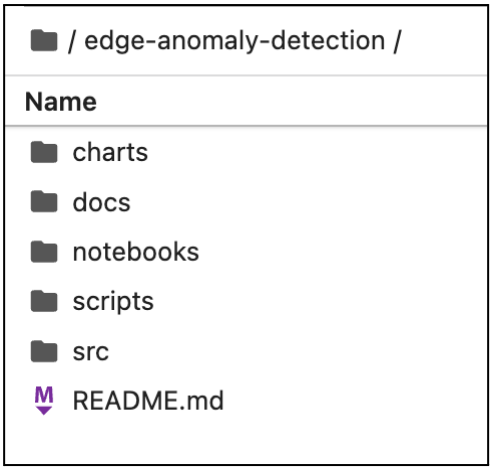
Note: After you have cloned the edge-anomaly-detection repository, proceed with cloning the following 2 repositories that will be used later in this workshop.

1. https://github.com/Enterprise-Neurosystem/edge-synthetic-data-generator.git
2. https://github.com/Enterprise-Neurosystem/edge-failure-prediction.git

After you've cloned the repositories, the `edge-anomaly-detection, edge-synthetic-data-generator and edge-failure-prediction` folders will appear under the **Name** pane.



Let's open up the edge-anomaly-detection folder by double clicking on the edge-anomaly-detection folder name.



The folder should contain the following sub-folders:

- `charts`
- `docs`
- `notebooks`
- `scripts`
- `src`

Change into the notebooks sub-folder. You will see the following 2 files: `01-sandbox.ipynb` & `02-anomaly-detection-notebook.ipynb`

If you have not worked with Jupyter Notebooks before, continue to the next section - Introduction to Jupyter Notebooks.

If you are familiar with Jupyter Notebooks, continue to the section - Anomaly Detection.

3c Introduction to Jupyter Notebooks

This section provides a small introduction on how to use Jupyter Notebooks. If you're already at ease with Jupyter, you can directly head to the next section - Anomaly Detection.

What's a notebook?

- A notebook is an environment where you have *cells* that can display formatted text, or code.

This is an empty cell:

And a cell where we have entered some code:

- Code cells contain Python code that can be run interactively. That means you can modify the code, then run it. The code will not run on your computer or in the browser, but directly in the environment you are connected to, Red Hat OpenShift Data Science in our case.
- To run a code cell, just select it (click in the cell, or on the left side of it), and click the Run/Play button from the toolbar (you can also press CTRL+Enter to run a cell, or Shift+Enter to run the cell and automatically select the following one).

The Run button on the toolbar:

Our cell after pressing Run:

As you can see, you have the result of the code that was run in that cell, as well as information on when this particular cell has been run.

- When you save a notebook, the code as well as the results are saved! So you can always reopen it to look at the results without having to run all the program again, while still having access to the code.

Notebooks are so named because they are just like a physical Notebook: it's exactly like if you were taking notes about your experiments (which you will do), along with the code itself, including any parameters you set. You see the output of the experiment in line (this is the result from a cell once it's run), along with all the notes you want to take (to do that, switch the cell type from the menu from `Code` to `Markup`).

Time to play

Now that we have covered the basics, just give it a try!

- In your Jupyter environment (the file explorer-like interface), there is a file called `01-sandbox.ipynb`. Double-click on it to launch the notebook (it will open another tab in the content section of the environment). Please feel free to experiment, run the cells, add some more and create functions. You can do what you want - it's your environment, and there is no risk of breaking anything or impacting other users. This environment isolation is also a great advantage brought by Red Hat OpenShift Data Science.
- You can also create a new notebook by selecting `File`->`New`->`Notebook` from the menu on the top left, then select a Python 3 kernel. This instructs Jupyter that we want to create a new notebook where the code cells will be run using a Python 3 kernel. We could have different kernels, with different languages or versions that we can run into notebooks, but that's a story for another time…

- You can also create a notebook by simply clicking on the icon in the launcher:

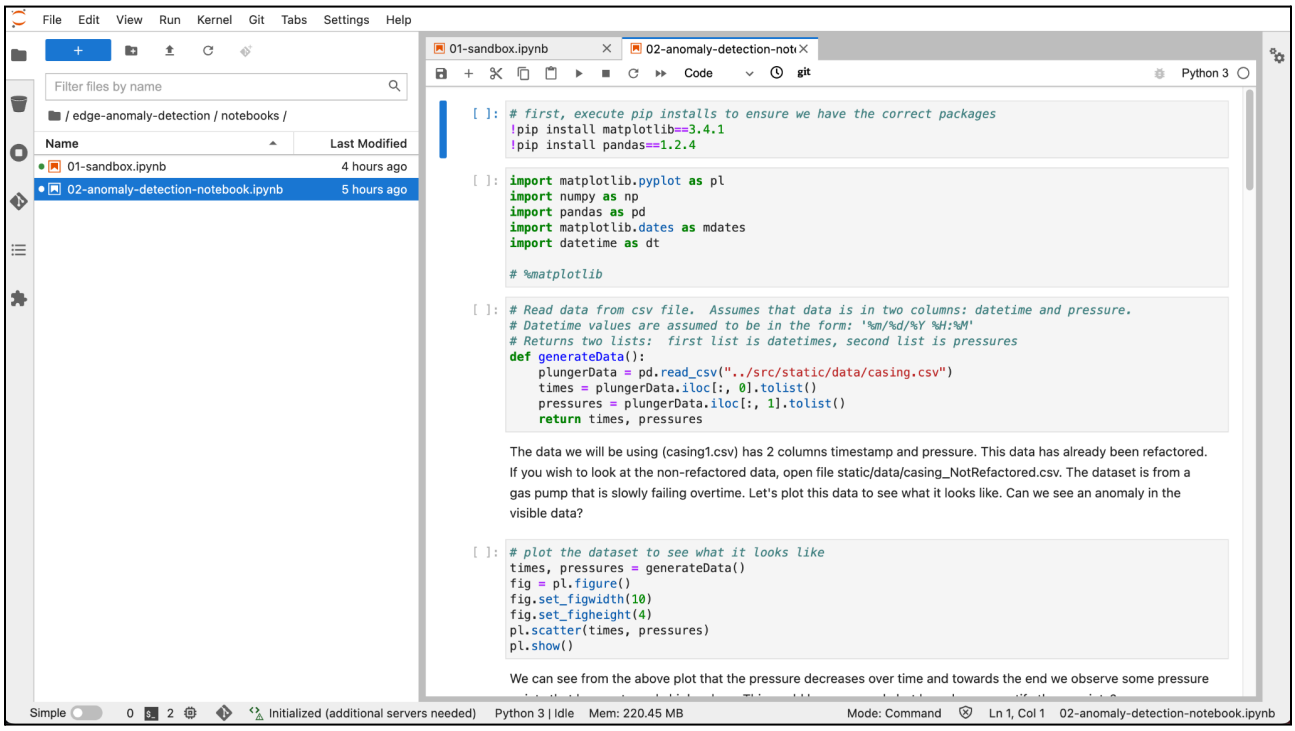
- If you want to learn more about notebooks, head to this page. Now that you're more familiar with notebooks, you're ready to go to the next section - Anomaly Detection.

4 Anomaly Detection 2:30-3:00pm

4a Examine time series data for anomalies

If you have not already done so, change into the notebooks sub-folder. You will see the following 2 files: `01-sandbox.ipynb`, `02-anomaly-detection-notebook.ipynb`.

Double click `02-anomaly-detection-notebook.ipynb` to open the notebook.

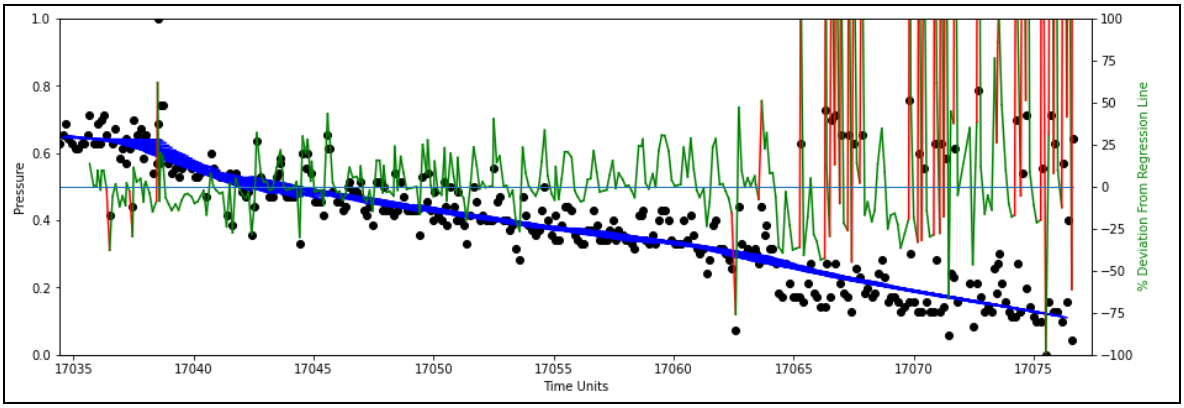


In this notebook, we'll explore a very small data set (2 columns consisting of timestamp and pressure) so that you can see what 'sensor' data looks like. The dataset is in csv file format.

You will work through this notebook, reading the explanations and executing the notebook's cells. You will examine the data and understand how we determine the data

shows an anomaly by using Linear Regression. Do this now. When you are finished we will discuss the data and what we expect when looking for anomalies.

The last notebook cell you execute will produce the following Anomaly Detection plot.



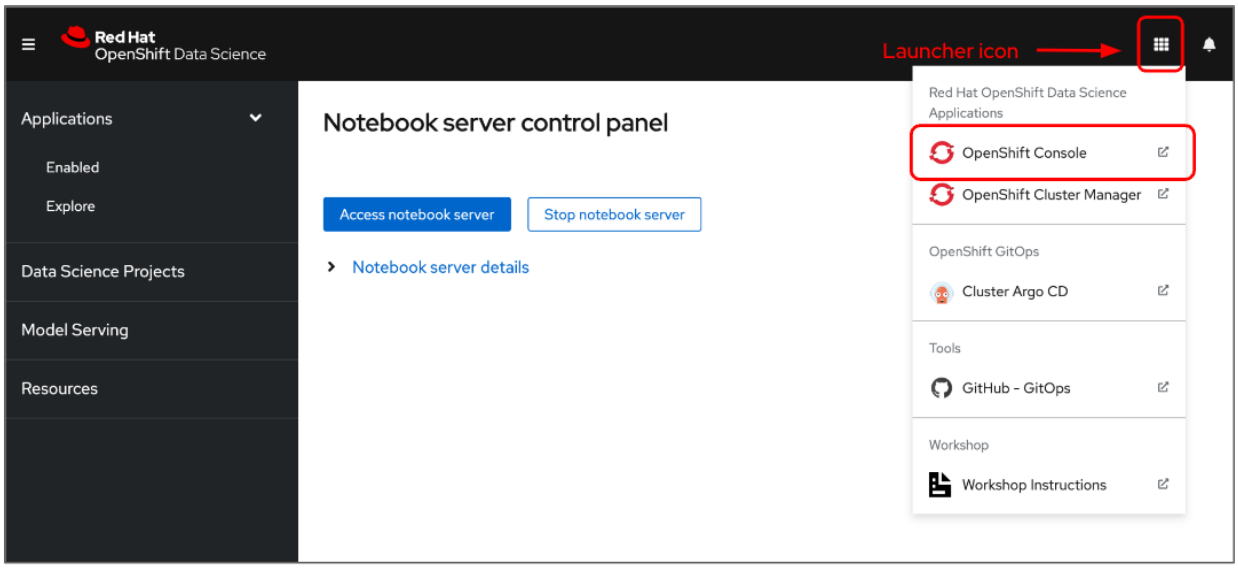
In this plot, the far right hand side of the plot is not only an anomaly, it is also the failure of a mechanical pump. Hypothesis: If we can detect anomalies we can then use this knowledge to predict when a mechanical pump will fail.

Let's take the web application developed for Anomaly Detection and deploy it using containerization. Go to the next section: Deploy the Anomaly Detection model & web application as a container

4b Deploy the Anomaly Detection model & web application as a container

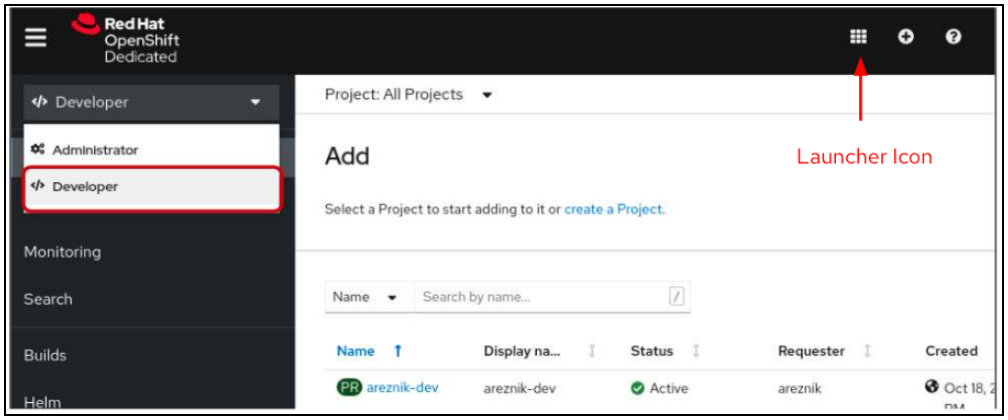
We need to containerize and deploy the Anomaly Detection Web Application code. We will package the code as a container image and run it directly in OpenShift as a Web Application.

From the Red Hat OpenShift Data Science platform choose the **Launcher**. From the Launcher drop down menu select **"OpenShift Console"**



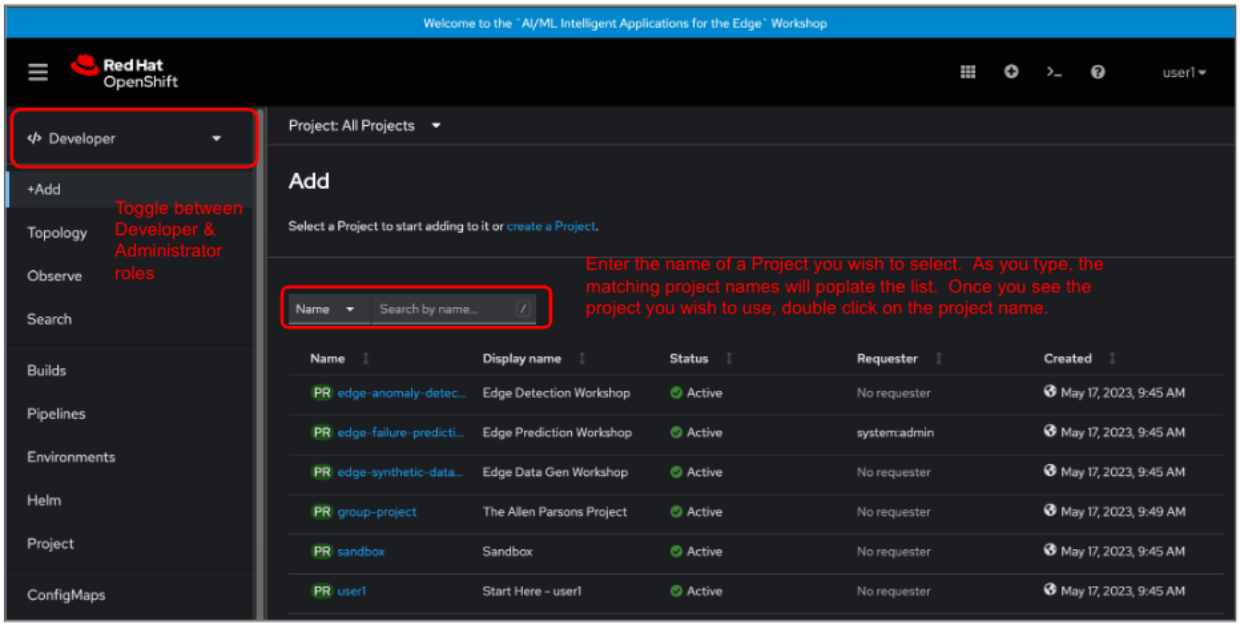
Note: The Launcher tool (icon) is also visible in the OpenShift console. **You can use this tool to move between** the Red Hat OpenShift Data Science Platform and the Red Hat OpenShift Dedicated platform.

Once you are in the OpenShift console, make certain you are in *'Developer'* mode.



Change to Developer mode. You will see a number of projects listed. Check if you can find the following listed projects as we will be using them in the lab.

1. edge-anomaly-detection
2. edge-failure-prediction
3. user<#> <- your user project. You will be working in your project user<#>



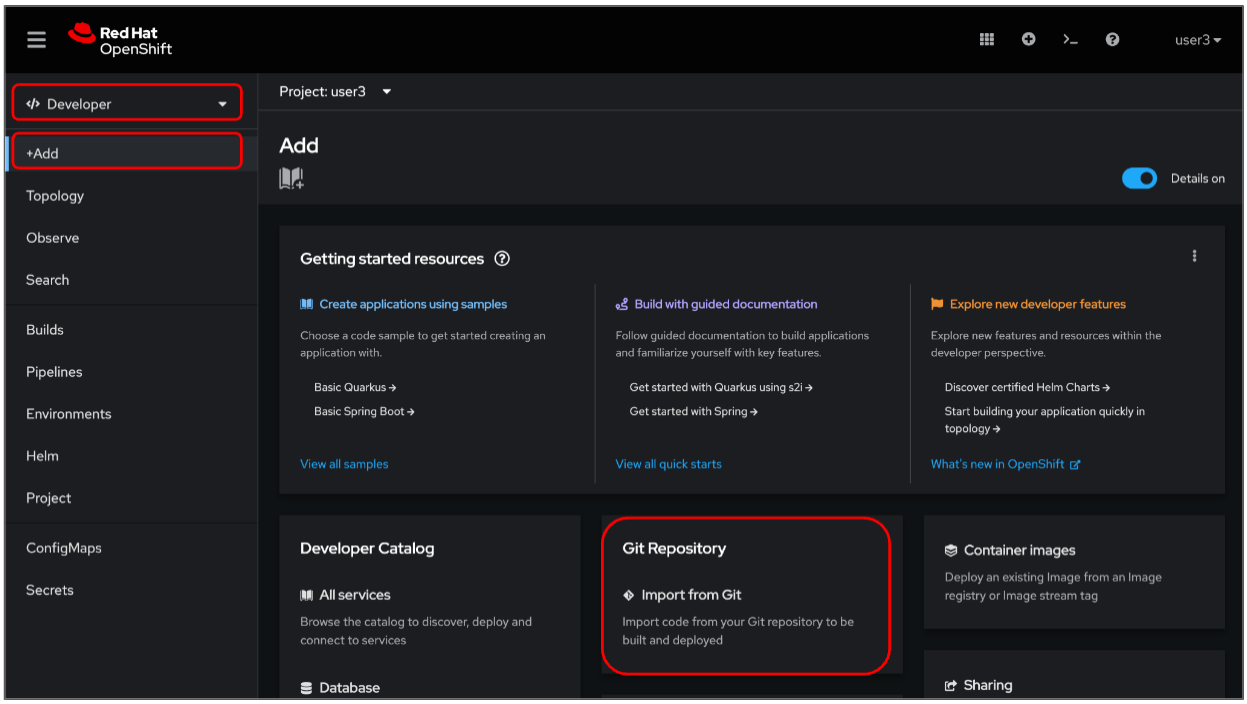
Hint: you can type in the name of a project and search for it in the '*Search by name…*' field!

During the workshop, if you do not have time to create containers in your own project, the projects **edge-anomaly-detection** & **edge-failure-prediction** will contain (pre-created) containers which you can view and run.

Make certain that you are in the 'Project' that is assigned to you (user<#>).

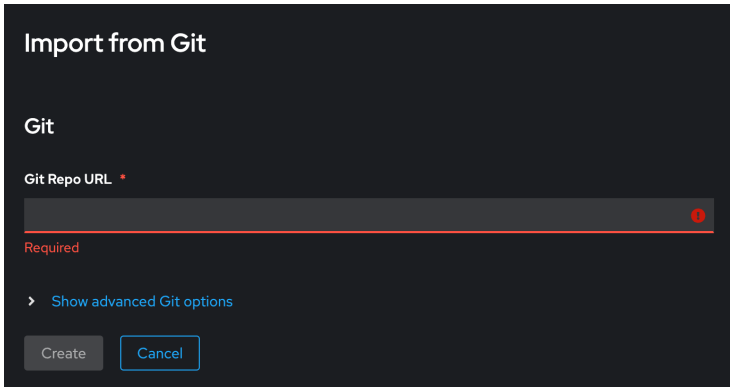
Project user<#>' is your work area (namespace) within OpenShift. This project name will be your userid that you were given at the beginning of the workshop. In the following screenshot, the project name is: user3. Click your project user<#>.

Once you have selected your project (e.g. user<#>), you can begin the process to create your container. Click the +Add menu. A number of options will appear.



We are going to create a containerized web application. We will be using the source code found in the **edge-anomaly-detection github repository** to create the containerized web application. To do this we use the **Git Repository**, *'Import from Git'* option.

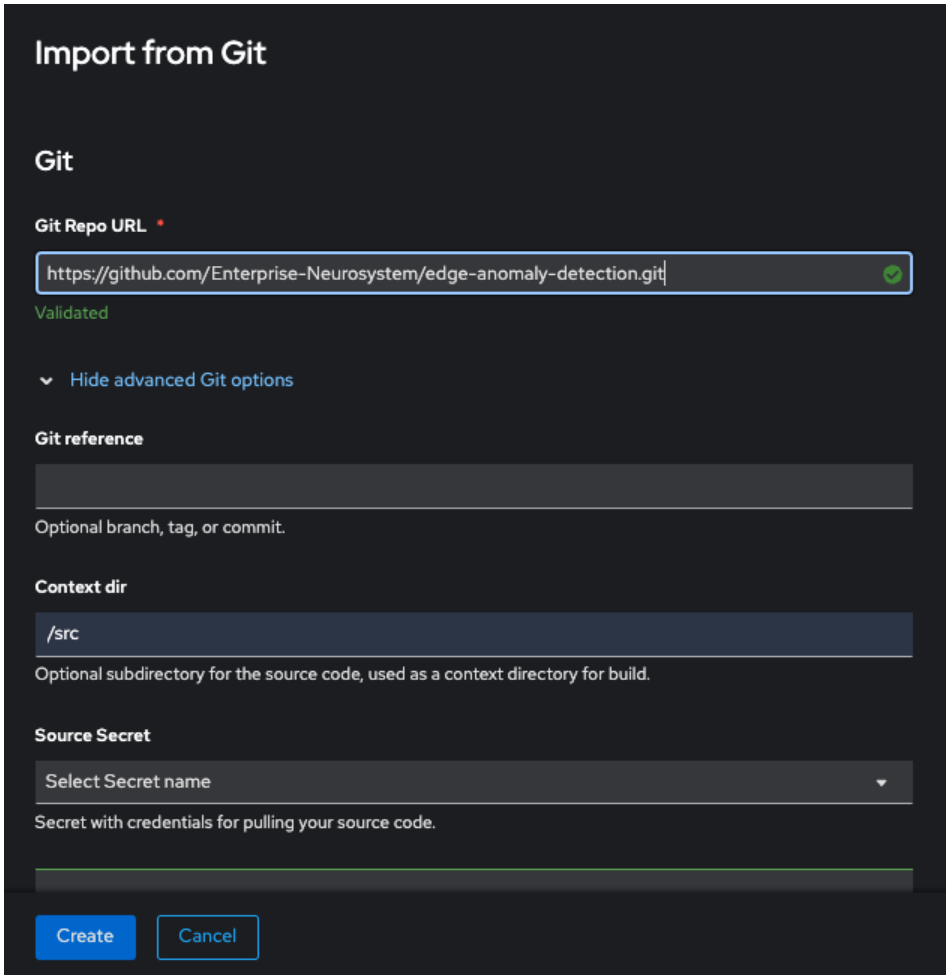
Click the 'Git Repository' option. The **Import from Git** page will appear.



In the Git Repo URL Field, enter:

`https://github.com/Enterprise-Neurosystem/edge-anomaly-detection.git`

The Git Repo URL will be validated. Once validated, further options are available.

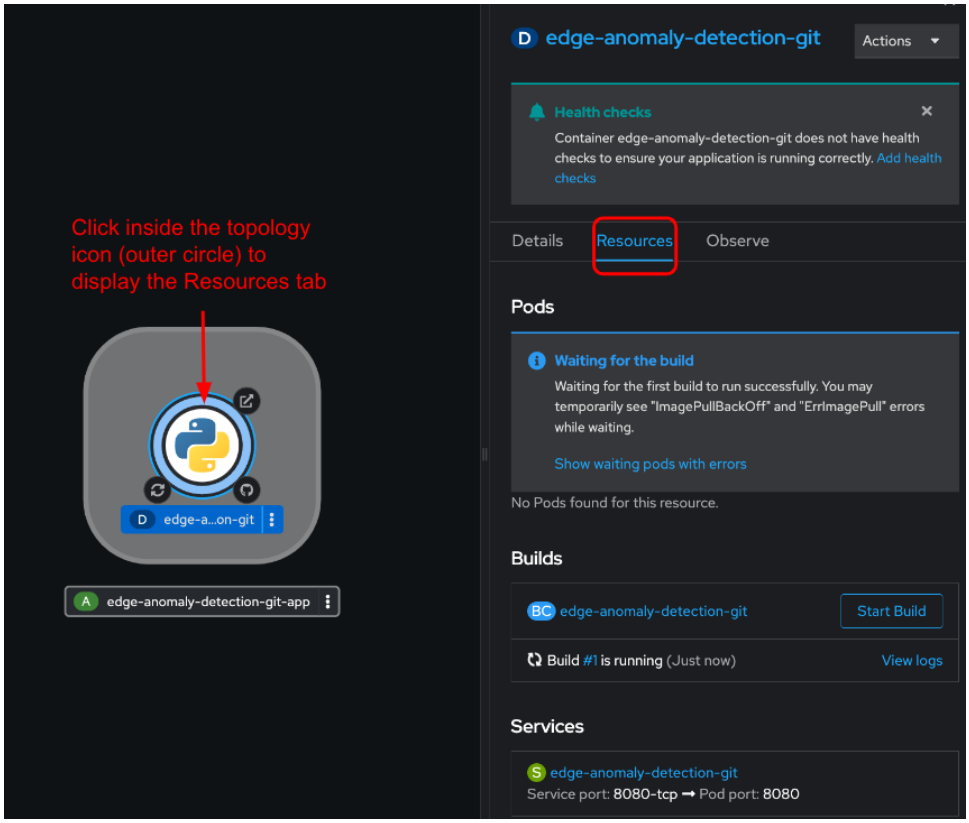


Next you will need to:

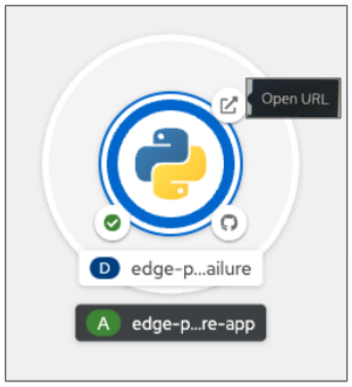
1. Open “Show advanced Git Options”
2. **Set “Context dir” to “src”**

If you continue to scroll down the page, you will see that everything is automatically selected to create a deployment of your application, as well as a route through which you will be able to access the application.
Make certain to name your app. For example, *app-anomaly-detection*. Now we are ready to press the ‘Create’ button to create our containerized application. Press the ‘Create’ button.

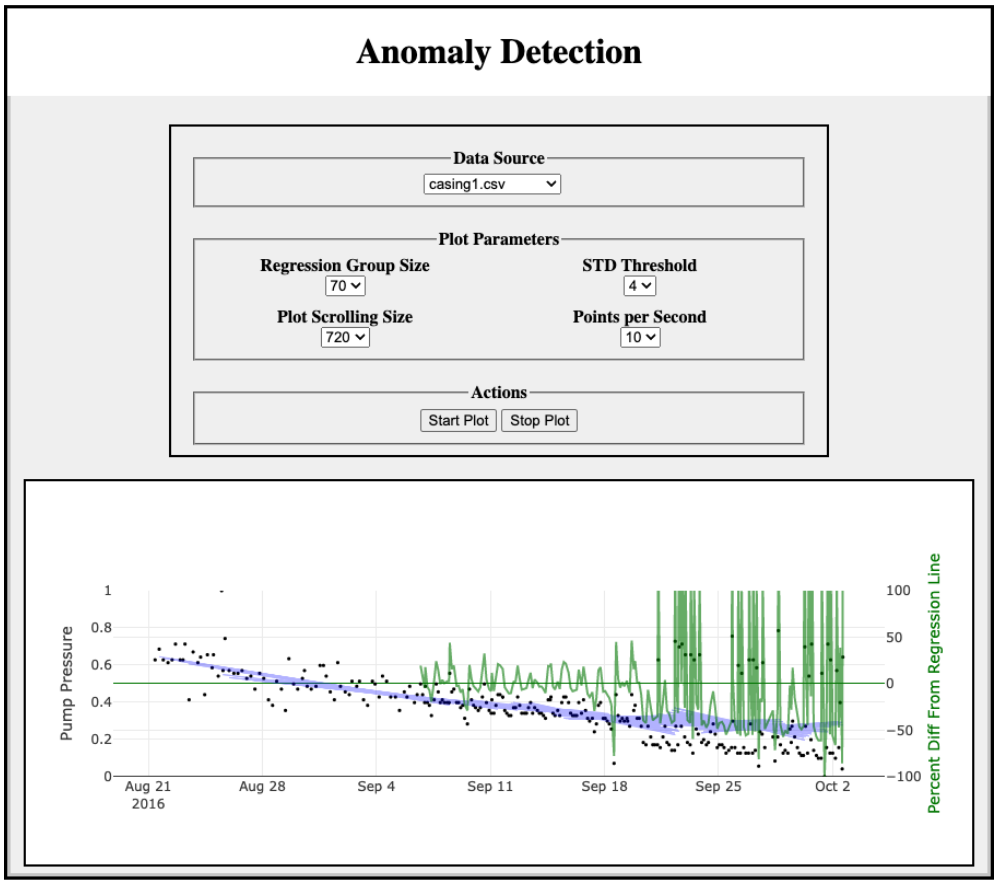
The automated process will take a few minutes. Some alerts may appear if OpenShift tries to deploy the application while the build is still running, but that’s OK. Eventually, OpenShift will deploy the application (rollout), and in the topology view, you should see a screen similar to the following screen capture.



Now we can head back to our ‘Developer’ view and display our containerized application in a browser by clicking on the URL icon in the Topology view.



Your containerized Anomaly Detection application will appear in your browser window. In this example we have gone ahead and executed the application.



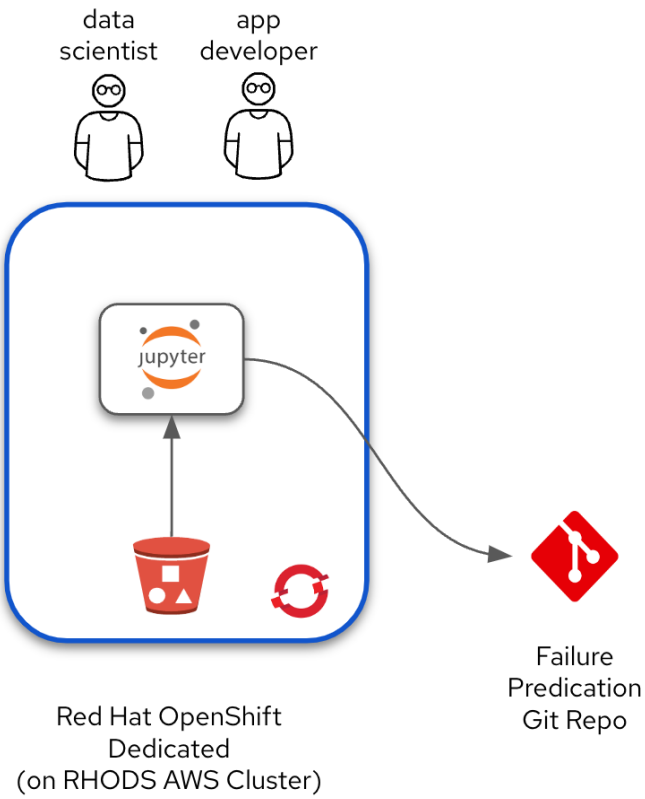
Try re-running the application with different plot parameters to see what happens.

Now that you have an idea of what sensor anomalies look like and how to detect them, let's take a look at how we can predict failures. Continue to the next section, Failure Prediction Web Application.

5. Failure Prediction Web Application 3:00-4:00pm

5a Business Background & Architecture

Let's discuss the architecture that makes up the Failure Prediction Web application.



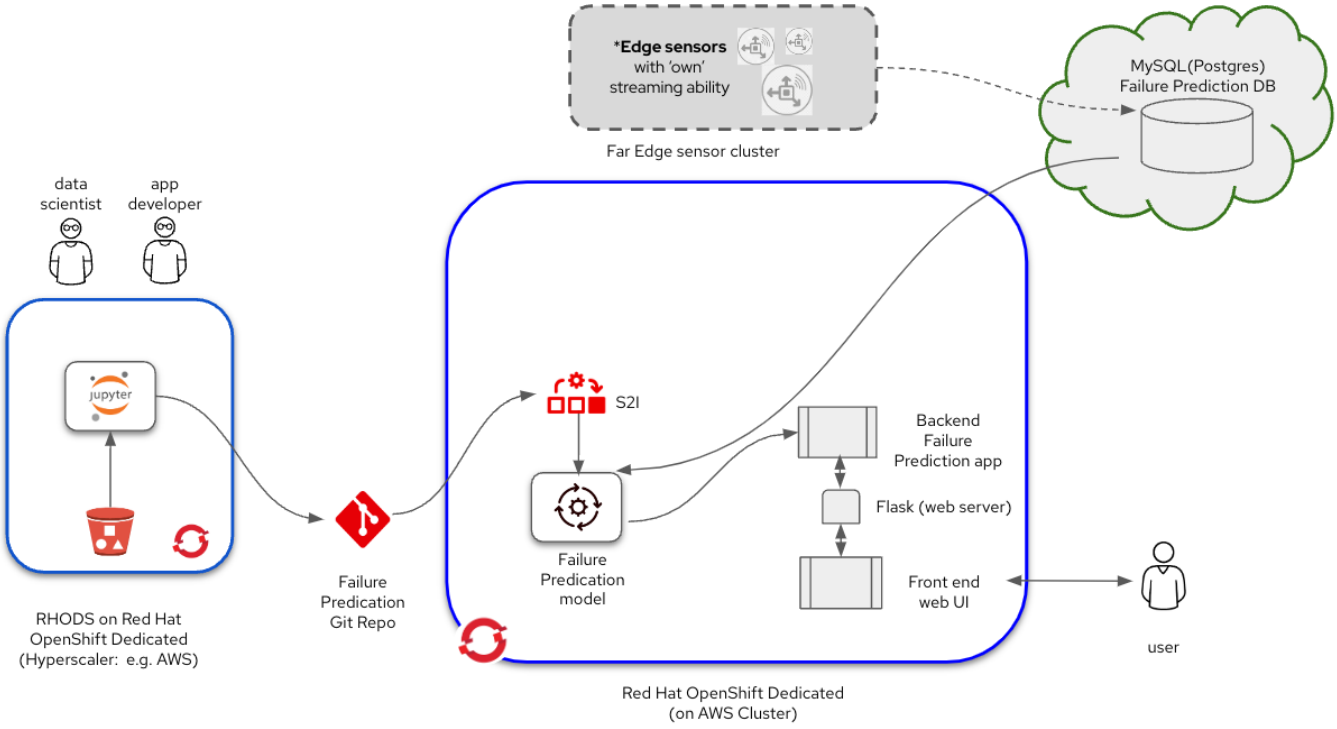
Here we have the DS and App Developer working and saving their work (python files) to a central github repository.

This is important to understand. Because we will then use the OpenShift platform that will take that repository and models, web UI and underlying business logic (services) and containerize them.

Once containerized we can then deploy the containerized image to be access by a URL (entry to our Failure Prediction model analysis)

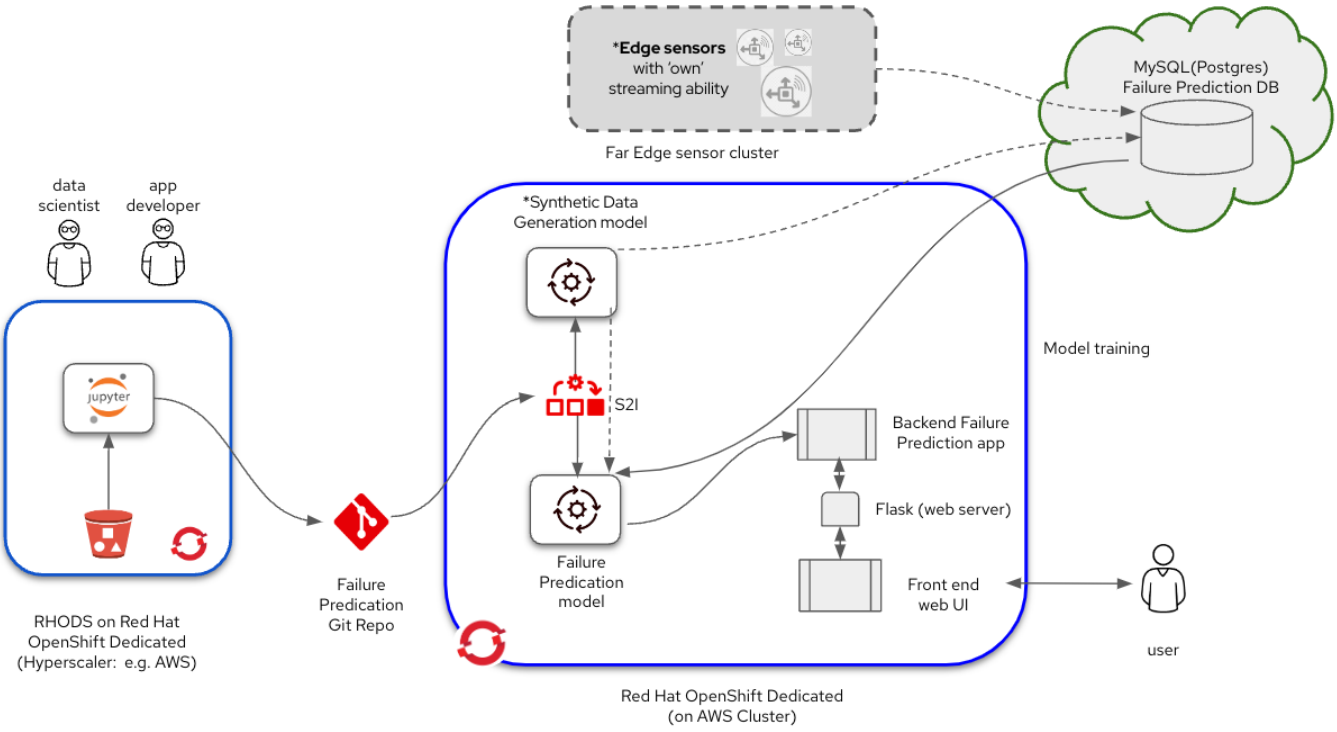
We envisioned the following scenario where data was generated by sensors, and placed into a database located outside of our network.

The fact is that some companies will save most of the time series data in order to train future models on that data..



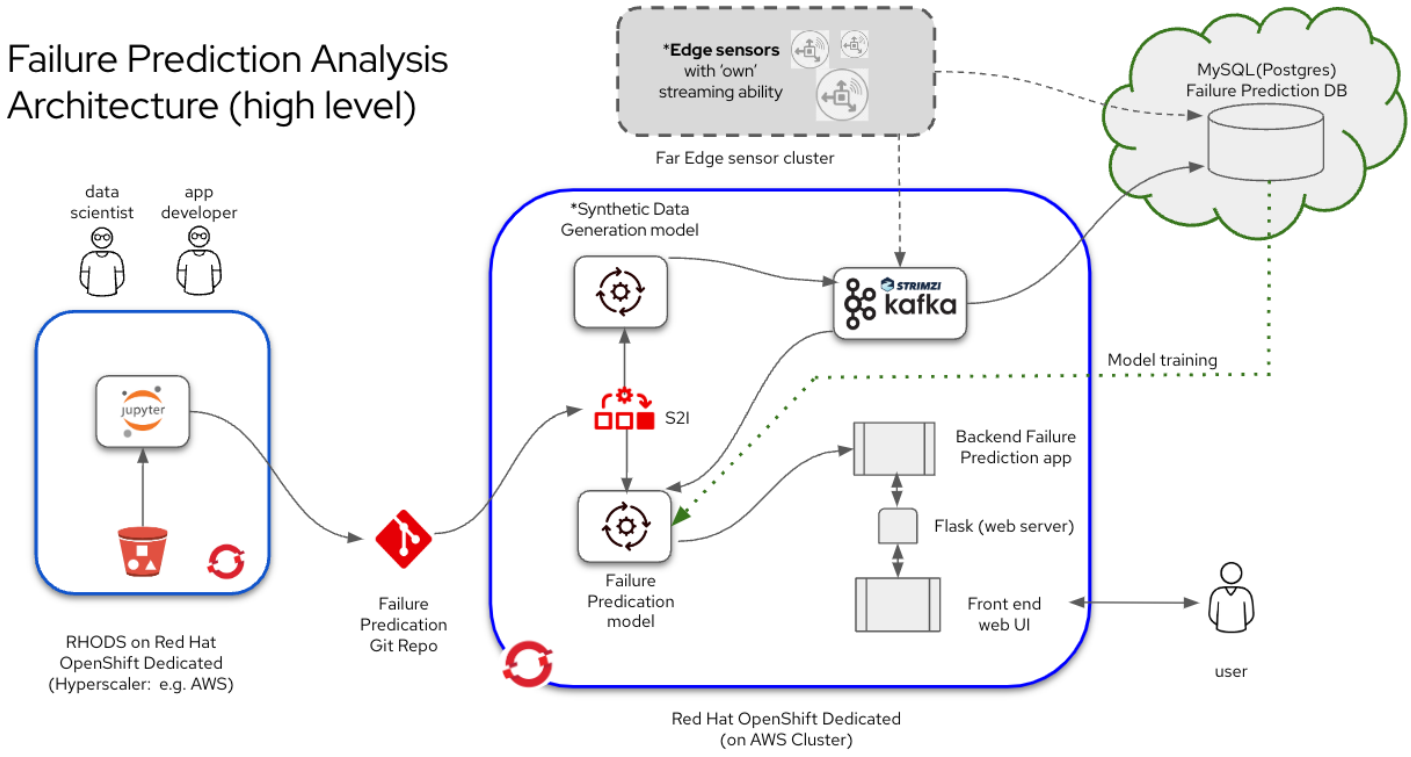
Of course we didn't have live sensors so we had to add a synthetic data generator which we will see in the next diagram

The addition of the Synthetic data generator was containerized and deployed via S2i. This allowed us to create data for consumption by our model and also store data in an external database residing in S3 storage in AWS



This looks good but we then needed a service to stream the data to the database and the model. That's where we introduced the Apache Kafka streaming service.

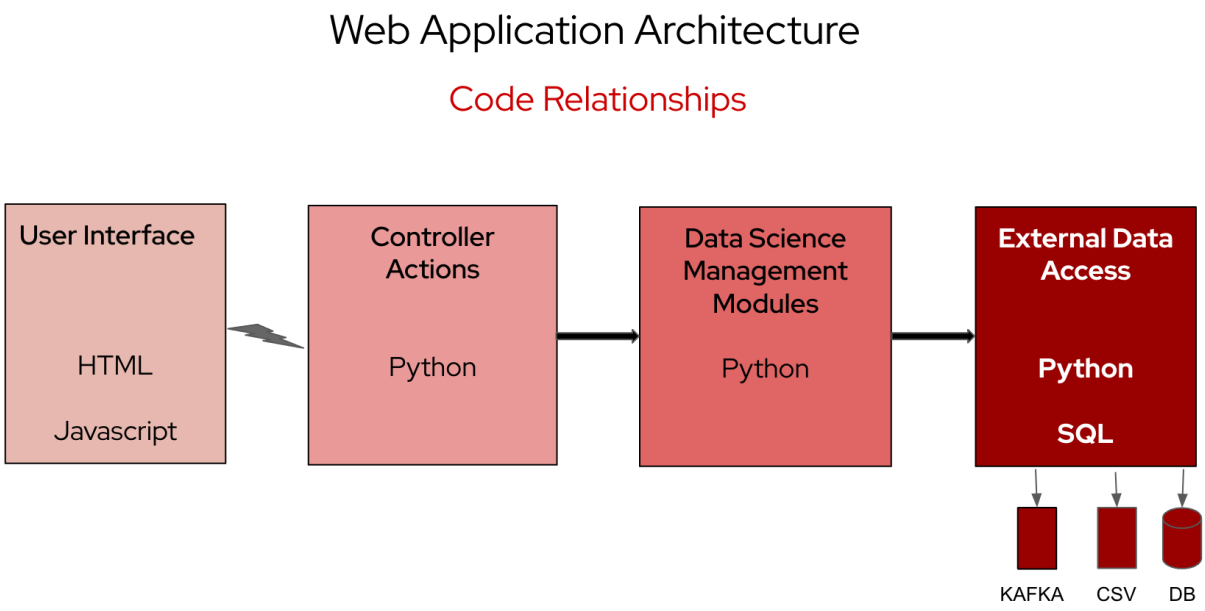
With the Apache Kafka streaming service we can take the synthetic data we generate, save it to the database and also stream the data to our failure prediction model for analysis.



> **Note**: in the future we can use the stored data to re-train our failure prediction model.

Analysis from the failure prediction model is then accessed through the web ui where it is displayed in plot format to our users.

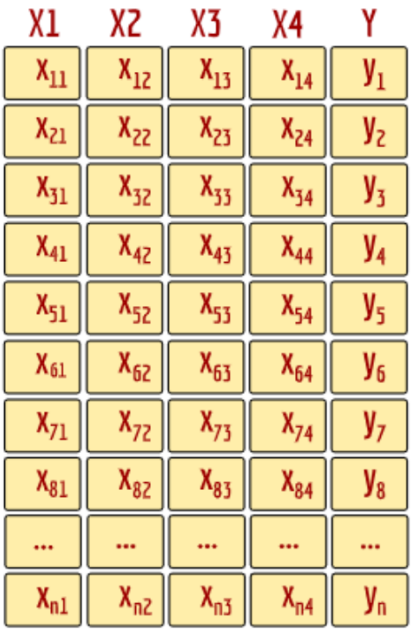
5b Web Application Architecture



<class discussion>, after which we will discuss re-shaping time series data to work with an LSTM model.

5c Reshaping Time Series Data

Original Time Series Data



Name: Alexander Laskorunsky
URL: https://medium.com/mlearning-ai/lstm-with-keras-data-reshaping-in-many-to-one-architecture-c7d3669e3a5c

This is what data would look like in a supervised model where you are trying to make a prediction from the rows of data.

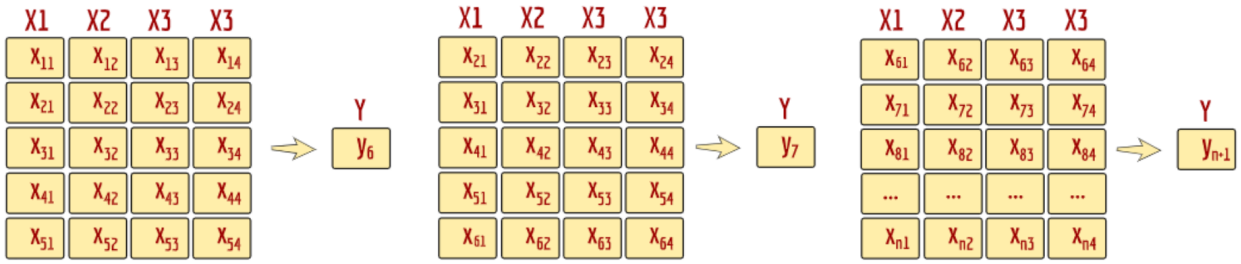
- Diagram shows 4 feature-xs (x14) and one target value (y)

Each column represents some value (e.g. diabetes, body mass index, gender, glucose level etc) The target is diabetes Y or N

The model can be a simple neural network which we can readily train with this data in the present form.

This is the typical data you come across, and if you want to make a prediction you supply 4 features to the model once it has been trained. Prediction is then simple.

Time Series to LSTM reshape

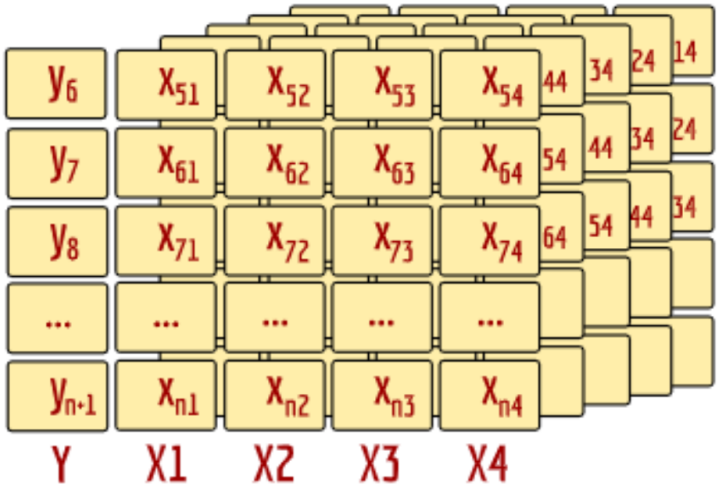


Name: Alexander Laskorunsky
URL: https://medium.com/mlearning-ai/lstm-with-keras-data-reshaping-in-many-to-one-architecture-c7d3669e3a5c

We have 4 features and a target output value. Each row represents a data point in a time series. We don't show the actual time values because we don't care what they are when we train the model. We assume that all of the time deltas between each data point are the same.

In the above example, we are using 5 data points to make a prediction for the 6th data point. Therefore we stack the above slices to make a 3D array.

Final 3D array for LSTM



Name: Alexander Laskorunsky
URL: https://medium.com/mlearning-ai/lstm-with-keras-data-reshaping-in-many-to-one-architecture-c7d3669e3a5c

Think of each of the groups of 5 features that we were looking at as a 2D array. In this chart each group is stacked one behind the other.

We use these 2D numpy arrays to train our model. We feed the model one array at a time. (E.g) feed one card at a time to the model

That's how we take a time series and reshape it so that it can be used in an LSTM model. For prediction, we create a buffer (programmatically) that holds 5 data points at a time. And we can't start predicting until the buffer has 5 data points in it. The buffer behaves as a queue, and we pop the bottom one off so that we can maintain a buffer size of 5 (in this case).

Now that we have seen the architectural overview, let's start building the failure prediction app. As usual, we start with the data. That is, we will examine and curate the data so that it can be readily used by our model.

We can see that the original shape of the time series was (n_rows, 5) where 5 denotes the number of features.

The final shape is: (n_rows - prediction_size - 1, prediction_size, num_features)

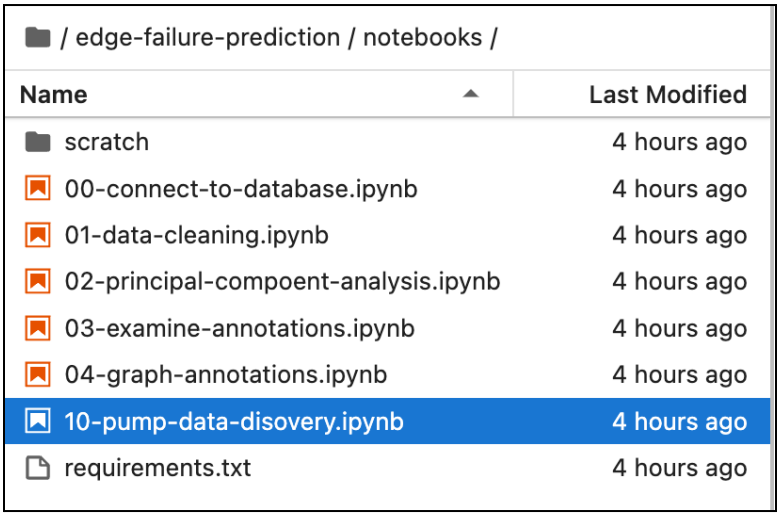
- E.g. (n_rows - 5 - 1, 5, 5)

5d Data Preparation

We want to do the following steps for data preparation/curation:

1. Obtain sensor Data
2. Plot the data
3. Convert the Machine Status (text) to numeric values. (This column will become the model label)
4. Examine Nulls & determine which columns to drop
5. Discover the Failure times
6. Split data by Failure times into Training, Test, Validation
7. Find the mean of the Training data
8. Replace the NaNs & blanks with the mean of the Training data
9. Create a MinMaxScaler for Training data
10. Transform all data using MinMaxScaler that was fit to Training data
11. Fit & Transform the Training data using PCA
12. Transform the Test and Validation data using the fitted PCA
13. Create a ranked list of the Principle Components. (This ranked list will now become the model features)

We placed all the data prep into one notebook - 10-pump-data-discovery.ipynb. Let's take a look at this notebook.

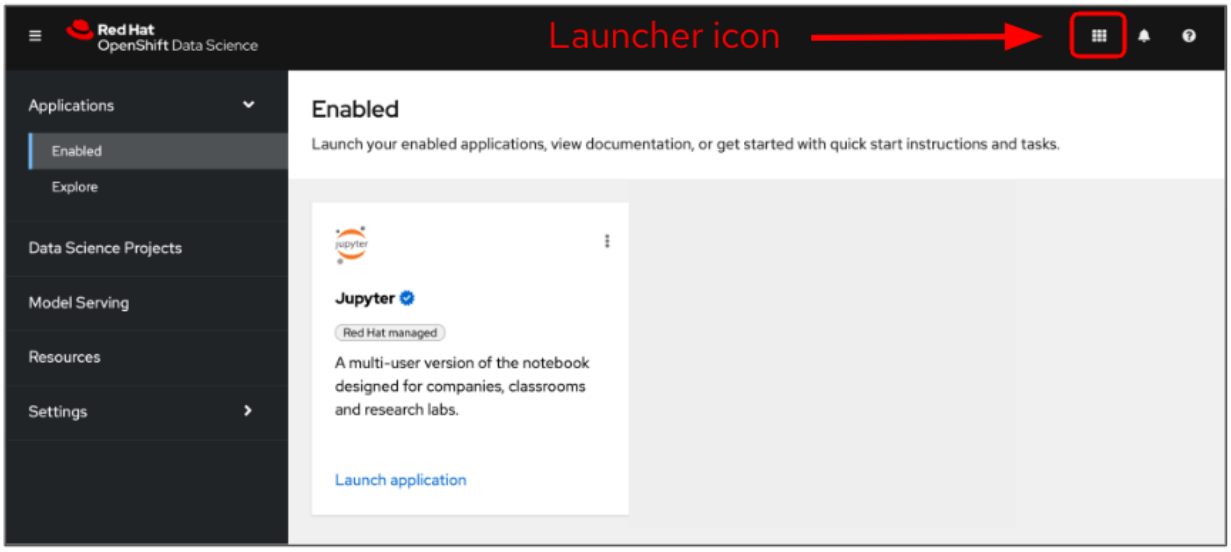


Now that we have seen how to curate the data, we will turn our attention to the Failure Prediction Web application which will incorporate what we learned.

5e Containerize & Run the Failure Prediction Web Application

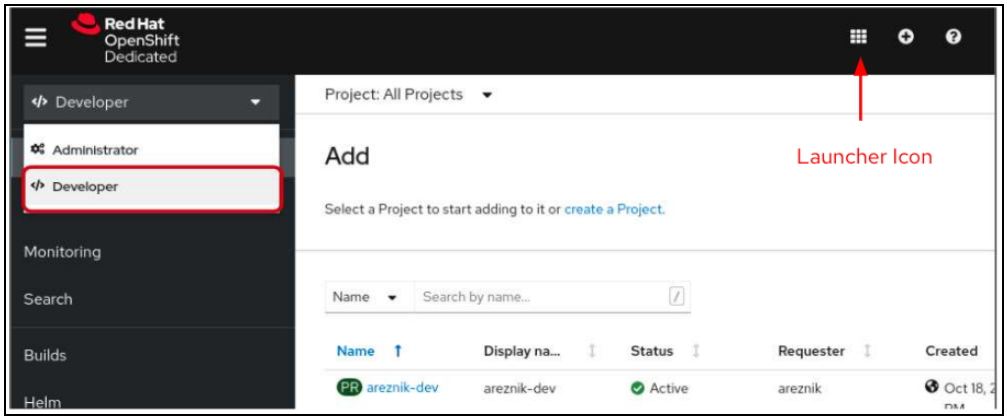
Let's start by containerizing and deploying the Failure Prediction Web Application code. We will package the code as a container image and run it directly in OpenShift as a Web Application.

From the Red Hat OpenShift Data Science platform choose the Launcher Tool and **Select "OpenShift Console"**

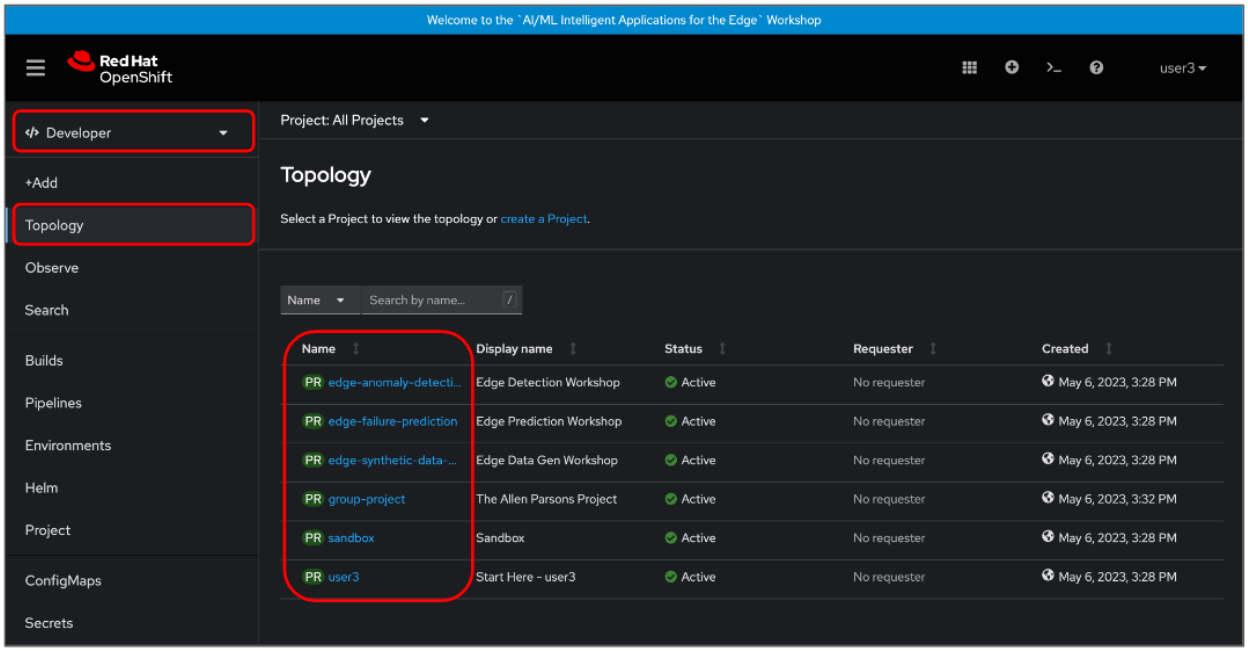


Within the OpenShift console, make certain you are in 'Developer' mode.

Note: The Launcher Icon is also visible in the OpenShift console. You can use this icon to move between the Red Hat OpenShift Data Science Platform and the Red Hat OpenShift Dedicated platform.



While you are in Developer mode, click on the 'Topology' menu item. You will see 6 projects:

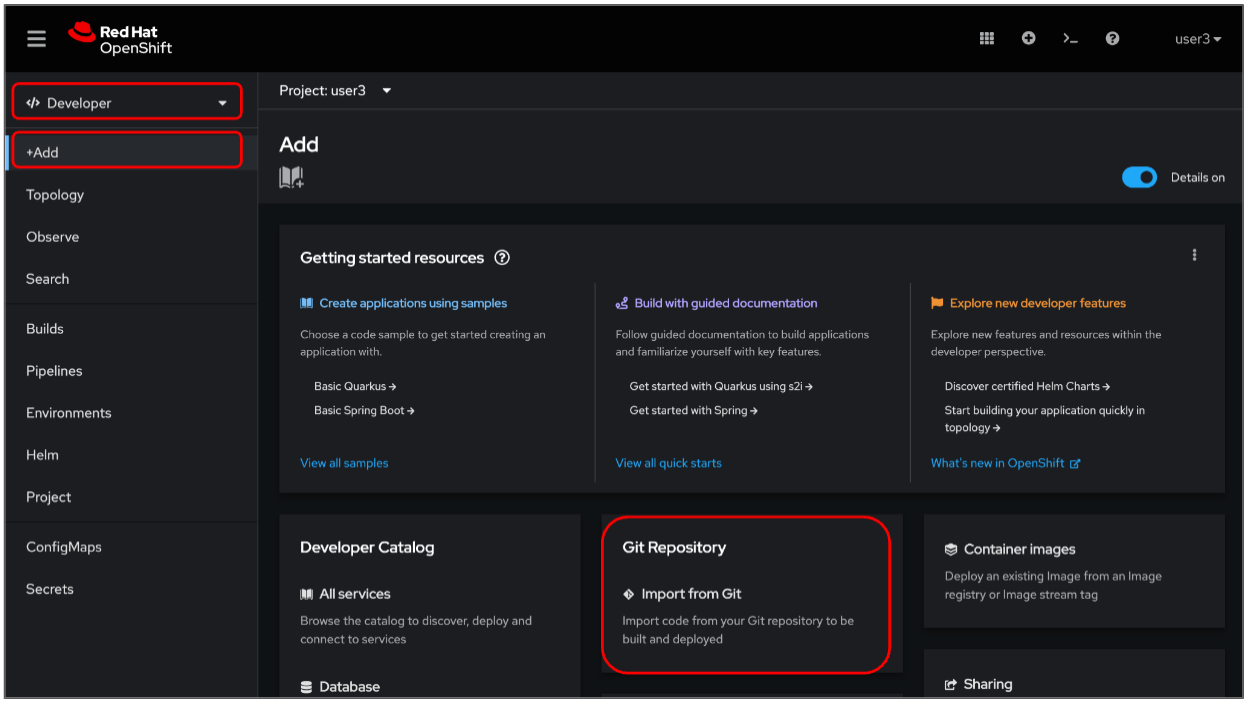


1. Edge-anomaly-detection
2. Edge-failure-prediction
3. Edge-synthetic-data-generator
4. Group-project
5. Sandbox
6. user<#>

You will begin by working in your project user<#>.

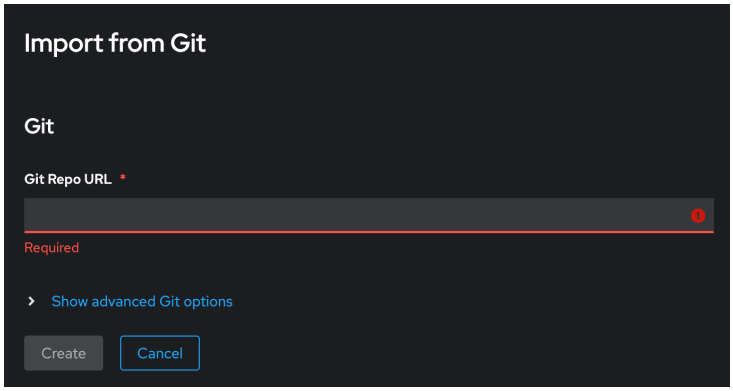
Make certain that you are in the ‘Project’ that is assigned to you (user<#>). Project user<#>’ is your work area within OpenShift. This project name will be your userid that you were given at the beginning of the workshop. In the following screenshot, the project name is: user3. Click your project user<#>. You will see the following screen where ‘No resources found’. This is ok as we are going to create your first containerized resource.

Once your project (e.g. user<#>) is opened, you can begin the process to create your container. Click the +Add menu. A number of options will appear.



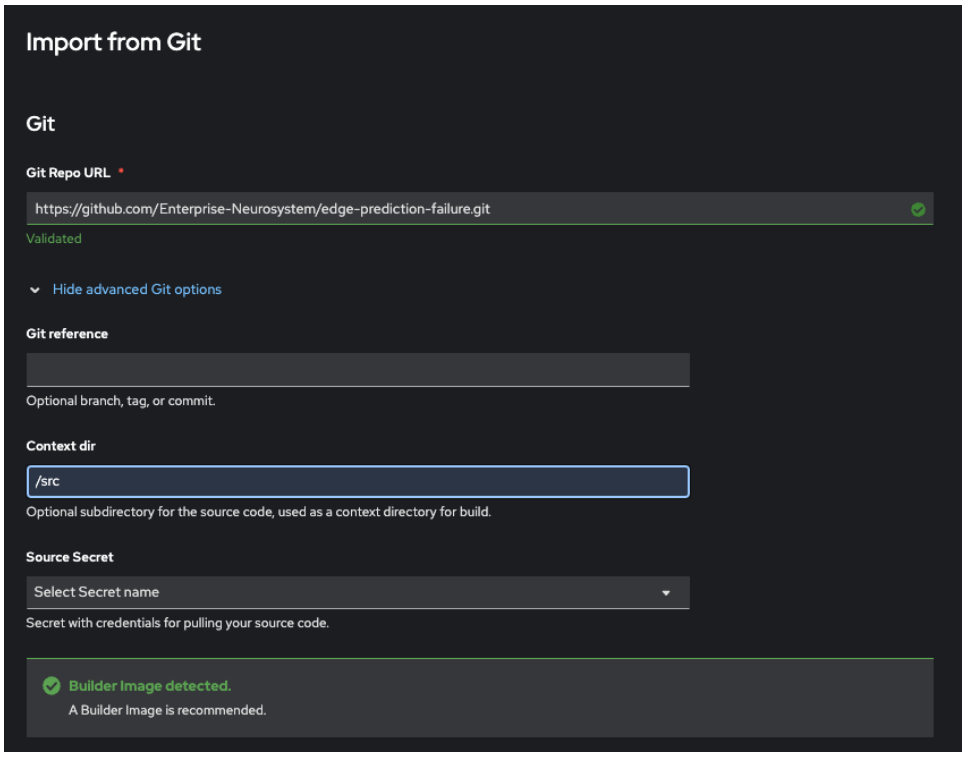
We are going to create a containerized web application. We will be using the source code found in the edge-failure-prediction repository to create the web application. To do this we use the Git Repository, Import from Git option.

Click the ‘Git Repository option. An **Import from Git** page will appear.



In the Git Repo URL Field, enter:
https://github.com/Enterprise-Neurosystem/edge-failure-prediction.git

Note that the Git Repo URL will be validated. Once validated, further options are available.

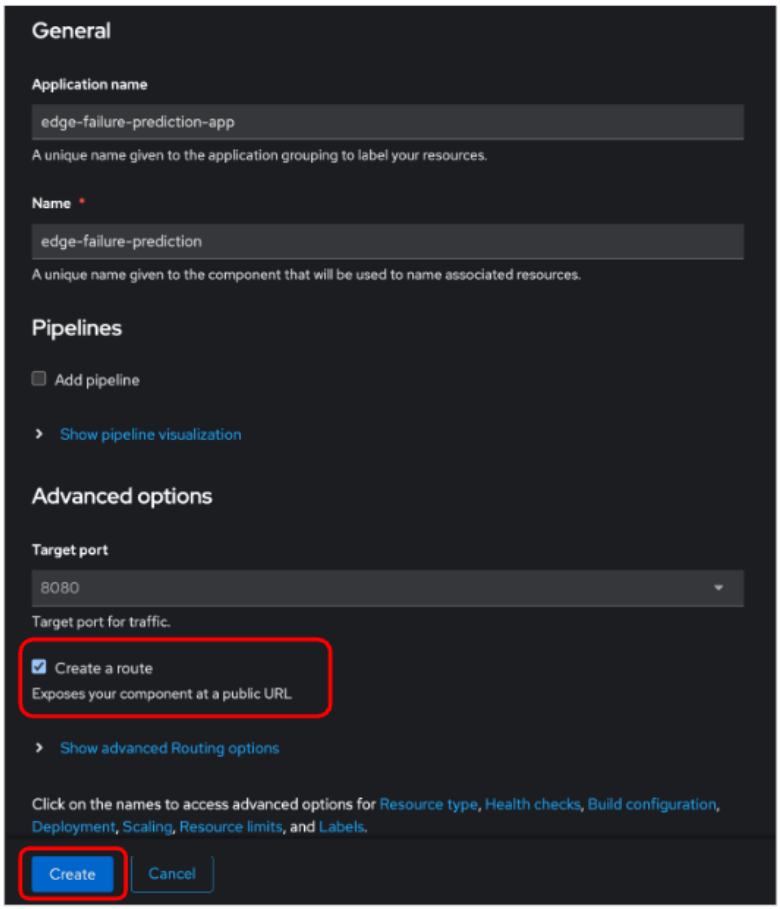


Next you will need to:

3. Open “Show advanced Git Options”
4. Set “Context dir” to “src”

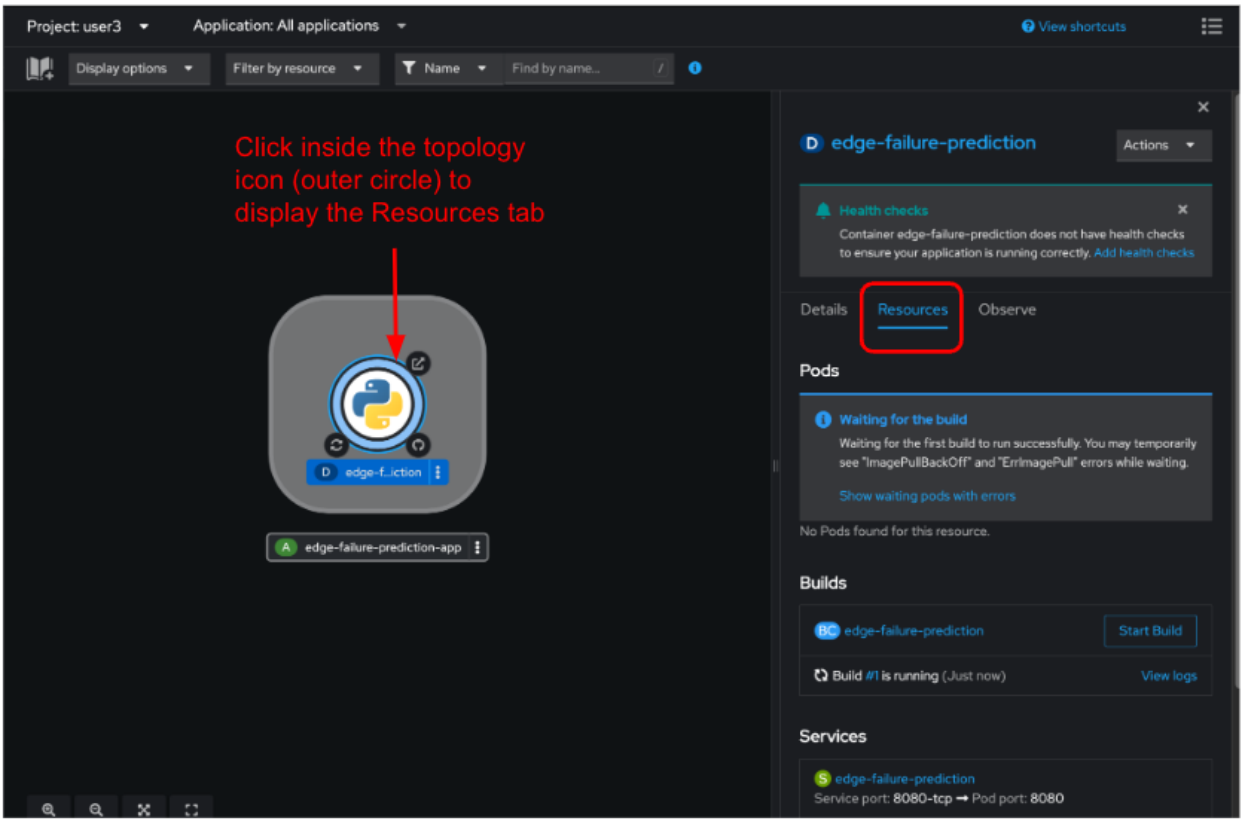
If you continue to scroll down the page, you will see that everything is automatically selected to create a deployment of your application, as well as a route through which you will be able to access the application.

Make certain to name your app. For example, app-failure-prediction. Now we are ready to press the ‘Create’ button to create our containerized application. Press the ‘Create’ button.

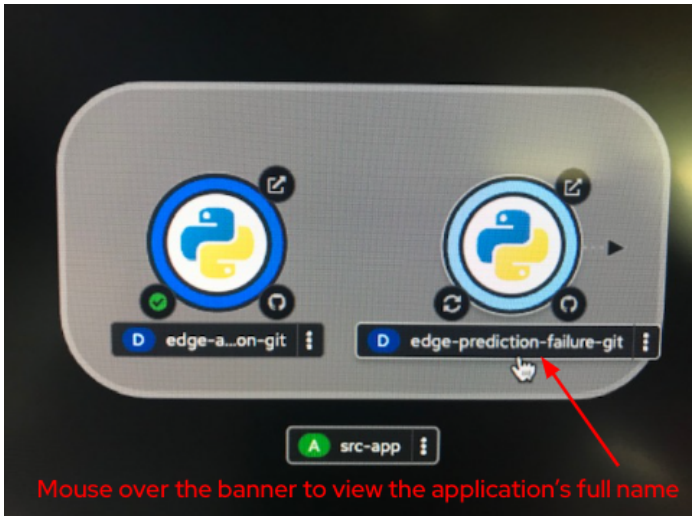


The automated process will take a few minutes. Some alerts may appear if OpenShift tries to deploy the application while the build is still running, but that’s OK.

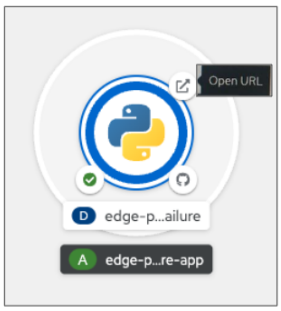
Eventually, OpenShift will deploy the application (rollout), and in the topology view, you should see a screen similar to the following screen capture.



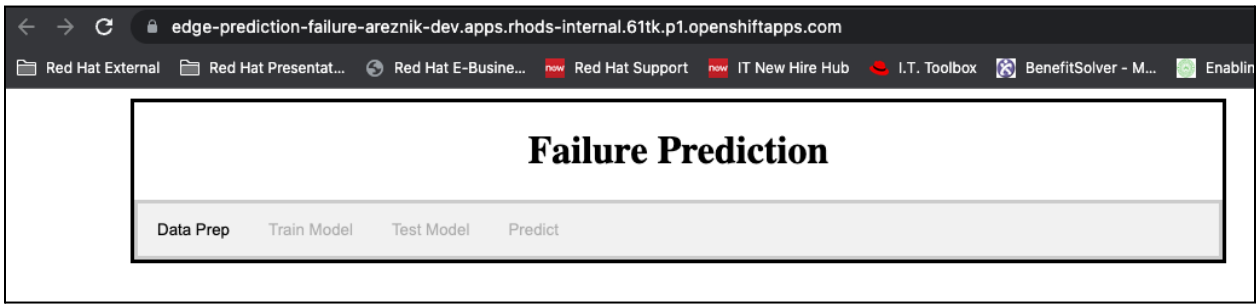
Note: if you gave the Failure Prediction application the same application name as the Anomaly Detection application (e.g. src-app) you will see both applications within the same container. See following screen capture.



Let's head back to our 'Developer' view and display our containerized application in a browser by clicking on the URL icon in the Topology view.



Your containerized Failure Prediction application will now appear in a browser window.



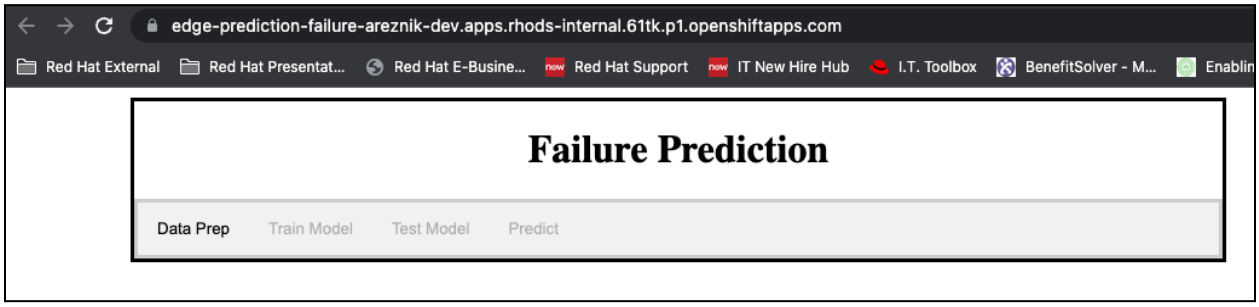
This application serves as a wizard that allows the user to:

- Prepare data for training, testing and prediction
- Train the model
- Test the model
- Make predictions on data that the user chooses

The goal of this application is to ultimately use a trained model that has learned what failures look like to specify there is a possibility of an imminent failure.

Opening Screen

When you first open the application, you will see **tabs**, some of which are initially disabled:



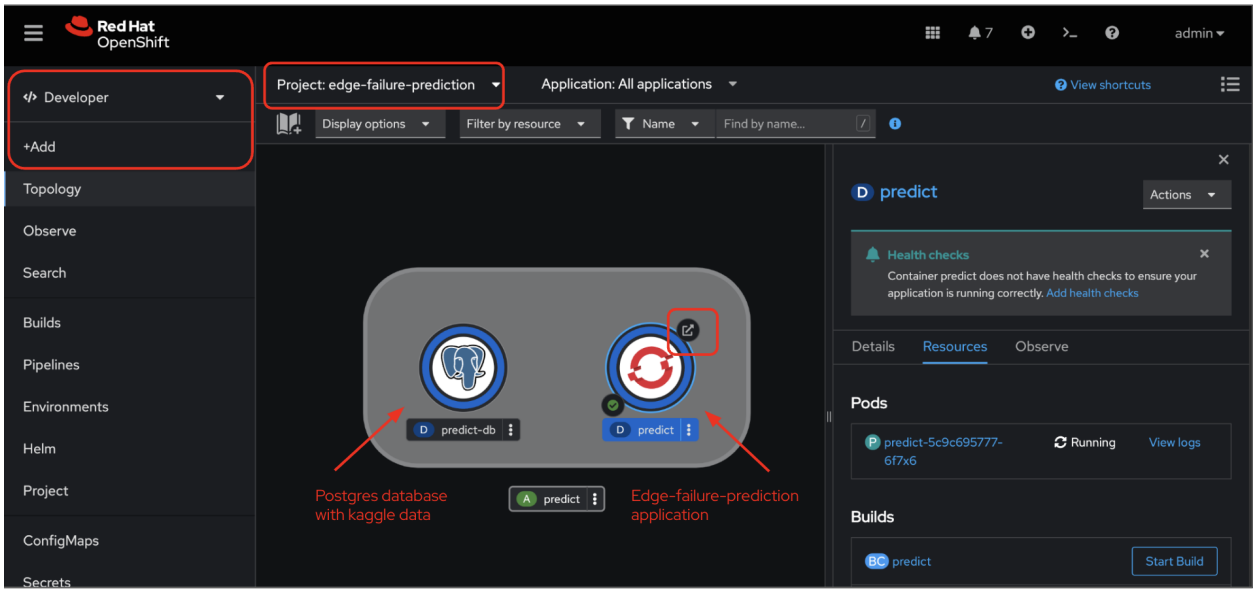
This user interface will guide you through all the processes from Data Preparation through Predictions and will make sure that you perform all processes in the correct order, just like a wizard.

The gray bar with the labels Data Prep, Train Model, Test Model, Predict serve as navigation tabs. When you click on a tab, the relevant screen will appear. Think of the tabs as menu selections that change the interface to suit the job at hand.

We don’t have time in this lab to set up the database and database connections which contain water pump data that we will be curating. Therefore since you now know how to create a

containerized application, you will **switch projects from <user># to edge-failure-prediction**.

Switch to the **edge-failure-prediction** project.



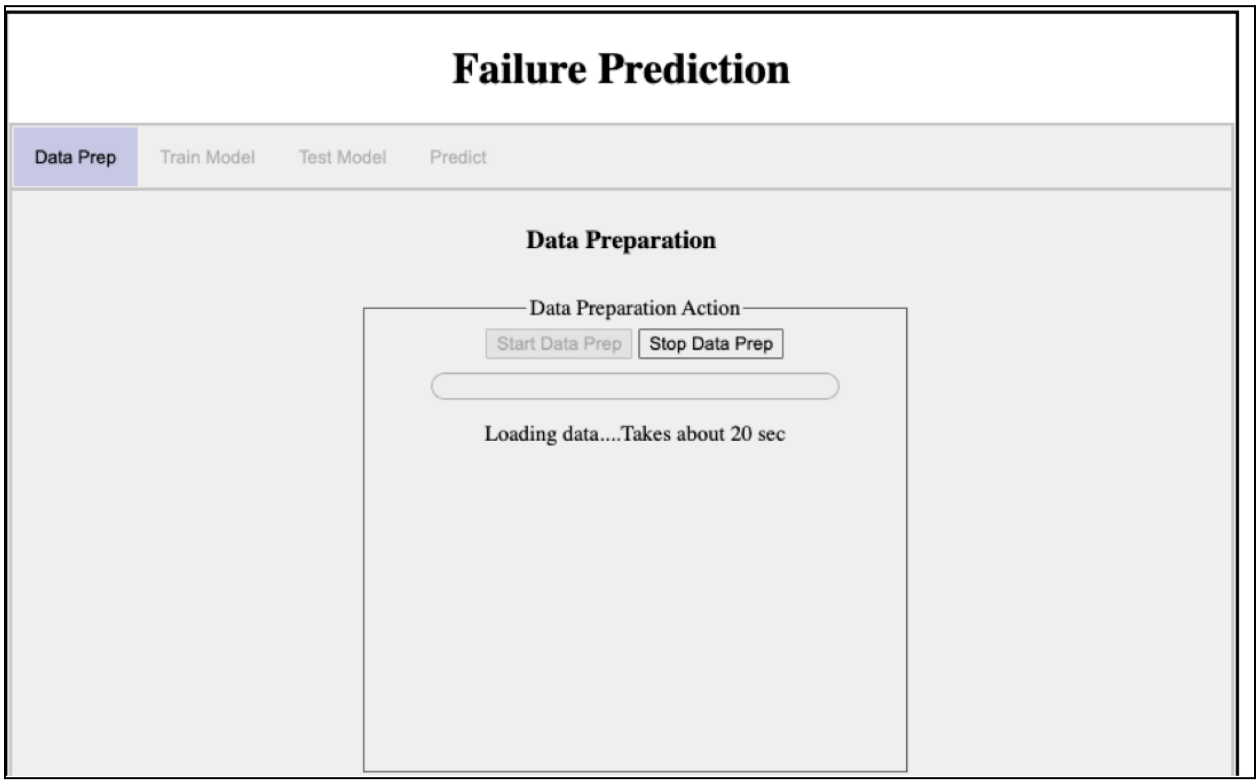
You will see the containerized postgres database and containerized edge-failure-prediction application. Go ahead and click the URL for the edge-failure-prediction application.

Data Preparation

When the application first loads in your browser, a large amount of data is loaded.

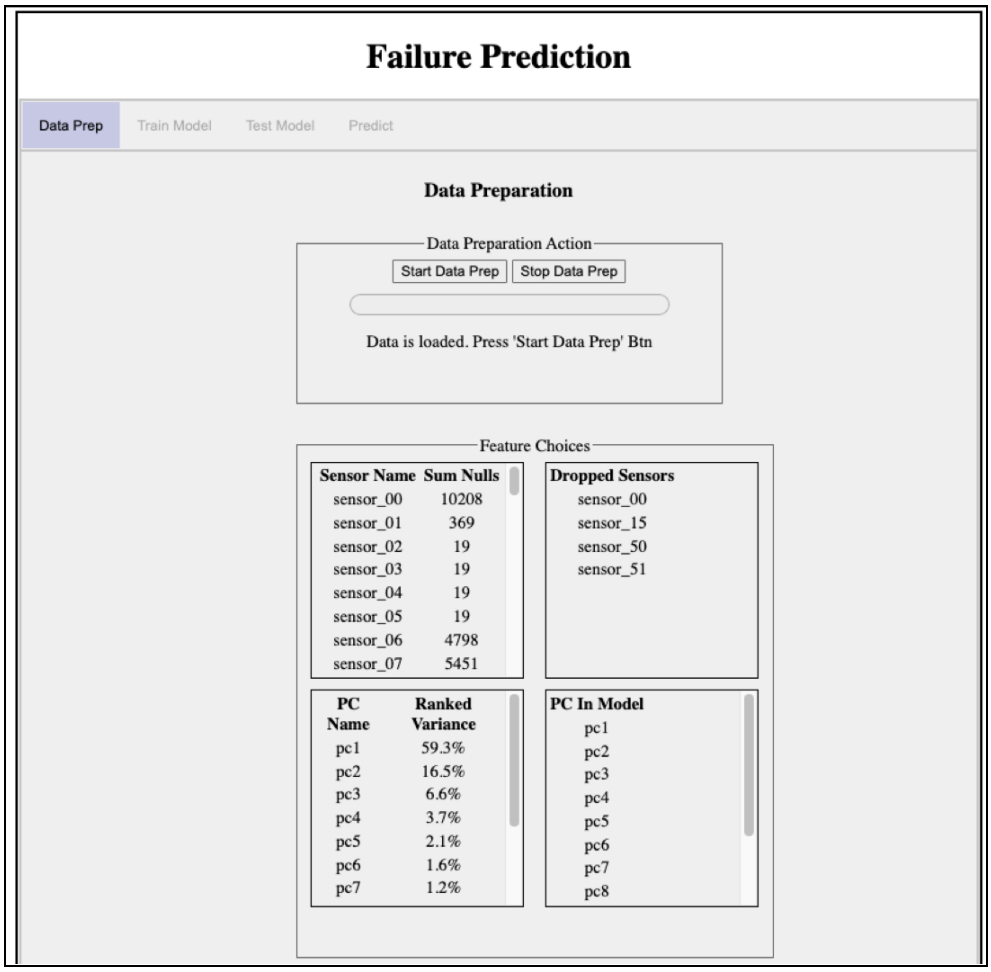
Click on the Data Prep tab.

If you click on the Data Prep tab before all the data has been loaded, you will see:



The message: **Loading data….** means that you cannot proceed past this page until the data are loaded. Notice that the Start Data Prep button is disabled, and will stay disabled until the data are loaded.

Once loaded, the **Start Data Prep** button will be enabled.



The information that you see in the box labeled **Feature Choices** refers to what you discovered in the start of the workshop. That is, you did some data discovery. The box in the upper left shows how many nulls were found in each sensor column. That is why, for example, the box in the upper right shows a list of the sensors that will be dropped, including sensor_00.

The box in the lower left shows a list of results from the **Principal Component Analysis (PCA)** that you found in the data discovery. In the original data there were about 52 sensors. We would like to shorten this list to make the model training take less time. Instead of going through a long process of deciding which sensors to keep, we use the PCA that finds which linear combinations of all the sensors gives the most variance. The linear combinations are denoted by pc1, pc2, pc3, …, (called Principal

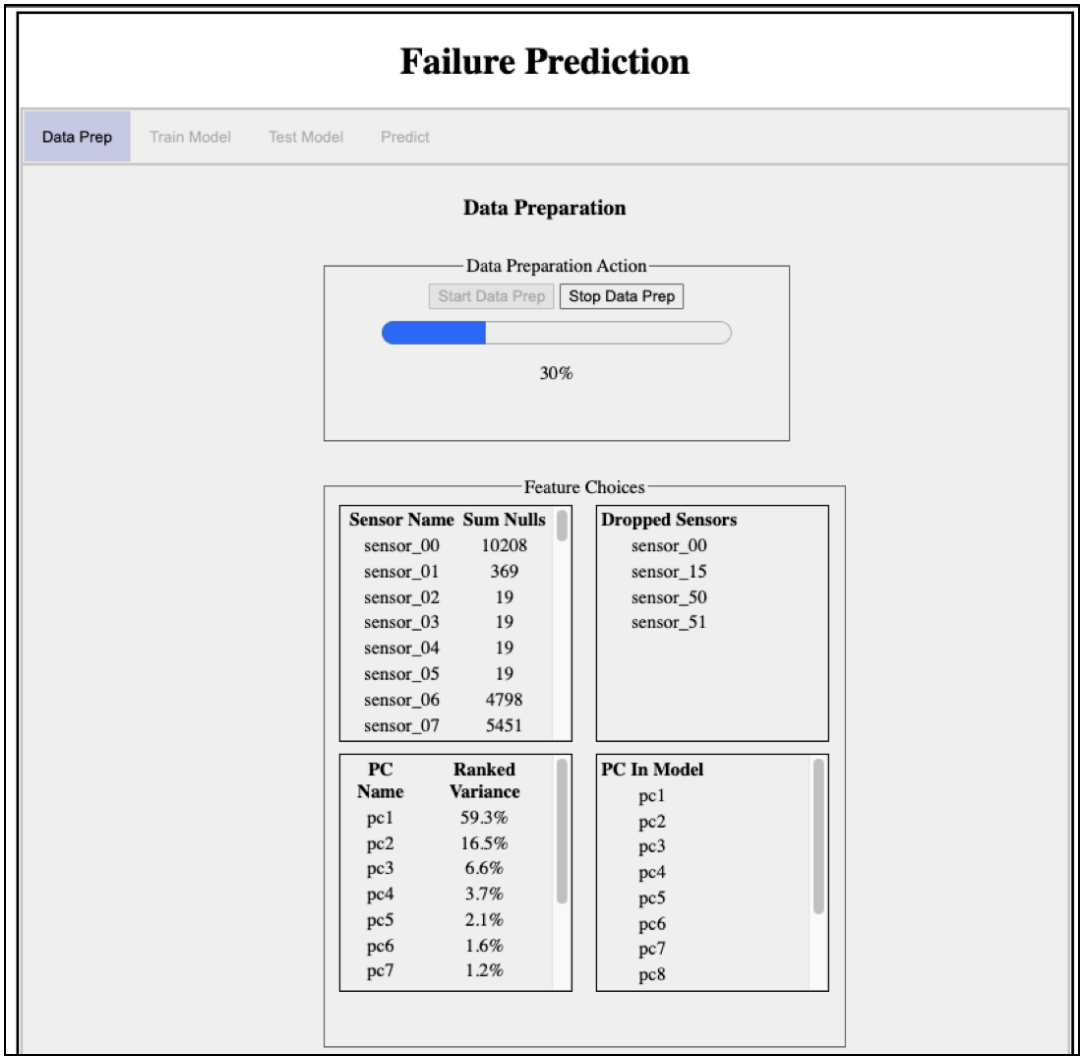
Components) where pc1 is the first ranked. We then take however many pc's we need so that the sum of their ranked variances is at least 95%. In our case it takes 12 pc's to reach that goal. We then use those 12 pc's to represent a transformation of our original 52 sensors when we train the model.

Now that we have all the data loaded and the PCA transformation is complete, we are ready to proceed with preparing the data.

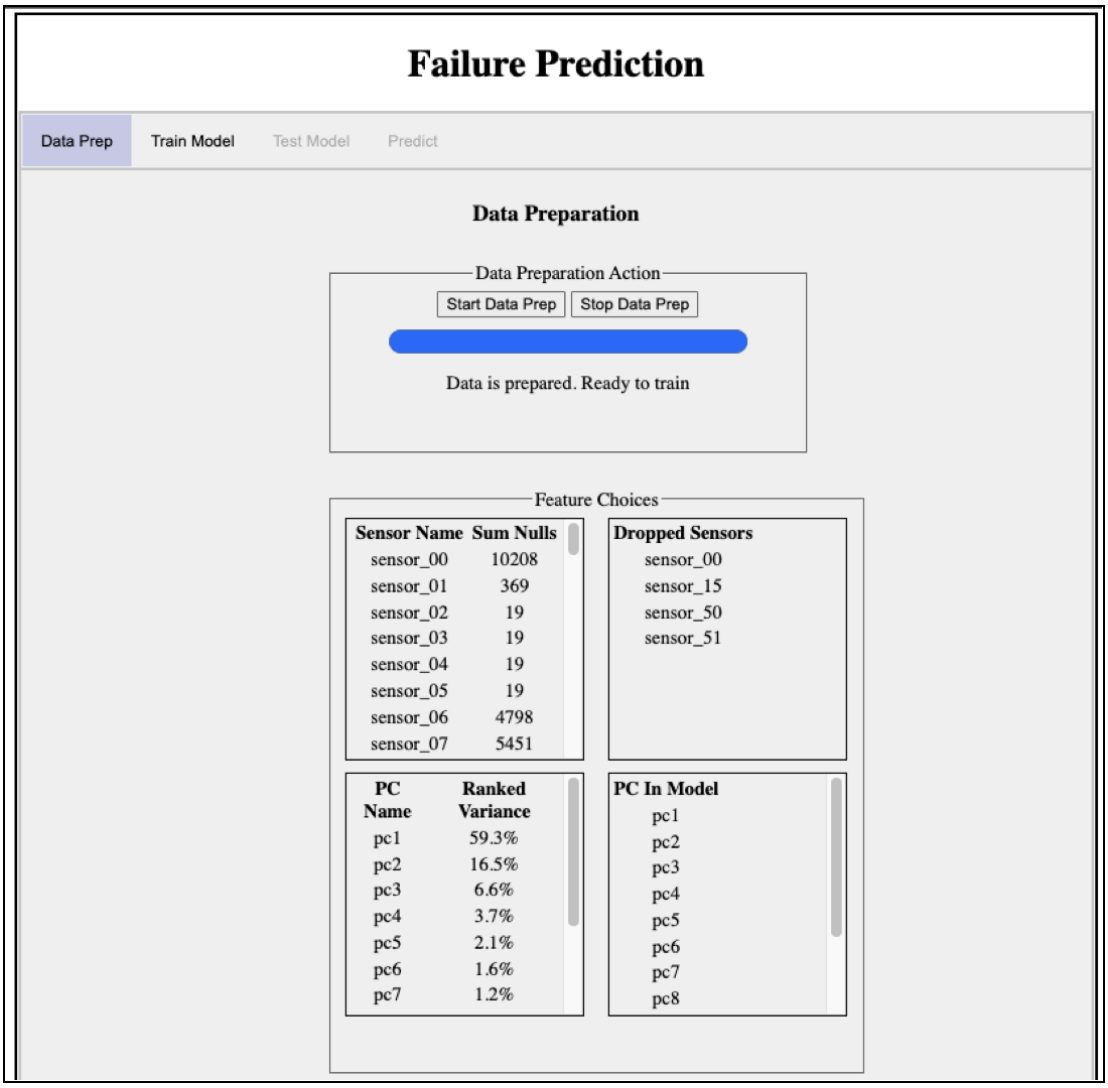
Click on the Start Data Prep button.

Each row of data that we will use has a timestamp together with the 12 pc's that we will now call **model features**. Data that is ordered by time is called a **Time Series**.

In the background, the application is reshaping the time series data so that it will be put into a form that the model understands. While this processing is taking place you will see:

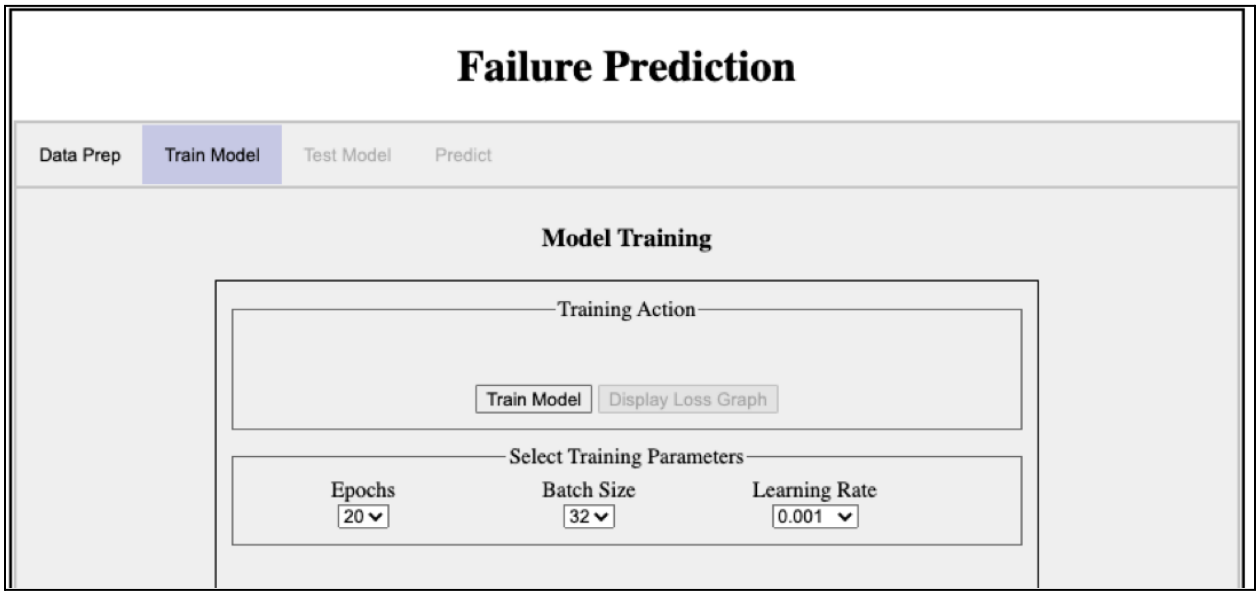


The progress bar will start slowly and at about 50% will speed up. When the data preparation is complete, a message will appear and the next tab, **Train Model** tab will be enabled:


Failure Prediction
Data Prep
Train Model
Test Model
Predict
Data Preparation
Data Preparation Action
Start Data Prep
Stop Data Prep
Data is prepared. Ready to train
Feature Choices
Sensor Name Sum Nulls
sensor_00 10208
sensor_01 369
sensor_02 19
sensor_03 19
sensor_04 19
sensor_05 19
sensor_06 4798
sensor_07 5451
Dropped Sensors
sensor_00
sensor_15
sensor_50
sensor_51
PC Name Ranked Variance
pc1 59.3%
pc2 16.5%
pc3 6.6%
pc4 3.7%
pc5 2.1%
pc6 1.6%
pc7 1.2%
PC In Model
pc1
pc2
pc3
pc4
pc5
pc6
pc7
pc8

Training the Model

Click on the Train Model tab.



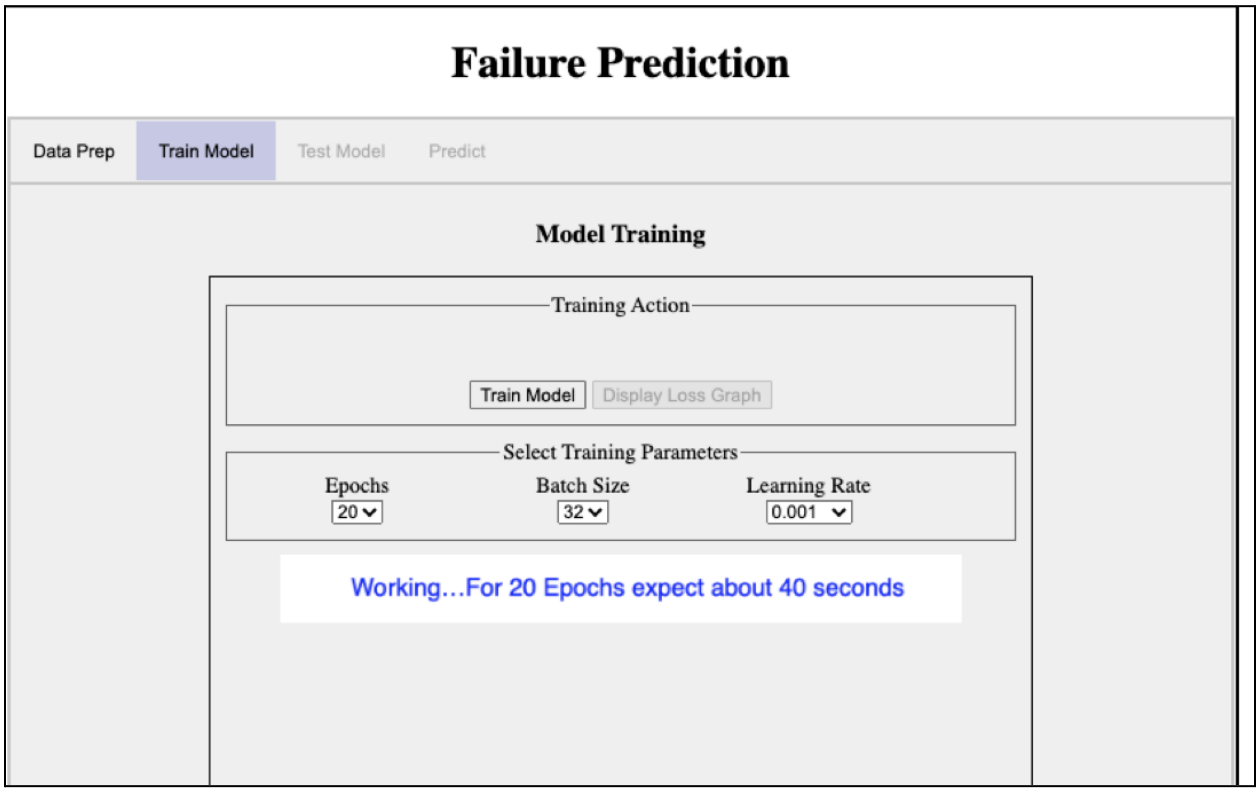
The training process of a model is sometimes called the **learning process**. This learning process is iterative. A model is a numerical approximation of an entity, and the training data are used to calculate coefficients of the approximation. The training data contain values for the features (input) together with values for the target (output). So during one pass (called an **epoch**) on the data, small samples of the training data (called a **batch**) are used to update the model's coefficients.

During training, a parameter called the **learning rate** is used to determine the "pace" of the iterative steps used to update the coefficients. If the learning rate is too large, then the model may step over a critical part of the updating process. If too small, the updates may cause the model to converge too quickly and possibly miss the optimal approximation. In this training session, the learning rate starts as the value in the pull down, but after 10 epochs its value is divided by a factor of 10.

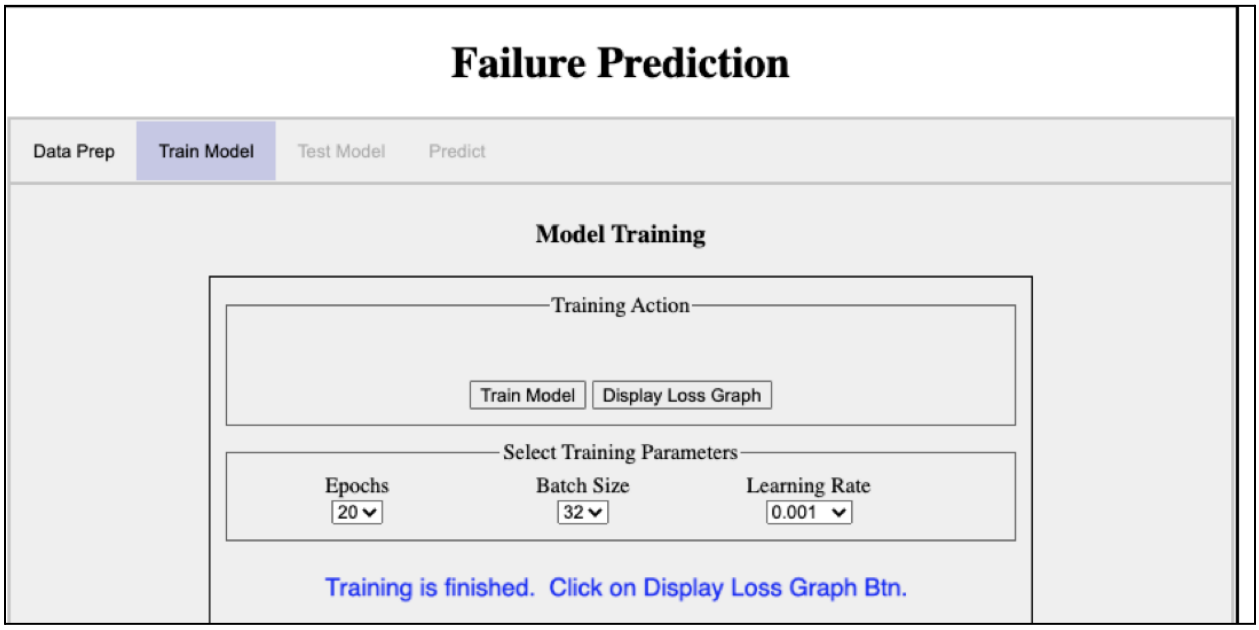
The default values in the pull downs are better left alone for this workshop.

Click on the Train Model button to start the training.

During training, there is no progress feedback. This is because the fit() function only gives progress in the form of text displayed in a terminal console. Since a web application has no console, there is no progress feedback available.

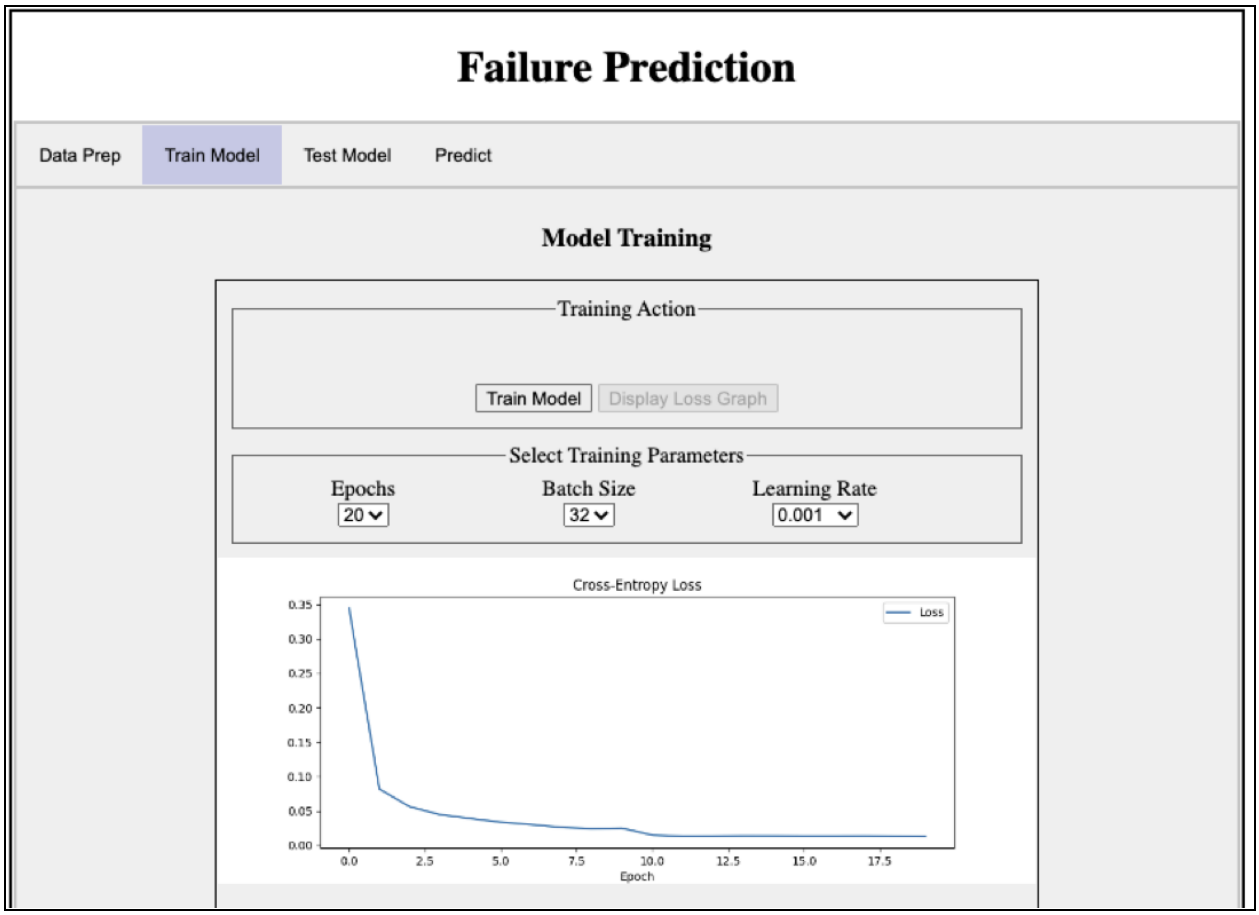


When the training is finished you will see:



Notice that the Display Loss Graph button is now enabled.

Click on the Display Loss Graph button and we will see a graph showing the Loss Graph:

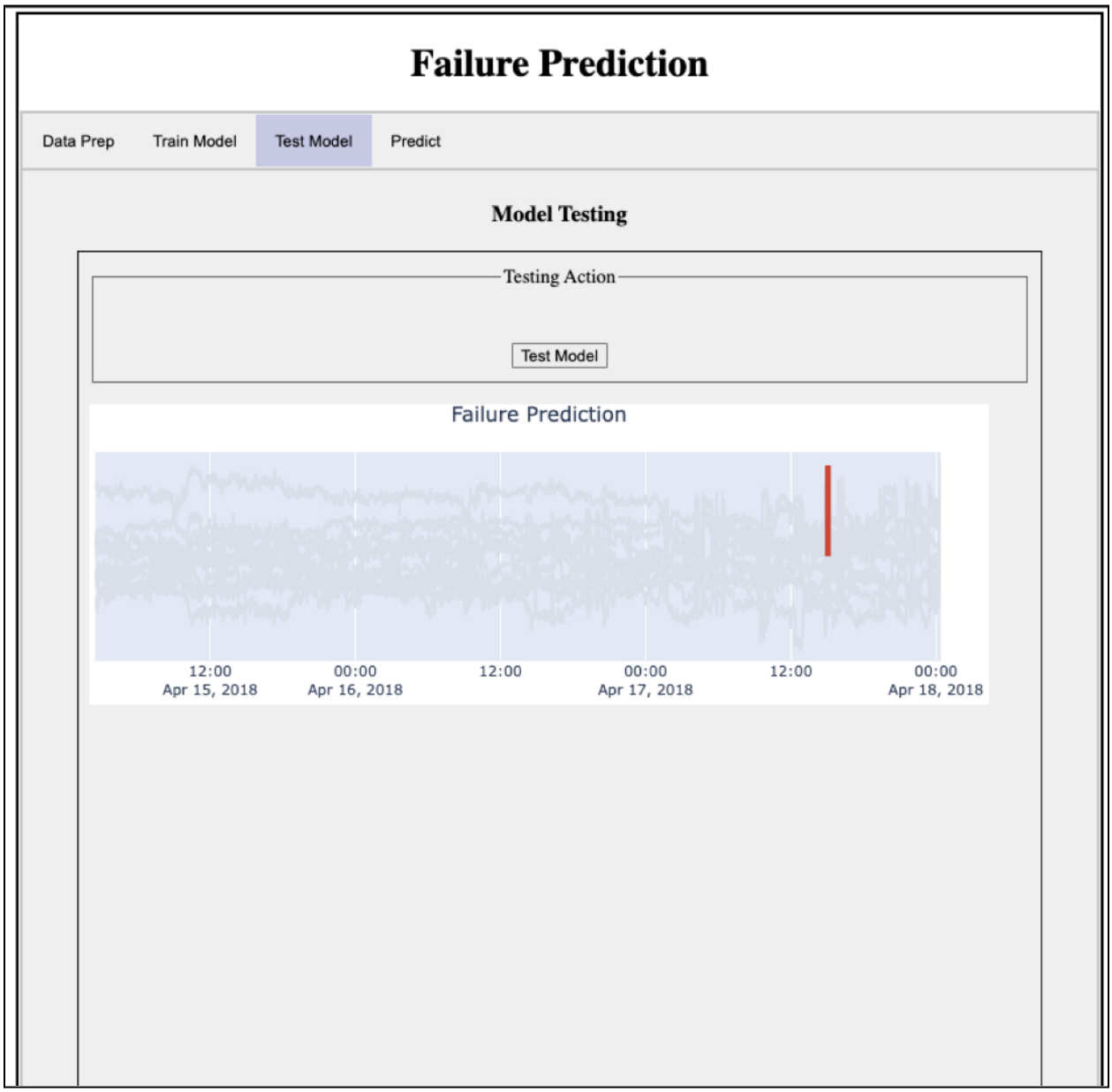


Besides the graph you will notice that the two tabs labeled Test Model and Predict have been enabled. We will run the Test first, although at this point we could just as well run a prediction.

Testing the Model

Click on the Test Model tab. The Test Model tab is sparse because there are no parameters to define for training.

Just **click on the Test Model button.** The result is something like:



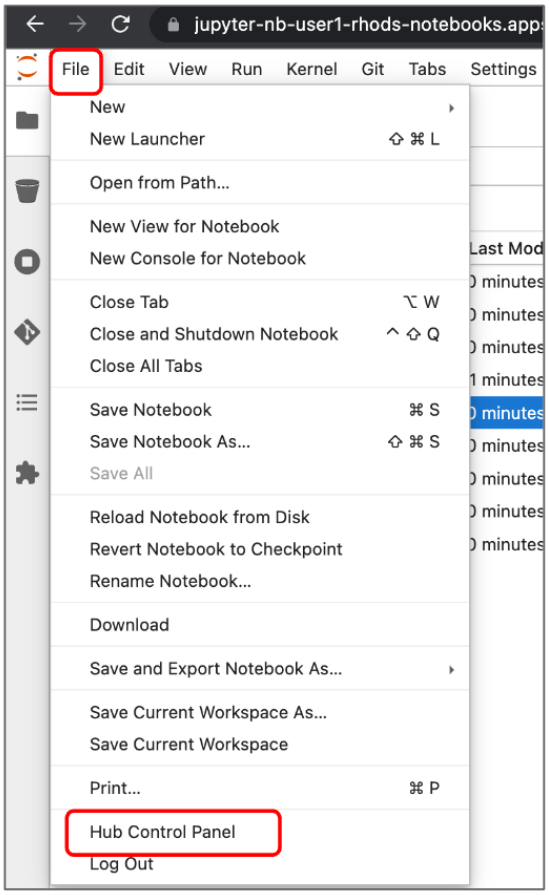
Do not be surprised if your neighbor gets a slightly different result. This is because of all the moving parts of the process, that is, because of the stochastic nature of training the model. Remember that a model is an approximation and the training involves uncertainties and randomness in order to make the approximation.

Before we can look at predicting mechanical pump failures in real time, we need to have real time mechanical pump sensor data to work with. For this workshop, we will not be connecting to a real time mechanical pump. Therefore; to **simulate real time mechanical pump sensor data, we created a synthetic data generator**. This generator will allow you to create real-time synthetic data that our failure prediction model can ingest and determine if a mechanical failure is imminent. Let's look at how we generate Synthetic Data. Continue to the next section - Generate Synthetic Data.

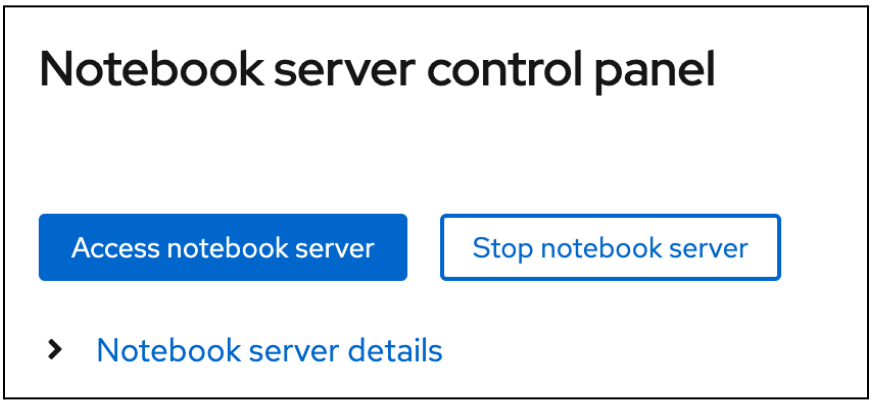
5f Generate Synthetic Data

We are now going to generate synthetic data that we will use in our Failure Prediction Model. **Navigate back to your JupyterLab environment.** We need to change the Notebook server image from 'Standard Data Science' to 'PyTorch'.

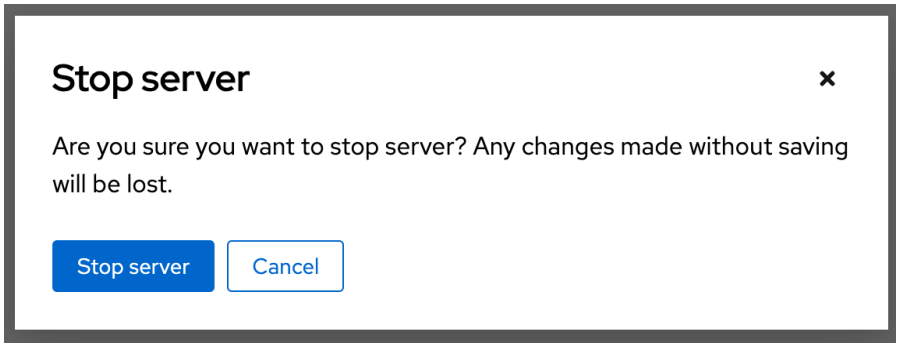
From the *'File'* menu, choose '*Hub Control Panel* option'.



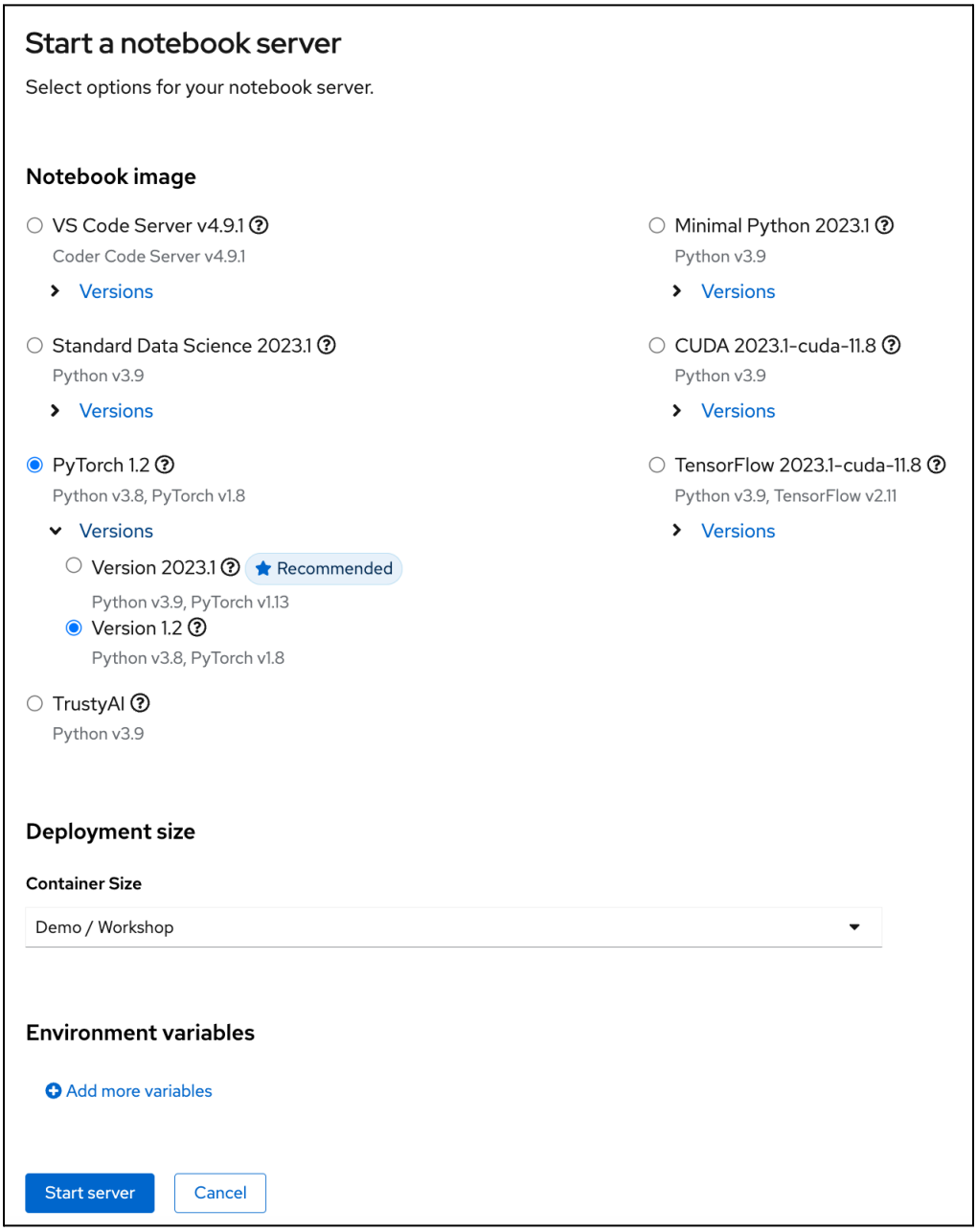
The ‘Notebook server control panel’ page will appear. Click the ‘Stop notebook server’ option.



The Stop server dialog will appear. Choose the ‘*Stop server*’ option.



The '**Start a Notebook Server**' options page will appear. This page should look familiar to you.

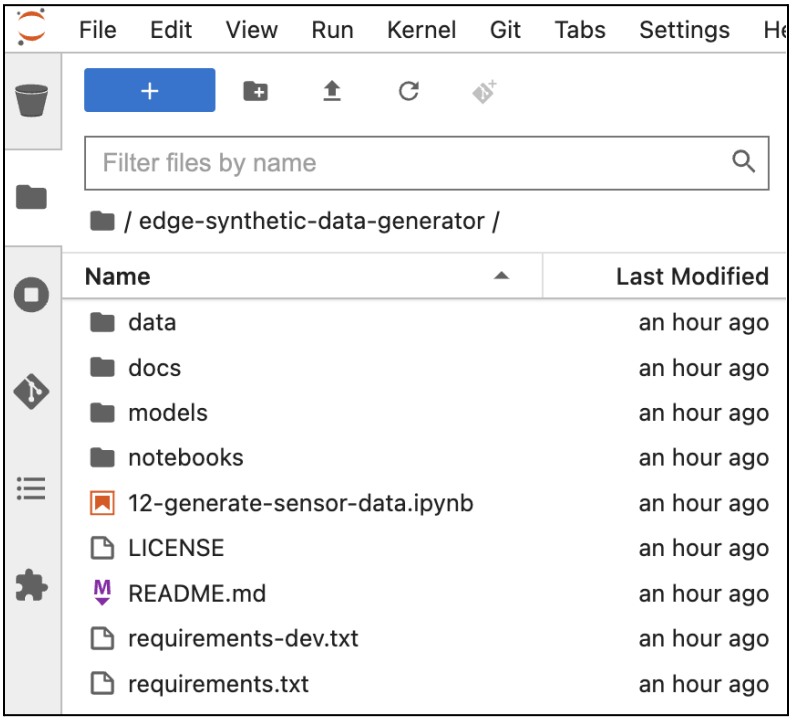


In the '*Start a Notebook' server* page, choose 'PyTorch 1.2 Version 1.2', 'Demo/Workshop' container size. Then press the "Start Server' button.

Once the server spins up you will be taken to the Jupyter Lab environment.

Previously you have already done a ‘git clone’ operation for the ‘edge-synthetic-data-generator’ git repo.

Let’s open up the edge-synthetic-data-generator folder by double clicking on the edge-synthetic-data-generator folder name.

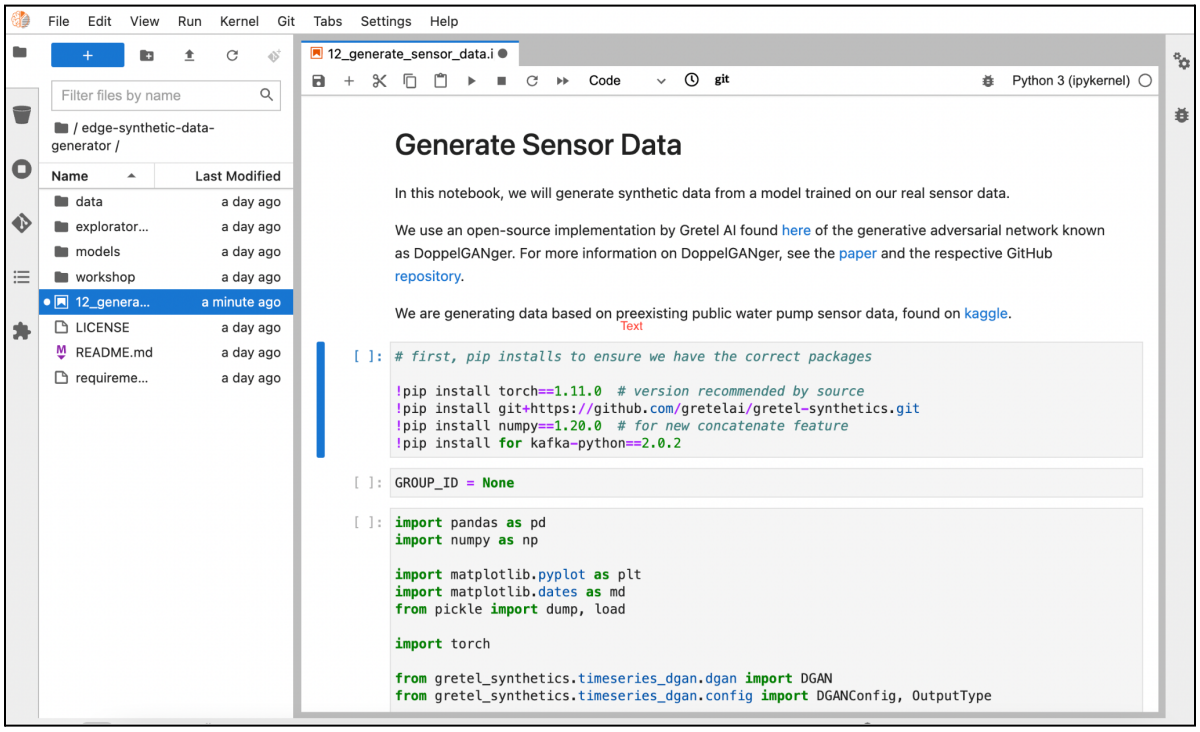


The folder should contain the following sub-folders:

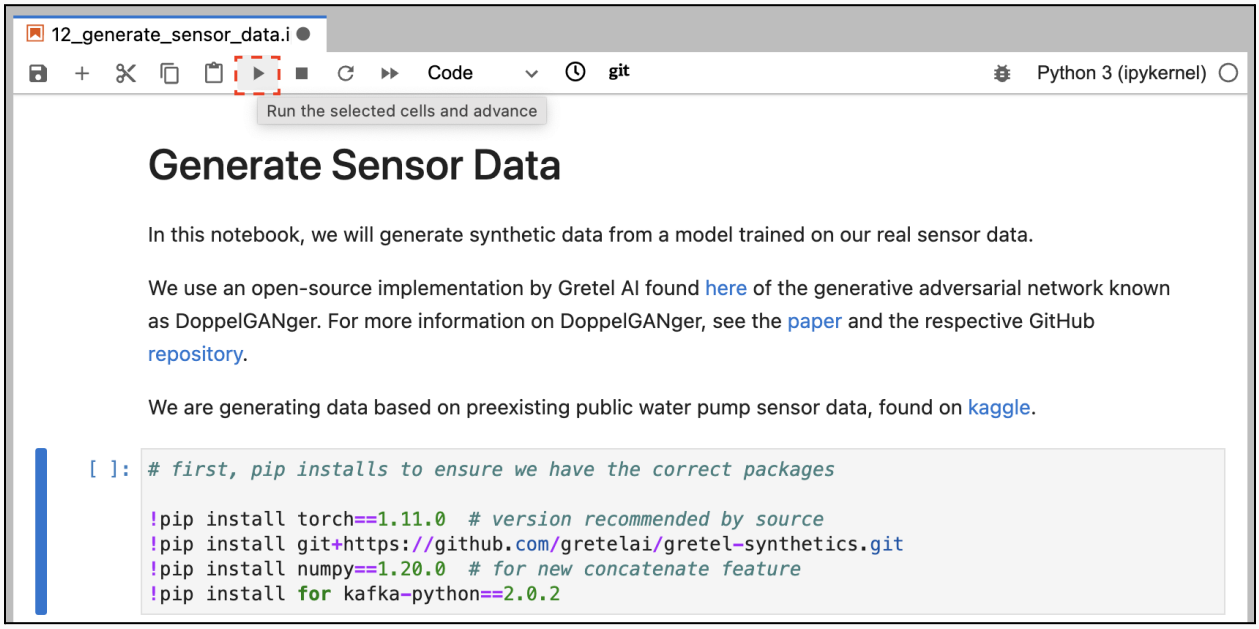
- `data`
- `docs`
- `models`
- `notebooks`

We will be working with the `12_generate_sensor_data.ipynb` jupyter notebook. Double click on the notebook name to open the notebook.

You will be running each of the cells, one at a time, in this notebook.



Run the first cell to install the required packages for this workshop by selecting it and clicking on the run button or press the <shift><enter> keys on your keyboard.

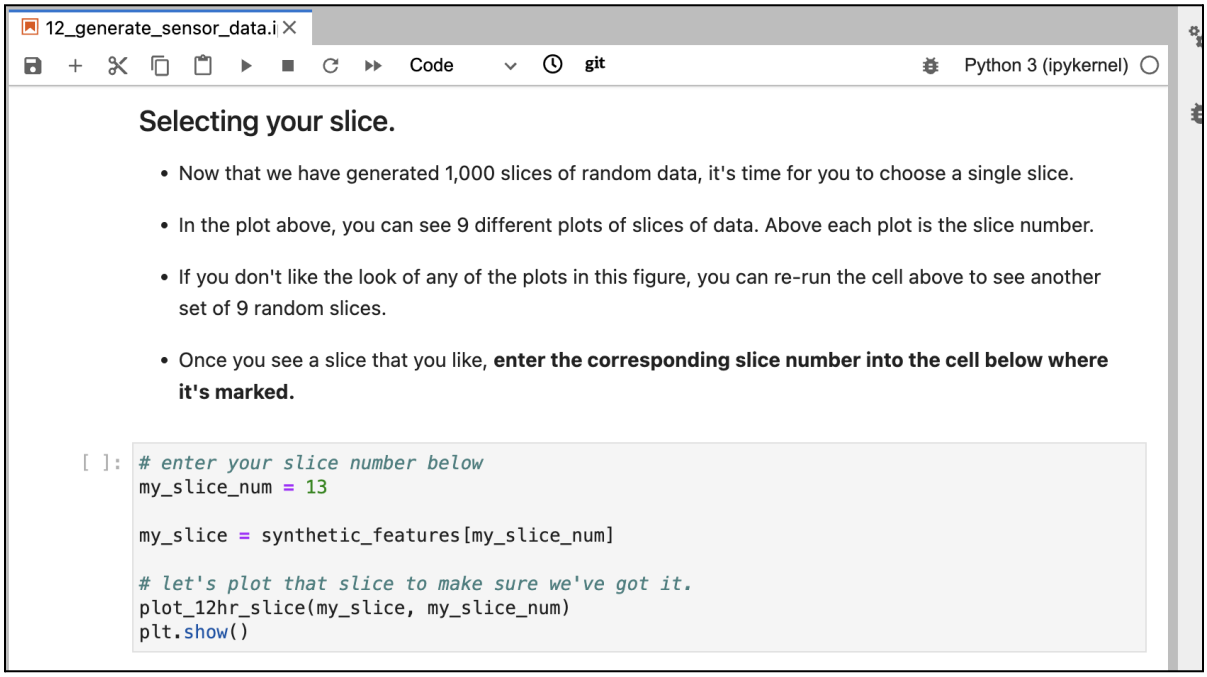


Next, in the following cell, replace 'None' with the GROUP_ID number that is part of your username. E.g. user3: `GROUP_ID = 3`

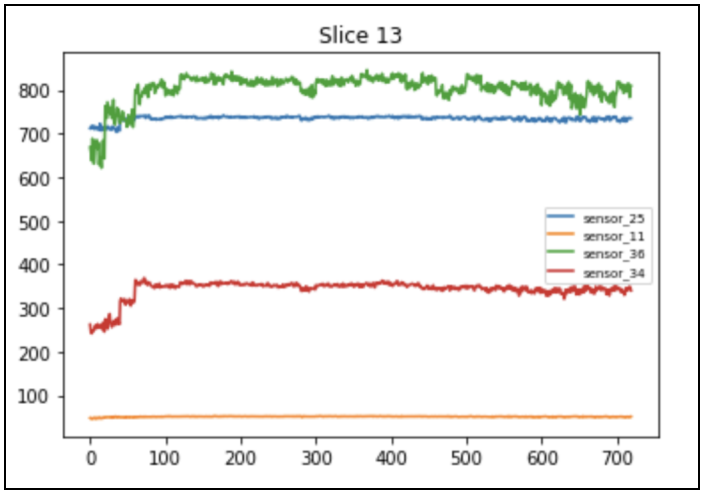
Afterwards, select and run cells 3-7 to generate synthetic data from a pre-trained model and plot some slices of the data.

i) Select a data slice

Once you have run the first 7 cells and reach the section titled 'Selecting your slice', enter in the slice number in the following cell. For example, if your slice number is 13, your code should resemble the following:



After inputting your slice number, run the cell to plot the sensor data for that slice:



Next we will prepare our sensor data, for streaming by Kafka, by running cells 9-11.

Stop at the following warning message:

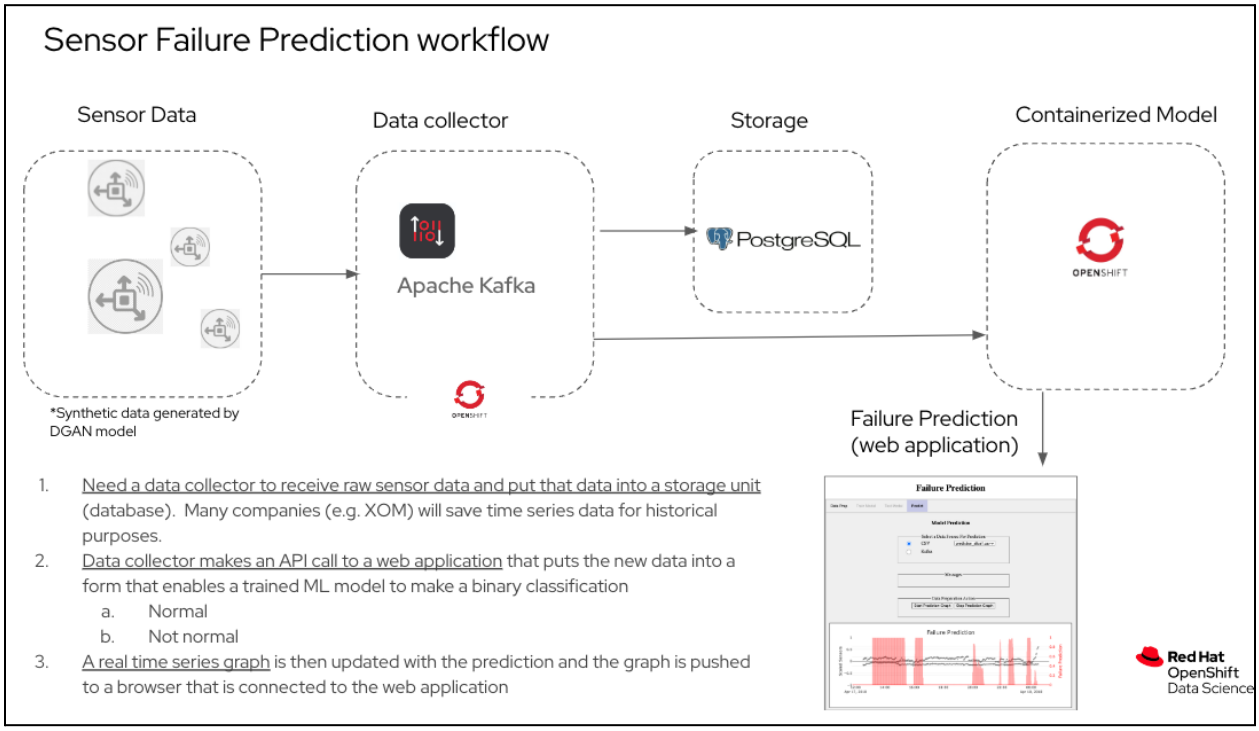
Launcher | 12_generate_sensor_data.i | Code | git | Python 3 (ipykernel)

WARNING: Don't run these last two cells until the failure prediction app is running and you have pressed the Start Prediction Graph button.

```
[ ]: def produce_data(sensor_slice):
    # create the producer
    producer = KafkaProducer(
        bootstrap_servers=[KAFKA_BOOTSTRAP_SERVER],
        security_protocol=KAFKA_SECURITY_PROTOCOL,
        sasl_plain_username=KAFKA_USERNAME,
        sasl_plain_password=KAFKA_PASSWORD,
        api_version_auto_timeout_ms=30000,
        max_block_ms=900000,
        request_timeout_ms=450000,
        acks="all",
        key_serializer=str.encode,
    )

    # sending our data 10 rows per second
    for row_index in range(0, 720, 10):
        # select our 10 rows
        ten_rows = sensor_slice[row_index : row_index + 10, :]
        # wait a second
        sleep(1)
        for i in range(10):
            one_row = ten_rows[i].tolist()
            jsonpayload = json.dumps(
                {
                    "timestamp": one_row[0],
                    "sensor_data": one_row[1:-1],
                    "machine_status": one_row[-1],
                }
            )
            producer.send(KAFKA_TOPIC, jsonpayload.encode("utf-8"), key=str(GROUP_ID))
    producer.flush()  # Important, especially if message size is small
```

Before you run the remaining 2 cells, let's take a look at the Sensor Failure Prediction workflow so that we understand what is happening with our sensor data and in particular why we use Kafka.



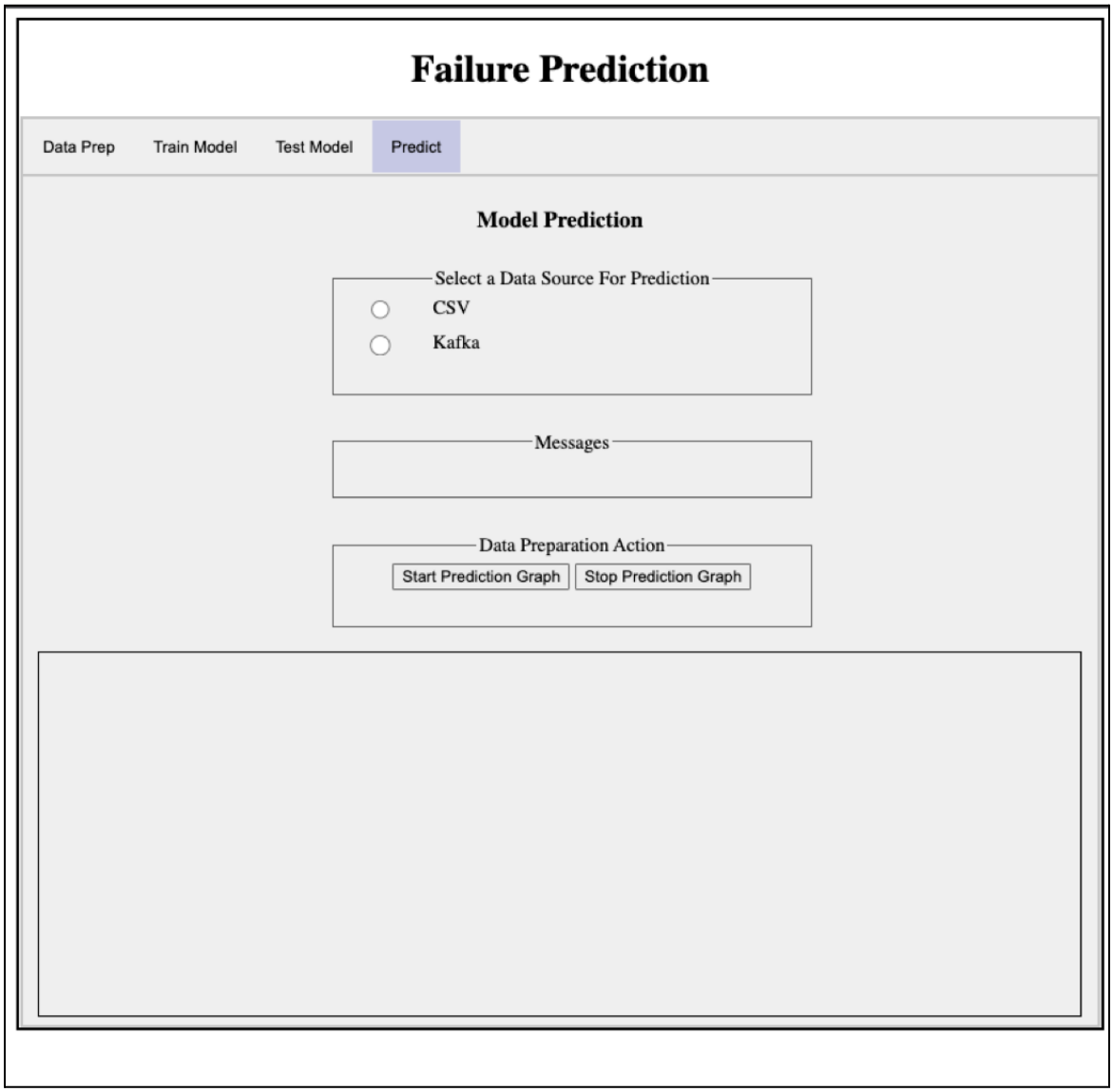
When we generate our synthetic data, it needs a service to push it to our model for failure prediction. The service that we use is called **Apache Kafka Streaming**. This service takes our generated synthetic data and pushes it to our deployed (and containerized) Failure Prediction web application.

We didn't have time in this workshop for you to set up this streaming service. Therefore we have set up the Kafka service for you.

Now that you understand what role the generated synthetic data plays, and what role Kafka plays, we will leave this notebook. Don't close it, as we will come back to the notebook to perform the actual data streaming. We will turn our attention back to the Failure Prediction Web Application that will be using our data.

ii) Run the Failure Prediction

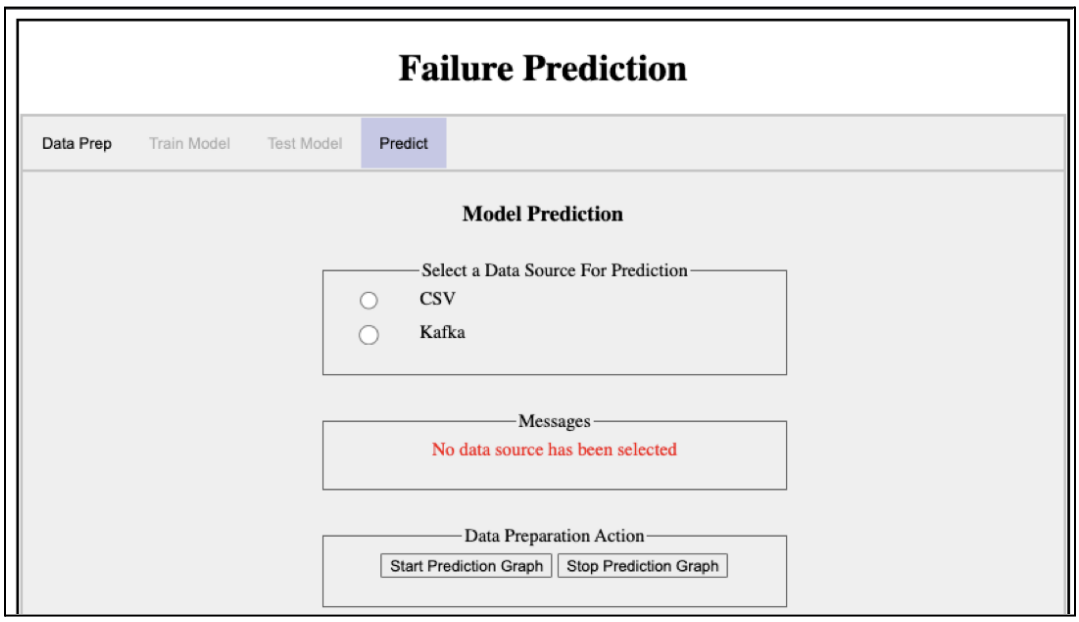
Click on the Prediction tab:



Before we can make a prediction, we must first select a data source for the prediction data. There are two options:

1. Use a CSV file which will be streamed one point at a time, simulating real time generation. The data in the CSV file is taken from the original *Kaggle data source as test data.
2. Use a data stream of *Synthetic* data with the help of Apache Kafka that also simulates the production of real time data.

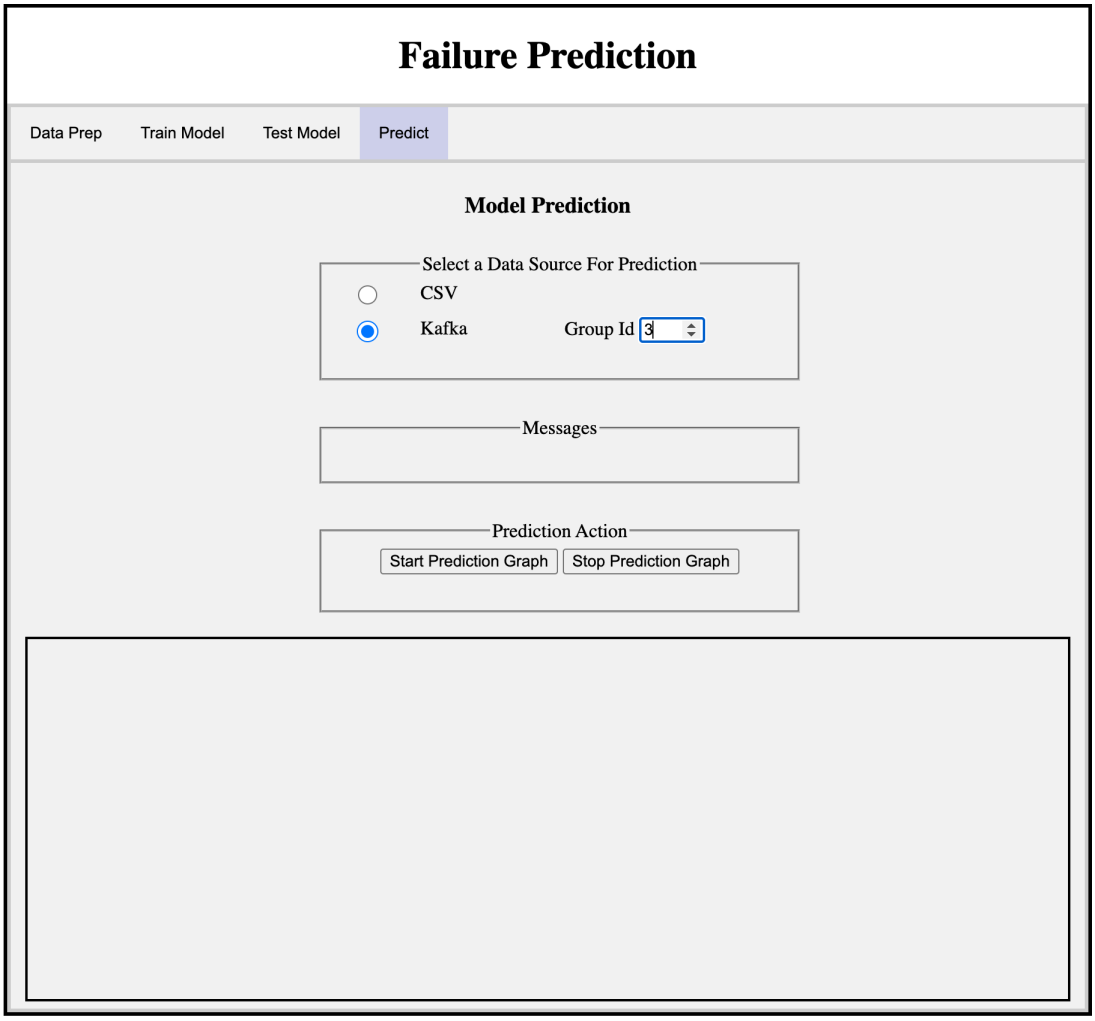
   NOTE: You cannot run a prediction if no data source has been selected. If you attempt to click on the Start Prediction Graph before selecting a data source you will get a red message:



We will use the Kafka Data Source for Prediction. **Select the Kafka radio button, enter the Group Id** that is the number in your username e.g. <user>#. This Group Id allows your browser session to pull out messages that you were identified with.

**The Kaggle water pump dataset can be found at:*

https://www.kaggle.com/datasets/nphantawee/pump-sensor-data

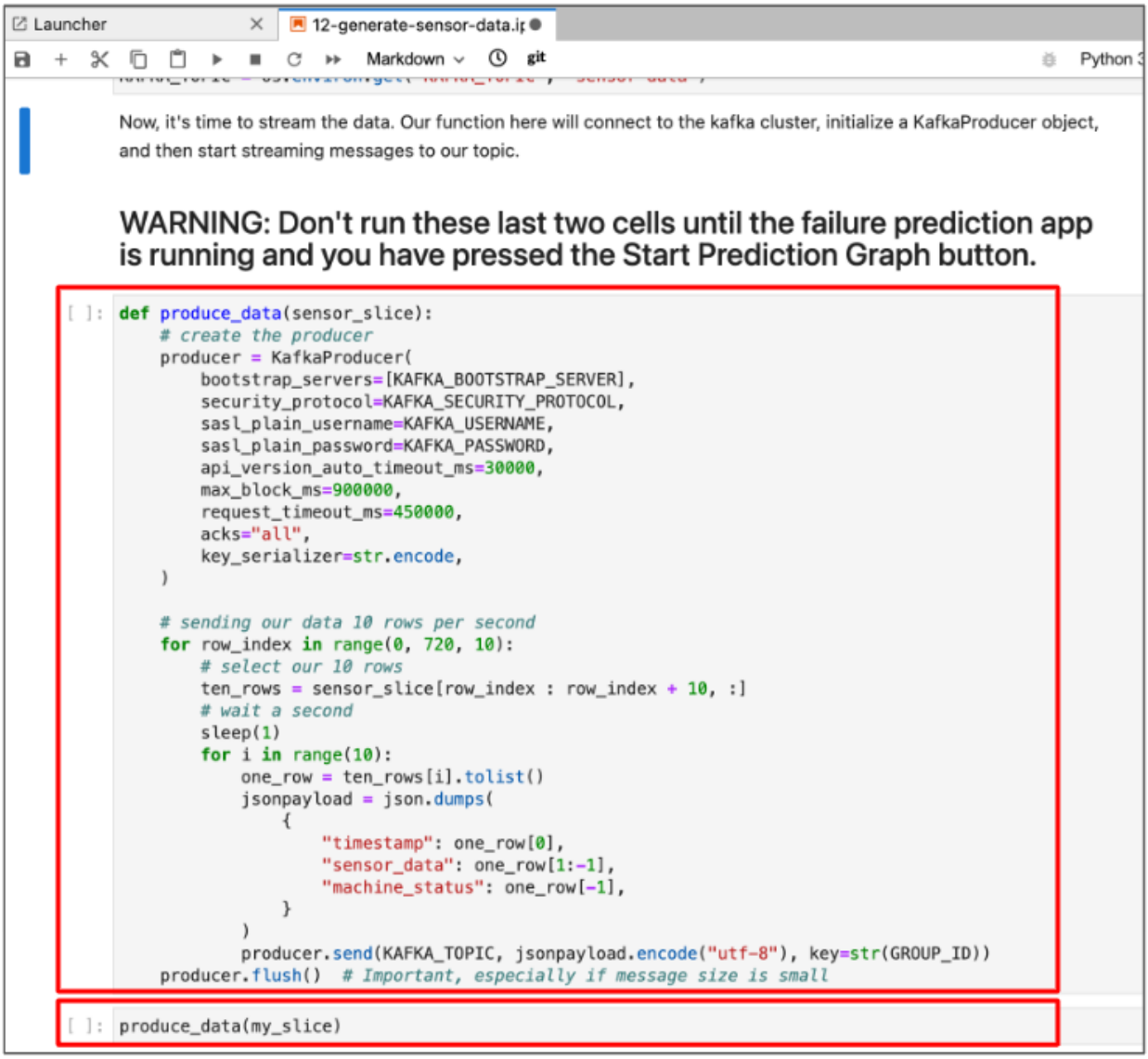


We have selected the 'Kafka' Data Source, and entered '3' as we are logged in with username: 'user3'.

> **Hint**: The purpose of the button labeled **Stop Prediction Graph** is to allow you to stop a prediction process before it is finished. You can use this button if you wish to choose another data source and run the prediction again before the last prediction process has finished.

Click on the Start Prediction Graph button.

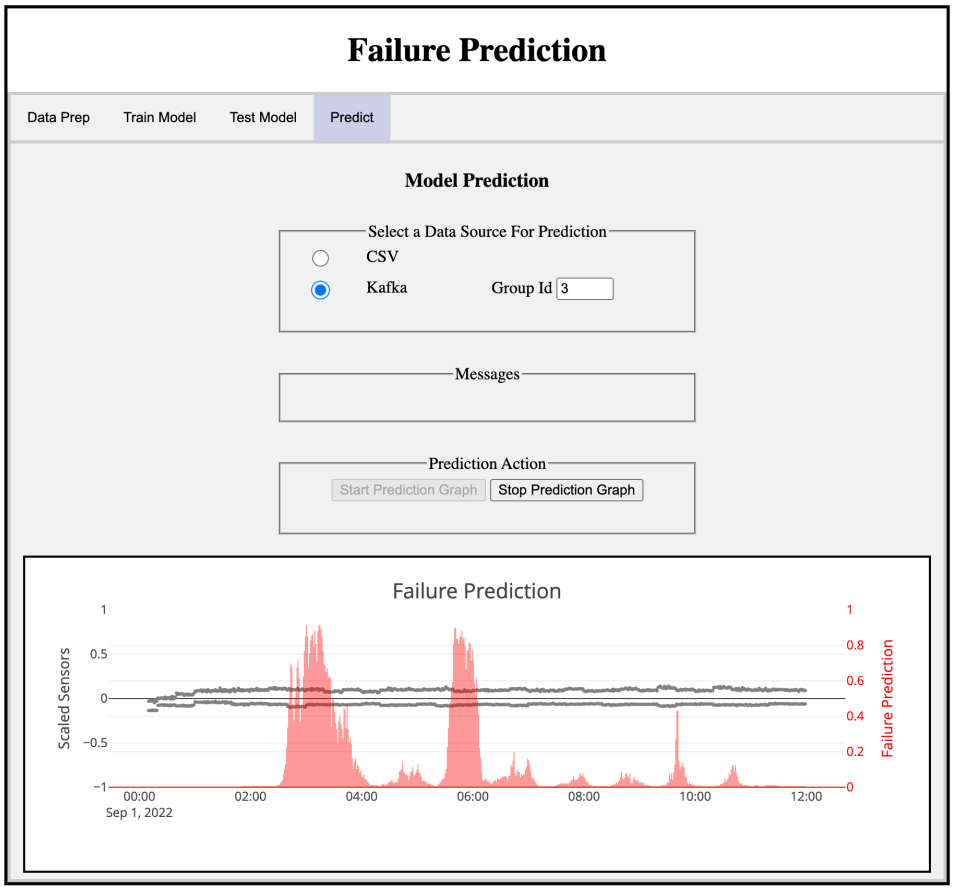
Head back to the Jupyter notebook where you generated synthetic data and run the remaining 2 cells which will (1) connect to the Kafka cluster based on the credentials you defined in the previous step, (2) initialize a KafkaProducer object, (3) stream your data to the sensor failure prediction model



Circle back to the web app to see your prediction graph updating as Kafka is streaming the synthetic data that you generated to your model.

There will be a pause while the data stream is prepared. Then you will see points from the prediction data source as well as a red prediction alarm if the model predicts failure is imminent.

Note: that the prediction data is only available 12 hours before failure.



The graph in your browser is interactive, so you can move your cursor to display the times.

If you are interested in how real time data is pushed from the server to the browser and then rendered in the browser, feel free to have a look at the following github repository.

- [GitHub - guiderae/WorkingDemos-RealTimeGraph1]

6 Enablement Team Skills Set

6a Data Scientist

• Analyze and prepare data for training/testing
• Develop, test, train model
• Develop algorithm for preparing prediction data
• Develop algorithm for supplying prediction results
• Work with MLOps and Software Engineer to develop APIs to be used in delivered product

6b ML Ops

• Create and maintain the infrastructure that supports the development and deployment of the product which will involve:
    ◦ CI/CD Pipeline (Including version control management)
    ◦ Configuration Management
    ◦ Deployment Automation, Containers
• Work with Data Scientist and Software Engineer to ensure smooth model integration

6c Software Engineer (Full Stack Developer)

• Design and implement the application architecture which includes:
    ◦ Design and implement the database to meet application requirements
    ◦ Integrate the database access with the application
    ◦ Integrate the ML model with the application for training, testing and prediction
    ◦ Design and implement the user interface to satisfy functionality requirements which will include re-training of the model
• Maintain application dependency list
• Work with Data Scientist to ensure training and testing are efficient
• Work with MLOps to monitor application performance

7 Model Serving in RHODS (demo with Guillaume Moutier) 4:00-4:30pm