

# Anomaly Detection and Failure Prediction workshop

1. Introduction
2. Log into the Red Hat OpenShift Data Science Platform
3. Start your Jupyter Image Notebook server
  - a. The Jupyter Lab environment
  - b. Clone a github repository
  - c. Introduction to Jupyter Notebooks
4. Anomaly Detection
  - a. Examine time series data for anomalies
  - b. Deploy the Anomaly Detection model & web application as a container
5. Deploy the Failure Prediction model & web application as a container
  - a. Discuss ‘curation’ of time series data
  - b. Log into the Failure Prediction web application
  - c. Generate Synthetic Data
  - d. Stream the synthetic data & Perform a Failure Prediction
6. Conclusion

## Introduction

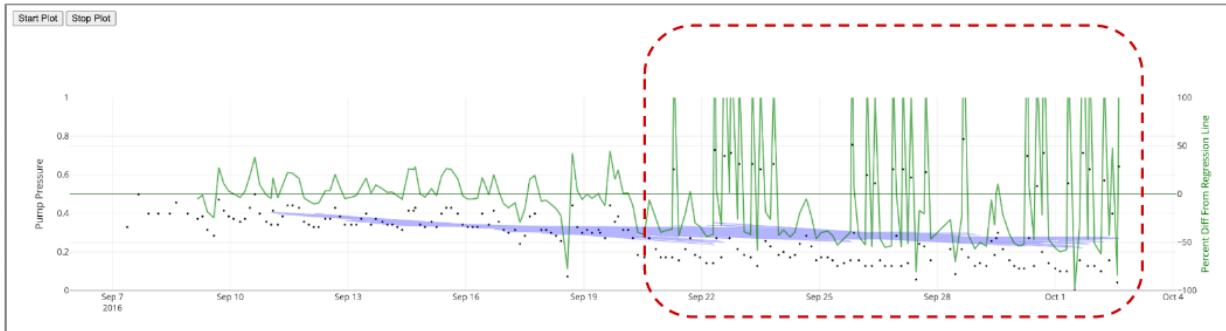
Welcome! In this introductory workshop, you'll learn how to:

1. Use the Red Hat OpenShift Data Science platform to:
  - a. Deploy an Anomaly Detection ‘containerized web application’ (in the Red Hat OpenShift platform) to display time series data which contains an anomaly.
  - b. Generate Synthetic sensor data to mimic real-time data generated by an Edge device. You will use this data in the next step.
  - c. 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.

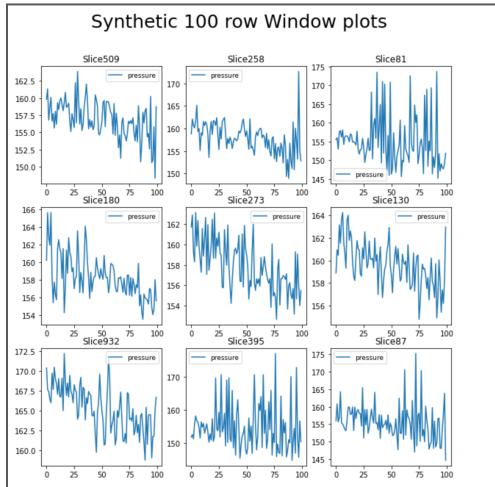
We'll start from a sensor dataset like this:

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

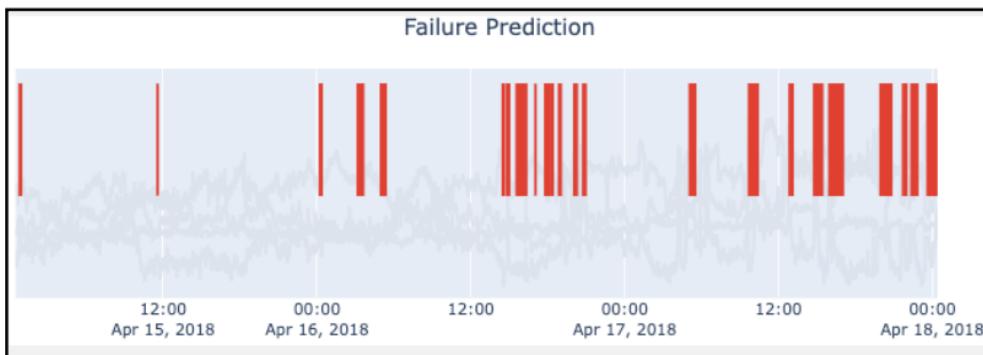
We will learn how to recognize anomalies in our data like this:



We will then generate synthetic data to mimic 'live' sensor data streaming from an Edge device connected to a Mechanical Pump. Plotted 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 that 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!

# Log into the Red Hat OpenShift Data Science Platform

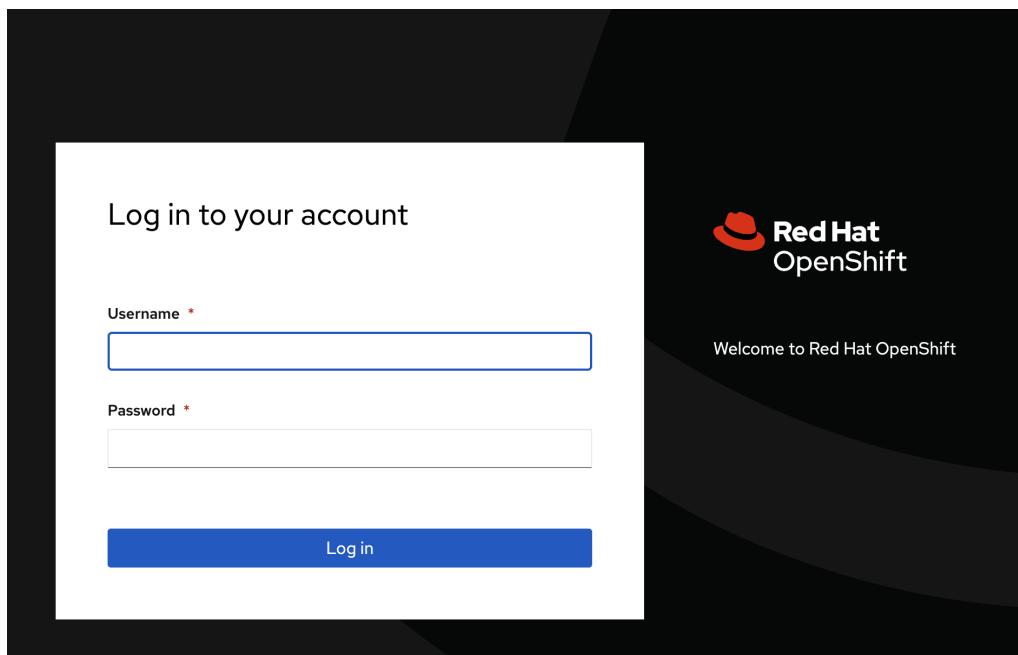
[Red Hat OpenShift Data Science](#) is a platform for [data scientists](#) and developers of artificial intelligence (AI) applications. It provides a fully supported environment that lets you rapidly develop, train, test, and deploy machine learning models on-premises and/or in the public cloud. OpenShift Data Science is provided as a managed cloud service add-on to OpenShift cloud services or as self-managed software that you can install on-premise or in the public cloud on OpenShift.

Let's get started!

**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>

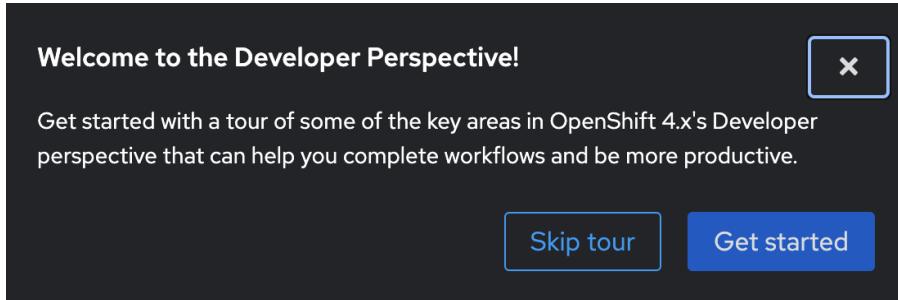
You will see the following login page appear.



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

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

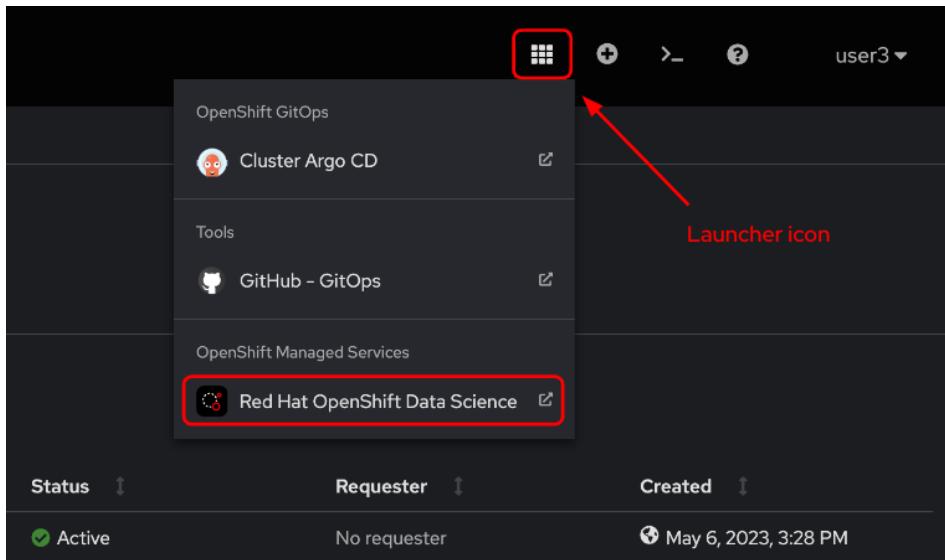
Upon successful login, you will see the **Red Hat OpenShift Dedicated** environment. You may also see a “Welcome to the Developer Perspective!” dialog box. Click “Skip tour” to close this box.



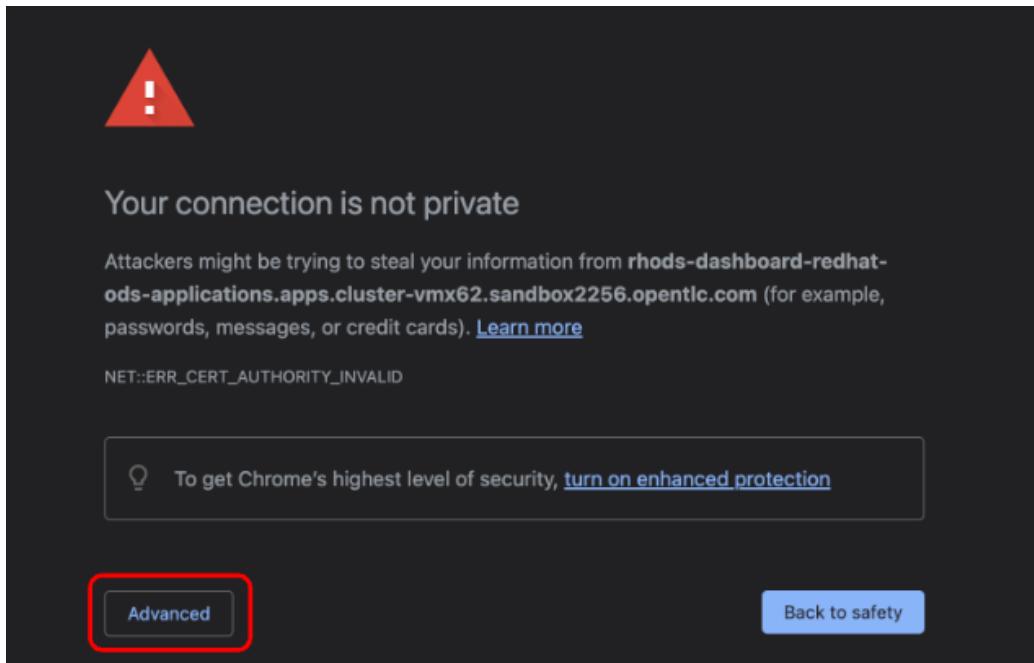
The main Red Hat OpenShift Dedicated environment will be displayed. This is a **kubernetes platform** from which you can create **containerized applications**.

A screenshot of the Red Hat OpenShift Dedicated web interface. The URL in the address bar is https://console-openshift-console.apps.cluster-vmx62.sandbox2256.opentlc.com/add/all-namespaces. The page title is "Welcome to the 'AI/ML Intelligent Applications for the Edge' Workshop". The left sidebar shows navigation options like Developer, Topology, Observe, Search, Builds, Pipelines, Environments, Helm, Project, ConfigMaps, and Secrets. The main content area shows a table of namespaces. The table has columns: Name, Display name, Status, Requester, and Created. The data is as follows:

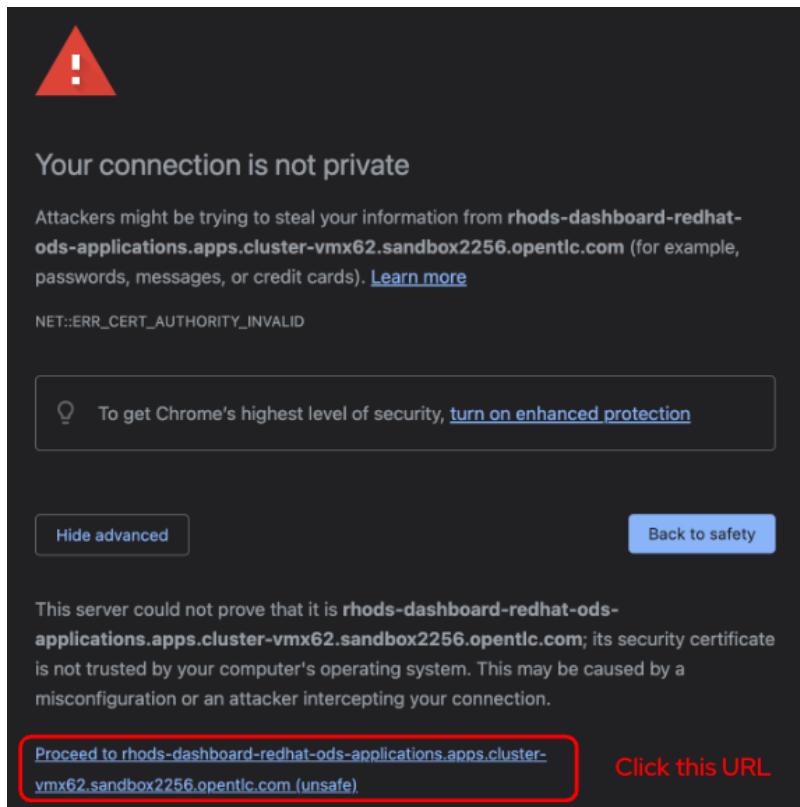
Part of this environment is the **Red Hat OpenShift Data Science platform**. It is accessed by selecting the launcher icon in the upper right corner. **Select the option ‘Red Hat OpenShift Data Science’**



Upon clicking the Launcher Icon you will see a screen which lists ‘*Your connection is not private*’. Below the title and message, there is an ‘**Advanced**’ button. Click it!



After clicking the “Advanced” button, the dialog box will expand and deliver a message about the server being unable to prove that it is the RHODS dashboard. This is ok.



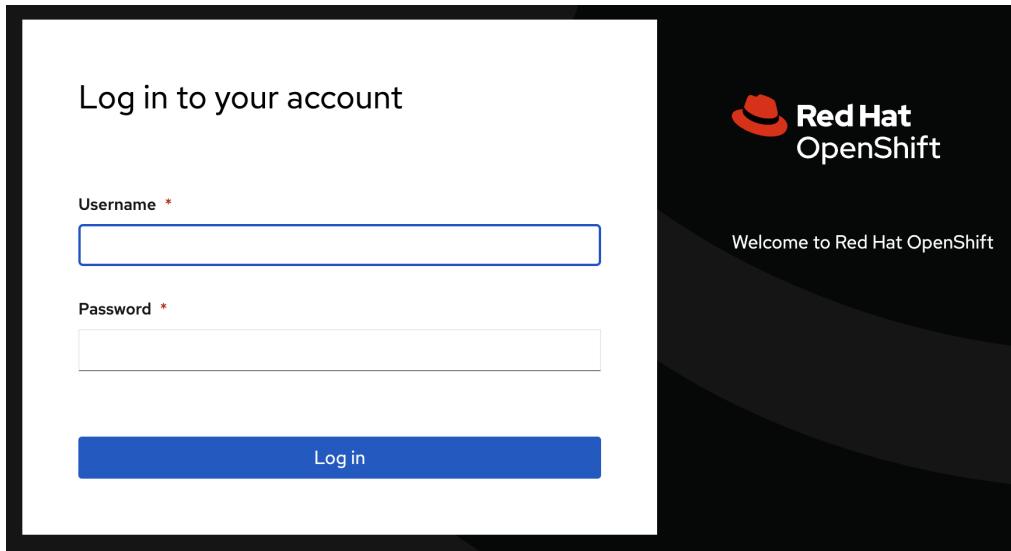
There will be a URL listed at the bottom of the dialog box. Click this URL to log into the RHODS dashboard.

After clicking the URL, a dialog box with a “Log in with OpenShift” button will appear.

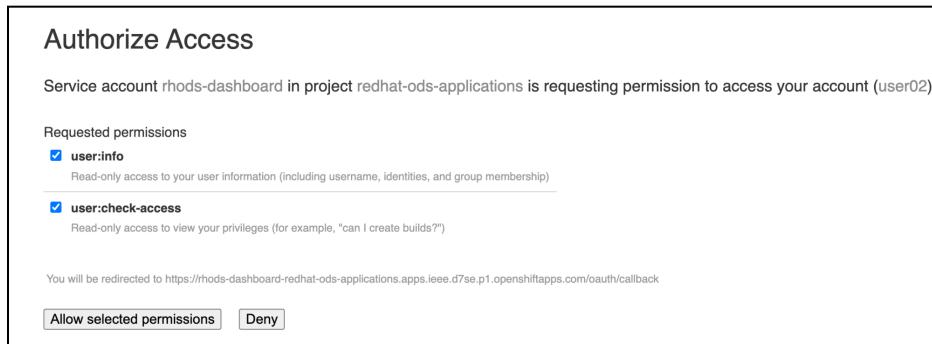


Click the button to proceed with logging into the RHODS dashboard.

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



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



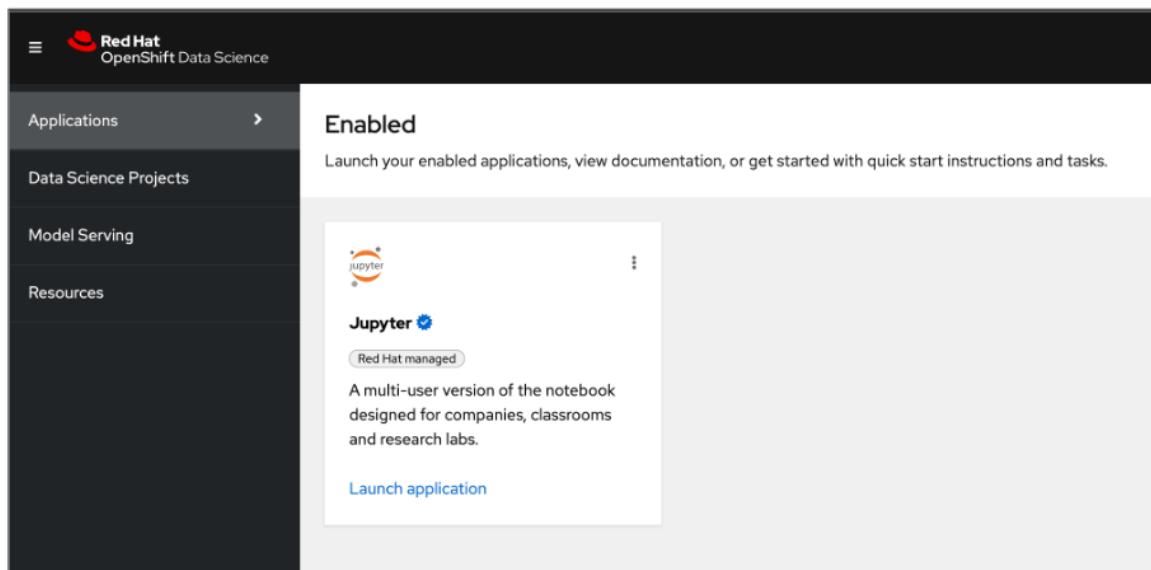
You will now be logged into the Red Hat OpenShift Data Science platform.

The screenshot shows the Red Hat OpenShift Data Science interface. The top navigation bar features the Red Hat logo and the text "Red Hat OpenShift Data Science". On the left, a sidebar menu includes "Applications" (selected), "Data Science Projects", "Model Serving", and "Resources". The main content area is titled "Enabled" and contains the message "Launch your enabled applications, view documentation, or get started with quick start instructions and tasks.". A card for the "Jupyter" application is displayed, showing its icon (a orange circle with white stars), the name "Jupyter", a "Red Hat managed" badge, a description stating it's a multi-user version of the notebook designed for companies, classrooms, and research labs, and a blue "Launch application" button.

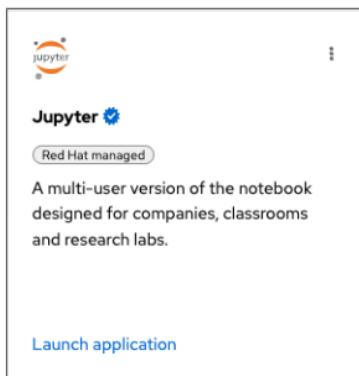
We are ready to start our Jupyter notebook server!

## Start your Jupyter notebook server

[Red Hat OpenShift Data Science](#) is a platform for [data scientists](#) and developers of artificial intelligence (AI) applications. It provides a fully supported environment that lets you rapidly develop, train, test, and deploy machine learning models on-premises and/or in the public cloud. OpenShift Data Science is provided as a managed cloud service add-on to OpenShift cloud services or as self-managed software that you can install on-premise or in the public cloud on OpenShift.



Let's launch the Jupyter Notebook server and customize options for your Jupyter Notebook server! Click the [Launch application](#) url.



## 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 screenshot shows a configuration interface for starting a notebook server. At the top, there are two tabs: 'Notebook Server' (which is selected) and 'Administration'. Below the tabs, the title 'Start a notebook server' is displayed, followed by the sub-instruction 'Select options for your notebook server.'

**Notebook image**

The 'Notebook image' section lists several pre-defined configurations, each with a radio button and a detailed description:

- VS Code Server v4.9.1 ⓘ  
Coder Code Server v4.9.1  
▶ Versions
- Standard Data Science py3.9-v2 ⓘ  
Python v3.9  
▶ Versions
- PyTorch py3.8-cuda-11.4.2-2 ⓘ  
Python v3.8, PyTorch v1.8  
▼ Versions
  - Version py3.9-v2 ⓘ ★ Recommended  
Python v3.9, PyTorch v1.13
  - Version py3.8-cuda-11.4.2-2 ⓘ  
Python v3.8, PyTorch v1.8
- TrustyAI ⓘ  
Python v3.9
- Minimal Python py3.9-v2 ⓘ  
Python v3.9  
▶ Versions
- CUDA py3.9-v2-cuda-11.8.0 ⓘ  
Python v3.9  
▶ Versions
- TensorFlow py3.9-v2-cuda-11.8.0 ⓘ  
Python v3.9, TensorFlow v2.11  
▶ Versions

**Deployment size**

The 'Deployment size' section is titled 'Container Size' and includes a dropdown menu with the option 'Small' selected.

**Environment variables**

The 'Environment variables' section contains a link to add more variables: '+ Add more variables'.

The following subsections list which prerequisite 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.

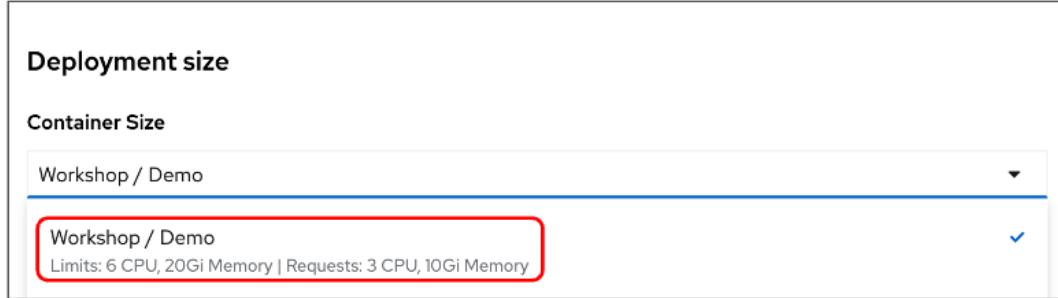
Available notebook images include:

- 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 py-3.8-v1**.

## 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.



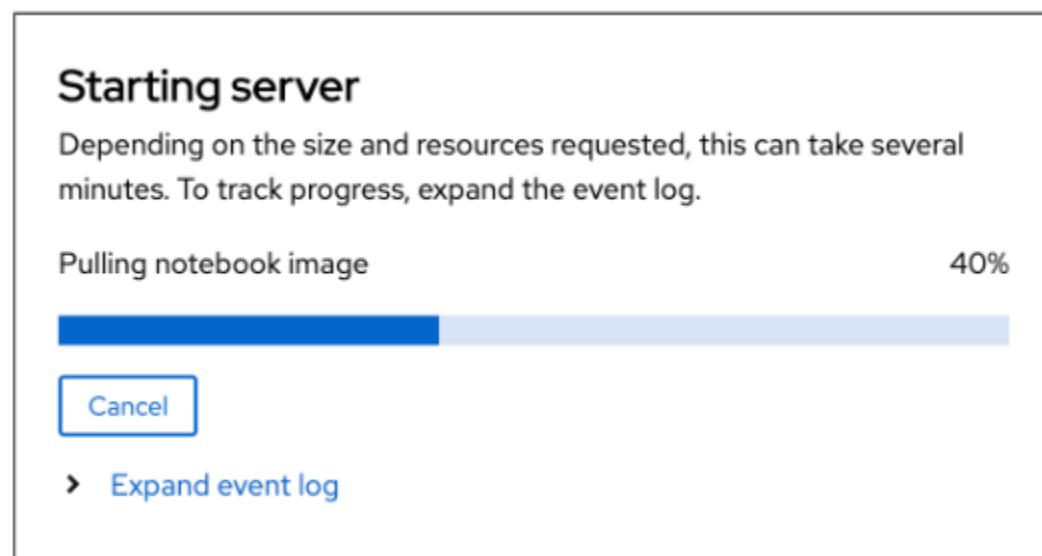
For this learning path, the container size has been pre-configured to 6 CPU, 20 Gi Memory

## Environment variables

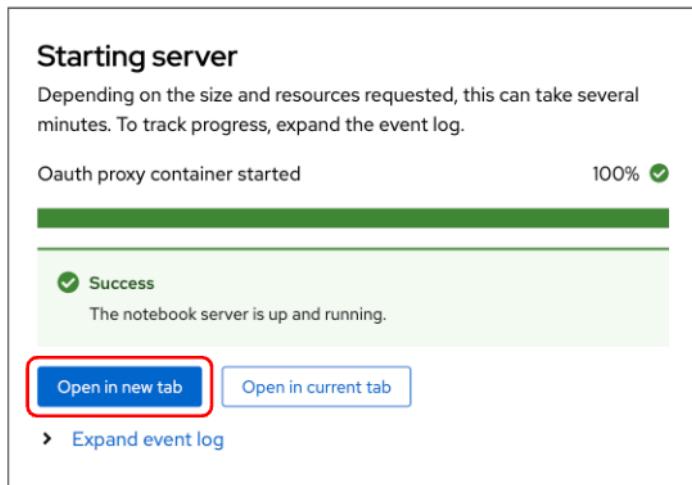
The environment variables section is useful for injecting dynamic information that you don't want to save in your notebook.

- **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.

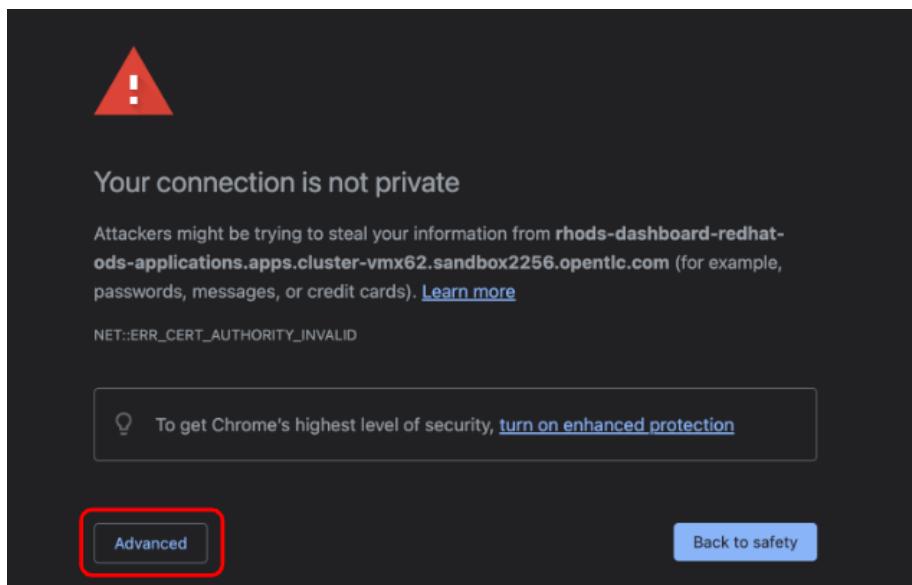


Your Notebook server may take a minute to spin up. If you are interested, you can click the “Expand event log” to see where in the process your Notebook Server is. Once your server is ready, you will see the progress bar turn green and reach 100%.

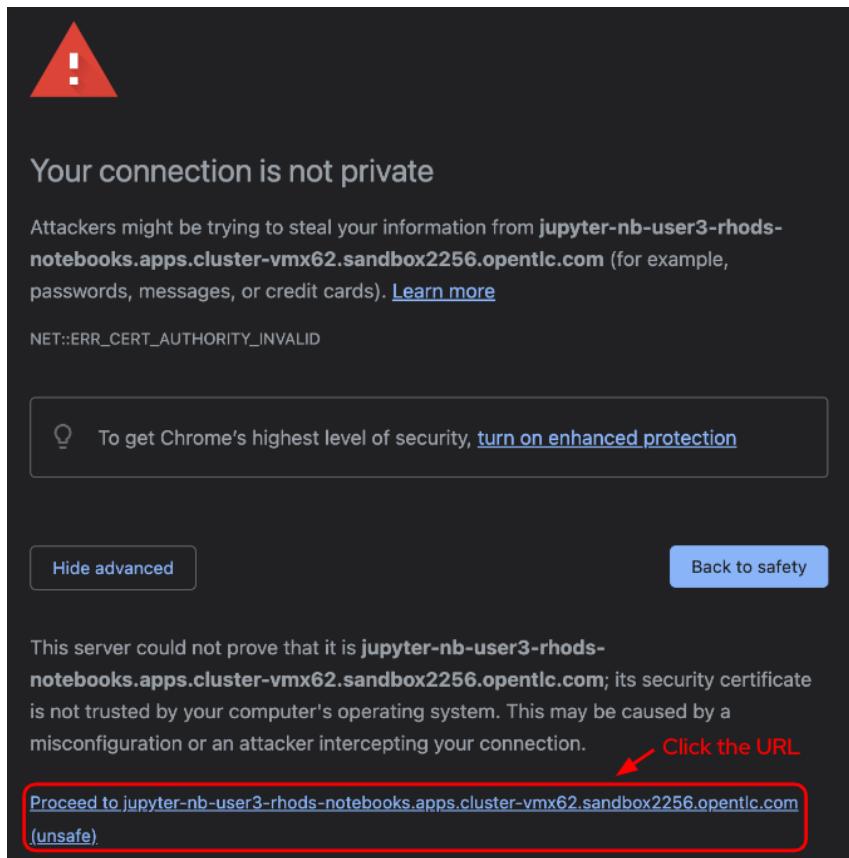


You can now choose to start your Jupyter Lab environment in the current tab or open the environment in a new tab. Click the ‘Open in new tab’ **button** to launch Jupyter Lab in a new tab.

The ‘*Your connection is not private*’ message will be displayed again.

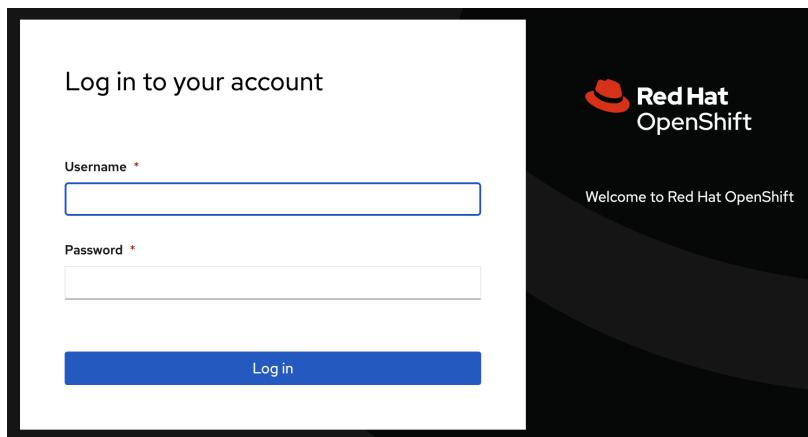


After clicking the “Advanced” button, the dialog box will expand and deliver a message about the server being unable to prove that this is ‘jupyter-nb-user’. This is ok.

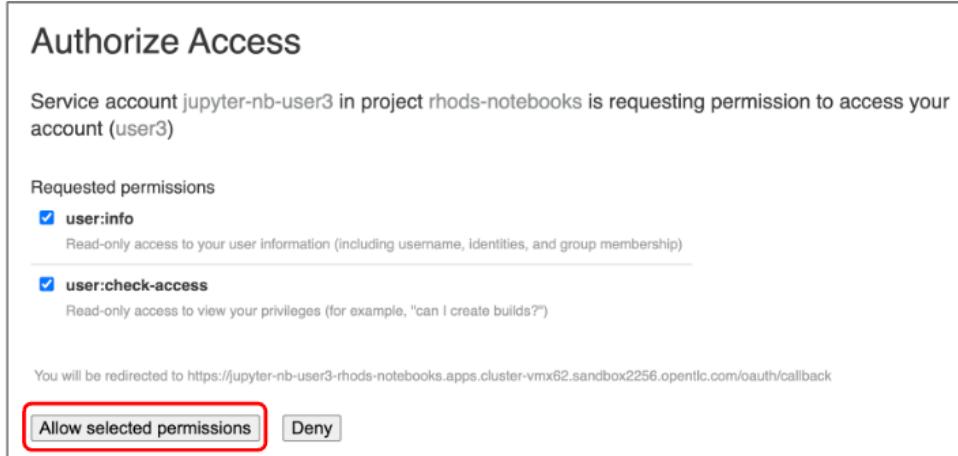


There will be a URL listed at the bottom of the dialog box. Click this URL.

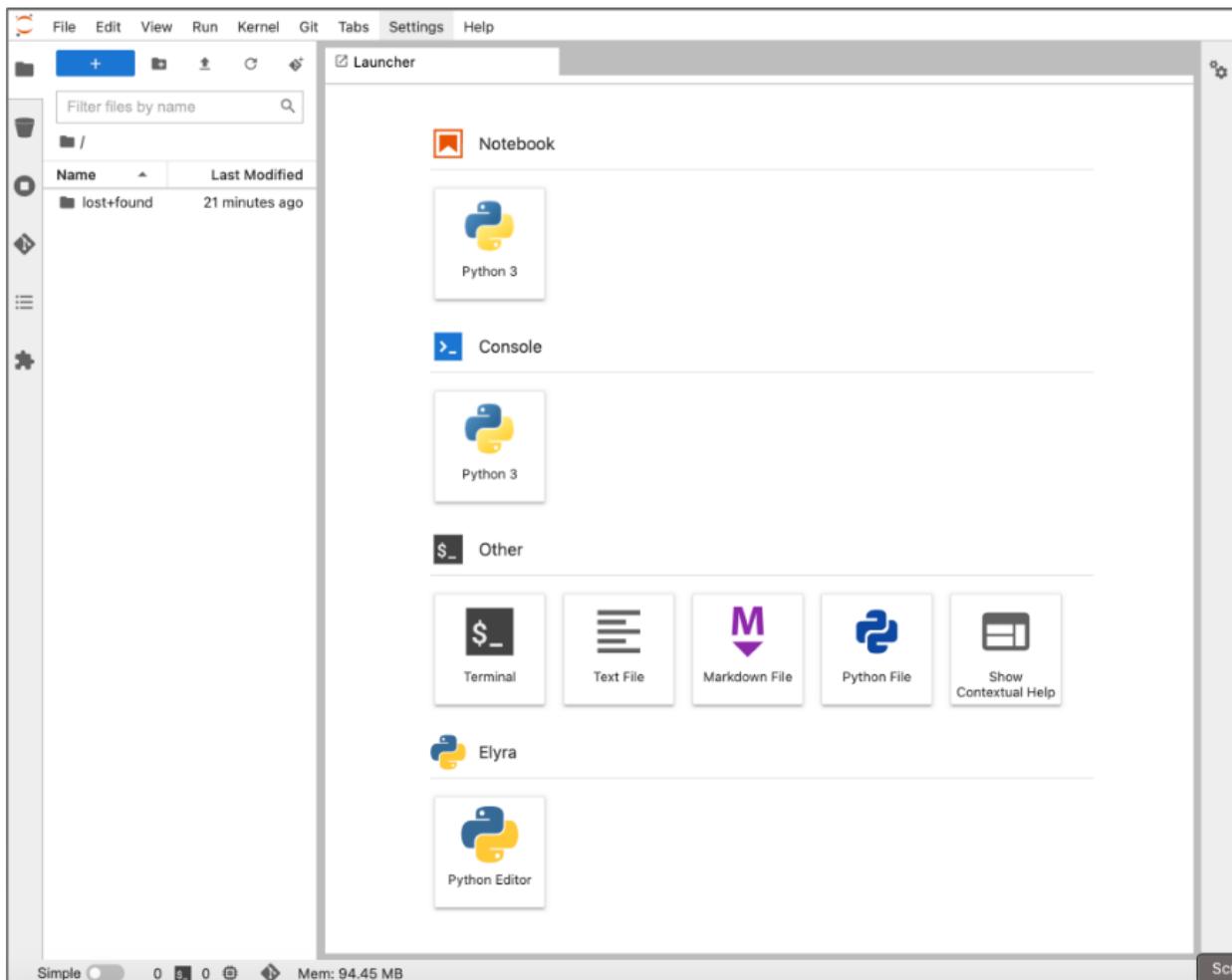
You will see another Log in page. Enter the username and password you were given at the beginning of the workshop, then click the ‘Log in’ button.



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



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



## The Jupyter Lab Environment

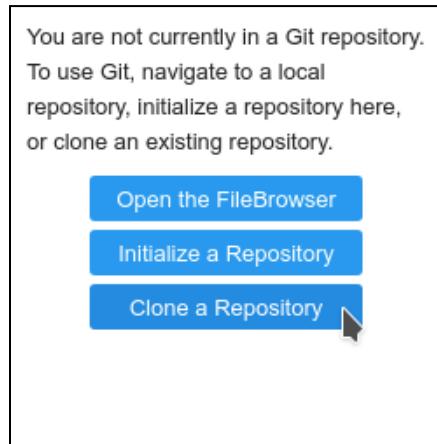
You are now inside your Jupyter Lab environment. As you can see, it's a web-based environment, but everything you'll do here is in fact happening on the Red Hat OpenShift Data Science cluster. This means that without having to install and maintain anything on your own computer, and without disposing 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.

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

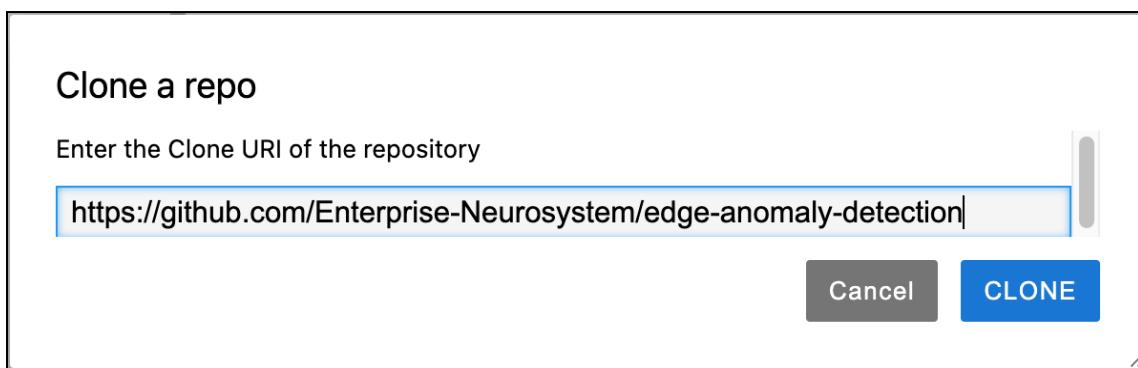


- Then click on Clone a Repository:



**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>, then **click the CLONE button**.



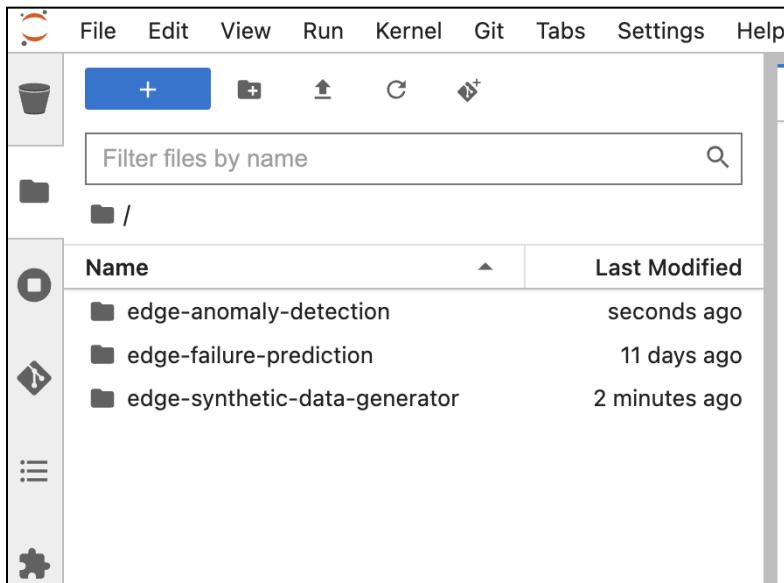
- It takes a few seconds, after which you can double-click and navigate to the newly-created folder, edge-anomaly-detection

**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.

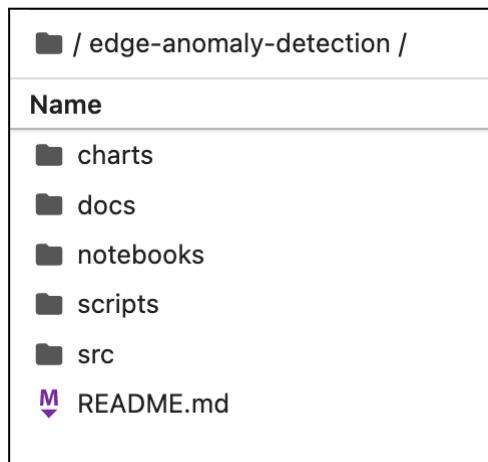
1. <https://github.com/Enterprise-Neurosystem/edge-synthetic-data-generator>

## 2. <https://github.com/Enterprise-Neurosystem/edge-failure-prediction>

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](#).

# 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:

A screenshot of a Jupyter Notebook cell containing Python code. The cell identifier 'In [ ]:' is visible at the top left. The code block contains:

```
In [ ]: def print_some_text(entered_text):
    print('This is what you entered:' + entered_text)

my_text = 'Hello world!'
print_some_text(my_text)
```

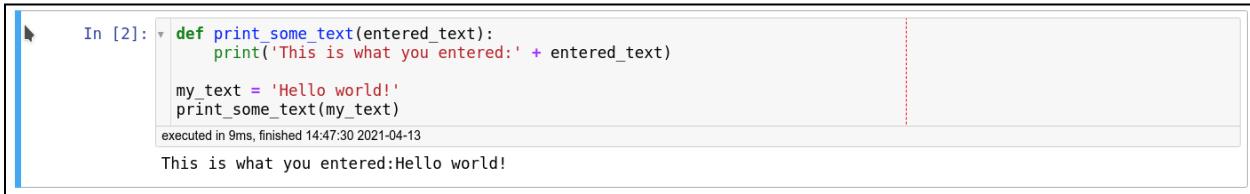
The code is displayed in a monospaced font, with syntax highlighting for keywords like 'def' and 'print'. The output area to the right of the code block is currently empty.

- 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:



In [2]:

```
def print_some_text(entered_text):
    print('This is what you entered:' + entered_text)

my_text = 'Hello world!'
print_some_text(my_text)
```

executed in 9ms, finished 14:47:30 2021-04-13

This is what you entered:Hello world!

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 inline (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](#).

# Anomaly Detection

## 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.

The screenshot shows the Jupyter Notebook interface with two tabs open: "01-sandbox.ipynb" and "02-anomaly-detection-notebook.ipynb". The "02-anomaly-detection-notebook.ipynb" tab is active, showing the following code:

```
[1]: # first, execute pip installs to ensure we have the correct packages
!pip install matplotlib==3.4.1
!pip install pandas==1.2.4

[2]: import matplotlib.pyplot as pl
import numpy as np
import pandas as pd
import matplotlib.dates as mdates
import datetime as dt

# %matplotlib

[3]: # Read data from csv file. Assumes that data is in two columns: datetime and pressure.
# Datetime values are assumed to be in the form: '%m/%d/%Y %H:%M'
# Returns two lists: first list is datetimes, second list is pressures
def generateData():
    plungerData = pd.read_csv("../src/static/data/casing.csv")
    times = plungerData.iloc[:, 0].tolist()
    pressures = plungerData.iloc[:, 1].tolist()
    return times, pressures

The data we will be using (casing1.csv) has 2 columns timestamp and pressure. This data has already been refactored. If you wish to look at the non-refactored data, open file static/data/casing_NotRefactored.csv. The dataset is from a gas pump that is slowly failing overtime. Let's plot this data to see what it looks like. Can we see an anomaly in the visible data?

[4]: # plot the dataset to see what it looks like
times, pressures = generateData()
fig = pl.figure()
fig.set_figwidth(10)
fig.set_figheight(4)
pl.scatter(times, pressures)
pl.show()
```

A tooltip message is displayed below the code cell:

We can see from the above plot that the pressure decreases over time and towards the end we observe some pressure

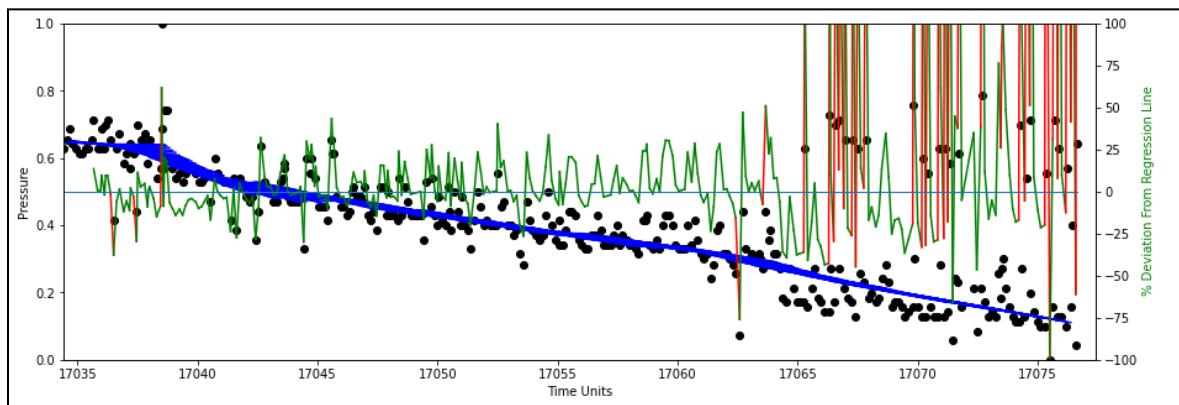
At the bottom of the interface, status information is shown:

Simple 0 2 Initialized (additional servers needed) Python 3 | Idle Mem: 220.45 MB Mode: Command Ln 1, Col 1 02-anomaly-detection-notebook.ipynb

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.



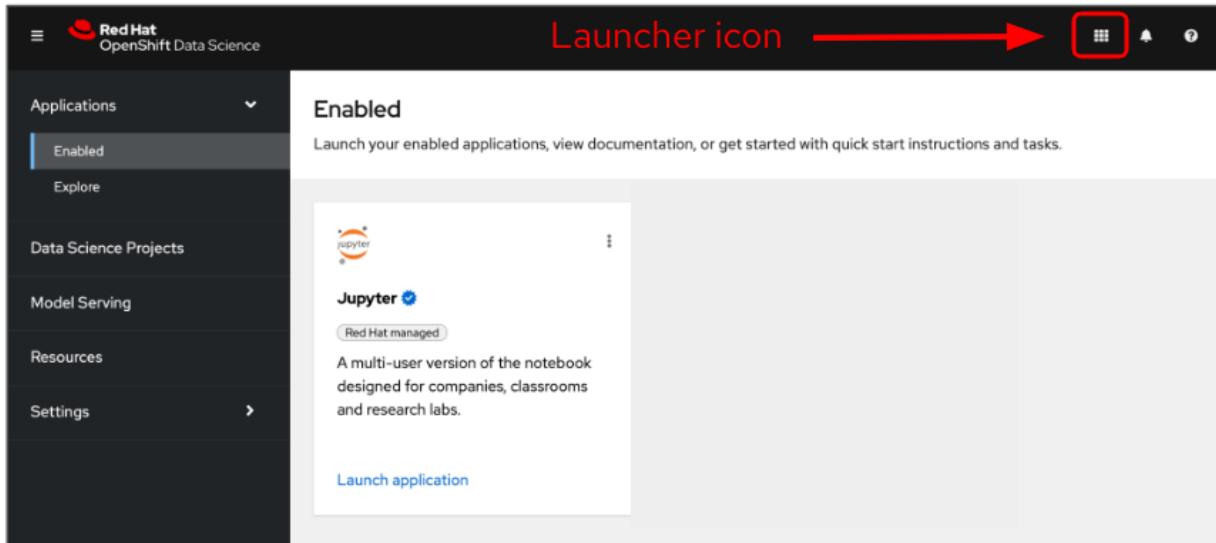
In this plot, the far right hand side of the plot is not only an anomaly, it is also the failure of a mechanical pump. Hypothesis: If we can detect anomalies we can then use this knowledge to predict when a mechanical pump will fail.

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

## Deploy the Anomaly Detection model & web application as a container

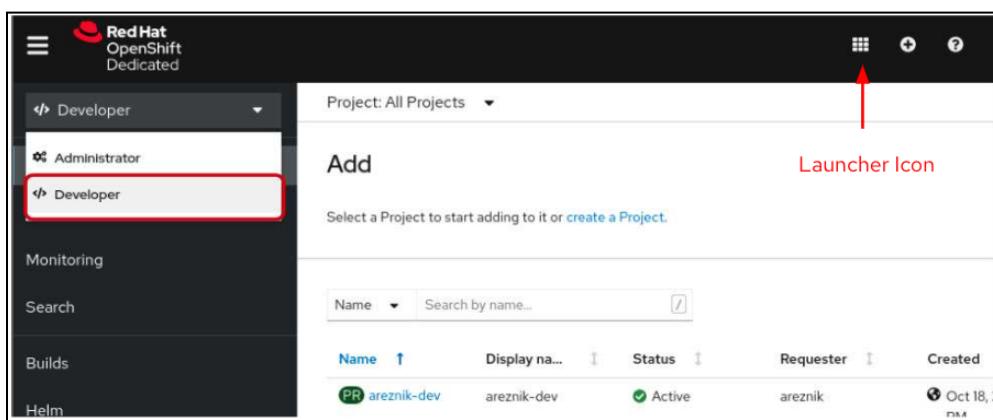
We need to containerize and deploy the Anomaly Detection Web Application code. We will package the code as a container image and run it directly in OpenShift as a Web Application.

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

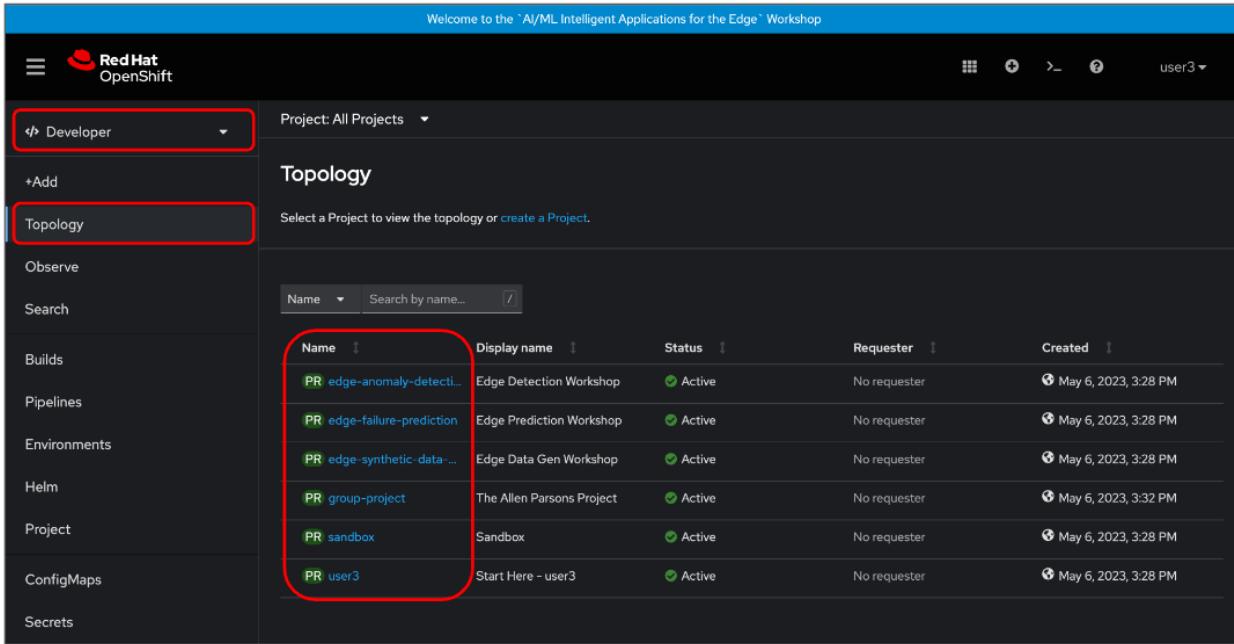


Within the OpenShift console, make certain you are in ‘Developer’ mode.

**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:



Welcome to the 'AI/ML Intelligent Applications for the Edge' Workshop

Project: All Projects ▾

## Topology

Select a Project to view the topology or [create a Project](#).

Name	Display name	Status	Requester	Created
PR edge-anomaly-detect...	Edge Detection Workshop	Active	No requester	May 6, 2023, 3:28 PM
PR edge-failure-prediction	Edge Prediction Workshop	Active	No requester	May 6, 2023, 3:28 PM
PR edge-synthetic-data-...	Edge Data Gen Workshop	Active	No requester	May 6, 2023, 3:28 PM
PR group-project	The Allen Parsons Project	Active	No requester	May 6, 2023, 3:32 PM
PR sandbox	Sandbox	Active	No requester	May 6, 2023, 3:28 PM
PR user3	Start Here - user3	Active	No requester	May 6, 2023, 3:28 PM

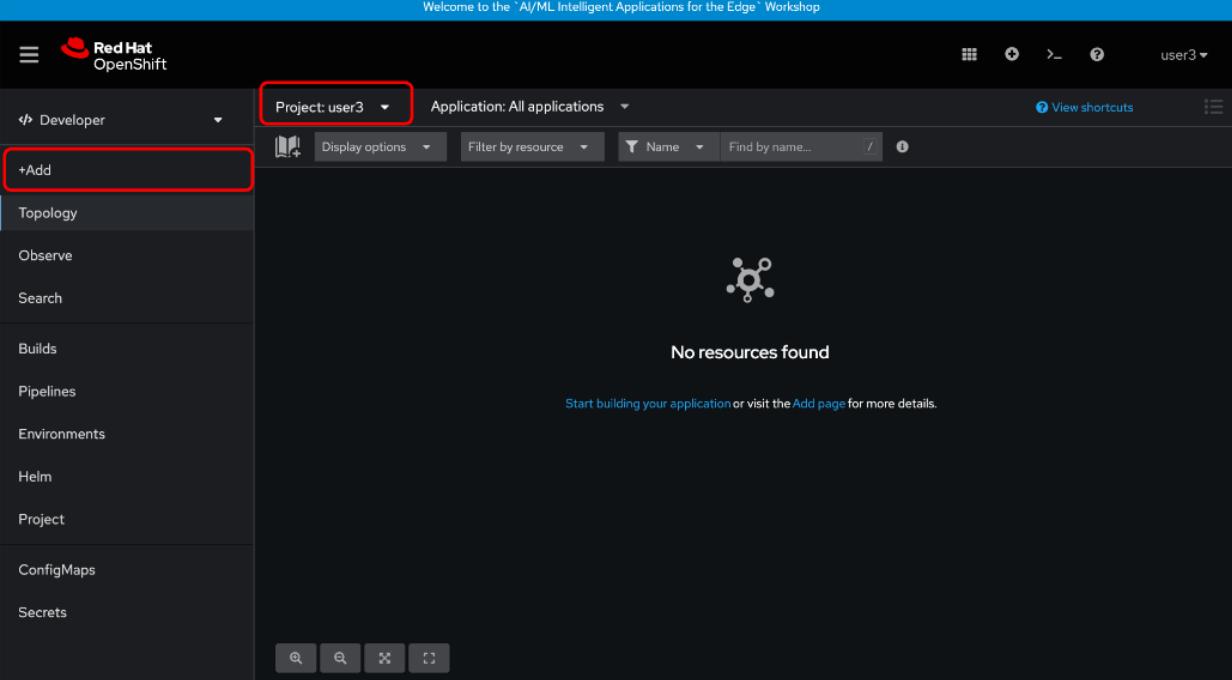
1. Edge-anomaly-detection
2. Edge-failure-prediction
3. Edge-synthetic-data-generator
4. Group-project
5. Sandbox
6. user<#>

You will be working in your project user<#>.

During the workshop, if you do not have time to create: projects edge-anomaly-detection, edge-failure-prediction, and edge-synthetic-data-generator you can open these (pre-created) projects and view the containerized applications.

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 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.



Welcome to the 'AI/ML Intelligent Applications for the Edge' Workshop

Project: user3 Application: All applications

+Add

No resources found

Start building your application or visit the [Add page](#) for more details.

Red Hat OpenShift

Developer

Topology

Observe

Search

Builds

Pipelines

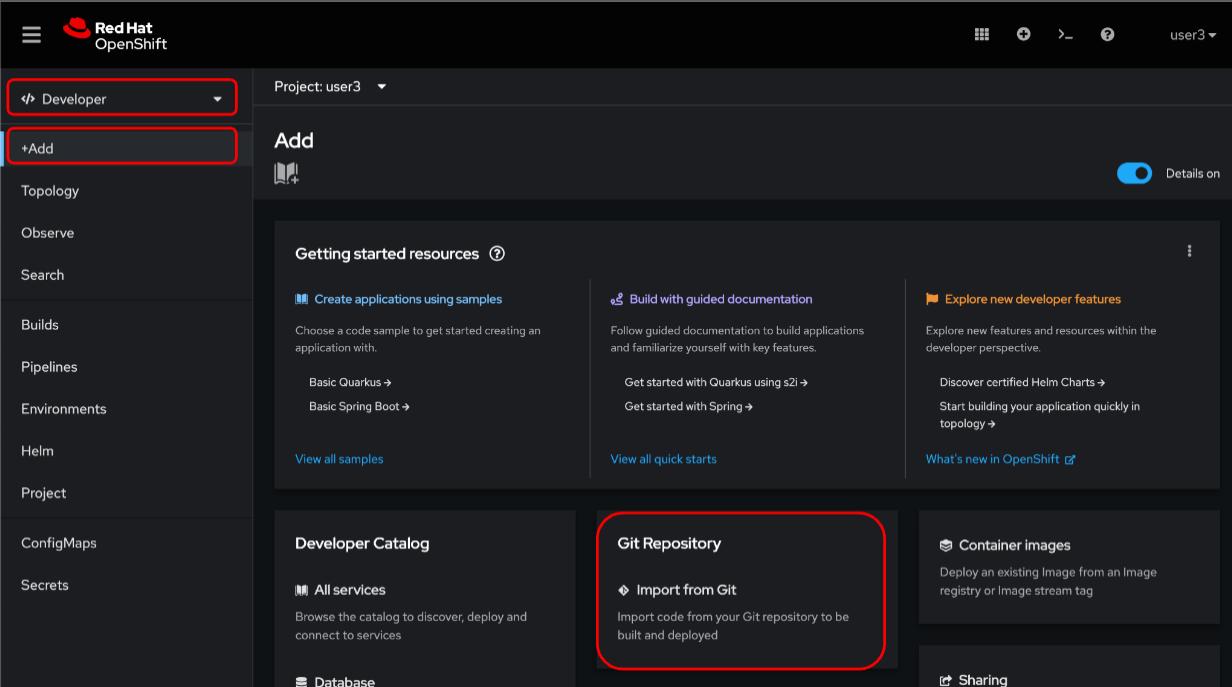
Environments

Helm

Project

ConfigMaps

Secrets



Project: user3

+Add

Add

Getting started resources

Create applications using samples

Choose a code sample to get started creating an application with.

Basic Quarkus →

Basic Spring Boot →

View all samples

Build with guided documentation

Follow guided documentation to build applications and familiarize yourself with key features.

Get started with Quarkus using s2i →

Get started with Spring →

View all quick starts

Explore new developer features

Discover certified Helm Charts →

Start building your application quickly in topology →

What's new in OpenShift ↗

Developer Catalog

All services

Browse the catalog to discover, deploy and connect to services

Database

Git Repository

Import from Git

Import code from your Git repository to be built and deployed

Container images

Deploy an existing Image from an Image registry or Image stream tag

Sharing

Red Hat OpenShift

Developer

Topology

Observe

Search

Builds

Pipelines

Environments

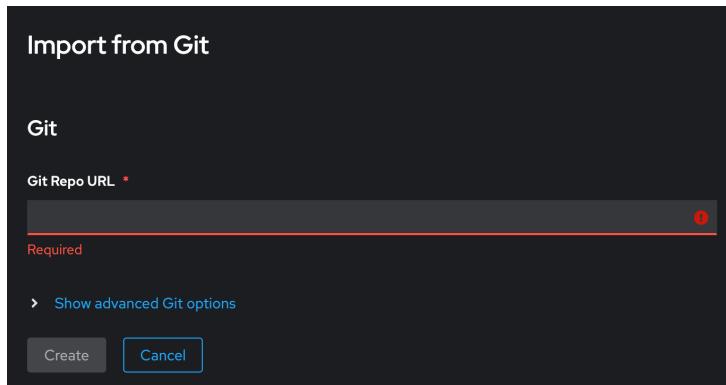
Helm

Project

ConfigMaps

Secrets

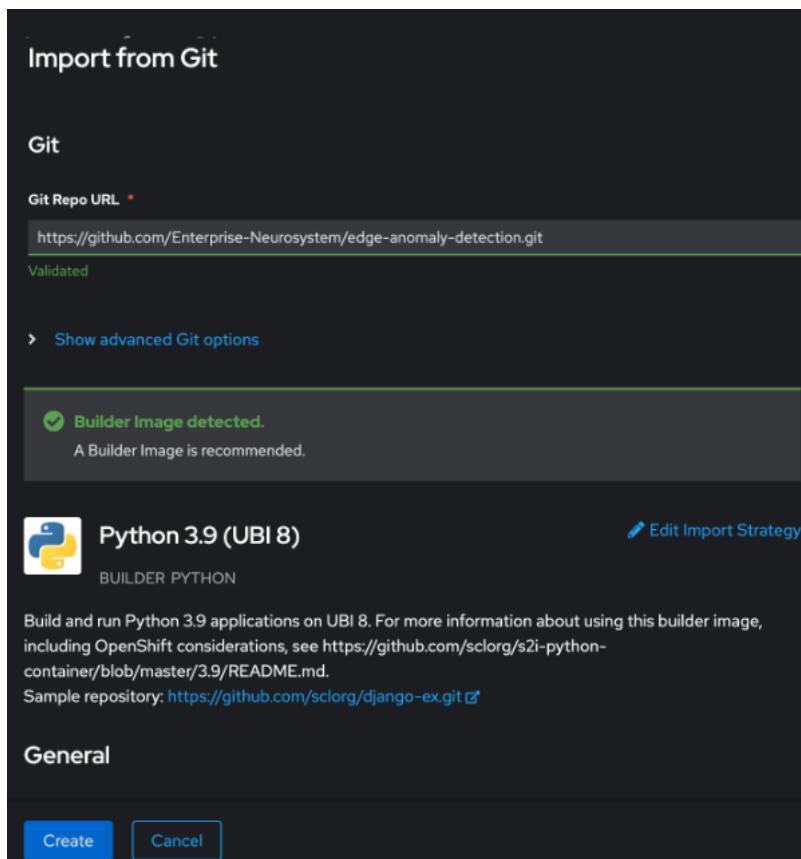
We are going to create a containerized web application. We will be using the source code found in the **edge-anomaly-detection repository** to create the web application. To do this we use the Git Repository, Import from Git option. **Click the ‘Git Repository option.** An **Import from Git** page will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



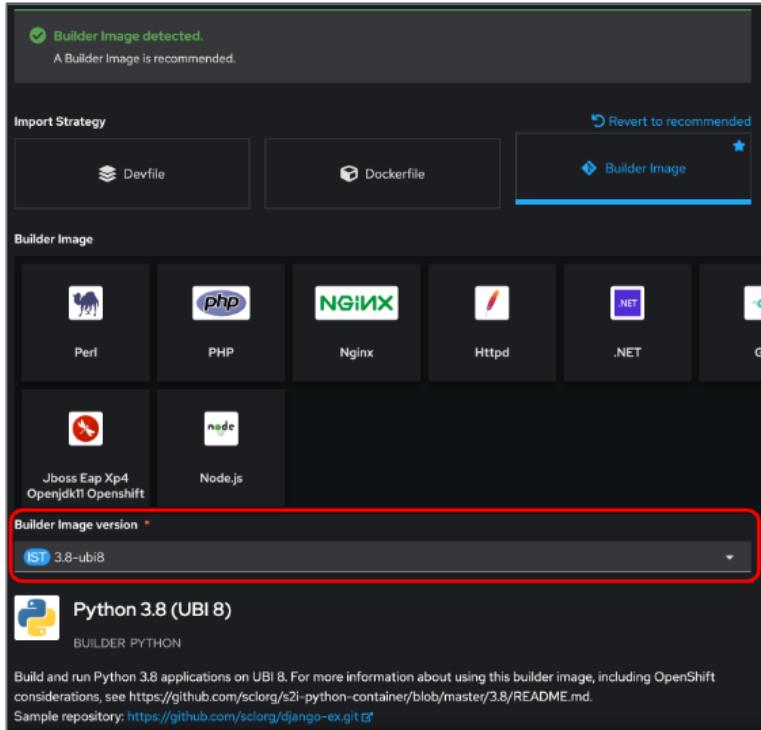
In the Git Repo URL Field, enter:

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

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



We need to change the Builder Image version to Python 3.8 (UBI8). **Click ‘Edit Import Strategy’**, then select 3.8 – ubi8 from the drop down list.



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.

**General**

**Application name**

edge-anomaly-detection-git-app

A unique name given to the application grouping to label your resources.

**Name \***

edge-anomaly-detection-git

A unique name given to the component that will be used to name associated resources.

**Pipelines**

Add pipeline

» Show pipeline visualization

**Advanced options**

**Target port**

8080

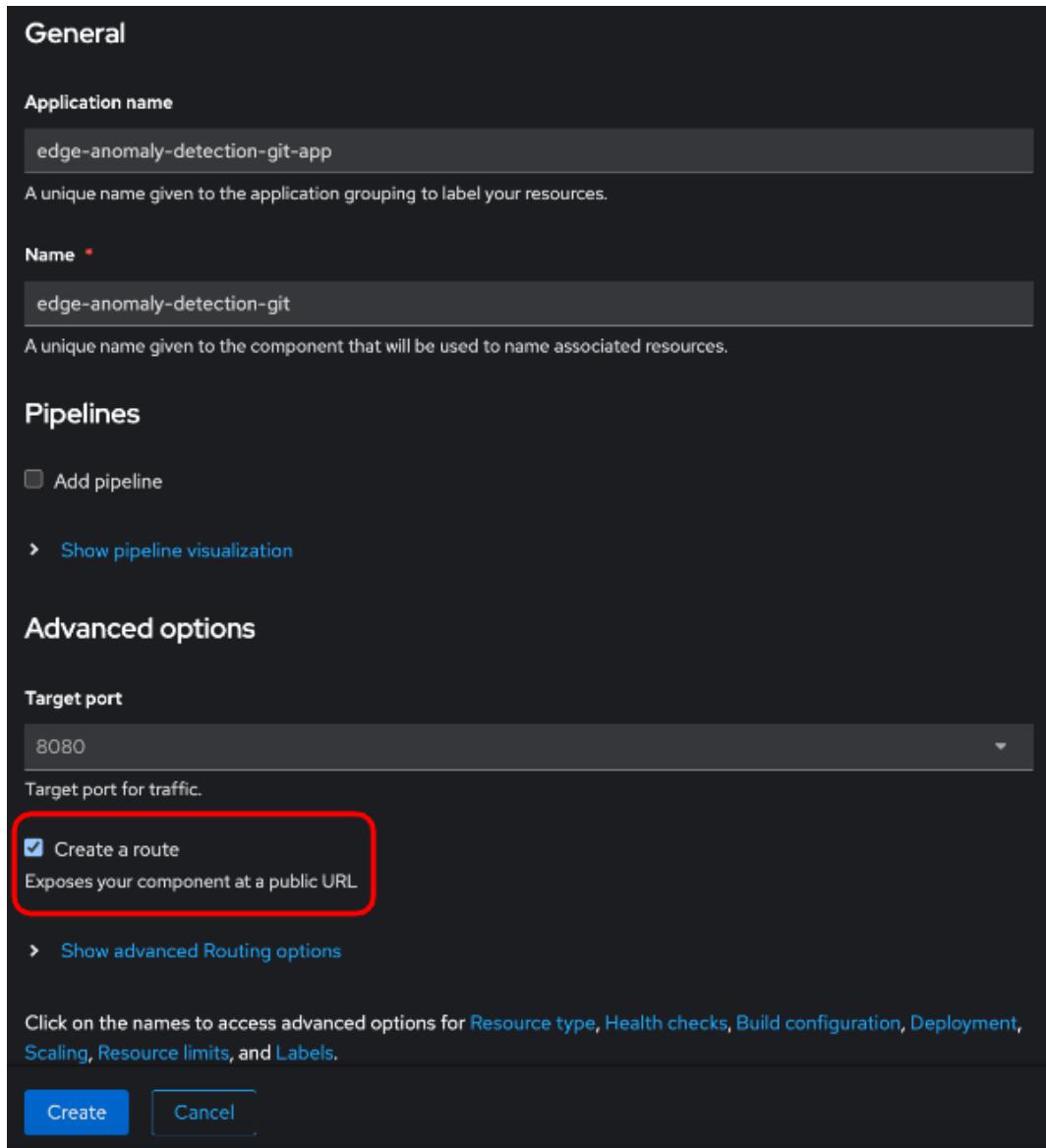
Target port for traffic.

Create a route  
Exposes your component at a public URL

» Show advanced Routing options

Click on the names to access advanced options for [Resource type](#), [Health checks](#), [Build configuration](#), [Deployment](#), [Scaling](#), [Resource limits](#), and [Labels](#).

**Create** **Cancel**



Make certain to name your app. For example, edge-prediction-failure. 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.

A screenshot of the OpenShift application details page for 'edge-anomaly-detection-git'. The left side shows a topology icon with a Python logo inside, and a tooltip says: 'Click inside the topology icon (outer circle) to display the Resources tab'. A red arrow points to the outer circle of the topology icon. Below the icon is a button labeled 'edge-anomaly-detection-git-app'. The right side of the screen displays the application details:

- Health checks:** Container edge-anomaly-detection-git does not have health checks to ensure your application is running correctly. [Add health checks](#)
- Resources:** This tab is highlighted with a red box.
- Pods:** Status: Waiting for the build. Message: Waiting for the first build to run successfully. You may temporarily see "ImagePullBackOff" and "ErrImagePull" errors while waiting. [Show waiting pods with errors](#). Result: No Pods found for this resource.
- Builds:** Status: BC edge-anomaly-detection-git. [Start Build](#). Result: Build #1 is running (Just now). [View logs](#).
- Services:** Status: S edge-anomaly-detection-git. Service port: 8080-tcp → Pod port: 8080.

Before we view the containerized application, we first have to annotate the routes to increase the time out. Switch the view from 'Developer' to 'Administrator'.

Navigate to 'Networking' and then 'Routes' on the side bar.

Administrator

Routes

Name Status Location

edge-anomaly-detection-git Accepted https://edge-anomaly-detection-git-edge-anomaly-detection.apps.cluster-vmx62.sandbox2256.opentlc.com

Click the url to edit route

Click on the project name ‘edge-anomaly-detection’ to add an annotation. An ‘Edit annotations’ dialog box appears

Project: edge-anomaly-detection

Routes

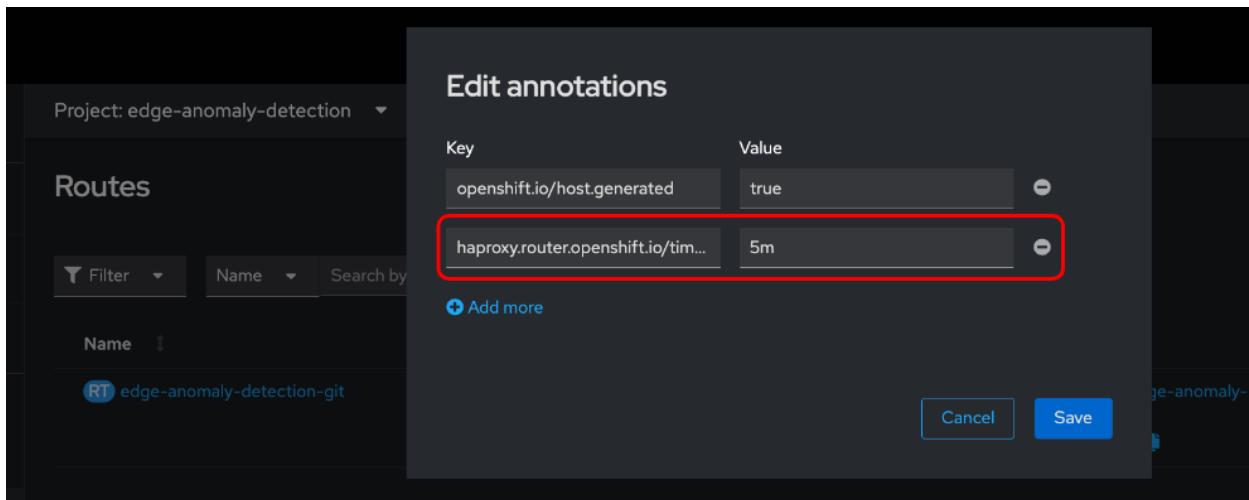
openshift.io/host.generated true

+ Add more

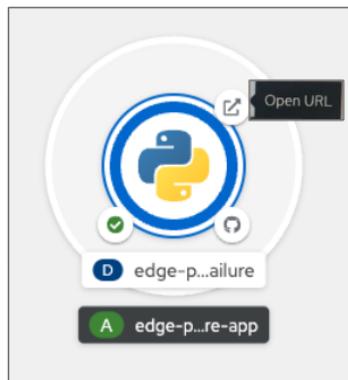
Cancel Save

Click on the **+Add more** link to edit the annotations and then add the following key-value pair:  
[haproxy.router.openshift.io/timeout](#), 5m

After you have added the annotation, click 'Save'. We add this annotation in order to increase the timeout so that our web application does not timeout before our linear regression finishes.

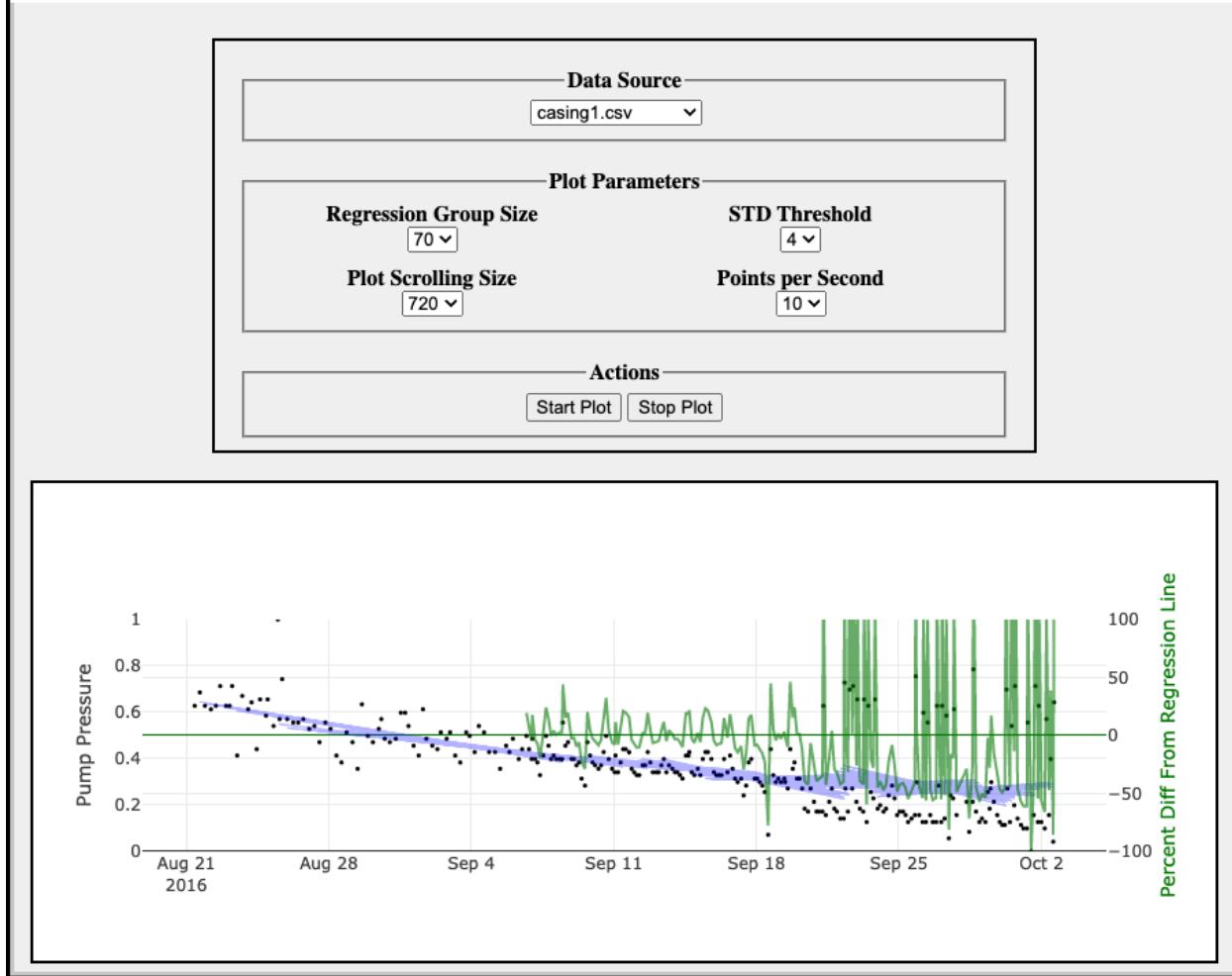


Now we can head back to our 'Developer' view and display our containerized application in a browser by clicking on the URL icon in the Topology view.



Your containerized Anomaly Detection application will now appear in a browser window.

## Anomaly Detection



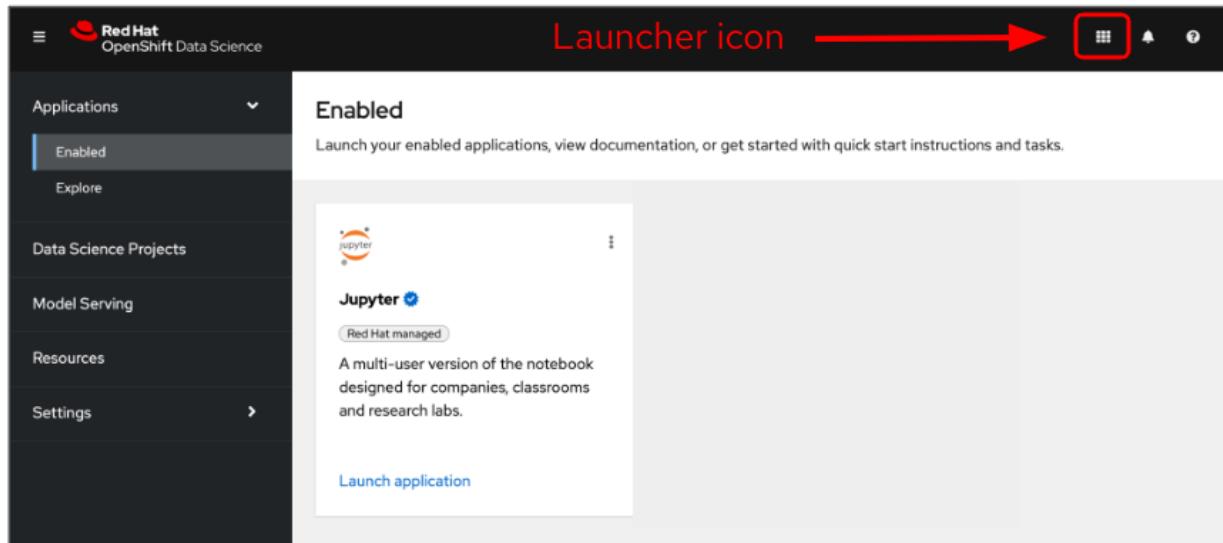
Try re-running the application with different plot parameters to see what happens.

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](#).

# Failure Prediction Web Application

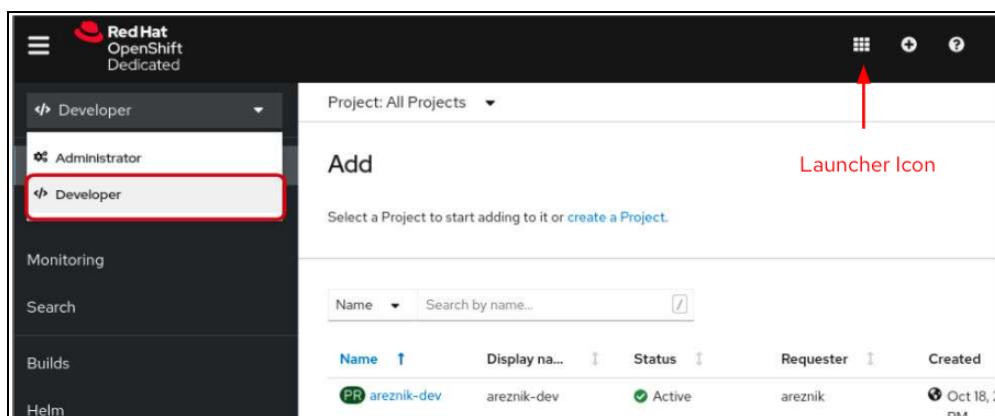
We need to containerize and deploy the Failure Prediction Web Application code. We will package the code as a container image and run it directly in OpenShift as a Web Application.

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

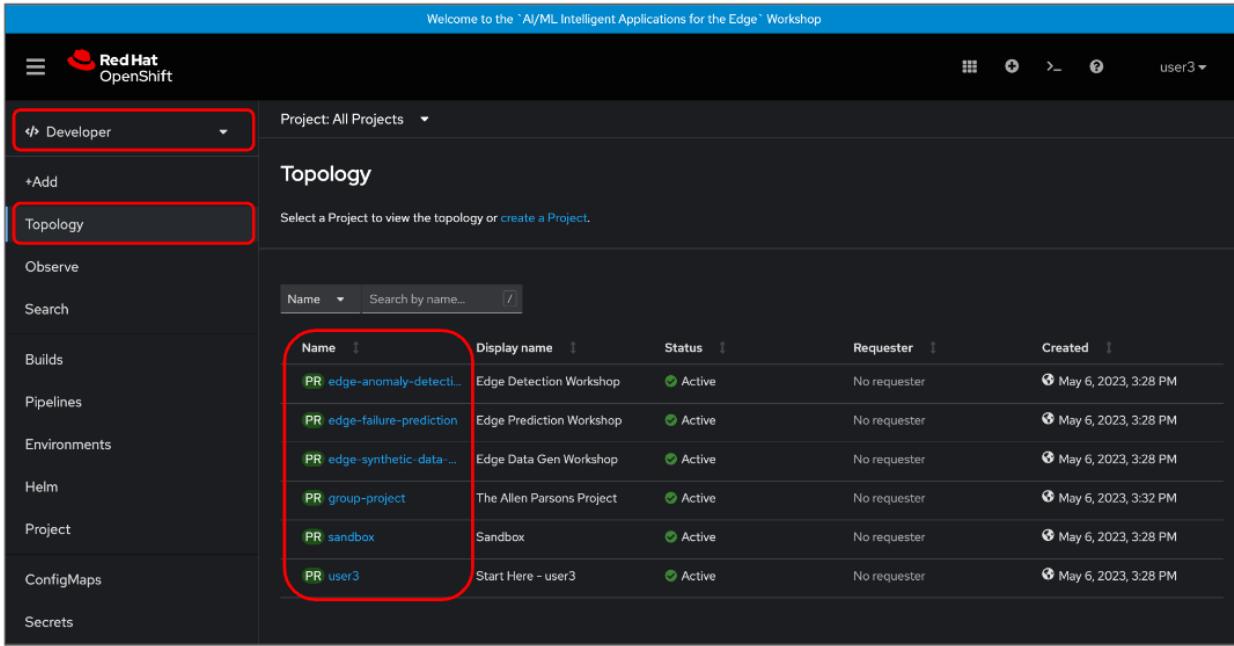


Within the OpenShift console, make certain you are in ‘Developer’ mode.

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:



Welcome to the 'AI/ML Intelligent Applications for the Edge' Workshop

Project: All Projects ▾

## Topology

Select a Project to view the topology or [create a Project](#).

Name	Display name	Status	Requester	Created
PR edge-anomaly-detect...	Edge Detection Workshop	Active	No requester	May 6, 2023, 3:28 PM
PR edge-failure-prediction	Edge Prediction Workshop	Active	No requester	May 6, 2023, 3:28 PM
PR edge-synthetic-data-...	Edge Data Gen Workshop	Active	No requester	May 6, 2023, 3:28 PM
PR group-project	The Allen Parsons Project	Active	No requester	May 6, 2023, 3:32 PM
PR sandbox	Sandbox	Active	No requester	May 6, 2023, 3:28 PM
PR user3	Start Here - user3	Active	No requester	May 6, 2023, 3:28 PM

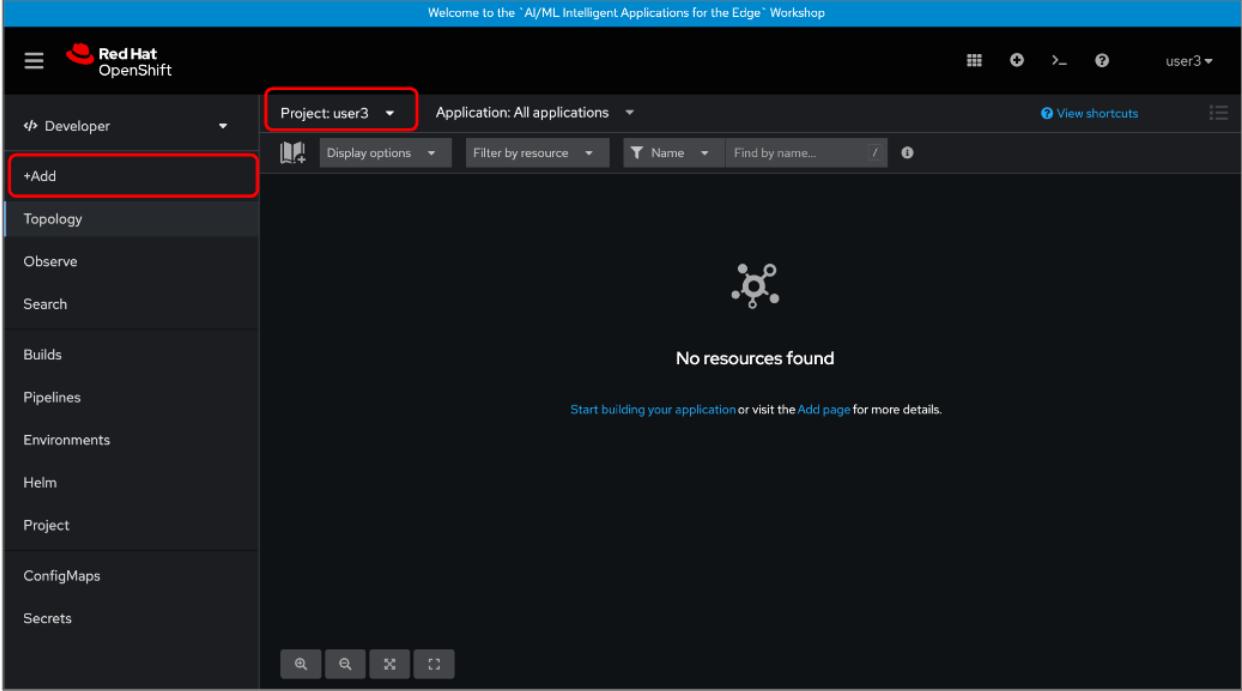
7. Edge-anomaly-detection
8. Edge-failure-prediction
9. Edge-synthetic-data-generator
10. Group-project
11. Sandbox
12. user<#>

You will be working in your project user<#>.

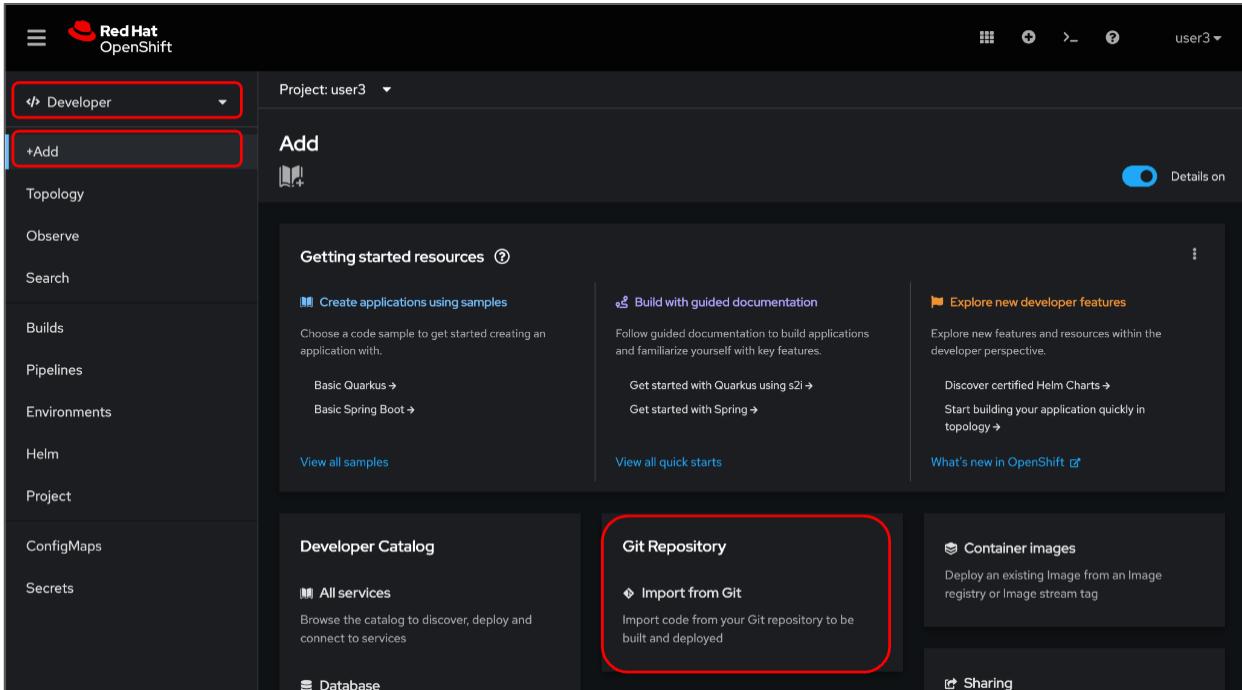
During the workshop, if you do not have time to create: projects edge-anomaly-detection, edge-failure-prediction, and edge-synthetic-data-generator you can open these (pre-created) projects and view the containerized applications.

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<#>) you can begin the process to create your container. Click the +Add menu. A number of options will appear.

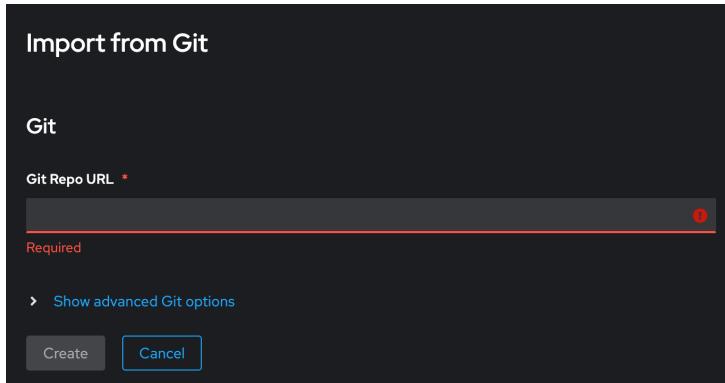


The screenshot shows the Red Hat OpenShift Developer interface. The left sidebar lists various developer tools: Topology, Observe, Search, Builds, Pipelines, Environments, Helm, Project, ConfigMaps, and Secrets. The main area is titled "Welcome to the 'AI/ML Intelligent Applications for the Edge' Workshop". It shows the "Project: user3" dropdown menu, which is highlighted with a red box. Below it, the "Application: All applications" dropdown and a search bar are visible. The central content area displays a network graph icon and the message "No resources found". A small note at the bottom says "Start building your application or visit the [Add page](#) for more details." At the bottom of the screen are several navigation icons.

This screenshot shows the "Add" page within the Red Hat OpenShift Developer interface. The left sidebar remains the same as the previous screenshot. The main area has a "Getting started resources" section with links to "Create applications using samples", "Build with guided documentation", and "Explore new developer features". Below this is a "Developer Catalog" section with "All services" and "Database" options. The "Git Repository" option is highlighted with a red box. It includes a sub-section "Import from Git" with the description "Import code from your Git repository to be built and deployed". To the right are sections for "Container images" and "Sharing". A "Details on" toggle switch is visible in the top right corner of the main content area.

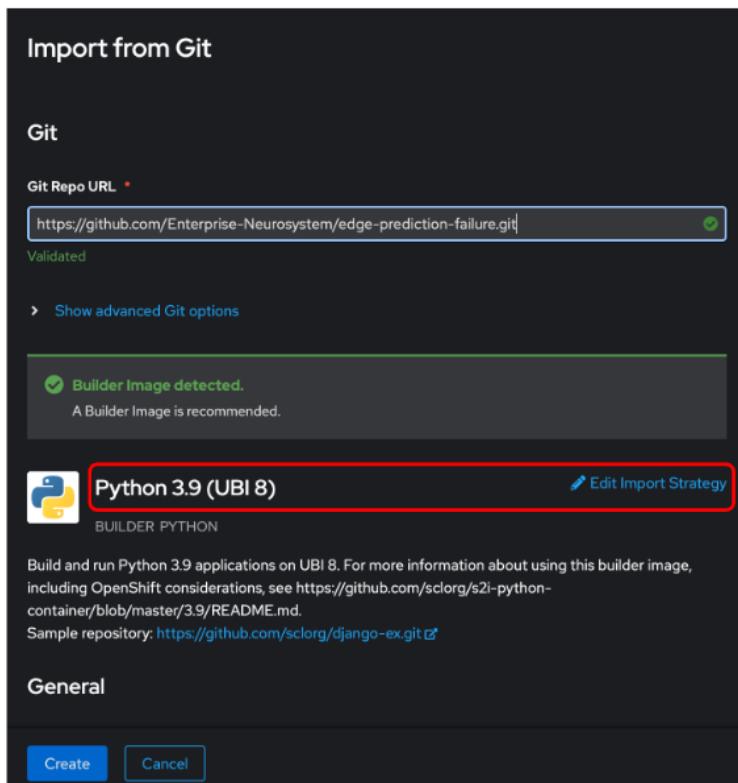
We are going to create a containerized web application. We will be using the source code found in the edge-failure-prediction repository to create the web application. To do this we use the Git Repository, Import from Git option. **Click the ‘Git Repository option.** An **Import from Git** page will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



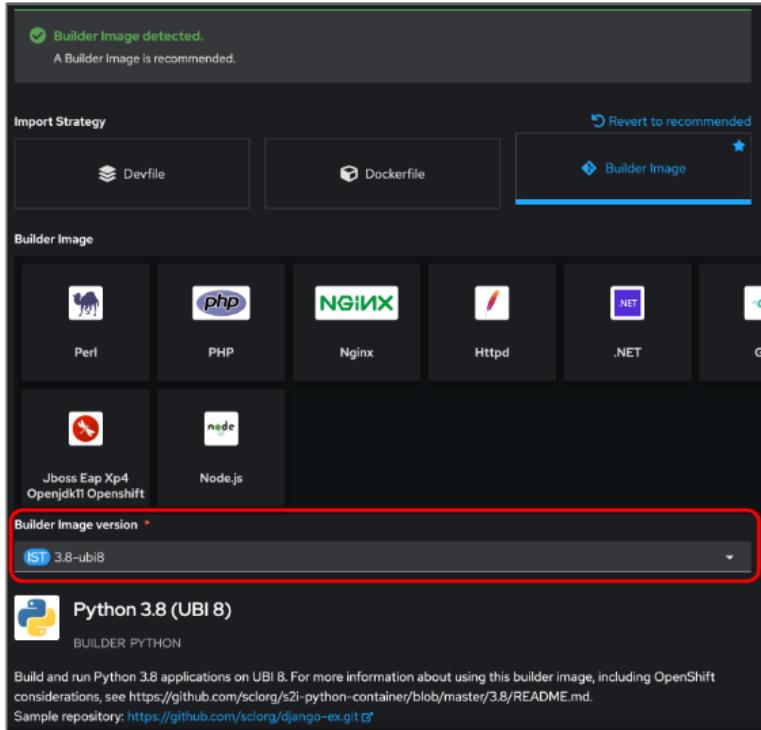
In the Git Repo URL Field, enter:

<https://github.com/Enterprise-Neurosystem/edge-failure-prediction.git>

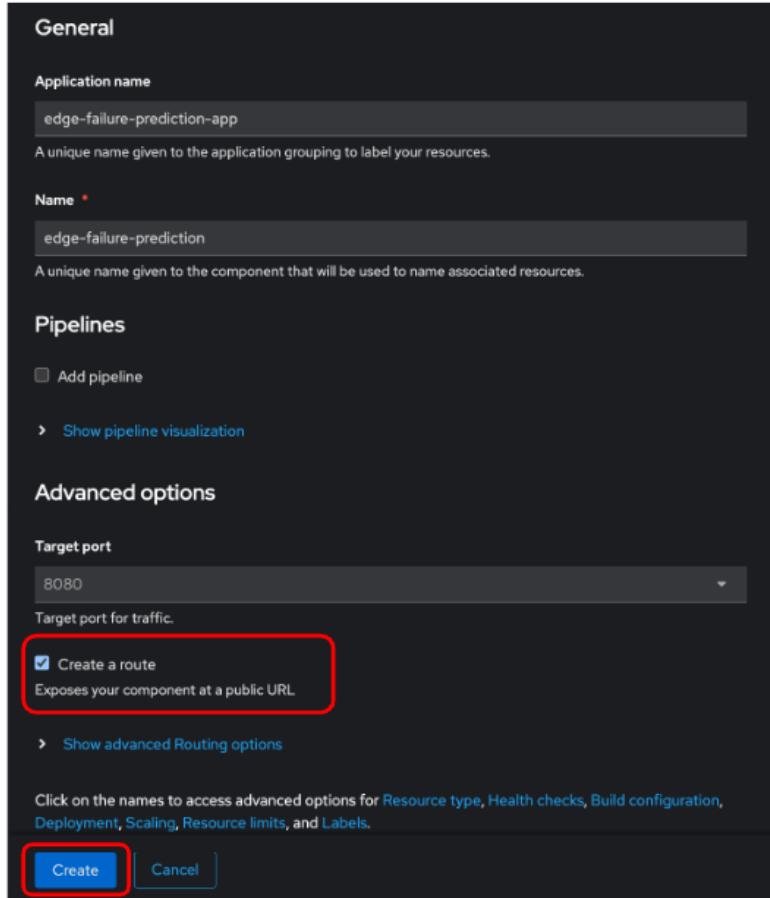
Note that the Git Repo URL will be validated. Once validated, further options are available.



We need to change the Builder Image version to Python 3.8 (UBI8). **Click ‘Edit Import Strategy’**, then select 3.8 – ubi8 from the drop down list.

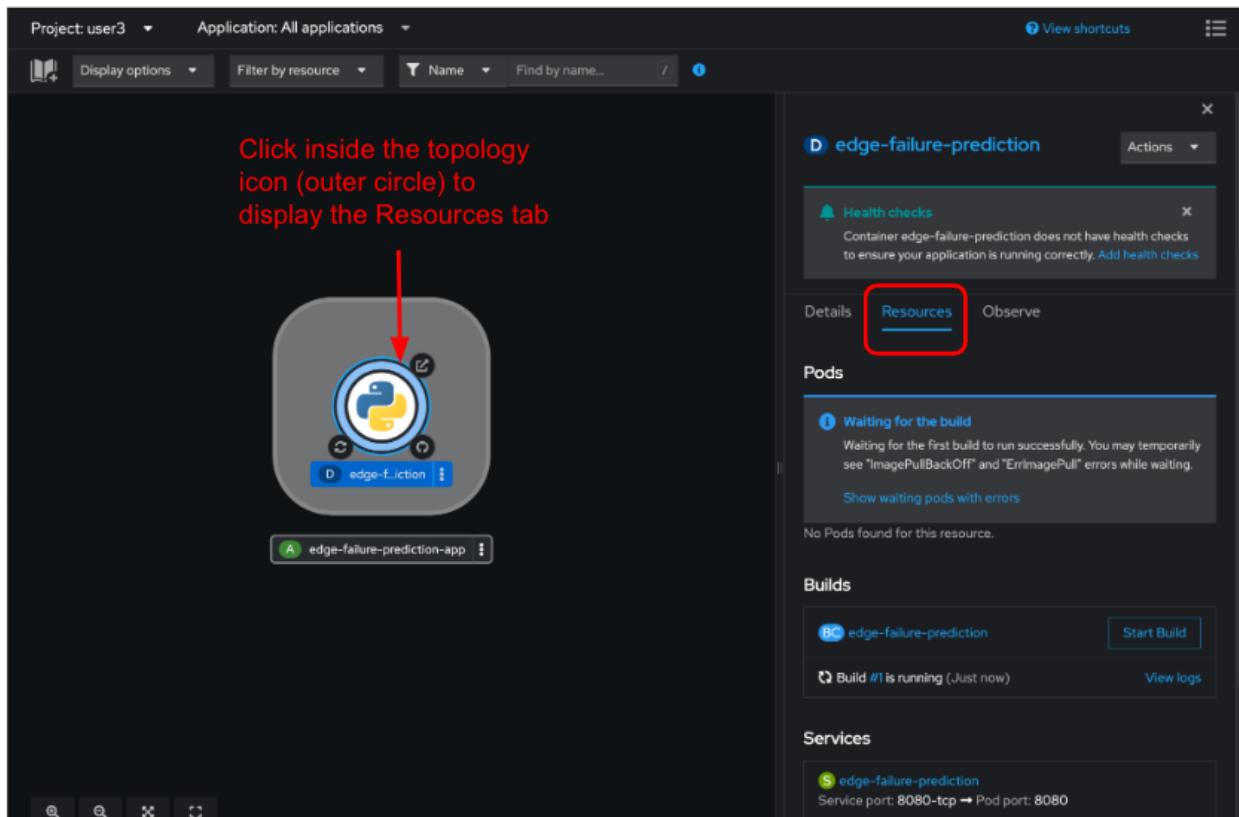


If you continue to scroll down the page, you will see that everything is automatically selected to create a deployment of your application, as well as a route through which you will be able to access the application.



Make certain to name your app. For example, edge-prediction-failure. 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.



Before we view the containerized application, we first have to annotate the routes to increase the time out. Switch the view from ‘Developer’ to ‘Administrator’.

The screenshot shows the Red Hat OpenShift Service on AWS web interface. The left sidebar is titled 'Developer' and lists various resources: Administrator, Developer, Observe, Search, Builds, Environments, Helm, Project, ConfigMaps, and Secrets. The 'Developer' item is highlighted with a red dashed box. The main content area shows an application named 'edge-prediction-failure-git'. At the top right of the application card, there is a 'Health checks' section with a warning message: 'Container edge-prediction-failure-git does not have health checks to ensure your application is running correctly.' Below this, there are tabs for 'Details', 'Resources' (which is selected), and 'Observe'. Under the 'Resources' tab, there is a 'Pods' section showing one pod named 'edge-prediction-failure-git-cc5dff679-67fd'. To the right of the pod card are status indicators: 'Running' and 'View logs'. Below the pods section is a 'Builds' section with a build named 'edge-prediction-failure-git' and a 'Start Build' button.

Navigate to 'Networking' and then 'Routes' on the side bar.

The screenshot shows the Red Hat OpenShift Service on AWS dashboard. On the left, there's a sidebar with navigation links: Home, Operators, Workloads, Networking, Services, Routes (which is highlighted with a red dashed box), Ingresses, NetworkPolicies, Storage, and Builds. The main area displays a table of resources with columns for Name, Status (all Active), Requester (No requester), and Created date (various times from Aug 22 to 23, 2022). The 'Routes' row has a timestamp of Aug 22, 2022, 10:28 PM.

Name	Status	Requester	Created
Workloads	Active	No requester	Aug 23, 2022, 4:26 PM
Networking	Active	No requester	Aug 22, 2022, 10:28 PM
Services	Active	No requester	Aug 22, 2022, 10:28 PM
Routes	Active	No requester	Aug 22, 2022, 10:28 PM
Ingresses	Active	No requester	Aug 22, 2022, 10:28 PM
NetworkPolicies	Active	No requester	Aug 22, 2022, 10:28 PM
Storage	Active	No requester	Aug 22, 2022, 10:28 PM
Builds	Active	No requester	Aug 22, 2022, 10:28 PM

Click on the project name ‘edge-prediction-failure’ and then scroll down to ‘Annotations’.

The screenshot shows the 'Routes' page for the 'edge-prediction-failure' project. At the top, it says 'Project: user08'. Below that, the title 'Routes' is displayed, along with a 'Create Route' button. The main area shows a table with one route entry:

Name	Location
edge-prediction-failure-git	<a href="https://edge-prediction-failure-git-user08.apps.ieee.d7se.p1.openshiftapps.com">https://edge-prediction-failure-git-user08.apps.ieee.d7se.p1.openshiftapps.com</a>

Click on the pencil icon to edit the annotations and then add the following key-value pair:

[haproxy.router.openshift.io/timeout](#), **5m** and click on 'Save'. We are doing this to increase the timeout so that our web application does not timeout before the model is fully trained.

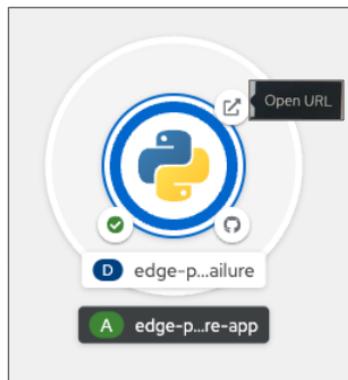
The screenshot shows the 'Edit annotations' dialog box from the Red Hat OpenShift interface. The dialog is centered over a larger configuration page for a service named 'edge-prediction-failure-g'. The annotations section of the main page is visible in the background, showing two existing annotations: 'app.ope...' and 'openshift.io/host.generated'. The 'Edit annotations' dialog contains two entries:

Key	Value
haproxy.router.openshift.io/tim...	5m
openshift.io/host.generated	true

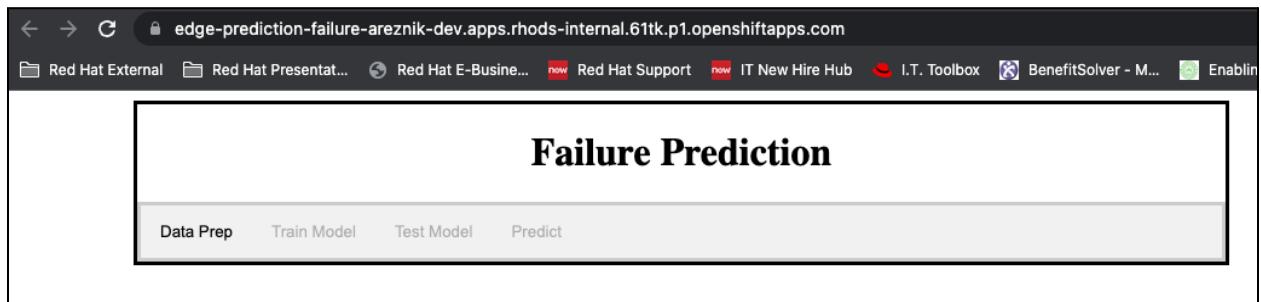
Below the table are 'Cancel' and 'Save' buttons. The 'Save' button is highlighted with a blue border, indicating it is the next action to be taken. The dialog has a dark header bar with the Red Hat OpenShift logo and a user dropdown.

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.

The screenshot shows the Red Hat OpenShift Service on AWS developer interface. On the left, there's a sidebar with navigation options: Administrator, Workloads, Networking, Services, Routes, Ingresses, NetworkPolicies, Storage, Builds, and User Management. The 'Administrator' option is currently selected. The main area displays a table of services. One service, 'edge-prediction-failure-git', is listed with the status 'Accepted'. Its location is 'https://edge-prediction-failure-git-user03.apps.ieee.d7se.p1.openshiftapps.com'. To the right of the URL, there are icons for opening the URL in a browser and for viewing logs. A blue 'Create Route' button is located in the top right corner of the main area.



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



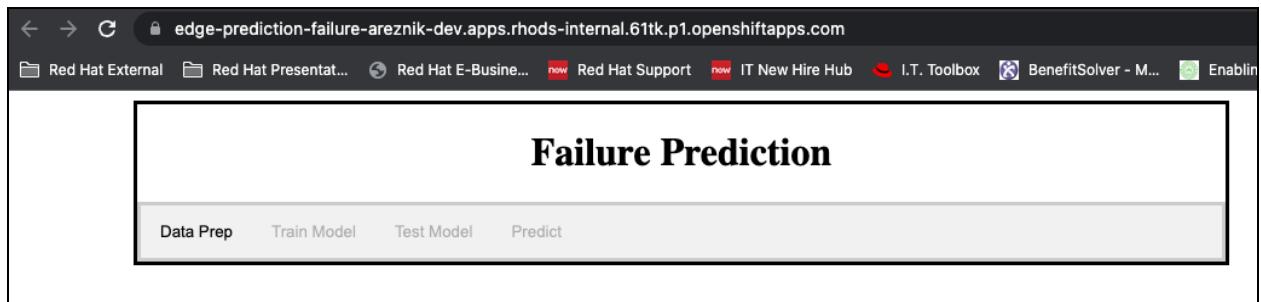
This application serves as a wizard that allows the user to:

- 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.

## 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 screenshot shows a web-based application titled "Failure Prediction". At the top, there is a navigation bar with four tabs: "Data Prep" (which is highlighted in purple), "Train Model", "Test Model", and "Predict". Below the navigation bar, the main content area is titled "Data Preparation". Within this area, there is a box labeled "Data Preparation Action" containing two buttons: "Start Data Prep" and "Stop Data Prep". A progress bar is shown below these buttons. The text "Loading data....Takes about 20 sec" is displayed at the bottom of the "Data Preparation" section. The entire application is enclosed in a black border.

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 screenshot shows the 'Failure Prediction' application interface. At the top, there is a navigation bar with tabs: 'Data Prep' (which is selected and highlighted in blue), 'Train Model', 'Test Model', and 'Predict'. Below the navigation bar is a section titled 'Data Preparation' containing a 'Data Preparation Action' box with 'Start Data Prep' and 'Stop Data Prep' buttons. A message below the buttons says 'Data is loaded. Press 'Start Data Prep' Btn'. To the right of this is a 'Feature Choices' box divided into three sections: 'Sensor Name Sum Nulls' (listing sensor names and their null counts), 'Dropped Sensors' (listing sensors that will be dropped), and 'PC In Model' (listing principal component names). The 'Sensor Name Sum Nulls' section data is as follows:

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

The 'Dropped Sensors' section data is:

Dropped Sensors
sensor_00
sensor_15
sensor_50
sensor_51

The 'PC In Model' section data is:

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%

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:

## Failure Prediction

Data Prep    Train Model    Test Model    Predict

### Data Preparation

Data Preparation Action

Start Data Prep
Stop Data Prep

30%

**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

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%

Dropped Sensors
sensor_00
sensor_15
sensor_50
sensor_51

PC In Model
pc1
pc2
pc3
pc4
pc5
pc6
pc7
pc8

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

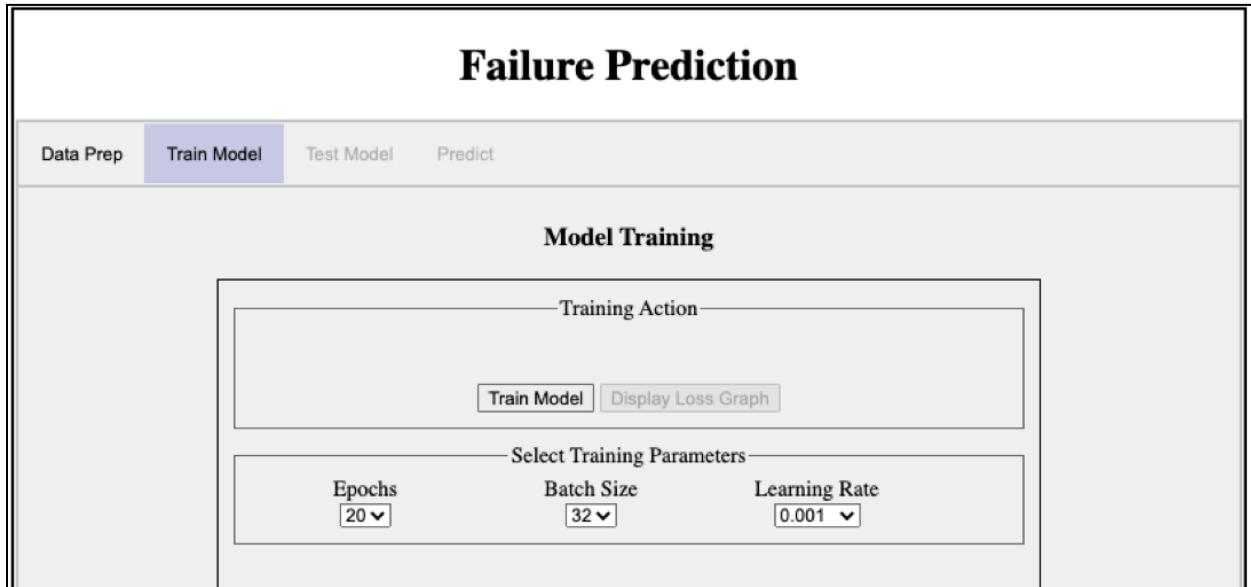
Dropped Sensors
sensor_00
sensor_15
sensor_50
sensor_51

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.

## Failure Prediction

Data Prep    **Train Model**    Test Model    Predict

### Model Training

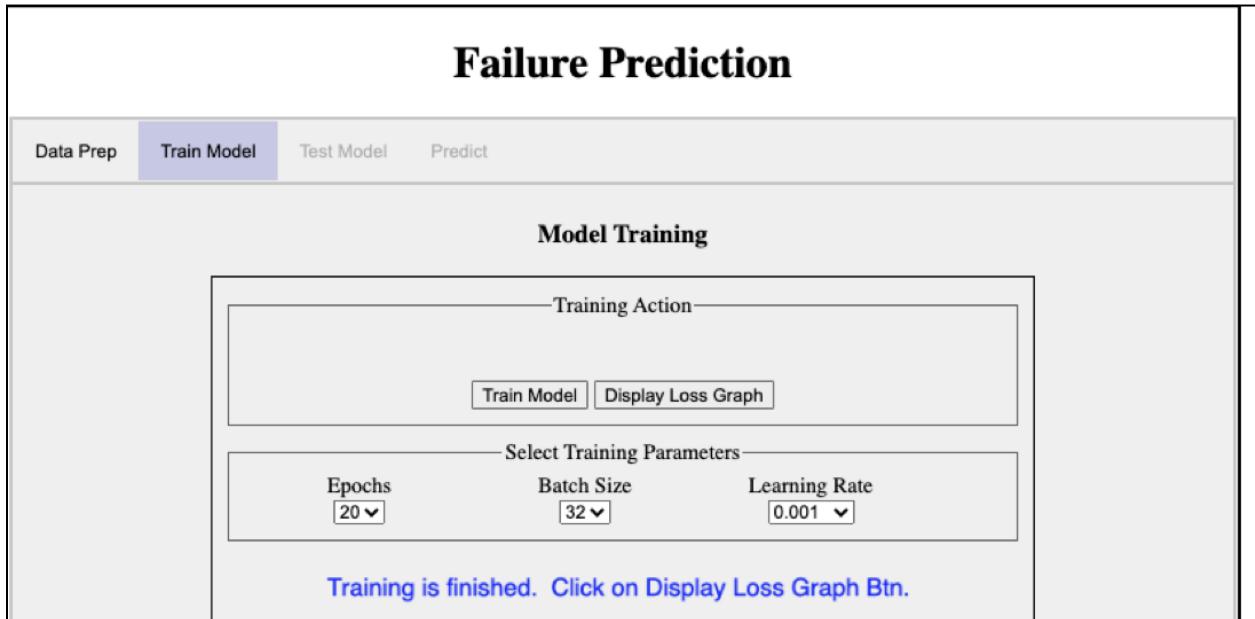
Training Action

Select Training Parameters

Epochs <input type="button" value="20 ▾"/>	Batch Size <input type="button" value="32 ▾"/>	Learning Rate <input type="button" value="0.001 ▾"/>
---	---	---

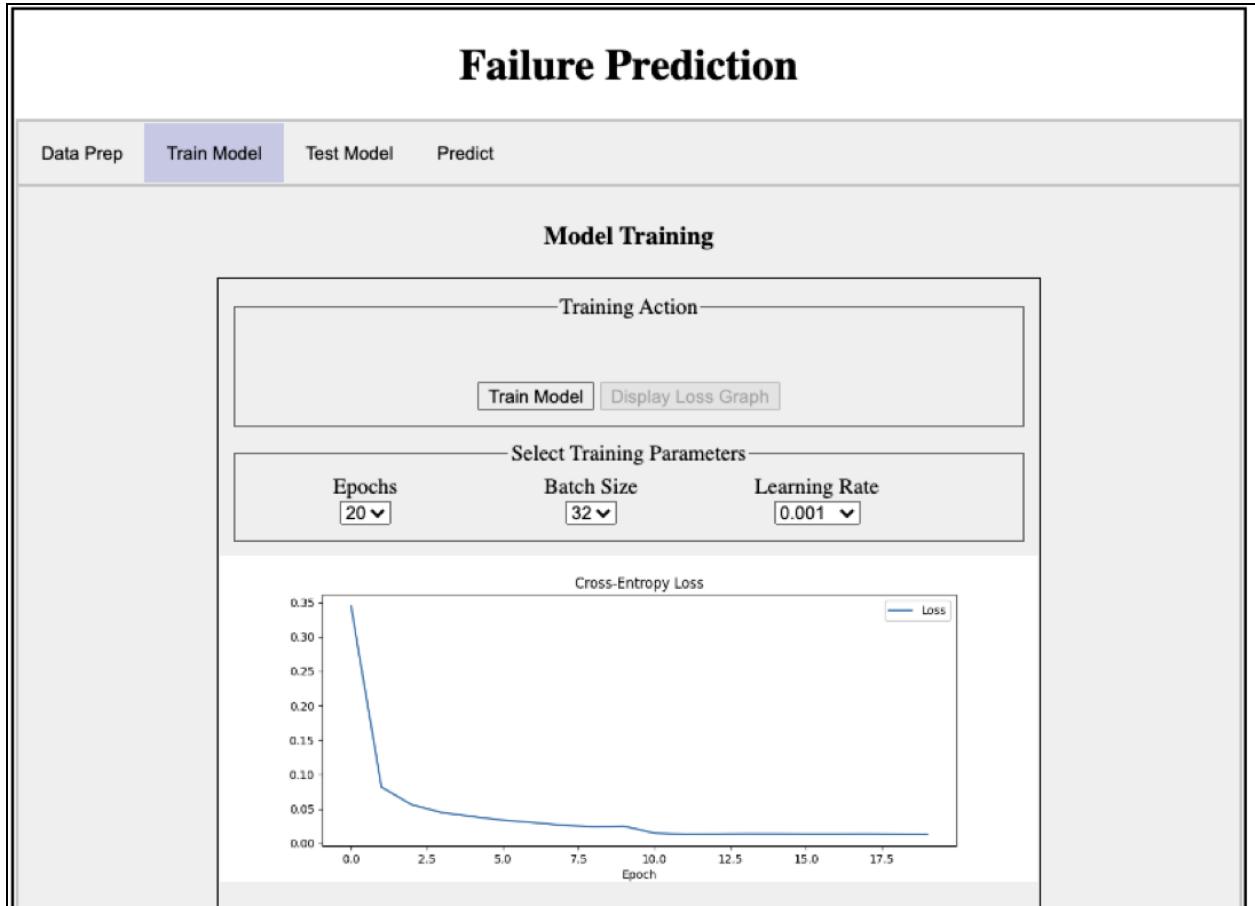
Working...For 20 Epochs expect about 40 seconds

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:



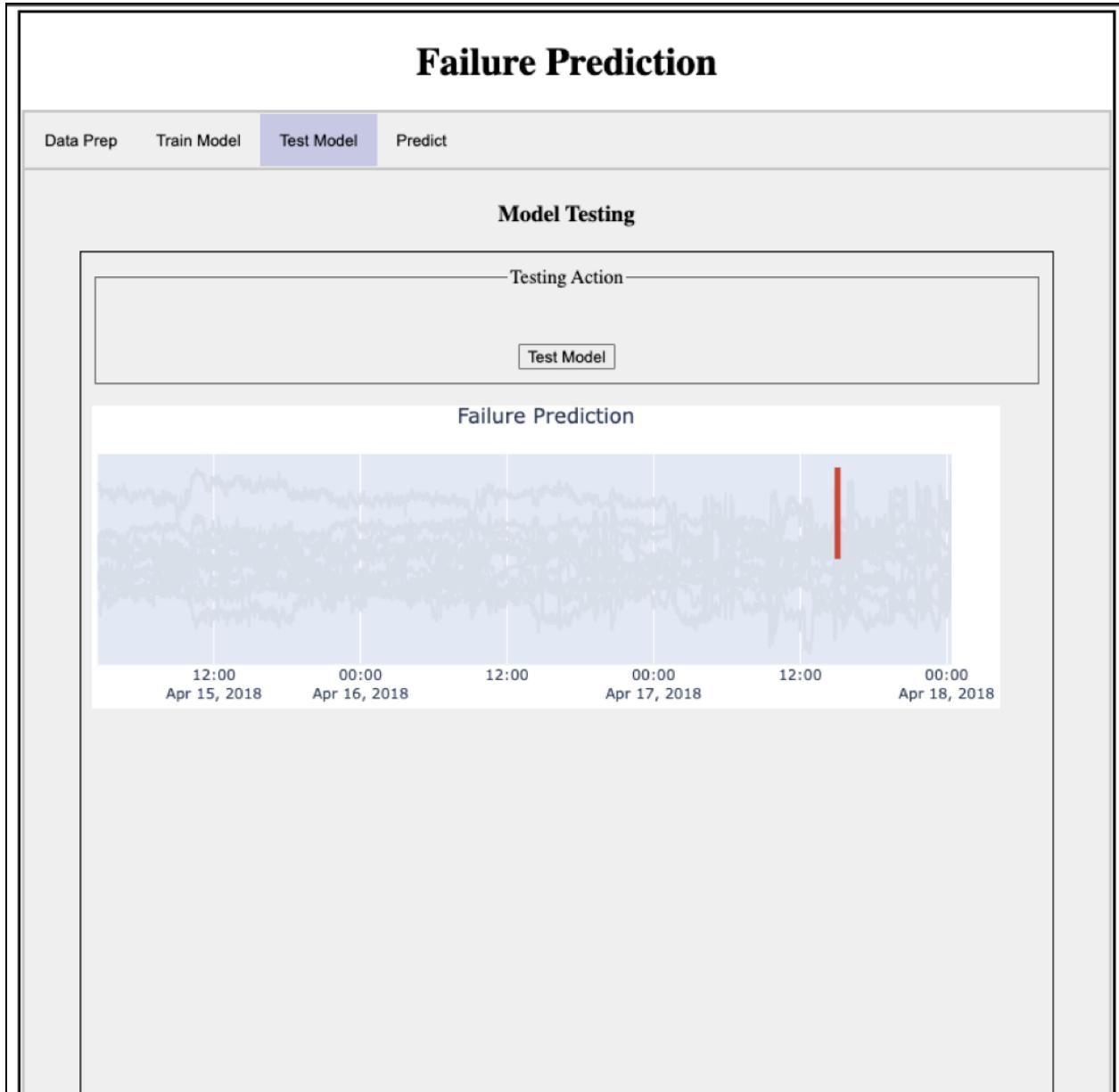
Besides the graph you will notice that the two tabs labeled Test Model and Predict have been enabled. We will run the Test first, although at this point we could just as well run a prediction.

### 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.

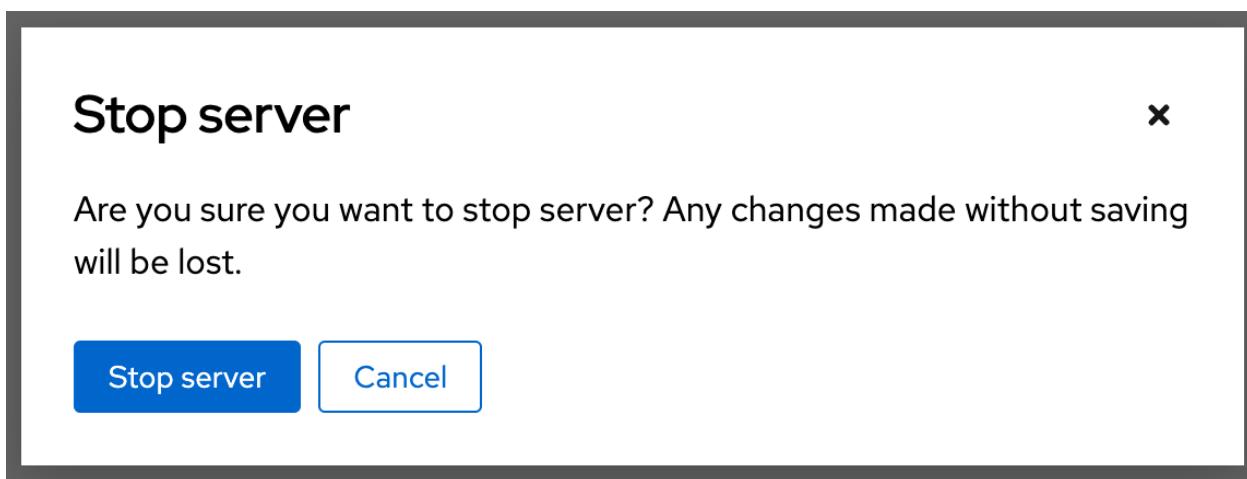
Before we can look at predicting mechanical pump failures in real time, we need to have real time mechanical pump sensor data to work with. For this workshop, we will not be connecting to a real time mechanical pump. Therefore; to simulate real time mechanical pump sensor data, we created a synthetic 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](#).

## 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 Stop server dialog will appear. Choose 'Stop server' option.



The 'Start a Notebook Server' options page will appear. Within this page, choose the Notebook image 'PyTorch-py3-cuda', a 'small' container size. Then press start server to create a new JupyterLab environment which is ready for you to start using an environment with Python 3.8 and PyTorch.

## Start a notebook server

Select options for your notebook server.

### Notebook image

- VS Code Server v4.9.1 ⓘ  
Coder Code Server v4.9.1  
» [Versions](#)
- Standard Data Science py3.9-v2 ⓘ  
Python v3.9  
» [Versions](#)
- PyTorch py3.8-cuda-11.4.2-2 ⓘ  
Python v3.8, PyTorch v1.8  
▼ [Versions](#)
  - Version py3.9-v2 ⓘ ★ Recommended  
Python v3.9, PyTorch v1.13
  - Version py3.8-cuda-11.4.2-2 ⓘ  
Python v3.8, PyTorch v1.8
- TrustyAI ⓘ  
Python v3.9
- Minimal Python py3.9-v2 ⓘ  
Python v3.9  
» [Versions](#)
- CUDA py3.9-v2-cuda-11.8.0 ⓘ  
Python v3.9  
» [Versions](#)
- TensorFlow py3.9-v2-cuda-11.8.0 ⓘ  
Python v3.9, TensorFlow v2.11  
» [Versions](#)

### Deployment size

#### Container Size

Small



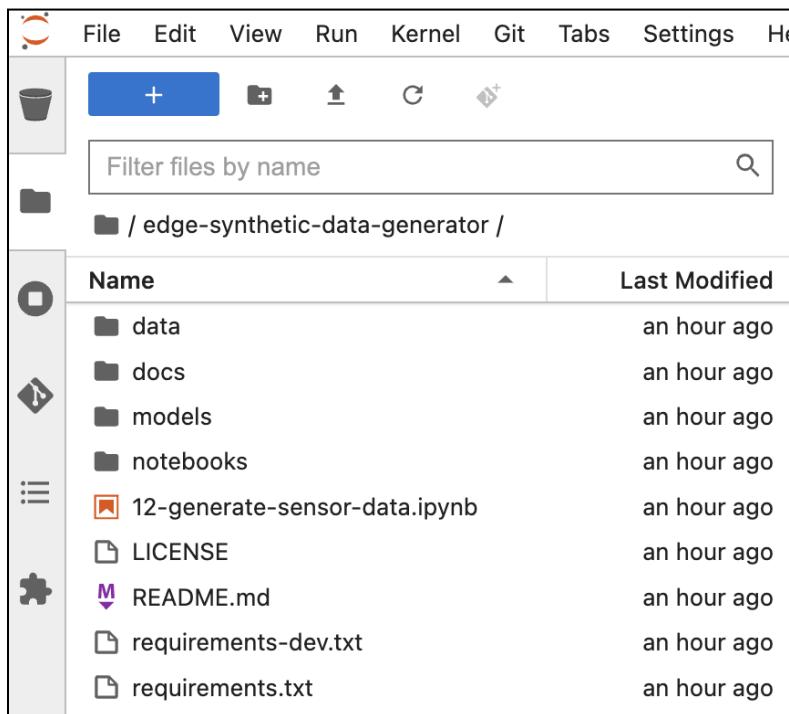
### Environment variables

[+ Add more variables](#)

In the new Start a Notebook server page, choose ‘PyTorch’, ‘small’ container size. Then press the “Start Notebook” server button.

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. **For the purposes of testing, we’ll have one person from each group generate synthetic data and stream it. So if your name is in bold on the Workshop Signup spreadsheet you will be responsible for doing so.**



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.

The screenshot shows a Jupyter Notebook interface. On the left is a file browser with a tree view of a directory structure under 'edge-synthetic-data-generator/'. The current file selected is '12\_generate\_sensor\_data.ipynb'. The main area contains the notebook content:

## Generate Sensor Data

In this notebook, we will generate synthetic data from a model trained on our real sensor data.

We use an open-source implementation by Gretel AI found [here](#) of the generative adversarial network known as DoppelGANger. For more information on DoppelGANger, see the [paper](#) and the respective GitHub [repository](#).

We are generating data based on preexisting public water pump sensor data, found on [kaggle](#).

```
[ ]: # first, pip installs to ensure we have the correct packages
!pip install torch==1.11.0 # version recommended by source
!pip install git+https://github.com/gretelai/gretel-synthetics.git
!pip install numpy==1.20.0 # for new concatenate feature
!pip install kafka-python==2.0.2

[ ]: GROUP_ID = None

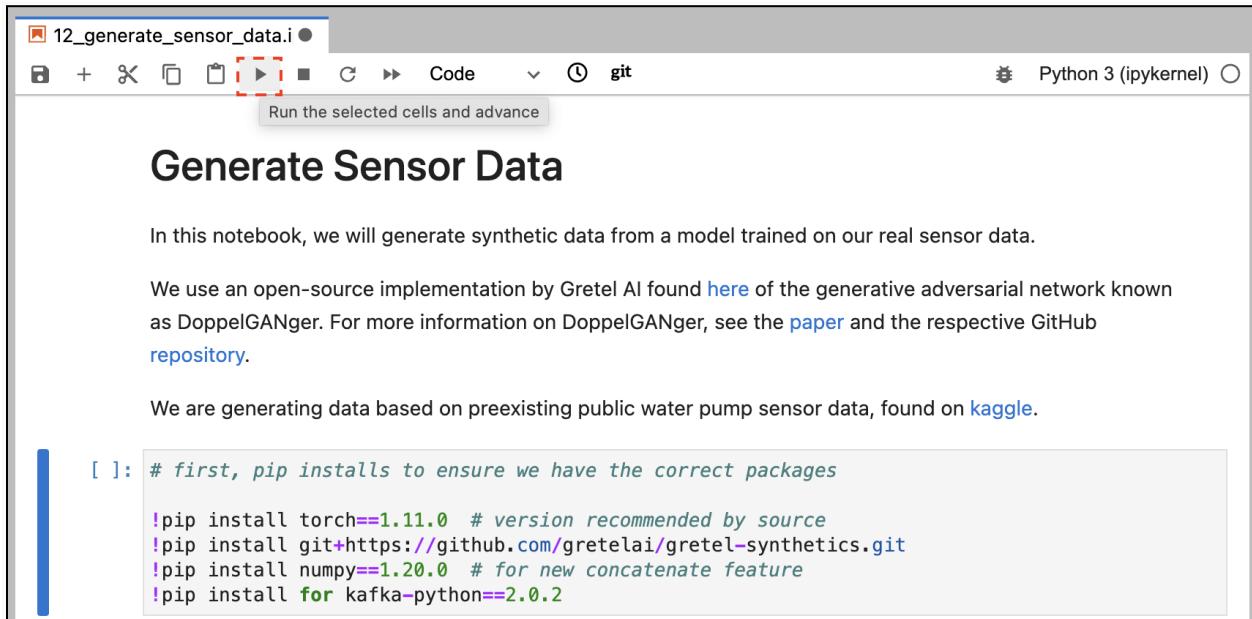
[ ]: import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.dates as md
from pickle import dump, load

import torch

from gretel_synthetics.timeseries_dgan.dgan import DGAN
from gretel_synthetics.timeseries_dgan.config import DGANConfig, OutputType
```

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.



12\_generate\_sensor\_data.ipynb

Code git Python 3 (ipykernel)

Run the selected cells and advance

## Generate Sensor Data

In this notebook, we will generate synthetic data from a model trained on our real sensor data.

We use an open-source implementation by Gretel AI found [here](#) of the generative adversarial network known as DoppelGANGER. For more information on DoppelGANGER, see the [paper](#) and the respective GitHub [repository](#).

We are generating data based on preexisting public water pump sensor data, found on [kaggle](#).

```
[ ]: # first, pip installs to ensure we have the correct packages
!pip install torch==1.11.0 # version recommended by source
!pip install git+https://github.com/gretelai/gretel-synthetics.git
!pip install numpy==1.20.0 # for new concatenate feature
!pip install kafka-python==2.0.2
```

Next, in the following cell, replace ‘None’ with the GROUP\_ID number that is part of your username. E.g. user3: GROUP\_ID = 3



12\_generate\_sensor\_data.ipynb

Code git Python 3 (ipykernel)

```
[ ]: GROUP_ID = None
```

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

## 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:

The screenshot shows a Jupyter Notebook interface with a cell titled "Selecting your slice.". The cell contains the following text and code:

**Selecting your slice.**

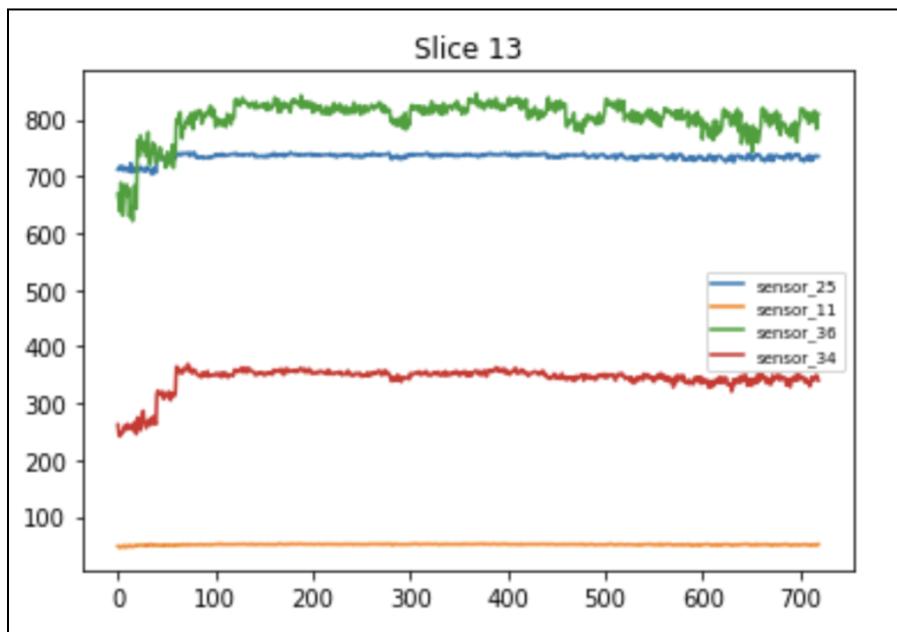
- Now that we have generated 1,000 slices of random data, it's time for you to choose a single slice.
- In the plot above, you can see 9 different plots of slices of data. Above each plot is the slice number.
- If you don't like the look of any of the plots in this figure, you can re-run the cell above to see another set of 9 random slices.
- Once you see a slice that you like, **enter the corresponding slice number into the cell below where it's marked.**

```
[ ]: # enter your slice number below
my_slice_num = 13

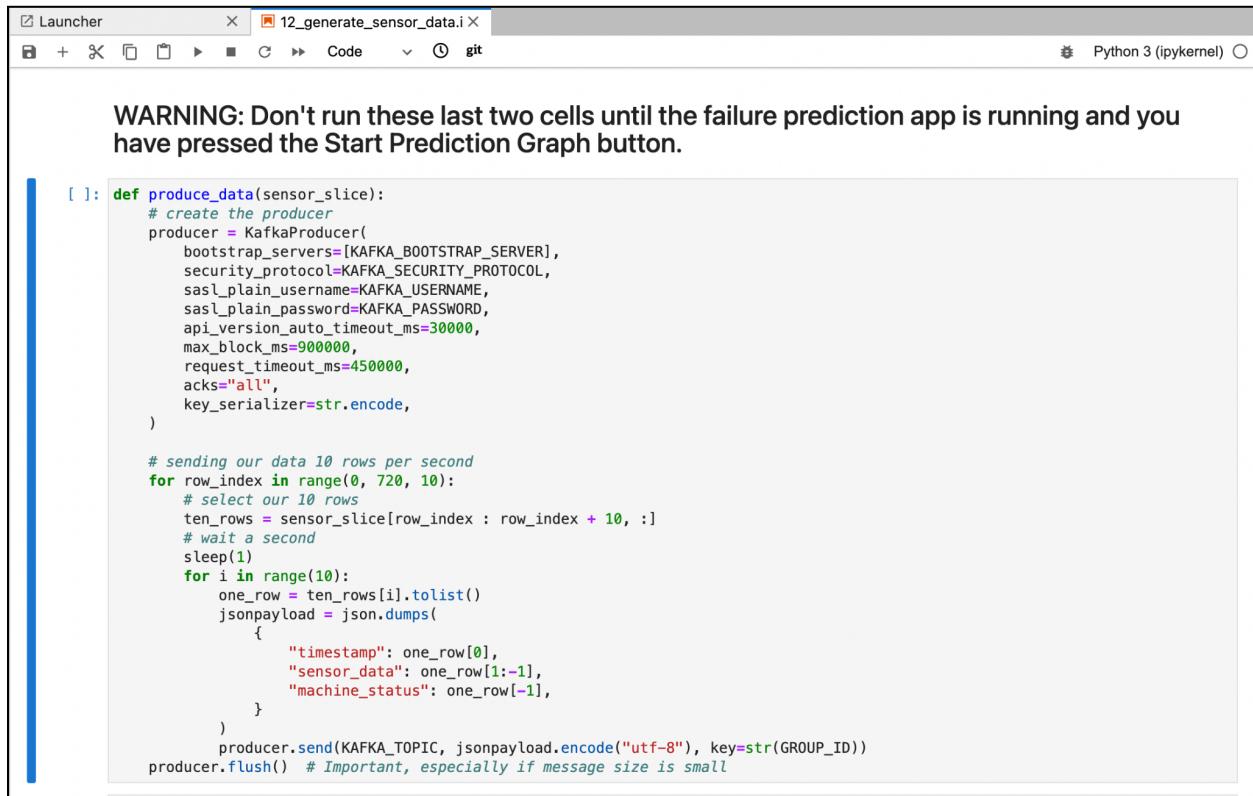
my_slice = synthetic_features[my_slice_num]

# let's plot that slice to make sure we've got it.
plot_12hr_slice(my_slice, my_slice_num)
plt.show()
```

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:



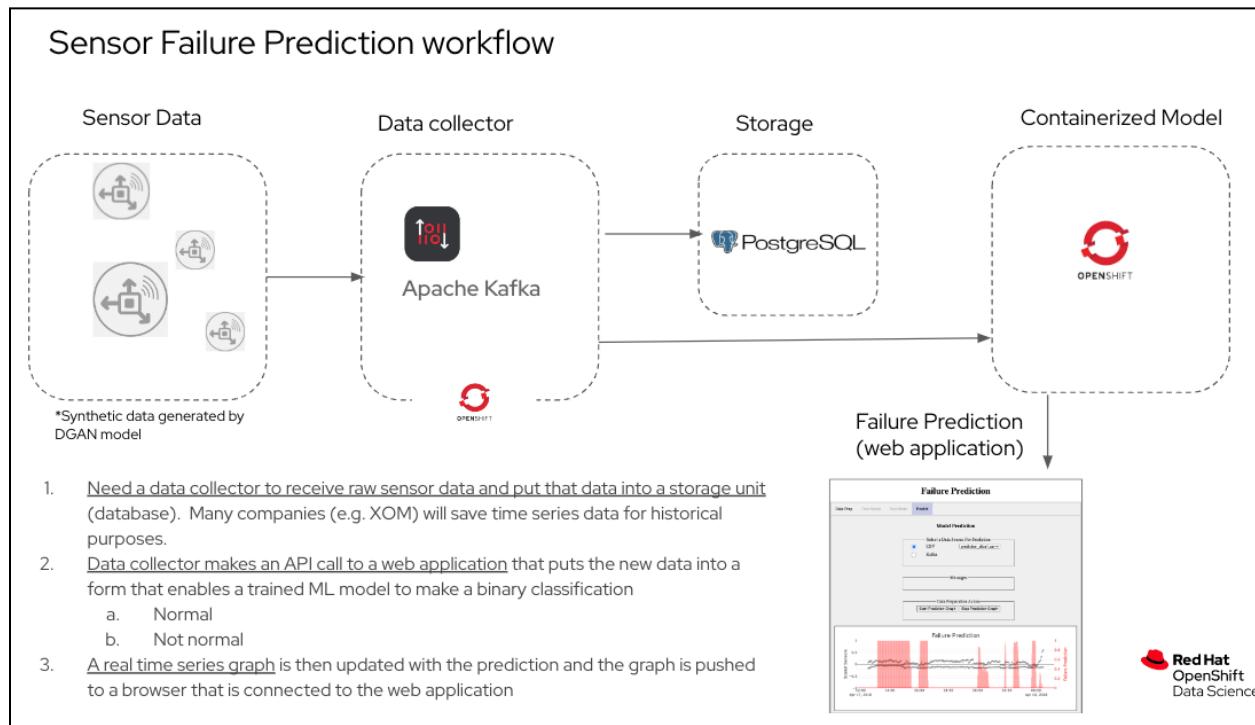
The screenshot shows a Jupyter Notebook interface with a single open cell. The cell contains Python code for generating sensor data and sending it to a Kafka topic. A warning message is displayed above the code.

```
[ ]: 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
```

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

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.



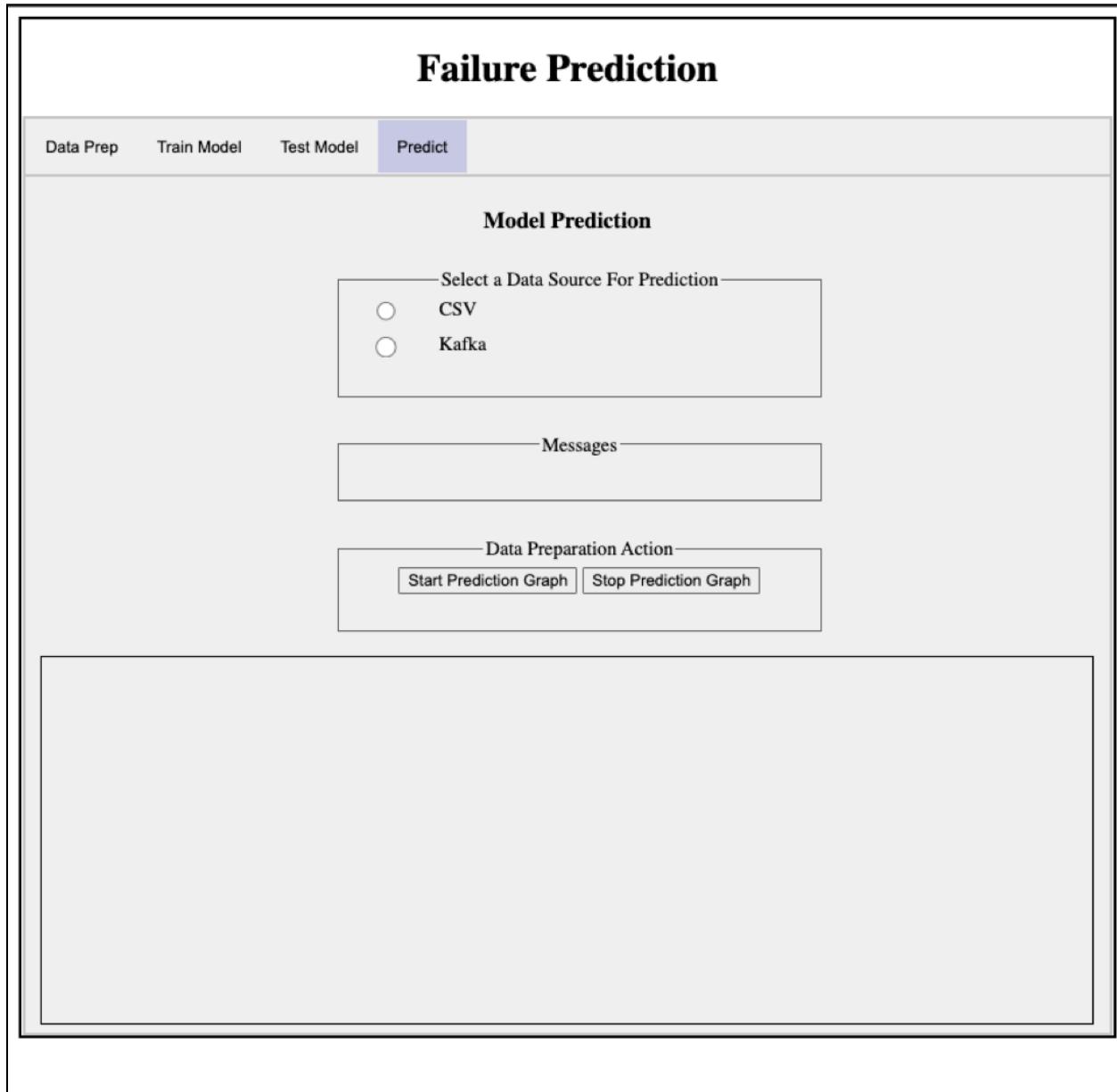
When we generate our synthetic data, it needs a service to push it to our model for failure prediction. The service that we use is called **Apache Kafka Streaming**. This service takes our generated synthetic data and pushes it to our deployed (and containerized) Failure Prediction web application.

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.

## Prediction With the Model

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:

The screenshot shows the 'Failure Prediction' application interface. At the top, there are four tabs: 'Data Prep', 'Train Model', 'Test Model', and 'Predict'. The 'Predict' tab is highlighted with a blue background. Below the tabs, the main content area is divided into three sections:

- Model Prediction:** A section titled "Select a Data Source For Prediction" containing two radio buttons: "CSV" and "Kafka".
- Messages:** A section displaying the message "No data source has been selected" in red text.
- Data Preparation Action:** A section containing two buttons: "Start Prediction Graph" and "Stop Prediction Graph".

If you choose the CSV radio button, a select box will list CSV file names available. Just **select a CSV filename.**

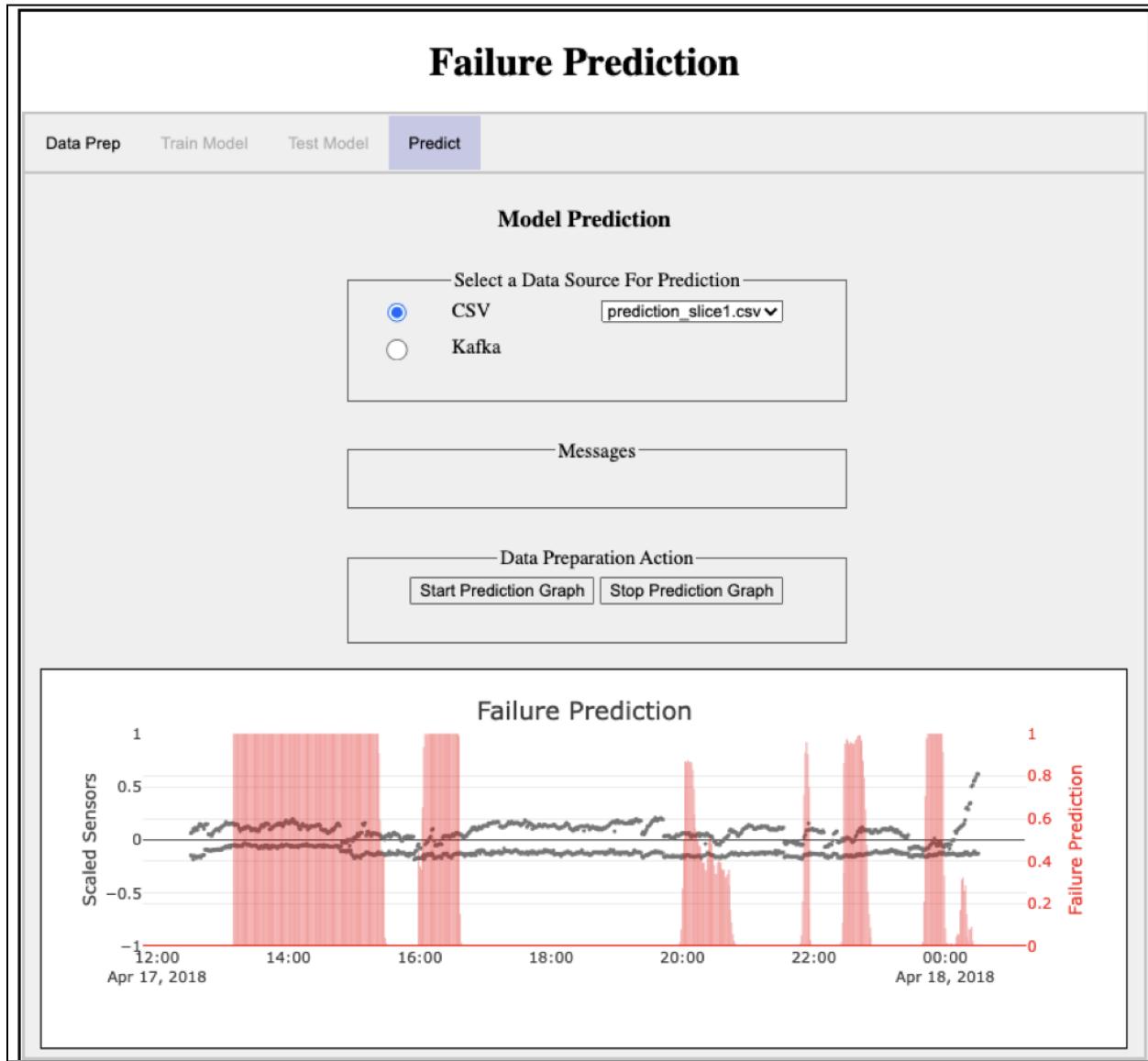
If you chose the Kafka radio button, **enter the Group Id** that you have been given. This Group Id allows your browser session to pull out messages that you were identified with.

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.

# CSV Option

**Click on the Start Prediction Graph button.**

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.

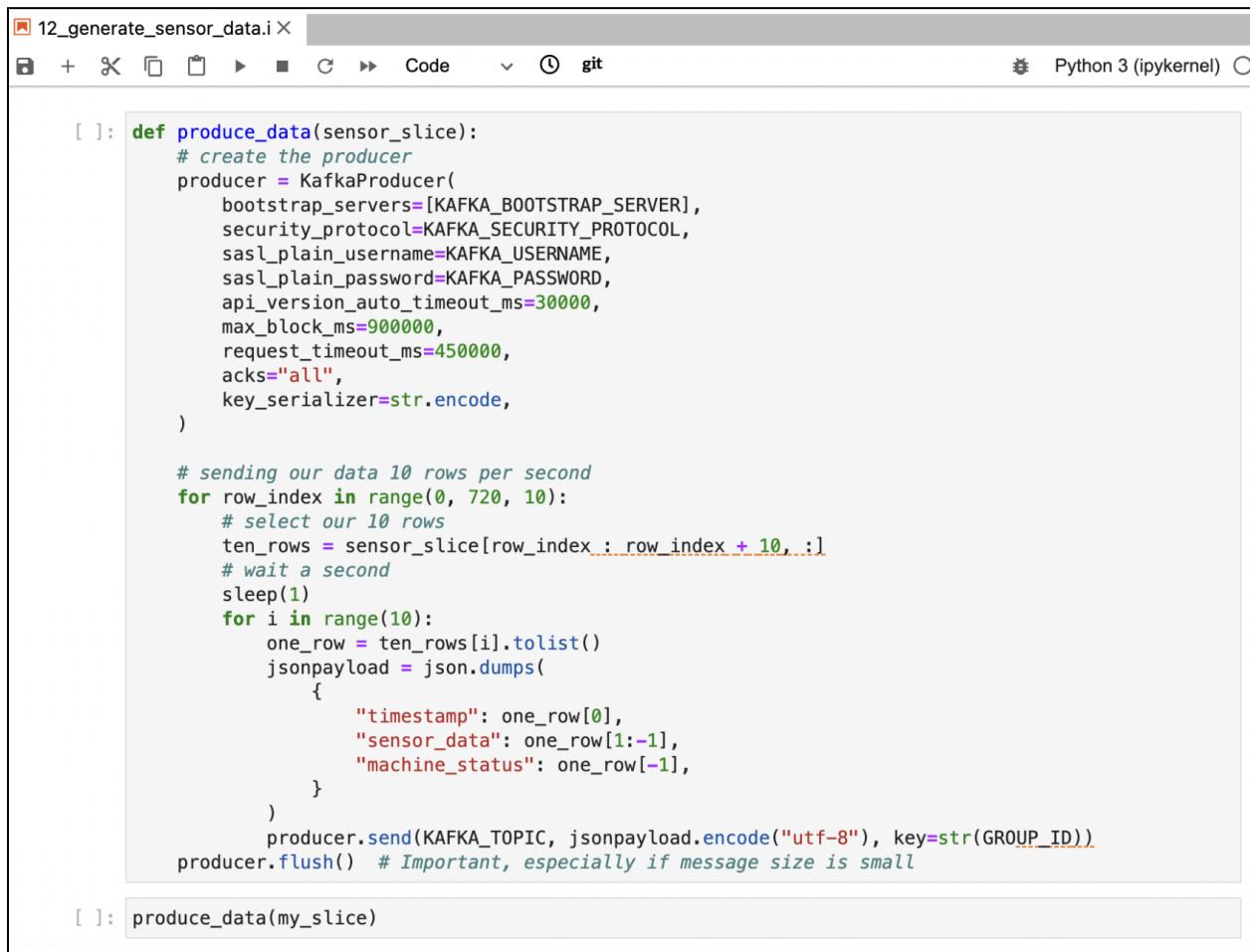


In this case actual failure was at 2018-04-18 00:30. The graph in your browser is interactive, so you can move your cursor to display the times.

# Kafka Option

**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



```
[ ]: 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

[ ]: produce_data(my_slice)
```

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.

**This concludes our workshop.**