Customizing Workbenches

Introduction

RHOAI provides preconfigured notebook images to create workbenches. Each of these images is designed to support your data science workflow with specific and popular technological stacks. For example, if you plan to work with PyTorch, then you might want to use the PyTorch image.

Supported Notebook Images

RHOAI ships with a collection of notebook images, which are optimized and supported by Red Hat and independent software vendors (ISVs). These images are designed for data scientists, providing them with the necessary tools to start working quickly. Moreover, Red Hat provides support for these images for at least one year, and releases major versions of each image typically every six months.

These images are regular container images. The source Dockerfiles for these images are available at the red-hat-data-services/notebooks GitHub repository. These images are also available in the Quay.io registry at https://quay.io/organization/modh.
For more information about notebook images, refer to the Notebook images for data scientists section in the Red Hat OpenShift AI documentation.
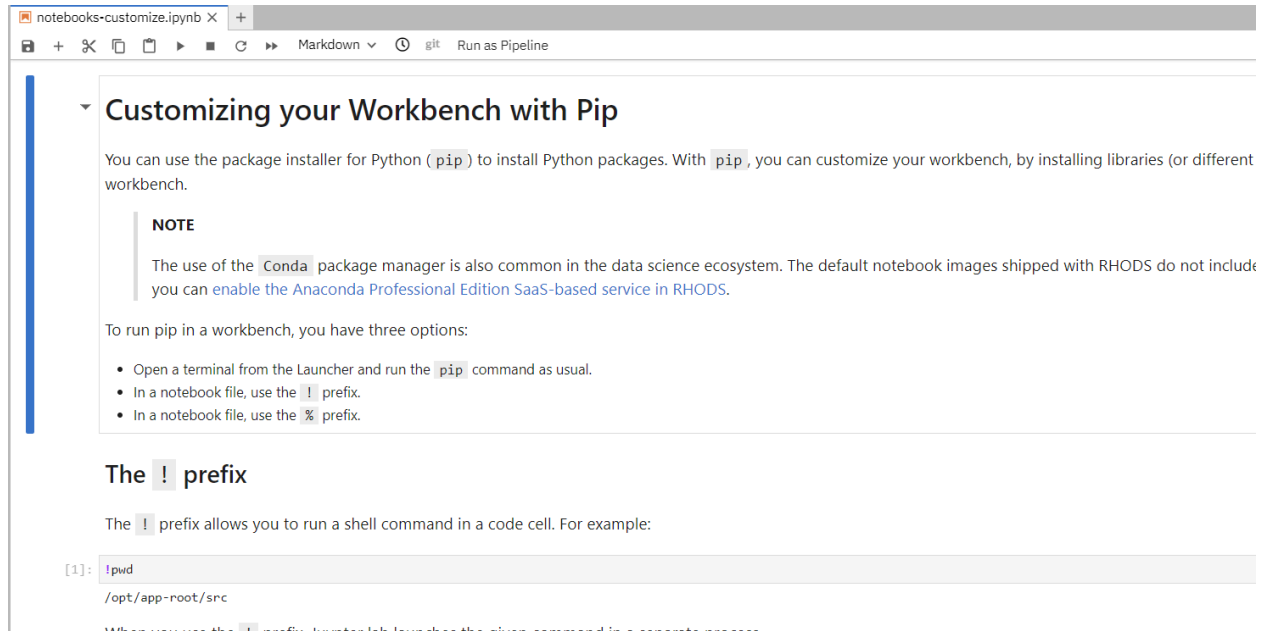
Custom Notebook Images

We will go through how you can add new custom notebook images to your environment in the Admin section, and in an exercise below show how you can create a workbench from a custom notebook image.

Exercise: Customizing Workbenches with JupyterLab

To use JupyterLab for customizing a workbench, follow these steps:
1. Enter your workbench.

2. Go to the folder rhoai-training/lab-material/customize-notebook.
3. Open notebooks-customize.ipynb and follow the instructions.

Exercise: Test a Custom Notebook Image

If you customize the notebook image, you can do more than things like customizing python packages.

Follow these steps to try a custom notebook image:
1. Go back to your Data Science Project view.
2. Start creating a new Workbench.
3. Use these settings, fill out the rest of the fields as you see fit:
   a. Notebook Image - Custom: VS Code
   b. Container size - Custom Size - Small
4. Create the workbench and open it when ready.
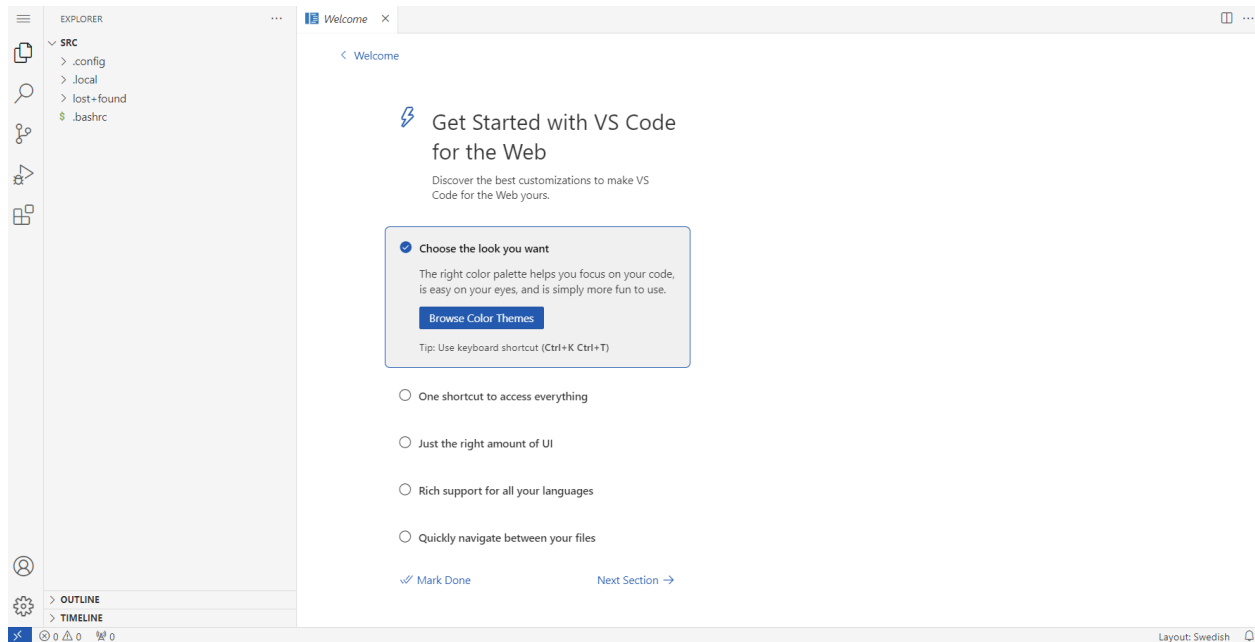
As you can see, we now have an environment that looks like VS Code instead of JupyterLab.

Pipelines

Introduction

Data Science Pipelines is Red Hat's implementation of Kubeflow Pipelines. Data Science Pipelines uses OpenShift Pipelines (Tekton) as the execution engine for pipelines, unlike the upstream Kubeflow Pipelines, which utilizes Argo Workflows. Another important distinction between the upstream Kubeflow Pipelines project and Data Science Pipelines, is that Data Science Pipelines are designed to support multi-tenancy.

Users are then able to create pipelines using a variety of methods, and submit them for execution.

Red Hat OpenShift AI offers two out-of-the-box mechanisms to work with Data Science Pipelines in terms of building and triggering pipelines.

The first mechanism is the Elyra Pipelines JupyterLab extension, which provides a visual editor for creating pipelines based on Jupyter notebooks as well as Python or R scripts.

The second mechanism is based on the Kubeflow Pipelines SDK. With the SDK, pipelines are built using Python scripts and submitted to the Data Science Pipelines runtime to be scheduled for execution.

Data Science Pipeline Concepts

- Pipeline - is a workflow definition containing the steps and their input and output artifacts.
- Run - is a single execution of a pipeline. A run can be a one off execution of a pipeline, or pipelines can be scheduled as a recurring run.
- Step - is a self-contained pipeline component that represents an execution stage in the pipeline.
- Artifact - Steps have the ability to create artifacts, which are objects that can be persisted after the execution of the step completes. Other steps may use those artifacts as inputs and some artifacts may be useful references after a pipeline run has completed. Artifacts automatically stored by Data Science Pipelines in S3 compatible storage.
- Experiment - is a logical grouping of runs for the purpose of organization.

> A pipeline is an execution graph of tasks, commonly known as a DAG (Directed Acyclic Graph). A DAG is a directed graph without any cycles, i.e. direct loops.

A data science pipeline is typically implemented to improve the repeatability of a data science experiment.

A data science pipeline may also fit within the context of a larger pipeline that manages the complete lifecycle of an application, and the data science pipeline is responsible for the process of training the machine learning model.

Data science pipelines may consist of several key activities that are performed in a structured sequence to train a machine learning model. These activities may include:

- Data Collection: Gathering the data from various sources, such as databases, APIs, spreadsheets, or external datasets.
- Data Cleaning: Identifying and handling missing or inconsistent data, removing duplicates, and addressing data quality issues to ensure that the data is reliable and ready for analysis.
- Feature Engineering: Creating or transforming features (variables) to improve the performance of machine learning models. This may involve

scaling, one-hot encoding, creating new variables, or reducing dimensionality.
- Data Preprocessing: Preparing the data for modeling, which may involve standardizing, normalizing, or scaling the data. This step is crucial for machine learning algorithms that are sensitive to the scale of features. This step may also include splitting the data into multiple subsets of data including a test and train dataset to allow the model to be validated using data the trained model has never seen.
- Model Training: After the data has been split into an appropriate subset, the model is trained using the training dataset. As part of the training process, the machine learning algorithm will generally iterate through the training data, making adjustments to the model until it arrives at the "best" version of the model.
- Model Evaluation: The model performance is assessed with the previously unseen test dataset using various metrics, such as accuracy, precision, recall, F1 score, or mean squared error. Cross-validation techniques may be used to ensure the model's robustness.

A single pipeline may include the ability to train multiple models, complete complex hyperparameter searches, or more.

Data Scientists can use a well crafted pipeline to quickly iterate on a model, adjust how data is transformed, test different algorithms, and more.

While the steps described above describe a common pattern for model training, different use cases and projects may have vastly different requirements and the tools and framework selected for creating a data science pipeline should help to enable a flexible design.

Exercise: Configure a Data Science Pipeline Server

Before we can run any pipelines we need to configure a pipeline server in our Data Science Project.

Follow these steps to do that:
1. Go back to the OpenShift AI dashboard
2. Navigate to Data Science Pipelines → Pipelines.
3. Click Create a pipeline server.

4.

Applications ›

Data Science Projects

Data Science Pipelines ›

Model Serving

Resources

Data connections   Add data connection

Name ↕	Type	Connected workbenches	Provider	
My Storage ⓘ	📦 Object storage	No connections	AWS S3	⋮
Pipeline Artifacts ⓘ	📦 Object storage	No connections	AWS S3	⋮

Pipelines   Import pipeline

No pipeline server

To import a pipeline, first create a pipeline server.

Create a pipeline server

Models and model servers   Add server

No model servers

Before deploying a model, you must first add a model server.

5. Click the little key/dropdown symbol next to the Access Key field.

6. Choose Pipeline Artifacts.
7. Leave the database configuration as the default.
8. Click Configure and wait until the spinner disappears and "No pipelines yet" is displayed.

If you have waited more than 5 minutes and the Pipeline server actions option does not appear, you can try to delete the pipeline server and create it again.
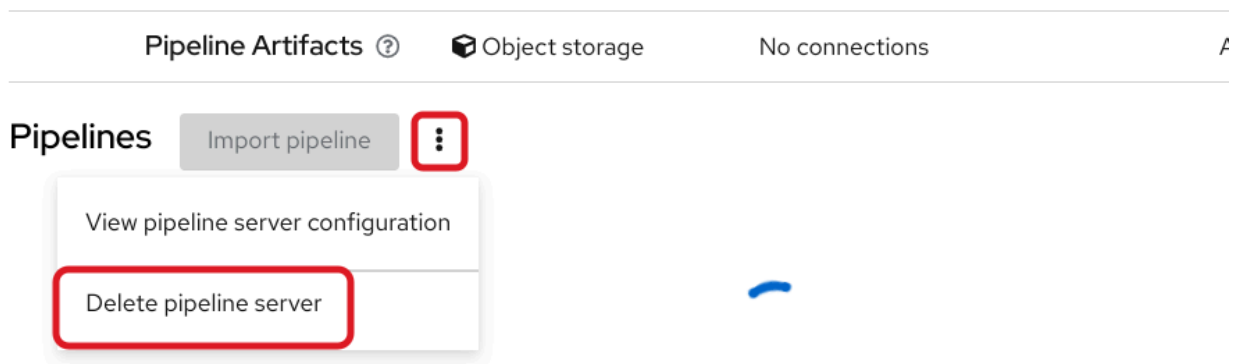
9. After the spinner has disappeared, we need to force an update in our workbench to get the correct settings, specifically the Runtime for the pipeline. To do that, edit your workbench and add a description to it.

Description

Pipelines time!

10. Press Update Workbench and it will restart and be ready with the pipeline setting.

Exercise: Build and Run an Elyra Pipeline

Introduction

Previously, you used a notebook to train and save your mode, now we will use Red Hat OpenShift AI pipelines to automate it.

In this section, you create a simple pipeline by using the GUI pipeline editor. The pipeline uses the notebook that you used in previous sections, to train a model and then save it to S3 storage.

Note: Your completed pipeline should look like the one in the 4 Train Save.pipeline file.

If your pipeline editor goes blank at any point, just go back and forth between two tabs in JupyterLab.

Create a pipeline

1. Open your workbench's JupyterLab environment. If the launcher is not visible, click + to open it.
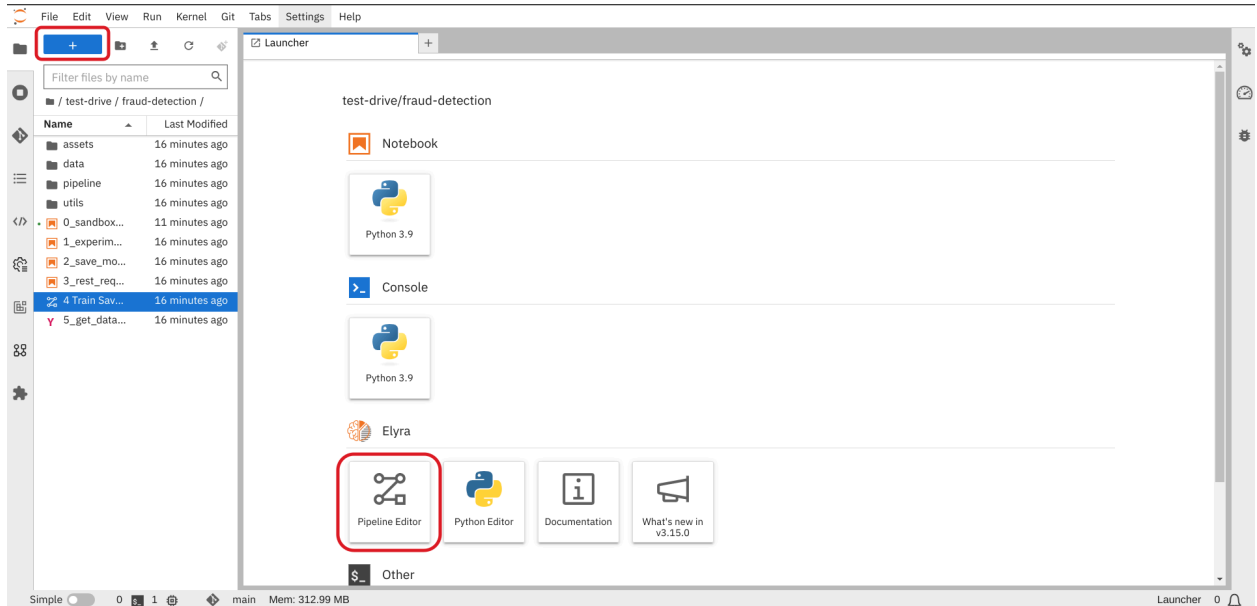
2.
3. Click Pipeline Editor.

4.
5. You've created a blank pipeline!
6. Set the default runtime image for when you run your notebook or python code.
   a. In the pipeline editor, click Open Panel
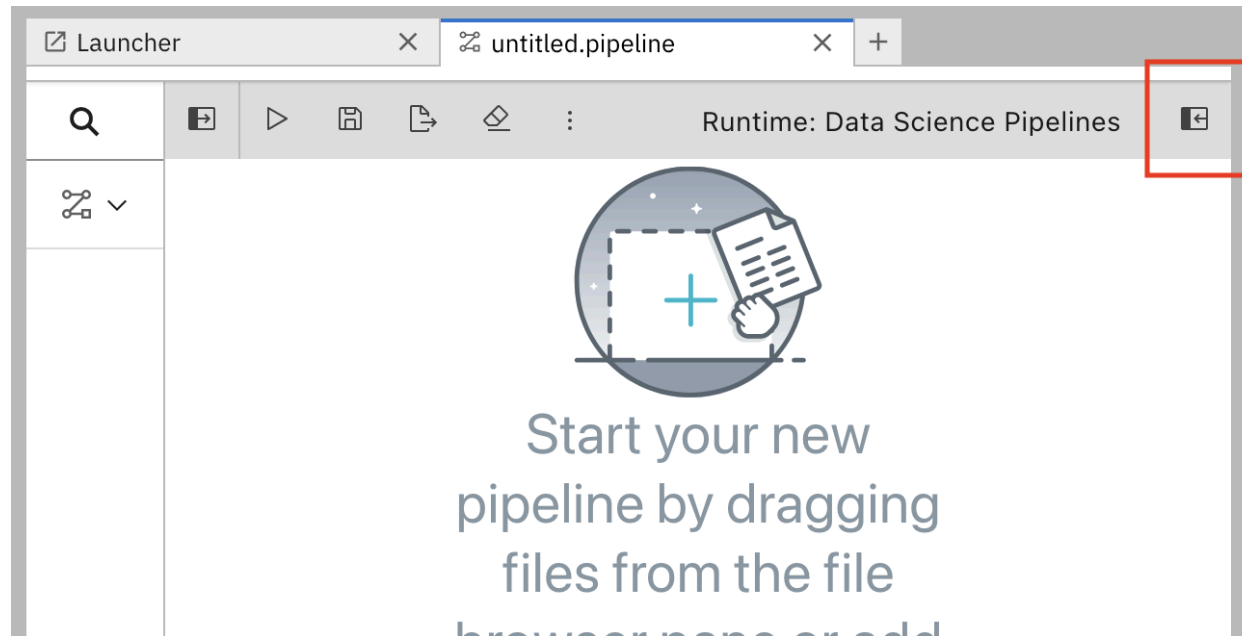
b.

c. Select the Pipeline Properties tab.
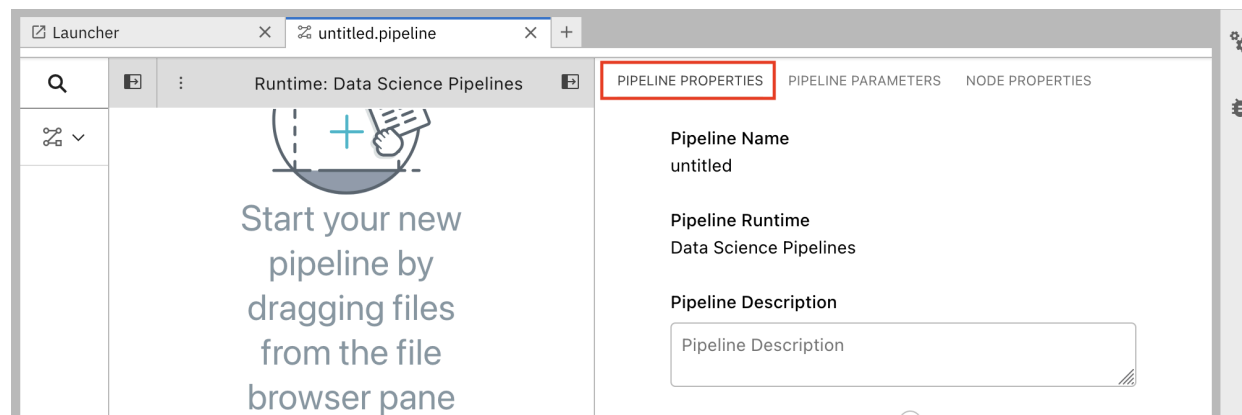
d.

e. In the Pipeline Properties panel, scroll down to Generic Node Defaults and Runtime Image. Set the value to Tensorflow with Cuda and Python 3.9 (UBI 9)

Kubernetes Tolerations (?)

Add

Shared Memory Size (?)

Memory Size (GB)

| 0 | ⌄ |

Kubernetes Pod Labels (?)

Add

Generic Node Defaults (?)

Runtime Image (?)

| TensorFlow with CUDA and Python 3.9 (UBI9) ▼ |

Kubernetes Secrets (?)

Add

Environment Variables (?)

Add   Refresh

Custom Node Defaults (?)

f.

7. Save the pipeline.

Add nodes to your pipeline

Add some steps, or nodes in your pipeline. Your two nodes will use the 1_experiment_train.ipynb and 2_save_model.ipynb notebooks from your lab-material/fraud-detection folder.

1. From the file-browser panel, drag the 1_experiment_train.ipynb and 2_save_model.ipynb notebooks onto the pipeline canvas.
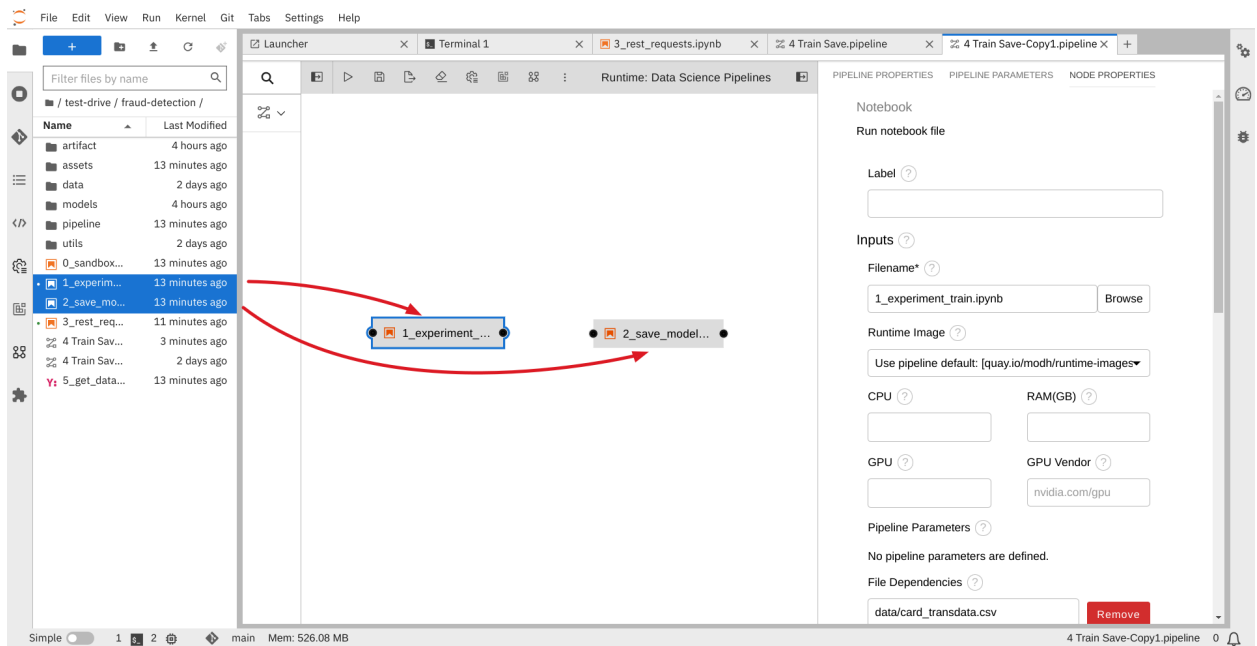
2.
3. Click the output port of 1_experiment_train.ipynb and drag a connecting line to the input port of 2_save_model.ipynb.

4.
5. Save the pipeline.

Specify the training file as a dependency

Set node properties to specify the training file as a dependency.
Note: If you don't set this file dependency, the file won't be included in the node when it runs and the training job would fail.

1. Click the 1_experiment_train.ipynb node.
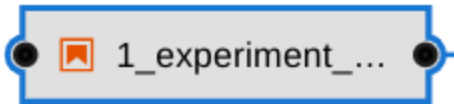
2.
3. In the Properties panel, click the Node Properties tab.
4. Scroll down to the File Dependencies section and then click Add.
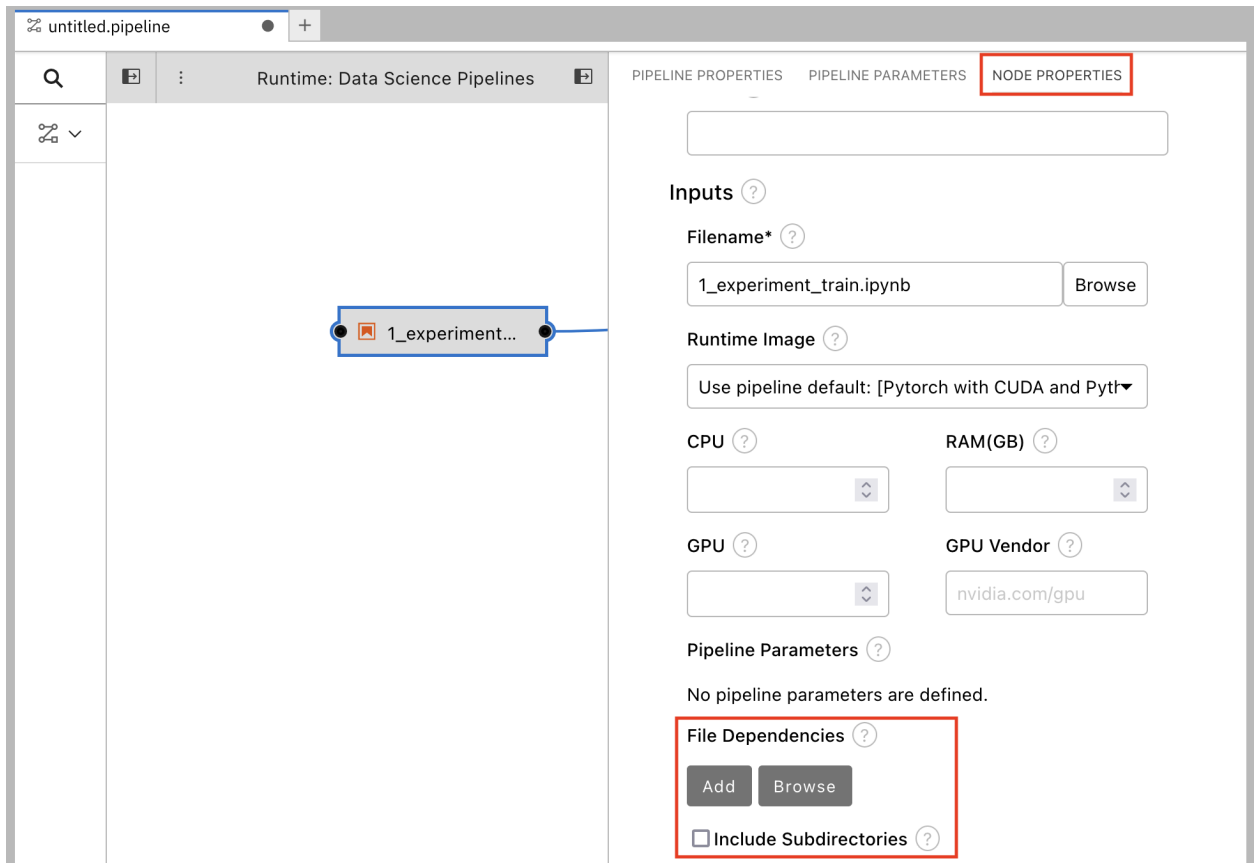
5.
6. Set the value to data/card_transdata.csv which contains the data to train your model.

7.
8. Save the pipeline.

Create and store the ONNX-formatted output file

In node 1, the notebook creates the models/fraud/1/model.onnx file. In node 2, the notebook uploads that file to the S3 storage bucket.

1. Select node 1 and then select the Node Properties tab.
2. Scroll down to the Output Files section, and then click Add.
3. Set the value to models/fraud/1/model.onnx.

Outputs ⑦

Output Files ⑦

| models/fraud/1/model.onnx | Remove |

Add

4.

Configure the data connection to the S3 storage bucket

In node 2, the notebook uploads the model to the S3 storage bucket.
You must set the S3 storage bucket keys by using the secret created by the My Storage data connection that you set up in the [Storage Data Connections](#) section of this lab.
You can use this secret in your pipeline nodes without having to save the information in your pipeline code. This is important, for example, if you want to save your pipelines - without any secret keys - to source control.
The secret is named aws-connection-my-storage.

> If you called your data connection something other than My Storage, you can obtain the secret name in the Data Science dashboard by hovering over the resource information icon ? in the Data Connections tab.
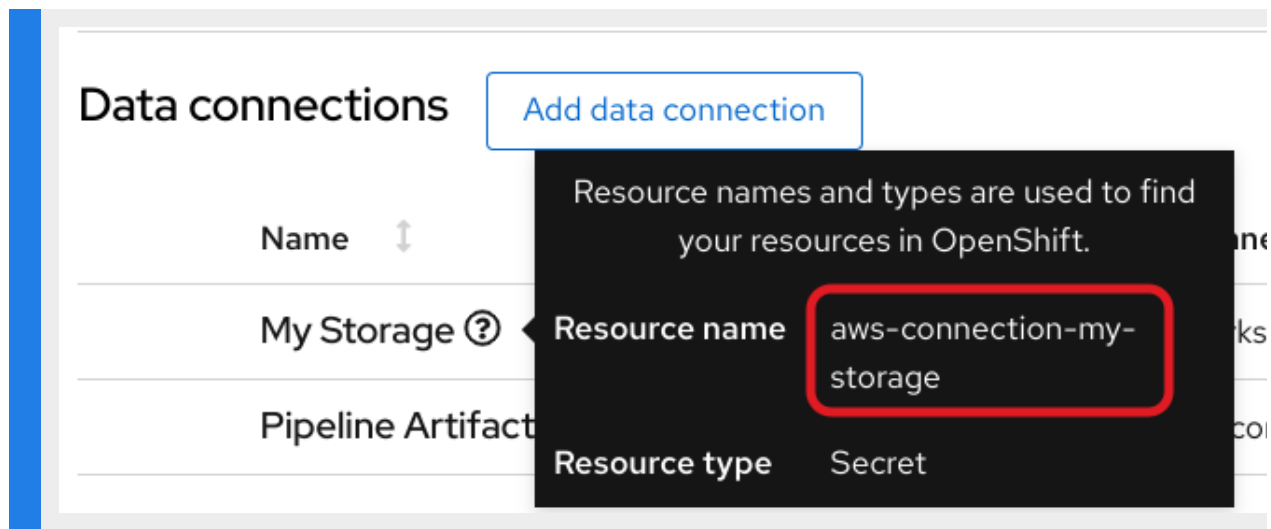
The aws-connection-my-storage secret includes the following fields:
- AWS_ACCESS_KEY_ID
- AWS_DEFAULT_REGION
- AWS_S3_BUCKET
- AWS_S3_ENDPOINT
- AWS_SECRET_ACCESS_KEY

You must set the secret name and key for each of these fields.

1. Remove any pre-filled environment variables.
    a. Select node 2, and then select the Node Properties tab.
       Under Additional Properties, note that some environment variables have been pre-filled. The pipeline editor inferred that you'd need them from the notebook code.
       Since you don't want to save the value in your pipelines, remove all of these environment variables.
    b. Click Remove for each of the pre-filled environment variables.

c.

2. Add the S3 bucket and keys by using the Kubernetes secret.
   a. Under Kubernetes Secrets, click Add.
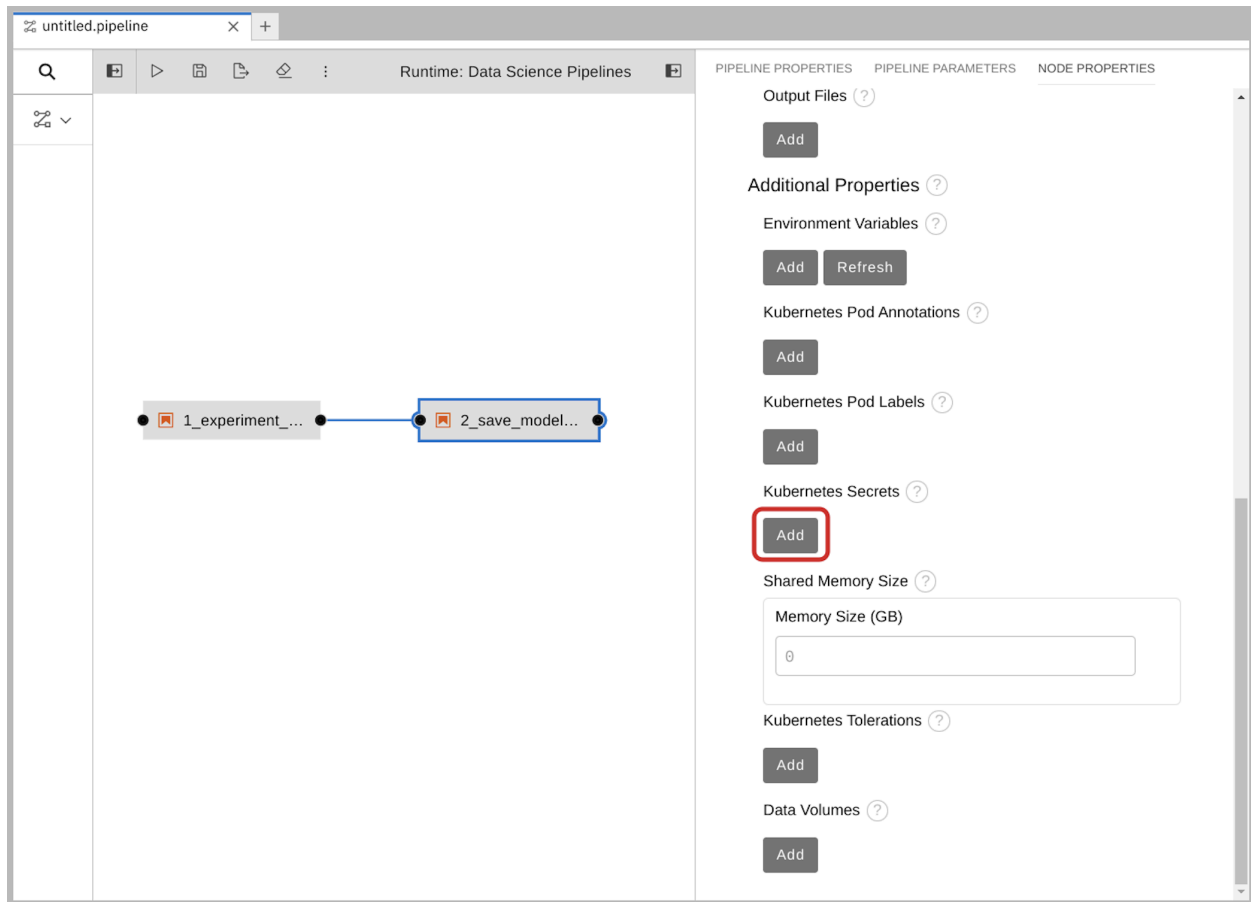
b.

c. Enter the following values and then click Add.

- ■ Environment Variable: AWS_ACCESS_KEY_ID
- ■ Secret Name: aws-connection-my-storage
- ■ Secret Key: AWS_ACCESS_KEY_ID

Kubernetes Secrets (?)

Environment Variable*

```
AWS_ACCESS_KEY_ID
```

Secret Name*

```
aws-connection-my-storage
```

Secret Key*

```
AWS_ACCESS_KEY_ID
```

Remove

Add

- ■
  d. Repeat Steps 3a and 3b for each set of these Kubernetes secrets:
     - ■ Environment Variable: AWS_SECRET_ACCESS_KEY
        - ■ Secret Name: aws-connection-my-storage
        - ■ Secret Key: AWS_SECRET_ACCESS_KEY
     - ■ Environment Variable: AWS_S3_ENDPOINT
        - ■ Secret Name: aws-connection-my-storage
        - ■ Secret Key: AWS_S3_ENDPOINT
     - ■ Environment Variable: AWS_DEFAULT_REGION
        - ■ Secret Name: aws-connection-my-storage
        - ■ Secret Key: AWS_DEFAULT_REGION
     - ■ Environment Variable: AWS_S3_BUCKET
        - ■ Secret Name: aws-connection-my-storage
        - ■ Secret Key: AWS_S3_BUCKET
3. Save and Rename the .pipeline file.

Run the Pipeline

Upload the pipeline on the cluster itself and run it. You can do so directly from the pipeline editor. You can use your own newly created pipeline for this or 4 Train Save.pipeline.

1. Click the play button in the toolbar of the pipeline editor.
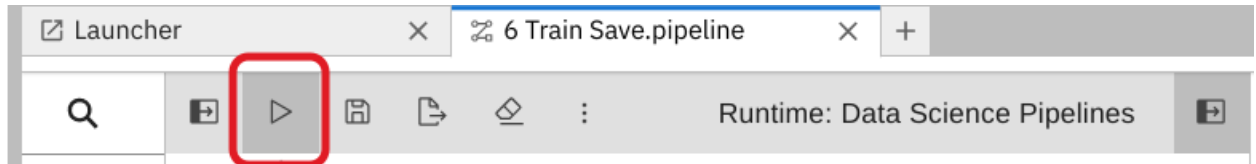
2.
3. Enter a name for your pipeline.
4. Verify the Runtime Configuration: is set to Data Science Pipeline.
5. Click OK.

> If Data Science Pipeline is not available as a runtime configuration, you may have created your notebook before the pipeline server was available. You can stop and edit the description on your notebook then restart it. If this is done after the pipeline server has been created, you will see the Data Science Pipeline option.

1.
Return to your data science project and expand the newly created pipeline.

2.
3. Click the pipeline or the pipeline run and then view the pipeline run in progress.

4.

The result should be a models/fraud/1/model.onnx file in your S3 bucket which you can serve, just like you did manually before.

Exercise: Use Kubeflow SDK

The following is a simplistic example of a KFP pipeline. The original is at https://github.com/kubeflow/kfp-tekton/blob/master/samples/flip-coin/condition.py and the example has downloaded it to a file named coin-toss.py.
You can find it in the lab-material/python-pipeline folder.

To compile it into a Tekton resource definition just run the following in a terminal

```
Unset
python3 coin-toss.py
```

It will generate a Tekton Pipeline Run , similar to this snippet

```
Unset
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: conditional-execution-pipeline
```

```
annotations:
```

Download the yaml file by right clicking on it and pressing Download.
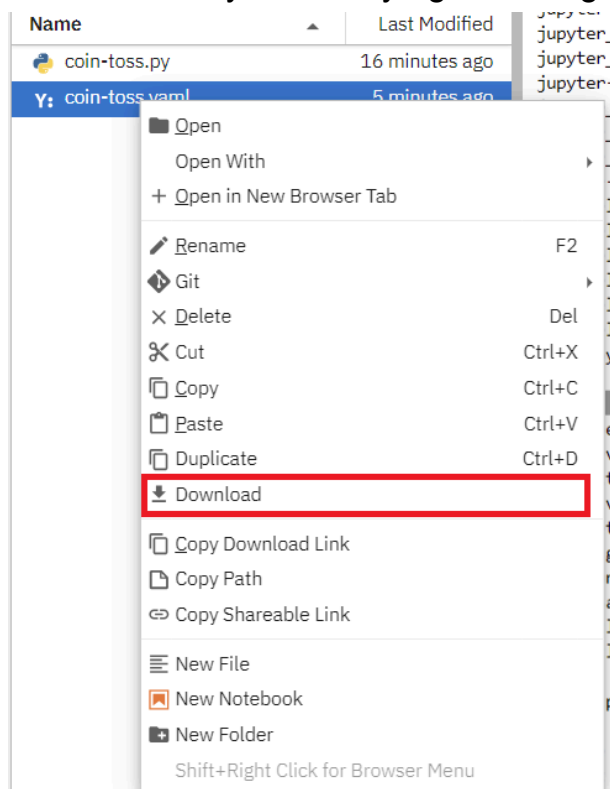
We can now upload this pipeline by going back to the Data Science Project and pressing Import Pipeline.

Give the pipeline a good name and description and then press Import pipeline.

Once imported, a new pipeline will show up in your Data Science Project. Click that pipeline name and the structure of the DAG will be shown. Each step in the pipeline will be run as a container on OpenShift.

Pipelines   Import pipeline ⋮

Pipeline name ↕	Last run	Last run status
❯ coin-toss	–	–

coin-toss-pipeline
Example coin toss

Graph   YAML

Actions ▼

Create run

View runs

Delete pipeline
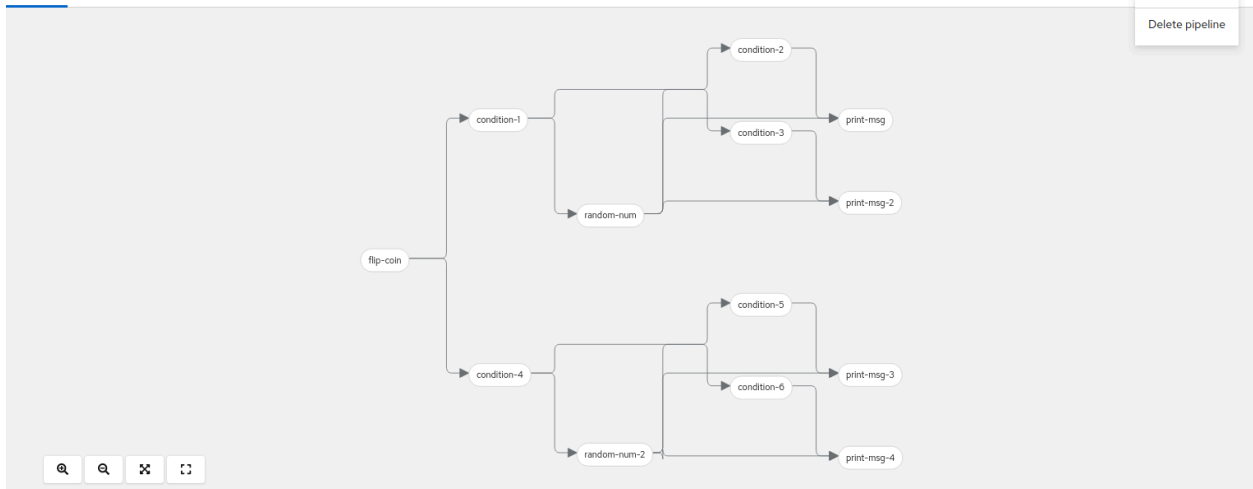
To execute the pipeline, click on Create Run in the menu and fill out the Name and Description. If the pipeline has Input Parameters or you need to schedule a recurring run, then that can be configured further down.

In this case, use the runtype "Run once immediately after creation".

Once ready, click Create to submit the pipeline for scheduling.

Create run

Create a run from a pipeline.

Jump to section

Name and description

Pipeline

Run type

Pipeline input parameters

Project

Project

rhods-enablement

Name *

> coin-toss-run1

Description

Pipeline

coin-toss-pipeline ▼

⊕ Import pipeline

Create   Cancel

The pipeline will execute and the outputs will be stored into the configured S3 bucket. As the pipeline executes the view will be updated to show the steps being executed. You can click on the graph nodes to reveal information about the steps.

Once the pipeline has completed, you can access the output and pipeline artifacts in the object storage browser of the Minio Storage UI, just like with the Elyra run.
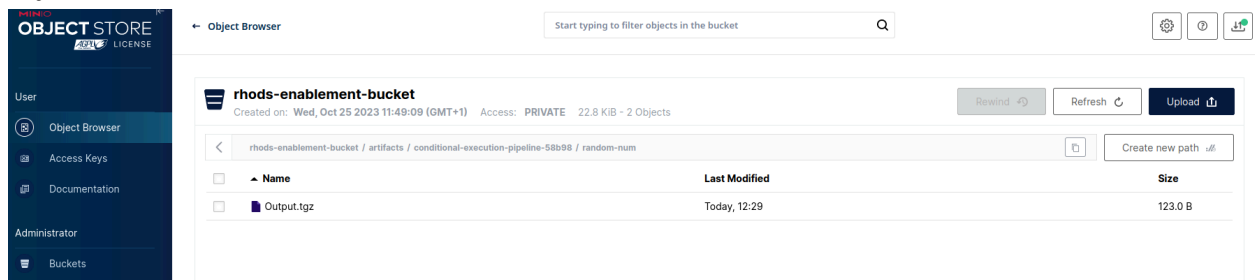
LLM Serving

Introduction

Now that you've learned the basics of Red Hat OpenShift AI using a Predictive AI model, it's time to explore Generative AI.

Generative AI is a subset of artificial intelligence that creates models that generate new data.

It's used in a variety of applications, including image generation, text generation,

and music generation. If you've used ChatGPT or StableDiffusion, you've used Generative AI.

In this section, we will explore using a large language model (LLM) downloaded from Huggingface.
We will be using a small LLM we can use with minimal resources. In fact, calling a "Small LLM" is a bit misleading, but "SLM" is not a thing, so…🤷

In Generative AI, the early parts of the workflow are very different, compared to Predictive AI:
- In predictive AI, you typically use historical data to train a new model from scratch.
- In generative AI, the models have often been trained for us already, and so we download them from a site like HuggingFace and then deploy them in our Model Serving UI.

At a high level, we will:
- Download a model from Huggingface
- Deploy the model by using single-model serving with a serving runtime
- Test the model API

Exercise: Downloading an Open Source LLM

First we need to download the Google Flan model from HuggingFace and then store it in our S3 bucket for the model server to access.
We will use a workbench and the data connection that you created in the [Storage Data Connections](#) section..

Prerequisites

- You created the data connection My Storage.

-

- You added the My Storage data connection to your workbench.

-

Procedure

First, let's navigate to the relevant notebooks.

1. Navigate to lab-material/llm
1. In your notebook environment, open the file 1_download_save.ipynb
2. Follow the instructions directly in the notebook.
3. The instructions will guide you through the process of downloading the model from HuggingFace and then uploading it to your models bucket.
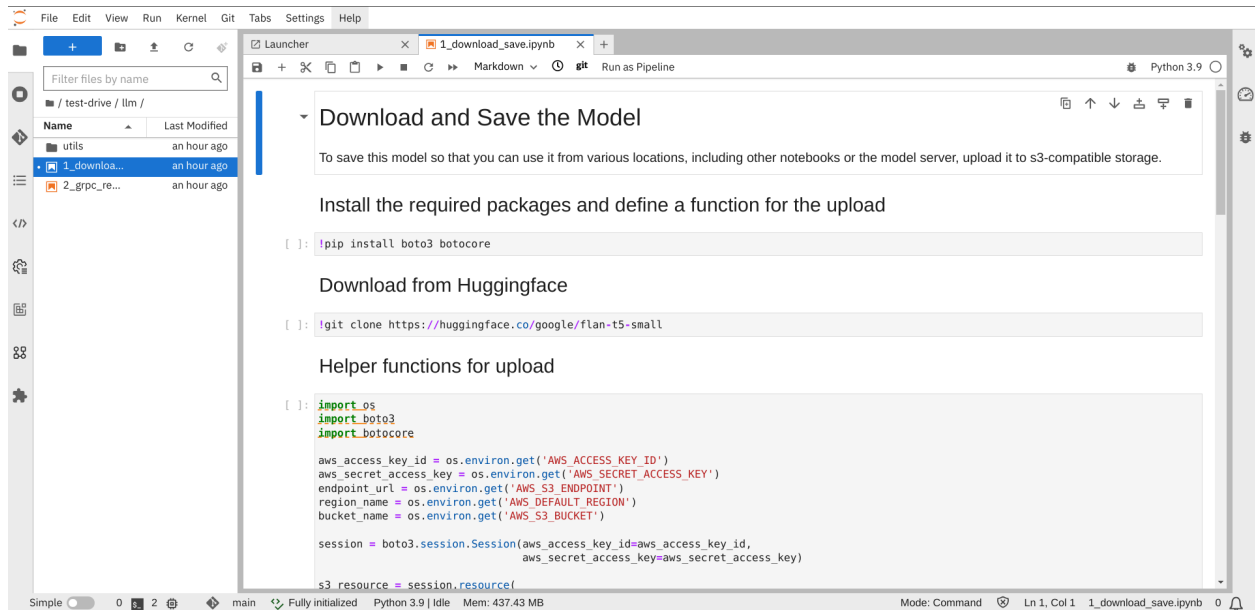
Verification

When you have completed the notebook instructions, you should see files listed in the directory/prefix: models/flan-t5-small
models/flan-t5-small/README.md
models/flan-t5-small/config.json
models/flan-t5-small/flax_model.msgpack
models/flan-t5-small/generation_config.json
models/flan-t5-small/model.safetensors
models/flan-t5-small/pytorch_model.bin
models/flan-t5-small/special_tokens_map.json
models/flan-t5-small/spiece.model
models/flan-t5-small/tf_model.h5
models/flan-t5-small/tokenizer.json
models/flan-t5-small/tokenizer_config.json

Exercise: Deploying a model

Now that the model is accessible in storage, you can deploy it as an API.
We will use a different runtime this time, called TGIS (Text Generation Inference Service), that's purpose made for deploying LLMs.

Procedure

1. In the OpenShift AI dashboard
2. Navigate to Models and model servers.
3. Click Deploy model.
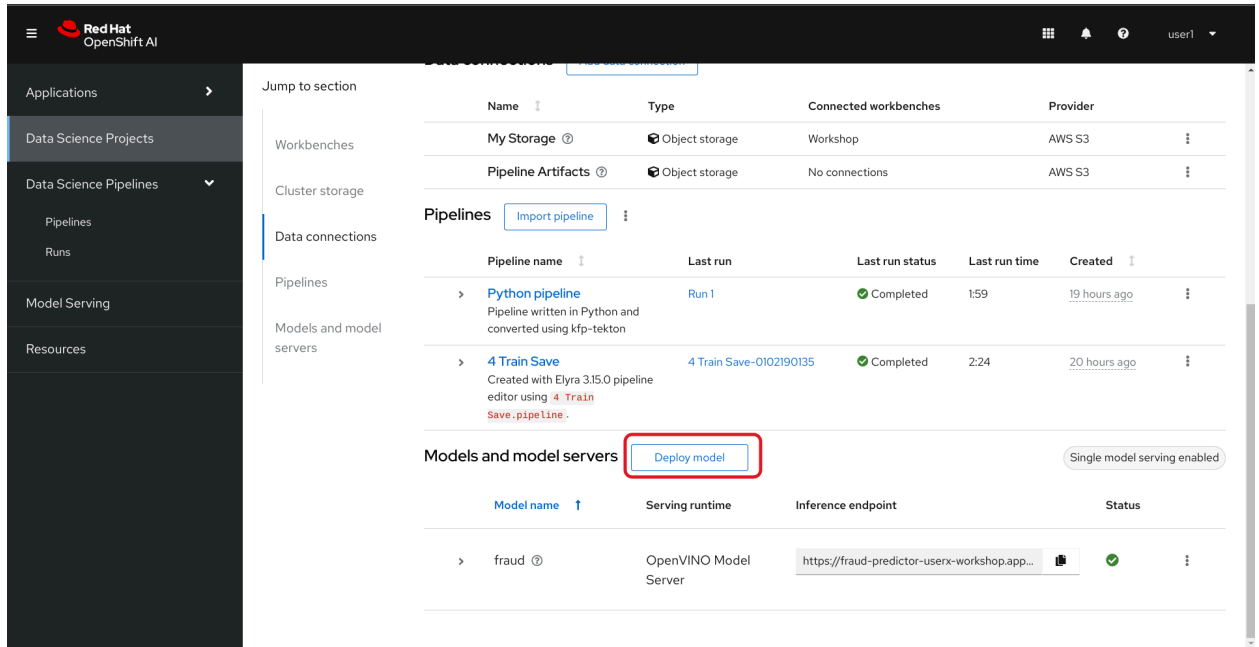
4.
5. In the form:
   a. Fill out the Model Name with the value flan-t5-small.
   b. Select the Serving runtime, Text Generation Inference Service.
   c. Select the Model framework, pytorch.
   d. Set the Model server replicas to 1.
   e. Select the Model Server size, Lab Custom Small.
   f. Select the Existing data connection: My Storage
   g. Enter the path to your uploaded model: models/flan-t5-small

Deploy model

Configure properties for deploying your model

Project

UserX Workshop

Model name *

flan-t5-small

Serving runtime *

Text Generation Inference Service ▾

Model framework (name – version) *

pytorch ▾

Model server replicas

Number of model server replicas to deploy

− 1 +

Compute resources per replica

Model server size

Lab Custom Small ▾

Accelerator ⓘ

None ▾

Model location

◉ Existing data connection

Name *

My Storage ▾

Path *

/ models/flan-t5-small

Enter a path to a model or folder. This path cannot point to a root folder.

○ New data connection

Deploy **Cancel**

6.

7. Click Deploy.

8. Wait for the model to deploy and for the Status to show a green checkmark.
9. This will probably take two or three minutes.

10.

At this point, the model should be served, and we now just need to confirm it responds to queries.

Exercise: Testing the model API

Now that you've deployed the model, you can test its API endpoints.
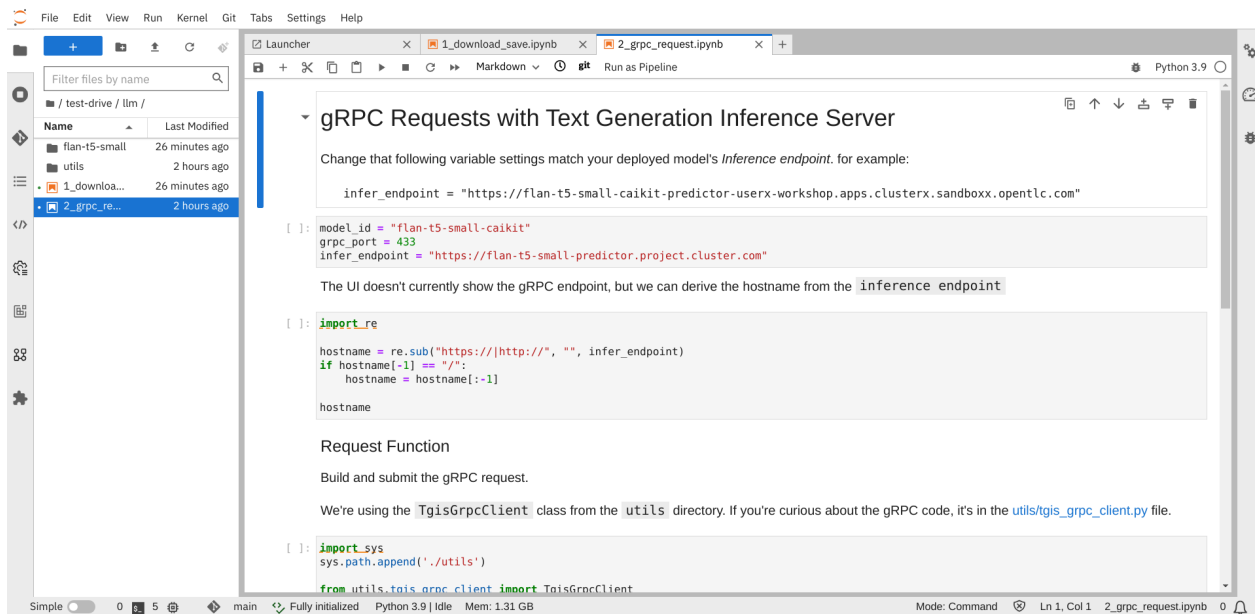
Procedure

1. In the Data Science dashboard
2. Navigate to the project details page
3. Scroll down to the Models and model servers section.
4. Take note of the model's Inference endpoint. You need this information when you test the model API.

5.
6. Return to the Jupyter environment

7. Open the file called 2_grpc_request.ipynb

8.

9. Read the code and follow the instructions.

10.   Play around with a few different responses and see how well it performs. Remember that this is a relatively small model, and as the models get larger it will perform better.