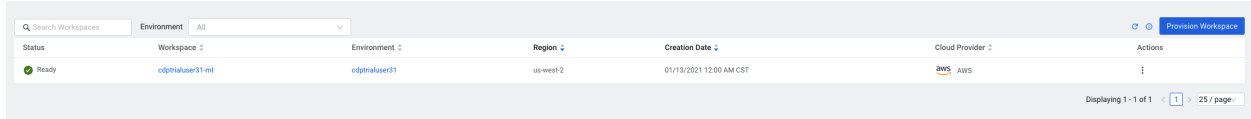


Cloudera Machine Learning Workshop

Part 1 - **REQUIRED IN ORDER FOR LAB TO WORK PROPERLY:** Create a CML Project

Workspaces are the heart of the Cloudera Machine Learning (CML)

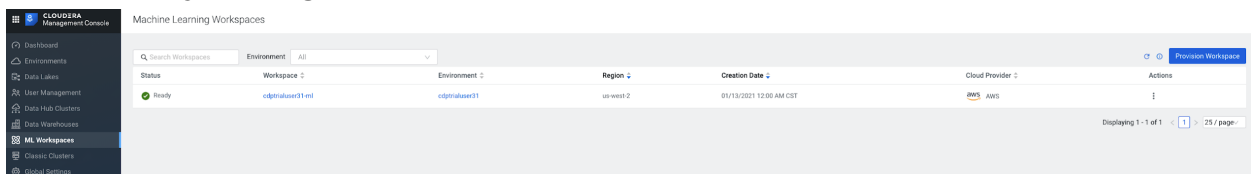
Machine Learning Workspaces



Status	Workspace	Environment	Region	Creation Date	Cloud Provider	Actions
Ready	cdptraluser01-ml	cdptraluser01	us-west-2	01/13/2021 12:00 AM CST	AWS	

Displaying 1 - 1 of 1 < 1 > 25 / page

A Workspace is a small cluster that runs on a kubernetes service to provide teams of data scientists to develop, test, train, and ultimately deploy machine learning models. **Click into the Workspace by clicking the Workspace name.**

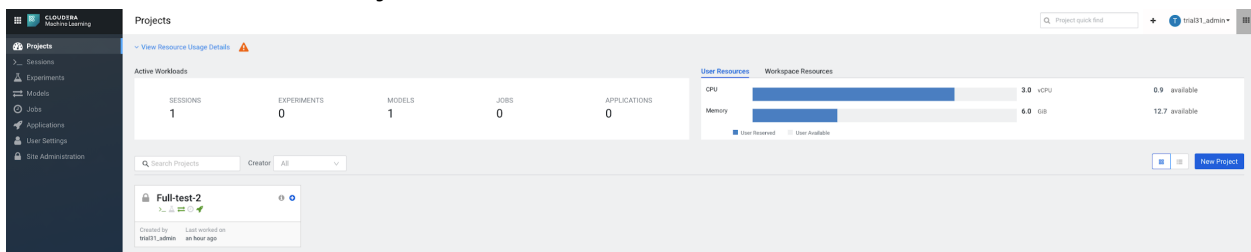


Machine Learning Workspaces

Status	Workspace	Environment	Region	Creation Date	Cloud Provider	Actions
Ready	cdptraluser01-ml	cdptraluser01	us-west-2	01/13/2021 12:00 AM CST	AWS	

Displaying 1 - 1 of 1 < 1 > 25 / page

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 develop and deploy models along with other CML features. **Click on “New Project”**



Projects

Active Workloads

SESSIONS	EXPERIMENTS	MODELS	JOB	APPLICATIONS
1	0	1	0	0

Workspace Resources

Resource	Used	Available
CPU	3.0	6.9
Memory	6.0 GB	12.7 GB

Full-test-2

Created by: User worked on 10/13/2021 at 10:00 AM

When creating a new project give a Name, Visibility, and initial configuration.

Project Name: Telco_churn

Visibility: Private

Initial Setup: Git -> <https://github.com/andy-hansen/cml.git>

Create a New Project

Project Name

Project Visibility

- ☒ **Private** - Only added collaborators can view the project.
- ☐ **Public** - All authenticated users can view this project.

Initial Setup

Blank

Template

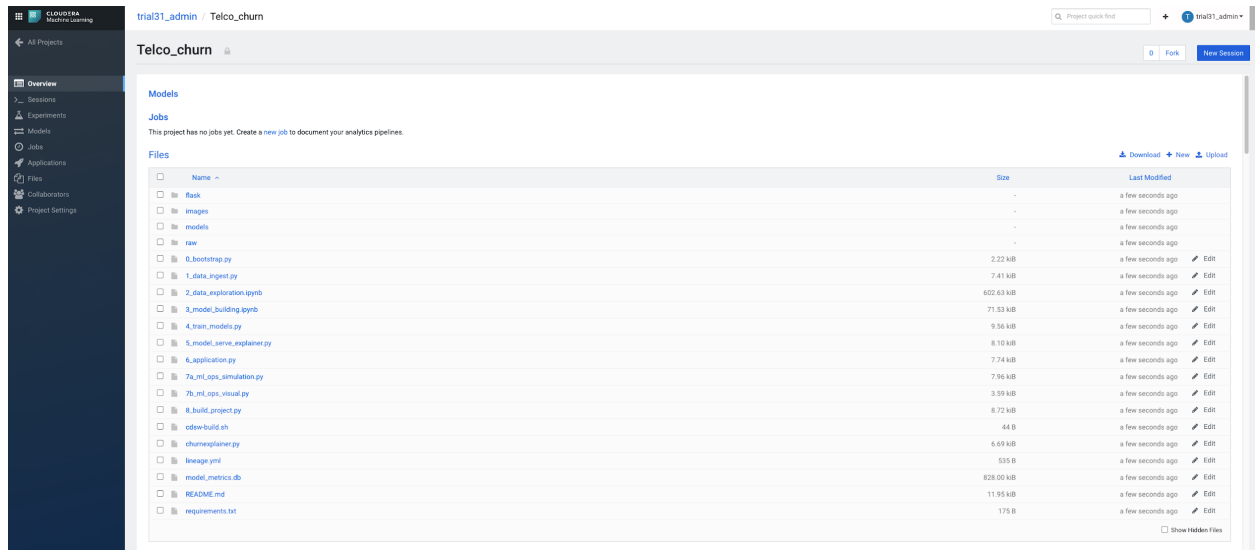
Local

Git

Create Project

Part 2: CML Project Overview

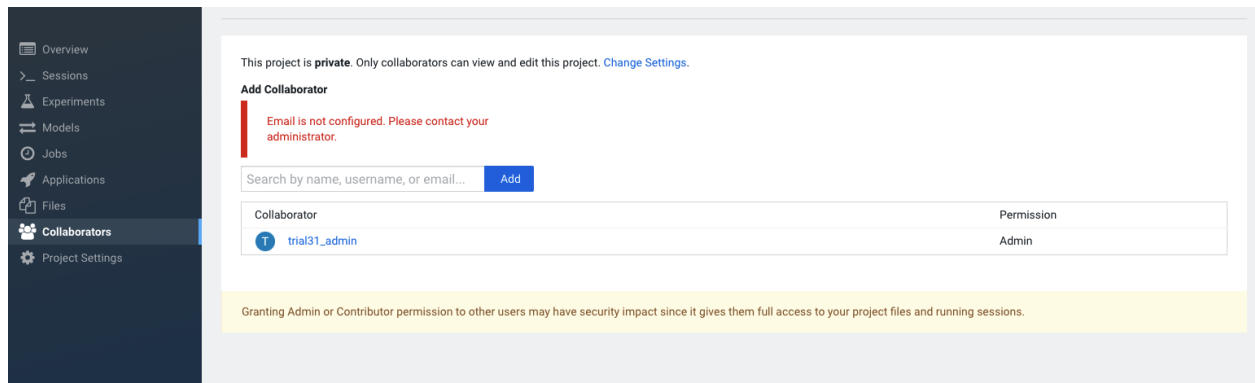
Overview gives you access to all the features of a CML project. We will only have files copied from the Github repo currently. Initially it is good to start on the management components of a project.



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](#).

-> Change the Runtime/Engine to Legacy Engine

-> Click Save Engine

CLUSTERA
Machine Learning

← All Projects

Overview

> Sessions

Experiments

Models

Jobs

Applications

Files

Collaborators

Project Settings

trial31_admin / Telco_churn / Settings

Q Proj

Project Settings

Options Runtime/Engine Advanced Tunnels Delete Project

Project Name

Telco_churn

Description

Description

Visibility

- ☒ **Private** - Only collaborators can view or edit the project.
- ☐ **Public** - All authenticated users can view this project. Collaborators can also edit the project.

Update Project

CLUSTERA
Machine Learning

← All Projects

Overview

> Sessions

Experiments

Models

Jobs

Applications

Files

Collaborators

Project Settings

trial31_admin / Telco_churn / Settings / Runtime/Engine

Q

Project Settings

Options Runtime/Engine Advanced Tunnels Delete Project

Default Engine: ☐ ML Runtime  ☒ Legacy Engine 

Engine Image

Select the Docker image that Cloudera Machine Learning should use to run sessions and jobs in this project. If you'd like to use a different image, contact your site administrator.

Default engine image, docker.repository.cloudera.com/cloudera/cdsw/engine:13-cml-2020.10-2

Save Engine

Third-party editors

Cloudera Machine Learning allows you to launch sessions with third-party, web-based editors. To add an editor to the Start New Session menu, first launch a session with the built-in Workbench editor and install the third-party editor of your choice. Then, come back to this page and provide a name for the editor and the command to start the editor server. Ensure that you start the server on the port specified by the `CDSW_APP_PORT` environment variable.

+ New Editor

CLOUDERA
Machine Learning

← All Projects

Overview

Sessions

Experiments

Models

Jobs

Applications

Files

Collaborators

Project Settings

trial31_admin / Telco_churn / Settings / Engine

Q Proj

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
<input type="text"/>	<input type="text"/>	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.

[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

Q

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 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.

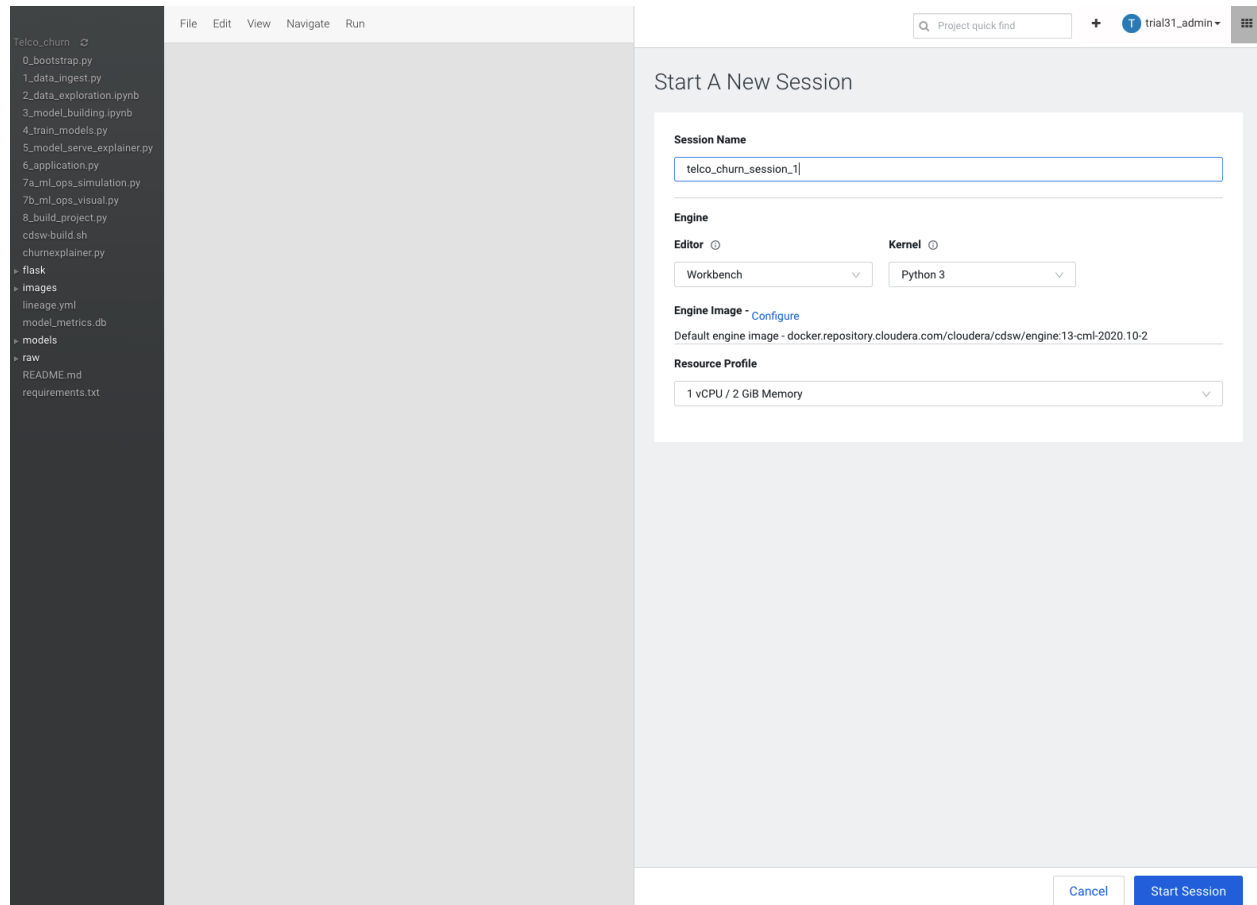
From the Overview page click on New Session

The screenshot displays the Databricks workspace interface for a project named 'Telco_churn'. The left sidebar shows the navigation menu with 'Overview' selected. The main content area is divided into sections: 'Models' (indicating no jobs yet), 'Jobs' (with a message to create a new job), and 'Files'. The 'Files' section contains a table listing project files and folders.

Name	Size	Last Modified
flask	-	a few seconds ago
images	-	a few seconds ago
models	-	a few seconds ago
raw	-	a few seconds ago
0_bootstrap.py	2.22 kB	a few seconds ago
1_data_ingest.py	7.41 kB	a few seconds ago
2_data_exploration.ipynb	602.63 kB	a few seconds ago
3_model_building.ipynb	71.53 kB	a few seconds ago
4_train_models.py	9.56 kB	a few seconds ago
5_model_serve_explainer.py	8.10 kB	a few seconds ago
6_application.py	7.74 kB	a few seconds ago
7a_ml_ops_simulation.py	7.96 kB	a few seconds ago
7b_ml_ops_visual.py	3.59 kB	a few seconds ago
8_build_project.py	8.72 kB	a few seconds ago
cdaw-build.sh	44 B	a few seconds ago
churnexplainer.py	6.69 kB	a few seconds ago
lineage.yml	535 B	a few seconds ago
model_metrics.db	828.00 kB	a few seconds ago
README.md	11.95 kB	a few seconds ago
requirements.txt	175 B	a few seconds ago

Session Name: telco_churn_session_1
Editor: Workbench
Kernel: Python 3
Engine Image: Default
Resource Profile: 1vCPU/2 GiB Memory

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.**

The screenshot displays a JupyterLab environment. On the left, a file explorer shows a project structure with files like `0_bootstrap.py`, `1_data_ingest.py`, `2_data_exploration.ipynb`, `3_model_building.ipynb`, `4_train_models.py`, `5_model_serve_explainer.py`, `6_application.py`, `7a_ml_ops_simulation.py`, `7b_ml_ops_visual.py`, `8_build_project.py`, `cdsw-build.sh`, `churnexplainer.py`, `flask`, `images`, `lineage.yml`, `model_metrics.db`, `models`, `raw`, `README.md`, and `requirements.txt`. The `8_build_project.py` file is selected.

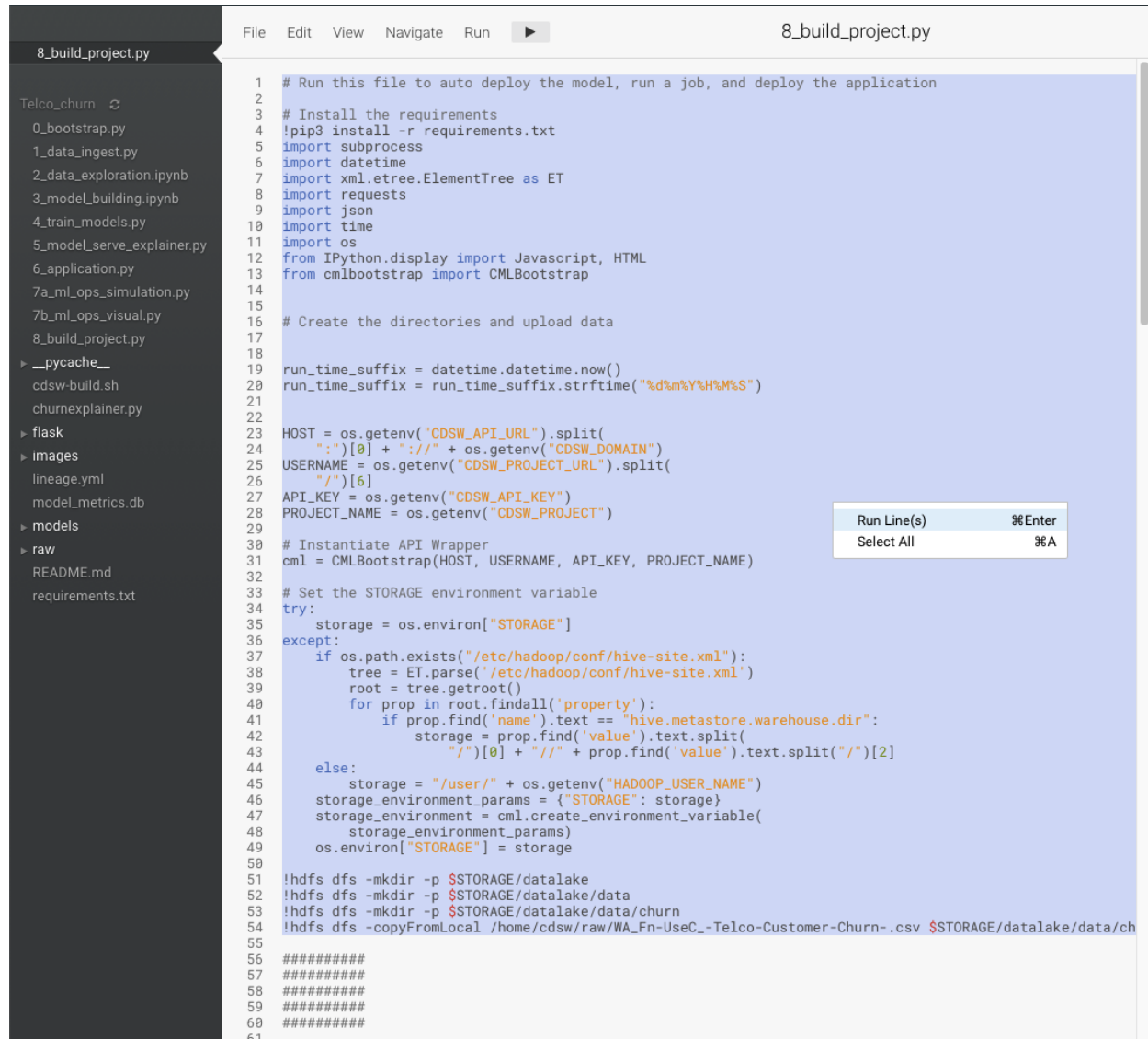
The central code editor shows the content of `8_build_project.py`. The script starts with a comment: `# Run this file to auto deploy the model, run a job, and deploy the`. It then proceeds to install requirements, import necessary modules, and set up environment variables. The script uses `os.getenv` to retrieve values for `HOST`, `USERNAME`, `API_KEY`, and `PROJECT_NAME`. It then creates a `CMLBootstrap` object and sets the `STORAGE` environment variable. The script also includes commands to create directories and copy files from a local path to a remote path.

The right pane shows a terminal window titled `telco_churn_session_1`. It displays the output of the script, including the installation of requirements and the execution of the `run` method. The terminal output shows the script's progress and the final status of the deployment.

Open up the script `8_build_project.py` from the left pane. In the editor pane we are going to select and run the script in several parts. Throughout the script you will see breaks in the code defined by Part 1, Part 2, Part 3, etc.

Part 1: Will install any required packages to be used. As an example, flask is installed as part of the project. A variable is also set as part of the project. Last but not least we are loading a file from the project as a test dataset and moving that to a s3 location.

Select and highlight the contents of part 1, then Right click -> Run Line(s)



```
1 # Run this file to auto deploy the model, run a job, and deploy the application
2
3 # Install the requirements
4 !pip3 install -r requirements.txt
5 import subprocess
6 import datetime
7 import xml.etree.ElementTree as ET
8 import requests
9 import json
10 import time
11 import os
12 from IPython.display import Javascript, HTML
13 from cmlbootstrap import CMLBootstrap
14
15 # Create the directories and upload data
16
17
18 run_time_suffix = datetime.datetime.now()
19 run_time_suffix = run_time_suffix.strftime("%d%m%Y%H%M%S")
20
21
22
23 HOST = os.getenv("CDSW_API_URL").split(
24     "://")[0] + "://" + os.getenv("CDSW_DOMAIN")
25 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
26     "://")[6]
27 API_KEY = os.getenv("CDSW_API_KEY")
28 PROJECT_NAME = os.getenv("CDSW_PROJECT")
29
30 # Instantiate API Wrapper
31 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
32
33 # Set the STORAGE environment variable
34 try:
35     storage = os.environ["STORAGE"]
36 except:
37     if os.path.exists("/etc/hadoop/conf/hive-site.xml"):
38         tree = ET.parse('/etc/hadoop/conf/hive-site.xml')
39         root = tree.getroot()
40         for prop in root.findall('property'):
41             if prop.find('name').text == "hive.metastore.warehouse.dir":
42                 storage = prop.find('value').text.split(
43                     "/")[0] + "://" + prop.find('value').text.split("/")[2]
44     else:
45         storage = "/user/" + os.getenv("HADOOP_USER_NAME")
46     storage_environment_params = {"STORAGE": storage}
47     storage_environment = cml.create_environment_variable(
48         storage_environment_params)
49     os.environ["STORAGE"] = storage
50
51 !hdfs dfs -mkdir -p $STORAGE/datalake
52 !hdfs dfs -mkdir -p $STORAGE/datalake/data
53 !hdfs dfs -mkdir -p $STORAGE/datalake/data/churn
54 !hdfs dfs -copyFromLocal /home/cds/raw/WA_Fn-UseC-Telco-Customer-Churn-.csv $STORAGE/datalake/data/ch
55
56 #####
57 #####
58 #####
59 #####
60 #####
61
```

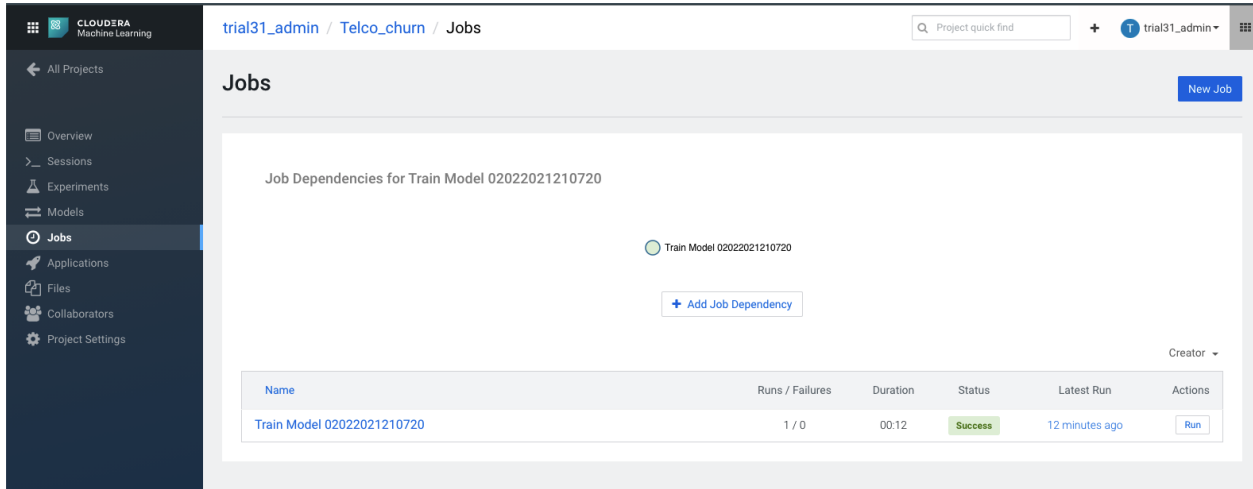
Run Line(s)	⌘Enter
Select All	⌘A

Part 2: The telco churn dataset is ingested from s3 and a hive table is created using Spark.
Select and highlight the contents of part 2, then Right click -> Run Line(s)

Part 3: Create a CML Job and start the job. A job automates the action of launching an engine, running a script, and tracking the results, all in one batch process. Jobs are created within the purview of a single project and can be configured to run on a recurring schedule. You can customize the engine environment for a job, set up email alerts for successful or failed job runs, and email the output of the job to yourself or a colleague.

Select and highlight the contents of part 3, then Right click -> Run Line(s)

Once the Job is started we will look at what was created. **Click the Project button** (top right corner of screen). **Click on Jobs** to explore the job that was created and details can be found by **Clicking on the Job Name**.



The screenshot shows the Cloudera Machine Learning interface. The left sidebar contains navigation links: All Projects, Overview, Sessions, Experiments, Models, Jobs (selected), Applications, Files, Collaborators, and Project Settings. The main content area is titled 'Jobs' and shows 'Job Dependencies for Train Model 02022021210720'. A dependency graph shows a single node 'Train Model 02022021210720' with a green status icon. Below the graph is a '+ Add Job Dependency' button. At the bottom, a table lists job runs.

Name	Runs / Failures	Duration	Status	Latest Run	Actions
Train Model 02022021210720	1 / 0	00:12	Success	12 minutes ago	Run

Part 4: Using CML, you can create any function within a script and deploy it to a REST API. In a machine learning project, this will typically be a predict function that will accept an input and return a prediction based on the model's parameters.

Select and highlight the contents of part 4, then Right click -> Run Line(s)

When getting to deploying the Model this can take a little time, and is a good spot to stretch the legs and refill your coffee.

```
179 with open('lineage.yml', 'w') as lineage:
180     lineage.write(yaml_text)
181
182 #####
183 #####
184 #####
185 #####
186 #####
187
188 # Create Model
189 example_model_input = {"StreamingTV": "No", "MonthlyCharges": 70.3,
190                        "StreamingMovies": "No", "DeviceProtection":
191                        "No"}
192
193 create_model_params = {
194     "projectId": project_id,
195     "name": "Model Explorer " + run_time_suffix,
196     "description": "Explain a given model prediction",
197     "visibility": "private",
198     "enableAuth": False,
199     "targetFilePath": "5_model_serve_explainer.py",
200     "targetFunctionName": "explain",
201     "engineImageId": default_engine_image_id,
202     "kernel": "python3",
203     "examples": [
204         {
205             "request": example_model_input,
206             "response": {}
207         }
208     ],
209     "cpuMillicores": 1000,
210     "memoryMb": 2048,
211     "nvidiaGPUs": 0,
212     "replicationPolicy": {"type": "fixed", "numReplicas": 1},
213     "environment": {}
214 }
215
216 new_model_details = cml.create_model(create_model_params)
217 access_key = new_model_details["accessKey"] # todo check for bad
218 model_id = new_model_details["id"]
219
220 print("New model created with access key", access_key)
221
222 # Disable model_authentication
223 cml.set_model_auth({"id": model_id, "enableAuth": False})
224
225 # Wait for the model to deploy.
226 is_deployed = False
227 while is_deployed == False:
228     model = cml.get_model({"id": str(
229         new_model_details["id"]), "latestModelDeployment": True, "
230         if model["latestModelDeployment"]["status"] == 'deployed':
231             print("Model is deployed")
232             break
233         else:
234             print("Deploying Model.....")
235             time.sleep(10)
236
237 #####
238 ##### Good time to take a break
239 #####
240
241 # Change the line in the flask/single_view.html file.
242 subprocess.call(["sed", "-i", "s/const(saccessKey.*/const accessK
243                 access_key + '";', "/home/cdsw/flask/single_view
244
245 # Change the model_id value in the 7a_model_operations.py and 7b.m
246 subprocess.call(["sed", "-i", "s/model_id=./model_id = '' +
247                 model_id + '"/, "/home/cdsw/7a_ml_ops_simulation
248                 subprocess.call(["sed", "-i", "s/model_id=./model_id = '' +
249
250
```

```
telco_churn_session_1
By trial31_admin - Python 3 Session - 1 vCPU / 2 GiB Memory - 28 minutes ago

Session Logs Spark UI
print(New model created with access key, access_key)

New model created with access key ml8mrkticueiyejlz1z12lfxmdyjkdw
Disable model_authentication

cml.set_model_auth({"id": model_id, "enableAuth": False})

{'accessKey': 'ml8mrkticueiyejlz1z12lfxmdyjkdw',
 'authEnabled': False,
 'createdAt': '2021-02-02T21:07:55.221Z',
 'creator': {'id': 3, 'type': 'user', 'username': 'trial31_admin'},
 'creatorId': 3,
 'crn': 'crn:cdp:ml:us-west-1:f209dbac-bd43-4198-b7d1-7ecf68c2a8f4:workspace:8a7b7049-802d-4806-9f1e-999f4f759df7/3156dd9a-c588-462c-ad6a-92a751a1da8d',
 'defaultReplicationPolicy': {'numReplicas': 1, 'type': 'fixed'},
 'defaultResources': {'cpuMillicores': 1000,
 'memoryMb': 2048,
 'nvidiaGPUs': 0},
 'description': 'Explain a given model prediction',
 'htmlUrl': 'https://ml-86b5138c-077.cdptrial.hs7l-0qe7.cloudera.site/trial31_admin/telco_c
hurn/models/4',
 'id': '4',
 'name': 'Model Explorer 02022021210720',
 'namespace': 'mlx-user-3',
 'project': {'crn': 'crn:cdp:ml:us-west-1:f209dbac-bd43-4198-b7d1-7ecf68c2a8f4:workspace:8a
7b7049-802d-4806-9f1e-999f4f759df7/8e442fec-d089-4ccb-a684-4074df4ba7f1',
 'id': 10,
 'name': 'Telco_churn',
 'slug': 'telco_churn',
 'projectId': 10,
 'projectOwner': {'id': 3, 'type': 'user', 'username': 'trial31_admin'},
 'updatedAt': '2021-02-02T21:07:57.479Z',
 'visibility': 'private'}

Wait for the model to deploy.

is_deployed = False
while is_deployed == False:
    model = cml.get_model({"id": str(
        new_model_details["id"]), "latestModelDeployment": True, "latestModelBuild": True})
    if model["latestModelDeployment"]["status"] == 'deployed':
        print("Model is deployed")
        break
    else:
        print("Deploying Model.....")
        time.sleep(10)

Deploying Model.....
Deploying Model.....
Deploying Model.....
Deploying Model.....
Model is deployed
```

Part 5: Applications give data scientists a way to create ML web applications/dashboards and easily share them with other business stakeholders. Applications can range from single visualizations embedded in reports, to rich dashboard solutions such as Tableau. They can be interactive or non-interactive.

Applications stand alongside other existing forms of workloads in CML (sessions, jobs, experiments, models). Like all other workloads, applications must be created within the scope of a project. Each application is launched within its own isolated engine. Additionally, like models, engines launched for applications do not time out automatically. They will run as long as the web application needs to be accessible by any users and must be stopped manually when needed.

Select and highlight the contents of part 5, then Right click -> Run Line(s)

When deploying the CML Flask Application you will be provided with a URL to follow at the end of your session pane output. **Click to Open Application UI**

```

1 # Change the line in the flask/single_view.html file.
2 subprocess.call(["sed", "-i", 's/const\saccessKey.*/const accessKey\saccess_key + "/', "/home/cdsw/flask/single_view.html"])
3
4 # Change the model_id value in the 7a_model_operations.py and 7b_model_operations.py
5 subprocess.call(["sed", "-i", 's/model_id = .*/model_id = "' + model_id + '"/', "/home/cdsw/7a_ml_ops_simulation.py"])
6 subprocess.call(["sed", "-i", 's/model_id = .*/model_id = "' + model_id + '"/', "/home/cdsw/7b_ml_ops_visual.py"])
7
8 # Create Application
9 create_application_params = {
10     "name": "Explainer App",
11     "subdomain": run_time_suffix[:],
12     "description": "Explainer web application",
13     "type": "manual",
14     "script": "6_application.py", "environment": {},
15     "kernel": "python3", "cpu": 1, "memory": 2,
16     "nvidia_gpu": 0
17 }
18
19 new_application_details = cml.create_application(create_application_params)
20 application_url = new_application_details["url"]
21 application_id = new_application_details["id"]
22
23 # print("Application may need a few minutes to finish deploying. Open link below in about a minute.")
24 print("Application created, deploying at ", application_url)
25
26 # Wait for the application to deploy.
27 is_deployed = False
28 while is_deployed == False:
29     # Wait for the application to deploy.
30     app = cml.get_application(str(application_id), {})
31     if app["status"] == 'running':
32         print("Application is deployed")
33         break
34     else:
35         print("Deploying Application.....")
36         time.sleep(10)
37
38 HTML("<a href='{0}'>Open Application UI</a>".format(application_url))
39
40 #####
41 #####
42 #####
43 #####
44
45 # This will run the model operations section that makes calls to the model
46 # metrics and track metric aggregations
47
48 exec(open("7a_ml_ops_simulation.py").read())

```

```

model_id + '"/', "/home/cdsw/7a_ml_ops_simulation.py"])
0
> subprocess.call(["sed", "-i", 's/model_id = .*/model_id = "' + model_id + '"/', "/home/cdsw/7b_ml_ops_visual.py"])
0
Create Application
> create_application_params = {
    "name": "Explainer App",
    "subdomain": run_time_suffix[:],
    "description": "Explainer web application",
    "type": "manual",
    "script": "6_application.py", "environment": {},
    "kernel": "python3", "cpu": 1, "memory": 2,
    "nvidia_gpu": 0
}
> new_application_details = cml.create_application(create_application_params)
> application_url = new_application_details["url"]
> application_id = new_application_details["id"]
print("Application may need a few minutes to finish deploying. Open link below in about a minute.")
> print("Application created, deploying at ", application_url)
Application created, deploying at https://02022021210720.ml-86b5138c-077.cdptrial.hs71-0qe7.cloudiera.site
Wait for the application to deploy.
> is_deployed = False
> while is_deployed == False:
    # Wait for the application to deploy.
    app = cml.get_application(str(application_id), {})
    if app["status"] == 'running':
        print("Application is deployed")
        break
    else:
        print("Deploying Application.....")
        time.sleep(10)
    Deploying Application.....
    Deploying Application.....
    Application is deployed
> HTML("<a href='{0}'>Open Application UI</a>".format(application_url))
Open Application UI

```

Within the Flask Application UI you can experiment with some of the parameters that will run against the model to predict how likely a customer is to churn:

Single Prediction View

Churn Probability **0.737**

Contract	Month-to-month	0.12	Month-to-month	One year	Two year	
Dependents	No	0	No	Yes		
DeviceProtection	Yes	0	No	No internet service	Yes	
InternetService	Fiber optic	0.19	DSL	Fiber optic	No	
MonthlyCharges	97.85	-0.24	mean 64.80	min 18.25	max 118.75	<input type="text"/> <input type="button" value="Submit"/>
MultipleLines	Yes	0.06	No	No phone service	Yes	
OnlineBackup	No	0	No	No internet service	Yes	
OnlineSecurity	No	0.05	No	No internet service	Yes	
PaperlessBilling	Yes	0	No	Yes		
Partner	No	0	No	Yes		
PaymentMethod	Bank transfer (automatic)	0	Bank transfer (automatic)	Credit card (automatic)	Electronic check	Mailed check
PhoneService	Yes	0.04	No	Yes		
SeniorCitizen	No	0	No	Yes		
StreamingMovies	Yes	0.09	No	No internet service	Yes	
StreamingTV	Yes	0.07	No	No internet service	Yes	
TechSupport	No	0	No	No internet service	Yes	
TotalCharges	1105.4	-0.08	mean 2283.30	min 18.80	max 8684.80	<input type="text"/> <input type="button" value="Submit"/>
gender	Female	0	Female	Male		
tenure	11	0.11	mean 32.42	min 1.00	max 72.00	<input type="text"/> <input type="button" value="Submit"/>

Part 6: Run the last code snippet in the workbench to complete the project build script (8_build_project.py). This goes through a process of simulating a model that drifts over 1000 calls to the model. The file contains comments with details of how this is done.

Select and highlight the contents of part 6, then Right click -> Run Line(s)

```
284
285 #####
286 #####
287 #####
288 #####
289 #####
290
291 # This will run the model operations section that makes calls to t
292 # mertics and track metric aggregations
293
294 exec(open("7a_ml_ops_simulation.py").read())
295
```

Update
Update
Adding
Update
Update
Adding
Update
Update

Line 291, Column 1 ★ 295 Lines Python Spaces 2

You can also share the workbench session with other users if they would like to view results of the code. A URL can be shared out if users outside of CML would like to view code/results of work.

Click on share at the top of the Session pane

← Project >_ Terminal Access ✍ Clear ⚡ Interrupt ■ Stop Sessions ▾ ⌵

telco_churn_session_1 **Running**

By **trial31_admin** — Python 3 Session — 1 vCPU / 2 GiB Memory — 33 minutes ago

Session Logs Spark UI Collapse Share Export PDF

`exec(open("7a_ml_ops_simulation.py").read())`

Added 0 records
Added 50 records
Added 100 records
Added 150 records
Added 200 records
Added 250 records
Added 300 records
Added 350 records
Added 400 records
Added 450 records
Added 500 records
Added 550 records

🔒 These results are being shared.

Stop Sharing

☐ Hide code and text

Who can view:

☐ Any logged in user with the link

☒ Specific users/teams with the link ([Change...](#))

Currently shared with no one.

A full CML project should now be running with a Job, Model, and Application deployed! Go back to the project and take a look at the Model and Applications page.

Going to the **Model** page will show you the deployed model. Clicking on the Model will give various tabs of monitoring and statistics in addition to previous deployments of the same model.

On the **Applications** page we can see our running flask app we looked at previously.

Models and Applications will continue to run even if the workbench session is stopped or timeout occurs. These will run on engines as part of the CML workspace.

CLOUDERA

Machine Learning

All Projects

Overview

Sessions

Experiments

Models

Jobs

Applications

Files

Collaborators

Project Settings

trial31_admin / Telco_churn

Project quick find

+ trial31_admin

Telco_churn

0 Fork

New Session

Models

Model	Status	Replicas	CPU	Memory	Last Deployed	Actions
Model Explainer 02022021210720	Deployed	1 / 1	1	2.00 GiB	Feb 2, 2021, 03:08 PM	Stop

Jobs

Name	Runs / Failures	Duration	Status	Latest Run	Actions
Train Model 02022021210720	1 / 0	00:12	Success	27 minutes ago	Run

Files

Download + New Upload

Name	Size	Last Modified
__pycache__	-	27 minutes ago
flask	-	26 minutes ago
images	-	38 minutes ago
models	-	38 minutes ago
raw	-	27 minutes ago
0_bootstrap.py	2.22 kiB	38 minutes ago
1_data_ingest.py	7.41 kiB	38 minutes ago
2_data_exploration.ipynb	602.63 kiB	38 minutes ago
3_model_building.ipynb	71.53 kiB	38 minutes ago
4_train_models.py	9.56 kiB	38 minutes ago
5_model_serve_explainer.py	8.10 kiB	38 minutes ago
6_application.py	7.74 kiB	38 minutes ago
7a_ml_ops_simulation.py	7.96 kiB	26 minutes ago