Cloudera ACE Machine Learning Workshop

Customer Churn Hands On Labs Guide

Objective

In this exercise we will implement an end-to-end machine learning workflow and cover:

- Data ingest
- Data exploration
- Model building and training
- Model serving
- Business applications and model operations

Business Use Case

In this Hands On Lab you will create a model to predict customer churn and present model-driven insights to your stakeholders.

You will use an interpretability technique to make your otherwise “black box model” explainable in an interactive dashboard. A mathematical explanation is beyond the scope of this lab but if you are interested in learning more we recommend the Fast Forward Labs Report on Model Interpretability.

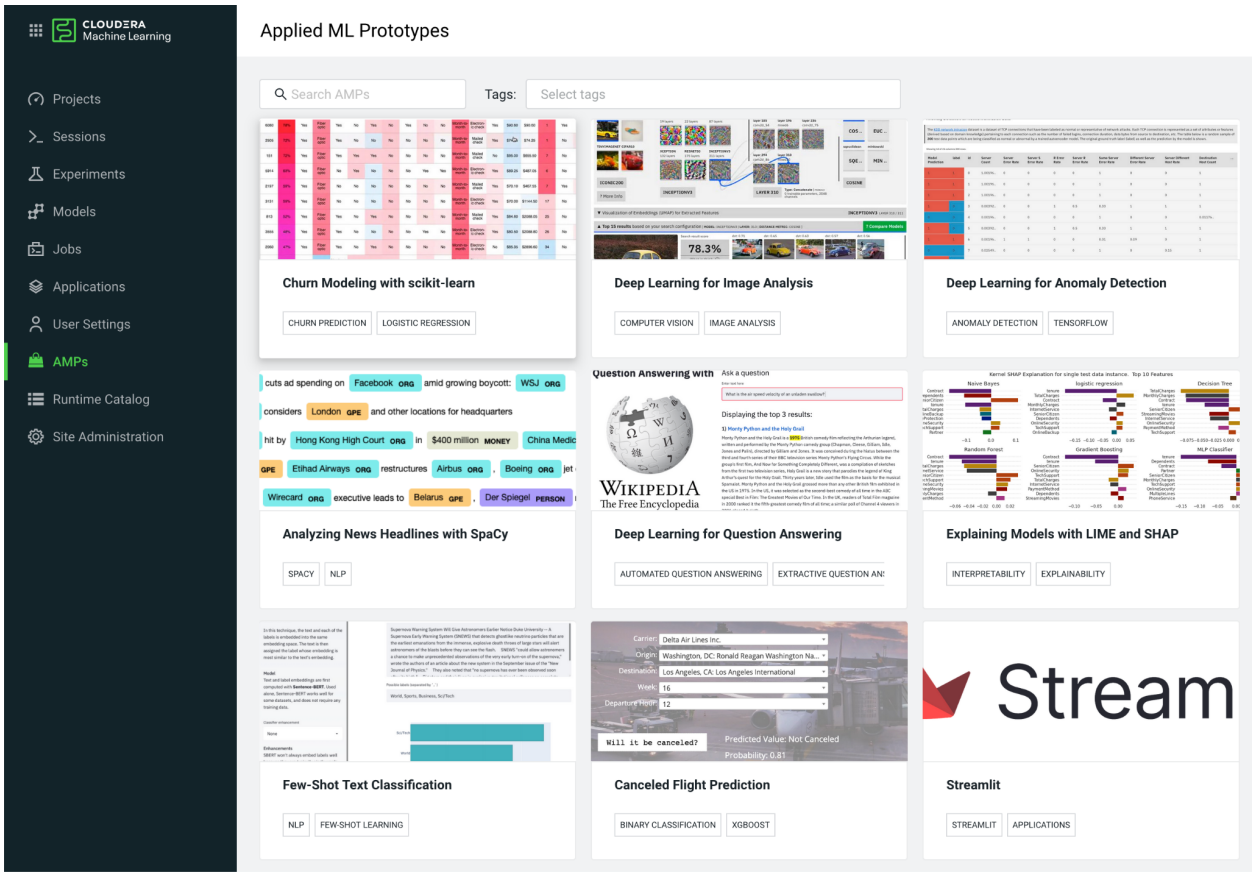
Finally, you will use basic ML Ops techniques to productionize and monitor your model.

Index

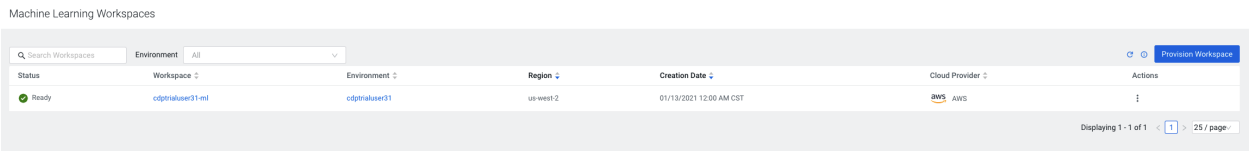
Step by Step Instructions

Part 1: Configure and deploy the Workshop Content as an AMP

AMPs (Applied Machine Learning Prototypes) are reference Machine Learning projects that have been built by Cloudera Fast Forward Labs to provide quickstart examples and tutorials. AMPs are deployed into the Cloudera Machine Learning (CML) experience, which is a platform you can also build your own Machine Learning use cases on.

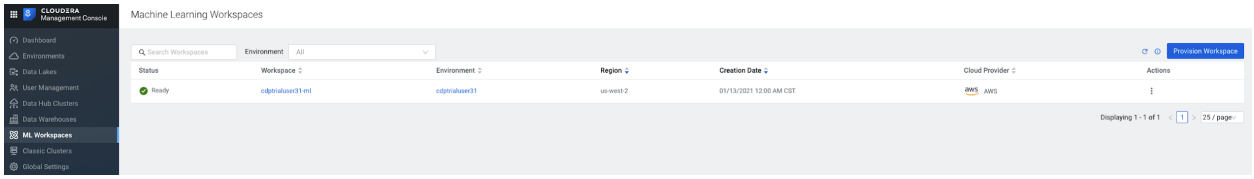


Navigate to the Machine Learning tile from the CDP Menu. Workspaces are the heart of the Cloudera Machine Learning (CML).

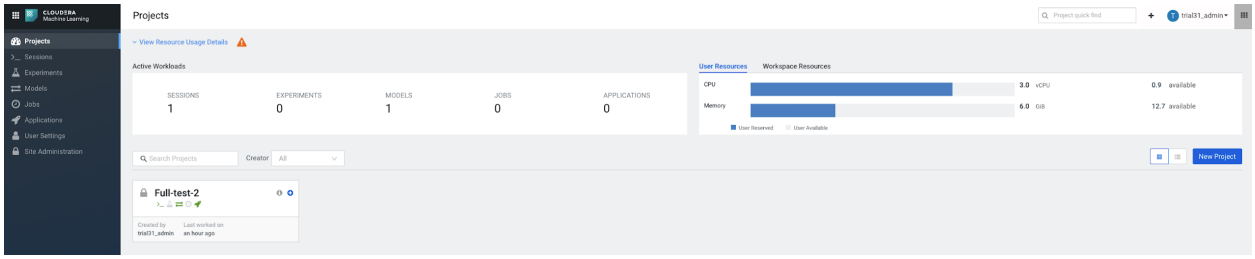


A Workspace is a cluster that runs on a kubernetes service to provide teams of data scientists to develop, test, train, and ultimately deploy machine learning models. It is designed to deploy a small number of infra resources and then autoscale compute resources as needed when end users implement more workloads and use cases.

Click into the Workspace by clicking the Workspace name.

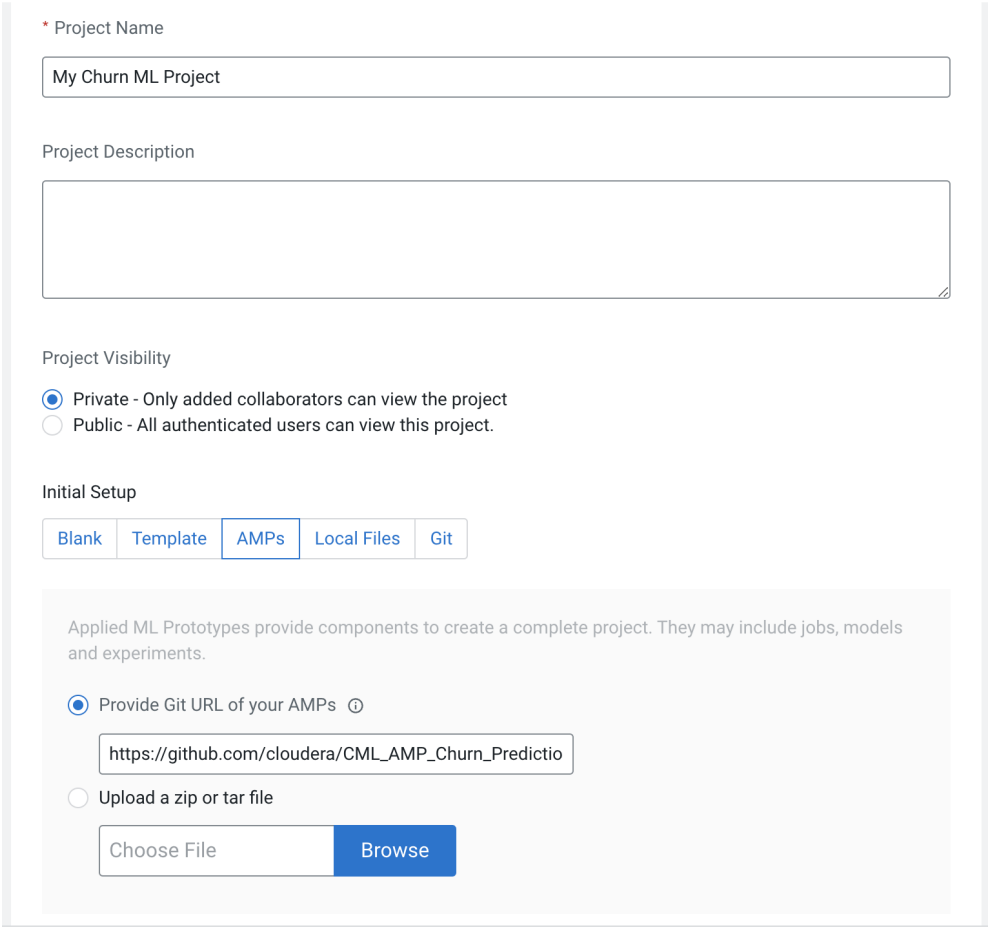


You can visualize all of the Projects and Resources that are part of the Projects page. Next we will create a Project where we will deploy an AMP. Click on “New Project”

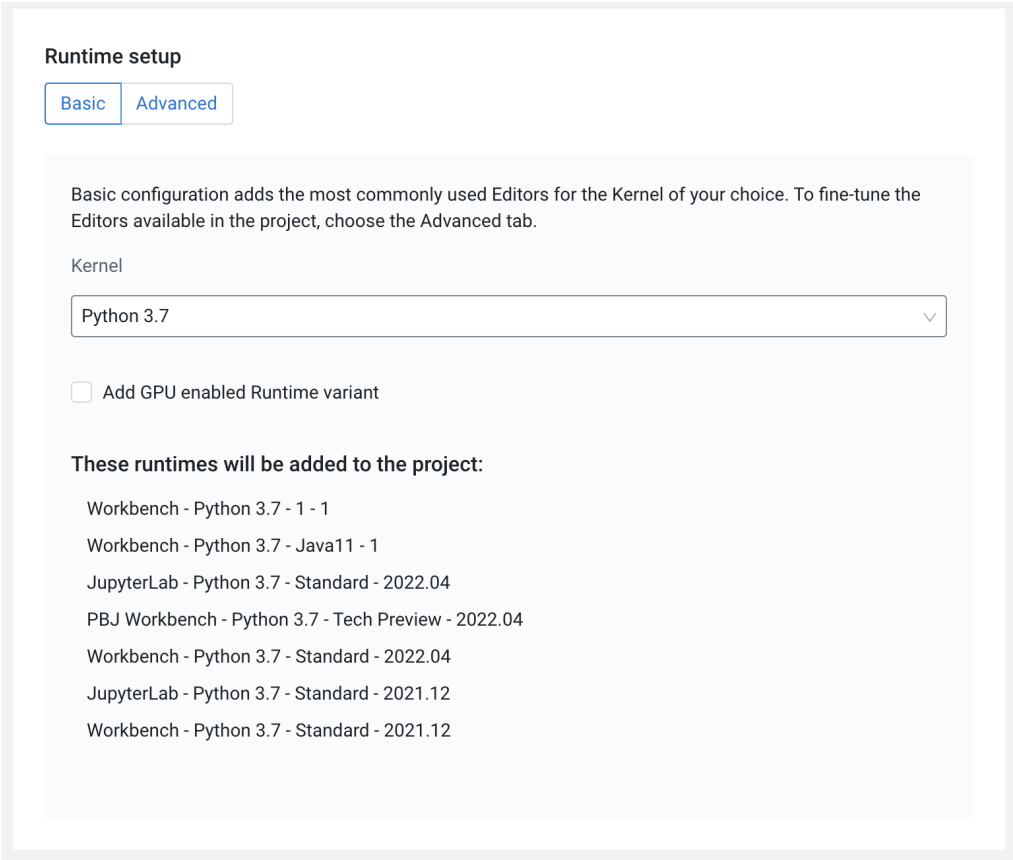


Create a new Project as shown below:

Project Name: Username Churn Project
Project Visibility: Private
Initial Setup: AMP’s
Git URL: https://github.com/cloudera/CML_AMP_Churn_Prediction.git



Scroll down and make sure you select the “Basic” runtime with the Python 3.7 kernel



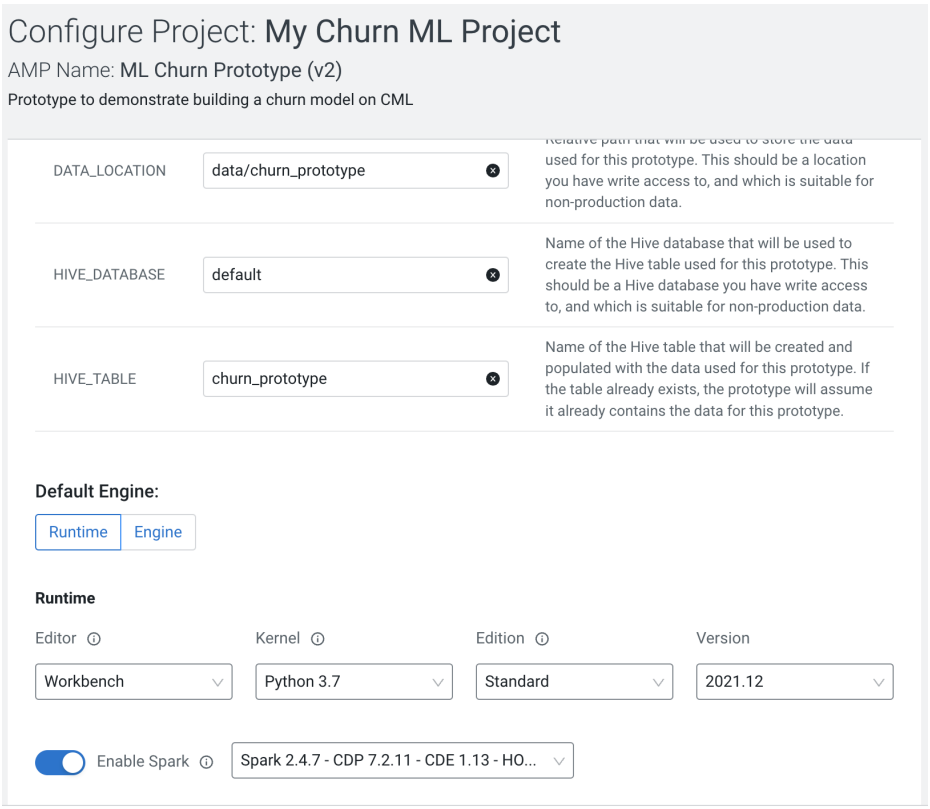
Next, in order to deploy the AMP we will provide a few Environment Variables used throughout the telco churn Project.

Configure Project with the following:

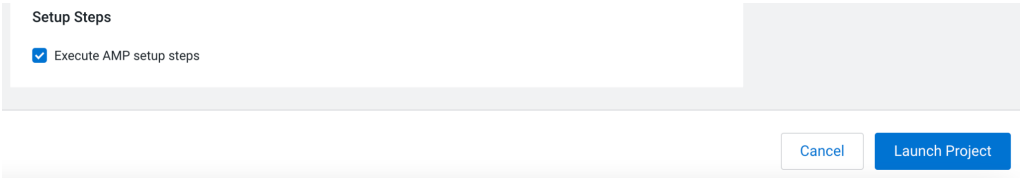
DATA_LOCATION: data/churn_prototype
HIVE_DATABASE: default
HIVE_TABLE: churn_prototype_username
Default Engine: Runtime
Confirm ‘Execute AMP setup steps’ is checked

The latest version of the AMP has been tested for CML Runtimes with Python 3.7 and Spark 2.4.7. If the workspace does not have the exact runtime that was tested you may get a warning.

However, you can still deploy the project with other runtimes. For example, you can deploy the project with Spark 2.4.7, CDP 7.2.11, CDE 1.13 HotFix 2.

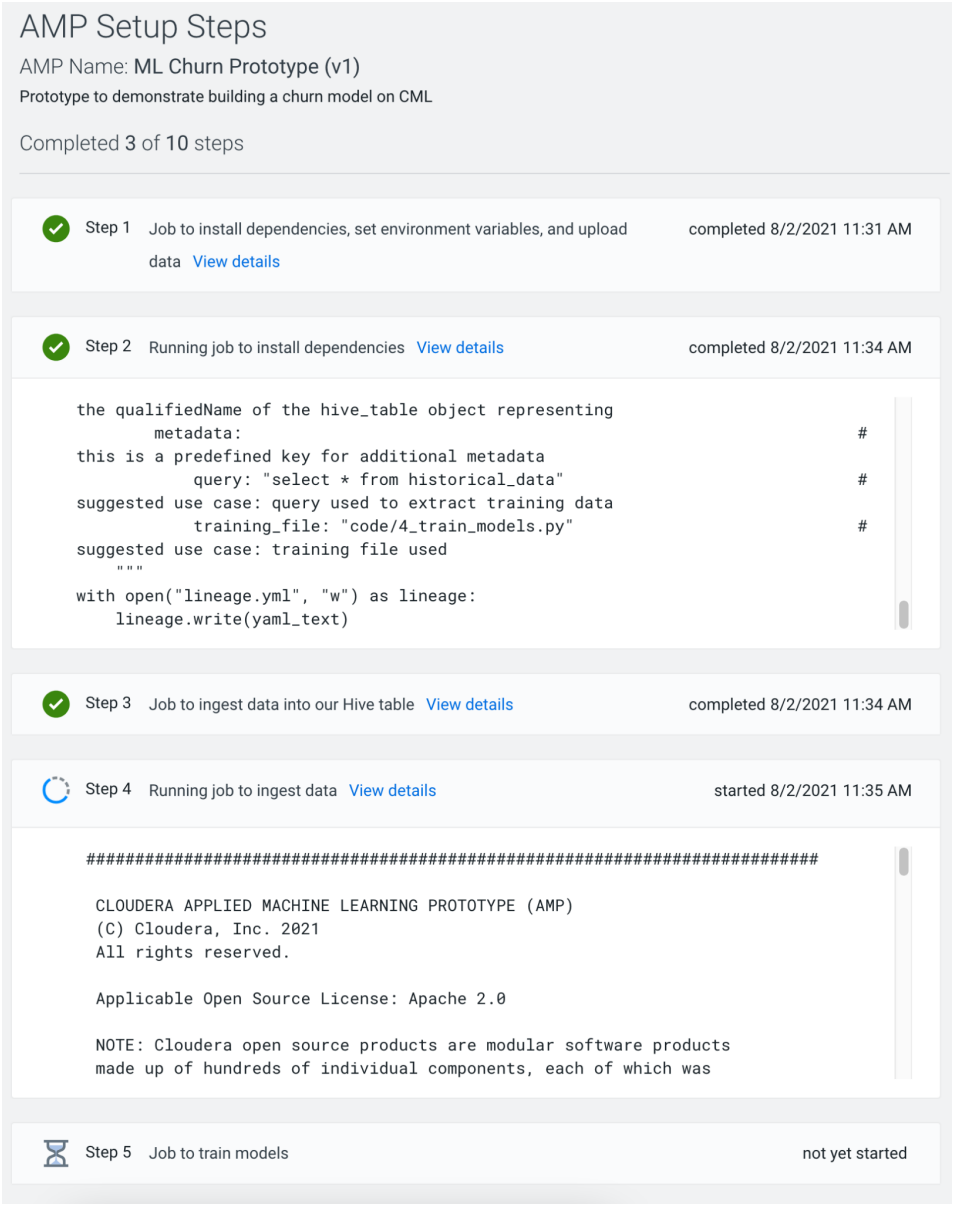


Launch Project



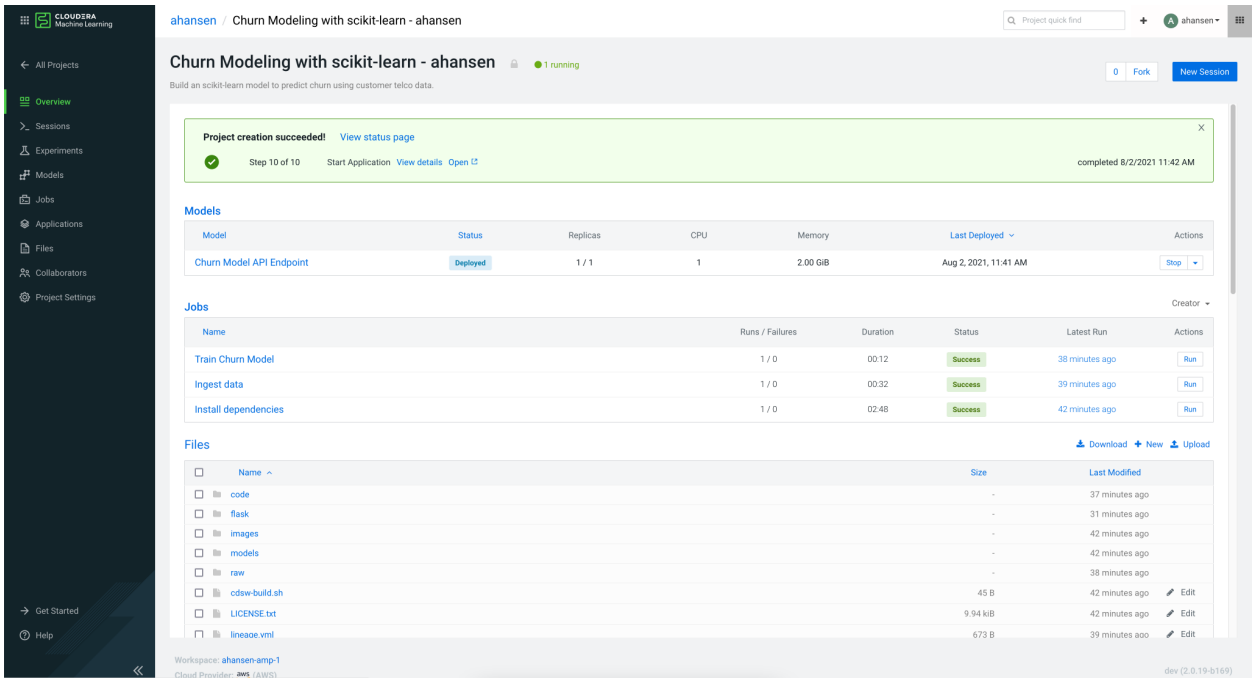
Notice the AMP process consists in a number of automated steps for project setup. These include cloning all project artifacts from the public GitHub repository, setting Project environment variables, and deploying all CML Sessions, Jobs, etc in order for the Project setup steps to complete. The process is powered by a simple yaml file and everyone can build their own AMP by following these instructions.

The AMP Setup process should take 10 minutes. This is a great time to navigate back to the Workspace homepage and explore more AMPs or, if you prefer, to simply grab a cup of coffee.

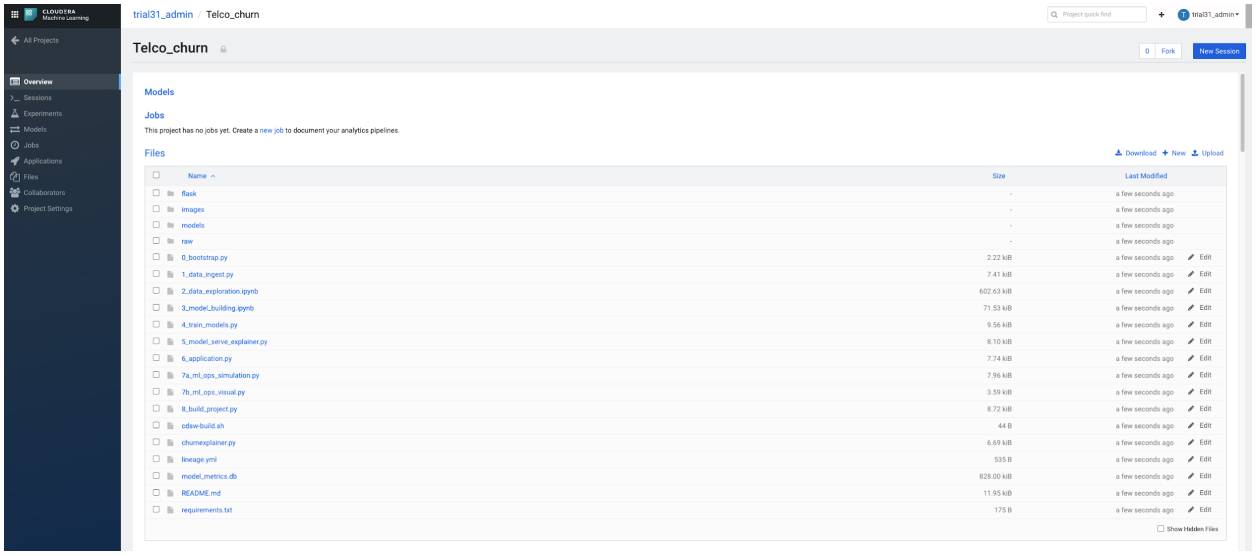


Part 2: Churn Model Project Overview

With the AMP used to create our CML Project, we can now explore what was deployed for us.

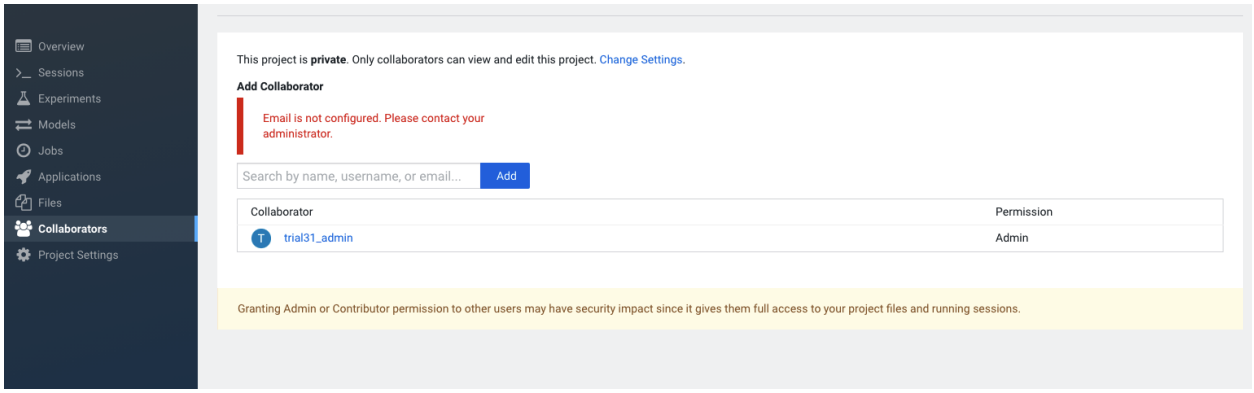


Overview gives you access to all the features of a CML project. We will only have files copied from the Github repo currently. Initially it is good to start on the management components of a project.



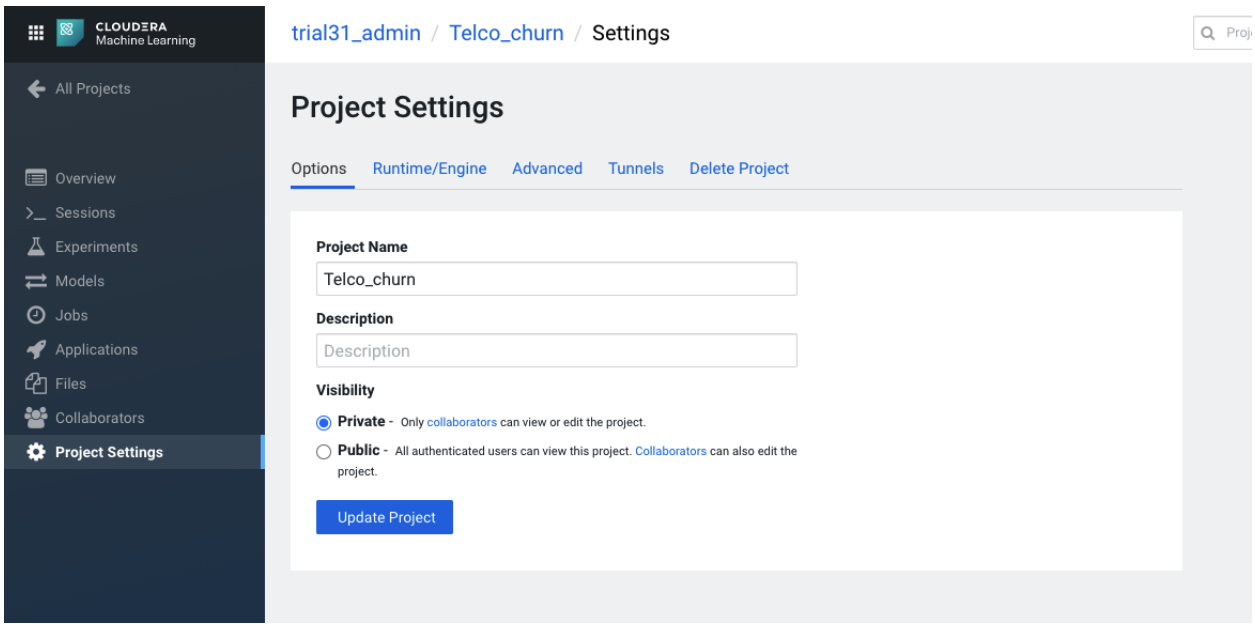
Collaborators:
For our demo we aren't adding additional collaborators.

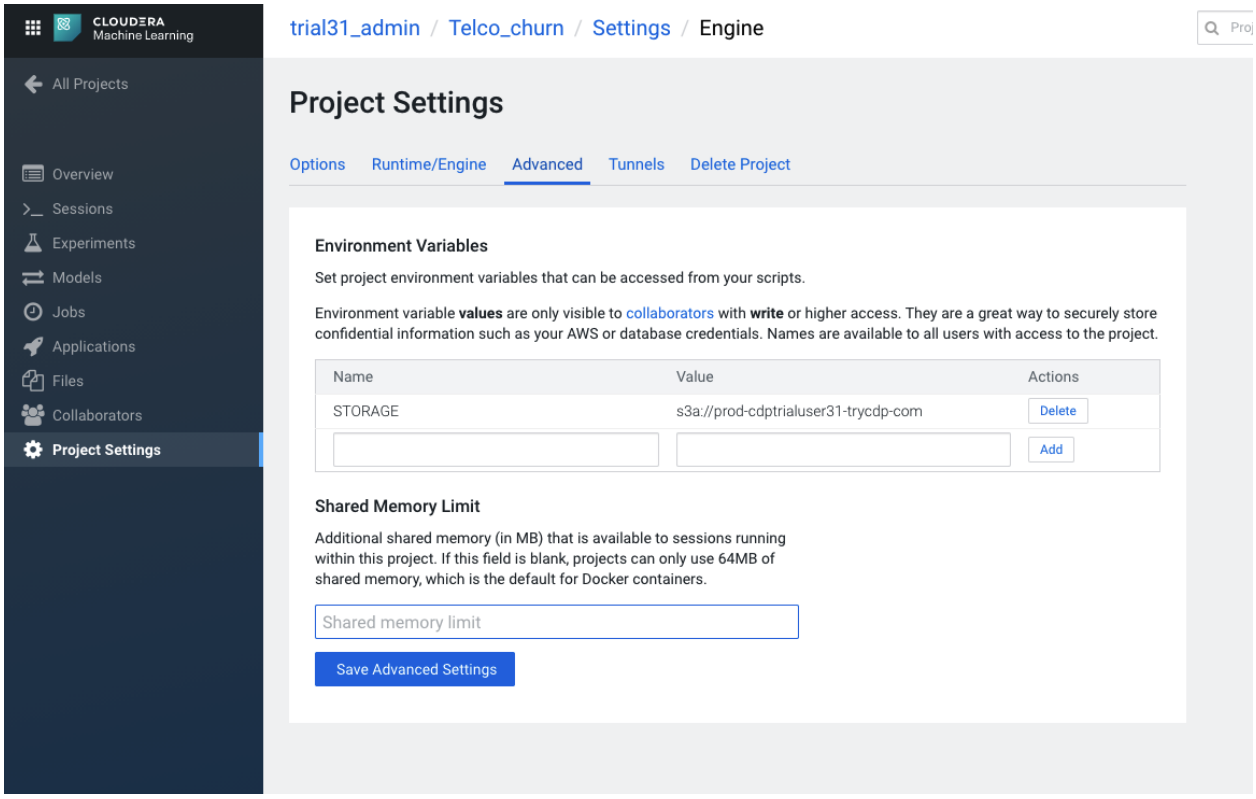
You can give access to other users with certain permissions for the encompassing project so teams of users can collaborate together. You can set up Admins, Contributor, Operator, and Viewer permissions.

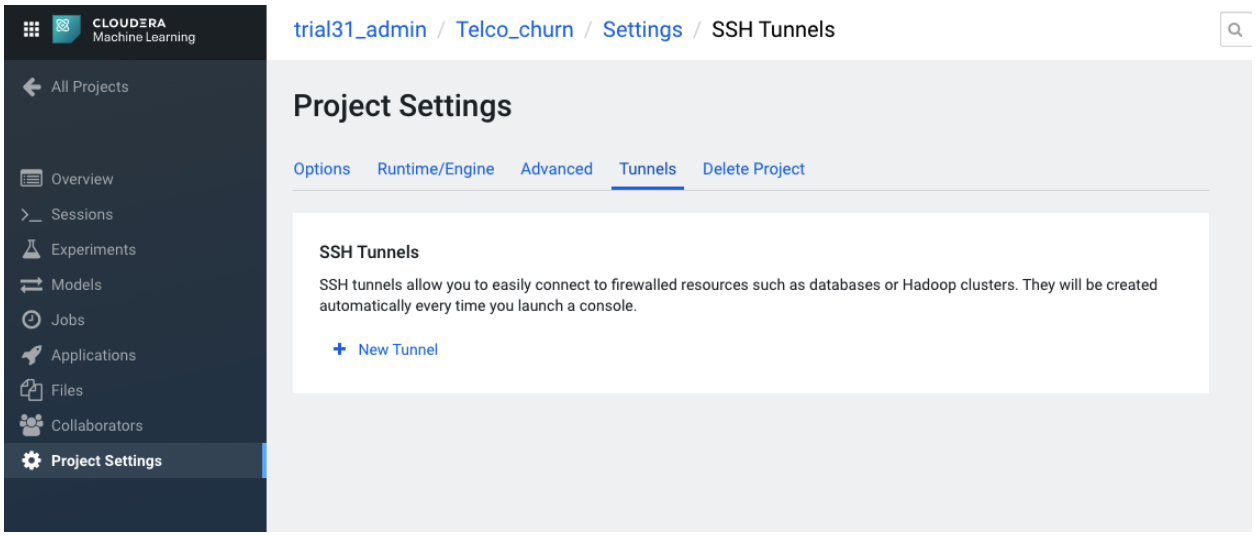


Project Settings:
Taking a look at Project Settings, this is where you can define several options for the current project. You have the ability to define different engines where your code in CML will run. There are project variables that can be defined and used throughout your code. SSH tunnels can also be configured to connect to other services as needed. More details can be found in our docs here.

Please do not change the default Project Settings



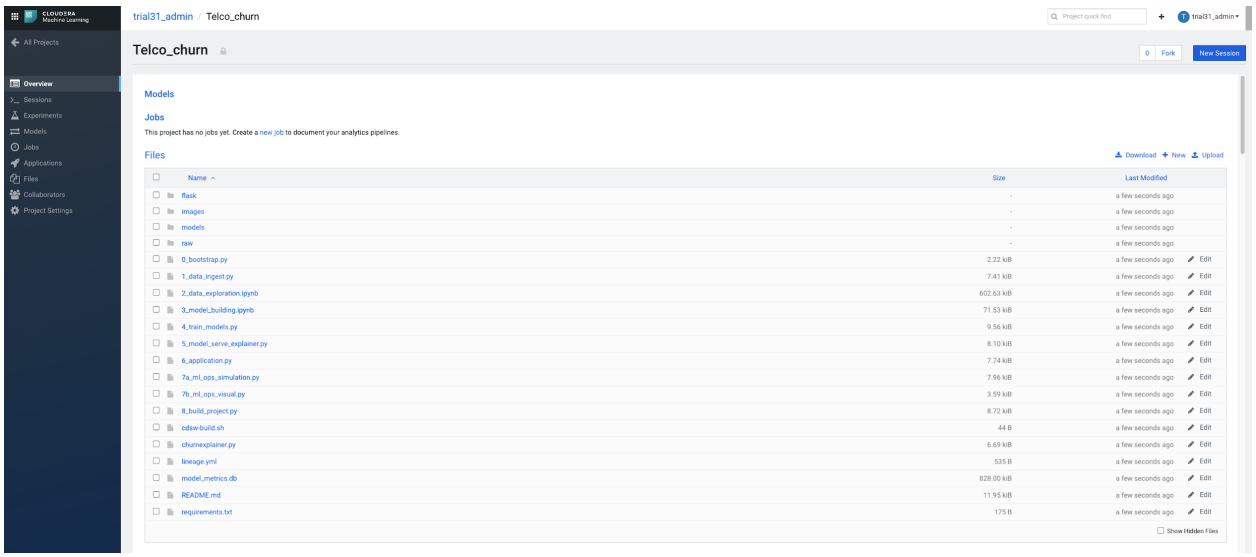

CLOUDERA
Machine Learning
All Projects
Overview
Sessions
Experiments
Models
Jobs
Applications
Files
Collaborators
Project Settings
trial31_admin / Telco_churn / Settings / Engine
Project Settings
Options
Runtime/Engine
Advanced
Tunnels
Delete Project
Environment Variables
Set project environment variables that can be accessed from your scripts.
Environment variable values are only visible to collaborators with write or higher access. They are a great way to securely store confidential information such as your AWS or database credentials. Names are available to all users with access to the project.
Name
Value
Actions
STORAGE
s3a://prod-cdptrialuser31-trycdp-com
Delete
Add
Shared Memory Limit
Additional shared memory (in MB) that is available to sessions running within this project. If this field is blank, projects can only use 64MB of shared memory, which is the default for Docker containers.
Shared memory limit
Save Advanced Settings


CLOUDERA
Machine Learning
All Projects
Overview
Sessions
Experiments
Models
Jobs
Applications
Files
Collaborators
Project Settings
trial31_admin / Telco_churn / Settings / SSH Tunnels
Project Settings
Options
Runtime/Engine
Advanced
Tunnels
Delete Project
SSH Tunnels
SSH tunnels allow you to easily connect to firewalled resources such as databases or Hadoop clusters. They will be created automatically every time you launch a console.
New Tunnel

Part 3: CML Sessions and Workbench

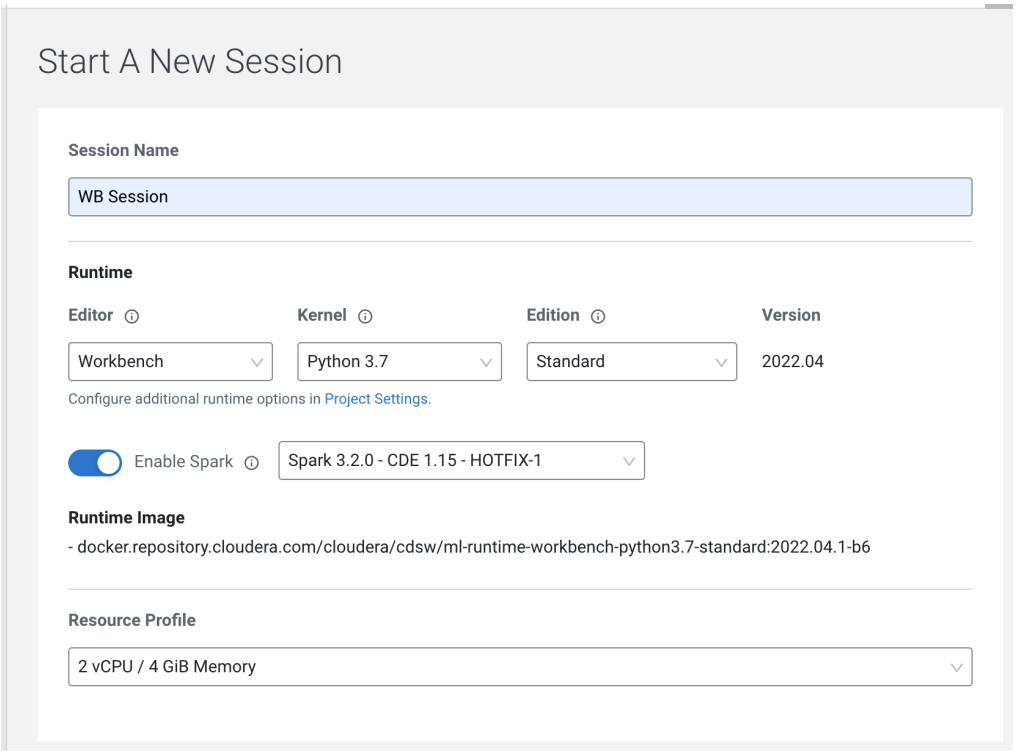
Sessions allow you to perform actions such as run R, Scala or Python code. They also provide access to an interactive command prompt and terminal. Sessions will be built on a specified Engine Image, which is a docker container that is deployed onto the Workspace. In addition you can specify how many resources are used per session.

From the Overview page click on New Session



Session Name: telco_churn_session_1
Editor: Workbench
Kernel: Python 3.7
Resource Profile: 1vCPU/2 GiB Memory
Runtime Edition: Standard
Runtime Version: Any available version
Enable Spark Add On: enable any Spark version

Then select Start Session



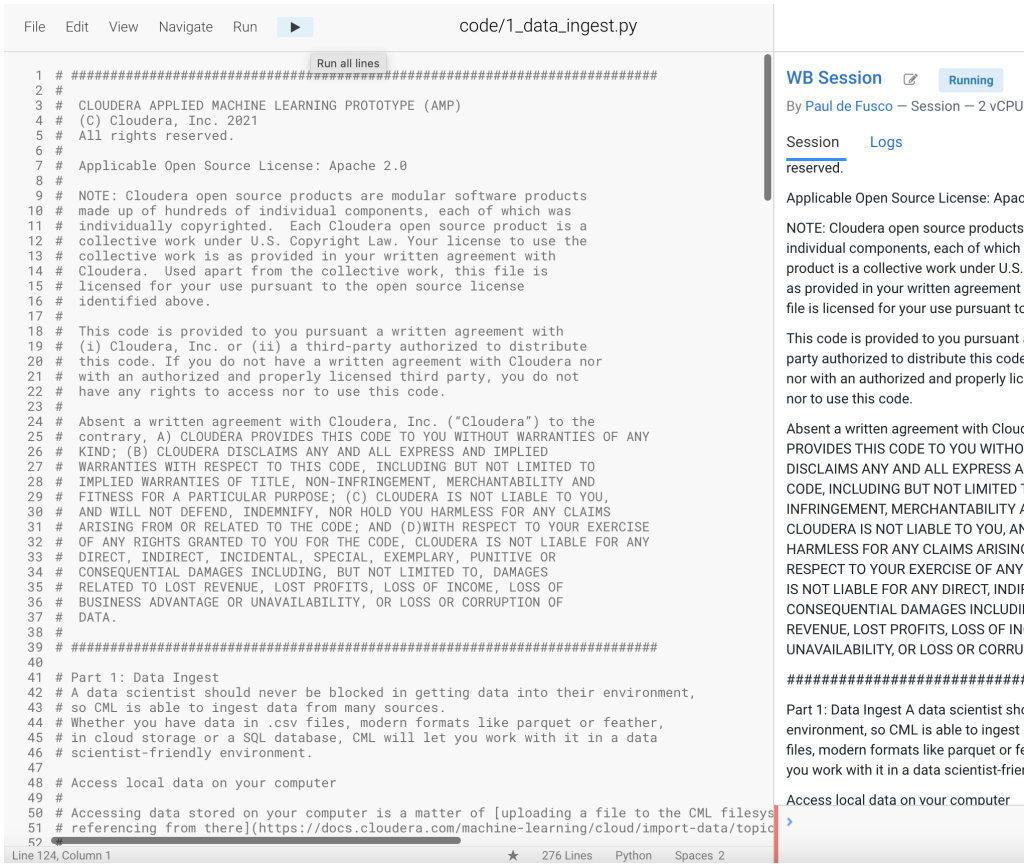
The Workbench is now starting up and deploying a container onto the workspace at this point. Going from left to right you will see the project files, editor pane, and session pane.

Once you see the flashing red line on the bottom of the session pane turn steady green the container has been successfully started.

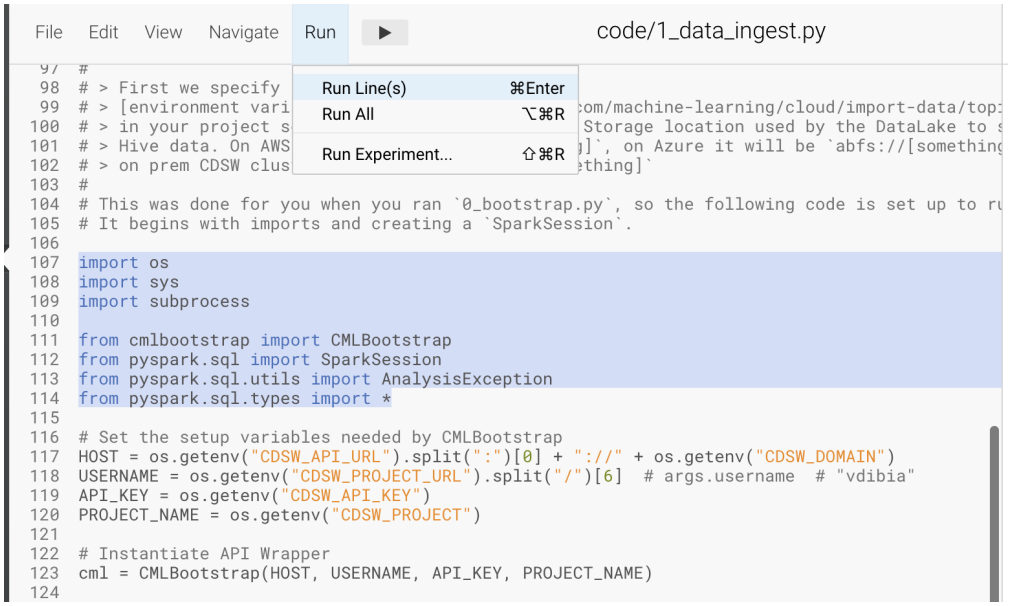
Script 1: Ingest Data

In this script you will ingest a raw csv file in a Spark Dataframe. The script has a .py extension and therefore is ideally suited for execution with the Workbench editor. No modifications to the code are required and it can be executed as is.

- You can execute the entire script in bulk by clicking on the “play icon” on the top menu bar. Once you do this you should notice the editor bar switches from green to red.



- As an alternative you can select subsets of the code and execute those only. This is great for troubleshooting and testing. To do so, highlight a number of lines of code from your script and then click on “Run” -> “Run Lines” from the top menu bar.



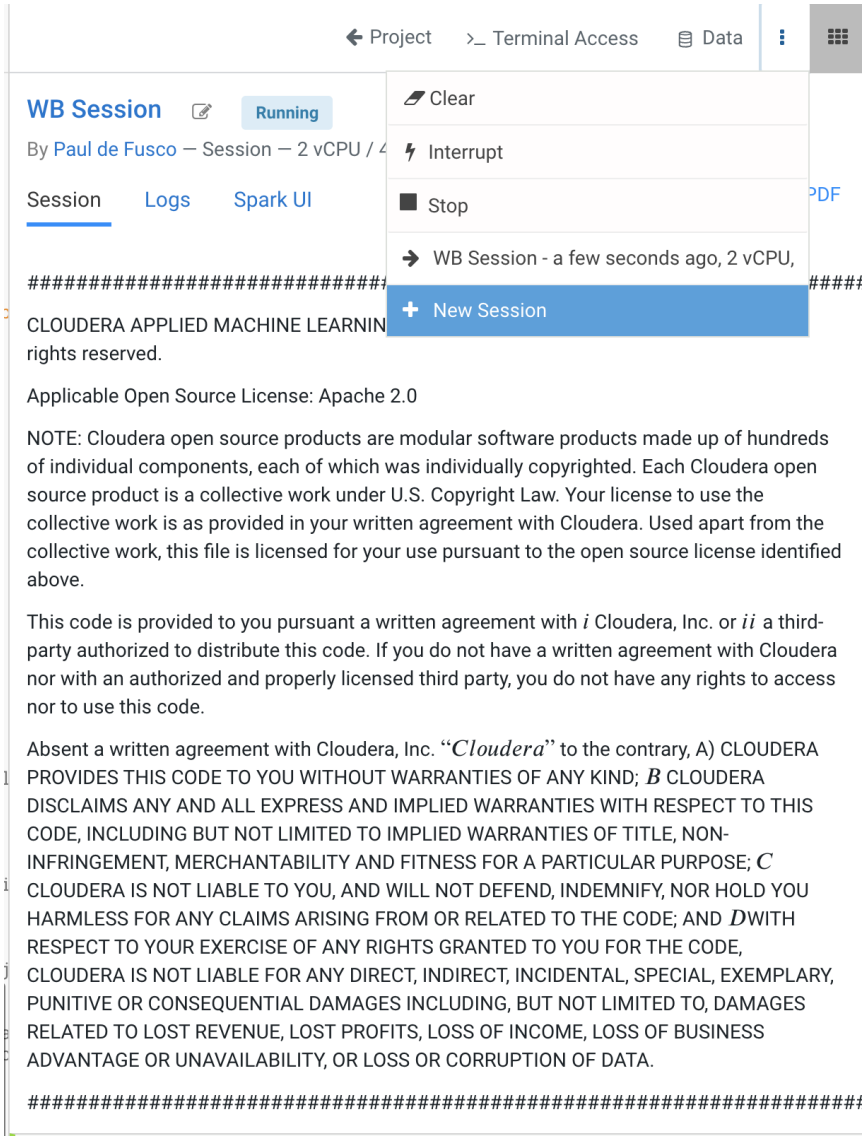
The code is explained in the script comments. However, here are a key few highlights:

- Because CML is integrated with SDX and CDP, you can easily retrieve large datasets from Cloud Storage (ADLS, S3, Ozone) with a simple line of code
- Apache Spark is a general purpose framework for distributed computing that offers high performance for both batch and stream processing. It exposes APIs for Java, Python, R, and Scala, as well as an interactive shell for you to run jobs.
- In Cloudera Machine Learning (CML), Spark and its dependencies are bundled directly into the CML engine Docker image.
- Furthermore, you can switch between different Spark versions at Session launch.

Notebooks 2 and 3: Interactive Analysis and Model Building with JupyterLab

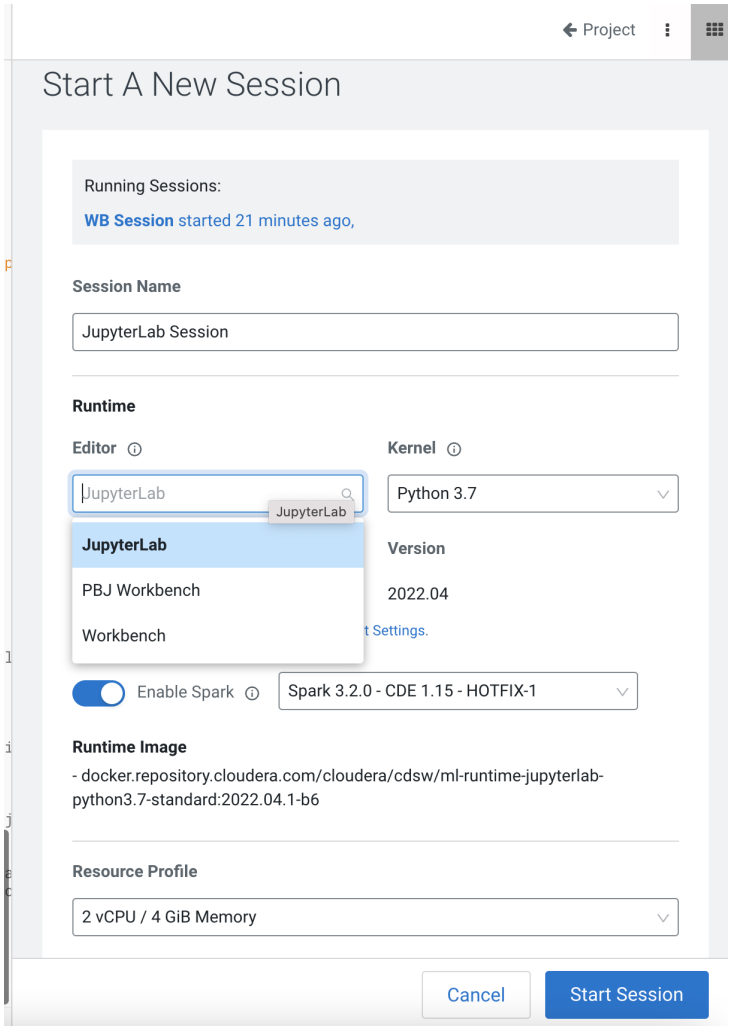
In the previous section you loaded a csv file with a python script. In this section you will perform more Python commands with Jupyter Notebooks. Notebooks have a ".ipynb" extension and need to be executed with a Session using the JupyterLabs editor.

Launch a new session by selecting on the three "vertical dots" on the right side of the top menu bar.

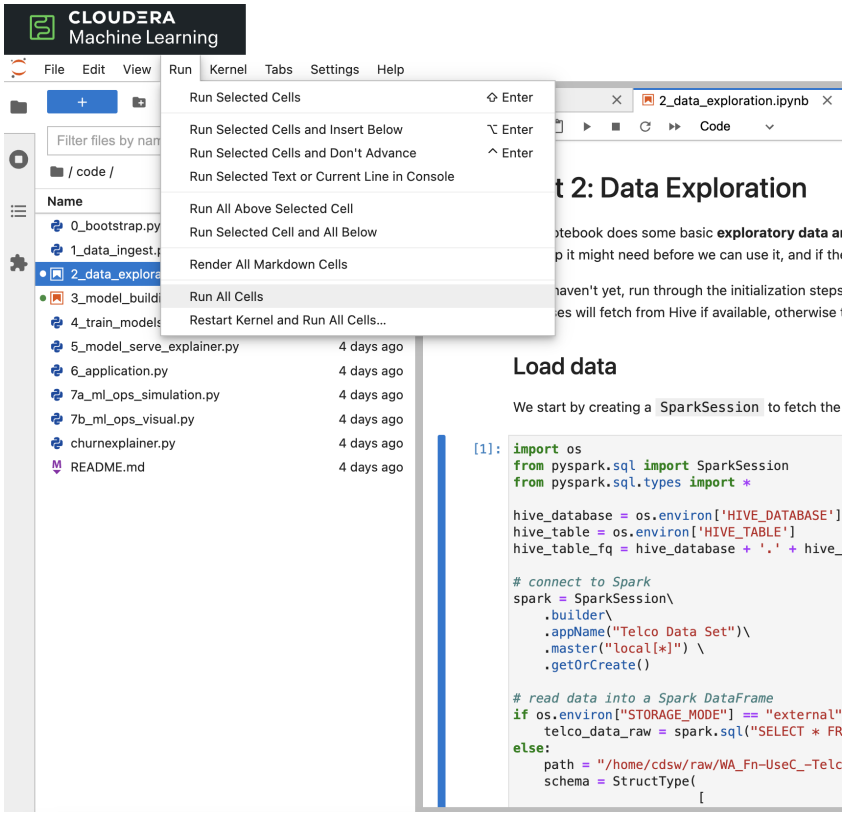


Launch the new Session with the following settings:

Session Name: telco_churn_session_2
Editor: JupyterLab
Kernel: Python 3.7
Resource Profile: 1vCPU/2 GiB Memory
Runtime Edition: Standard
Runtime Version: Any available version
Enable Spark Add On: enable any Spark version



After a few moments the JupyterLab editor should have taken over the screen. Open Notebook “2_data_exploration.ipynb” from the left side menu and investigate the code. Notebook cells are meant to be executed individually and give a more interactive flavor for coding and experimentation.



As before, no code changes are required and more detailed instructions are included in the comments. There are two ways to run each cell.

- Click on the cell you want to run. Hit “Shift” + “Enter” on your keyboard. Use this approach if you want to execute each cell individually.
- Open the “Run” menu from the top bar and then select “Run All”. Use this approach if you want to execute the entire notebook in bulk.

When you are finished with notebook “2_data_exploration.ipynb” go ahead and move on to notebook “3_model_building.ipynb”. As before, no code changes are required.

In summary, this section outlined the following concepts:

- With CML Runtimes, you can easily switch between different editors and work with multiple editors or programming environments in parallel if needed.
- First you stored a Spark Dataframe as a Spark table in the “1_ingest_data.py” python script using the Workbench editor.
- Then you retrieved the data in notebook “2_data_exploration.ipynb” using a JupyterLab session via Spark SQL. Spark SQL allows you to easily exchange files across sessions.
- Your Spark table was tracked as Hive External Tables and automatically made available in Atlas, the Data Catalog, and CDW. This is powered by SDX integration and requires no work on the CDP Admin or Users. We will see more on this in Part 7.

- Finally, in notebook “3_model_building.ipynb” you created a model with SciKit Learn and Lime, and then stored it in your project. Optionally, you could have saved it to Cloud Storage.
- CML allows you to work with any other libraries of your choice. This is the power of CML… any open source library and framework is one pip install away.

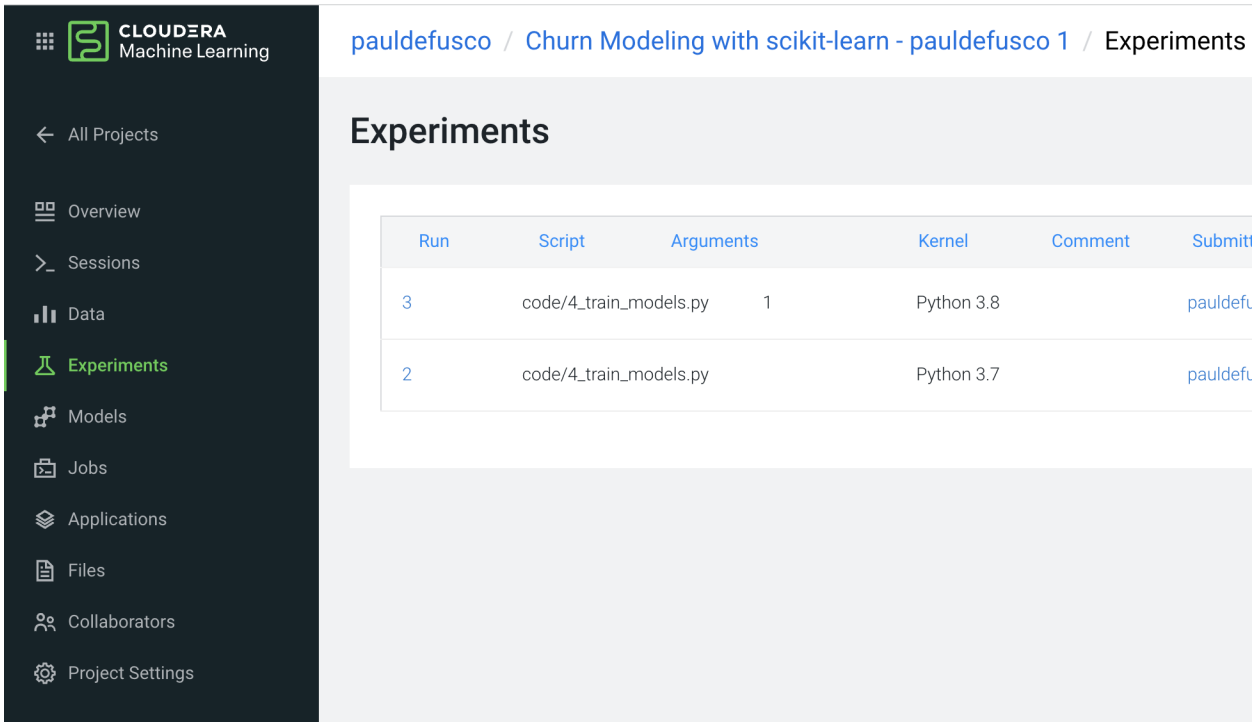
Part 4: CML Jobs and Experiments

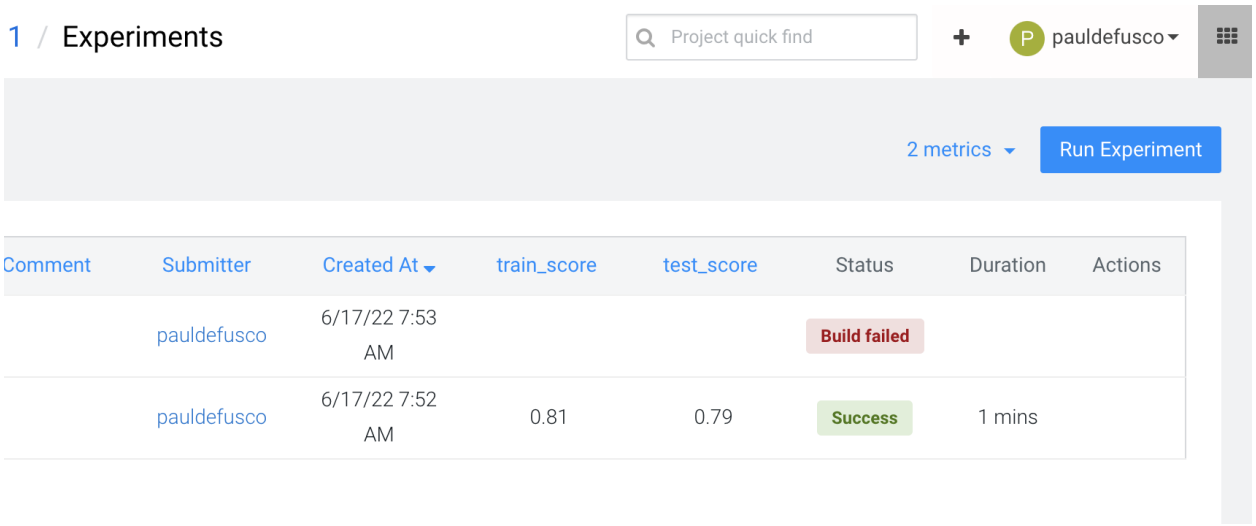
Sessions allow you to perform actions such as run R or Python code and are ideal for interactive development. Experiments and Jobs also run code in isolated containers but provide additional capabilities. They are key tools to automation and streamlining of ML activities and a cornerstone of ML Ops.

- With Experiments you can streamline the execution of your batch script. This makes it the tool of choice for hyperparameter tuning or to simplify session management when you are iterating with many versions of the same model.
- With Jobs you can schedule and orchestrate the execution of your batch scripts. You can use cron expressions or build chains of dependencies with other jobs. This makes it the tool of choice for processing Data Pipelines as well as automating A/B tests and other ML Ops related tasks.

Script 4: Running scripts as an Experiment

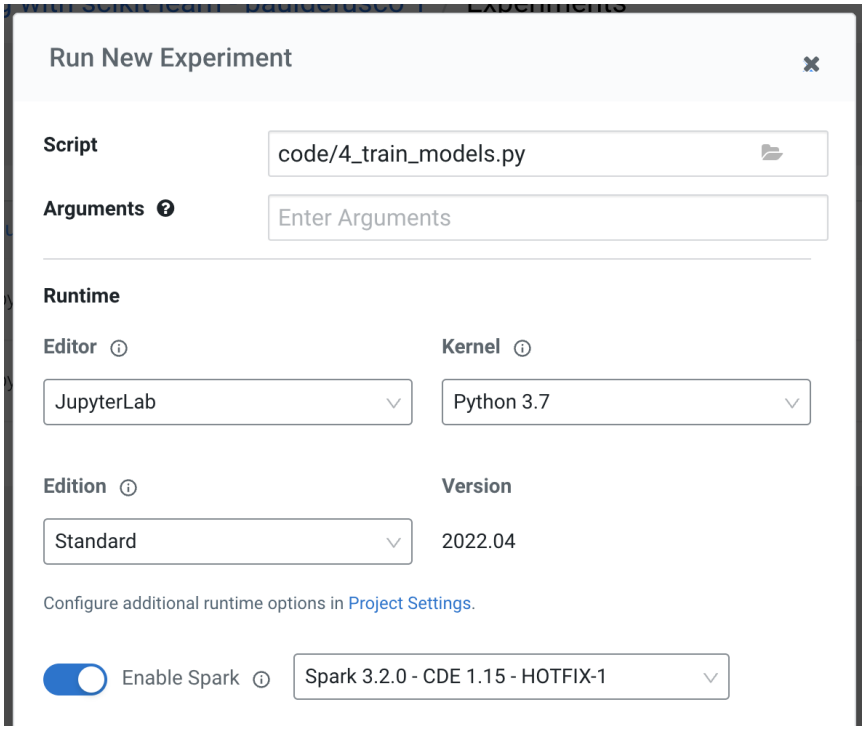
From the CML Project home page navigate to the Experiments tab. Then click on the “New Experiment” icon on the top right.





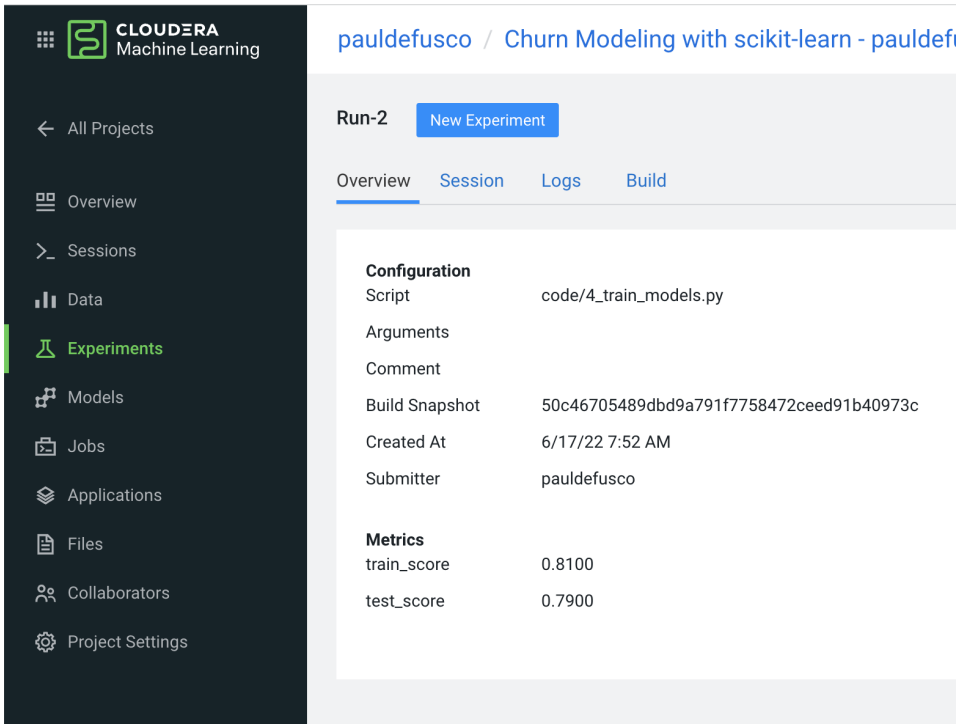
Similarly to Sessions, the form allows you to select runtime options to tailor resources to your needs. Select script "4_train_model.py" from the code folder and make sure to enable the Spark Addon. Any Spark version is fine.

Experiment Name: train_model_experiment
Editor: Workbench
Kernel: Python 3.7
Resource Profile: 1vCPU/2 GiB Memory
Runtime Edition: Standard
Enable Spark Add On: enable any Spark version



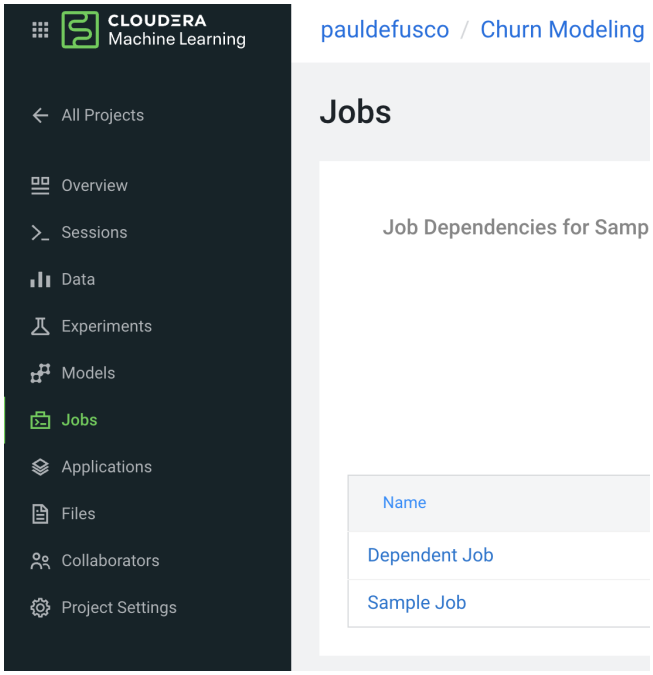
Launch the experiment and observe the status will change as CML automatically builds, deploys and executes the script in an isolated container.

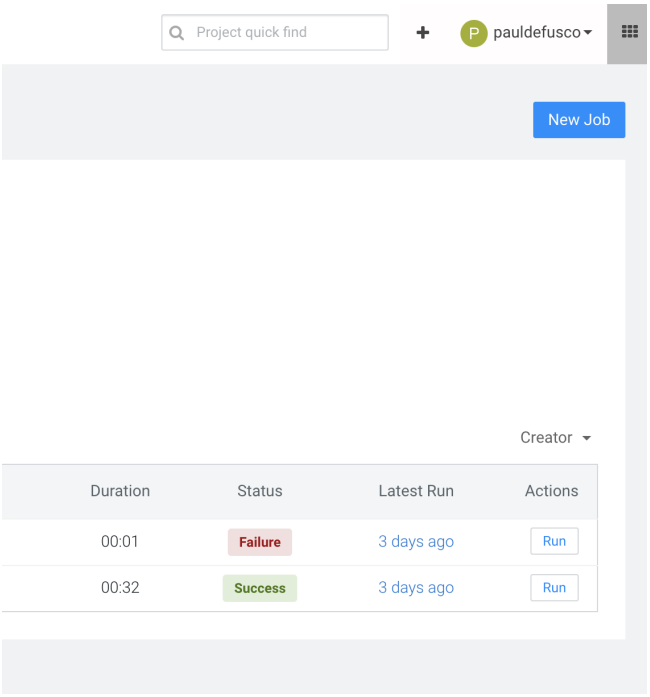
When execution has completed, open the experiment by clicking on the ID value from the Experiments landing page. From the Overview tab inspect results. It looks like this Experiment yielded an Accuracy of 81% and 79% on the training and test sets respectively.



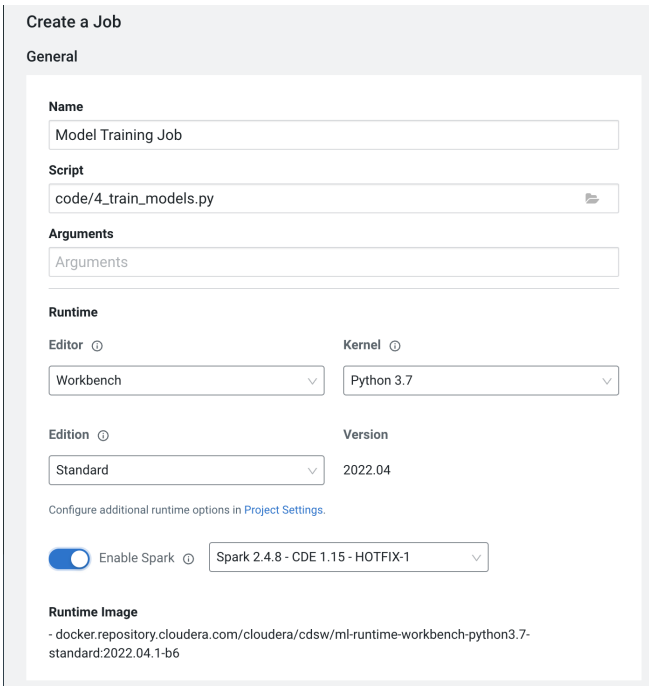
Script 4: Running scripts as a Job

Navigate to the CML Jobs page and select “New Job” on the top right.



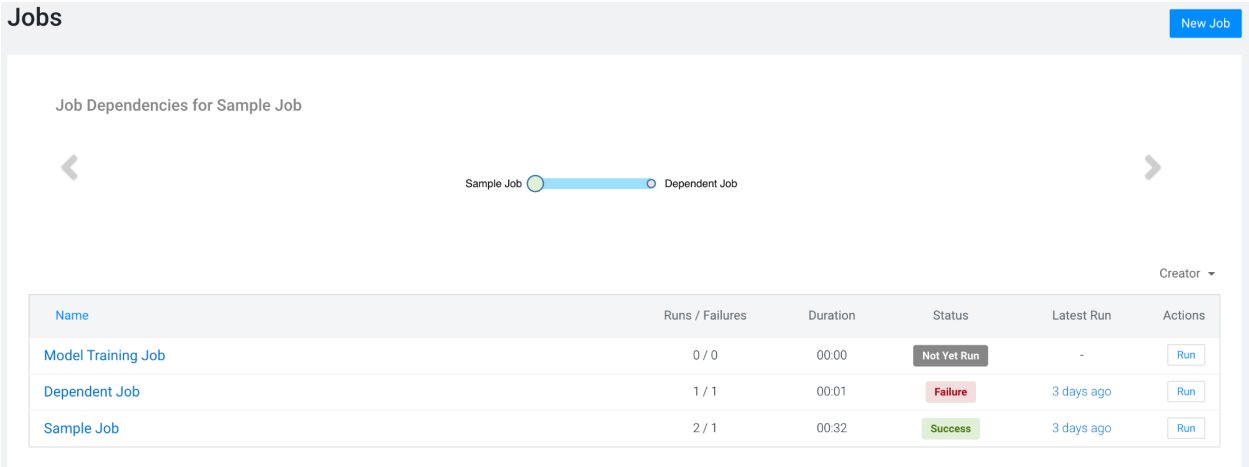


Familiarize yourself with the form. Notice that you can set alerts and attachments in relation to each execution. Pick the following options and then create the job.

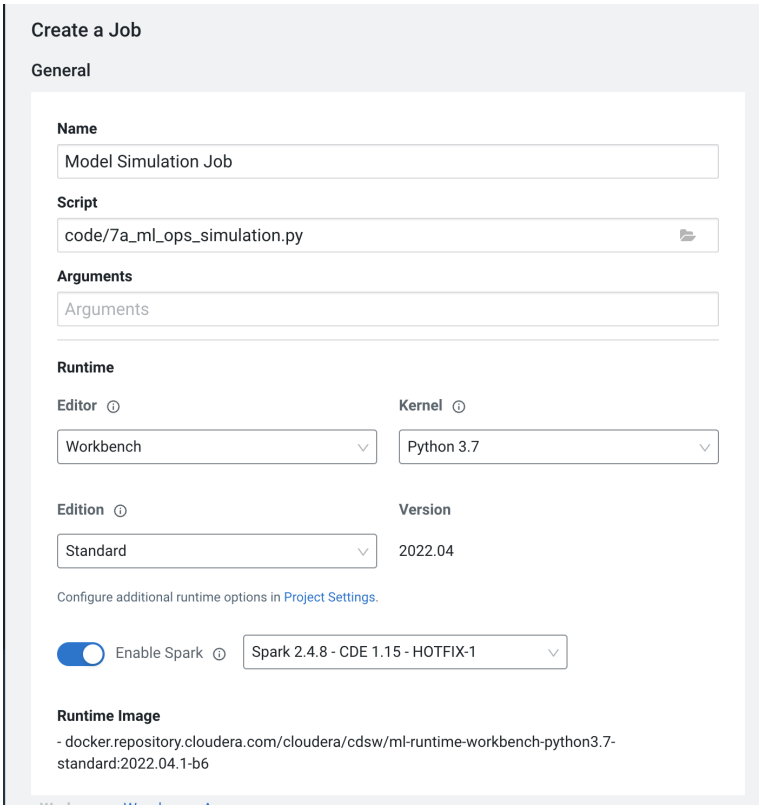


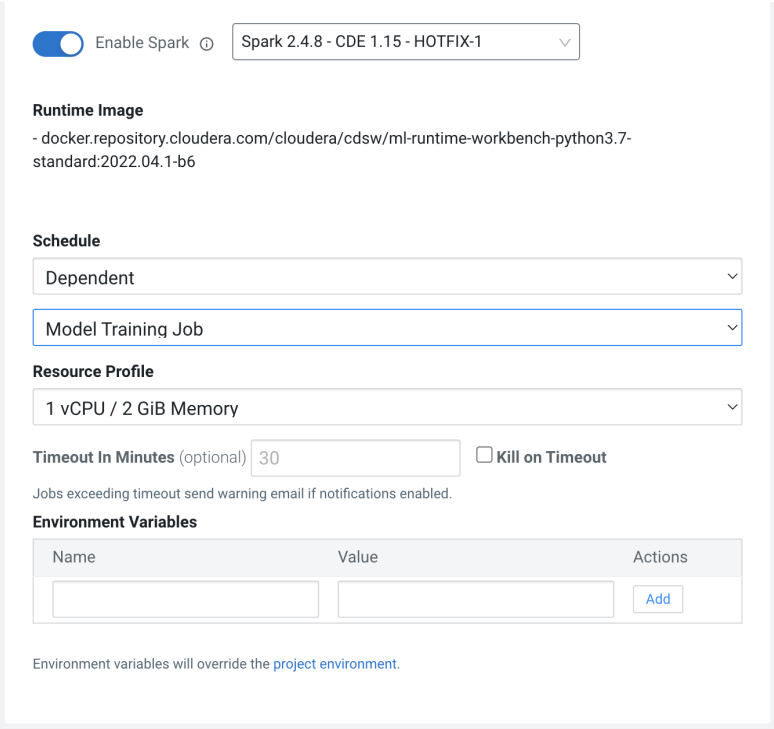
Job Name: train_model_job
Script: code/4_train_models.py
Editor: Workbench

Kernel: Python 3.7
Resource Profile: 1vCPU/2 GiB Memory
Runtime Edition: Standard
Enable Spark Add On: any Spark version
Schedule: Manual



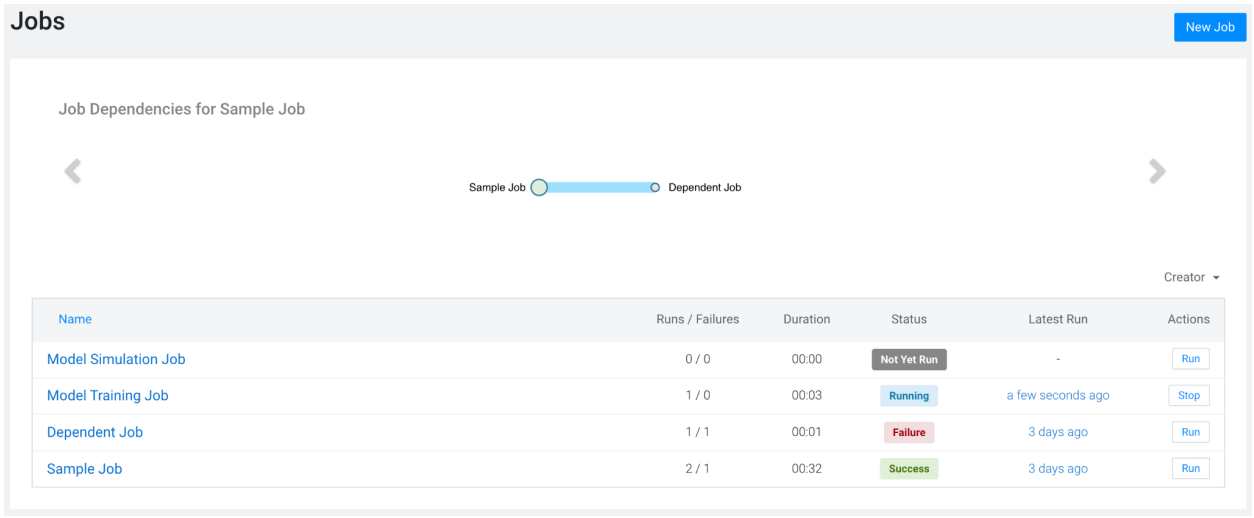
Notice the Jobs landing page has been populated with a new entry corresponding to your new job. Its status is “Not Yet Run”. Before executing the job, create another one with the following configurations. This time you will schedule it as “Dependent”.

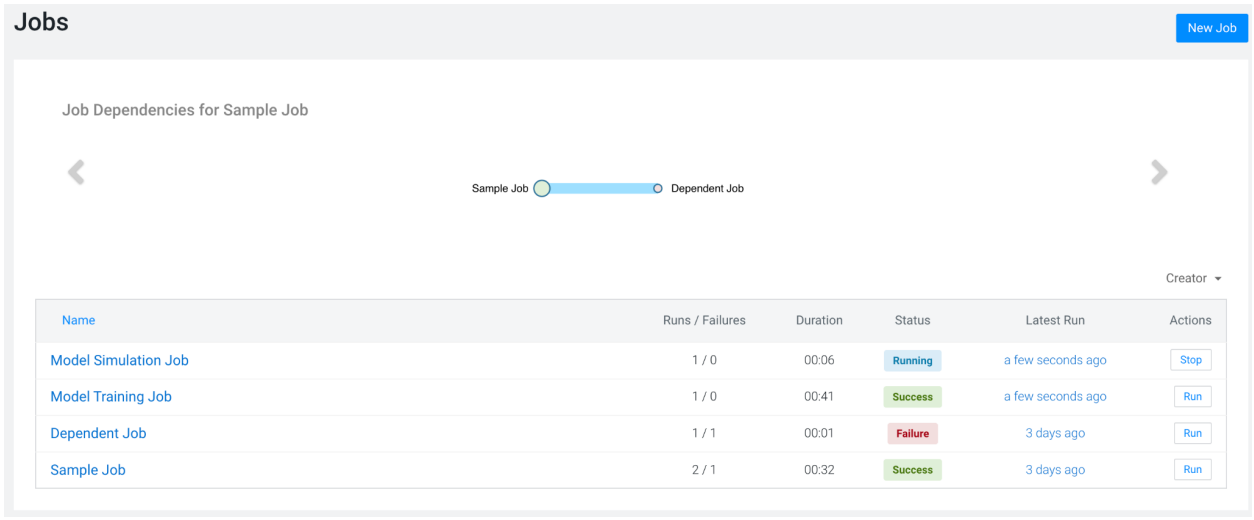


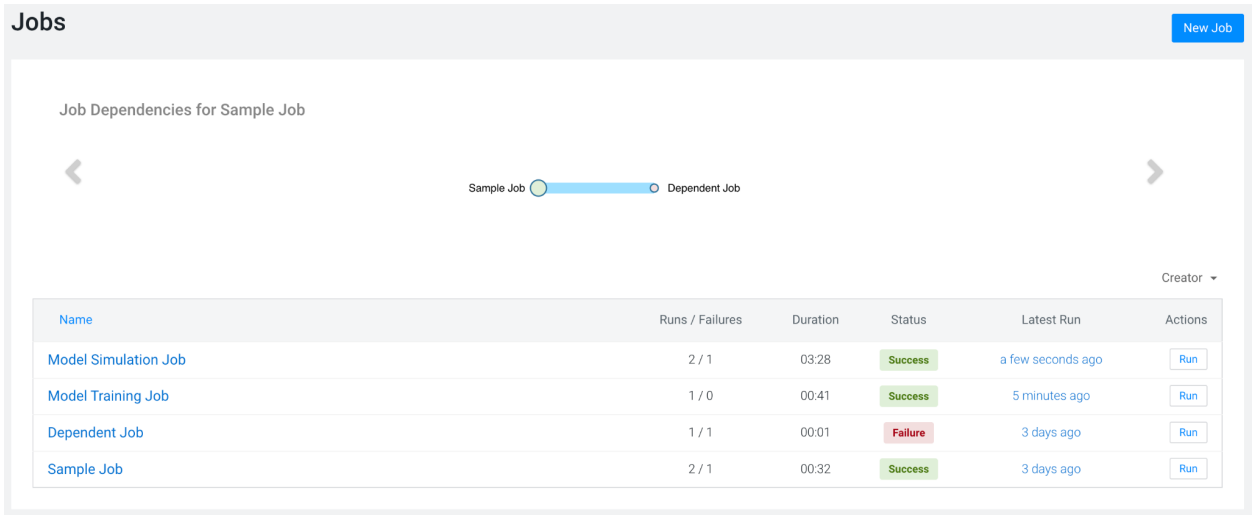


Job Name: model_simulation_job
Script: code/7a_ml_ops_simulation.py
Editor: Workbench
Kernel: Python 3.7
Resource Profile: 1vCPU/2 GiB Memory
Runtime Edition: Standard
Enable Spark Add On: any Spark version
Schedule: Dependent -> Module Training Job

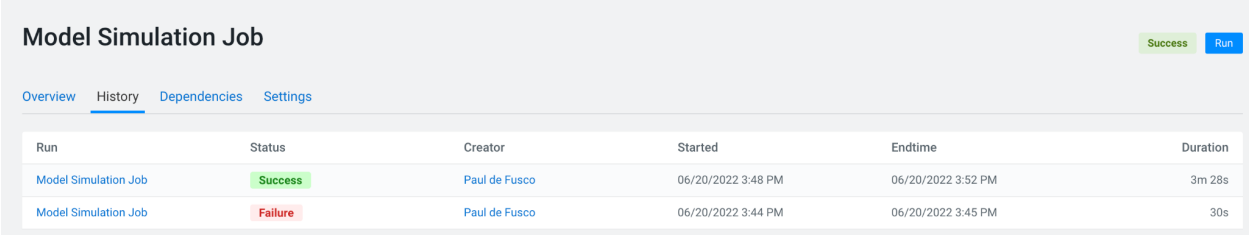
Navigate back to the Jobs landing page and manually trigger the “Module Training Job”. Observe it run for a minute or two. Notice that once it completes, the “Module Simulation Job” will launch automatically.


Jobs
New Job
Job Dependencies for Sample Job
Sample Job
Dependent Job
Creator
Name
Runs / Failures
Duration
Status
Latest Run
Actions
Model Simulation Job
0 / 0
00:00
Not Yet Run
-
Run
Model Training Job
1 / 0
00:03
Running
a few seconds ago
Stop
Dependent Job
1 / 1
00:01
Failure
3 days ago
Run
Sample Job
2 / 1
00:32
Success
3 days ago
Run


Jobs
New Job
Job Dependencies for Sample Job
Sample Job
Dependent Job
Creator
Name
Runs / Failures
Duration
Status
Latest Run
Actions
Model Simulation Job
1 / 0
00:06
Running
a few seconds ago
Stop
Model Training Job
1 / 0
00:41
Success
a few seconds ago
Run
Dependent Job
1 / 1
00:01
Failure
3 days ago
Run
Sample Job
2 / 1
00:32
Success
3 days ago
Run

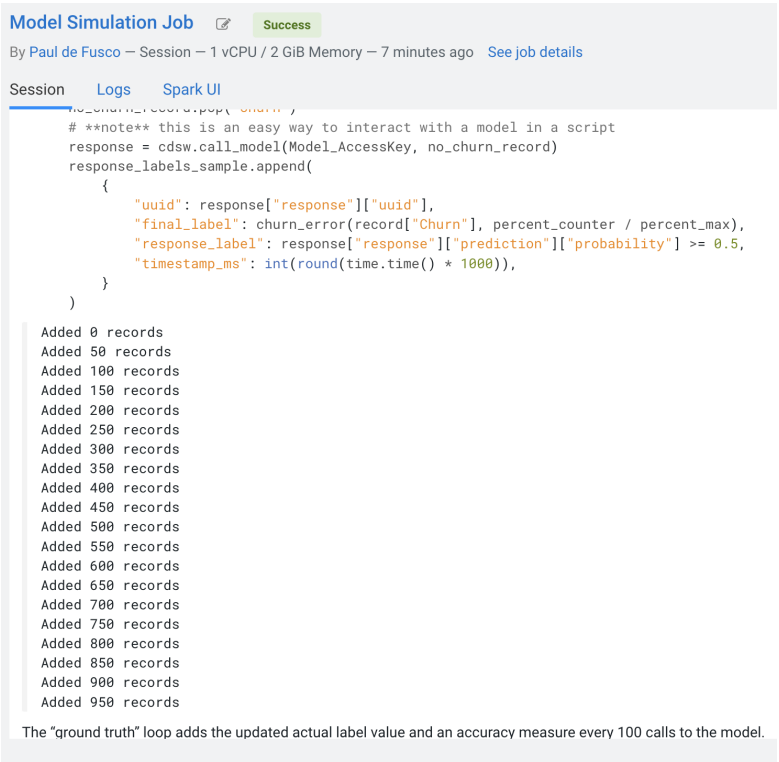

Jobs
New Job
Job Dependencies for Sample Job
Sample Job
Dependent Job
Creator
Name
Runs / Failures
Duration
Status
Latest Run
Actions
Model Simulation Job
2 / 1
03:28
Success
a few seconds ago
Run
Model Training Job
1 / 0
00:41
Success
5 minutes ago
Run
Dependent Job
1 / 1
00:01
Failure
3 days ago
Run
Sample Job
2 / 1
00:32
Success
3 days ago
Run

Optional: familiarize yourself with the code that just executed. Open the “Model Simulation Job”, then select the “History” tab and open the execution that succeeded.



You can use the contents in the “Session” tab to display exactly the code that was executed. This is particularly useful if you have a failure.

Scroll down and validate that the script sampled the data in our table and then “Added records” to a model. That model is actually the same classifier we just trained in the first job. It is hosted by an endpoint to serve requests over http. We will see more in the next section.



In summary, this section outlined the following concepts:

- With Experiments you can streamline your development process to more quickly iterate over model development trials. A typical use case is Hypterparameter tuning. The Experiments feature in CML allows you to track and visualize outcomes methodically.

- With Jobs you can schedule and orchestrate your batch scripts. Jobs allow you to build complex pipelines and are an essential part of any CI/CD or ML Ops pipeline. Typical use cases span from Spark ETL, Model Batch Scoring, A/B Testing and other model management related automations.
- To train a model is to run a batch script that uses a ML framework of choice to parameterize a function based on input data. Once the training routine is complete, the instance of the model is stored as a file. The model file is different from the model script. The model file can be reloaded from other scripts or notebooks for reuse.

Part 5: CML Models and Applications

Once a model is trained its predictions and insights must be put to use so they can add value to the organization. Generally this means using the model on new, unseen data in a production environment that offers key ML Ops capabilities.

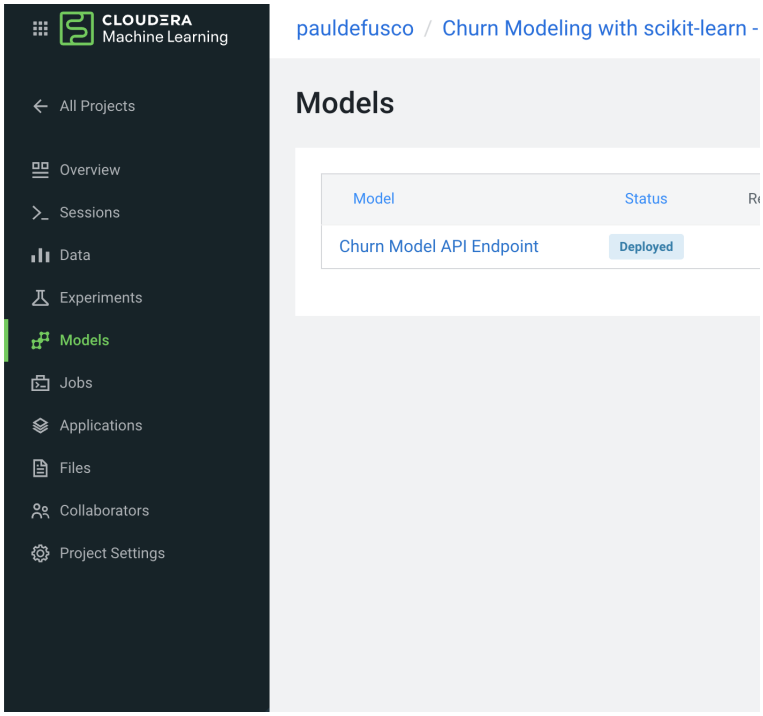
One such example is Batch Scoring via CML Jobs. The model is loaded in a script and the predict function provided by the ML framework is applied to data in batch. The script is scheduled and orchestrated to perform the scoring on a regular basis. In case of failures, the script or data are manually updated so the scoring can resume.

This pattern is simple and reliable but has one pitfall. It requires the user or system waiting for the scoring job to run at its scheduled time. What if predictions are required on a short notice? Perhaps when a prospect navigates on an online shopping website or a potential anomaly is flagged by a third party business system?

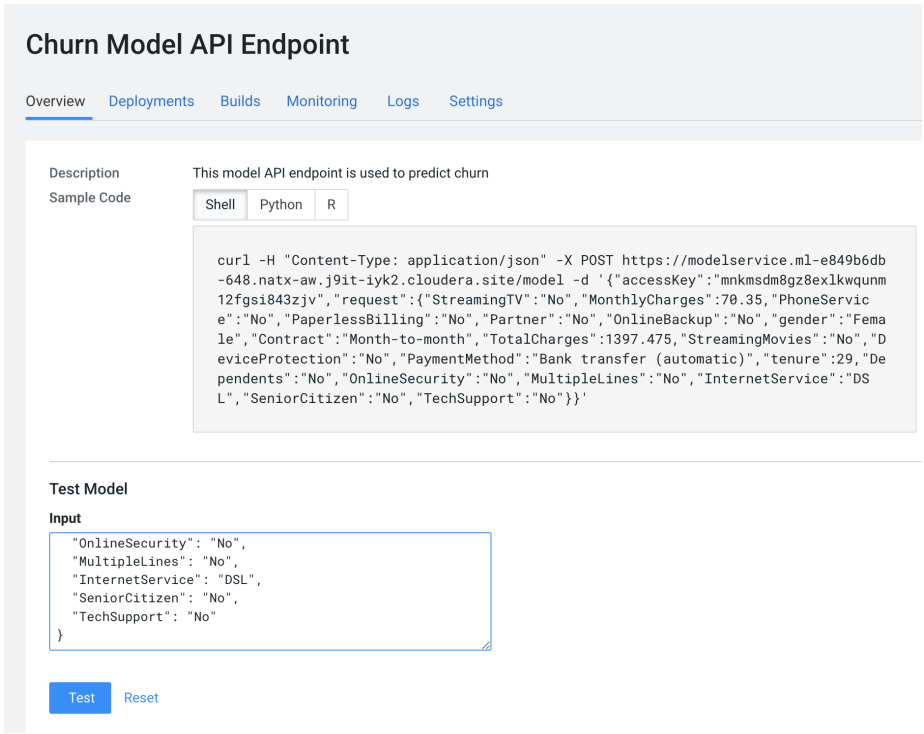
- CML Models allow you to deploy the same model script and model file in a REST Endpoint so the model can now serve responses in real time. The endpoint is hosted by a container.
- CML Models provides tracking, metadata and versioning features that allow you to manage models in production.
- Similarly, CML Applications allows you to deploy visual tools in an endpoint container. This is typically used to host apps with open source libraries such as Flask, Shiny, Streamlit and more.
- Once a model is deployed to a CML Models container, a CML Application can forward requests to the Model endpoint to provide visual insights powered by ML models.

Inspecting a Model Endpoint

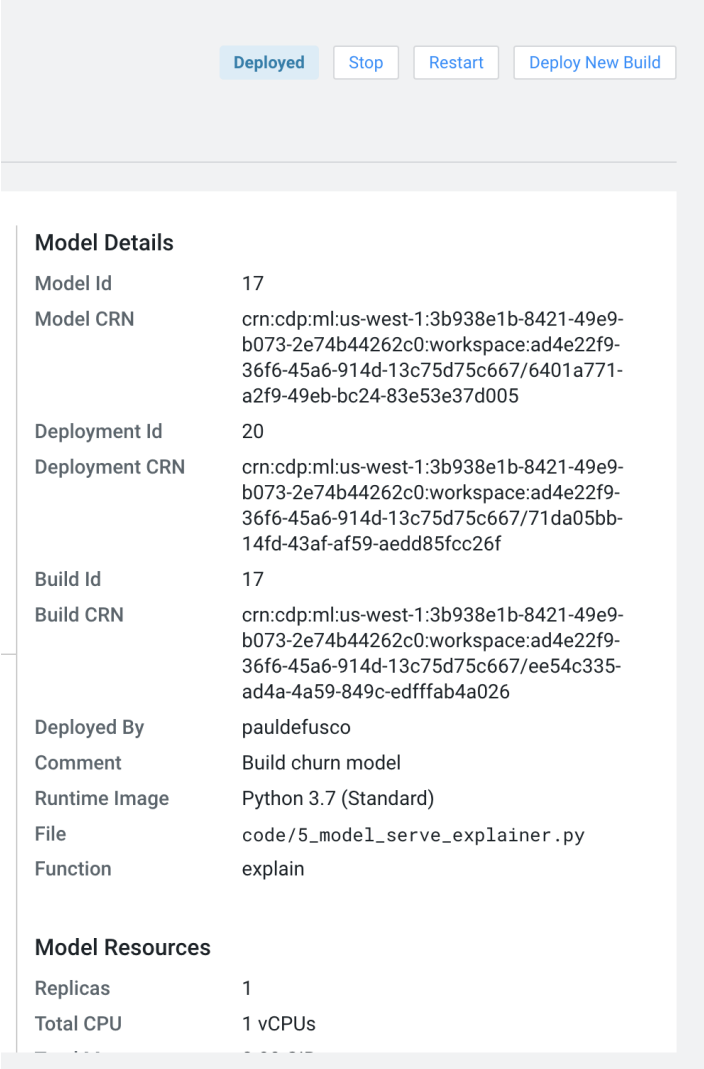
Navigate to the CML Models landing page. Notice that a model has already been created for you by the AMP setup automation.



Click on the Model name and explore the UI. The code for a sample request is provided on the left side.



On the right side observe the model's metadata. Each model is assigned a number of attributes including Model Name, Deployment, Build and creation timestamp.



Scroll back to the left side and click on the "Test" icon. The window is pre-populated for you with a sample json dictionary. The dictionary is a set of key value pairs representing a customer's attributes. For example, a customer who is currently on a DSL Internet Service plan.

The test simulates a request submission to the Model endpoint. The model processes the input and returns the output along with metadata and a prediction for the customer. In addition, the request is assigned a unique identifier. We will use this metadata for ML Ops later in part 6.

Test Model

Input


```
  "OnlineSecurity": "No",
  "MultipleLines": "No",
  "InternetService": "DSL",
  "SeniorCitizen": "No",
  "TechSupport": "No"
}
```


Test Reset

Result

Status success


```
{
    "model_deployment_crn": "crn:cdp:ml:us-west-1:3b938e1b-84
    "prediction": {
        "data": {
            "Contract": "Month-to-month",
            "Dependents": "No",
            "DeviceProtection": "No",
            "InternetService": "DSL",
            "MonthlyCharges": 70.35,
            "MultipleLines": "No",
            "OnlineBackup": "No",
            "OnlineSecurity": "No",
            "PaperlessBilling": "No",
            "Partner": "No",
            "PaymentMethod": "Bank transfer (automatic)",
            "PhoneService": "No",
```

Response


```
{
    "model_deployment_crn": "crn:cdp:ml:us-west-1:3b938e1b-8421-49e9-b073
    "prediction": {
        "data": {
            "Contract": "Month-to-month",
            "Dependents": "No",
            "DeviceProtection": "No",
            "InternetService": "DSL",
            "MonthlyCharges": 70.35,
            "MultipleLines": "No",
            "OnlineBackup": "No",
            "OnlineSecurity": "No",
            "PaperlessBilling": "No",
            "Partner": "No",
            "PaymentMethod": "Bank transfer (automatic)",
            "PhoneService": "No",
            "SeniorCitizen": "No",
            "StreamingMovies": "No",
            "StreamingTV": "No",
            "TechSupport": "No",
            "TotalCharges": 1397.475,
            "gender": "Female",
            "tenure": 29
        },
        "explanation": {
            "Contract": 0.11753693425620225,
            "InternetService": -0.15772997525799545,
            "MonthlyCharges": 0.0520604589033855,
            "MultipleLines": -0.02942972434722526,
            "OnlineSecurity": 0.05260308649410543,
            "PhoneService": -0.032876986650731695,
            "StreamingMovies": -0.03410779919944891,
            "StreamingTV": -0.045846594458528366,
            "TotalCharges": -0.07970724880651439,
            "tenure": 0.10511782642448833
        },
        "probability": 0.03353621326546117
    },
    "uuid": "fd4943c0-dc3a-41c8-97cd-d32e8e0550f9"
}
```


Script 5: Inspecting a Model Script

Navigate back to the Project Overview page and open the “5_model_serve_explainer.py” script. Scroll down and familiarize yourself with the code.

- Notice the method “explain” method. This is the Python function whose purpose is to receive the Json input as a request and return a Json output as a response.
- Within the method, the classifier object is used to apply the model object’s predict method.
- In addition, notice that a decorator named “@cdsw.model_metrics” is applied to the “explain” method. Thanks to the decorator you can use the “cdsw.track_metric” methods inside the “explain” method to register each scalar value associated with each request.
- The values are saved in the Model Metrics Store, a built in database used for tracking model requests.

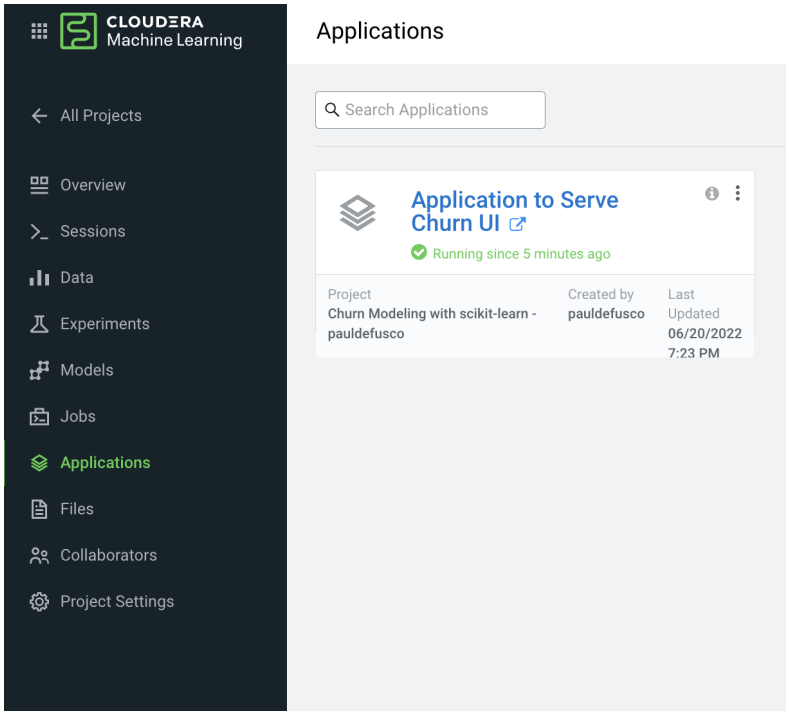
Navigate back to the Project Overview page. Open the “models/telco_linear” subfolder and notice the presence of the “telco_linear.pkl” file. This is the physical model file loaded by the .py script you just inspected above.

Files / code / 5_model_serve_explainer.py

```
from collections import ChainMap
import cdsw, numpy
from churnexplainer import ExplainedModel

# Load the model saved earlier.
em = ExplainedModel.load(model_name="telco_linear")

# *Note:* If you want to test this in a session, comment out the line
# `@cdsw.model_metrics` below. Don't forget to uncomment when you
# deploy, or it won't write the metrics to the database


@cdsw.model_metrics
# This is the main function used for serving the model. It will take in the JSON formatted arguments , calculate the probablity of
# churn and create a LIME explainer explained instance and return that as JSON.
def explain(args):
    data = dict(ChainMap(args, em.default_data))
    data = em.cast_dct(data)
    probability, explanation = em.explain_dct(data)

    # Track inputs
    cdsw.track_metric("input_data", data)

    # Track our prediction
    cdsw.track_metric("probability", probability)

    # Track explanation
    cdsw.track_metric("explanation", explanation)

    return {"data": dict(data), "probability": probability, "explanation": explanation}
```

Interacting with the Visual Application

Navigate to the CML Applications landing page. Notice that an application has already been created for you by the AMP setup automation.



Click on the application in order to open it. This will automatically redirect you to the Visual Application landing page where the same data you worked with earlier is presented in an interactive table.

On the left side notice the probability column. This is the target variable predicted by the Machine Learning Model. It reflects the probability of each customer churning. The value is between 0 and 1. A value of 0.49 represents a 49% probability of the customer churning. By default, if the probability is higher than 50% the classifier will label the customer as “will churn” and otherwise as “will not churn”.

The 50% threshold can be increased or decreased implying customers previously assigned a “will churn” label may flip to “will not churn” and vice versa. This has important implications as it provides an avenue for tuning the level selectivity based on business considerations but a detailed explanation is beyond the scope of this content.

Next, click on the customer at the top of the table to investigate further.

Refractor

id	Probability	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	D
2672	0.584	Femal	No	No	No	4	Yes	No	Fiber	No	No	N
1540	0.123	Male	No	No	No	63	Yes	Yes	Fiber	No	Yes	Y
1724	0.061	Male	No	Yes	No	64	Yes	Yes	Fiber	Yes	Yes	Y
2085	0.060	Femal	No	Yes	Yes	48	No	No ph	DSL	No	No	N
2434	0.045	Male	No	Yes	Yes	17	Yes	No	No	No in	No in	N
5073	0.009	Male	No	Yes	Yes	53	Yes	No	DSL	Yes	Yes	N
1150	0.009	Femal	No	Yes	Yes	45	Yes	Yes	No	No in	No in	N
4151	0.008	Male	No	Yes	Yes	39	Yes	Yes	No	No in	No in	N
4249	0.003	Femal	No	Yes	No	70	Yes	Yes	No	No in	No in	N
1712	0.001	Male	No	Yes	No	72	Yes	No	No	No in	No in	N

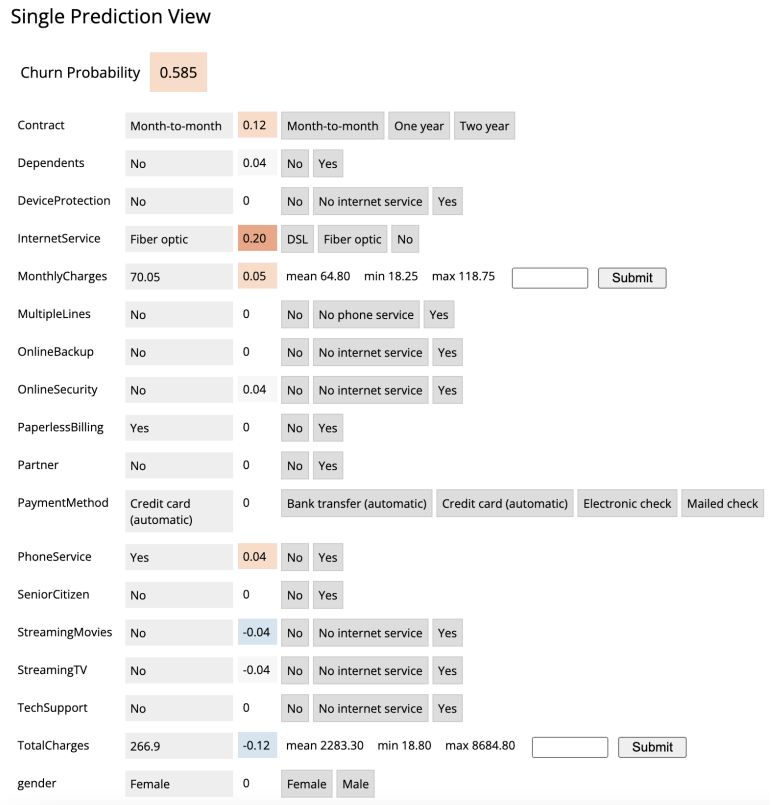
A more detailed view of the customer is automatically loaded. The customer has a 58% chance of churning.

The Lime model applied to the classifier provides a color coding scheme highlighting the most impactful features in the prediction label being applied to this specific customer.

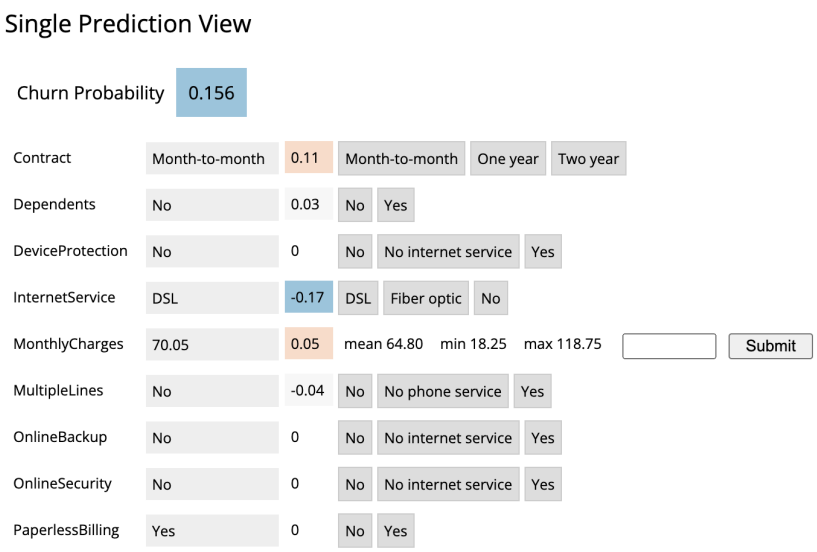
For example, this customer’s prediction of “will churn” is more significantly influenced by the “Internet Service” feature.

- The dark red color coding signals that the customer is negatively impacted by the current value for the feature.

- The current values of Monthly Charges and Phone Service also increase the likelihood of churn while the values of the Streaming Movies and Total Charges features decrease the likelihood of churn.



Let’s see what happens if we change the value for the most impactful feature in this given scenario i.e. “Internet Service”. Currently the value is set to “Fiber Optic”. Hover over the entry in the table and select “DSL”.



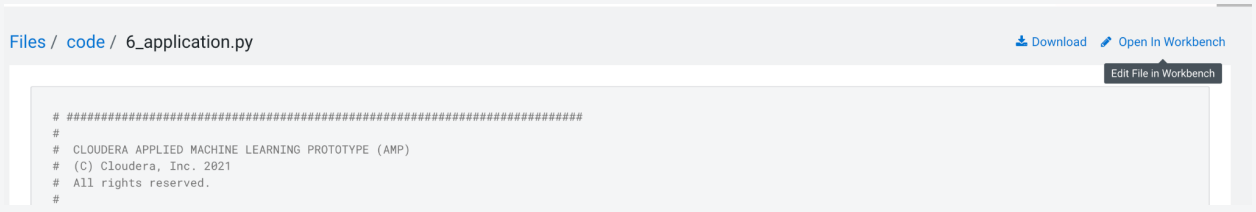
The table has now reloaded and the churn probability for this customer has dramatically decreased to roughly 15%.

This simple analysis can help the marketer optimize strategy in accordance to different business objectives. For example, the company could now tailor a proactive marketing offer based on this precious information. In addition, a more thorough financial analysis could be tied to the above simulation perhaps after adjusting the 50% threshold to increase or decrease selectivity based on business constraints or customer lifetime value assigned to each customer.

Script 6: Exploring the Application Script

Navigate back to the CML Project Home folder. Open the “Code” folder and then script “6_application.py”. This is a basic Flask application that serves the HTML and some specific data used for.

Click on “Open in Workbench” to visualize the code in a more reader friendly-mode.



Now you will be able to explore the code with the Workbench Editor. The “Launch Session” form will automatically load on the right side of your screen. There is no need to launch a session so you can just minimize it.

As always no code changes are required. Here are some key highlights::

- At lines 177 - 191 we load the model and use the “Explain” method to load a small dataset in the file. This is similar to what you did in script 5. If you want to display more data or fast changing data there are other ways to do this, for example with Cloudera SQL Stream Builder.
- At line 248 we run the app on the "CDSW_APP_PORT". This value is already preset for you as this is a default environment variable. You can reuse this port for other applications.

Part 6: CML Models Operations

The following steps assume you have executed the model_simulation_job CML Job as shown in Part 4. If you haven’t done it please go back and make sure to run the model simulation script.

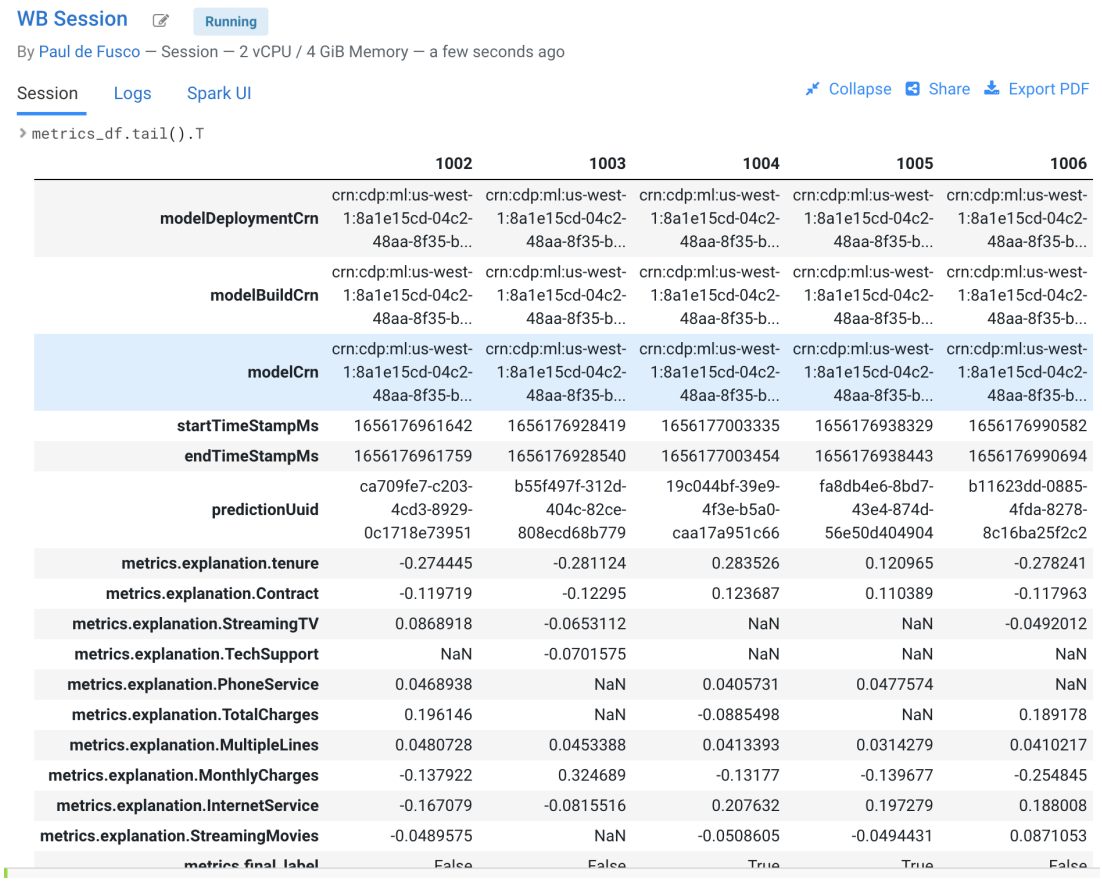
Navigate back to the project overview and launch a new session with the following configurations.

Session Name: telco_churn_ops_session
Editor: Workbench
Kernel: Python 3.7
Resource Profile: 1vCPU/2 GiB Memory
Runtime Edition: Standard
Runtime Version: Any available version
Enable Spark Add On: any Spark version

Once the session is running, open script “7b_ml_ops_visual.py” and explore the code in the editor. You can execute the whole script end to end without modifications.

Observe the code outputs on the right side. Here are the key highlights:

- Model predictions are tracked in the CML Models Metrics Store. This is enabled by the use of the Python decorator and the use of “cdsw.track_metrics” methods in script 5. What is being tracked is completely up to the script developer.
- You can then extract the predictions and related metadata and put the information in a Pandas dataframe. Again, the Python library you use does not matter and is entirely up to the developer.
- This is exactly what the first diagram on the right side of your screen shows. Each column represents a prediction request reaching your CML Model endpoint. Each row represents a metric you are tracking in the CML Models Metrics Store.



	1002	1003	1004	1005	1006
modelDeploymentCrn	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...
modelBuildCrn	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...
modelCrn	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...
startTimeStampMs	1656176961642	1656176928419	1656177003335	1656176938329	1656176990582
endTimeStampMs	1656176961759	1656176928540	1656177003454	1656176938443	1656176990694
predictionUuid	ca709fe7-c203-4cd3-8929-0c1718e73951	b55f497f-312d-404c-82ce-808ecd68b779	19c044bf-39e9-4f3e-b5a0-caa17a951c66	fa8db4e6-8bd7-43e4-874d-56e50d404904	b11623dd-0885-4fda-8278-8c16ba25f2c2
metrics.explanation.tenure	-0.274445	-0.281124	0.283526	0.120965	-0.278241
metrics.explanation.Contract	-0.119719	-0.12295	0.123687	0.110389	-0.117963
metrics.explanation.StreamingTV	0.0868918	-0.0653112	NaN	NaN	-0.0492012
metrics.explanation.TechSupport	NaN	-0.0701575	NaN	NaN	NaN
metrics.explanation.PhoneService	0.0468938	NaN	0.0405731	0.0477574	NaN
metrics.explanation.TotalCharges	0.196146	NaN	-0.0885498	NaN	0.189178
metrics.explanation.MultipleLines	0.0480728	0.0453388	0.0413393	0.0314279	0.0410217
metrics.explanation.MonthlyCharges	-0.137922	0.324689	-0.13177	-0.139677	-0.254845
metrics.explanation.InternetService	-0.167079	-0.0815516	0.207632	0.197279	0.188008
metrics.explanation.StreamingMovies	-0.0489575	NaN	-0.0508605	-0.0494431	0.0871053
metrics.final_label	False	False	True	True	False

- Once the tracked metrics have been saved to a Python data structure they can be used for all sorts of purposes.
- For example, the second diagram shows a basic line plot in Seaborn where the models' output probabilities are plotted as a function of time. On the X axis you can see the timestamp associated with each request. On the Y axis you can find the associated output probability.

WB Session Running

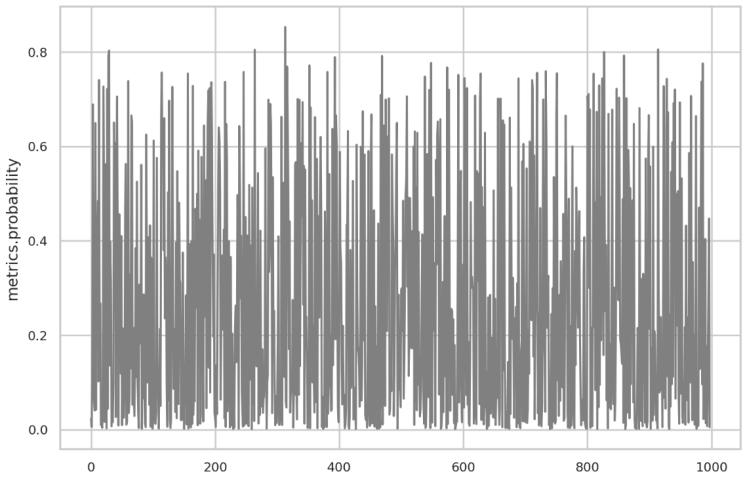
By Paul de Fusco — Session — 2 vCPU / 4 GiB Memory — a few seconds ago

Session Logs Spark UI Collapse Share Export PDF

Plot metrics.probability

```
prob_metrics = metrics_df.dropna(subset=["metrics.probability"]).sort_values(
    "startTimeStampMs"
)
sns.lineplot(
    x=range(len(prob_metrics)), y="metrics.probability", data=prob_metrics, color="grey"
)
```

```
<AxesSubplot:ylabel='metrics.probability'>
```



- Similarly, you can plot processing time as shown in the third diagram. This represents the time duration required to process a particular request.
- As an example, this information could be used to trigger the deployment of more resources to support this model endpoint when a particular threshold is passed. You can deploy more resources manually via the UI, or programmatically and in an automated CI/CD pipeline with CML APIv2 and CML Jobs.

WB Session Running

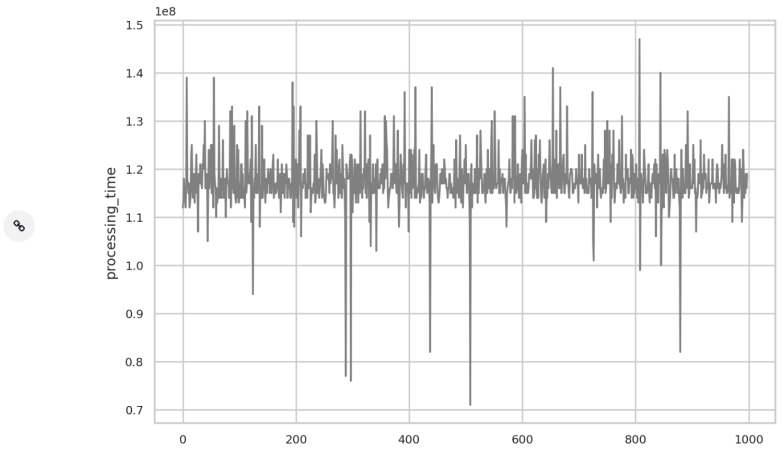
By Paul de Fusco — Session — 2 vCPU / 4 GiB Memory — a few seconds ago

Session Logs Spark UI Collapse Share

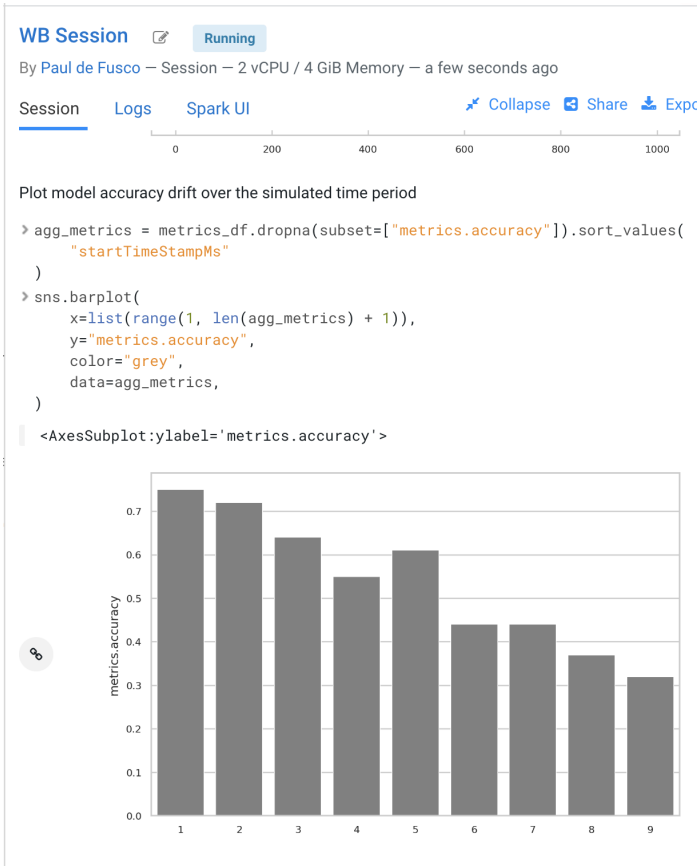
Plot processing time

```
time_metrics = metrics_df.dropna(subset=["processing_time"]).sort_values(
    "startTimeStampMs"
)
sns.lineplot(
    x=range(len(prob_metrics)), y="processing_time", data=prob_metrics, color="grey"
)
```

```
<AxesSubplot:ylabel='processing_time'>
```



- You can also monitor the model's accuracy over time. For example, the below diagram shows a line plot of prediction accuracy sorted over time. As you can see, the trend is negative and the model is making increasingly less accurate predictions.
- Just like with processing time and other metrics, CML allows you to implement ML Ops pipelines that automate actions related to model management. For example, you could use a combination of CML Jobs and CML APIv2 to trigger the retraining and redeployment of a model when its accuracy reaches a particular threshold over a particular time period.
- As always this is a relatively basic example. CML is an open platform for hands-on developers which gives users the freedom to implement more complex ML Ops pipelines.



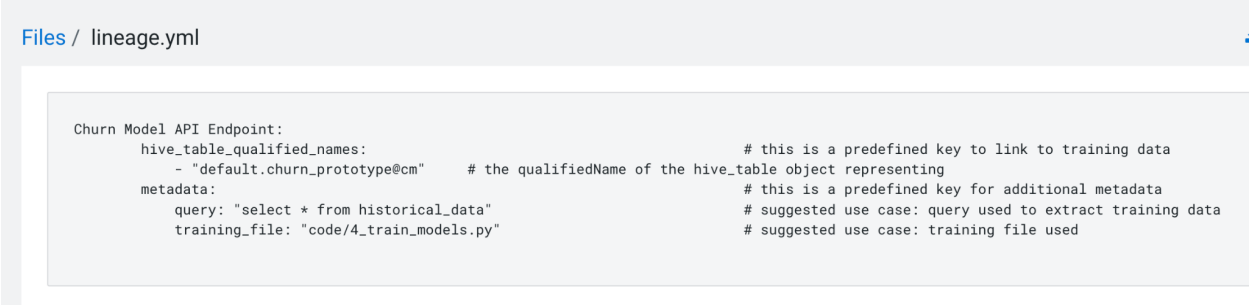
- Ground truth metrics can be collected with the cdsw.track_delayed_metrics method. This allows you to compare your predictions with the actual event after the prediction was output. In turn, this allows you to calculate the model accuracy and create visualizations such as the one above.
- For an example of the cdsw.track_delayed_metrics method open the “7a_ml_ops_simulation.py” script and review lines 249 - 269. Keep in mind that this is just a simulation.
- In a real world scenario the requests would be coming from an external system or be logged in a SQL or NoSQL database. In turn, the script above would be used to set ground truth values in batch via a CML Job or in real time with a CML Model endpoint.

Part 7: Model Lineage Tracking

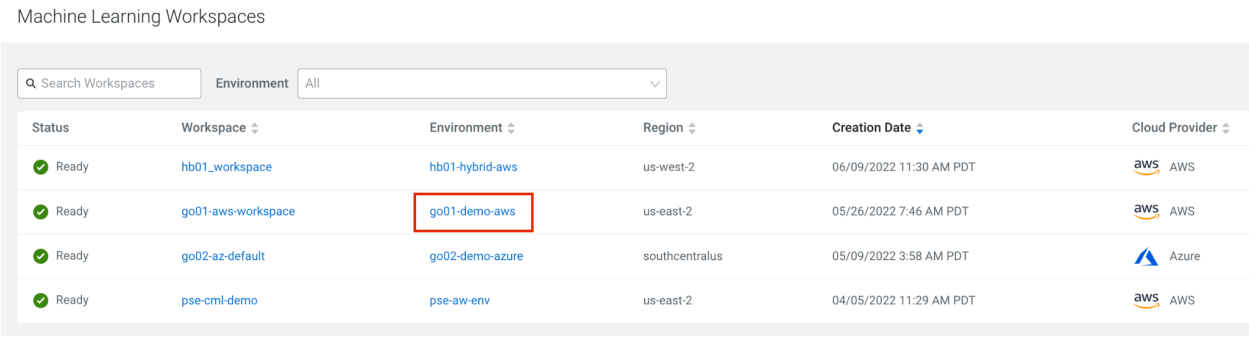
CDP is an end-to-end hybrid enterprise data platform. Every user, workload, and dataset and machine learning model can be governed from a central location via SDX, the Shared Data Experience.

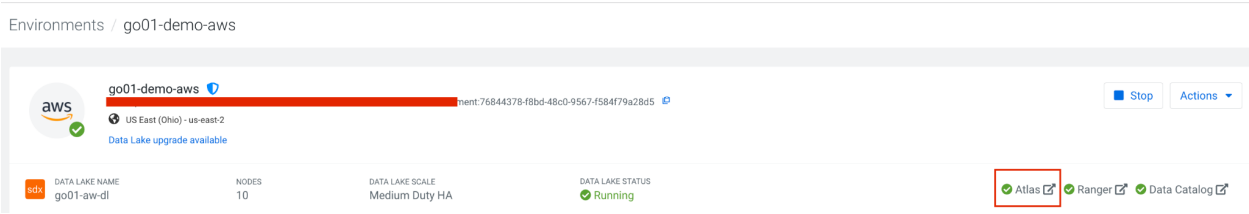
Under the hood, SDX tracks and secures activity related to each CDP Data Service via “Hooks” and “Plugins”, including CML. If you want your models to be logged in SDX you have to add them to the lineage.yml file located in your project home folder.

Navigate back to the Project Home folder and open the lineage.yml file located in the files section.

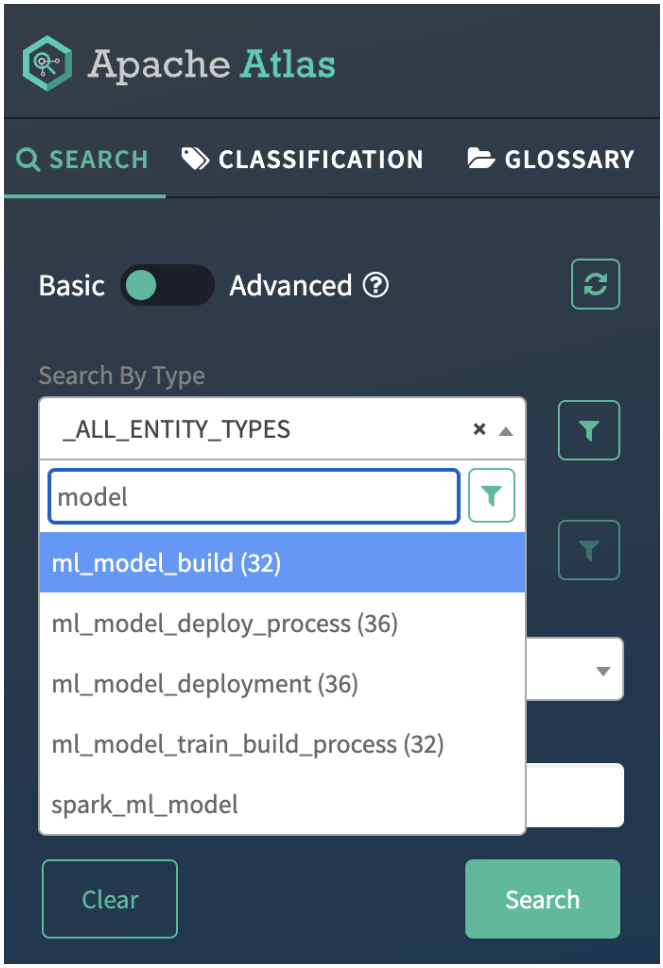


Navigate out of your CML project and go back to the Workspaces landing page. Click on the Data Lake associated with your CML Workspace and then open the Apache Atlas Service.

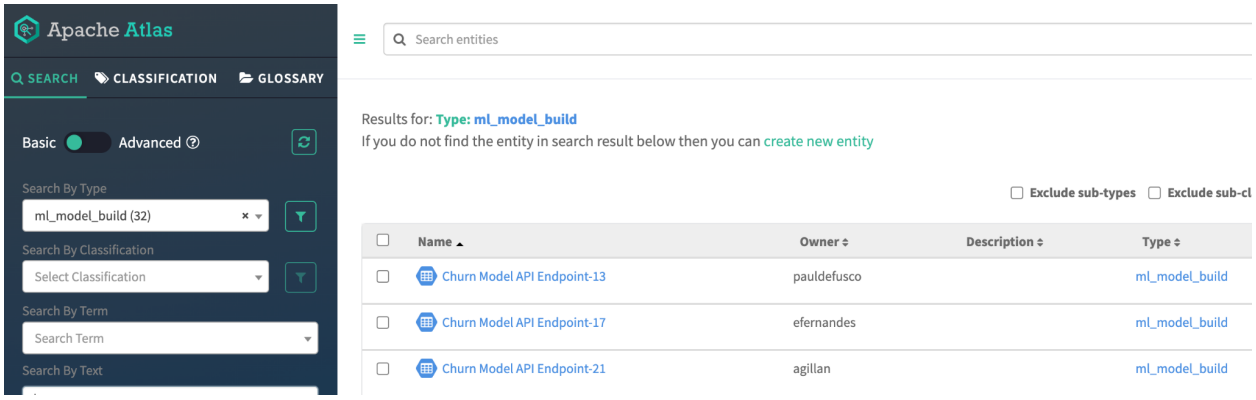




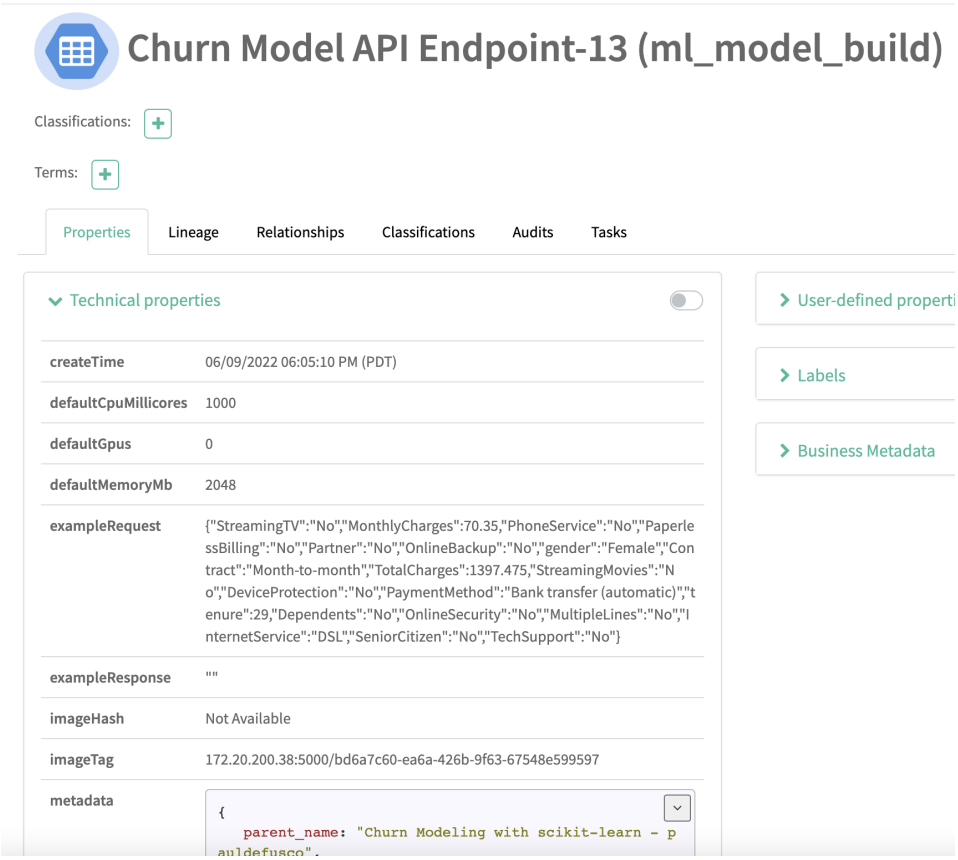
From the Atlas UI, search for ML models by entering the “ml_model_build” type. Notice that there are various Atlas entities to browse for models.

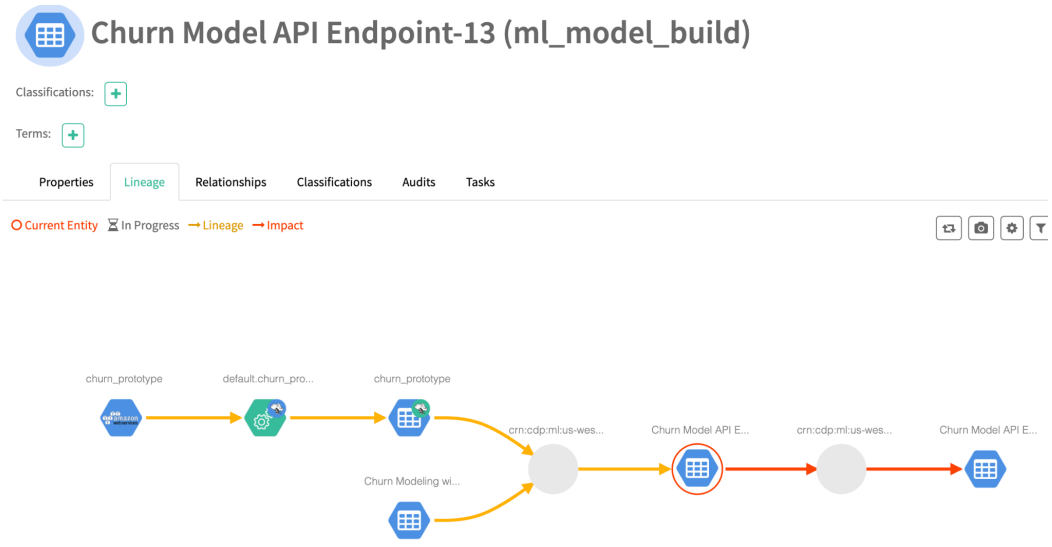


In the output, select the model you created. Notice that each model is assigned a unique ID at the end. That ID corresponds to the Model Build from CML. Open your model by selecting the Model Name.



Familiarize yourself with the Model properties tab. Notice that each model logged is associated with rich metadata. You can customize Atlas Model metadata by editing the lineage.yml file in the CML Project Home folder





Atlas and Ranger provides a rich set of Governance and Security capabilities. For example, you can apply Atlas tags to your entities across Data Services and then propagate Ranger policies to automatically secure applications across complex pipelines.

A detailed exploration of SDX in the context of CML is not in scope for this workshop but please visit the "Next Steps" section to find out more on this and other topics.

Conclusions

In this workshop you created an end to end project to support a Machine Learning model in Production.

- You easily created a Spark Session and explored a large dataset with the PySpark library. Thanks to CML Runtimes and Sessions you were able to switch between editors, resources, and optionally Python and Spark versions at the click of a button.
- You created a Model REST Endpoint to serve predictions to internal or external business applications. Then, you built an interactive dashboard to make the "black box model" interpretable for your business stakeholders.
- You explored the foundations of a basic ML Ops pipeline to easily retrain, monitor, and reproduce your model in production. With the CML Models interface you unit tested and increased model observability. Then, you monitored its performance with CML APIv2.
- Finally, you used CDP SDX to log and visualize Model Metadata and Lineage.

Next Steps

If you want to learn more about CML and CDP we invite you to visit the following assets and tutorials or ask your Cloudera Workshop Lead for a follow up.

- Learn how to use Cloudera Applied ML Prototypes to discover more CML Projects using MLFlow, Streamlit, Tensorflow, PyTorch and other popular libraries. The AMP Catalog is maintained by the Cloudera Fast Forward Labs team and allows you to automatically deploy complex use cases within minutes.
- CML HowTo: A series of tips and tricks for the CML beginner
- Sentiment Analysis in R: and end to end ML project with SparklyR and GPU training
- CSA2CML: Build a real time anomaly detection dashboard with Flink, CML, and Streamlit
- SDX2CML: Explore ML Governance and Security features in more detail to increase legal compliance and enhance ML Ops best practices.
- CML2CDE: Create CI/CD Pipelines for Spark ETL with CML Notebooks and CDE Virtual Cluster
- API v2: Familiarize yourself with API v2, CML's goto Python Library for ML Ops and DevOps
- Distributed PyTorch with Horovod: A quickstart for distributing Horovod with the CML Workers API