DSCI-272: Predicting with Cloudera Machine Learning

None

Cloudera Education

Table of contents

DSCI-272: Predicting with Cloudera Machine Learning 4

Exercises 4

Introduction to CML on CDP 5

Login to Cloudera Data Platform 5

Overview of CML Workspace 7

Project and Session Overview 11

Streamlit on CML 21

Go to CML Workspace 22

Create a New Project Using the Streamlit AMP 23

Explore the Streamlit Project 29

Modify the Application 37

Delete the Project 38

Bonus 39

Data - Access, Audit, and Mask 40

Access Data 40

Visualize Duocar Data 50

Create Workload Password (if needed) 56

Create a New Dataset 68

Create a Dashboard 71

Create Filters 91

Share the Dashboard 95

Experiment Tracking 97

Create a New Project from Github 97

View Code and Run Job 100

Creating an Experiment 108

Change the Input and Compare Runs 117

Commit the Changes to Git 122

Using Workbench for Lecture and Exercises 127

Autoscaling, Performance, and GPU Settings 137

View Workspace Details and Allocated GPUs 137

Create a New Project 139

Create a Session without a GPU 141

Create a Session with a GPU 144

Continuous Model Monitoring with Evidently 151

Start the Continuous Model Monitoring AMP 151

Launch the Price Regressor Monitoring dashboard 155
Explore drift and variations in the model performance 159
Identify the file that creates the Evidently dashboard 160

DSCI-272: Predicting with Cloudera Machine Learning

Exercises

- Introduction to CML on CDP
- Streamlit on CML
- Data - Access, Audit, and Mask
- Experiment Tracking
- Visualize Duocar Data
- Workbench Lecture and Exercises
- Autoscaling, Performance, and GPU Settings
- Continuous Model Modeling with Evidently AI

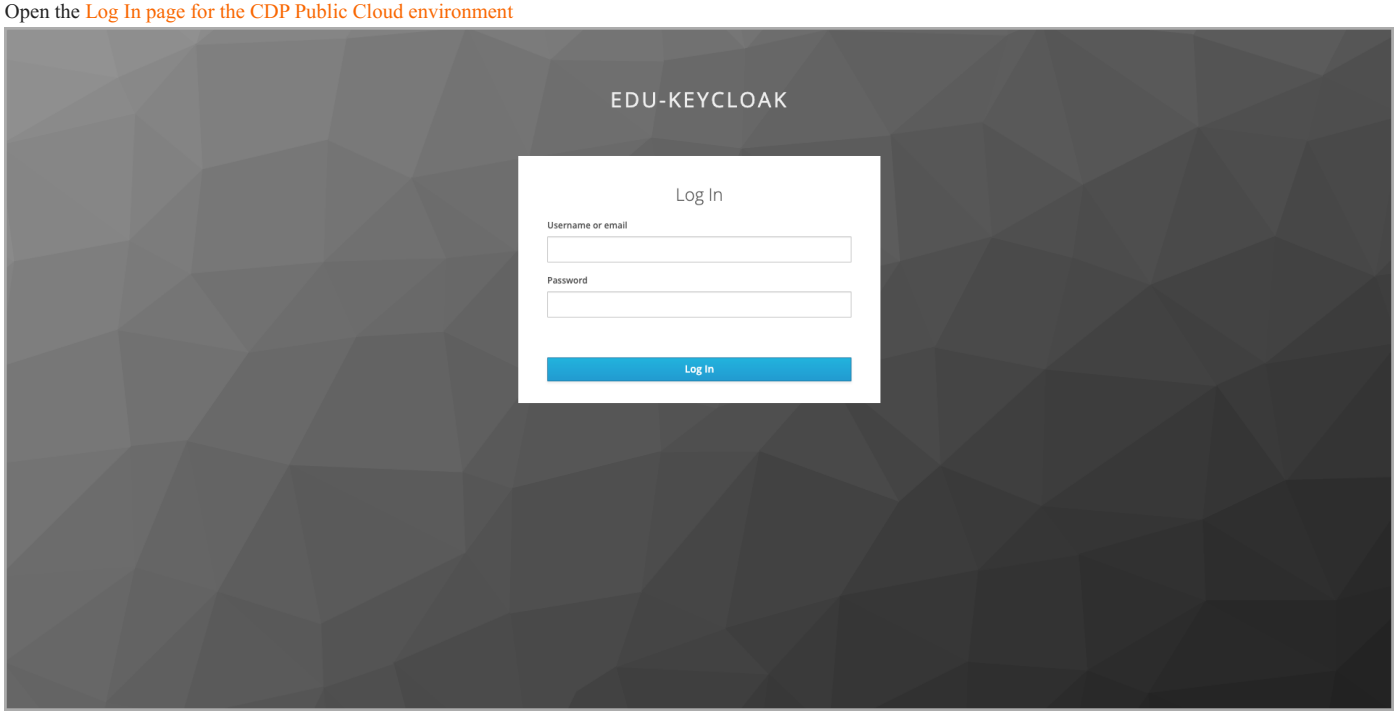
Introduction to CML on CDP

This exercise will introduce the Cloudera Data Platform (CDP) and Cloudera Machine Learning (CML). In this exercise, you will:

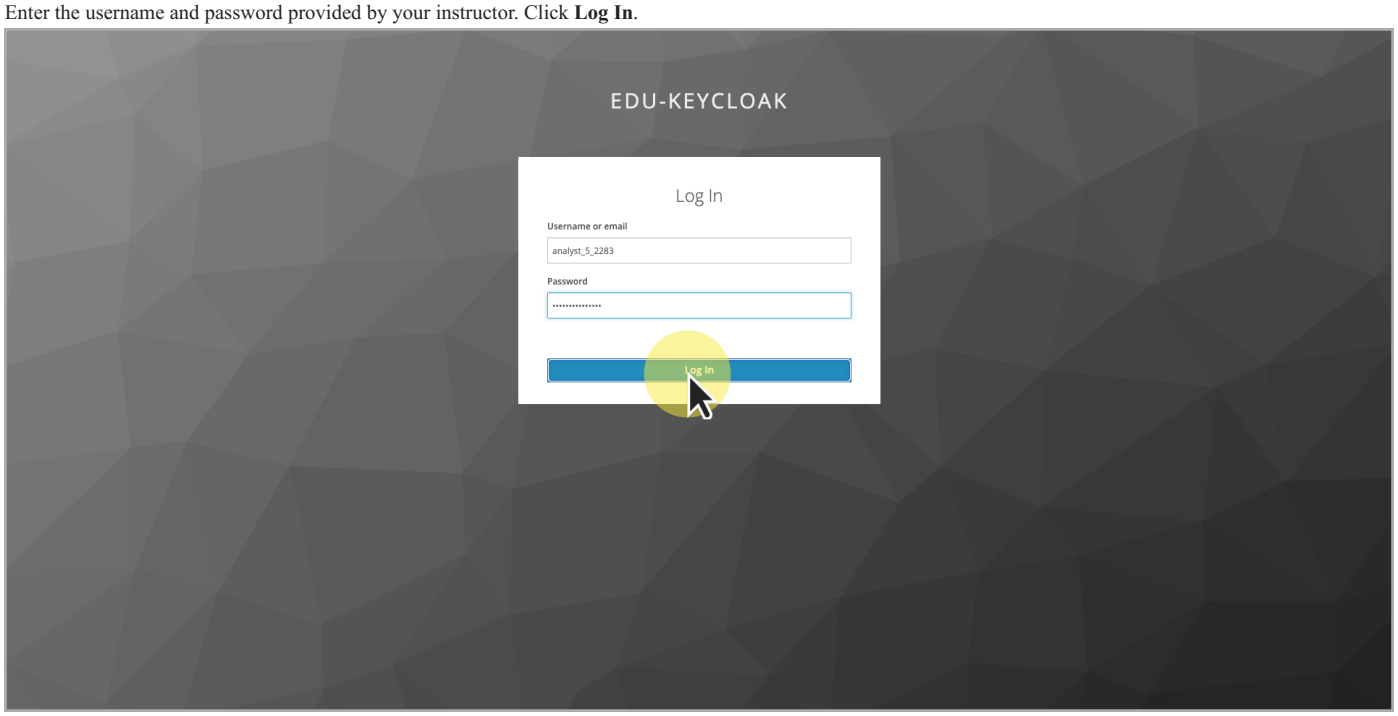
- login to the Cloudera Data Platform exercise environment,
- view the Cloudera Machine Learning workspace,
- create a new CML project,
- create a new session,
- and delete a CML project.

Login to Cloudera Data Platform

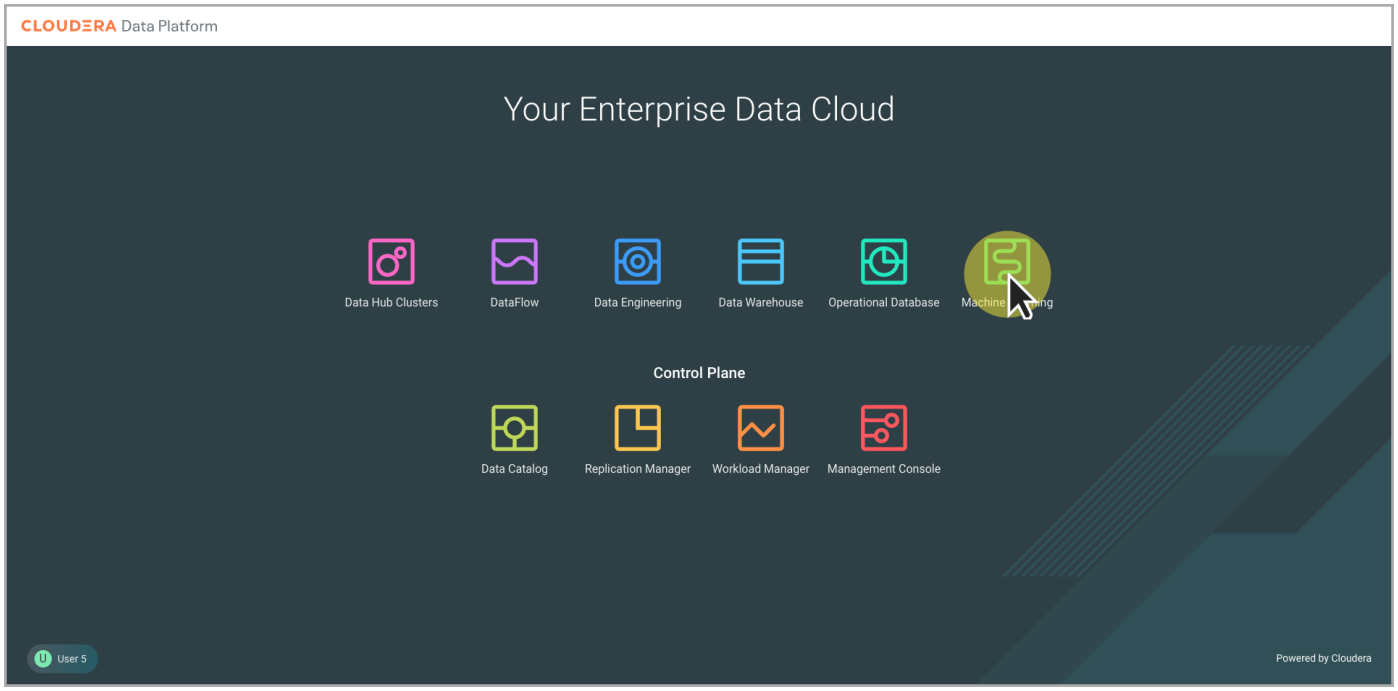
1. Open the Log In page for the CDP Public Cloud environment



2. Enter the username and password provided by your instructor. Click **Log In**.

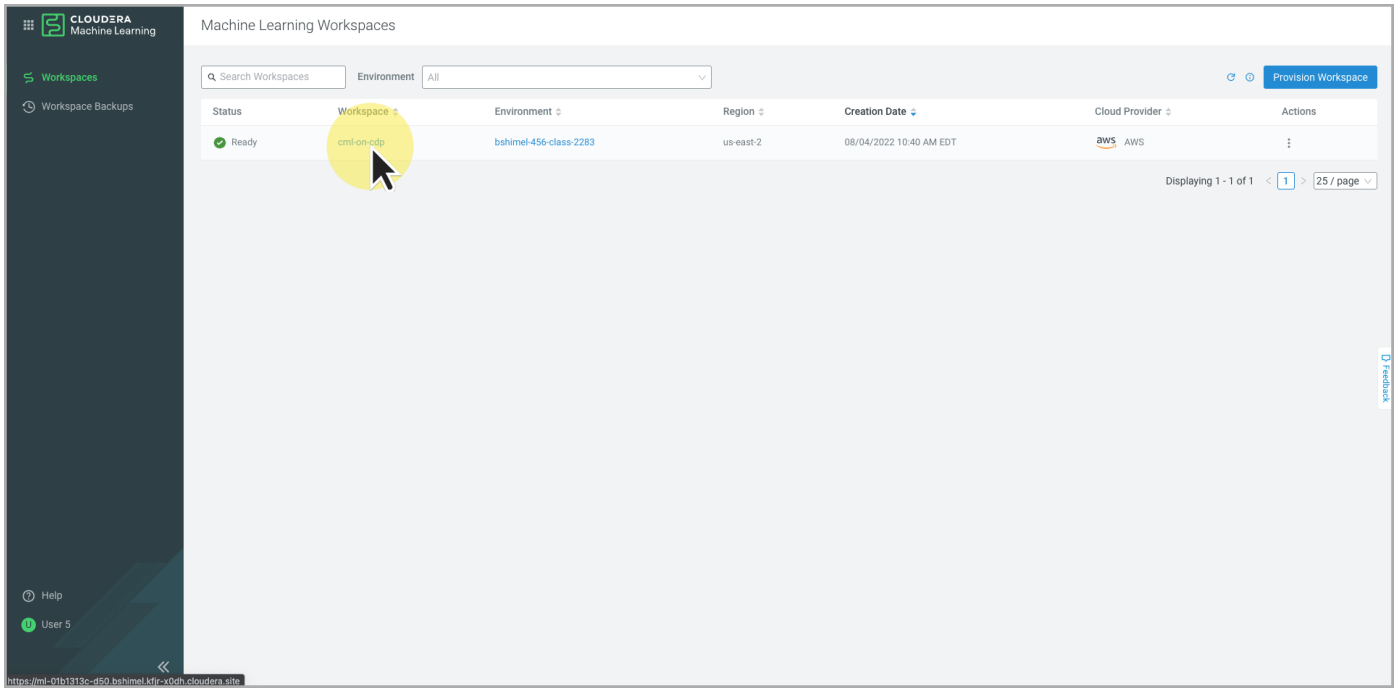


3. The Cloudera Data Platform page is displayed. Some of the additional features of the platform will be explored in other exercises. For now, click **Machine Learning**.

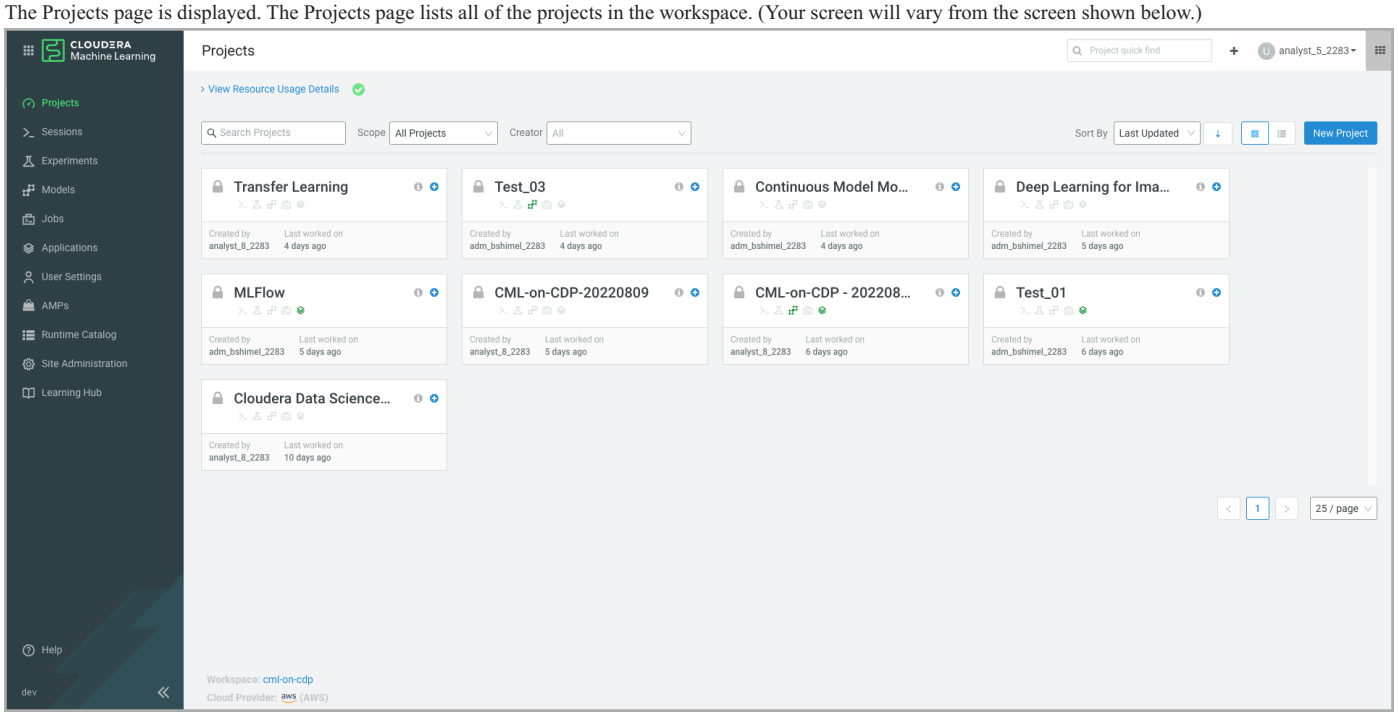


Overview of CML Workspace

1. The list of Machine Learning workspaces is displayed. In this case, one workspace is provisioned. Click on **cml-on-cdp**.

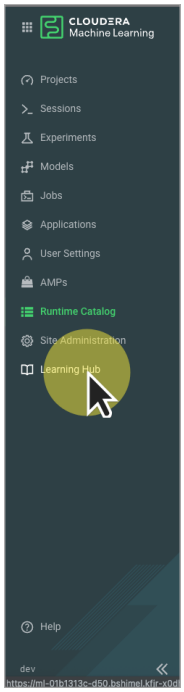


2. The Projects page is displayed. The Projects page lists all of the projects in the workspace. (Your screen will vary from the screen shown below.)

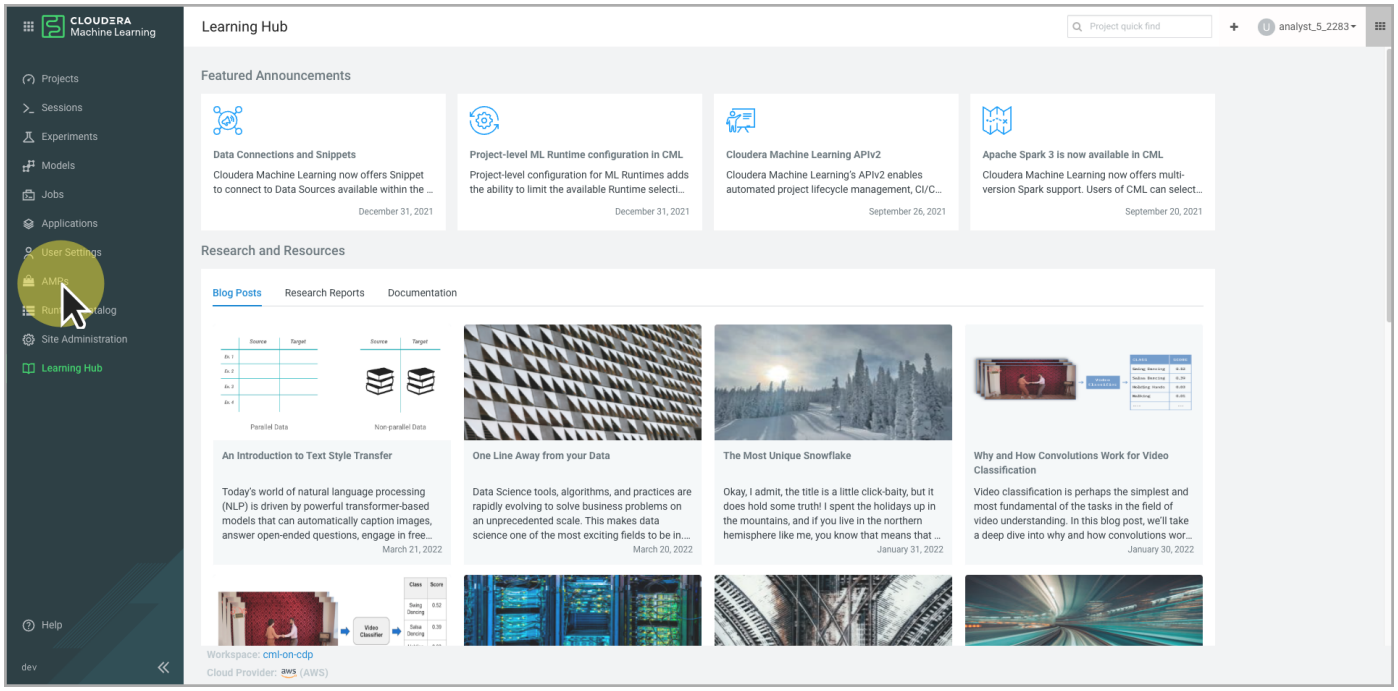


The workspace has several helpful features.

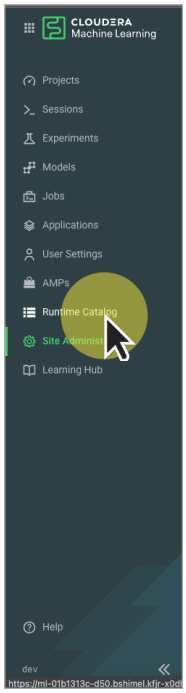
1. Select **Learning Hub** from the workspace menu.



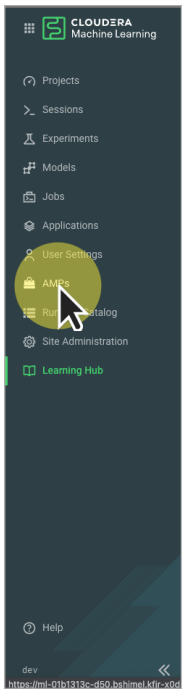
2. View the Learning Hub content. The Learning Hub is a great resource to learn about the new features in CML and read blog posts, research reports, and documentation.



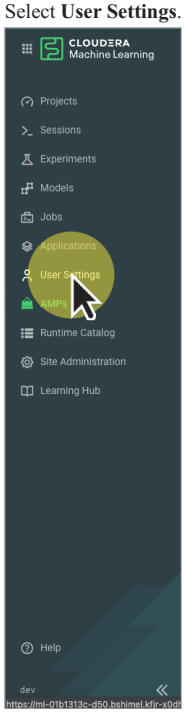
3. Select **Runtime Catalog**. The Runtime Catalog shows a list of available runtimes and allows new runtimes to be added.



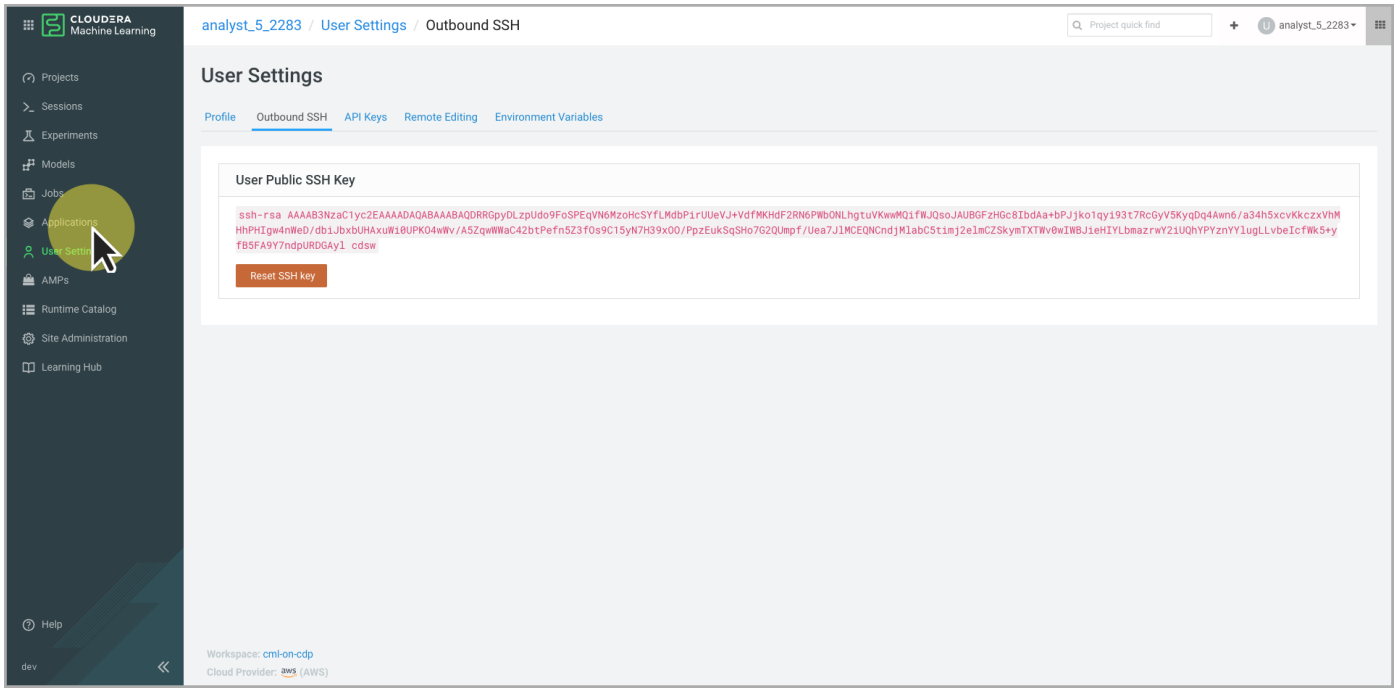
4. Select **AMPs**. The AMPs page shows a list of availble Applied ML Prototypes (AMPs).



5. Select **User Settings**.



6. Select the **Outbound SSH Key** tab. The Outbound SSH Key is used for connecting to external resources, for example Github.



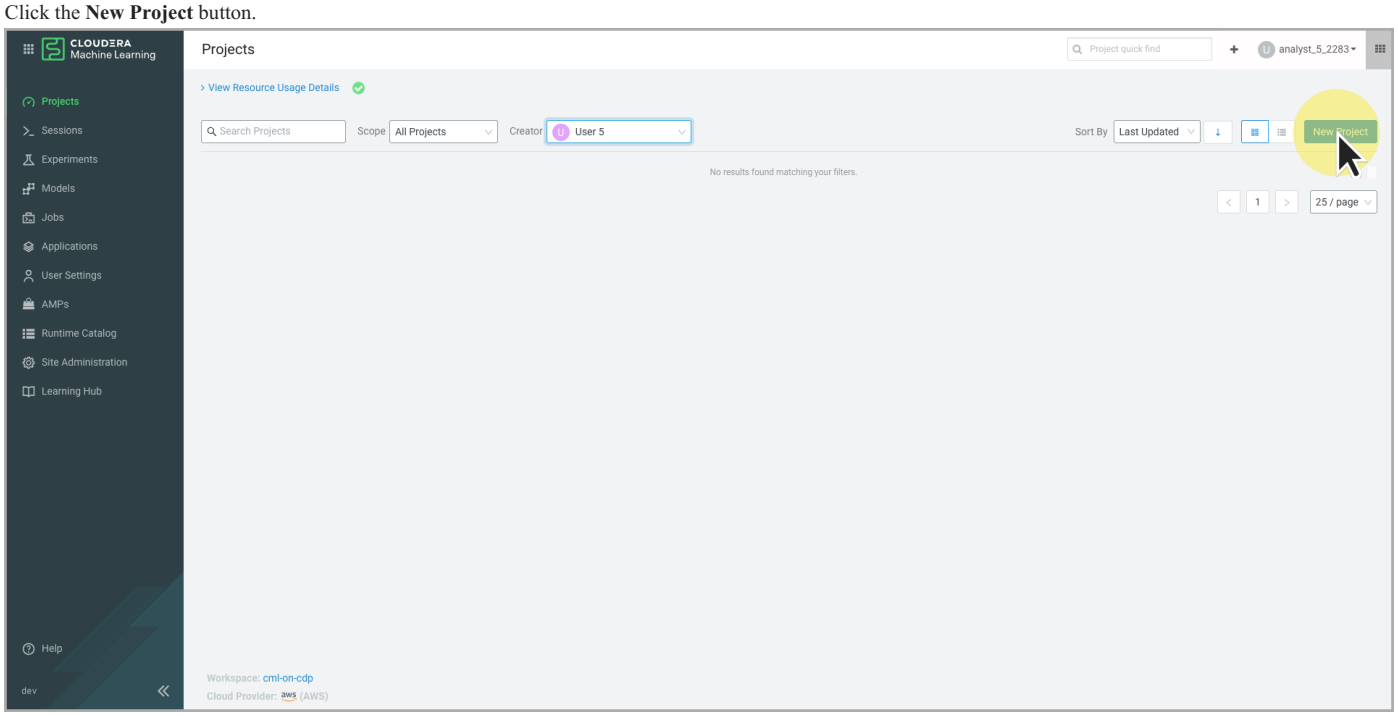
Applications, Jobs, Models, Experiments, and Sessions will be covered later.

Project and Session Overview

Create a New Project

1. Select **Projects** from the workspace menu.

2. Click the **New Project** button.



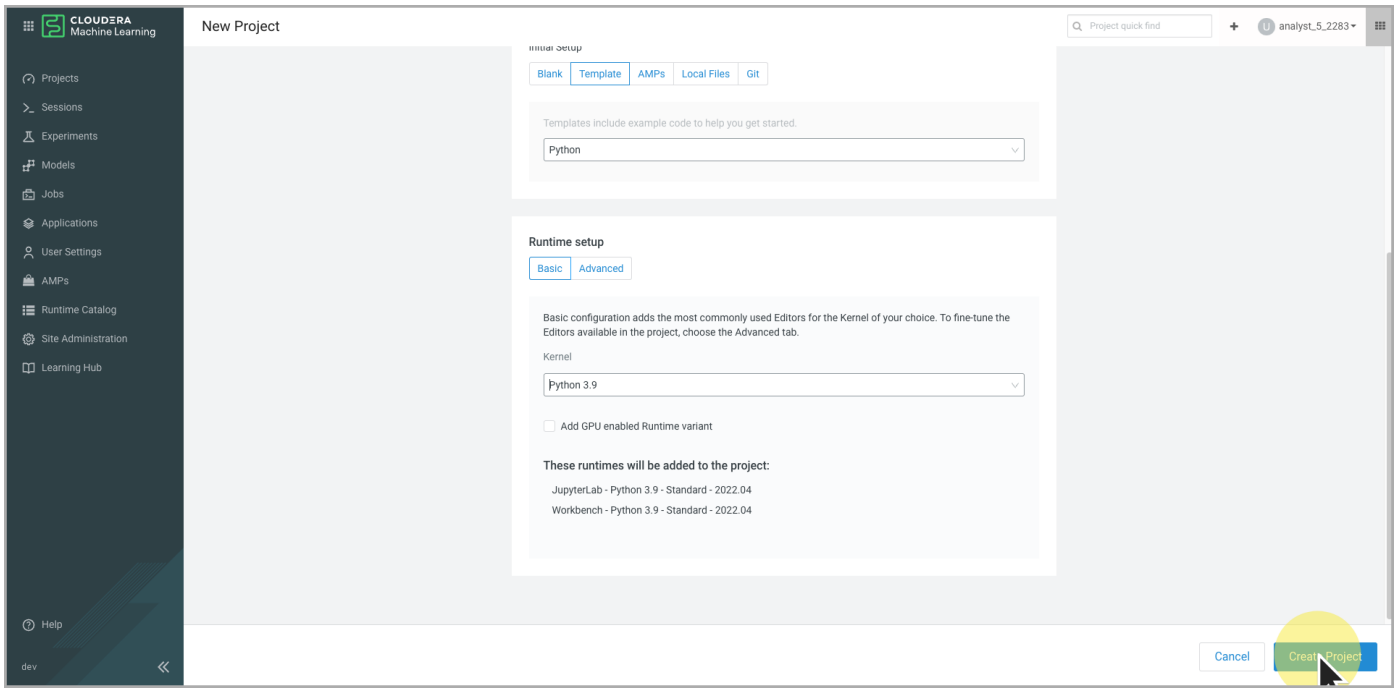
3. For **Project Name**, enter `Student # - First Project`, where # is your student number.

4. Leave the following items set to their default values:

- **Description**: Empty
- **Project Visibility**: **Private**
- **Initial Setup**: **Template - Python**
- **Runtime setup**: **Basic**

5. Select **Python 3.9** as the runtime **Kernel**.

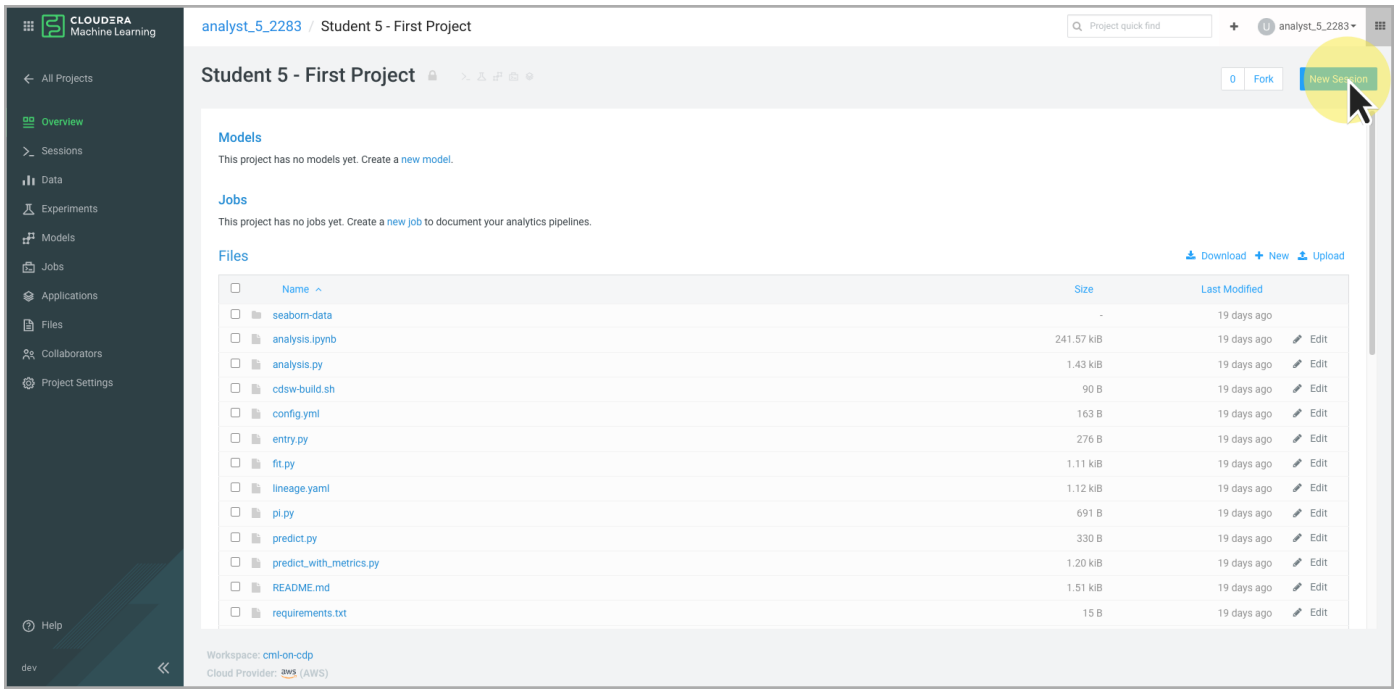
6. Click **Create Project**.



Create a New Session

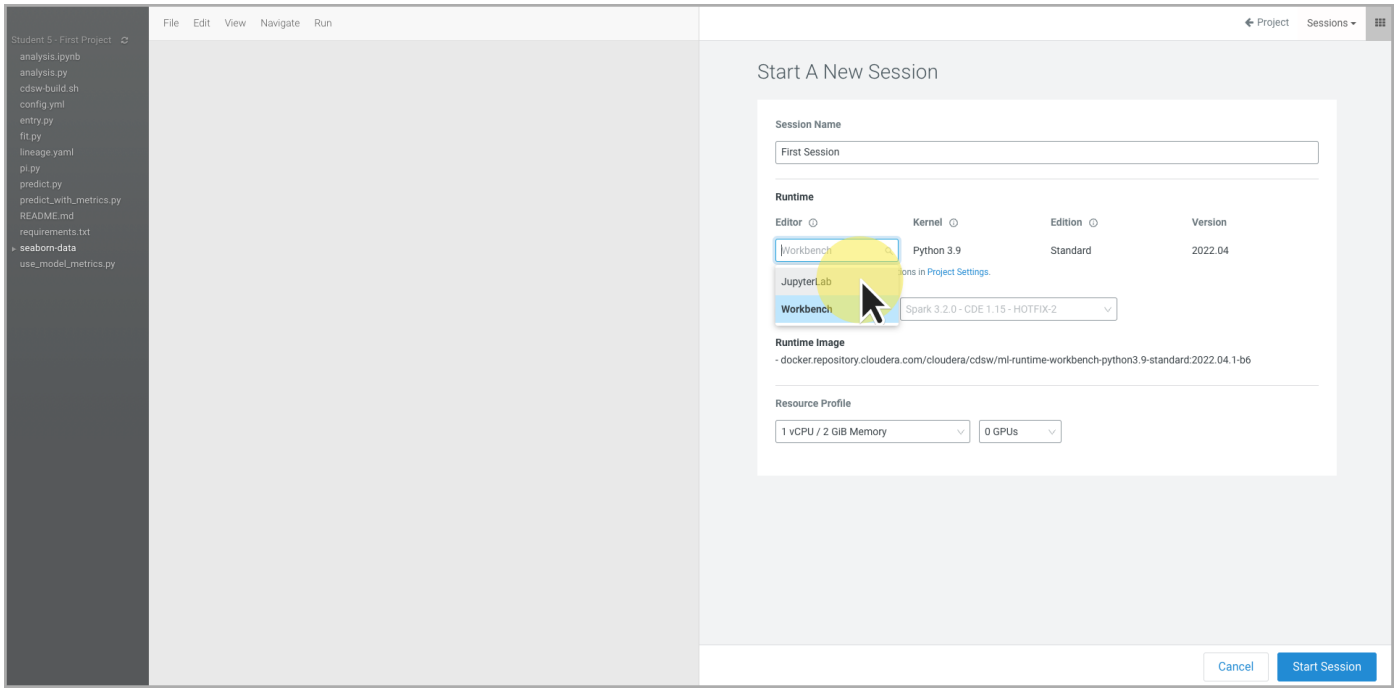
After the new project is created, the project overview page is displayed and the left-hand menu displays project items instead of workspace items. Next, create a session to interact with your project.

1. Click **New Session**.

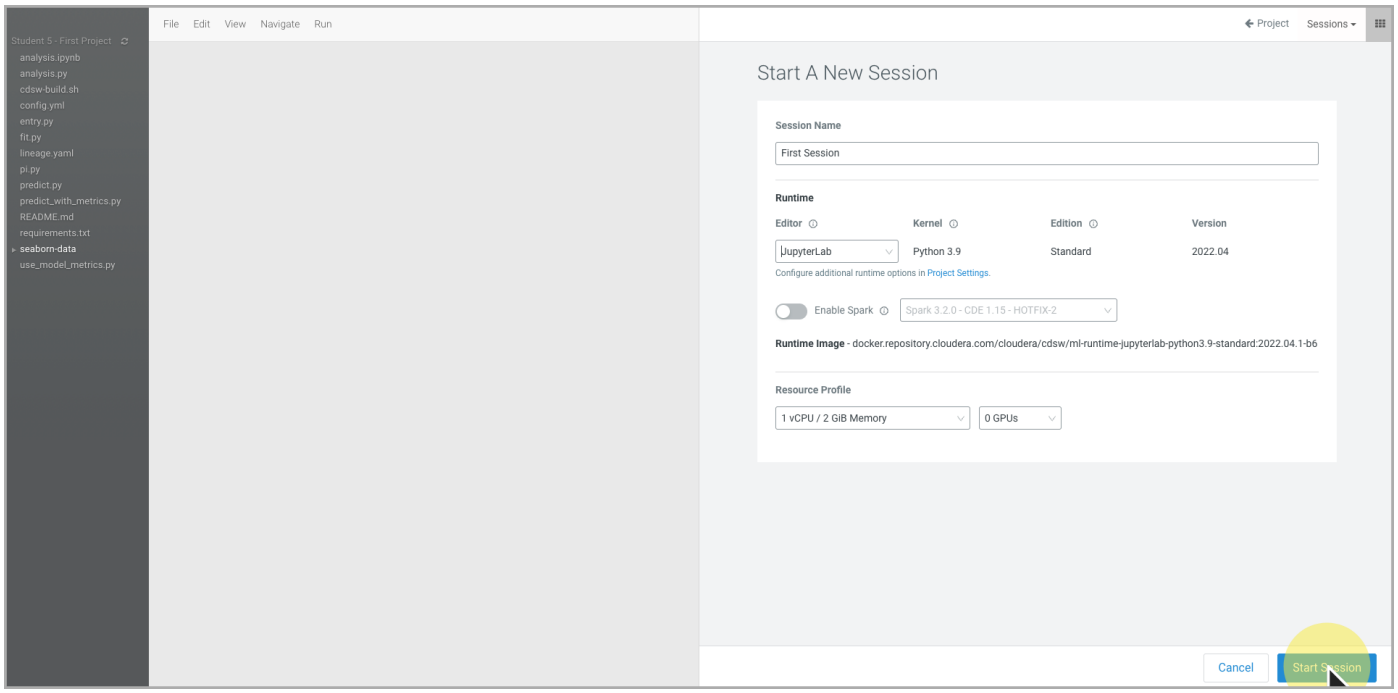


2. Enter `First Session` for the **Session Name**.

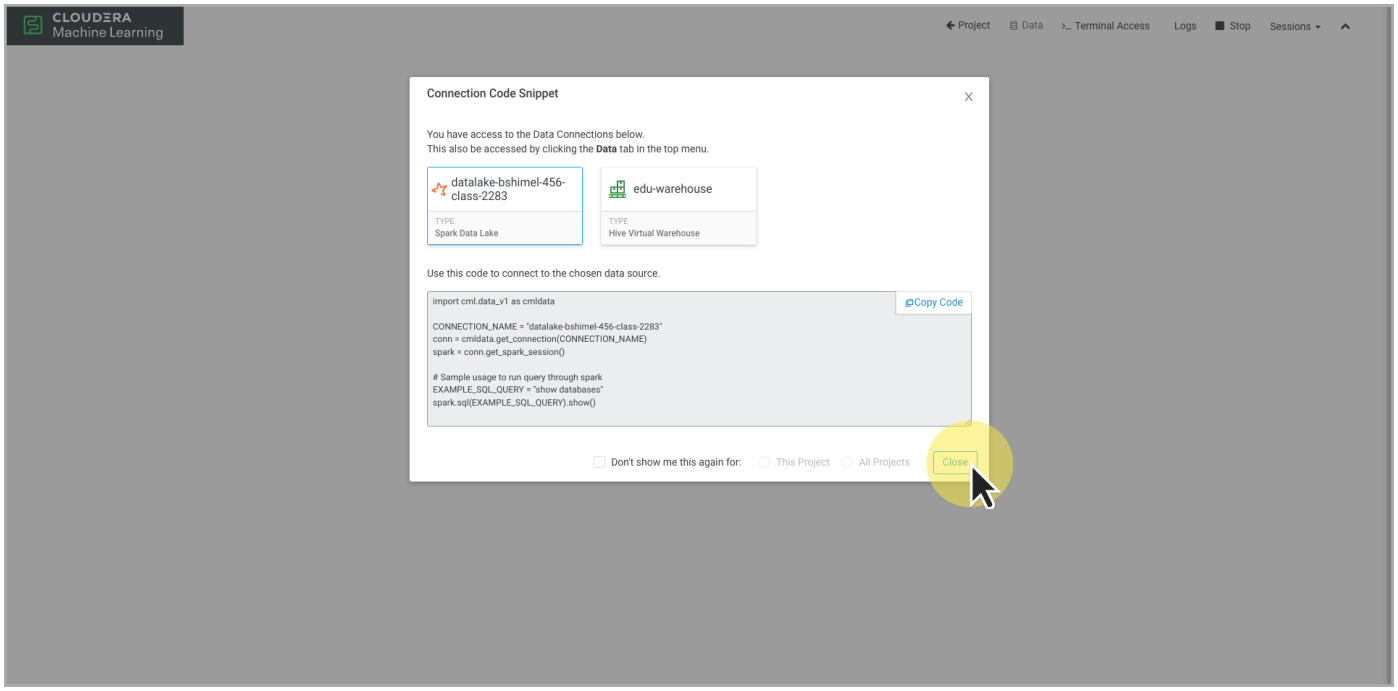
3. Select **JupyterLab** as the **Editor**.



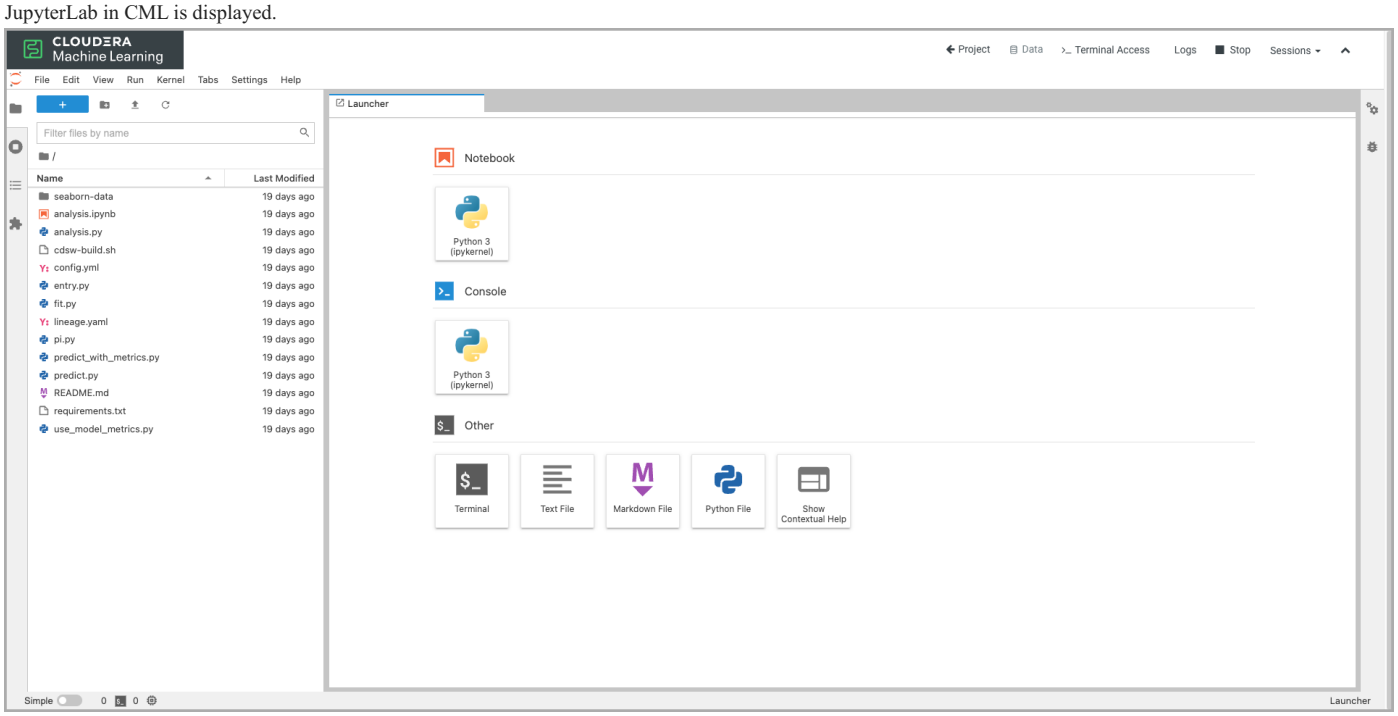
4. Click **Start Session**.



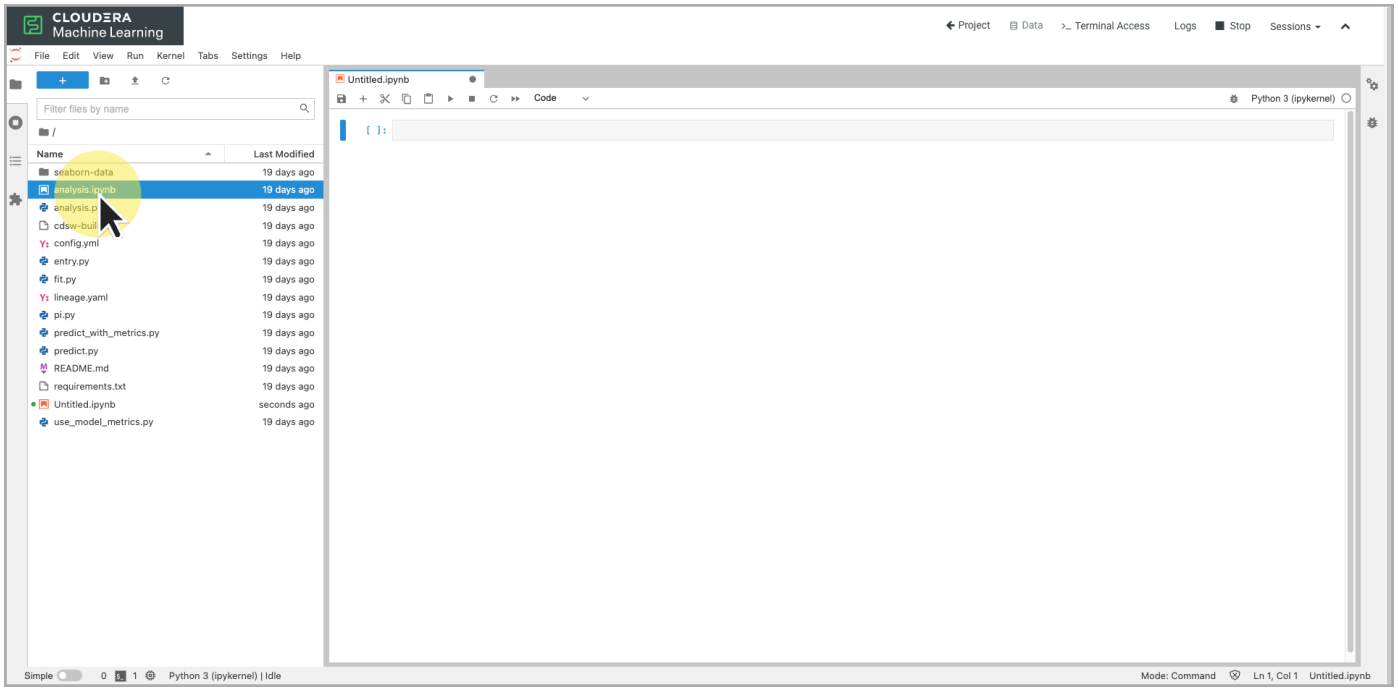
5. Close the **Connection Code Snippet** dialog.



6. JupyterLab in CML is displayed.



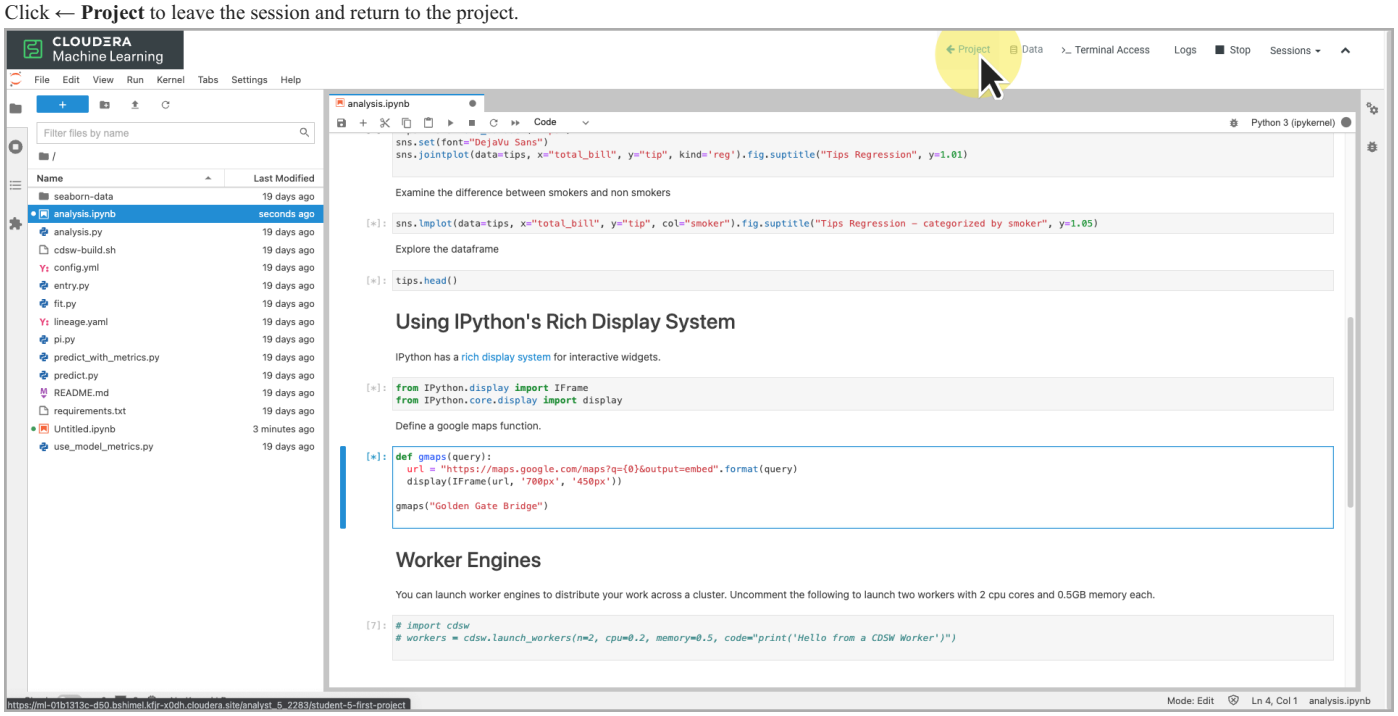
7. Double-click the `analysis.ipynb` notebook to open it.



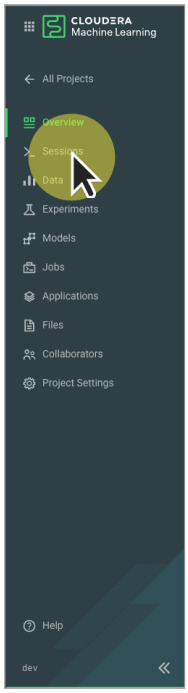
Note

This exercise is just an introduction to creating a session that uses JupyterLabs. If you are are familar with JupyterLabs and would like to work though the analysis.ipynb notebook, be sure to uncomment `%pip install -r requirements.txt` in the first cell.

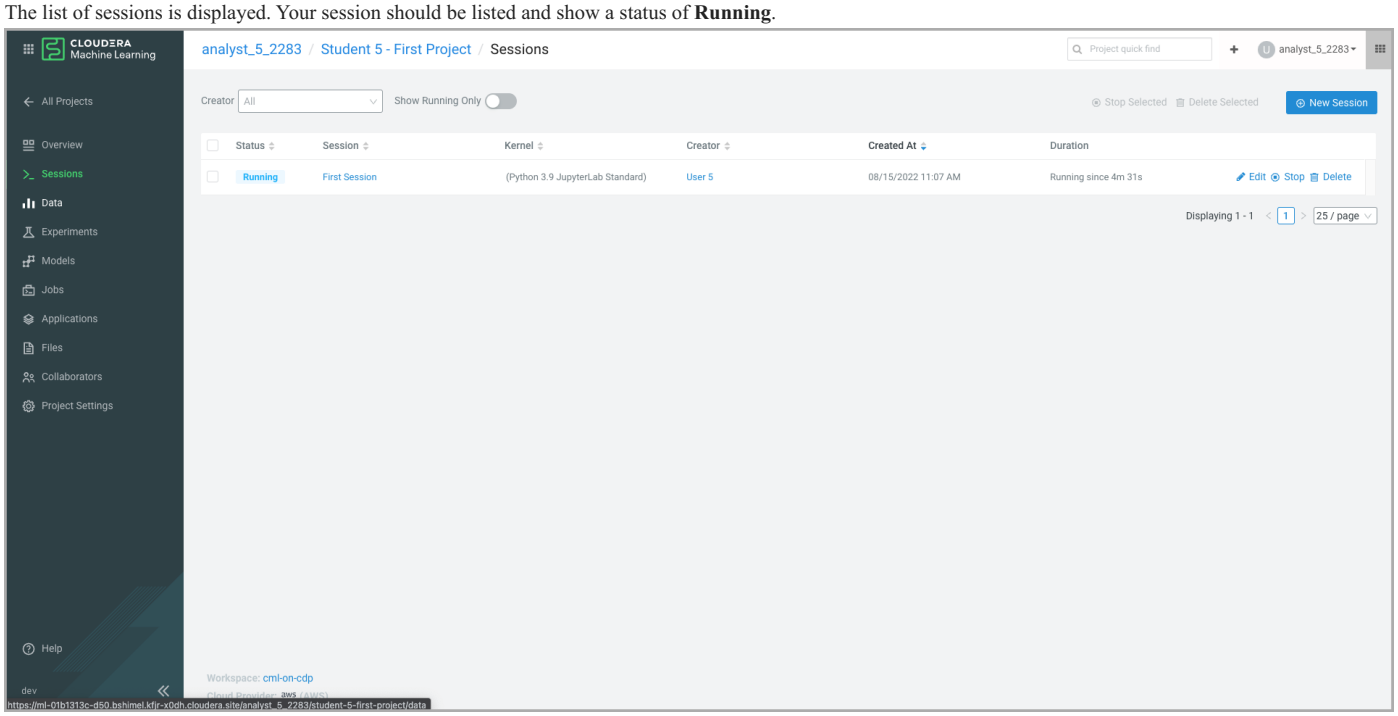
8. Click ← **Project** to leave the session and return to the project.



9. Select **Sessions**.

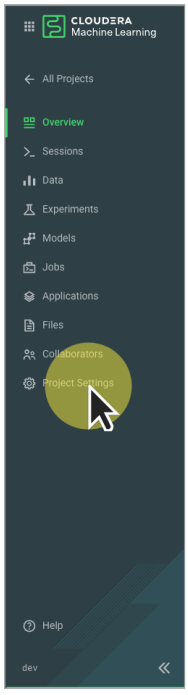


10. The list of sessions is displayed. Your session should be listed and show a status of **Running**.

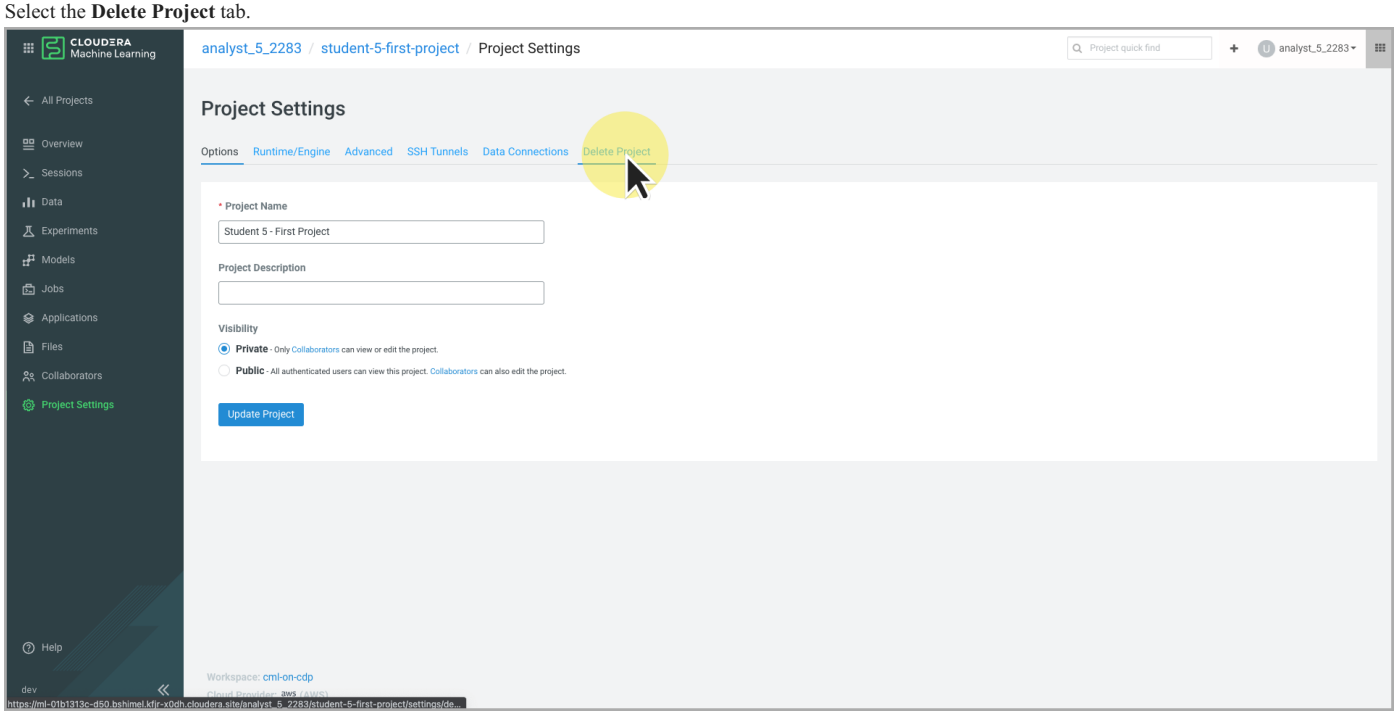


Delete a Project

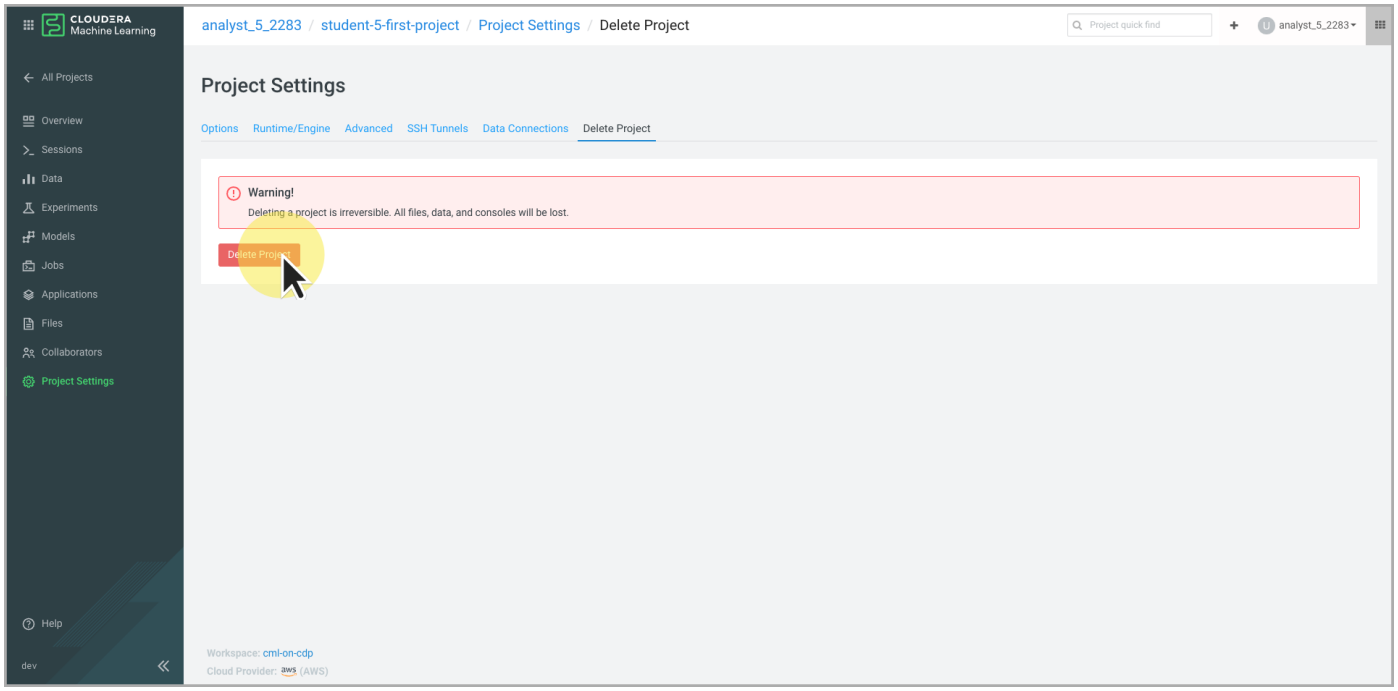
1. Select Project Settings.



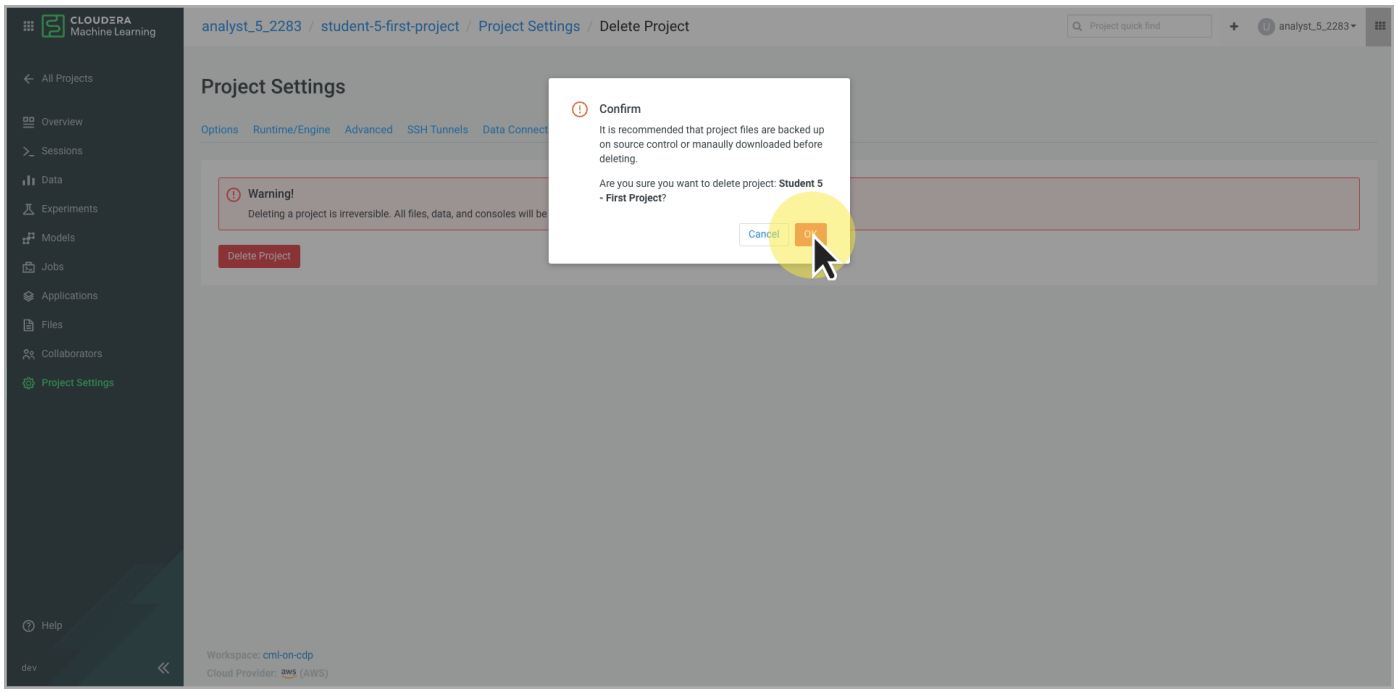
2. Select the **Delete Project** tab.



3. Click **Delete Project**.



4. Click **OK** to confirm.



End of Exercise

Streamlit on CML

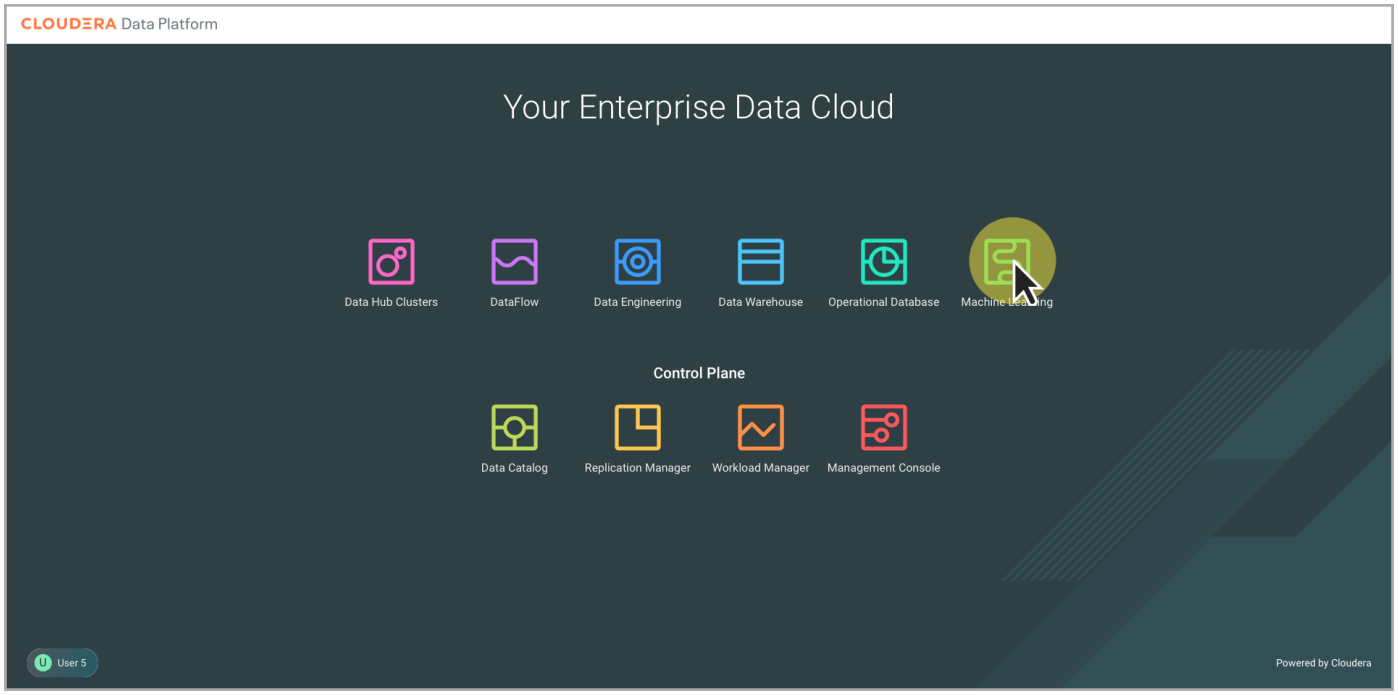
"Streamlit turns data scripts into shareable web apps in minutes." Streamlit is a web application framework that allows data scientists to quickly build web applications to share data and analytics.

In this exercise, you will use an AMP (Applied ML Prototype) to deploy a simple Streamlit application using CML. You will learn how to deploy an AMP and how applications work in CML.

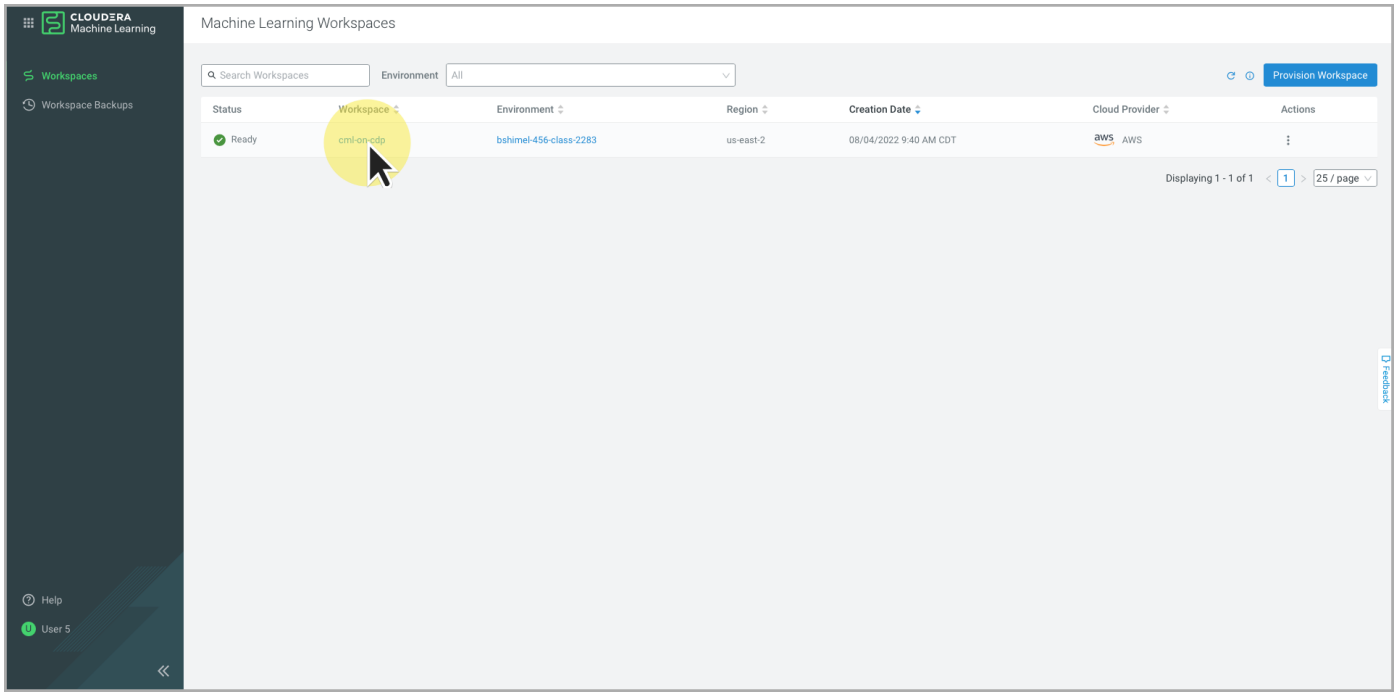
Go to CML Workspace

Begin the exercise by nagivating to your workspace.

1. Click **Machine Learning**.



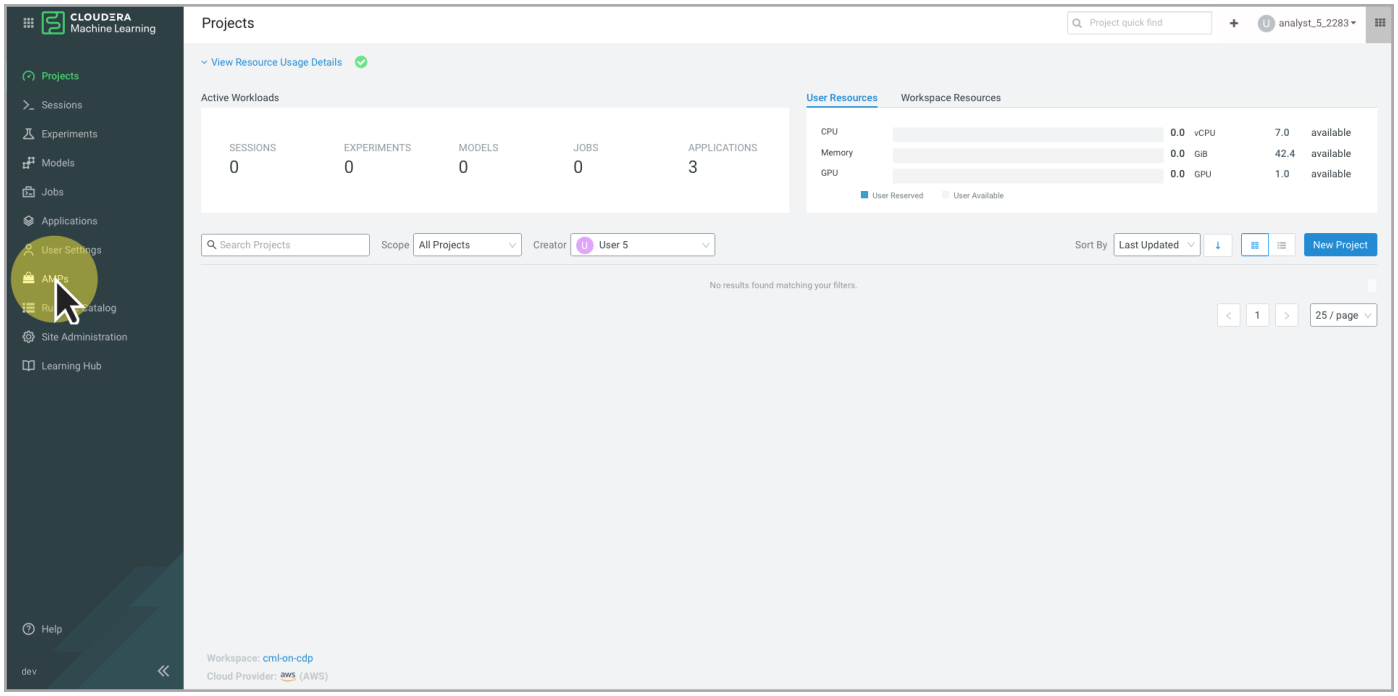
2. Select the workspace.



Create a New Project Using the Streamlit AMP

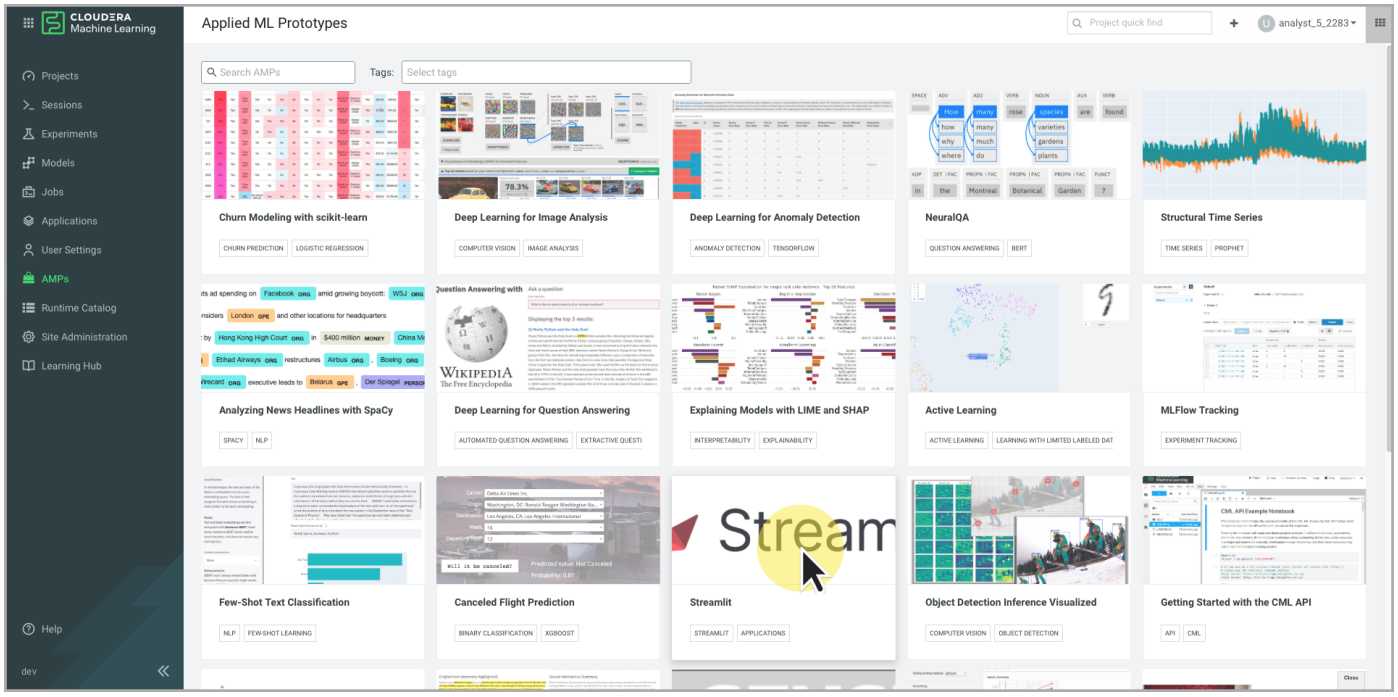
Creating a project from an AMP is easy.

1. Click **AMPs** in the workspace menu.



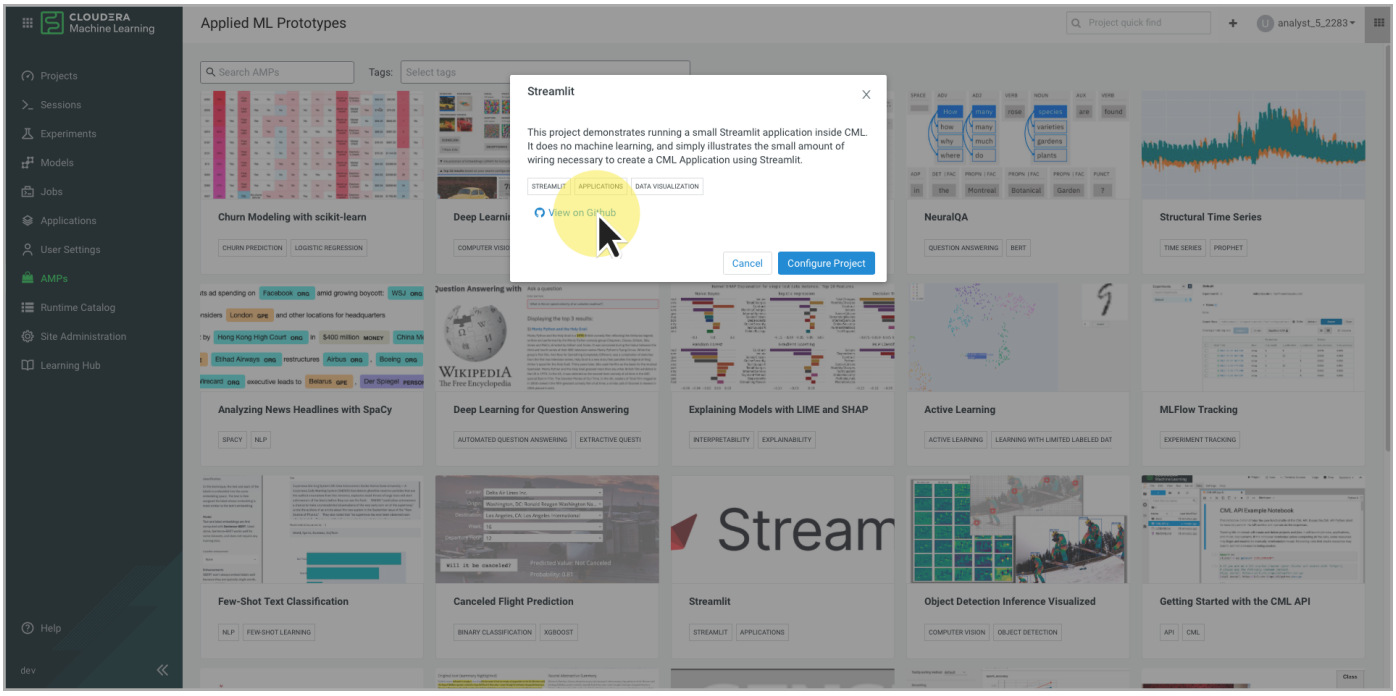
You will see a catalog of AMPs. New AMPs are being added all of the time. In addition to AMPs provided by Cloudera, you can create your own AMPs and your own AMP catalog.

1. Click the **Streamlit** tile.

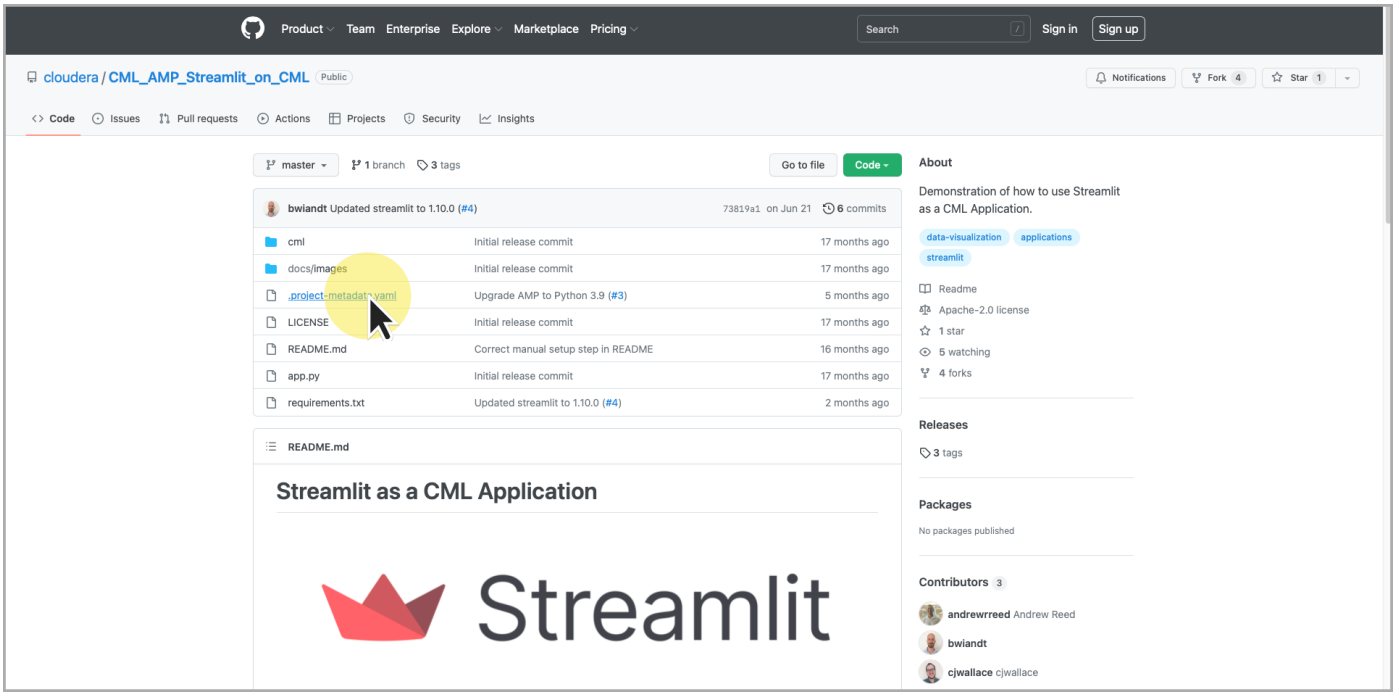


A dialog will popup that describes the AMP. The dialog also contains a link to the Github repository for the AMP. Viewing an AMP's repository is a good way to learn more about the AMP and how AMPs are created, in general.

1. Click **View on Github**.



2. Click the `project-metadata.yaml` file.



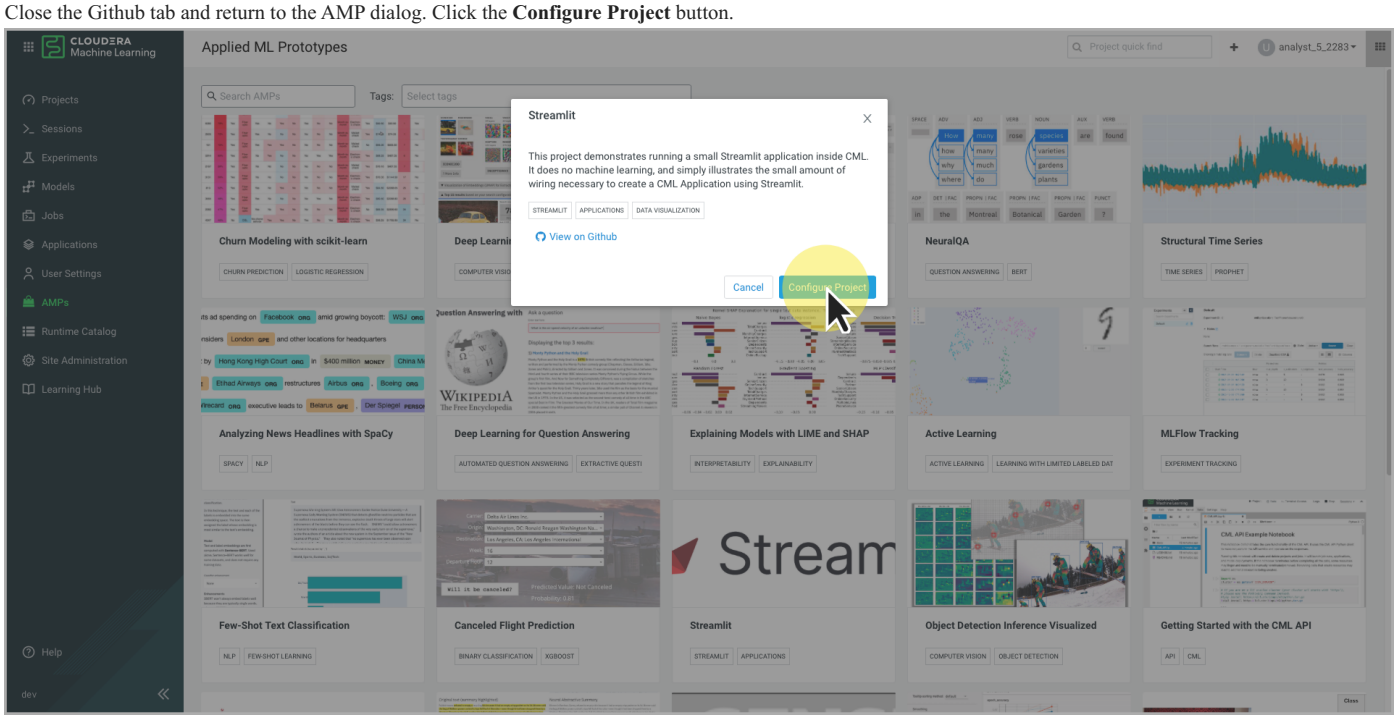
In the project-metadata.yaml file, you can see the steps that are used to deploy the AMP. In this case, you can see that AMP runs a session to install the Python dependecies and then starts a new application called Streamlit App.

```
name: Streamlit
description: Run a Streamlit app inside CML.
author: Cloudera Inc.
specification_version: 1.0
prototype_version: 2.0
date: "2023-03-24"

runtimes:
```

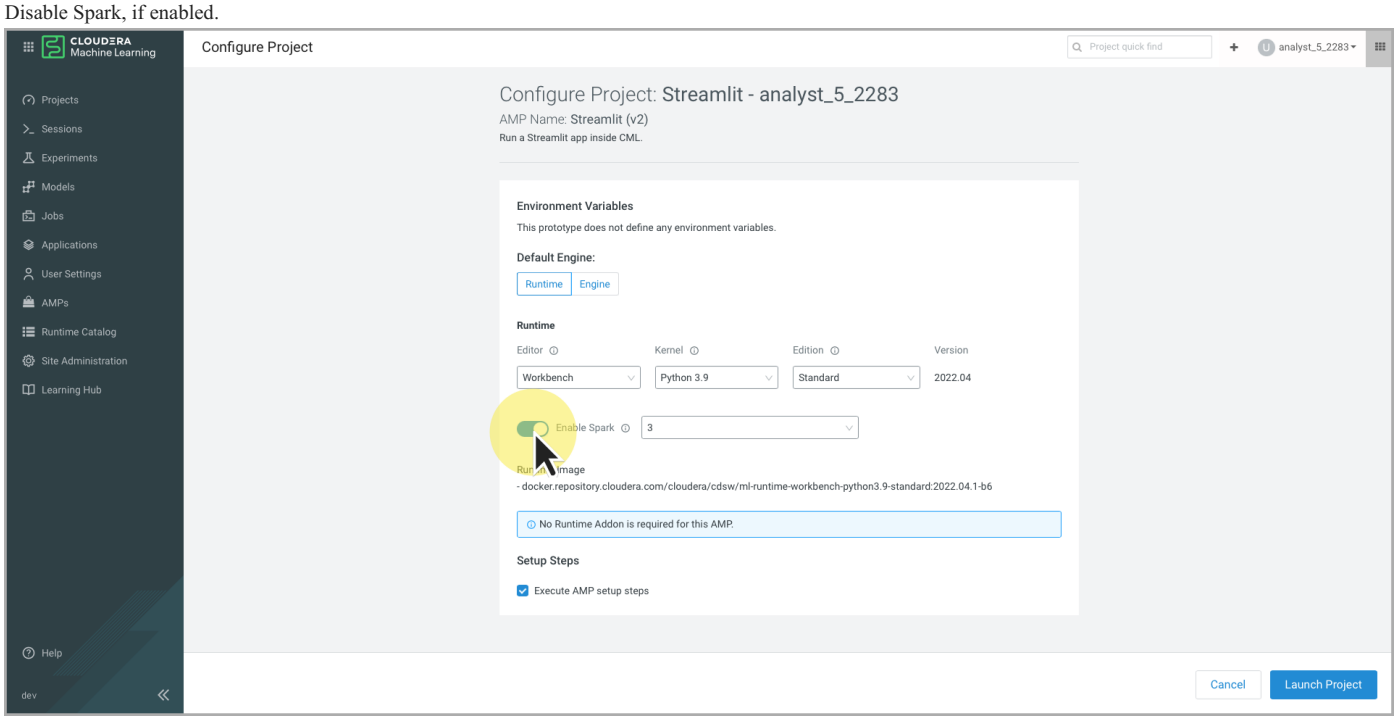
```
- editor: Workbench
    kernel: Python 3.9
    edition: Standard

tasks:
- type: run_session
    name: Install Dependencies
    script: cml/install_dependencies.py
    kernel: python3
    cpu: 1
    memory: 2

- type: start_application
    name: Streamlit App
    subdomain: streamlit
    script: cml/launch_app.py
    short_summary: Start Streamlit application
```

```
    environment_variables:
    TASK_TYPE: START_APPLICATION
```

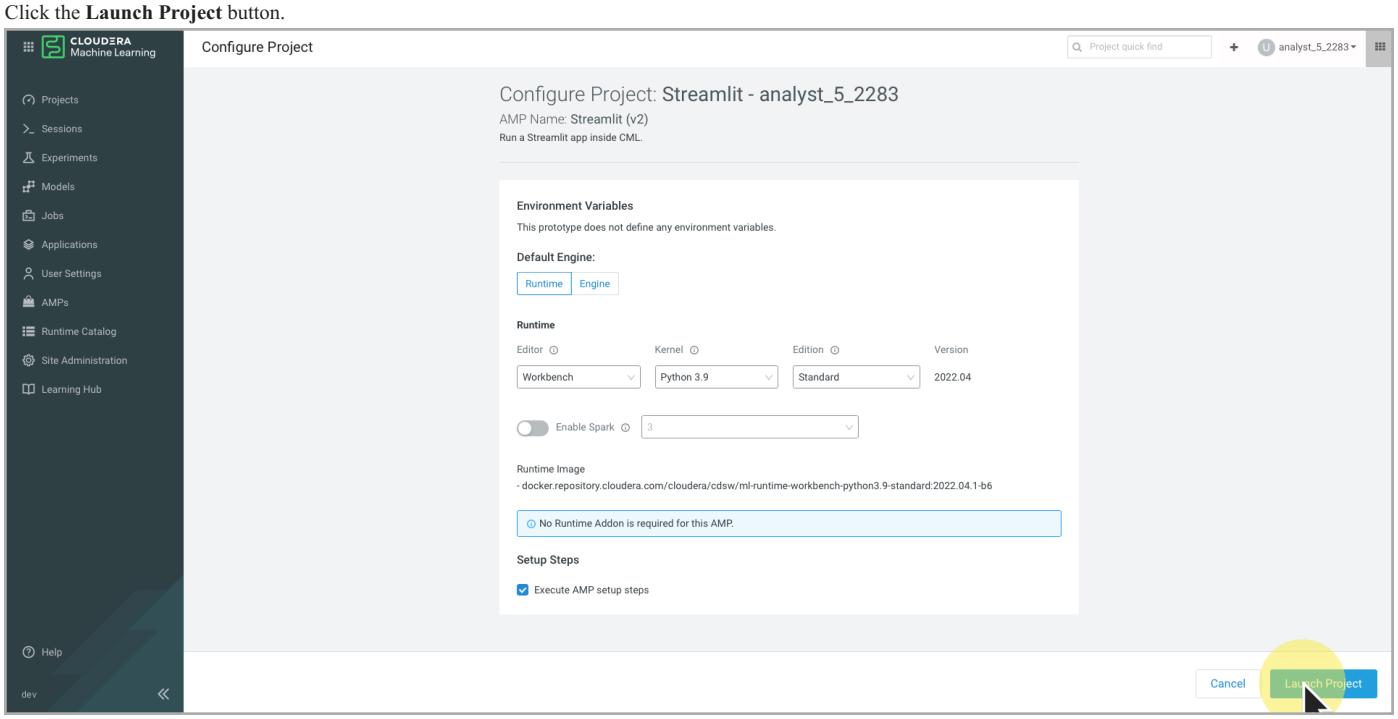
1. Close the Github tab and return to the AMP dialog. Click the **Configure Project** button.



2. Disable Spark, if enabled.

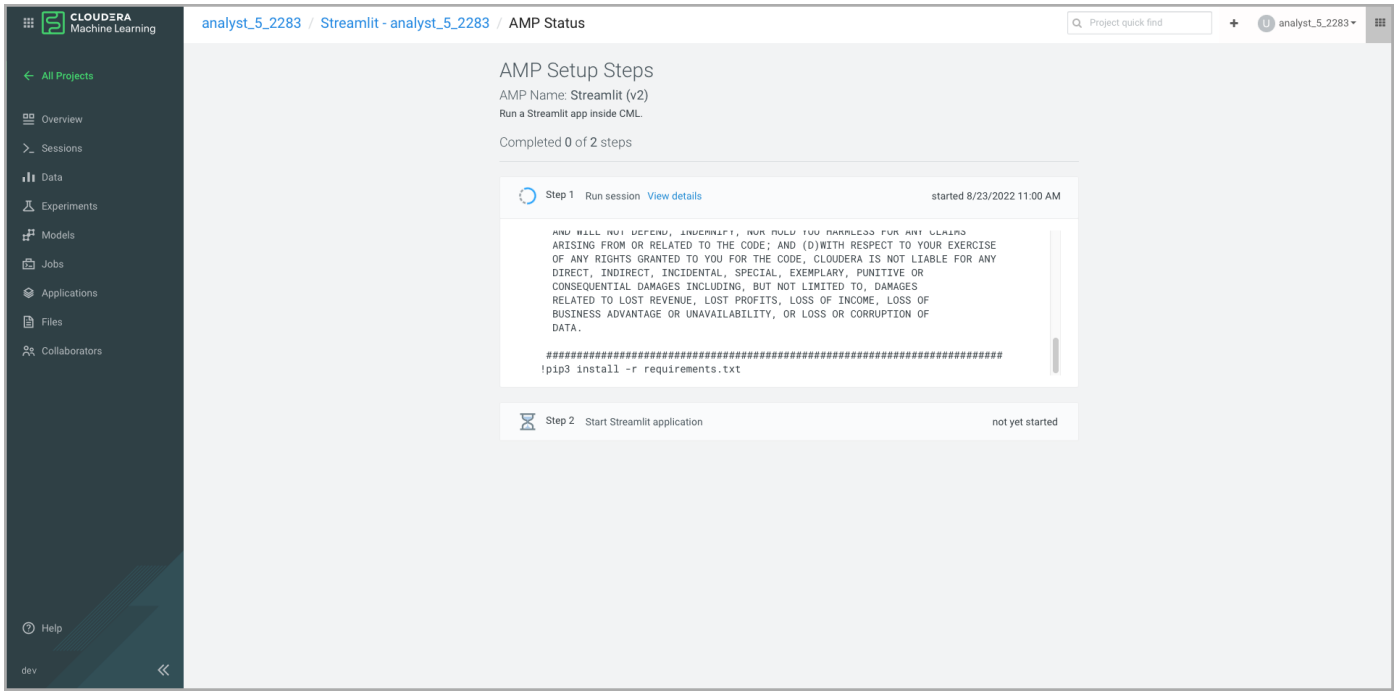


3. Click the **Launch Project** button.



The AMP will begin building. You will see the two steps that were in the `project-metadata.yaml` file execute.

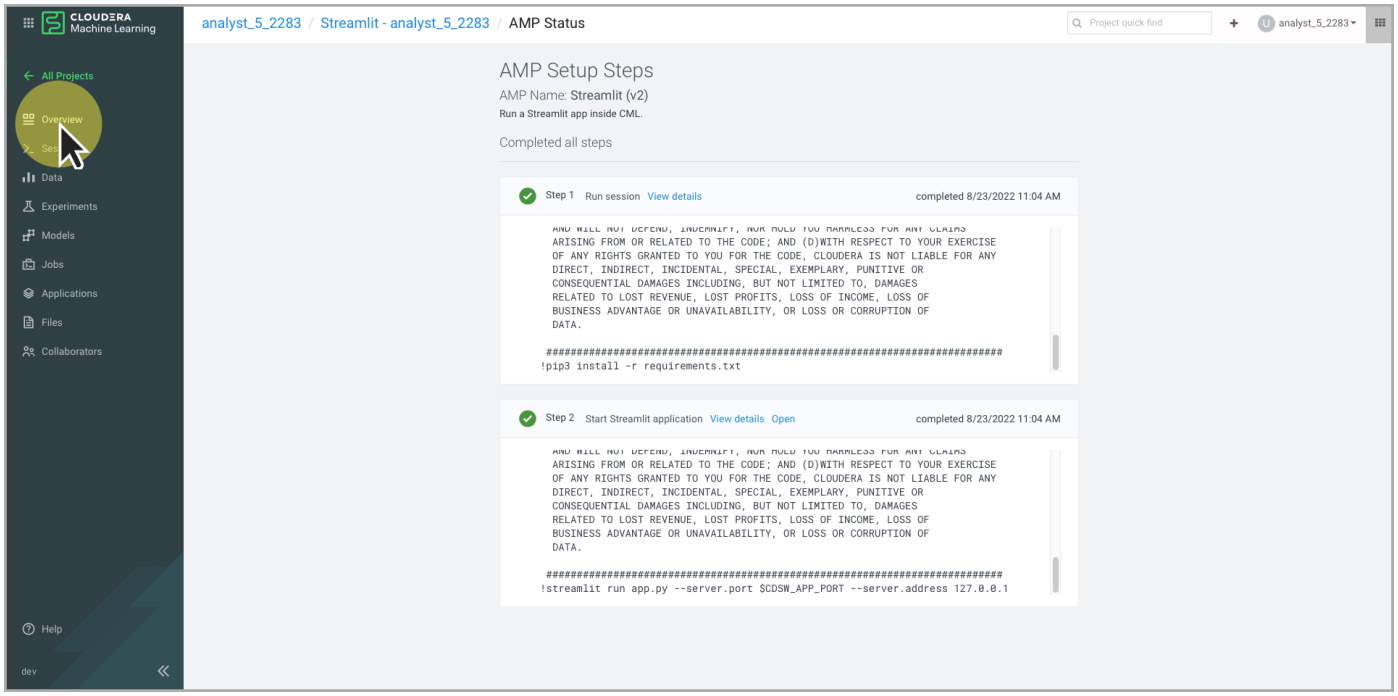
1. Wait for AMP to complete. It will take approximately five minutes for the AMP to complete both steps.



Explore the Streamlit Project

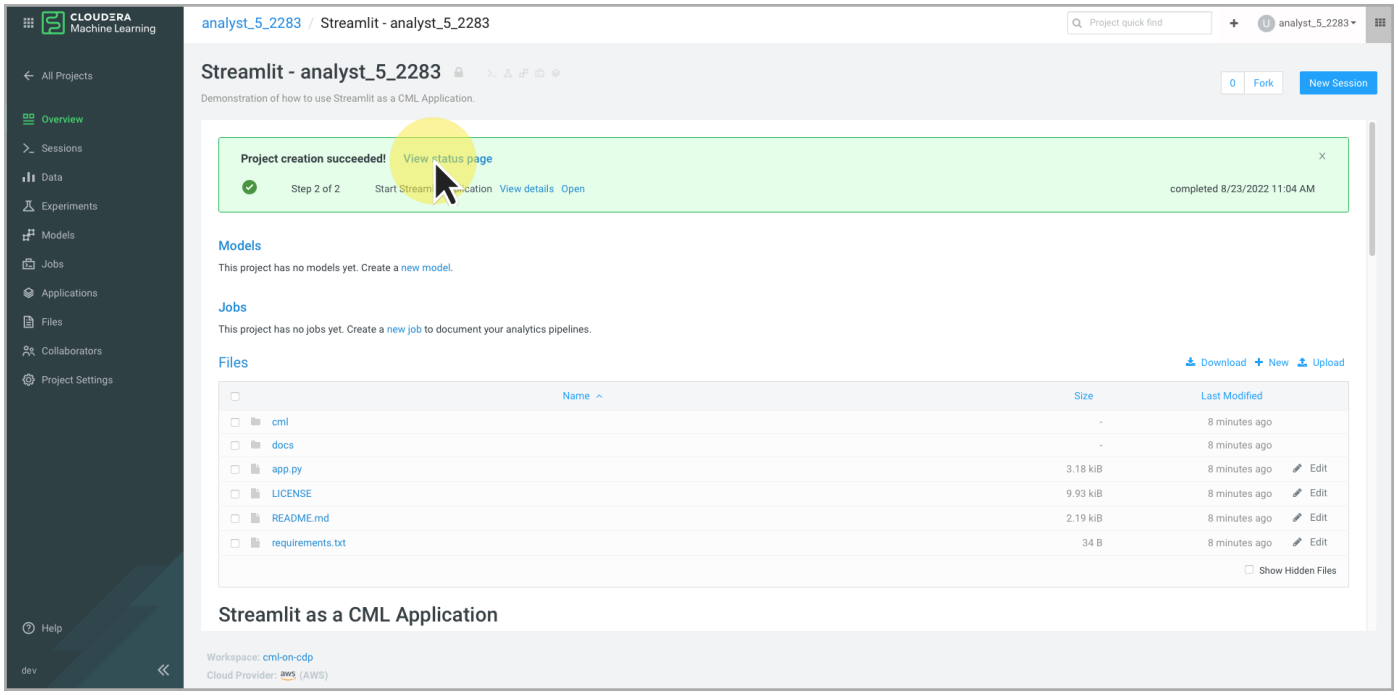
Now that the AMP has completed creating a new project, you can view the contents of the project to see what was created.

1. Click **Overview** in the project menu.

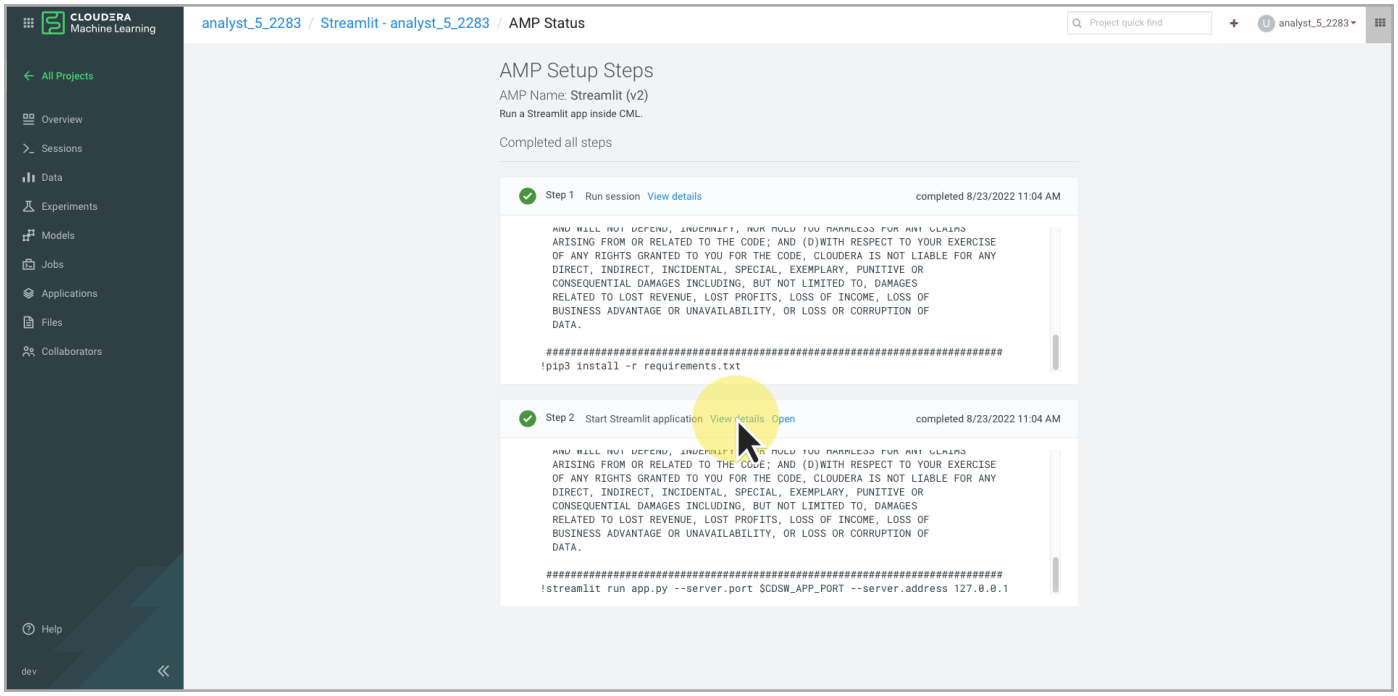


The project overview shows any models or jobs that were created, the files contained in the project, and the contents of the `README.md` markdown file which describe the project. At the top of the overview page is the message regarding the project creation status which contains a useful link to the return to the project creation status page.

1. Click **View status page**.

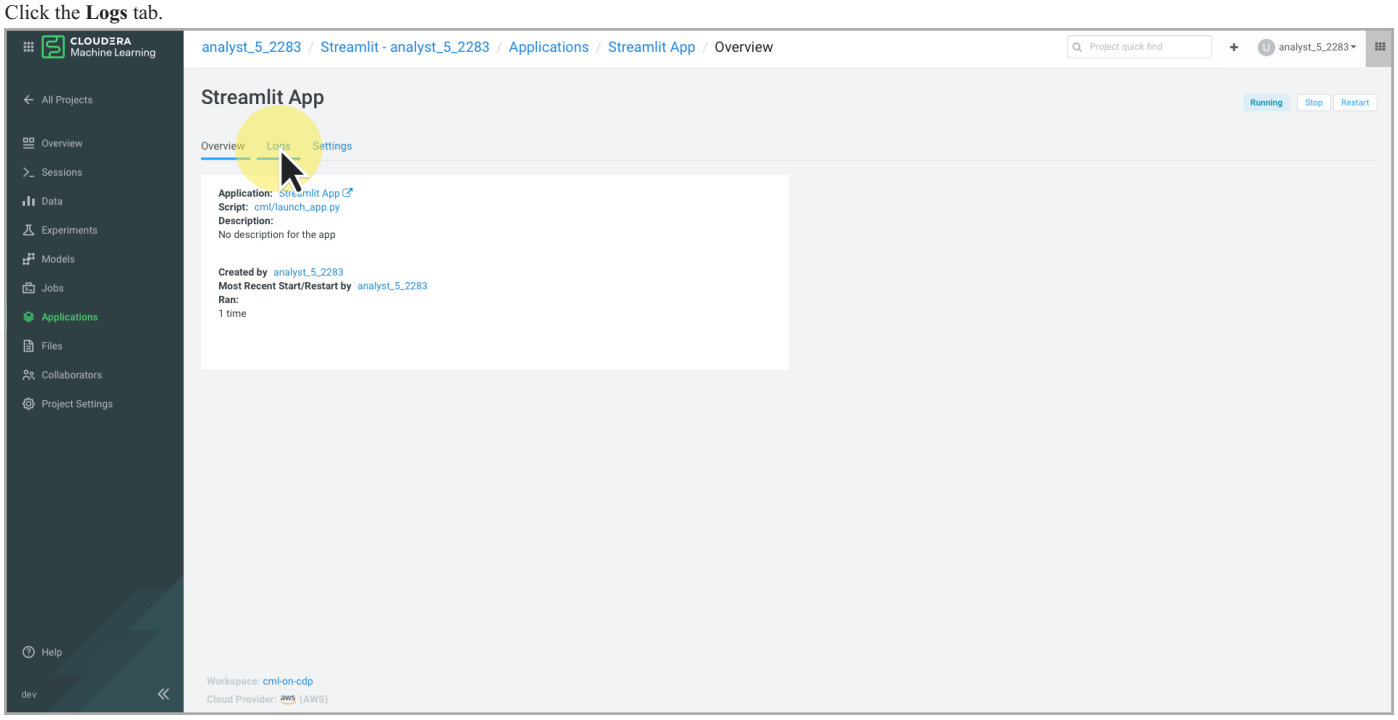


2. Click **View details** for Step 2 - Start Streamlit Application.

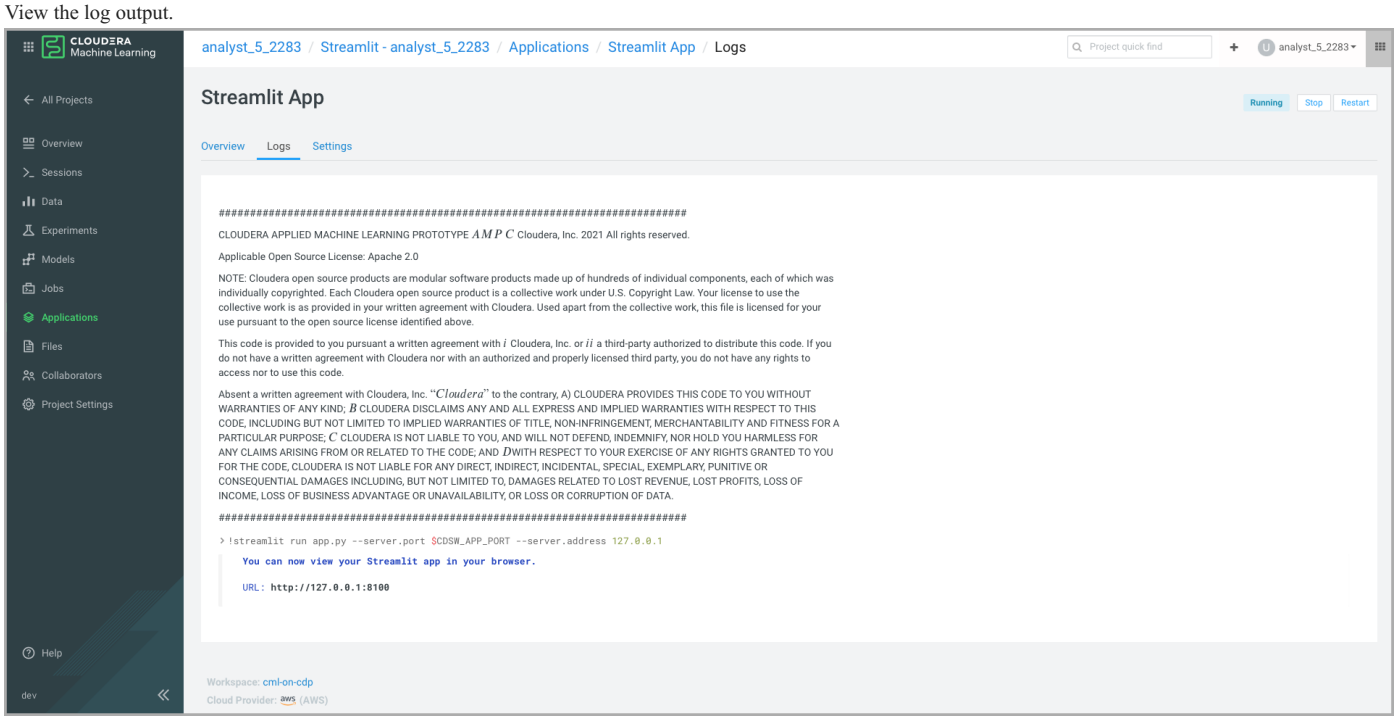


A new tab will open with an overview of the application deployment. The overview provides a link to the application, the script that created the application, and when the application was last started.

1. Click the **Logs** tab.



2. View the log output.



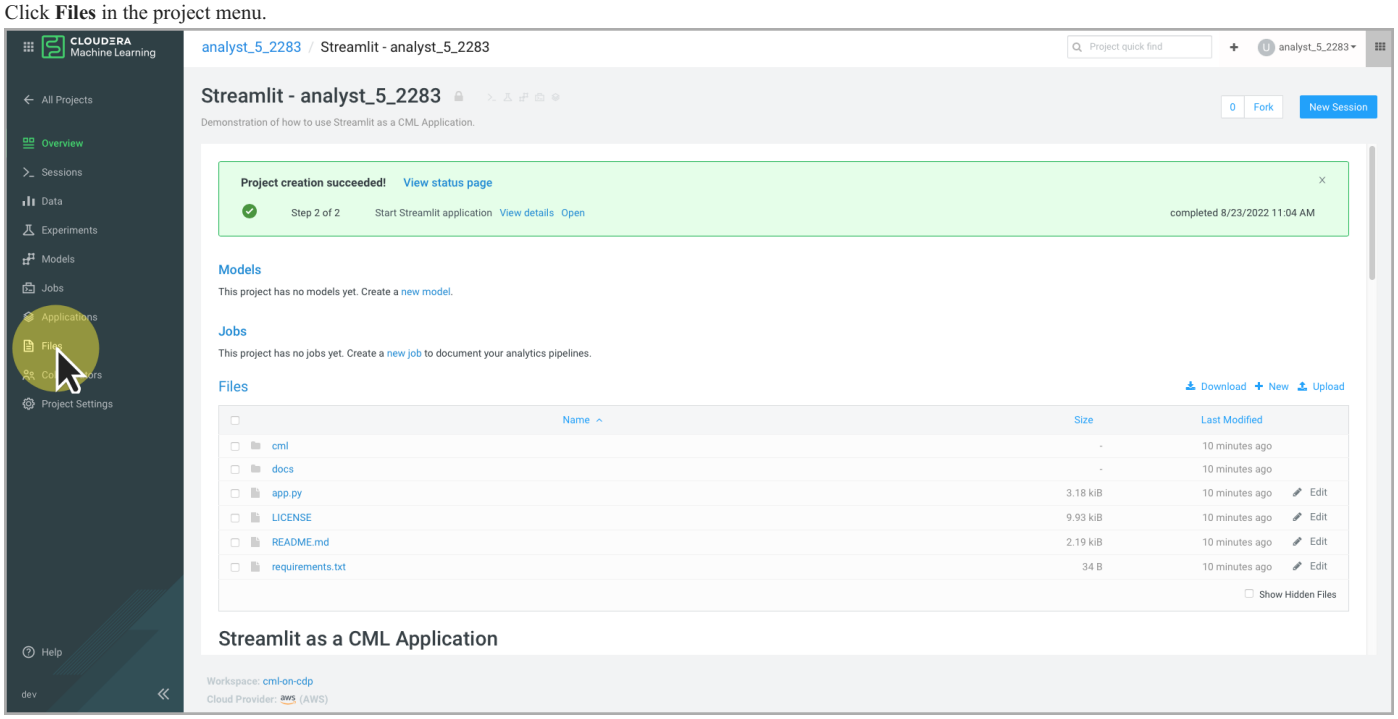
Logs shows the output from the application. In this case, you can see the command that was used to start the application:

```
!streamlit run app.py --server.port $CDSW_APP_PORT --server.address 127.0.0.1
```

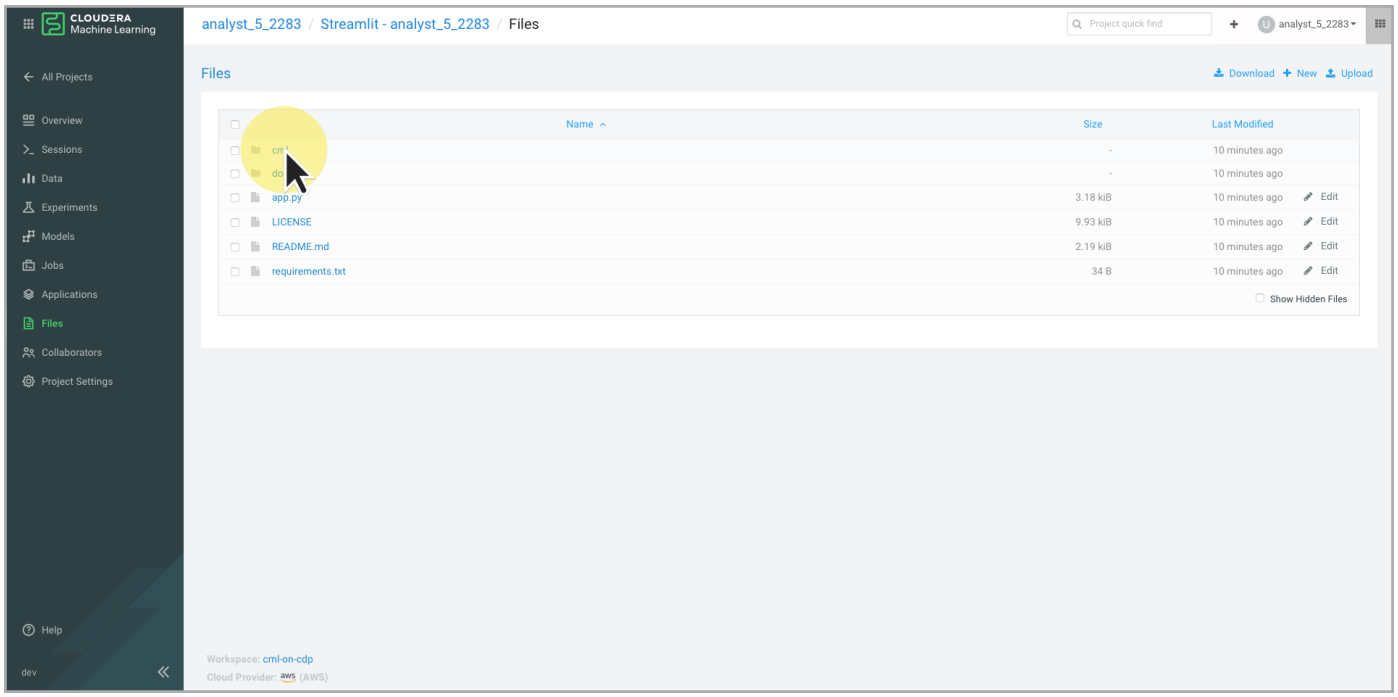
Note

CML applications are long-running web applications. Unlike sessions, applications do not timeout from inactivity. The application must use either `CDSW_APP_PORT` or `CDSW_READONLY_PORT` as its server port.

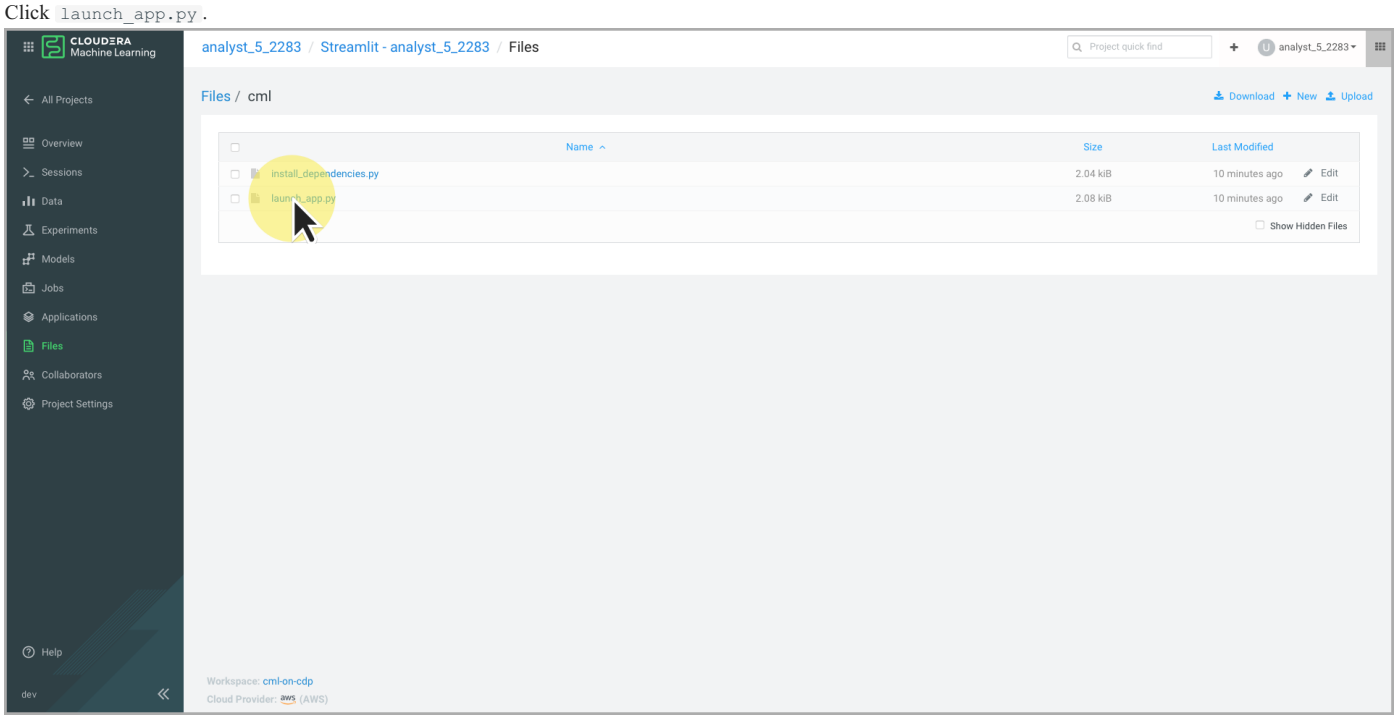
1. Click **Files** in the project menu.



2. Click the `CML` folder.

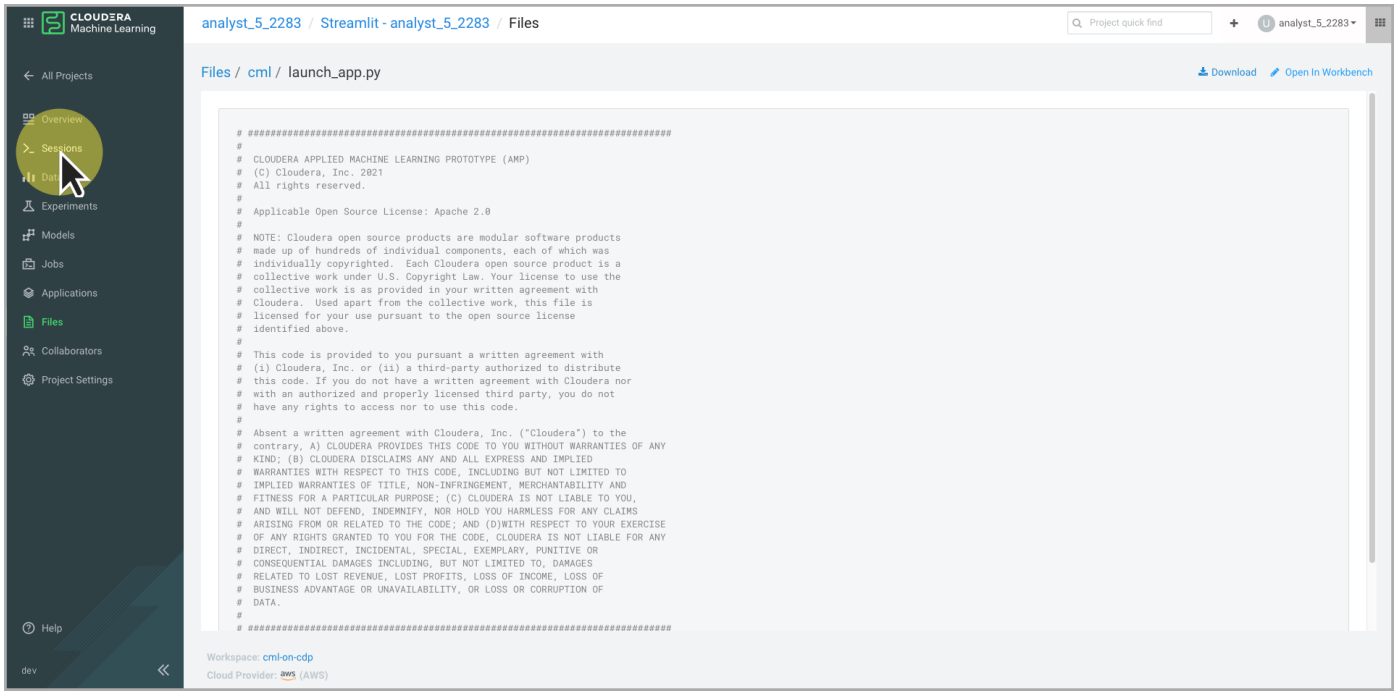


3. Click `launch_app.py`.



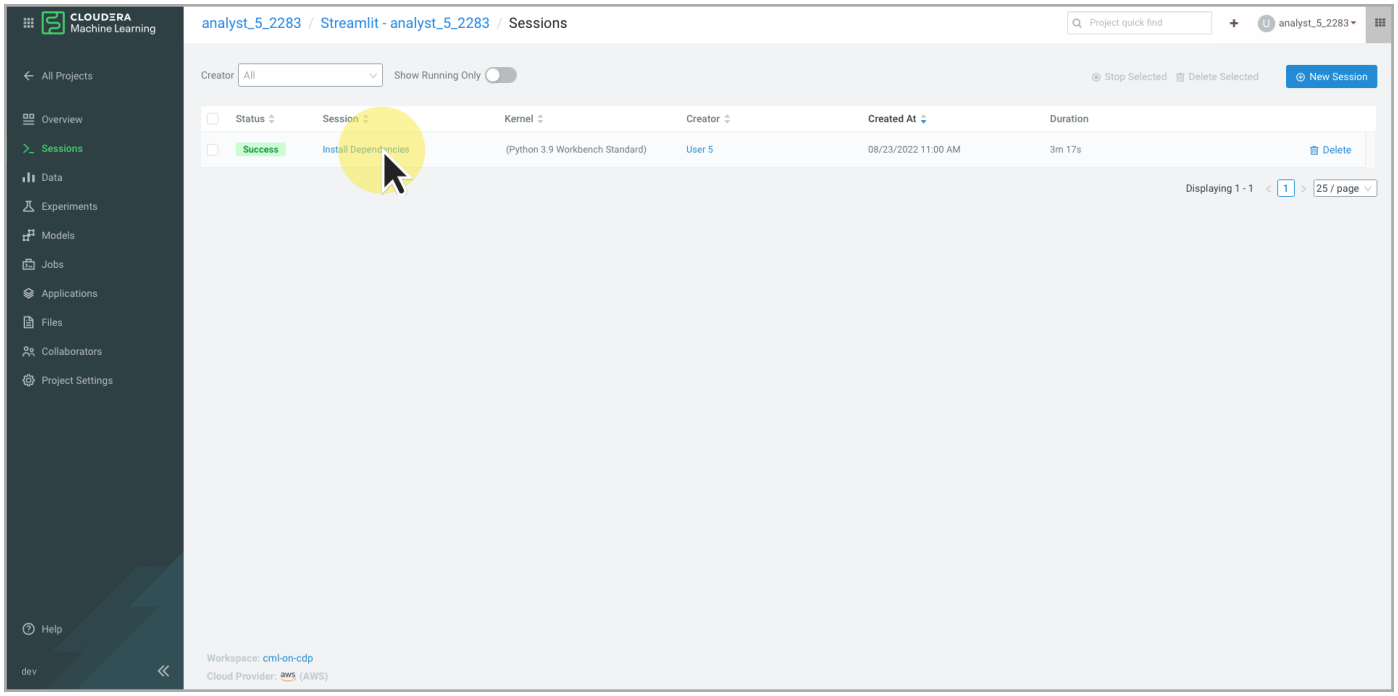
This is the file that was used to start the application. If you look back at the `project-metadata.yaml` file, you will see where it was specified in the `start_application` task.

1. Click **Sessions** in the project menu.



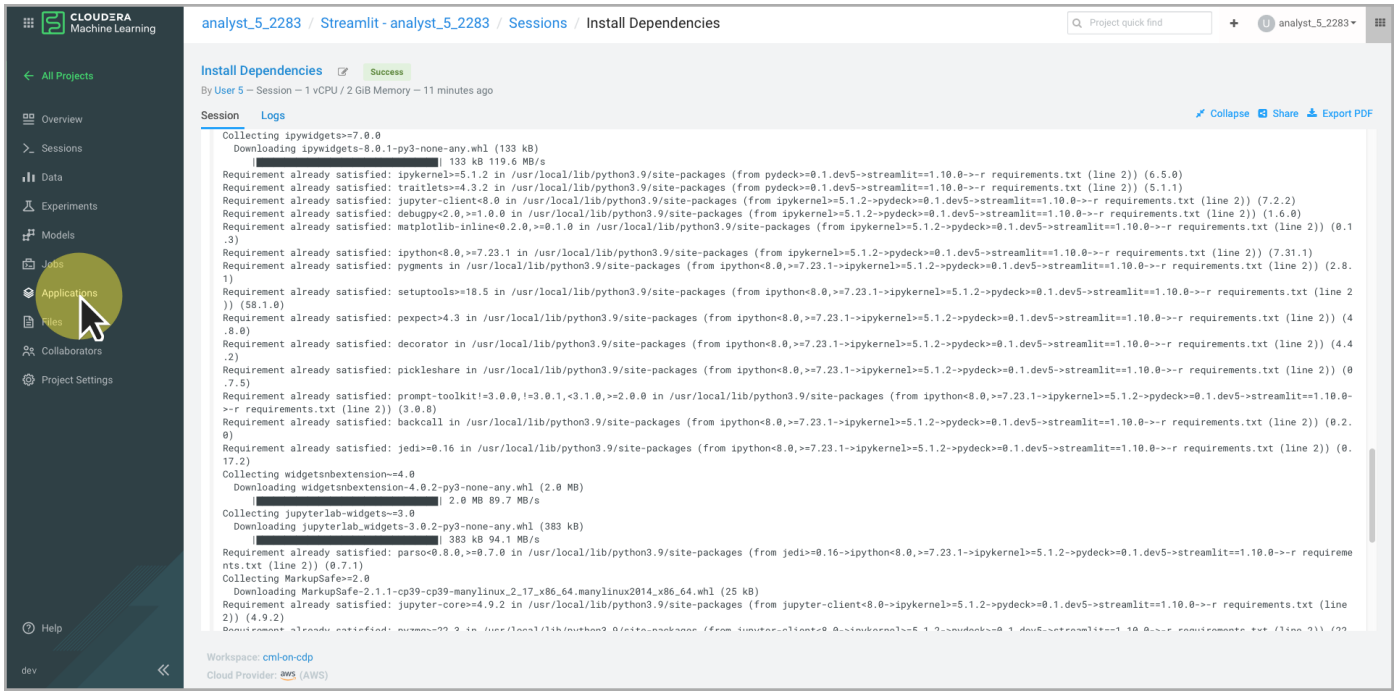
This is a list of all of the project sessions. Here, you can see the session used by the AMP to install the Python dependencies.

1. Click **Install Dependencies** in the list of sessions.

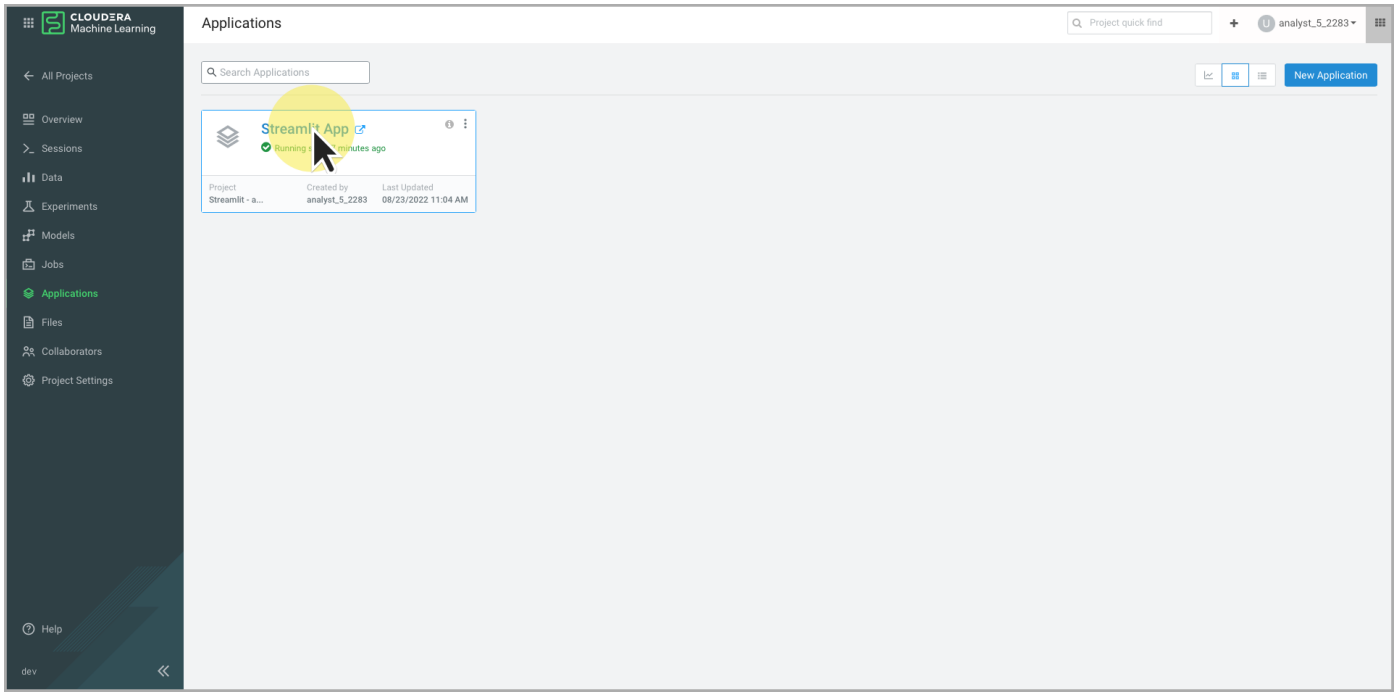


Again, you see the output of the session. Similarly, you can view the application created by the AMP.

1. Click **Applications** in the project menu.

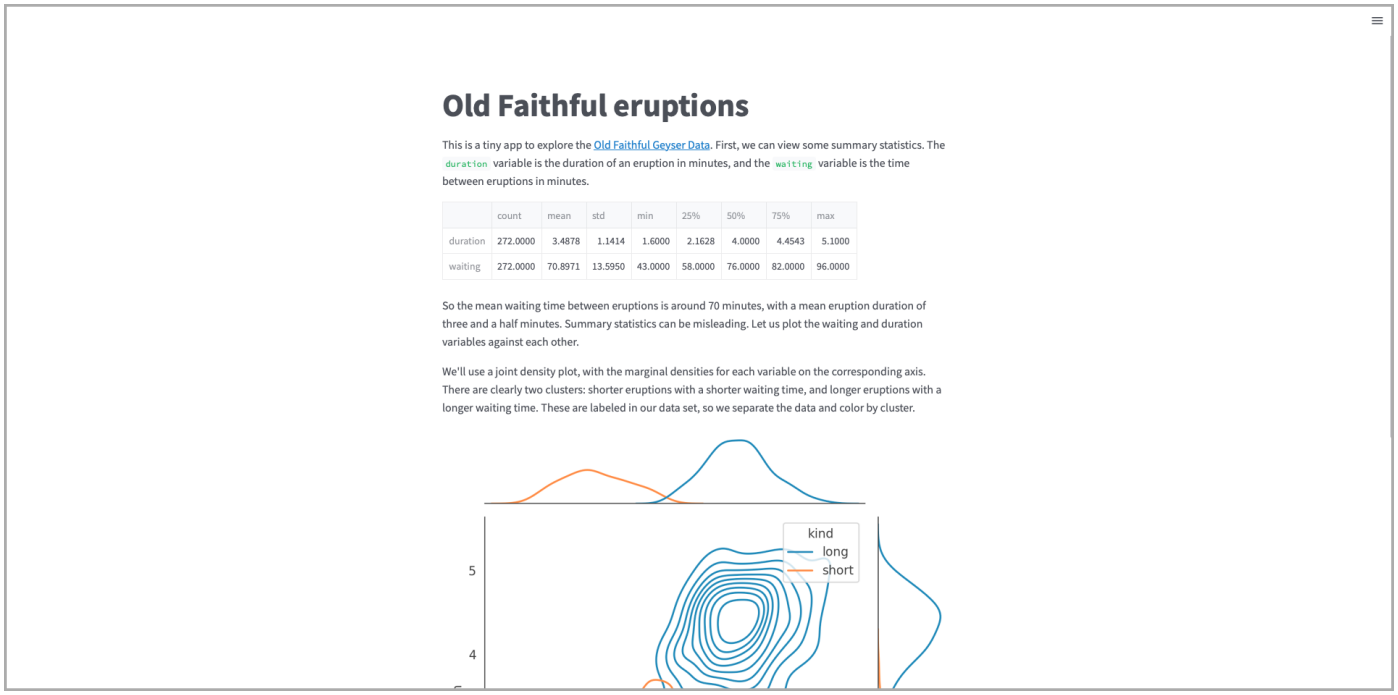


2. Click **Streamlit App**.



This opens a new tab and displays the application. The URL can be shared and viewed by others. By default, authentication for applications is enforced on all ports and users cannot create public applications. If desired, the Admin user can allow users to create public applications that can be accessed by unauthenticated users. Therefore, users will typically have to sign into the CDP environment before opening an application's URL.

1. View Streamlit App.

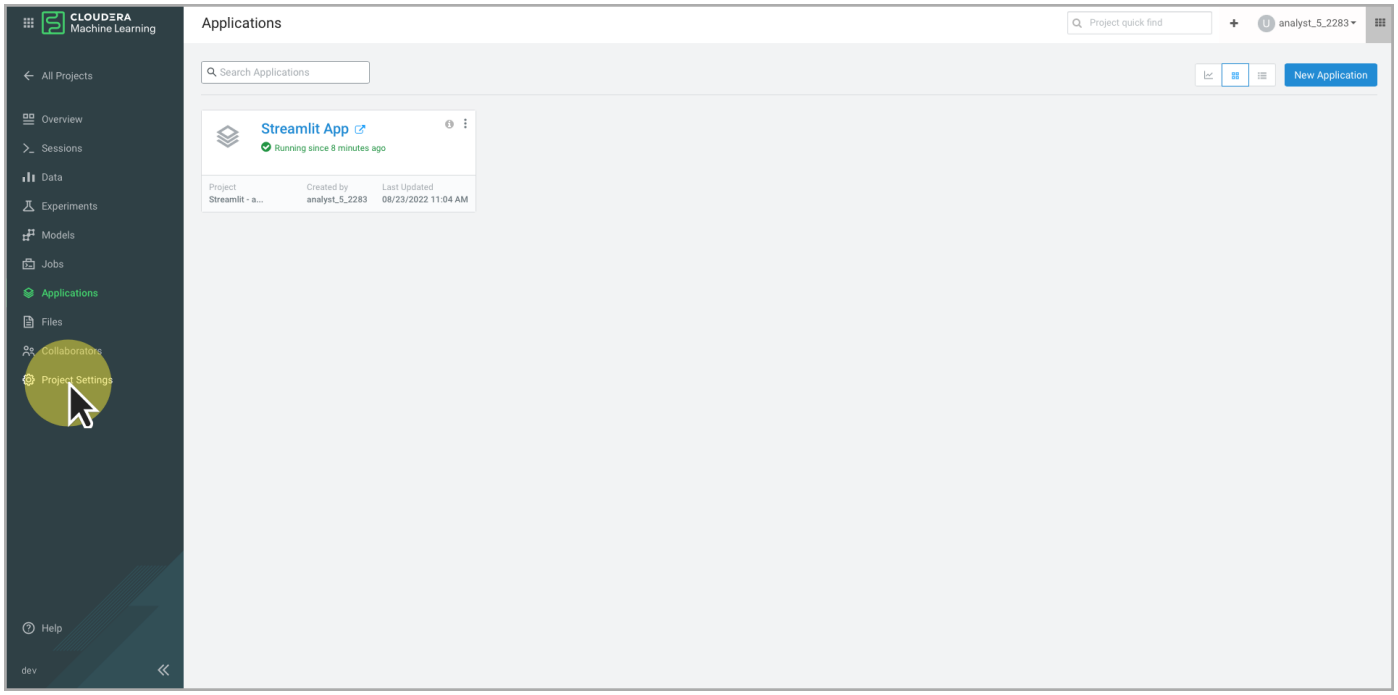


Modify the Application

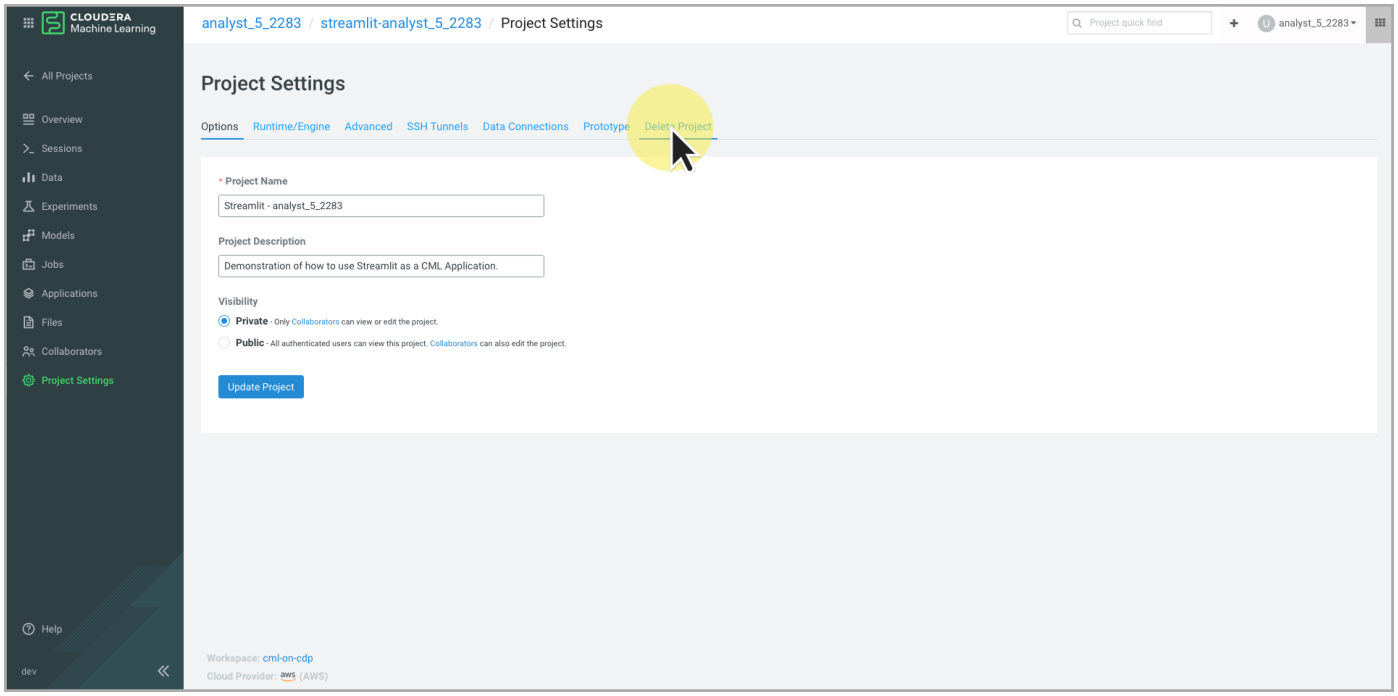
1. Return to the Streamlit project.
2. Click **Files** in the project menu.
3. Compare the `app.py` code to the output of the app itself.
4. Click the **Open in Workbench** button.
5. Change some of the markdown content and reload the app.
6. Did you need to rebuild the app?
7. Why is the first markdown block statement in a `st.markdown` call and not the second one?
8. Study the `jointplot` seaborn function: https://seaborn.pydata.org/generated/seaborn.jointplot.html
9. Change the type of the `jointplot`.

Delete the Project

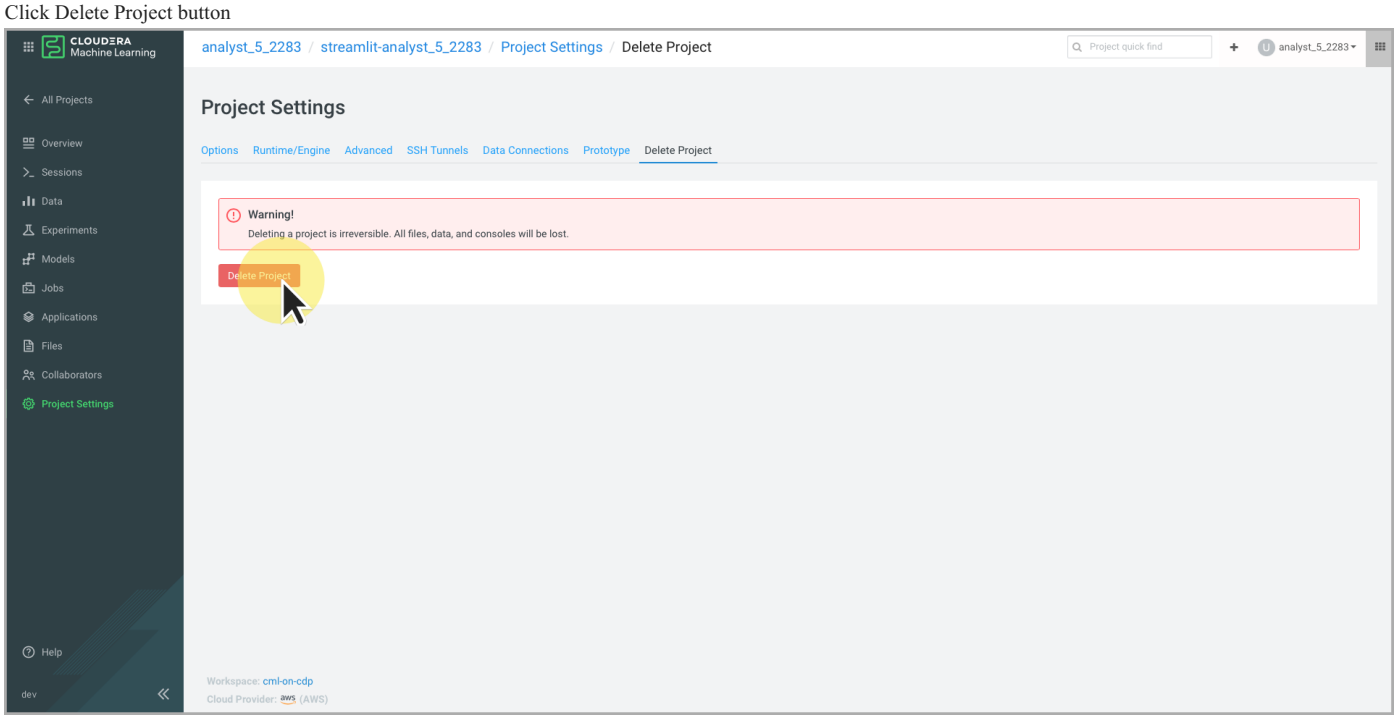
1. Click Project Settings



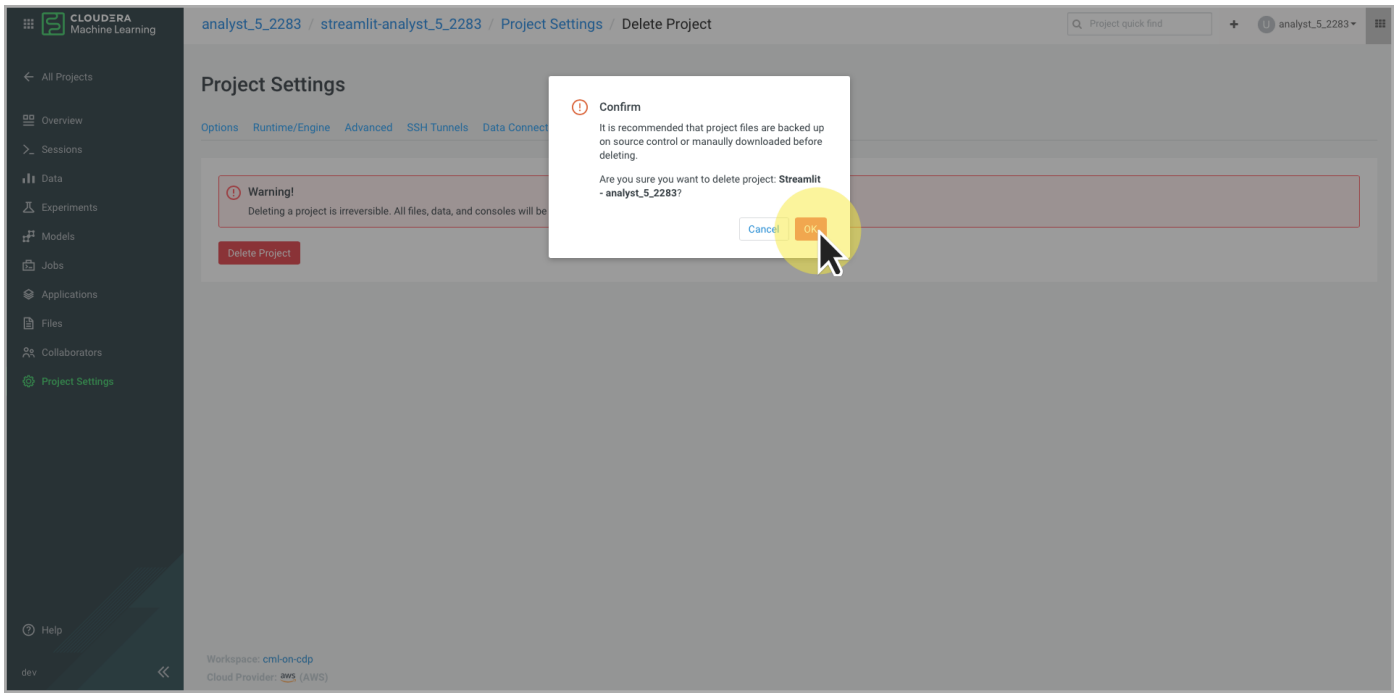
2. Click Delete Project tab



3. Click Delete Project button



4. Click OK to confirm



Bonus

If you finish early, explore other AMPs in the AMP catalog.

End of Exercise

Solution

Data - Access, Audit, and Mask

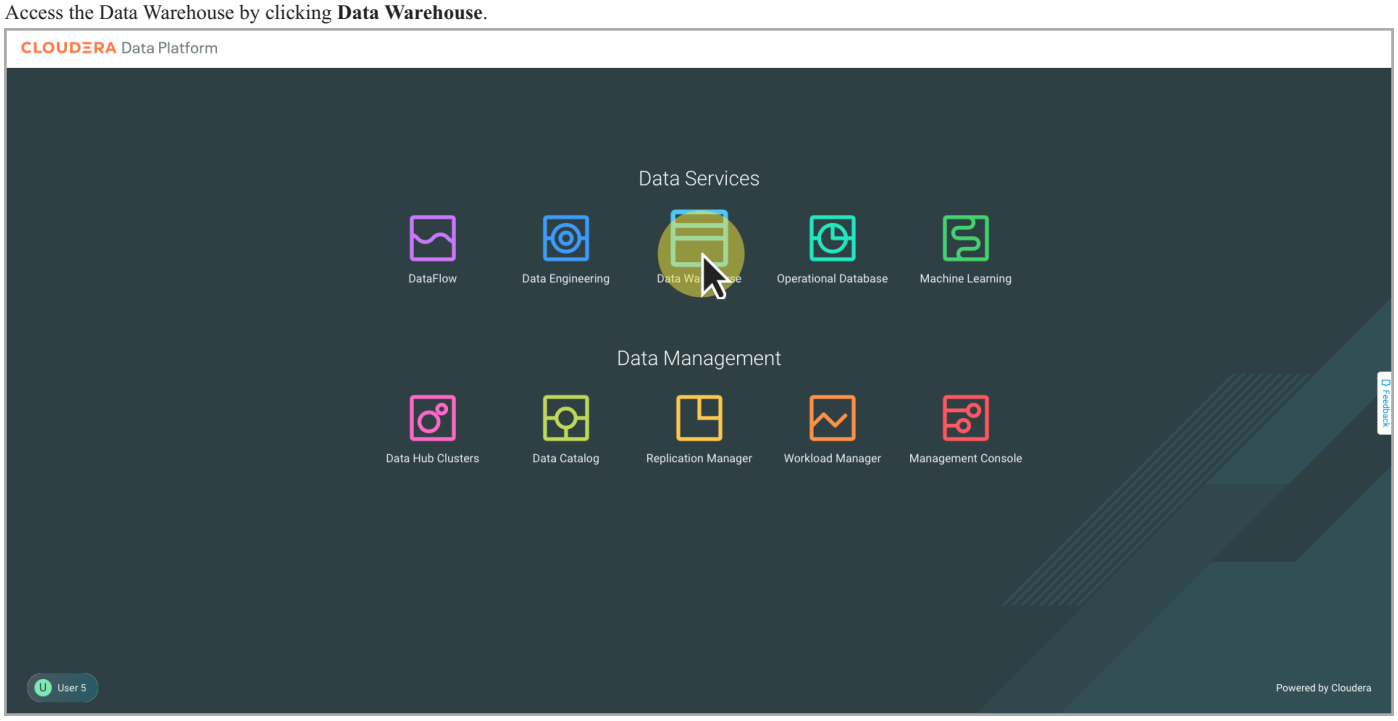
This exercise uses the fictitious Duocar dataset. The data is stored in Amazon S3 and has been added to the company's data warehouse. Some of the data contains Personal Identifiable Information (PII). The company has requested the administrator only allow access to the information to those who need access to it to perform their job. In this instance, the full birth date is considered to be PII. Therefore, the birth date fields have been classified as PII and a PII policy has been created to mask the month and day.

Note

This exercise uses birth dates as PII. While a birth date may be considered PII in some scenarios, this exercise is completely hypothetical and has been designed to demonstrate the software and concepts. What data is PII and how to protect it is a legal issue and is beyond the scope of this course.

Access Data

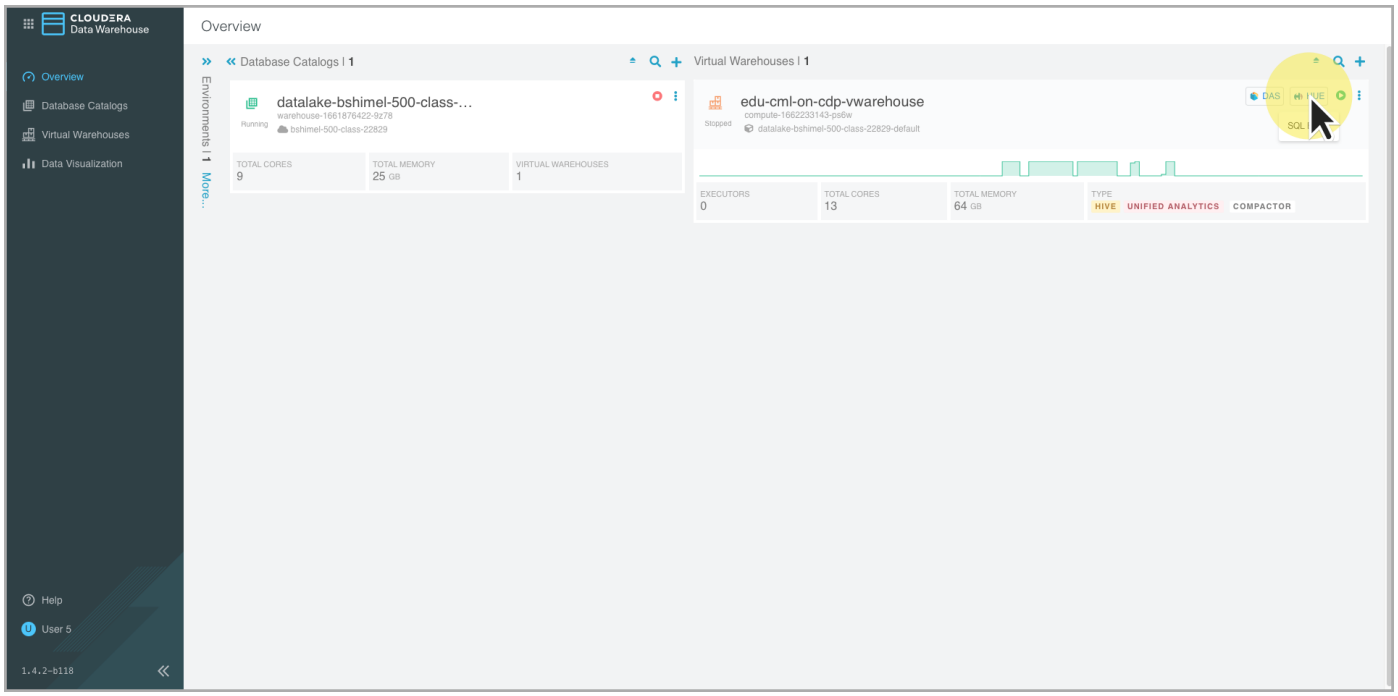
1. Access the Data Warehouse by clicking **Data Warehouse**.



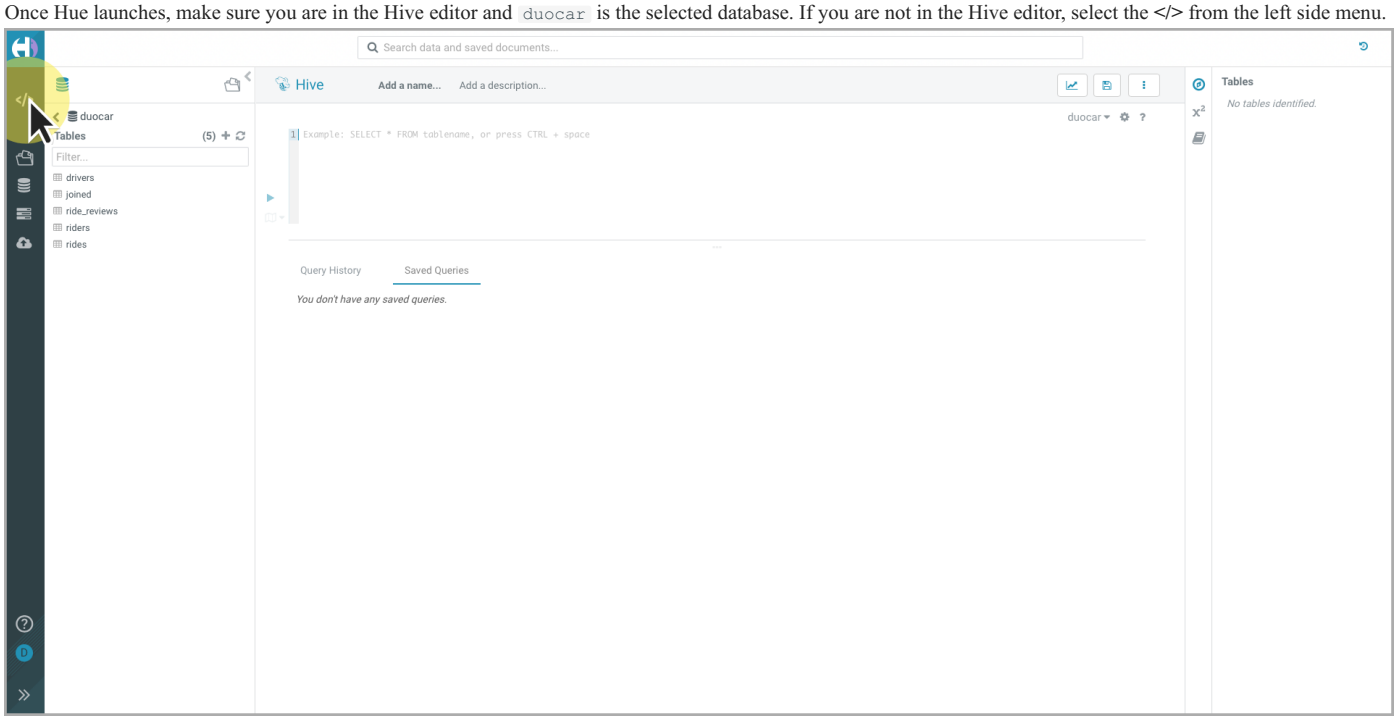
Warning

There is a known issue opening Hue with Safari on MacOS that results in an "Invalid CORS request." Please use another browser, like Chrome or Firefox, if you experience this issue.

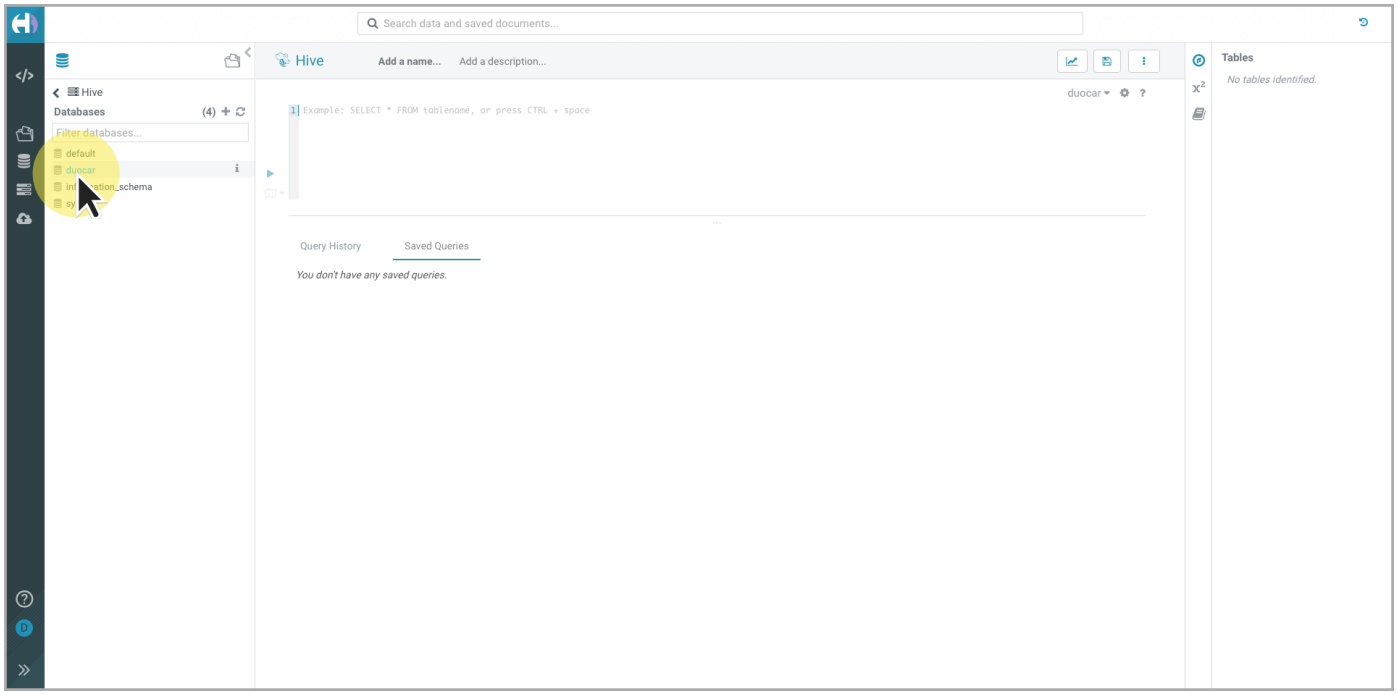
1. Click **Hue** to launch Hue in a new browser tab.



2. Once Hue launches, make sure you are in the Hive editor and `duocar` is the selected database. If you are not in the Hive editor, select the </> from the left side menu.

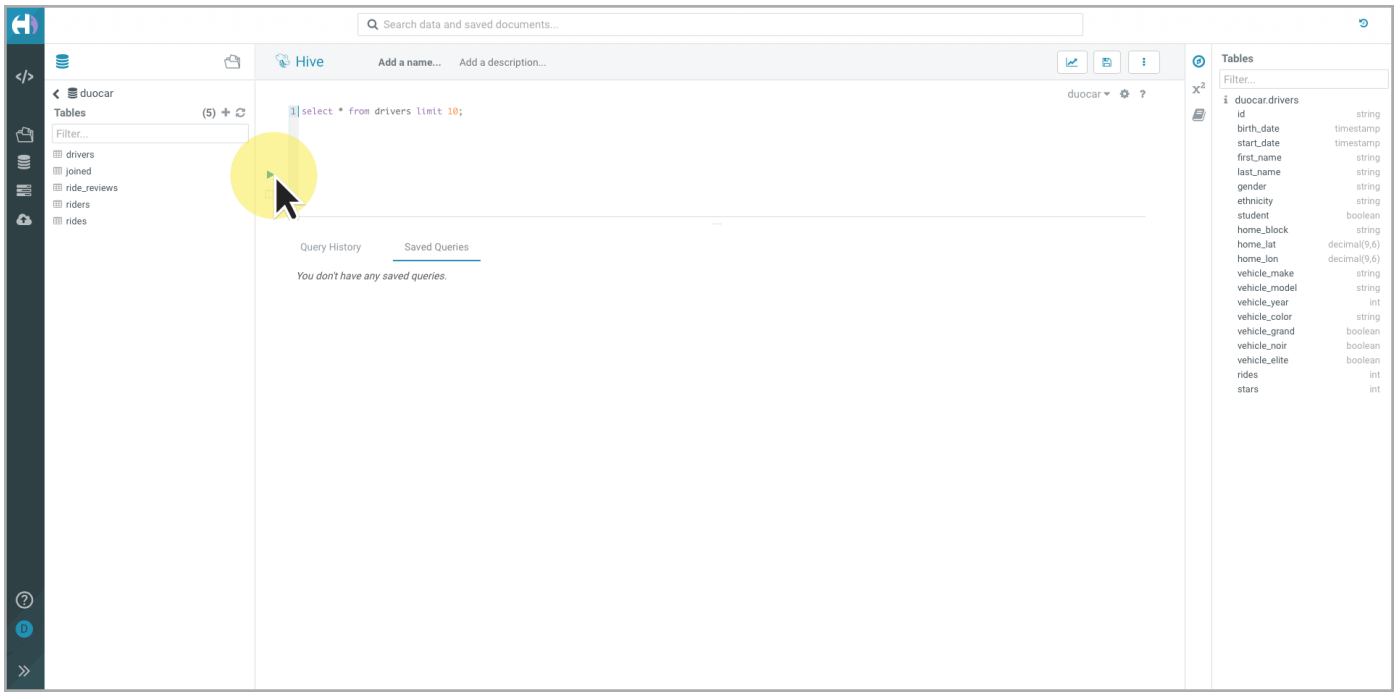


3. If `duocar` is not selected, select `duocar` from the list of databases. If `default` selected and Tables is displayed, click the < to navigate back to the list of databases and then select `duocar`.

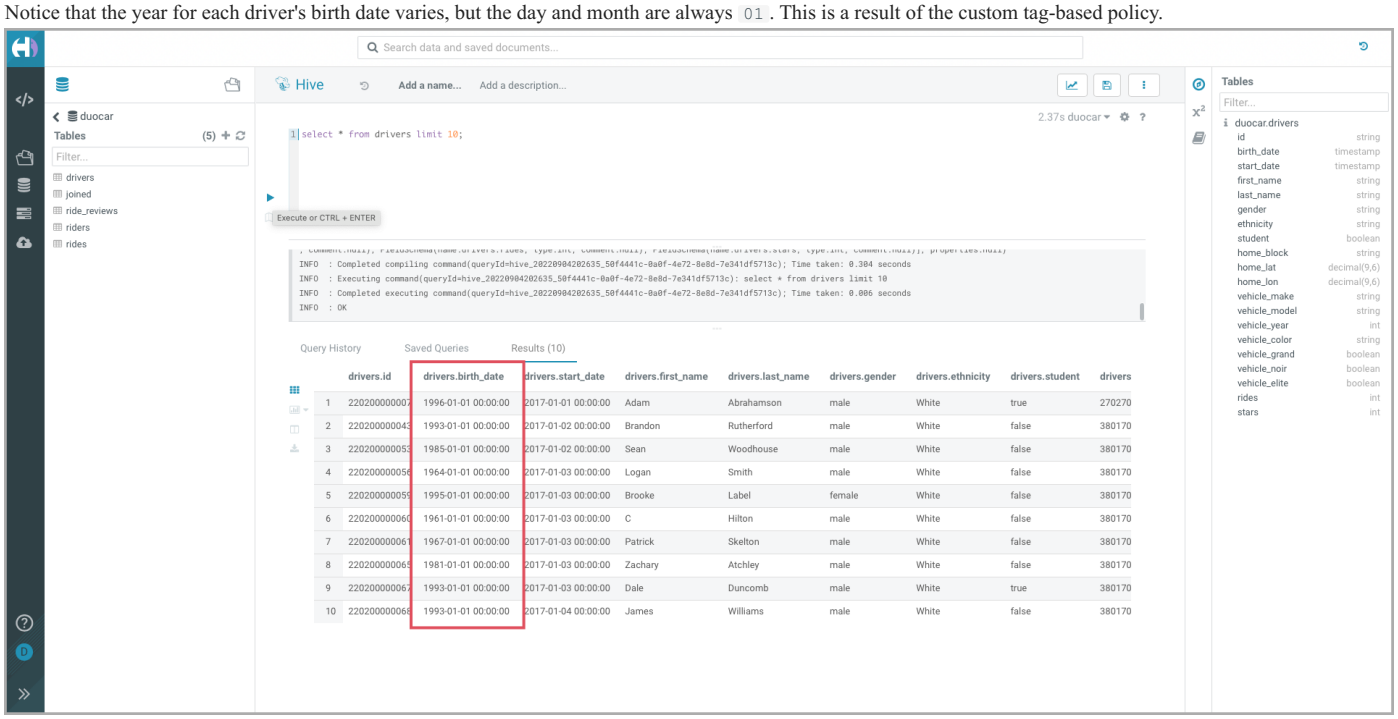


4. Next, execute the following SQL by entering it into the editor and clicking the play/run button to show the drivers' data:

```
select * from drivers limit 10;
```



5. Notice that the year for each driver's birth date varies, but the day and month are always `01`. This is a result of the custom tag-based policy.

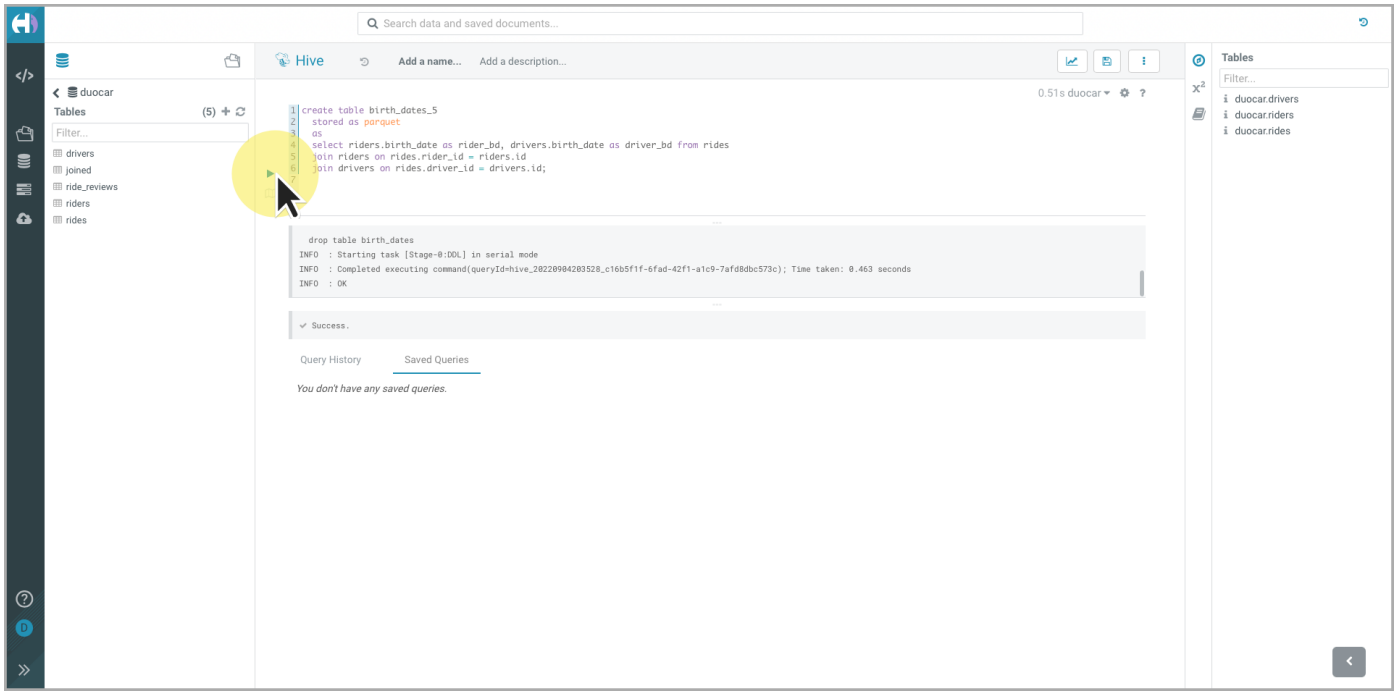


Create a new joined table of birth dates.

Now that you have seen the effects of masking on the original tables, what happens when you create a new table from tables that have been masked? In the following steps, you will create a new table from the `drivers` and `riders` tables. You will view the new table and its data. You will also view the table in the data catalog.

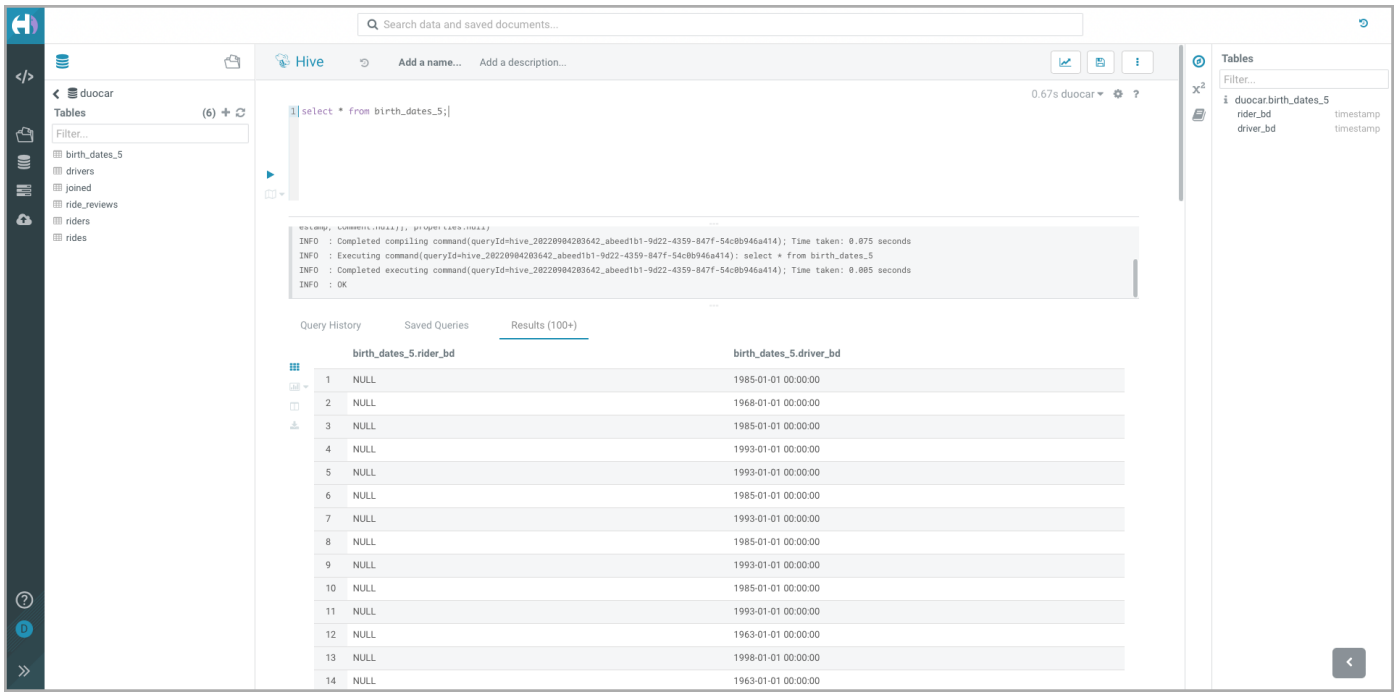
1. Use the following SQL to create a new table that contains the driver's and rider's birth date for each ride. **Replace the XX in the table name with your student number.**

```
create table birth_dates_XX
    as
    select riders.birth_date as rider_bd, drivers.birth_date as driver_bd from rides
    join riders on rides.rider_id = riders.id
    join drivers on rides.driver_id = drivers.id;
```



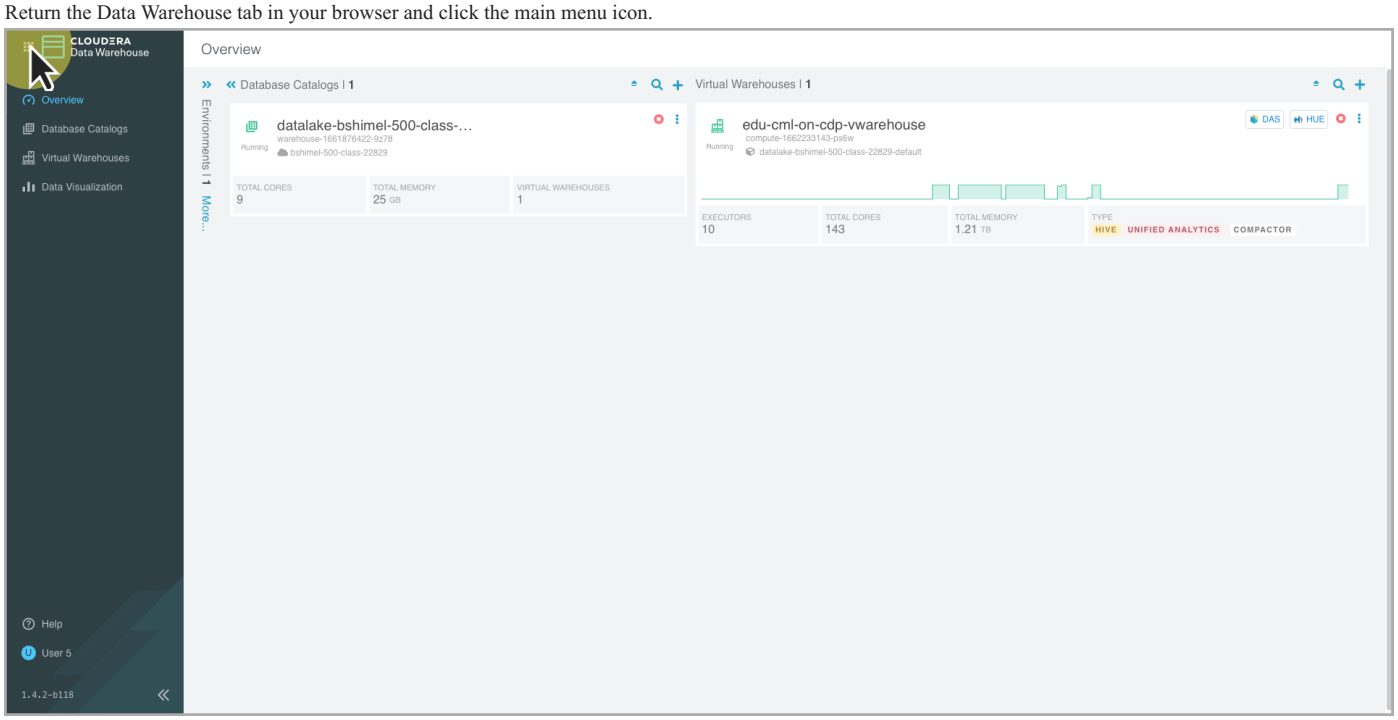
2. You may have to wait for Tez session. If so, it usually takes a few minutes for a session to start.

3. Once the table is created, use a SELECT statement to view the new table's contents.

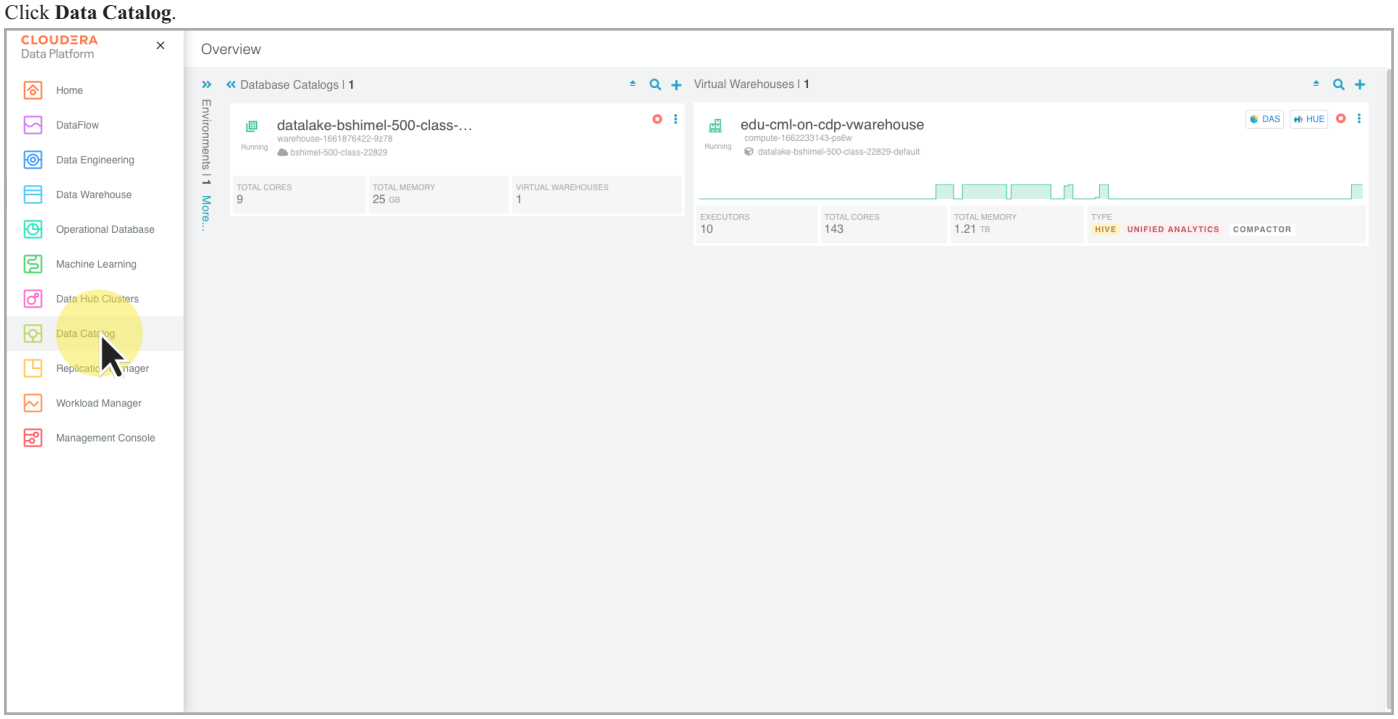


	birth_dates_5.rider_bd	birth_dates_5.driver_bd
1	NULL	1985-01-01 00:00:00
2	NULL	1968-01-01 00:00:00
3	NULL	1985-01-01 00:00:00
4	NULL	1993-01-01 00:00:00
5	NULL	1993-01-01 00:00:00
6	NULL	1985-01-01 00:00:00
7	NULL	1993-01-01 00:00:00
8	NULL	1985-01-01 00:00:00
9	NULL	1993-01-01 00:00:00
10	NULL	1985-01-01 00:00:00
11	NULL	1993-01-01 00:00:00
12	NULL	1963-01-01 00:00:00
13	NULL	1998-01-01 00:00:00
14	NULL	1963-01-01 00:00:00

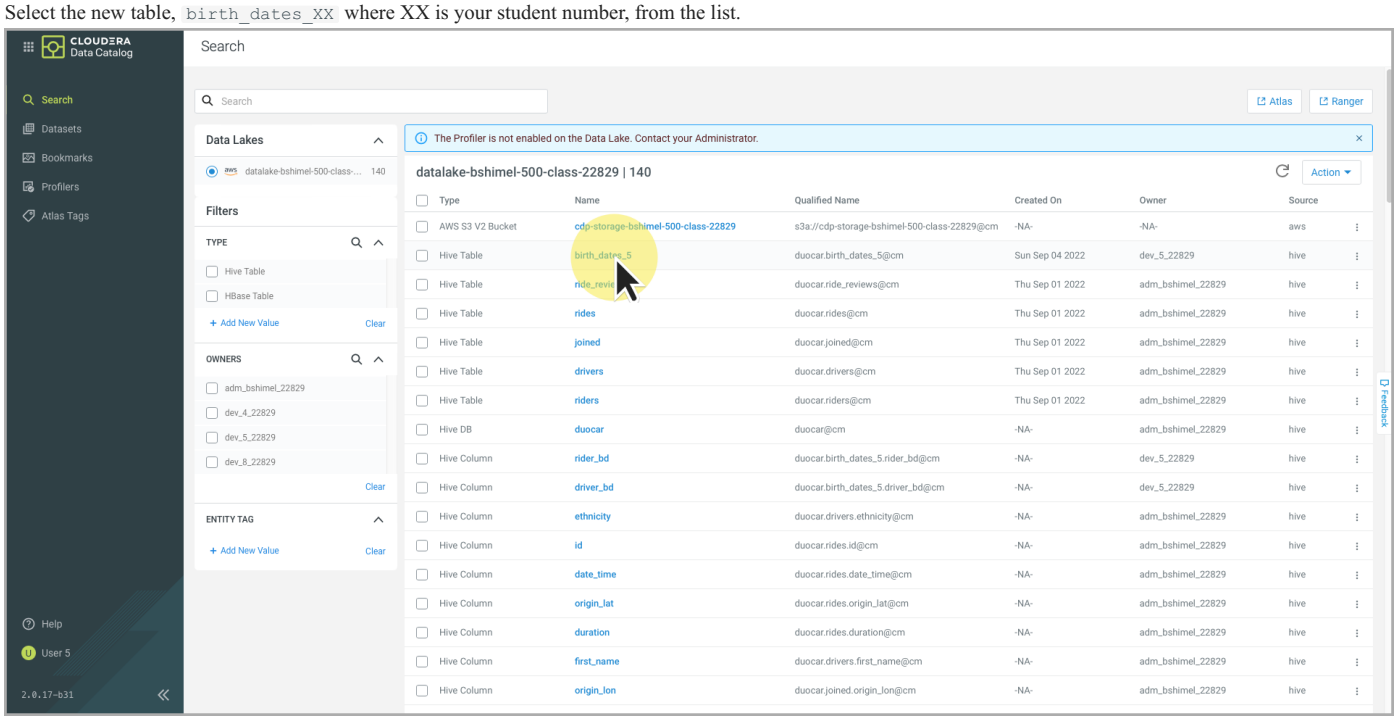
4. Return the Data Warehouse tab in your browser and click the main menu icon.



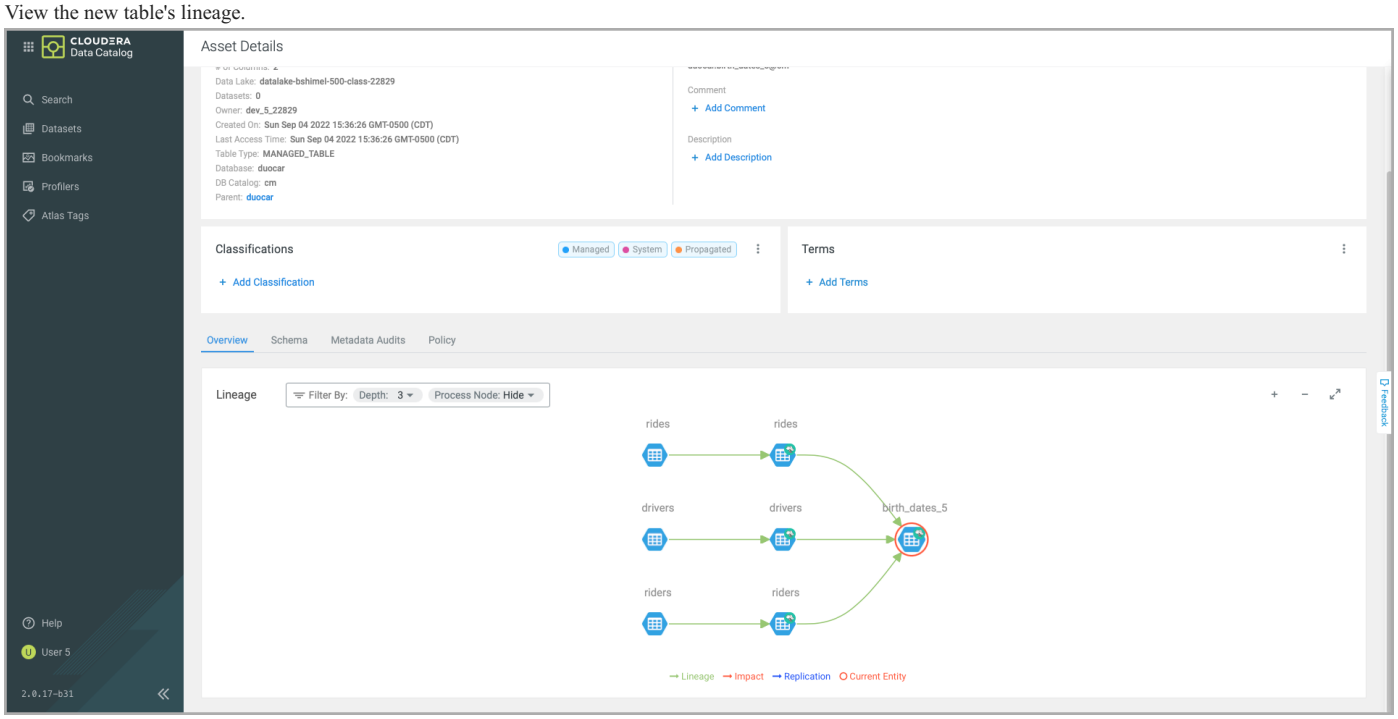
5. Click **Data Catalog**.



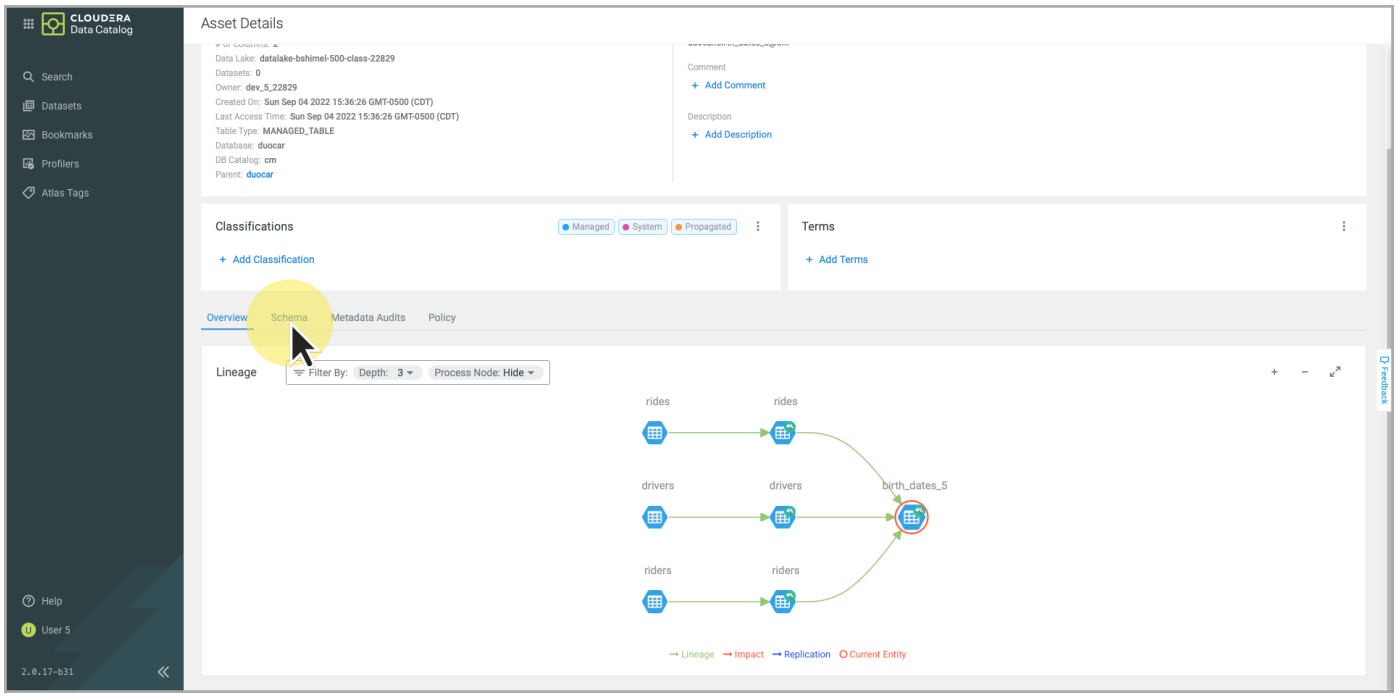
6. Select the new table, `birth_dates_XX` where XX is your student number, from the list.



7. View the new table's lineage.



8. Select the **Schema** tab. What are the classifications for the new fields?



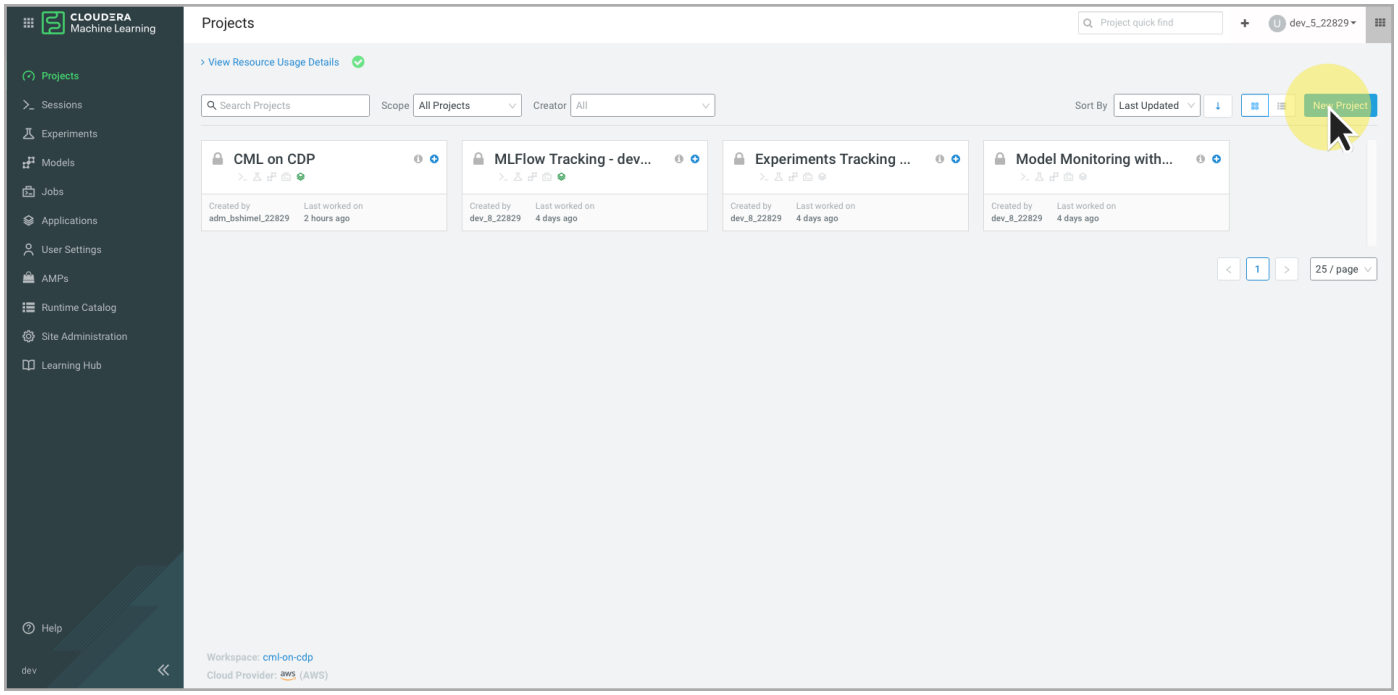
End of Exercise

Visualize Duocar Data

The Data Visualization application is good for exploring your data and sharing with others. In this exercise, you will:

- use the Data Visualization application to make a connection to the data warehouse,
- and create a dashboard to explore the ride data.

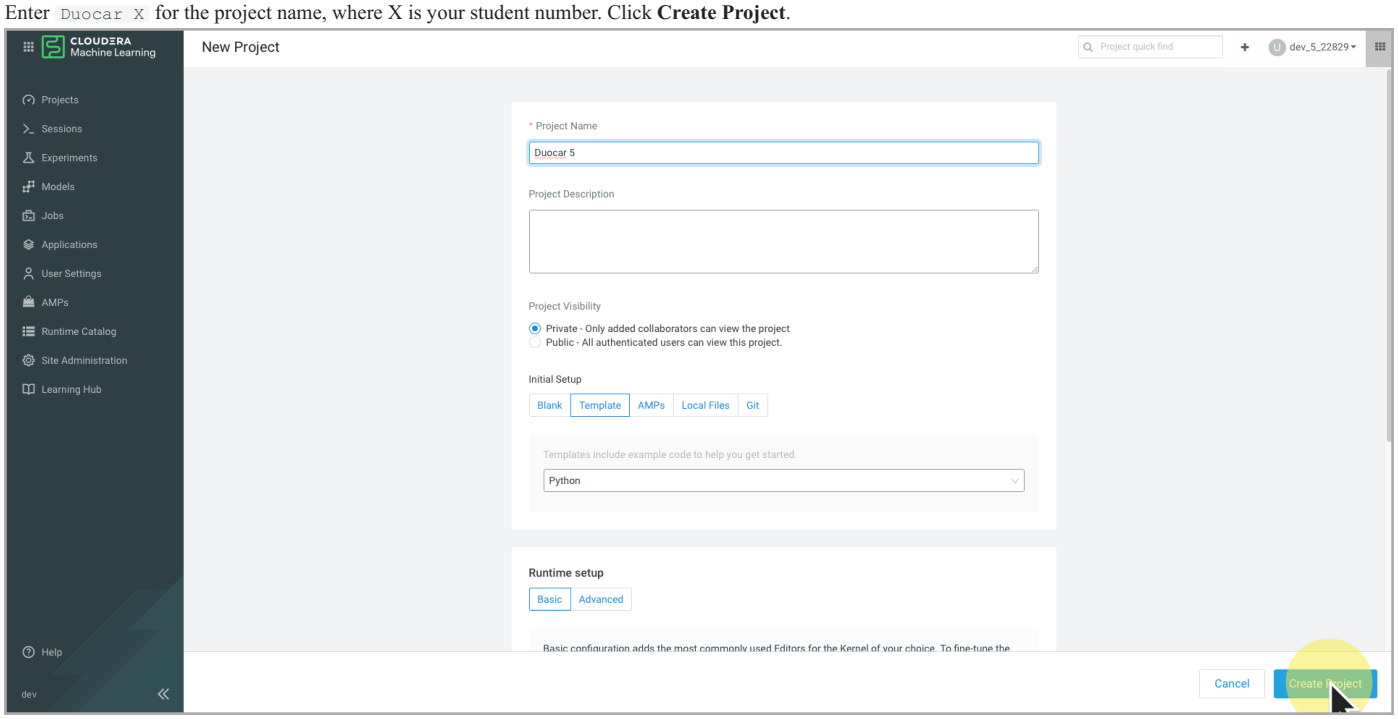
1. Open your CML workspace and click **New Project**.



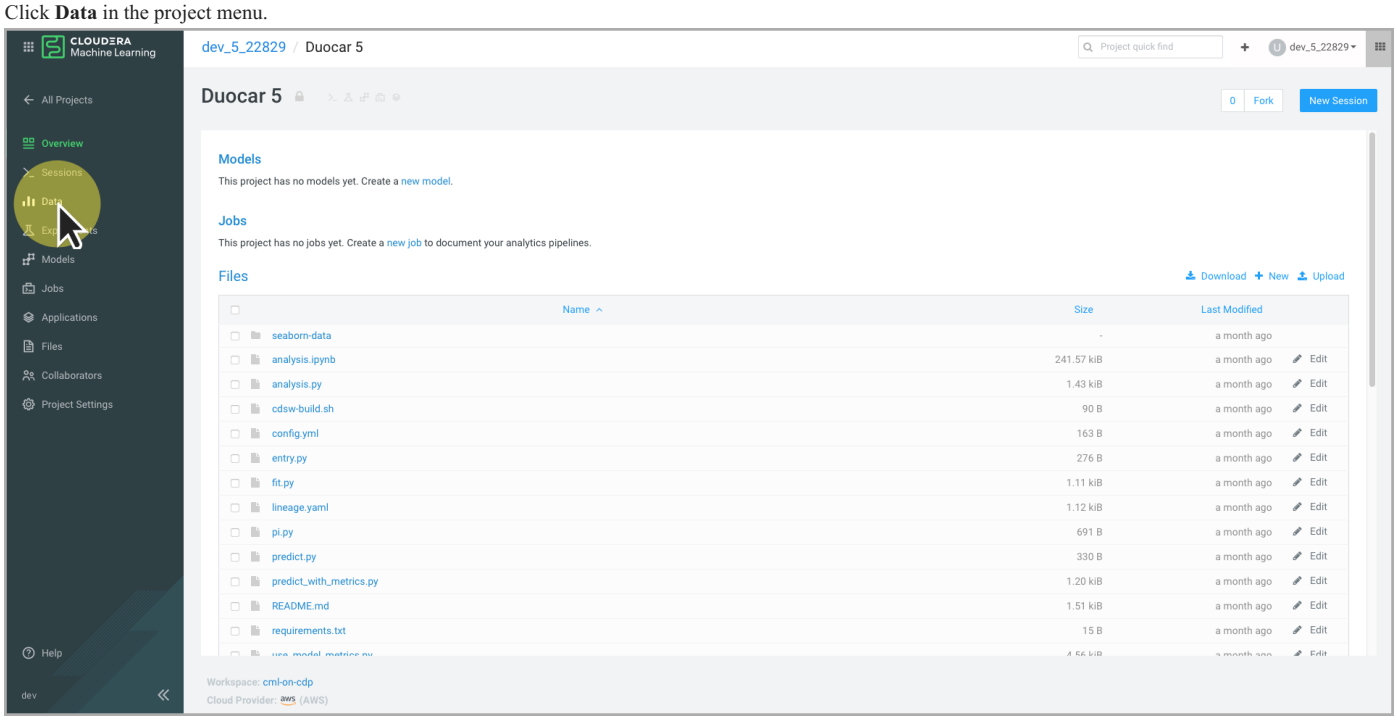
Note

It is not necessary to create a new project to use Data Visualization. In this case, each student is creating their own project in order to demonstrate some features which are project-wide.

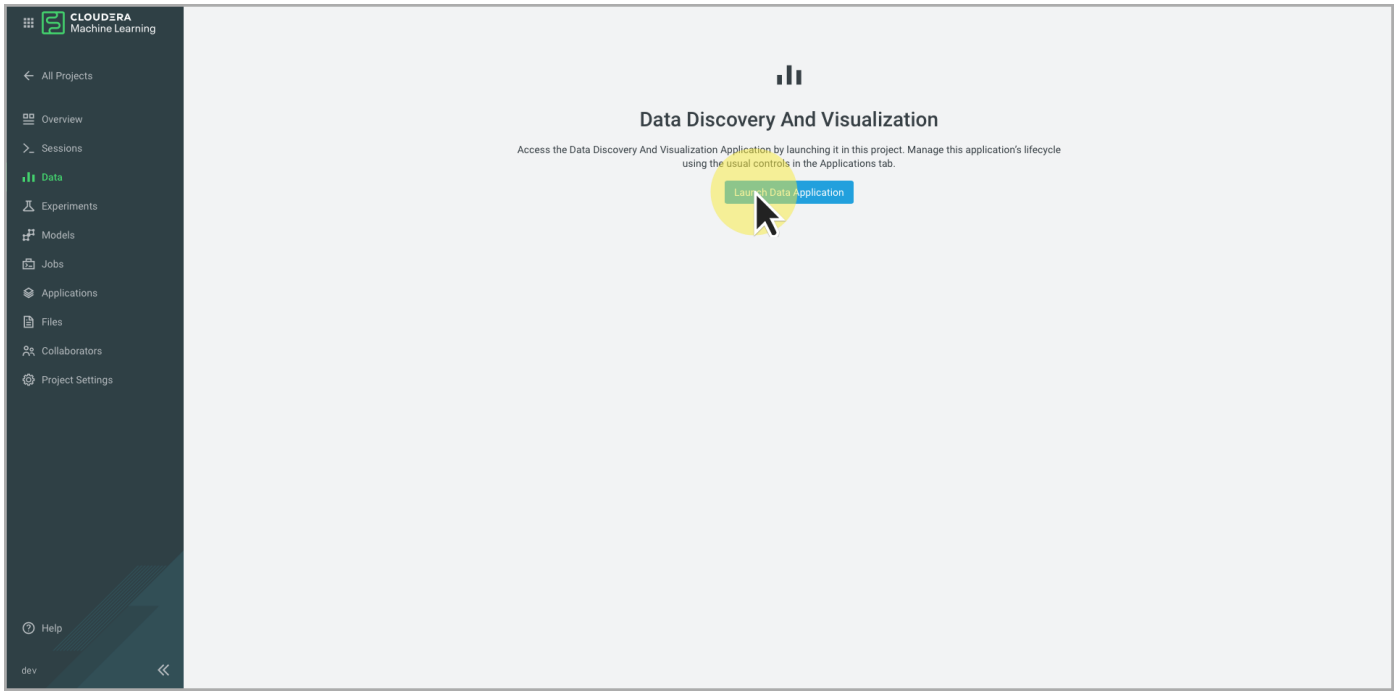
1. Enter `Duocar X` for the project name, where X is your student number. Click **Create Project**.



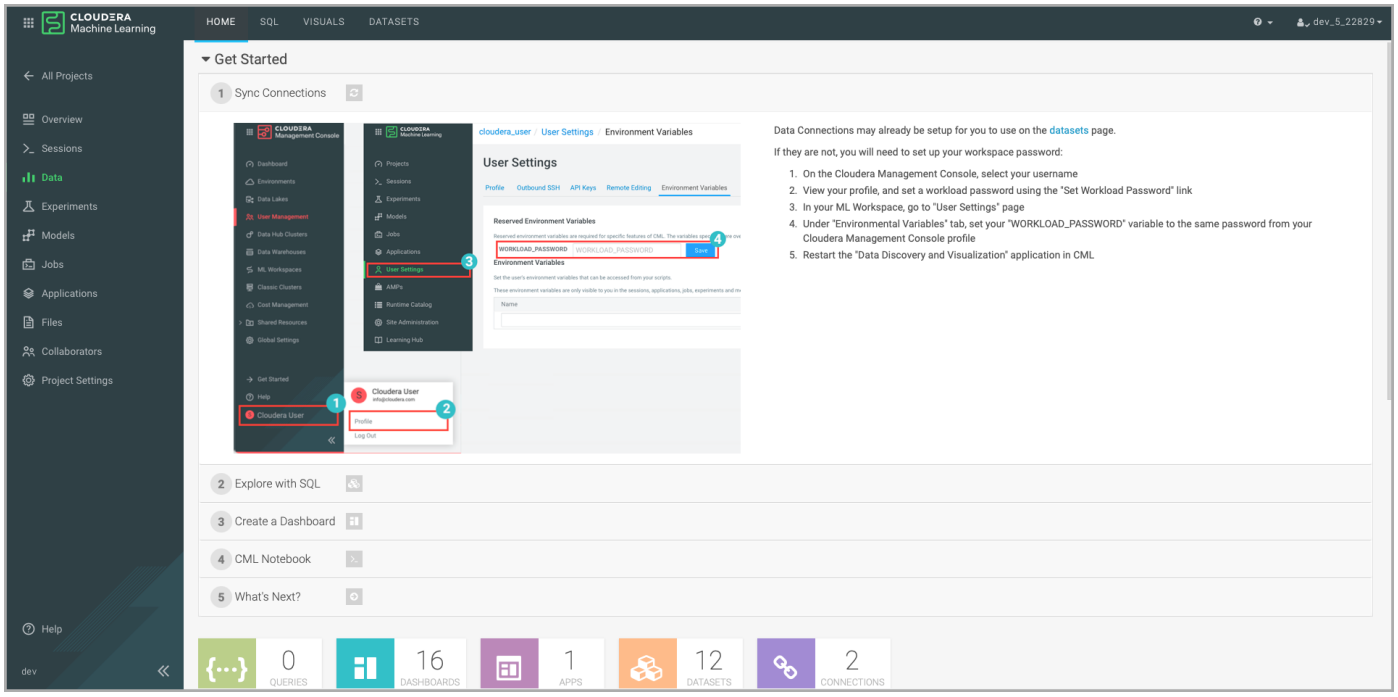
2. Click **Data** in the project menu.



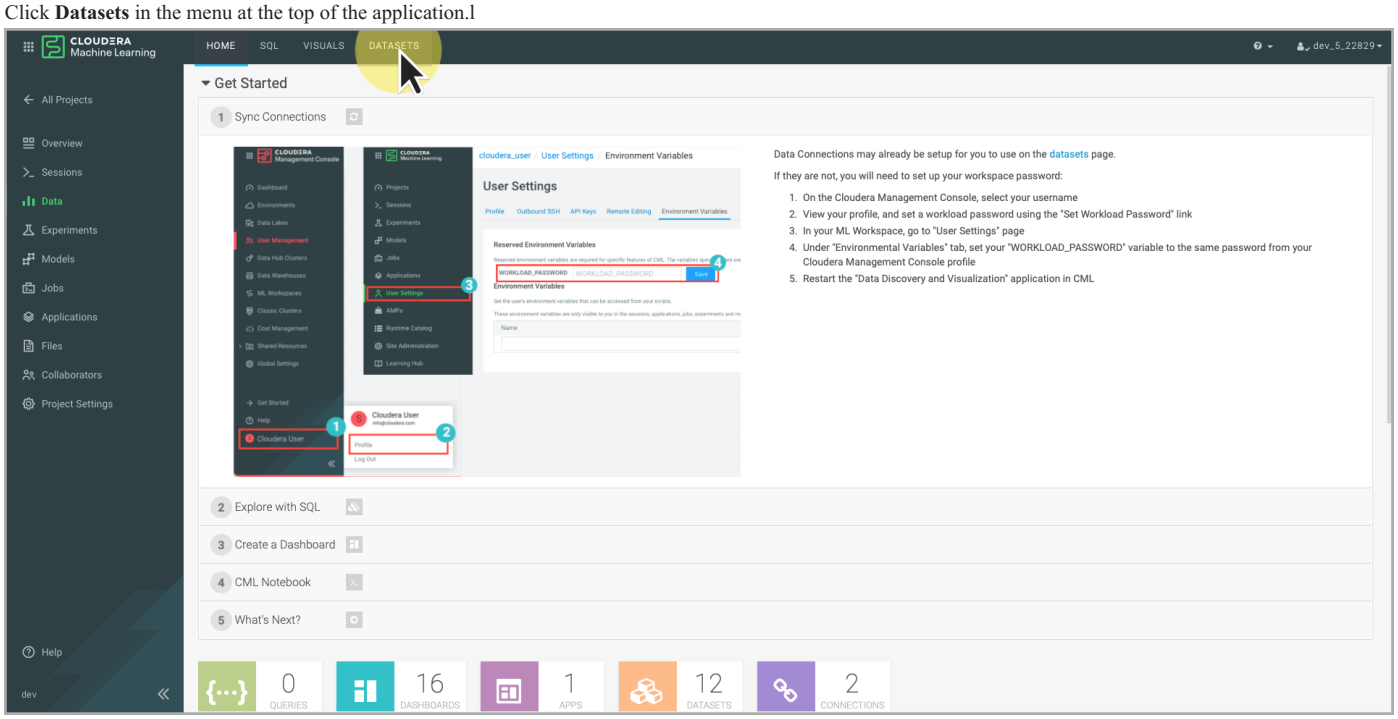
3. Data Discovery and Visualization is just an application. The Data link in the project menu is just a convenient shortcut to launch and access the application. Click **Launch Data Application** to start the application.



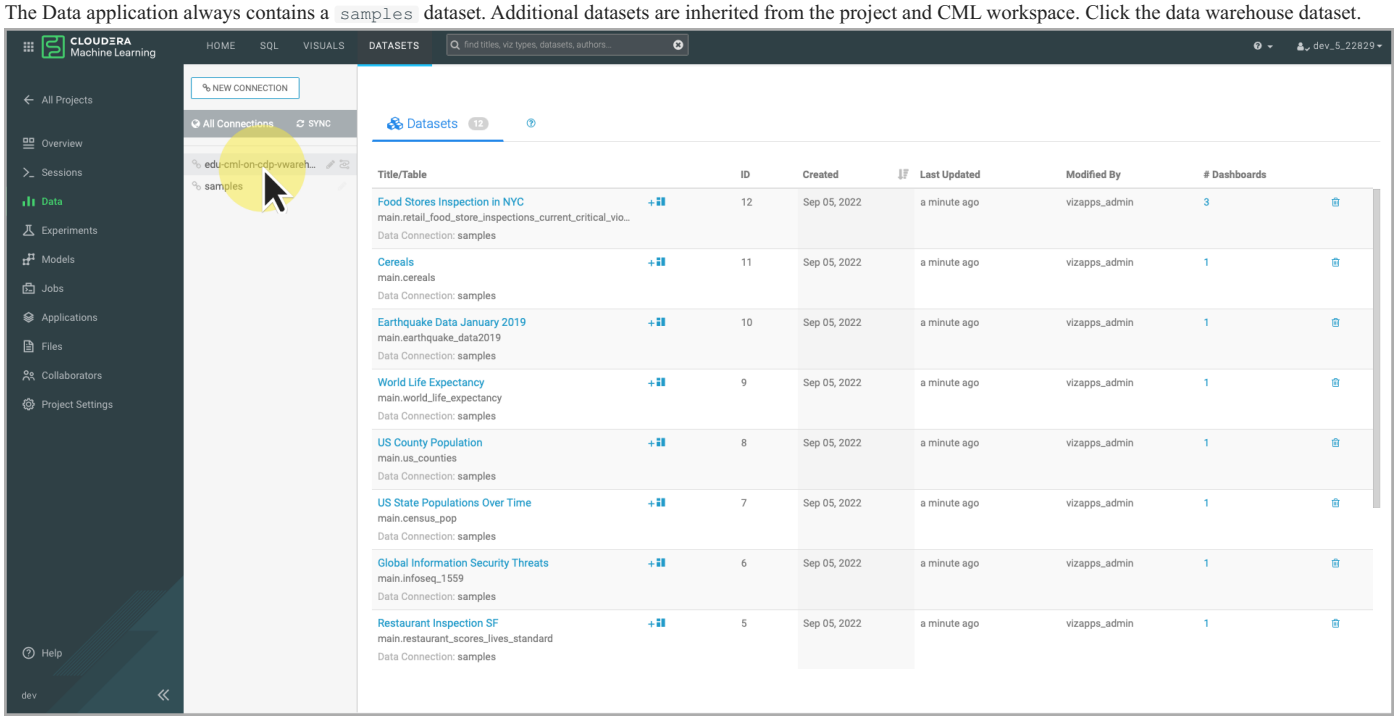
4. The Data application homepage has useful information to get started and shortcut to common and recently used items.



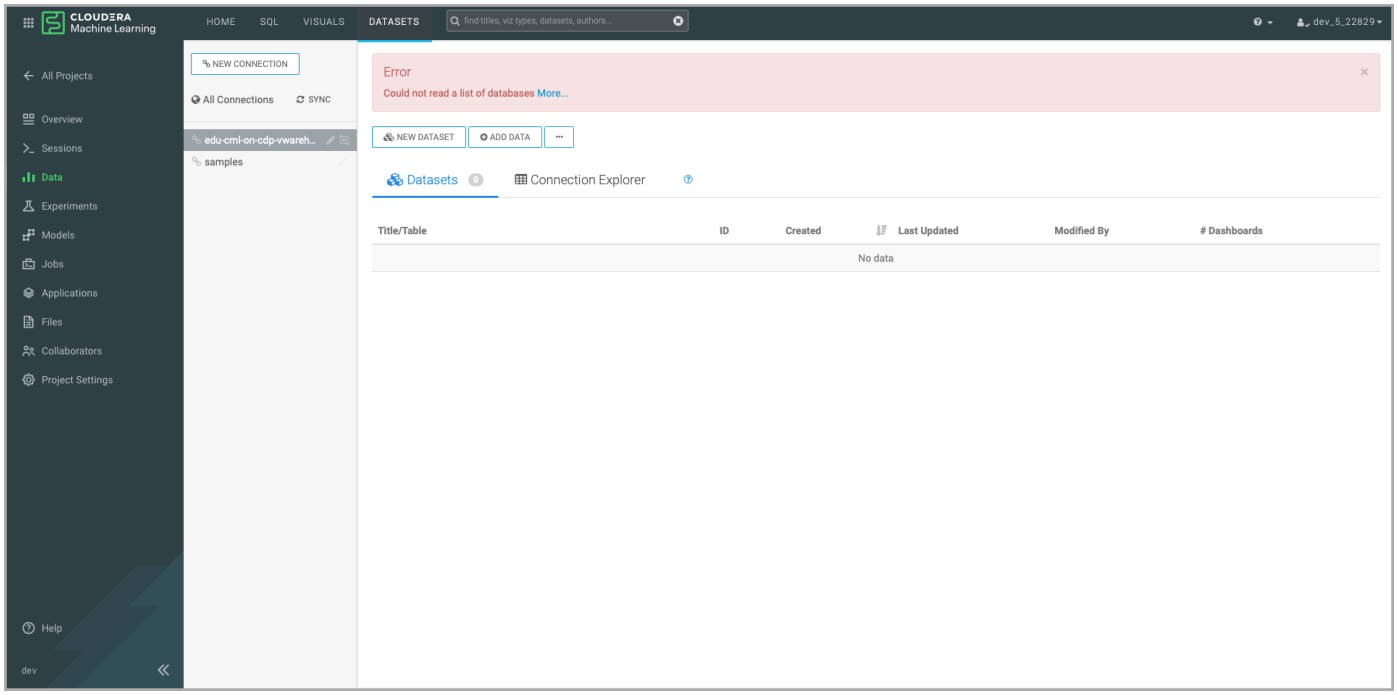
5. Click **Datasets** in the menu at the top of the application.l



6. The Data application always contains a `samples` dataset. Additional datasets are inherited from the project and CML workspace. Click the data warehouse dataset.



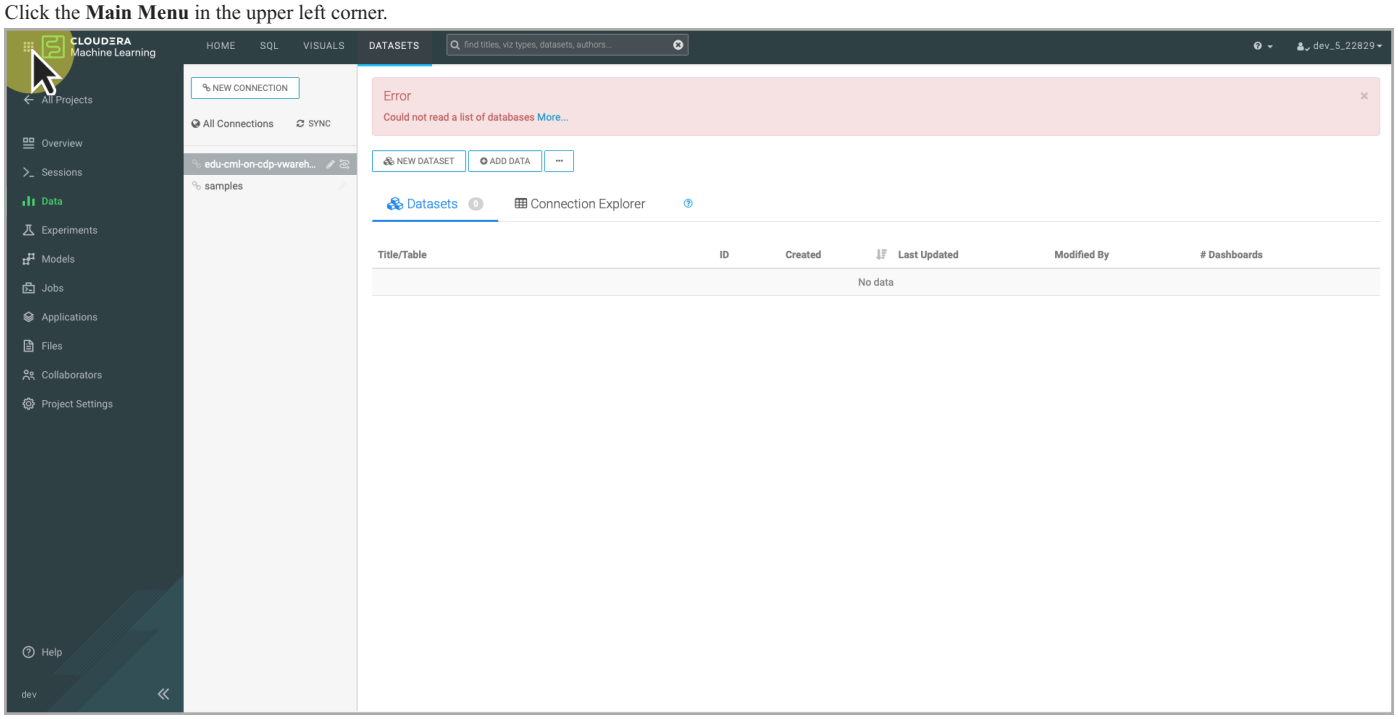
7. If this is your first time accessing a dataset, you will receive an error that Data application cannot read a list of databases. This is because the Data application needs a Workload Password to access the data warehouse.



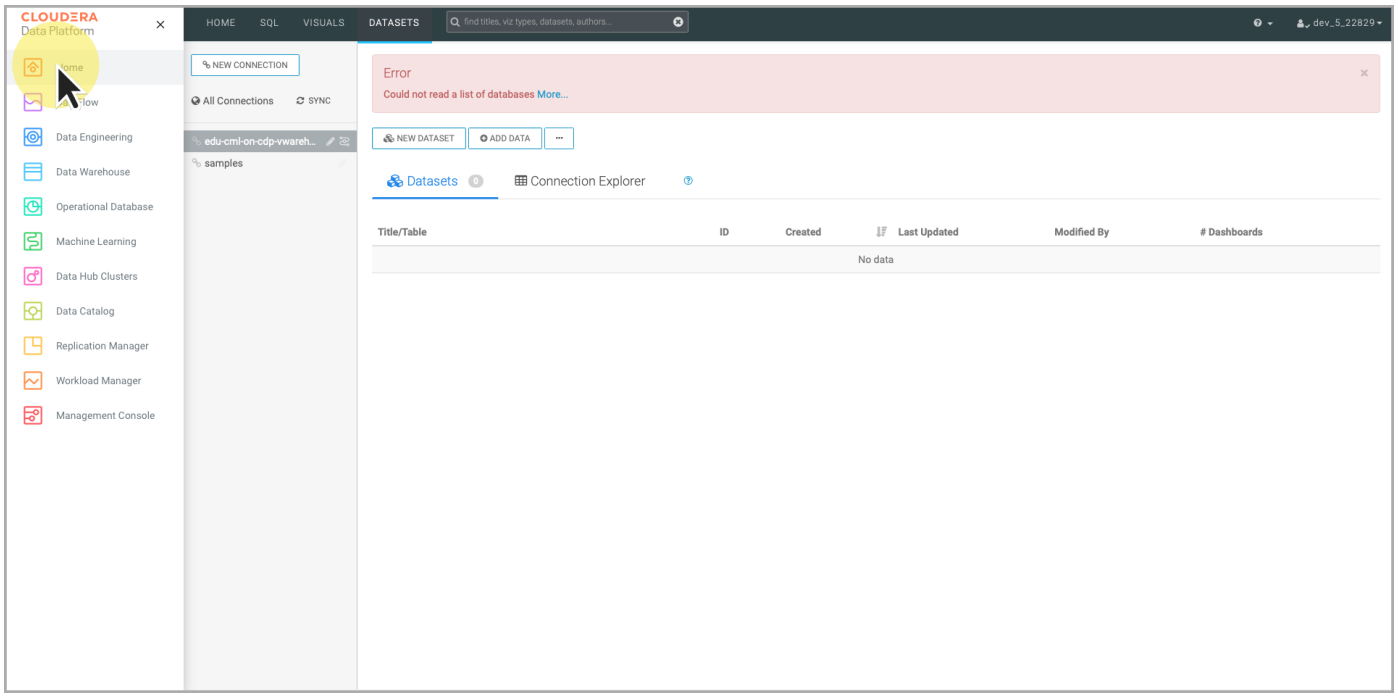
Create Workload Password (if needed)

If you received an error on the prior step, continue to follow the instructions below. If you did not receive an error, skip ahead to create a new dataset.

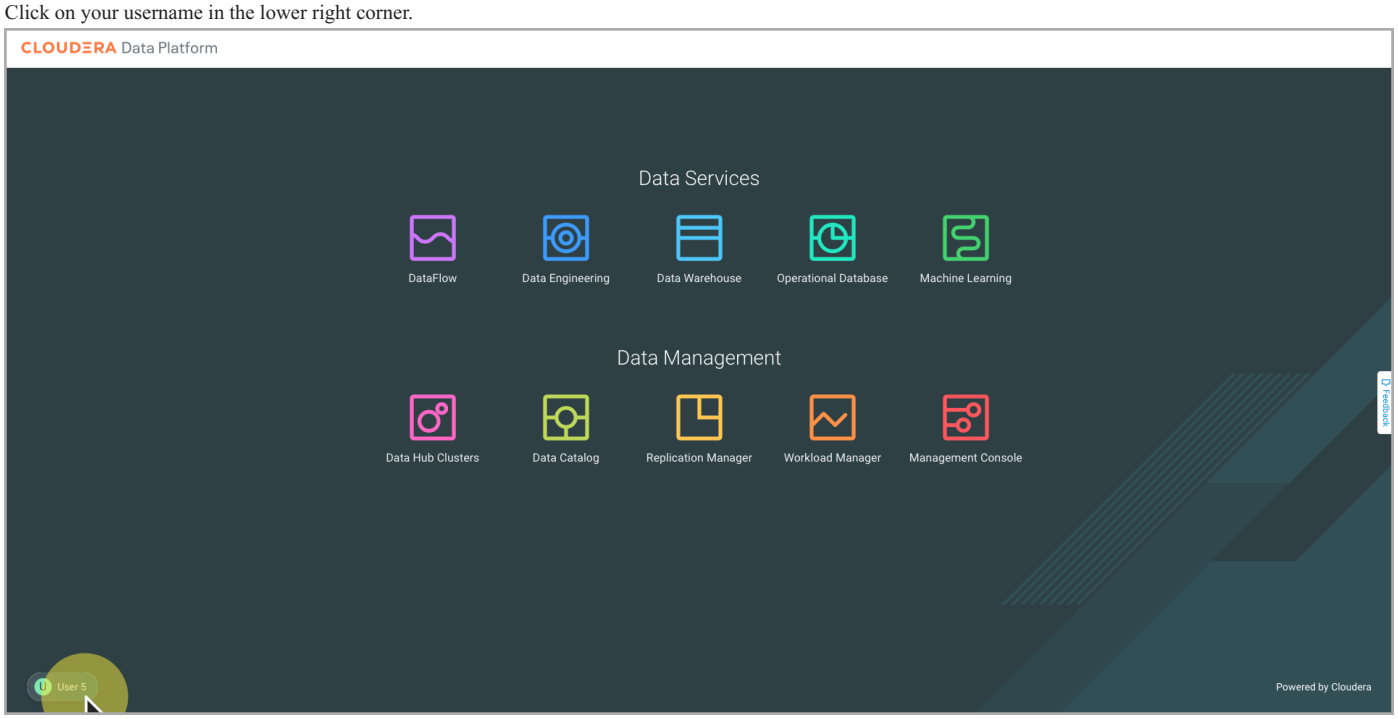
1. Click the **Main Menu** in the upper left corner.



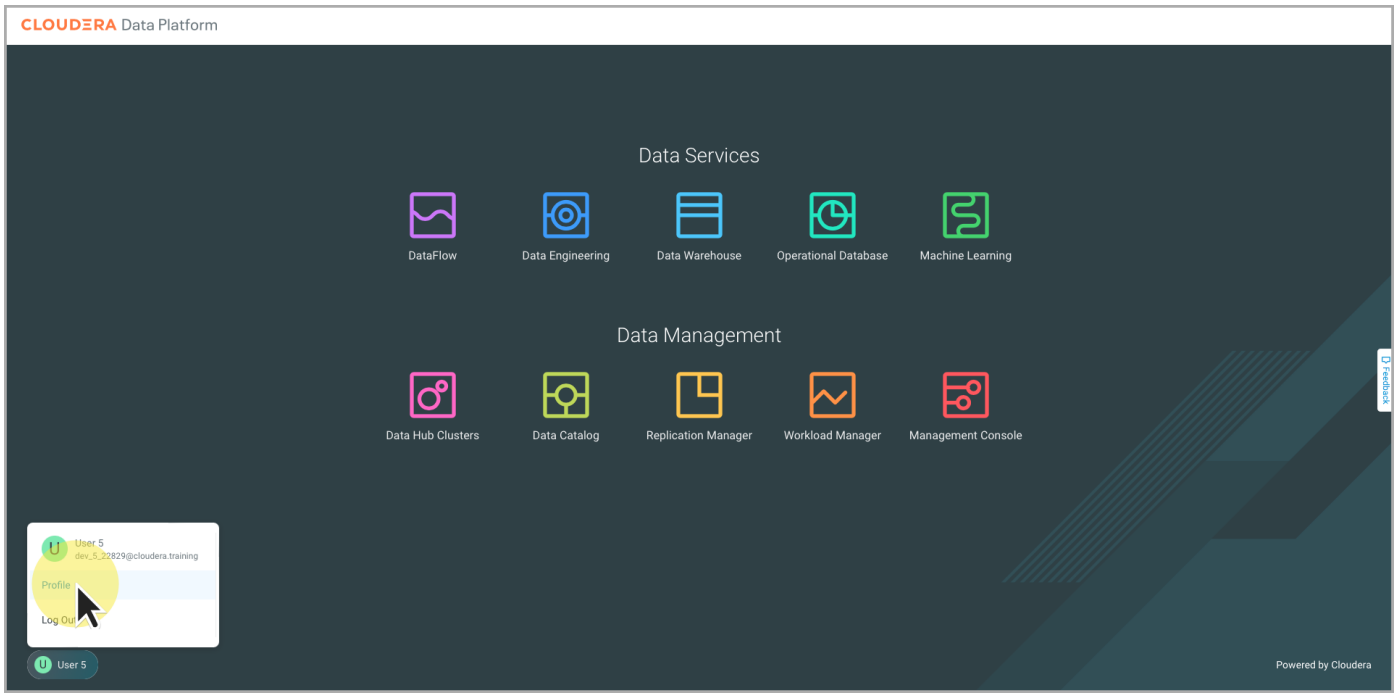
2. Click **Home**.



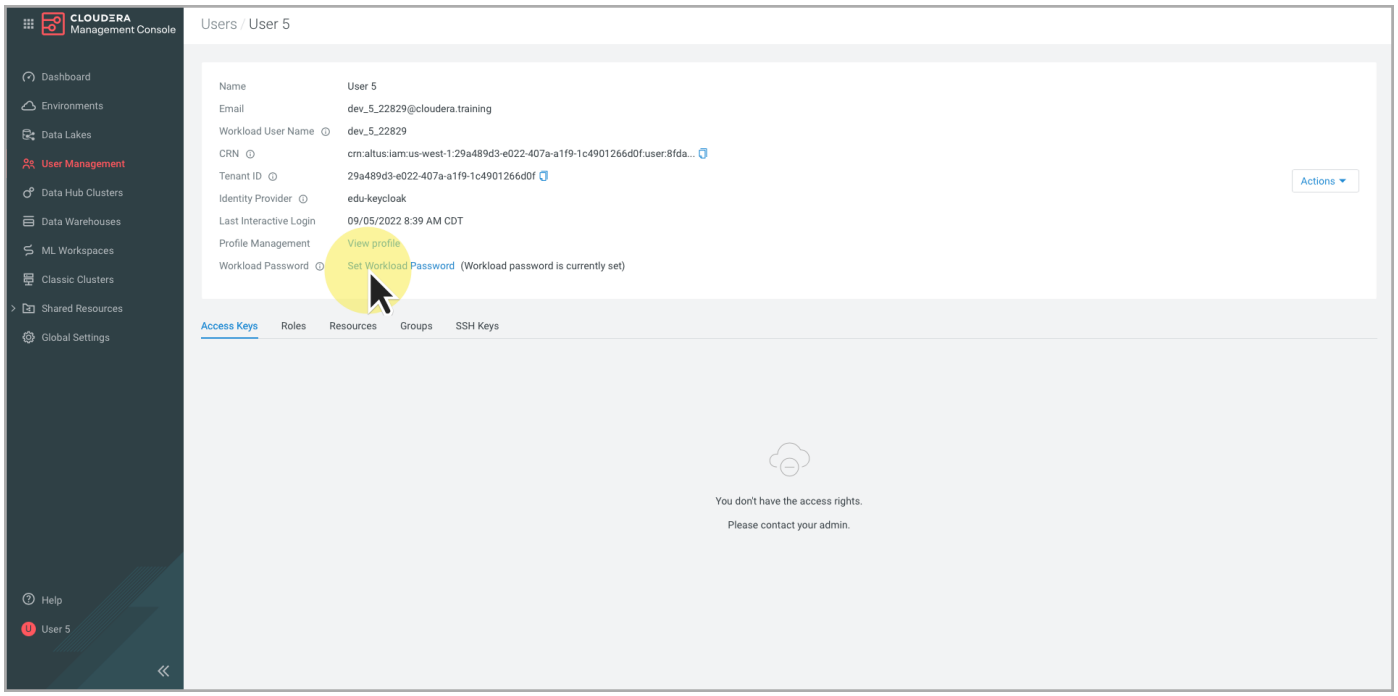
3. Click on your username in the lower right corner.



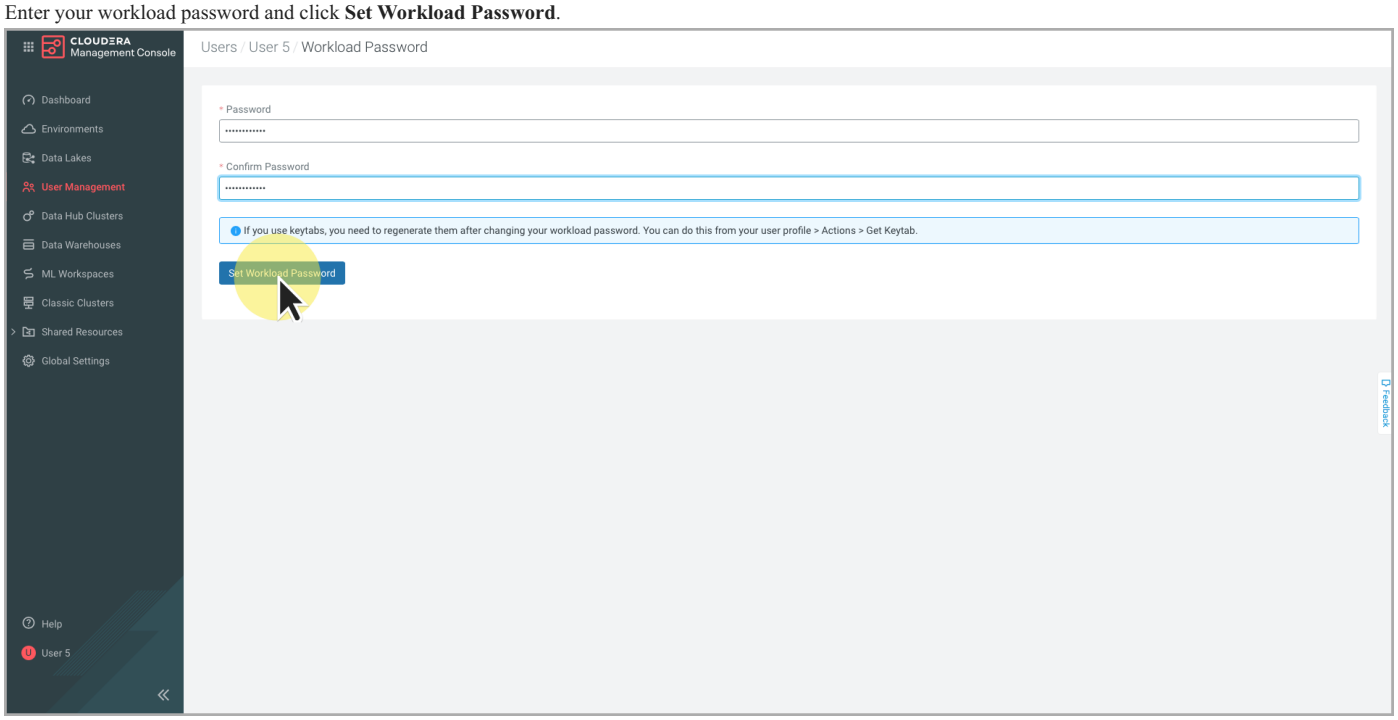
4. Click **Profile**.



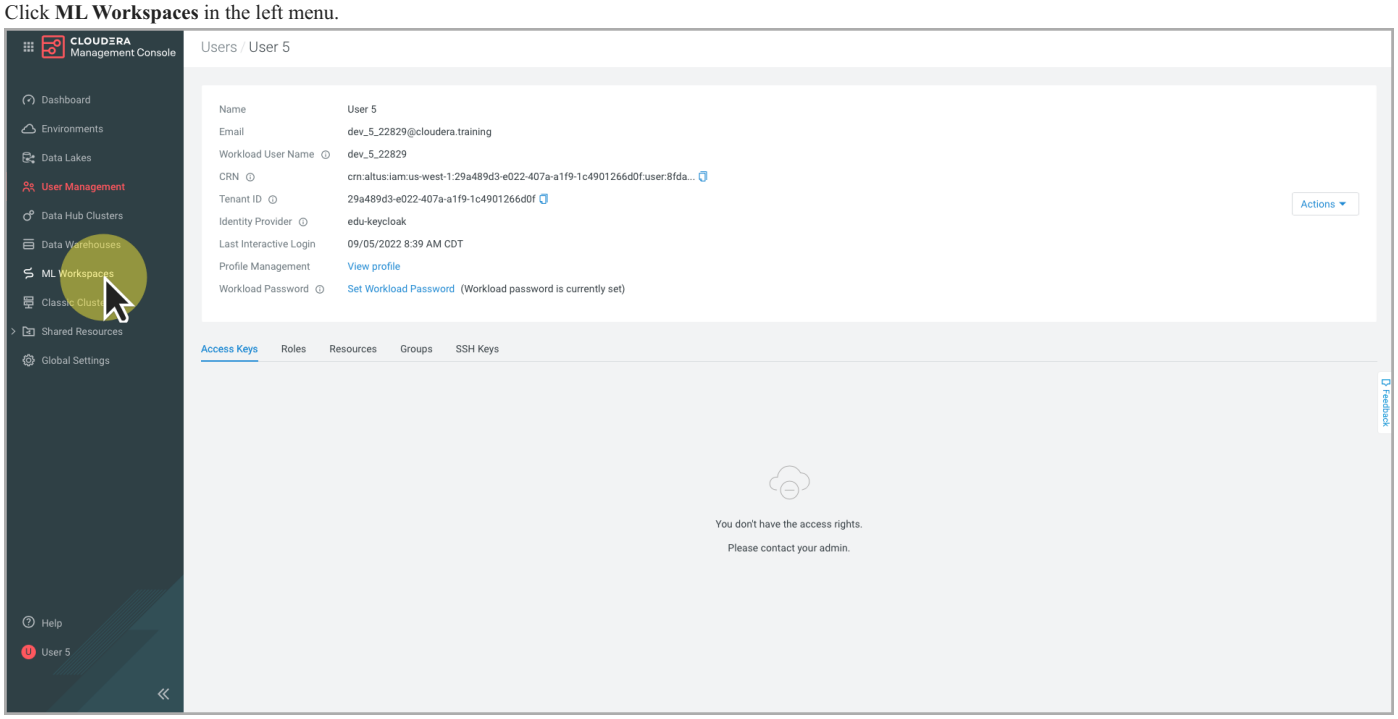
5. Click **Set Workload Password**.



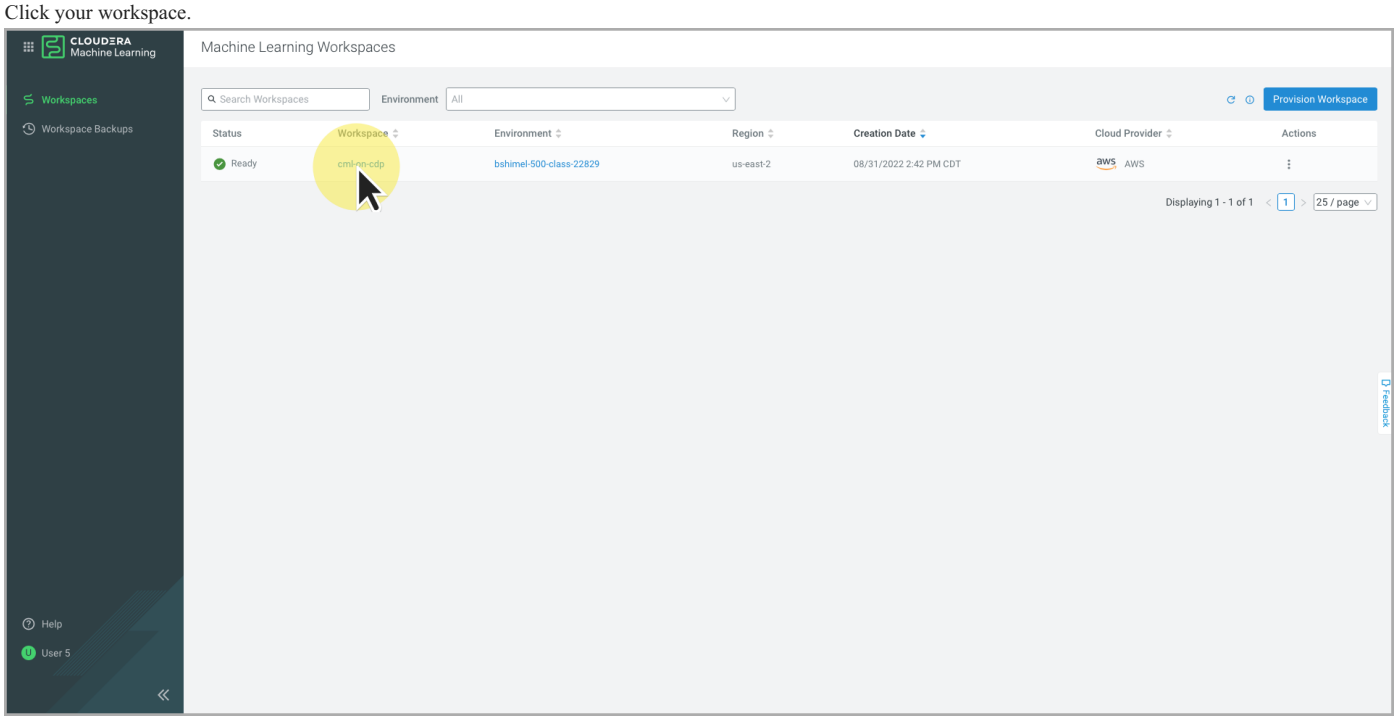
6. Enter your workload password and click **Set Workload Password**.



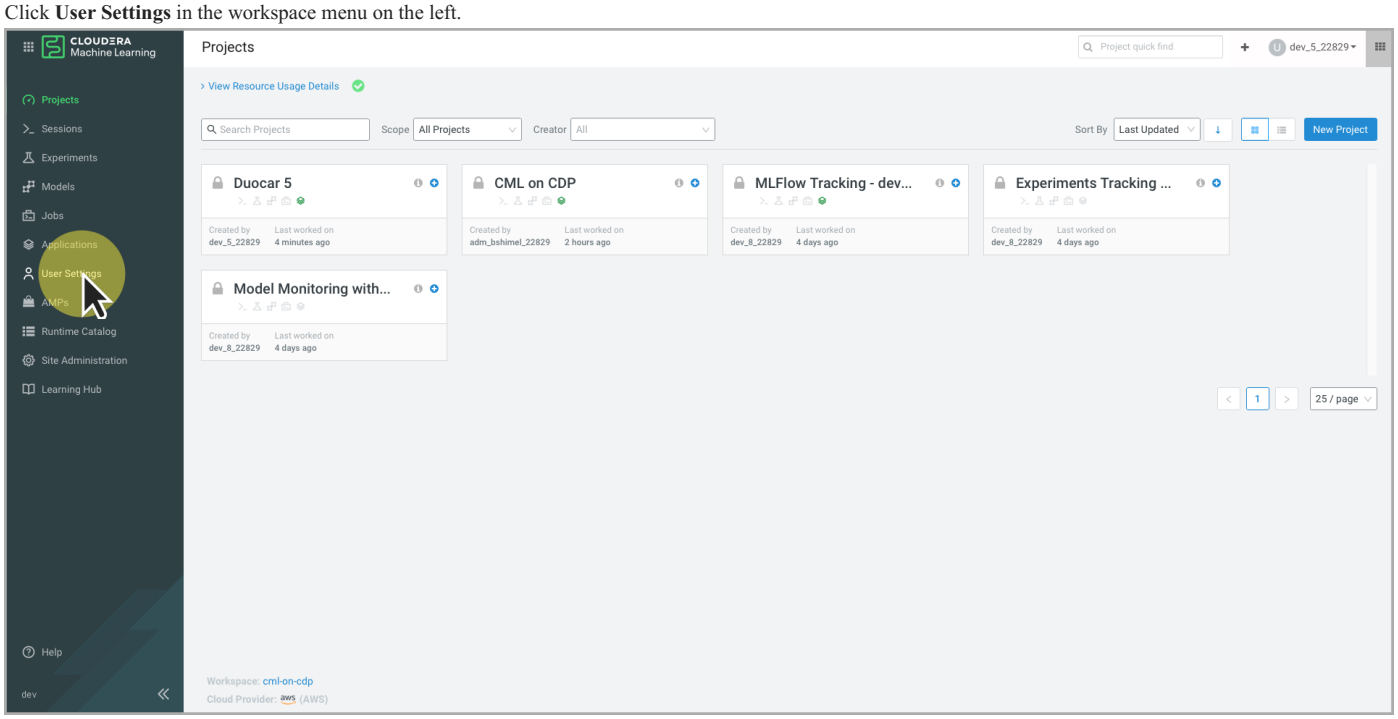
7. Click **ML Workspaces** in the left menu.



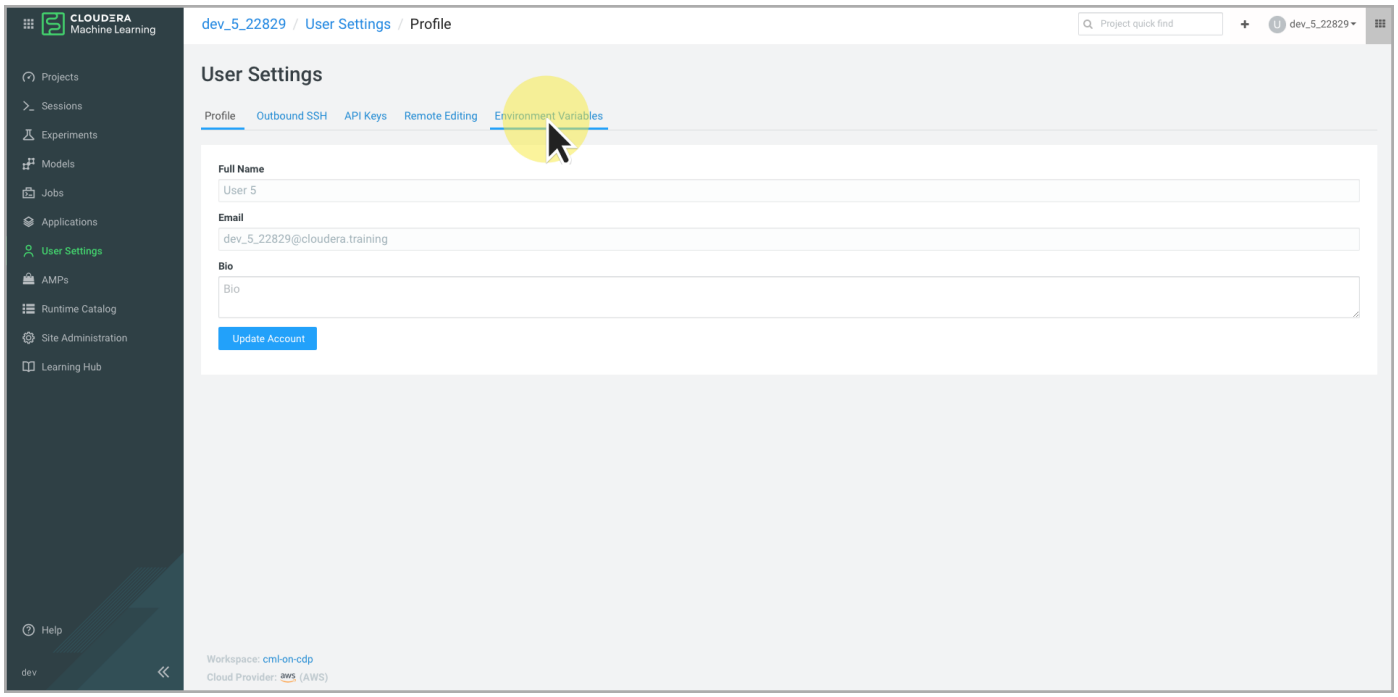
8. Click your workspace.



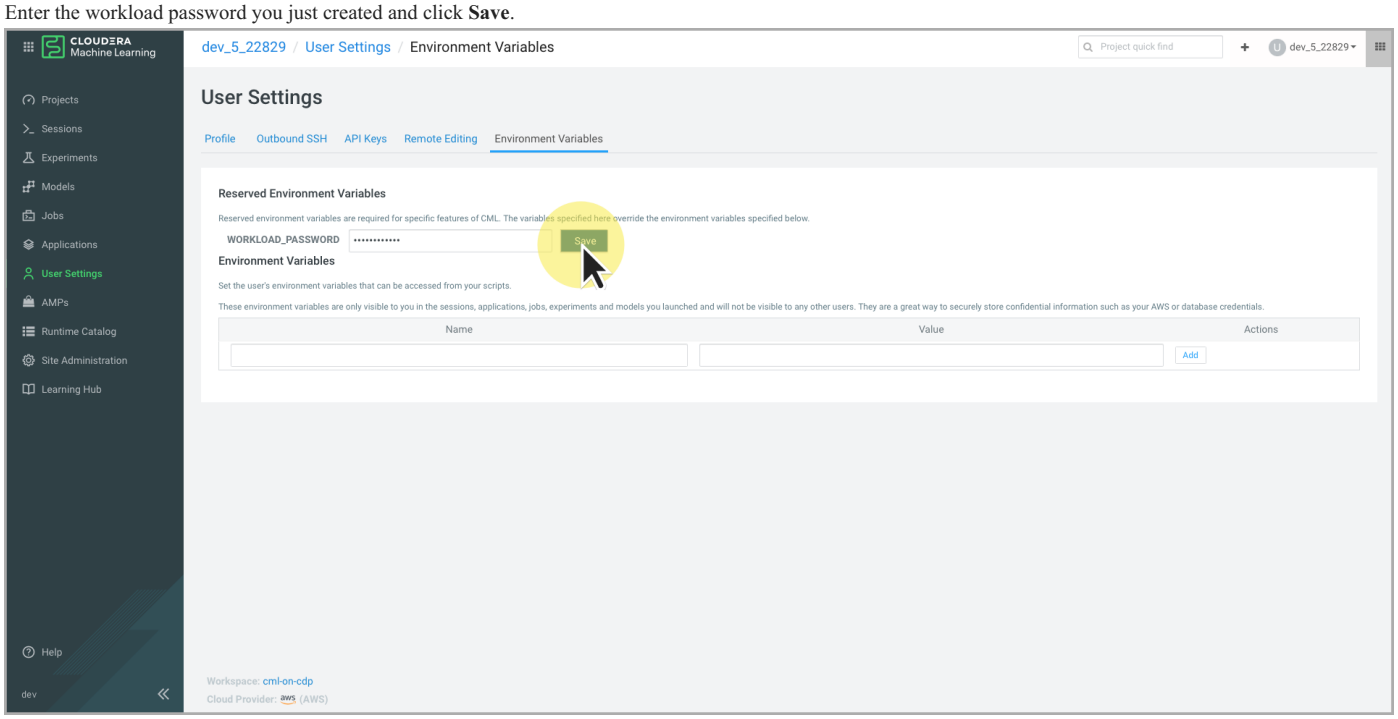
9. Click **User Settings** in the workspace menu on the left.



10. Click the **Environment Variables** tab.

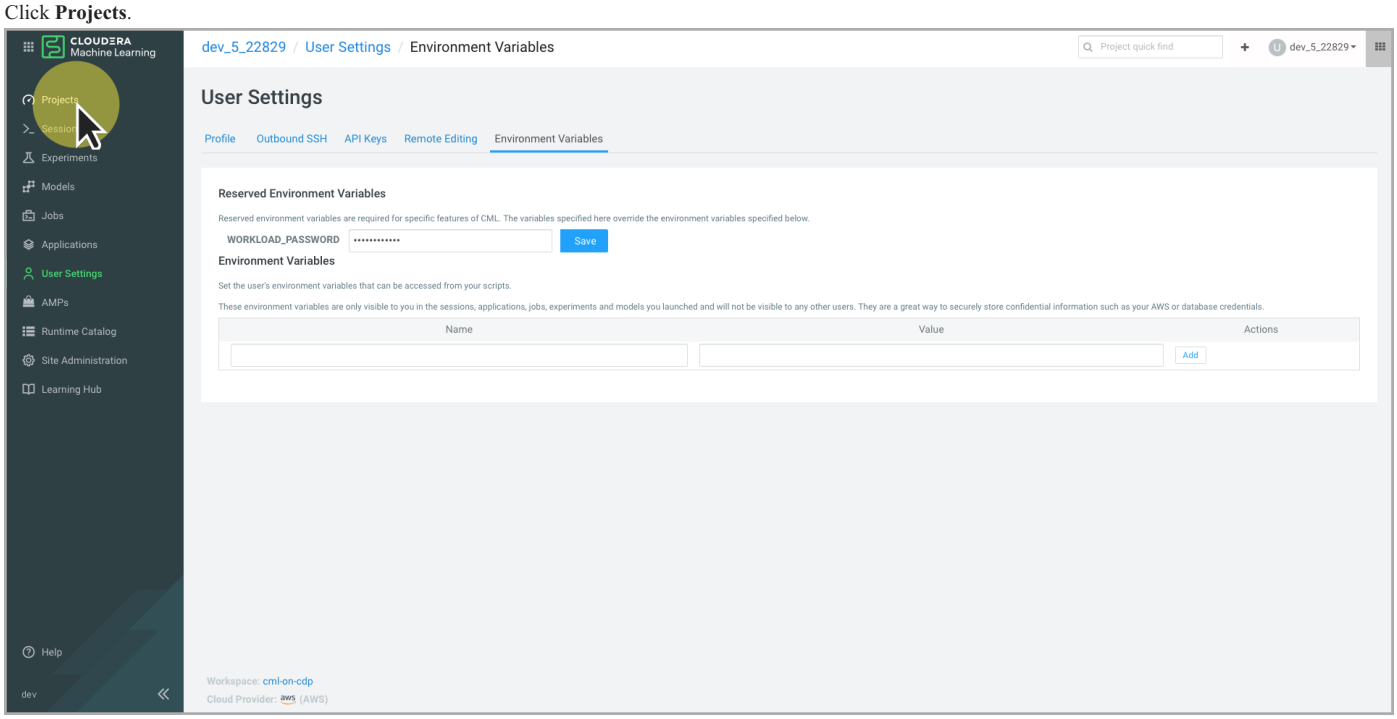


11. Enter the workload password you just created and click **Save**.

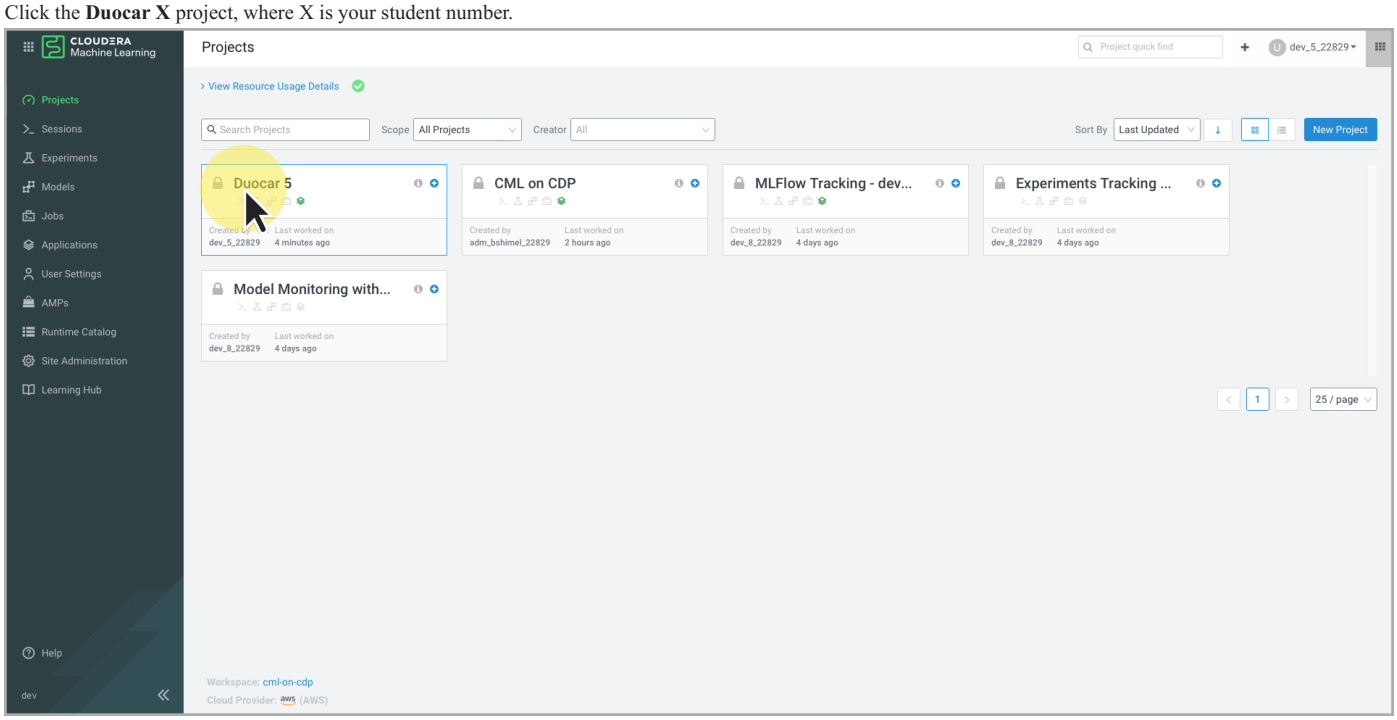


Now that your workload password has been set. The **Data Discovery and Visualization** application needs to be restarted.

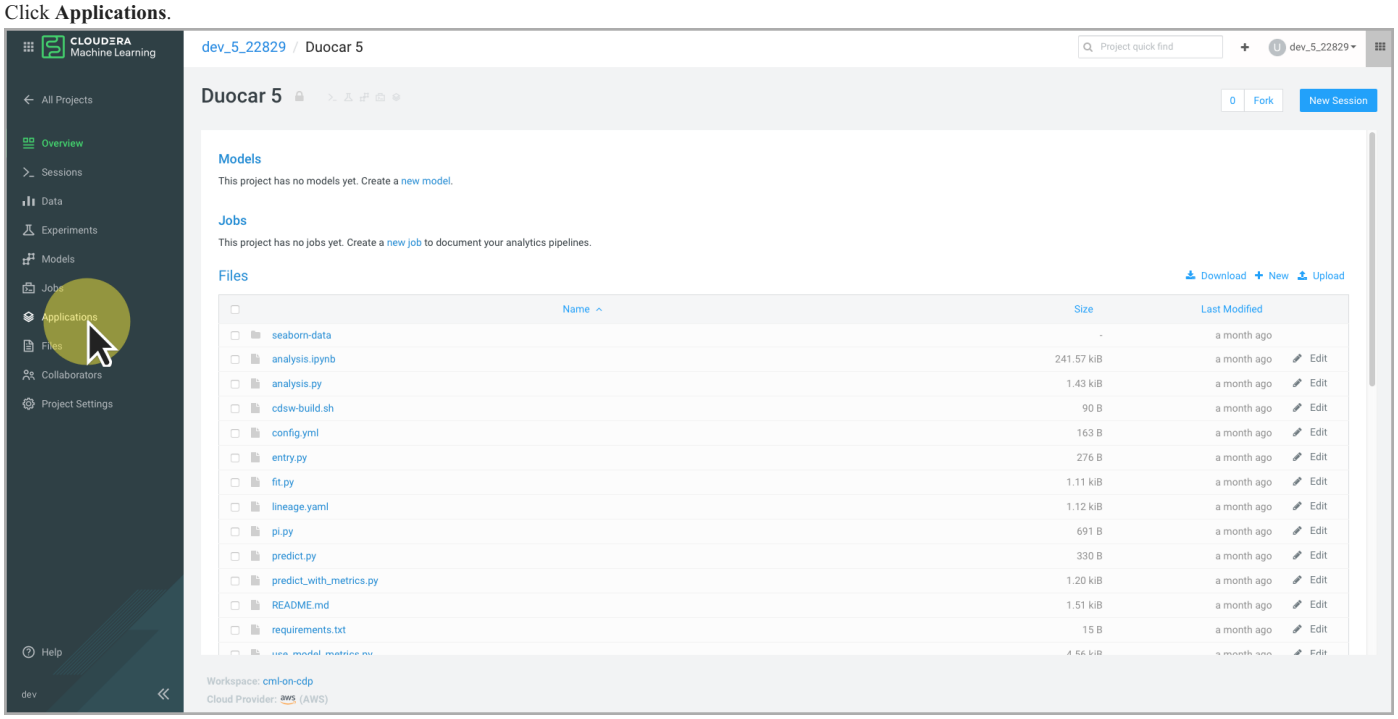
1. Click **Projects**.



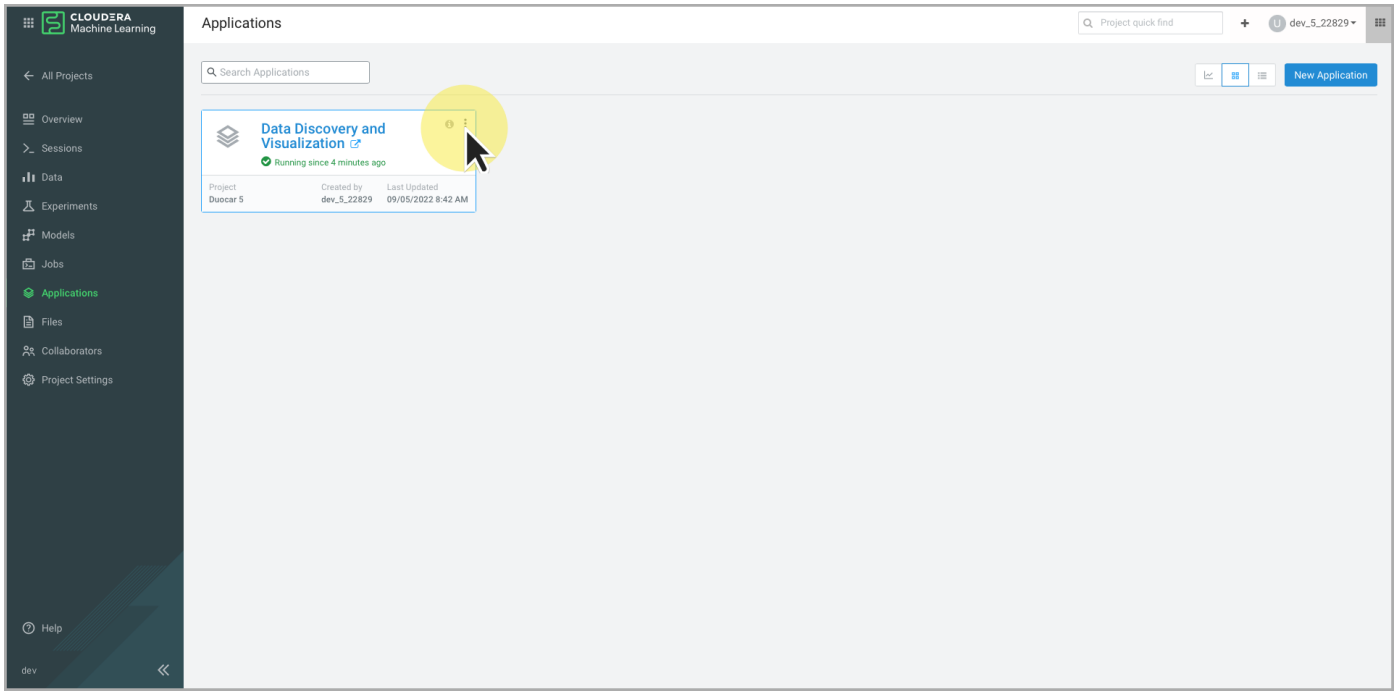
2. Click the **Duocar X** project, where X is your student number.



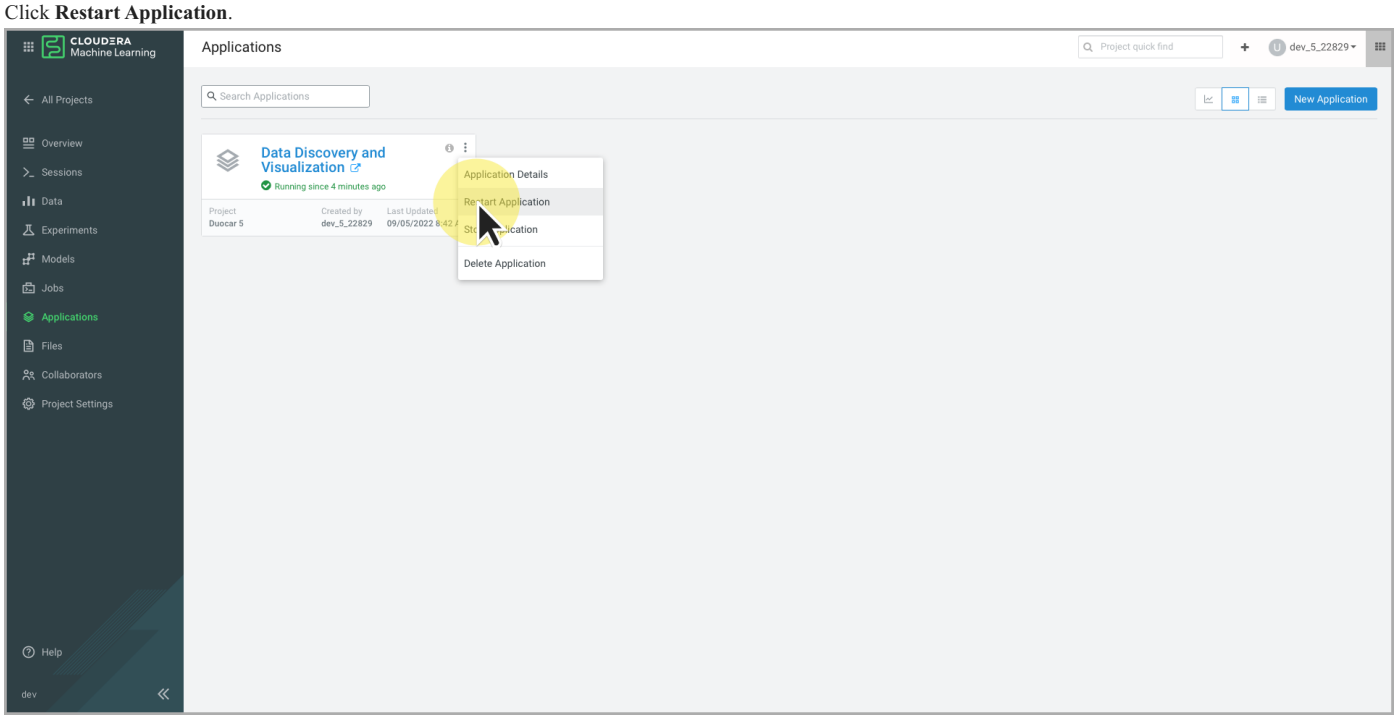
3. Click **Applications**.



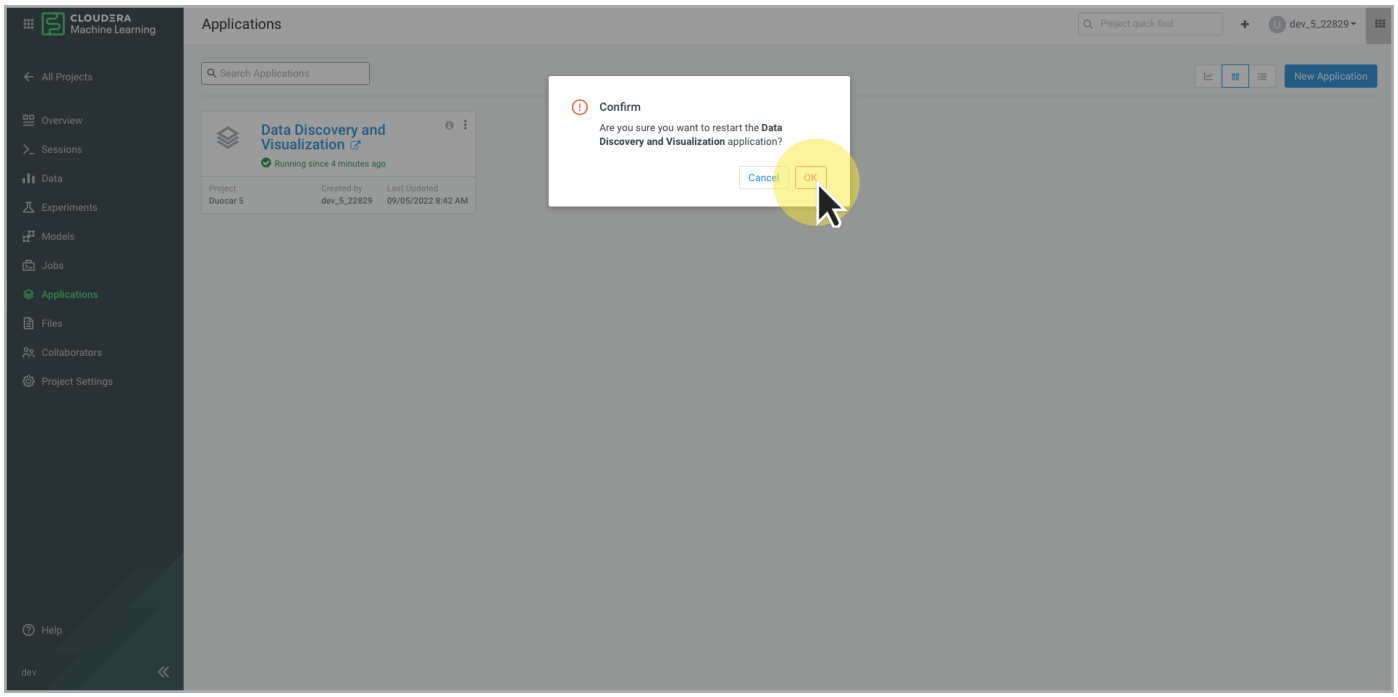
4. Click ⋮ menu in the upper right corner of the **Data Discovery and Visualization** application.



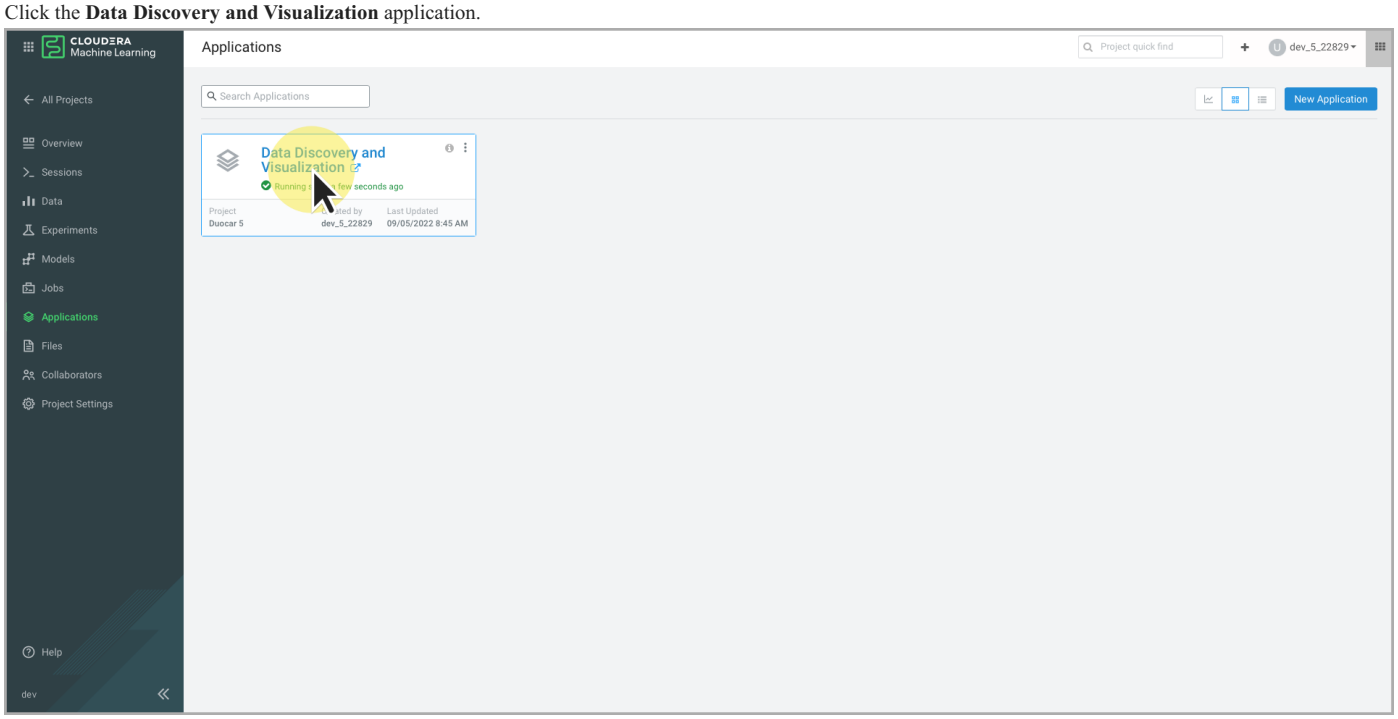
5. Click **Restart Application**.



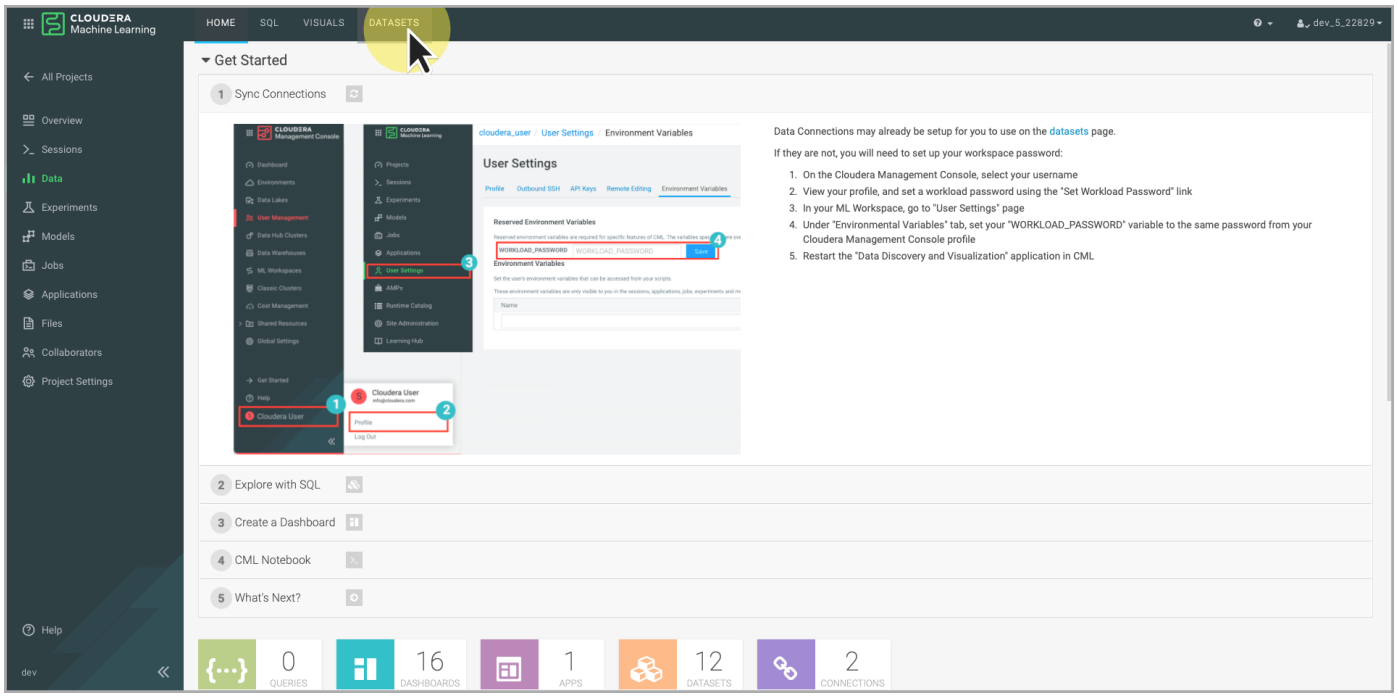
6. Click **OK** to confirm.



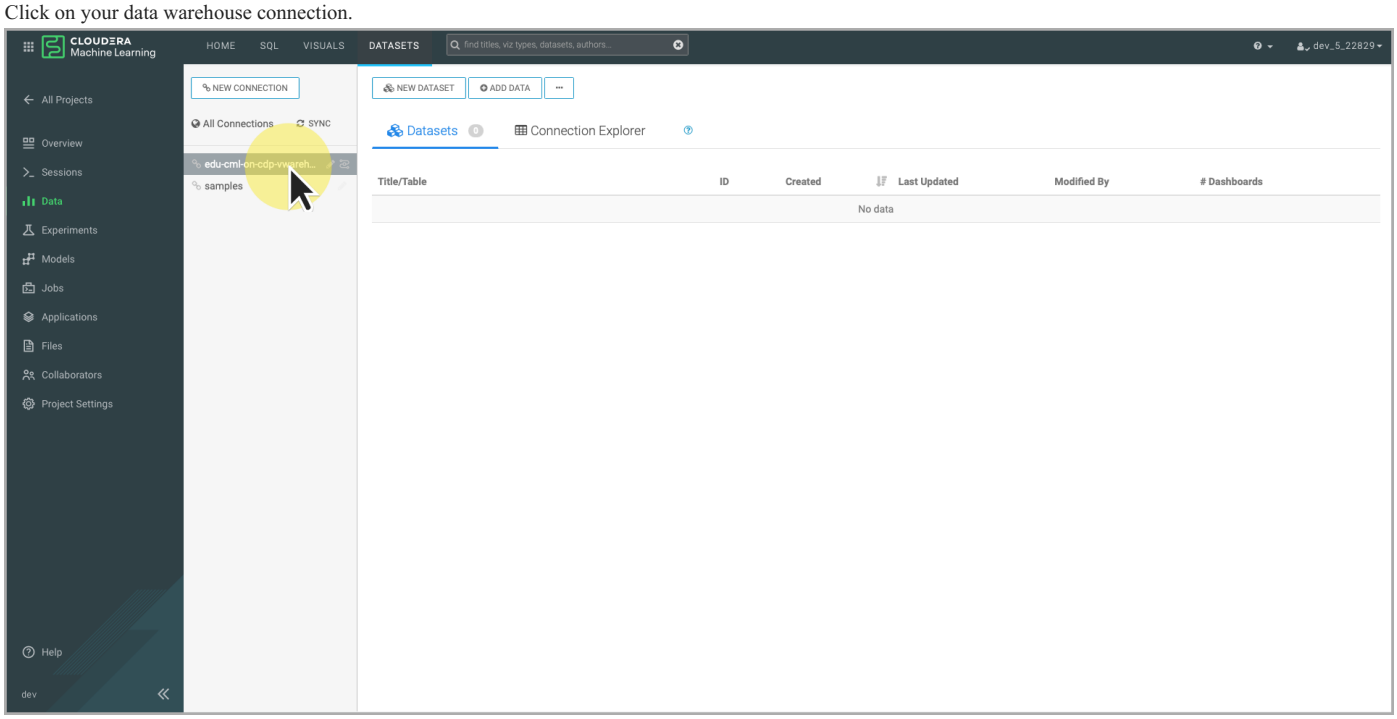
7. Click the **Data Discovery and Visualization** application.



8. Click **Datasets**.

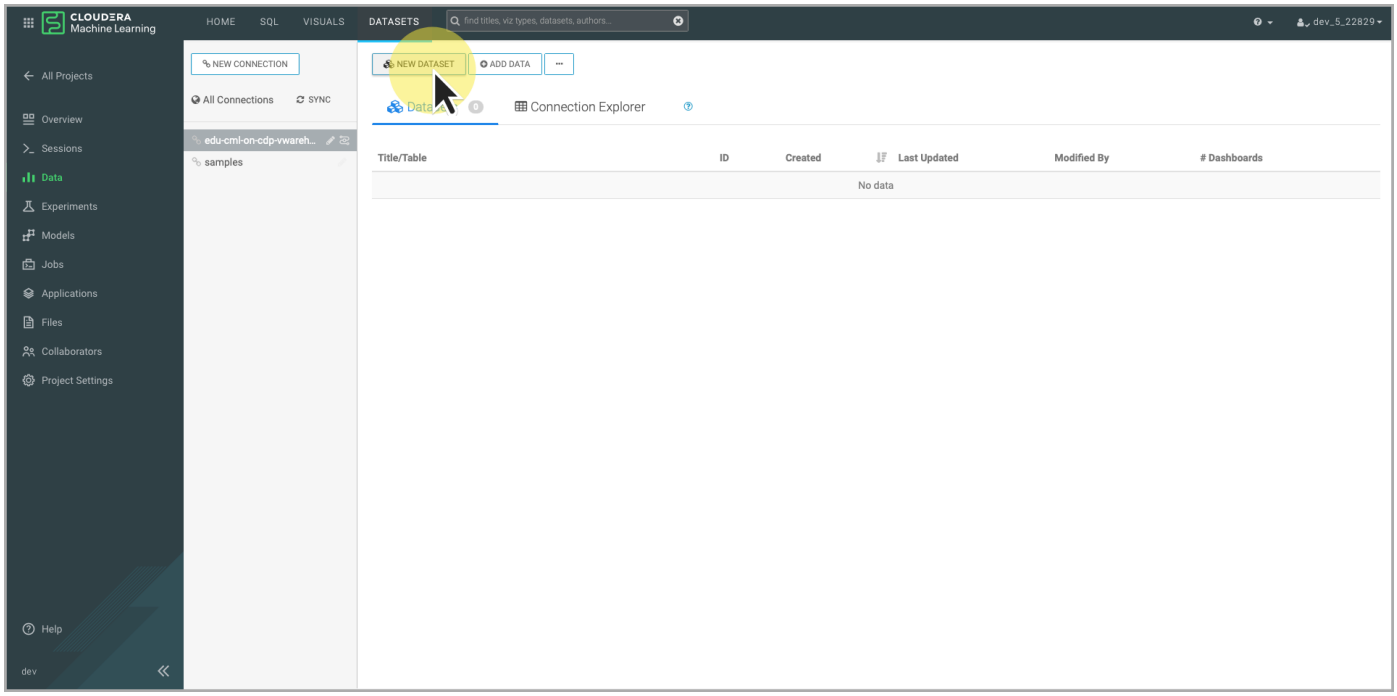


9. Click on your data warehouse connection.

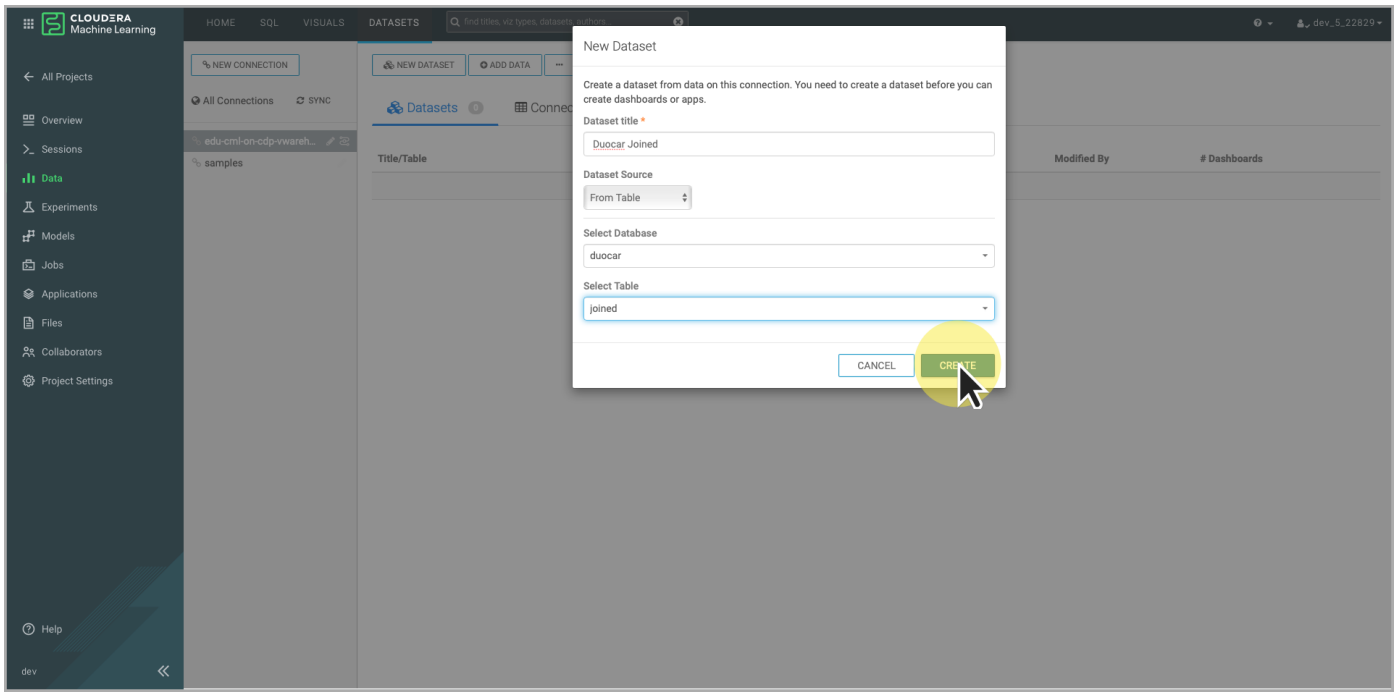


Create a New Dataset

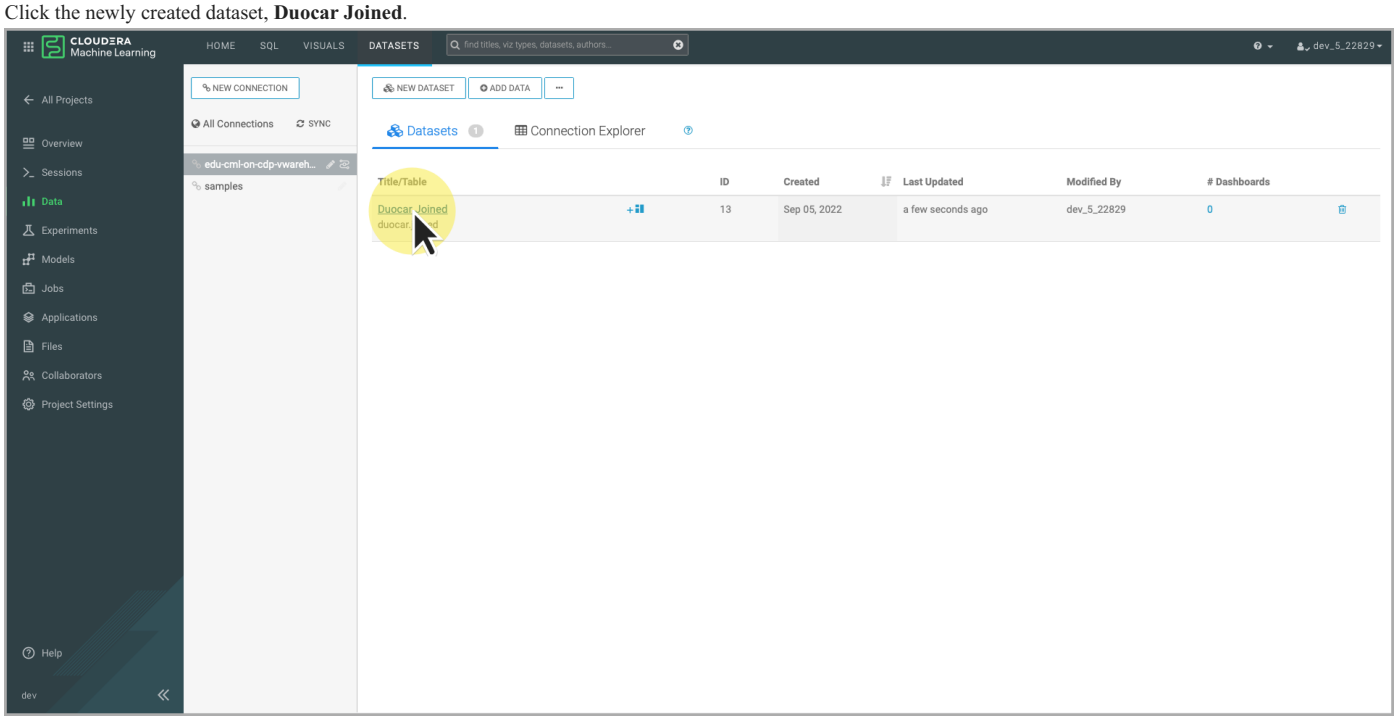
1. Click **New Dataset**



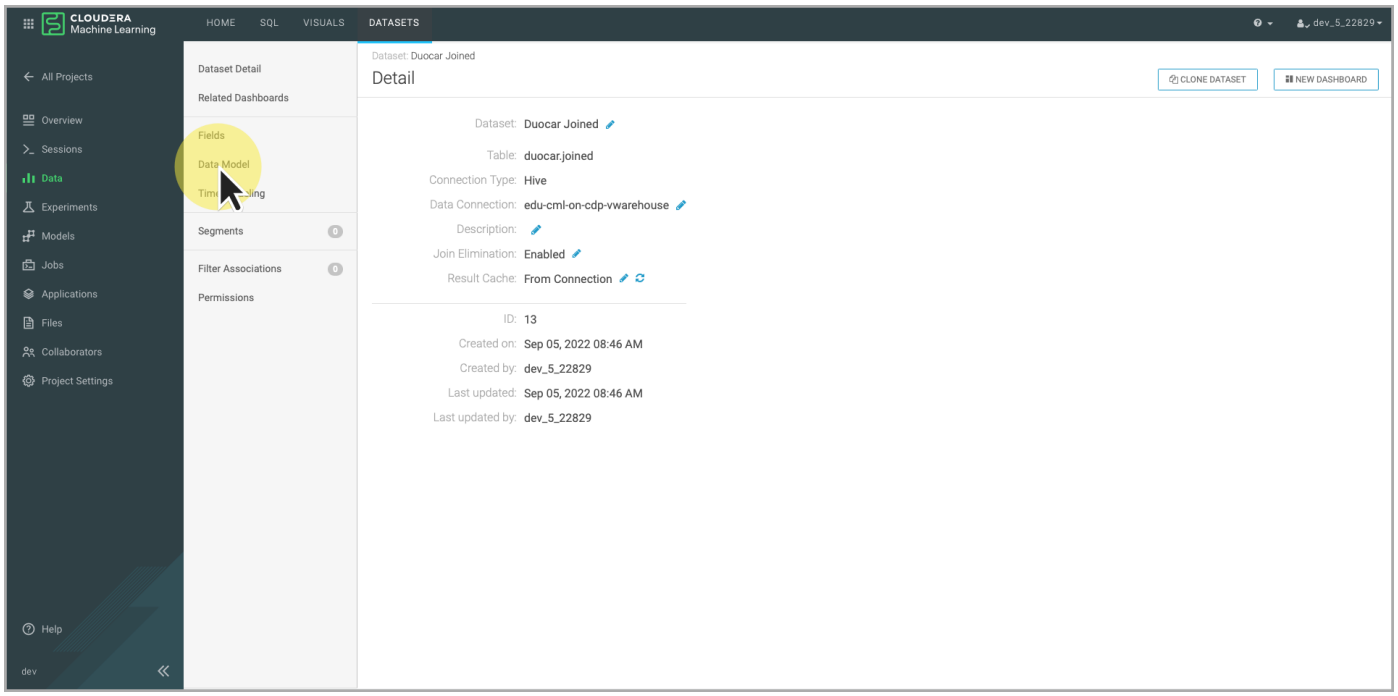
2. Enter `Duocar Joined` for the **Dataset Title**. Select `duocar` as the database. Select `joined` as the table. Click **Create**.



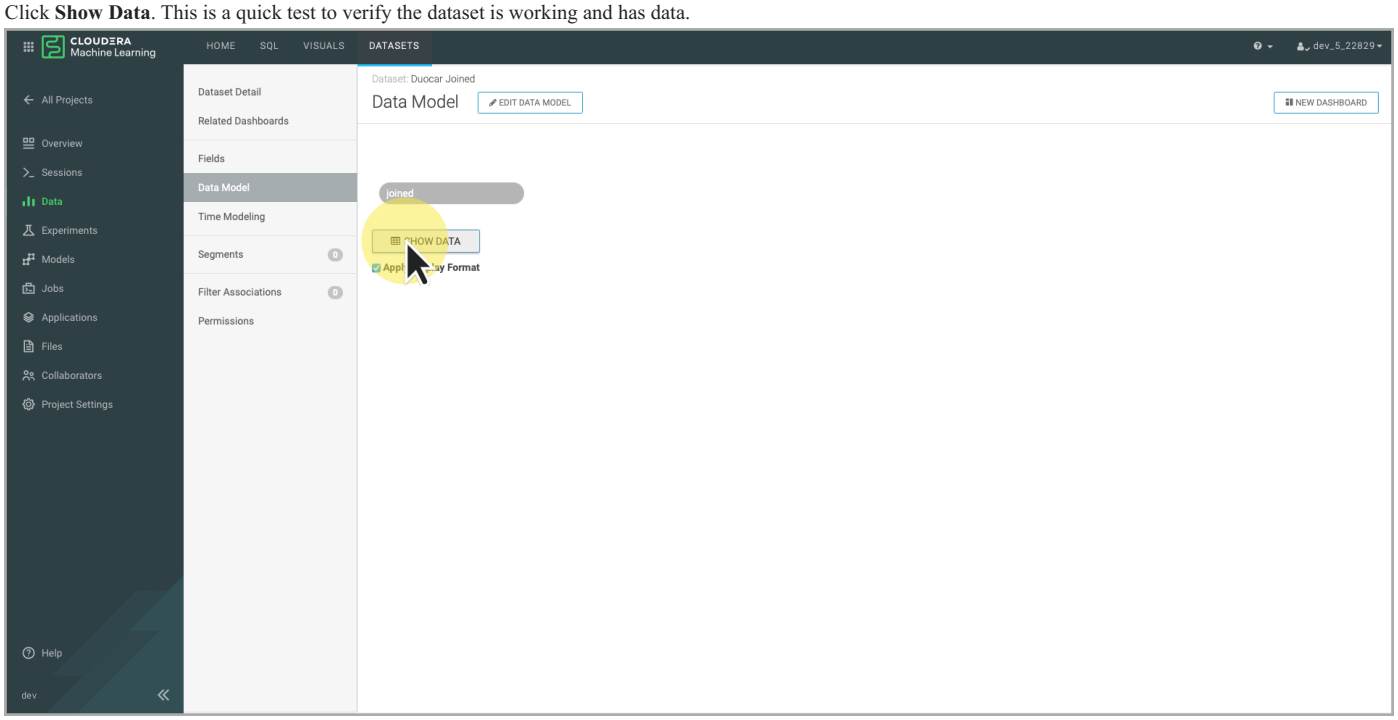
3. Click the newly created dataset, **Duocar Joined**.



4. Click **Data Model** in the **Dataset Detail menu**.

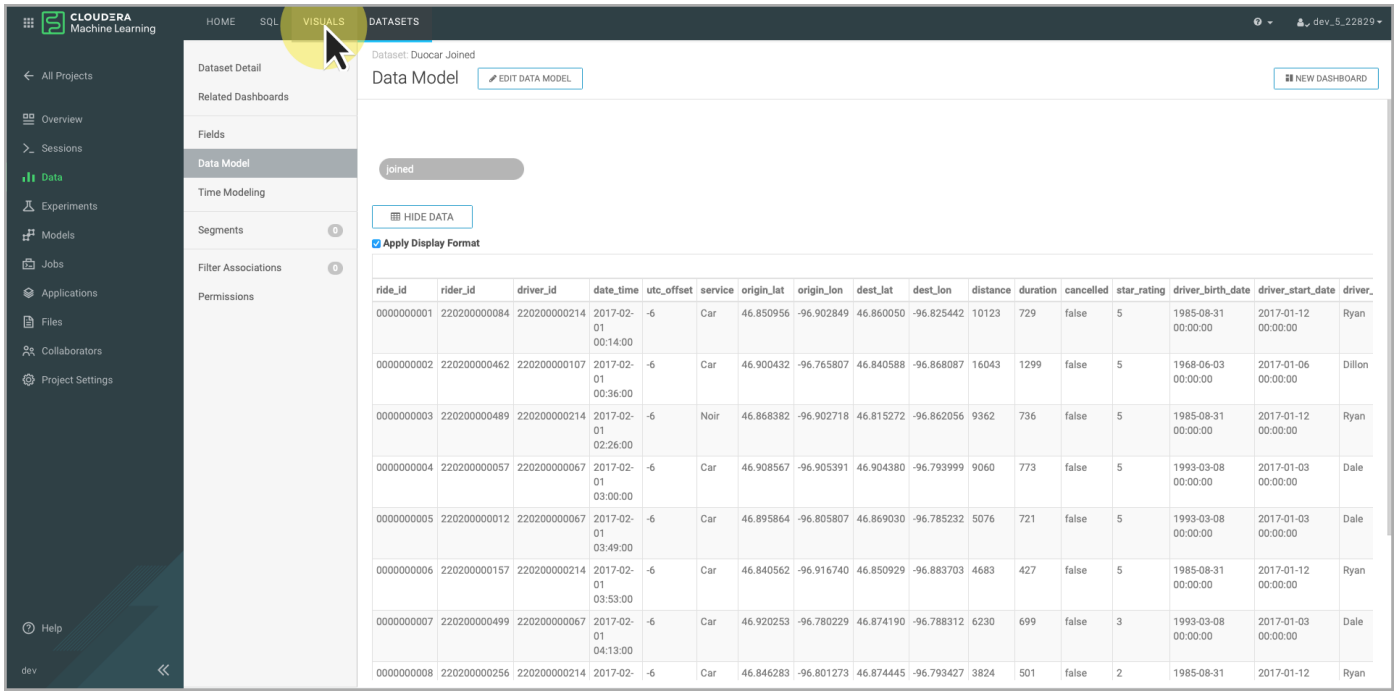


5. Click **Show Data**. This is a quick test to verify the dataset is working and has data.

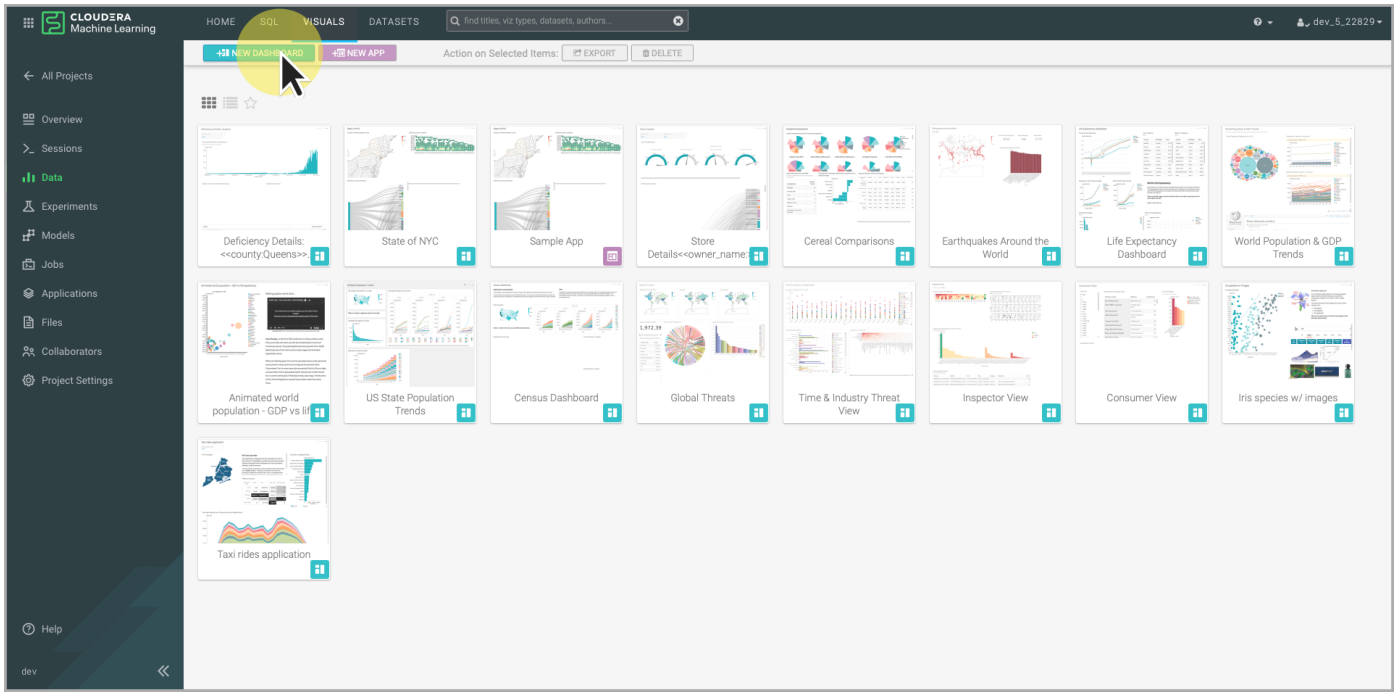


Create a Dashboard

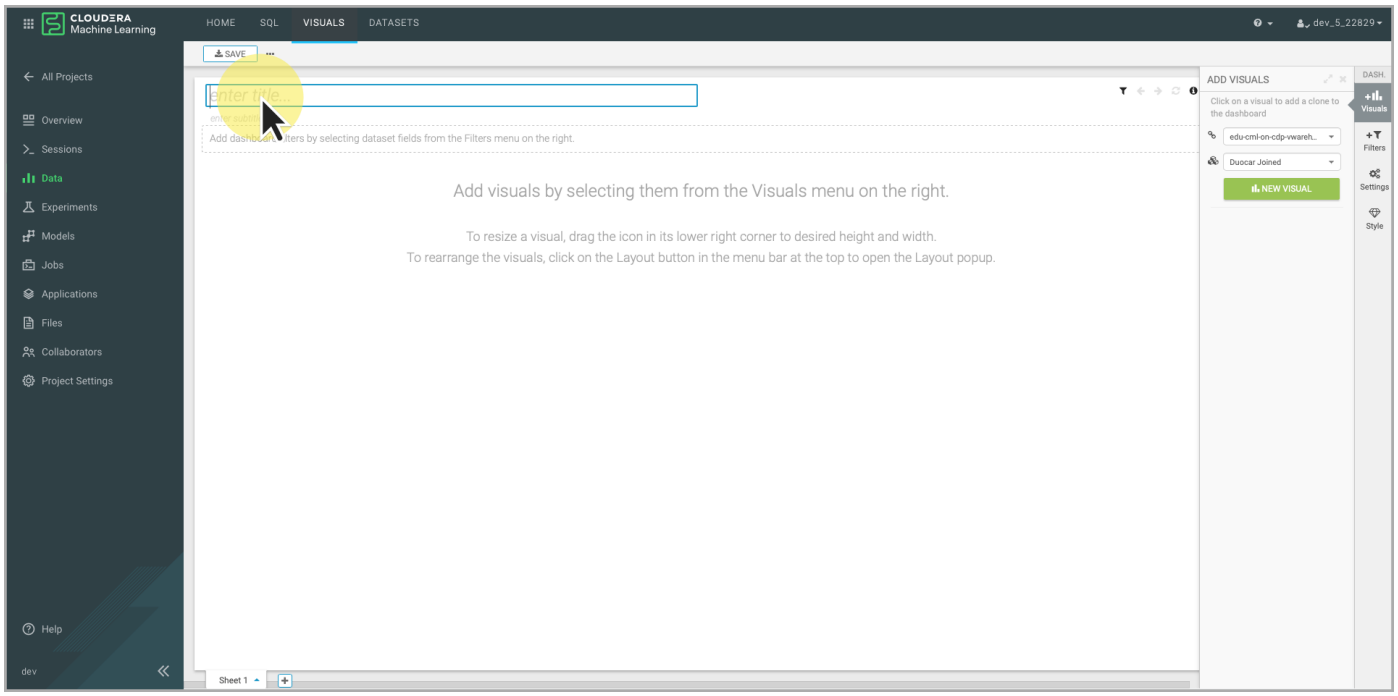
1. Click **Visuals** in the menu at the top of the application.



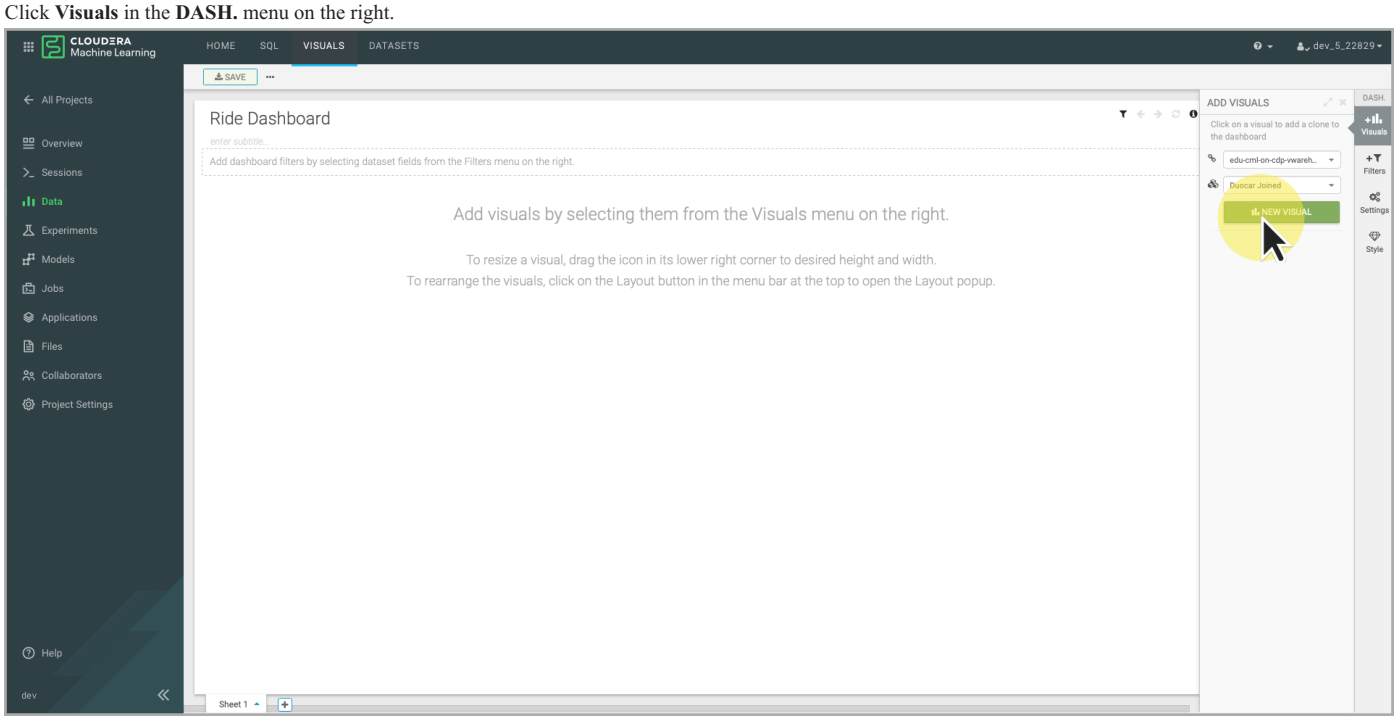
2. Click the **New Dashboard** button.



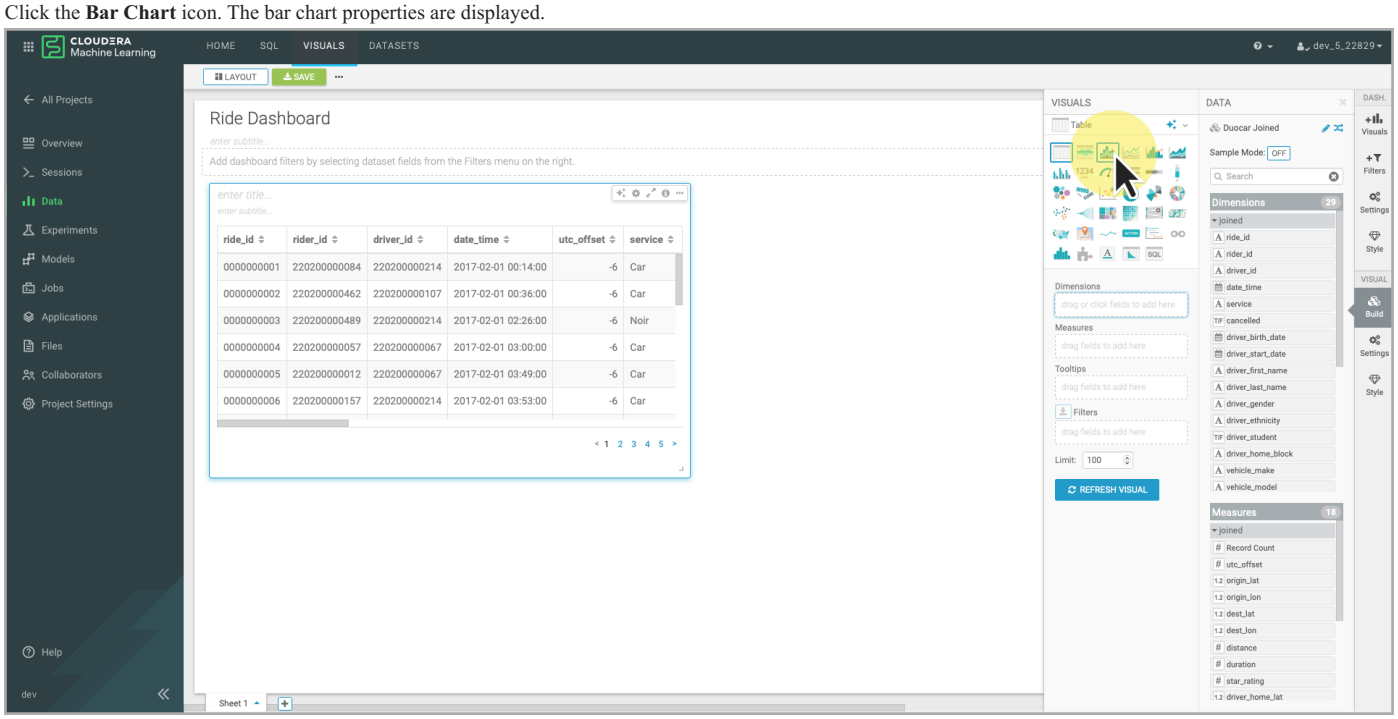
3. Enter `Ride Dashboard` for the title.



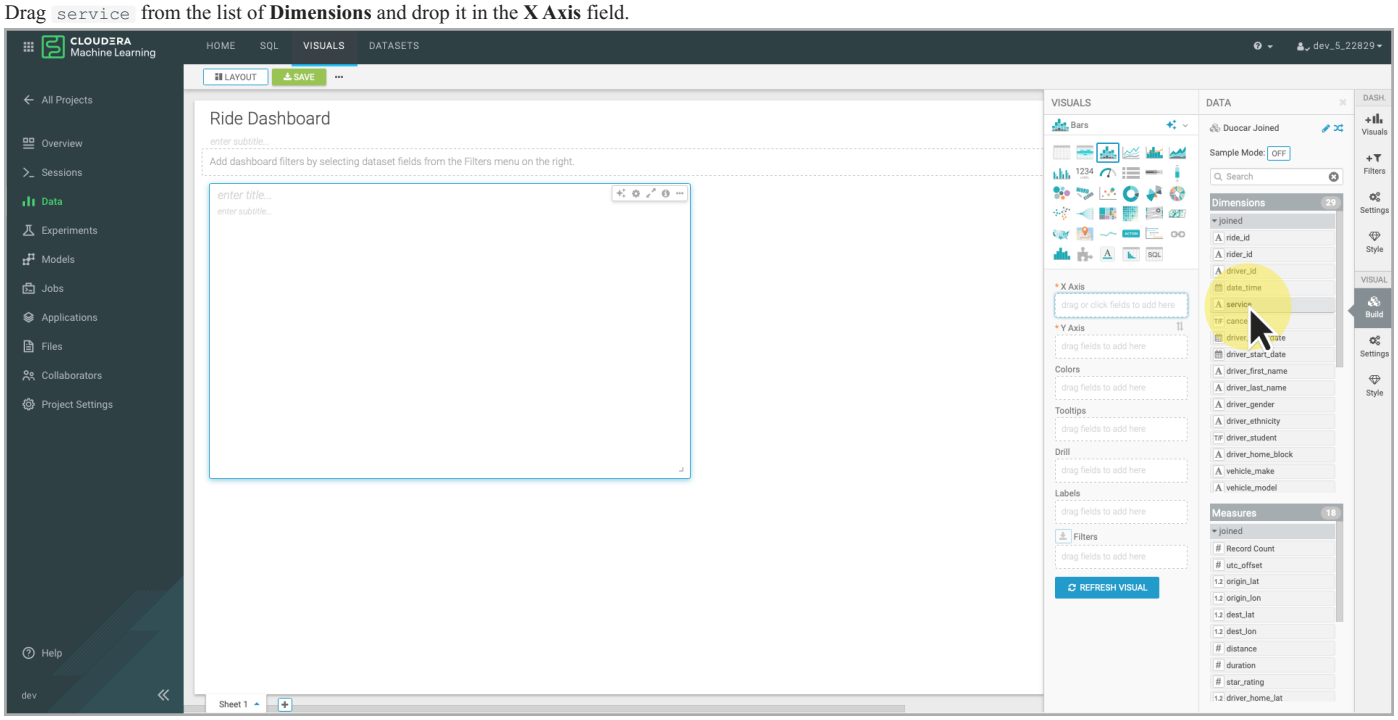
4. Click **Visuals** in the **DASH.** menu on the right.



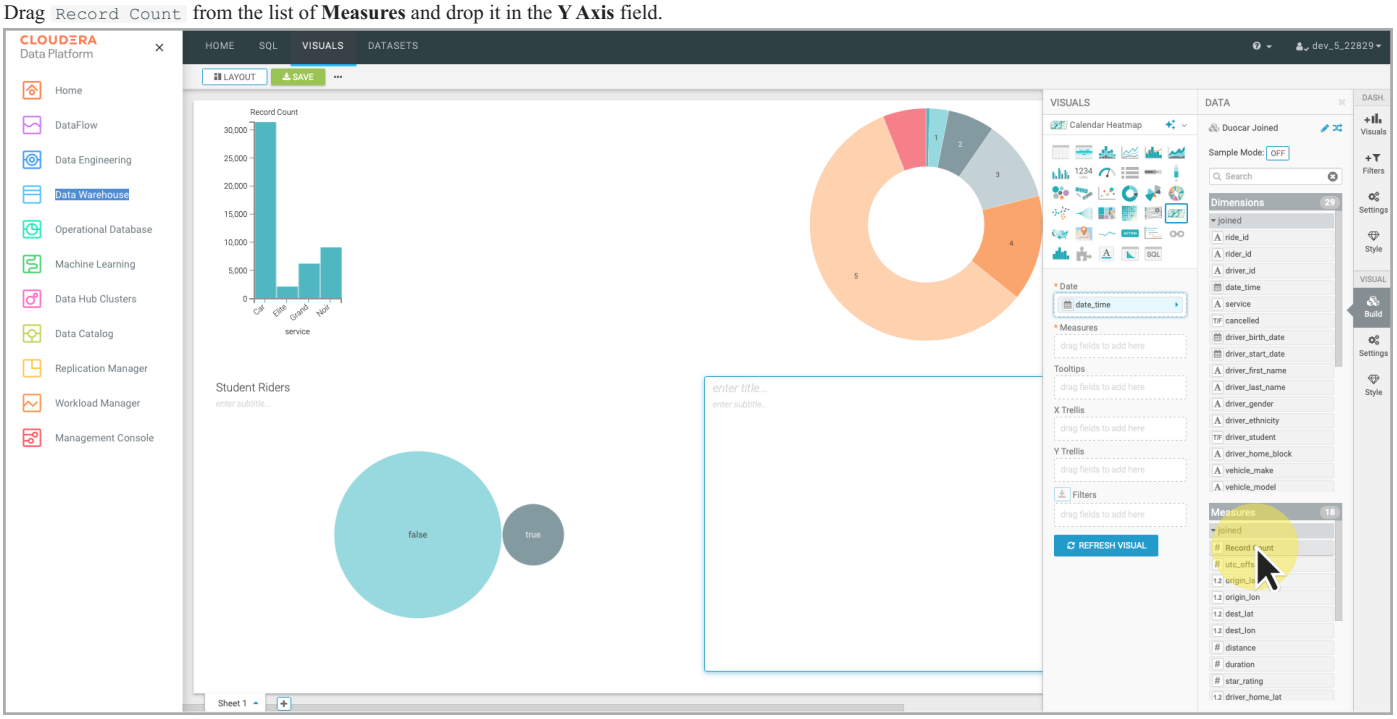
5. Click the **Bar Chart** icon. The bar chart properties are displayed.



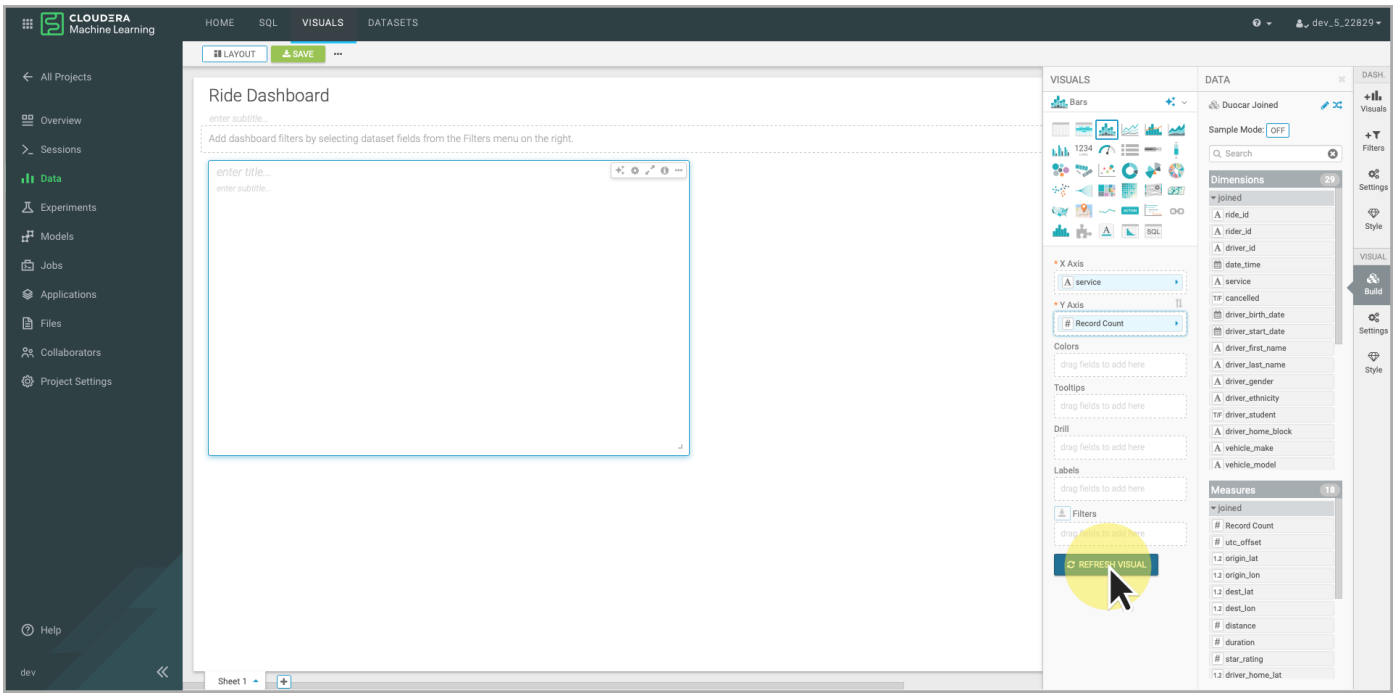
6. Drag `service` from the list of **Dimensions** and drop it in the **X Axis** field.



7. Drag `Record Count` from the list of **Measures** and drop it in the **Y Axis** field.



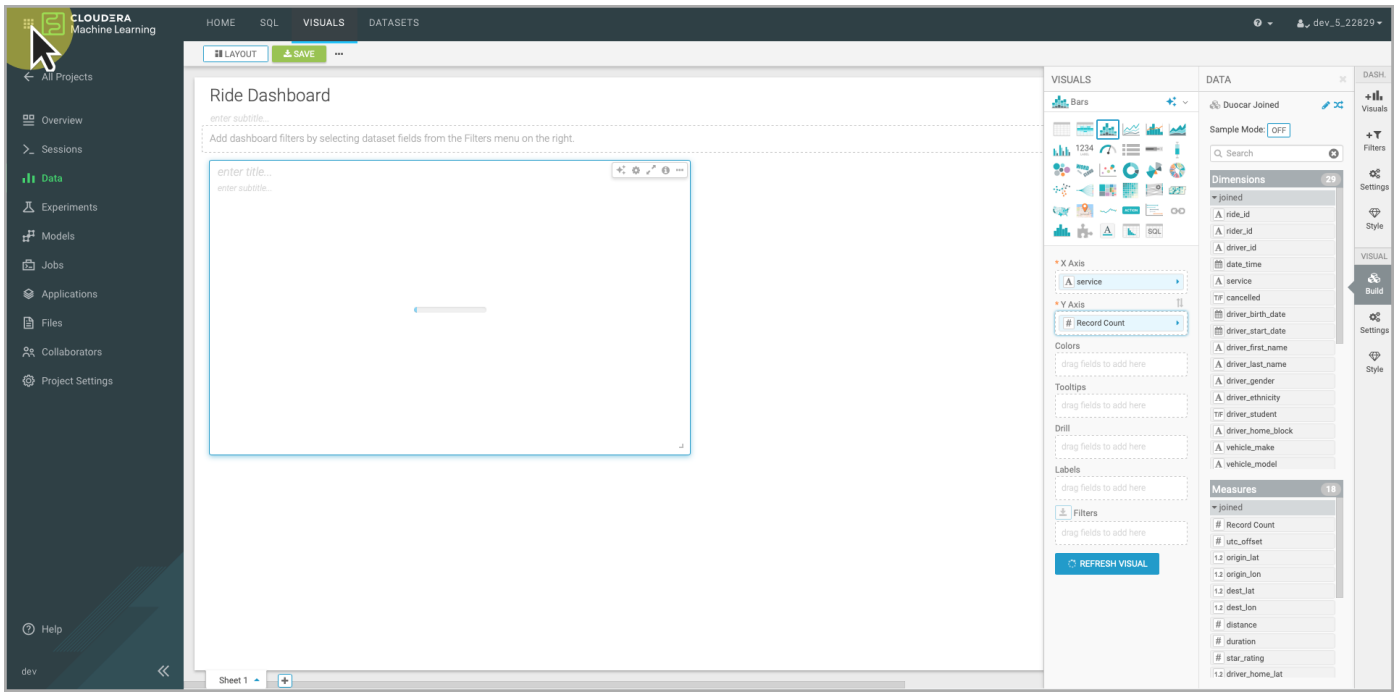
8. Click the **Refresh Visual** button.



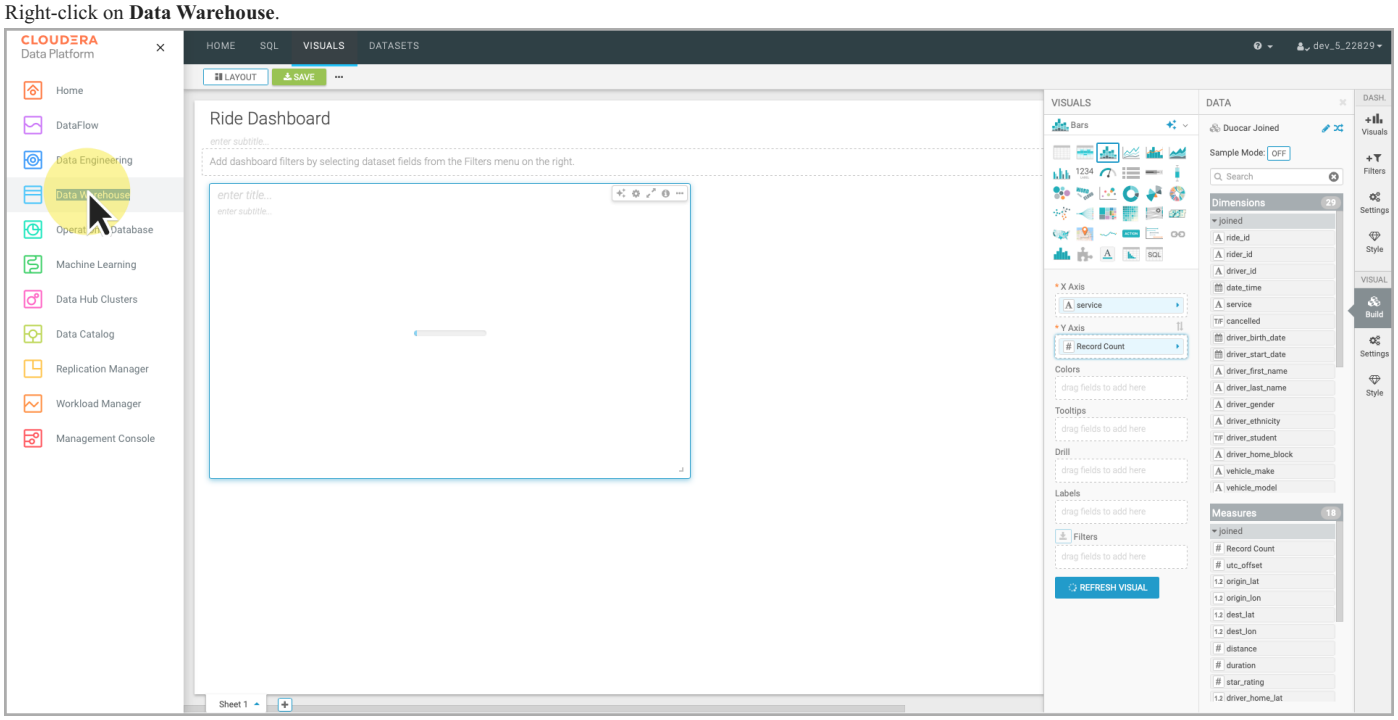
Note

The first time you make a connection to the data, the data warehouse may need start. This will cause the visual to take a long time to update as it is waiting for the data. The next five steps check the status of the data warehouse and watch as it auto provisions pods and enters a running state. If your visual updates quickly, you can skip these steps.

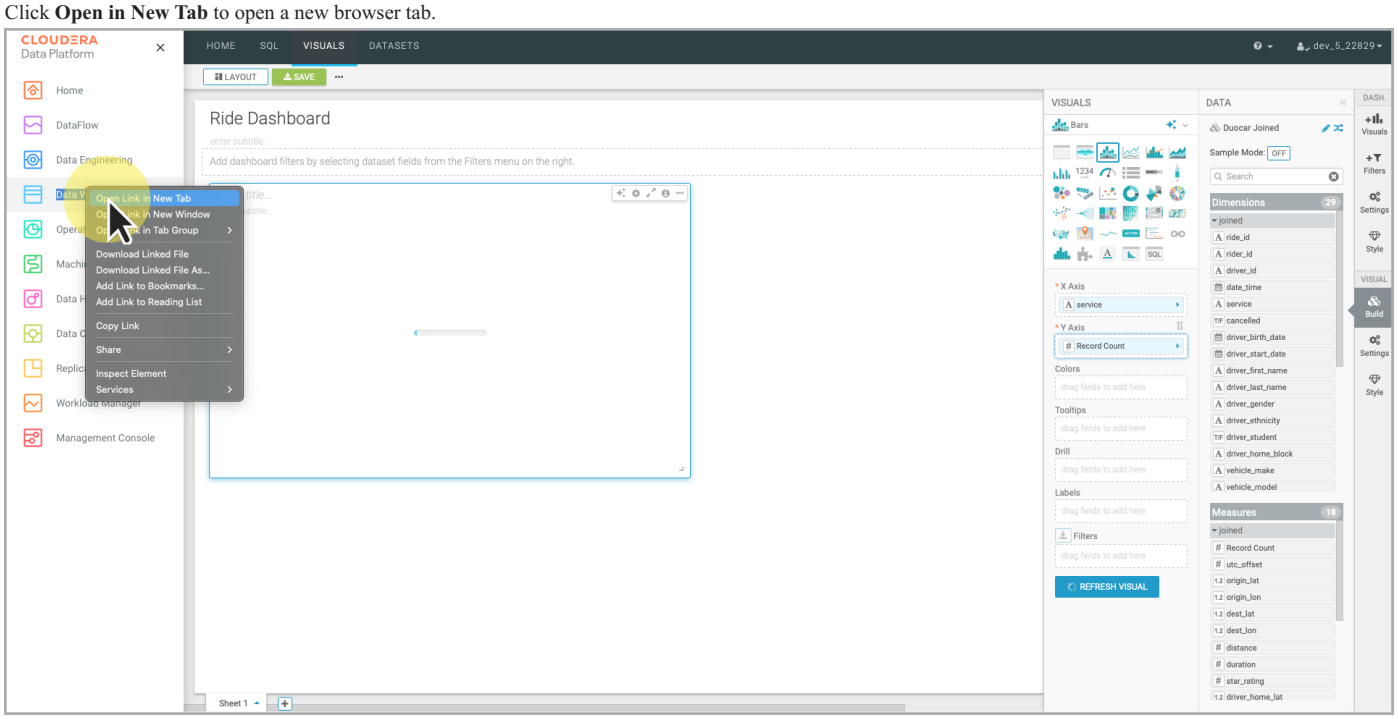
1. Click **Main Menu**.



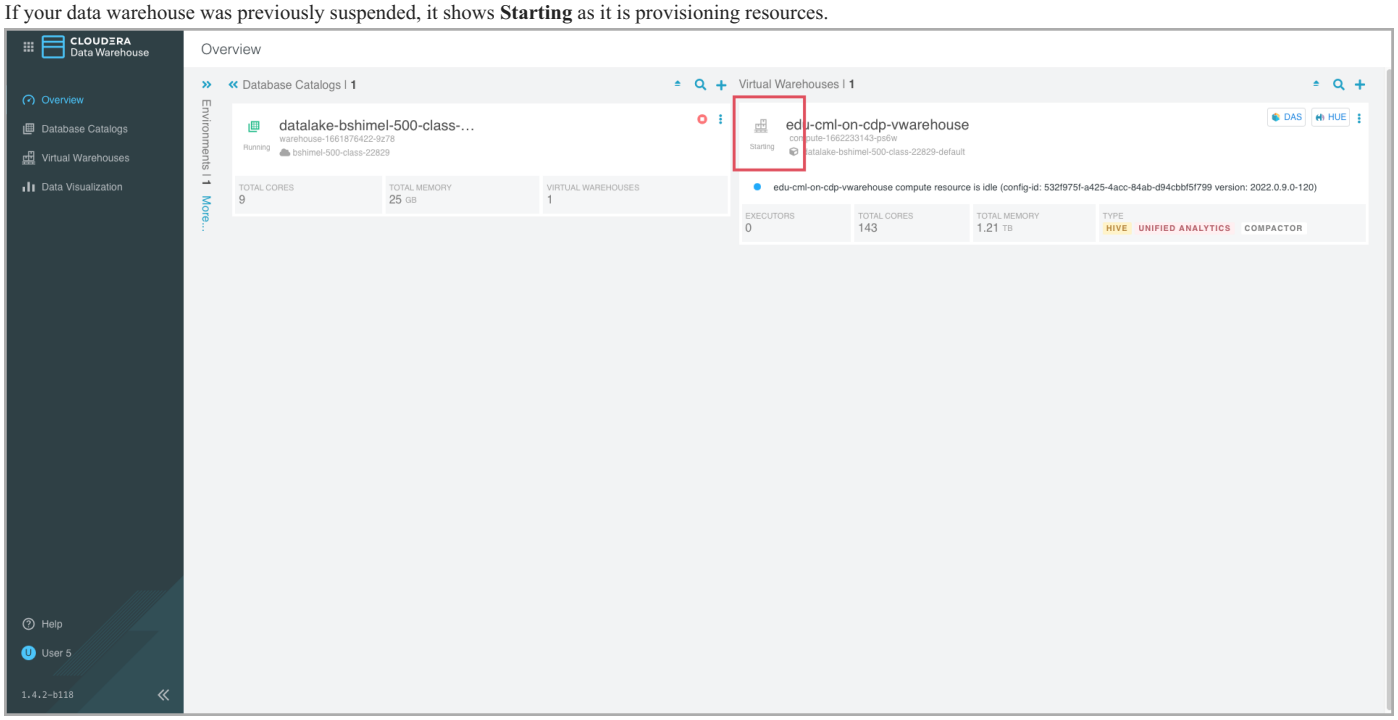
2. Right-click on **Data Warehouse**.



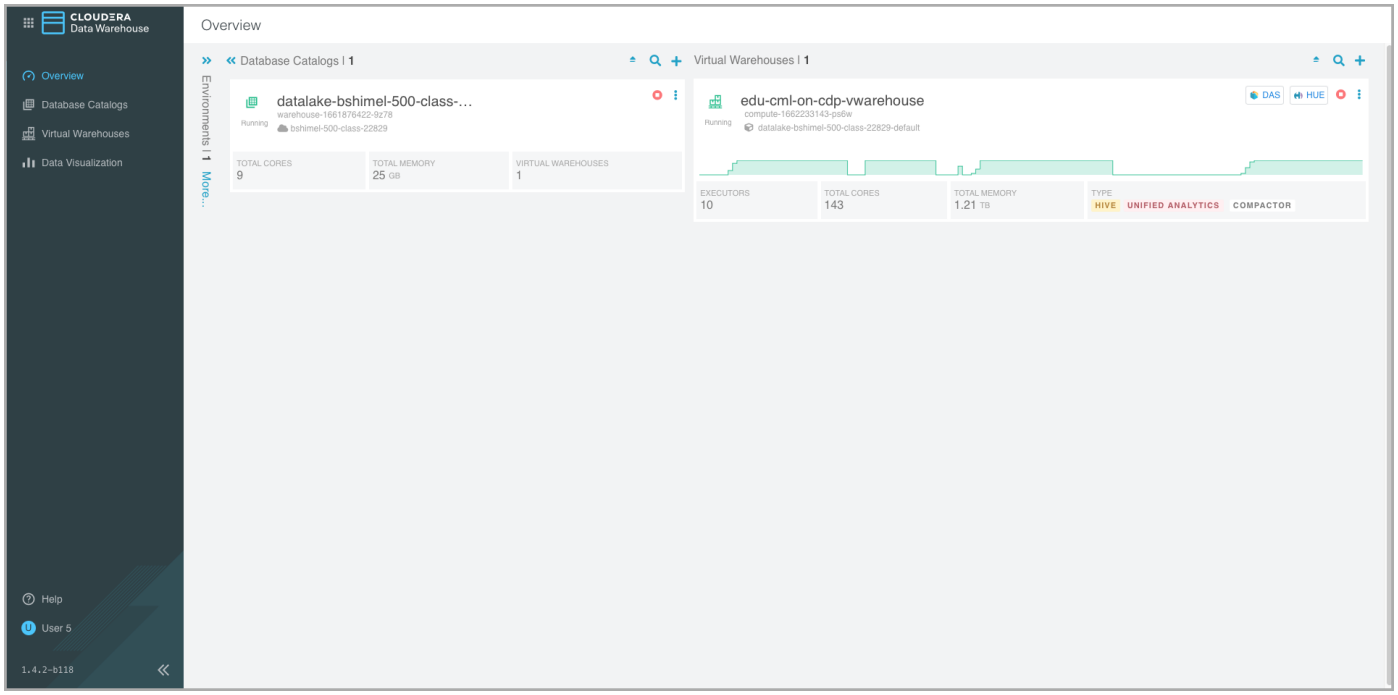
3. Click **Open in New Tab** to open a new browser tab.



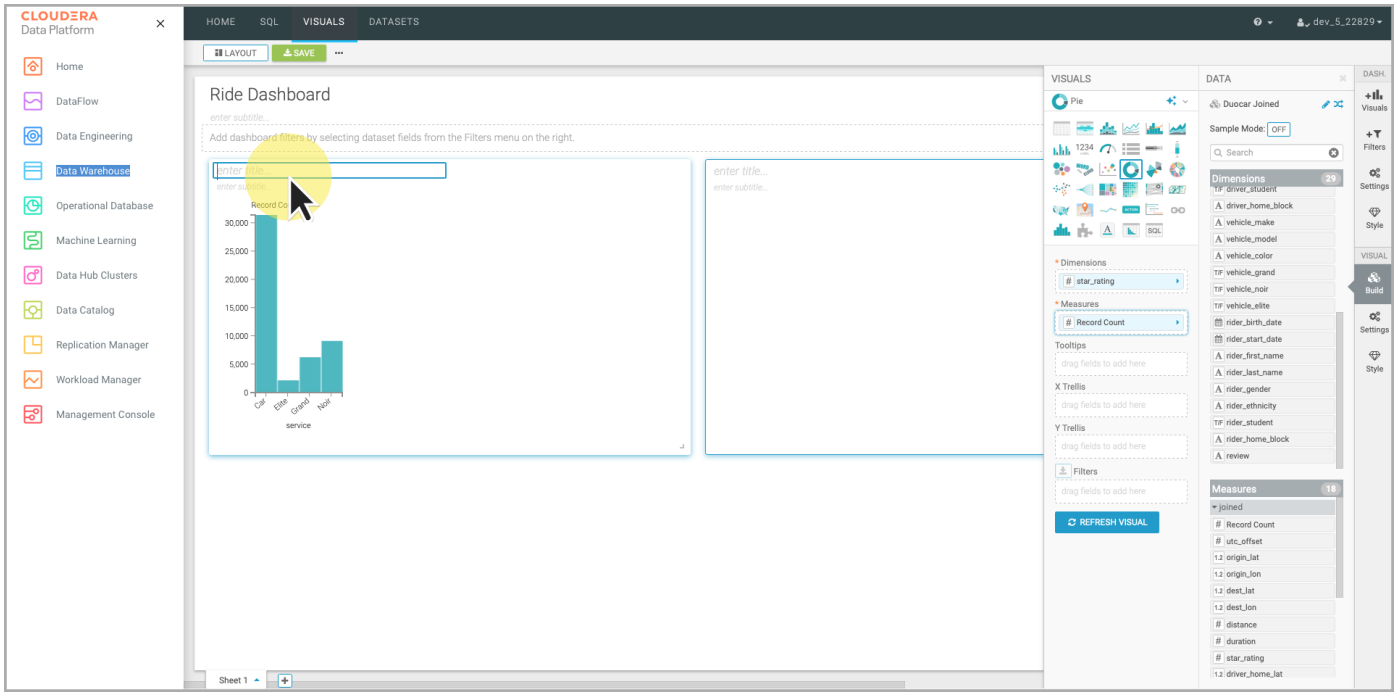
4. If your data warehouse was previously suspended, it shows **Starting** as it is provisioning resources.



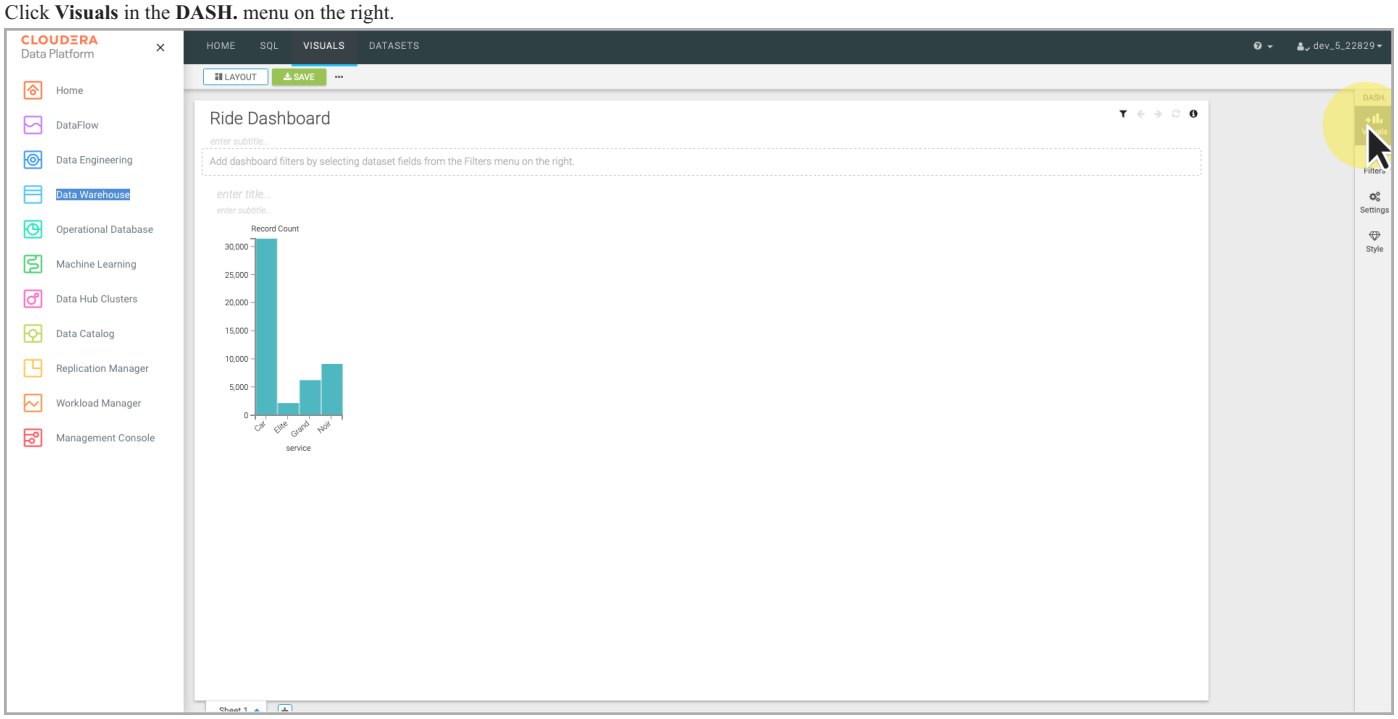
5. Once the data warehouse has provisioned its resources, the state will change to **Running**. Close the browser tab and return to the tab with the **Data Discovery and Visualization** application.



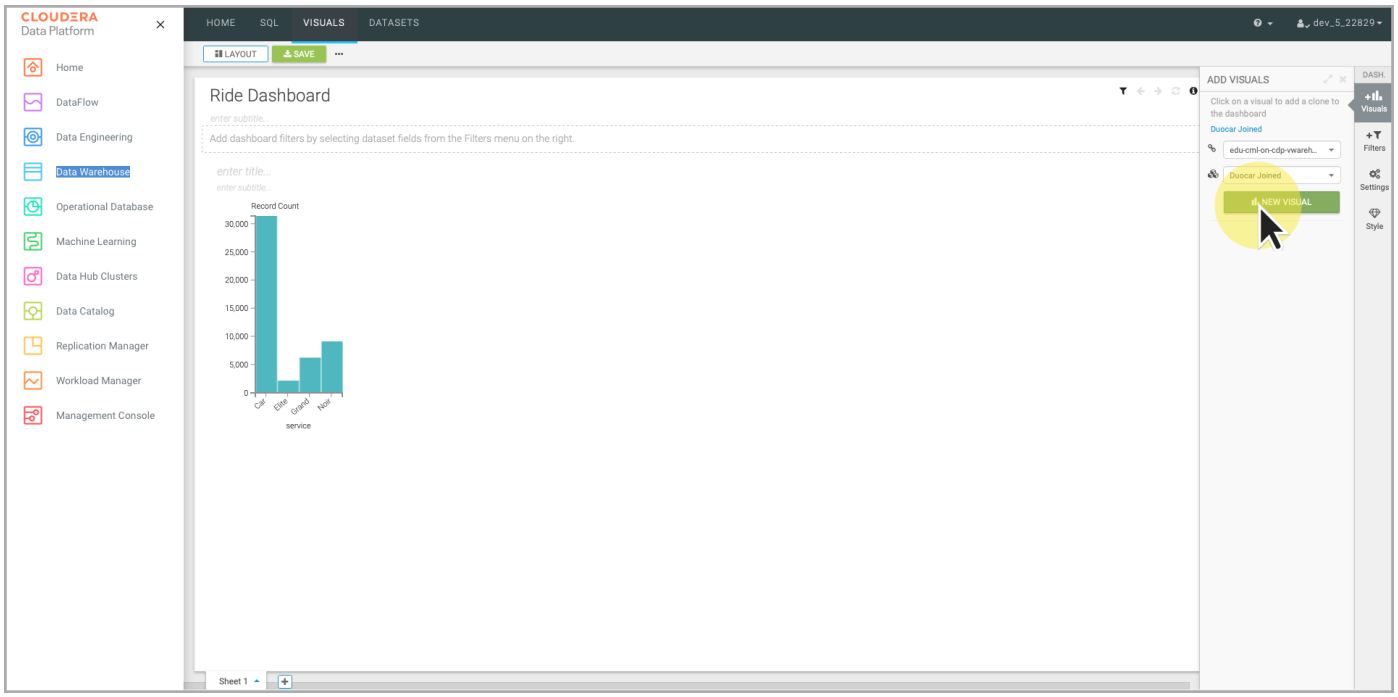
6. Enter `Rides by Service` as the title for the bar chart visual.



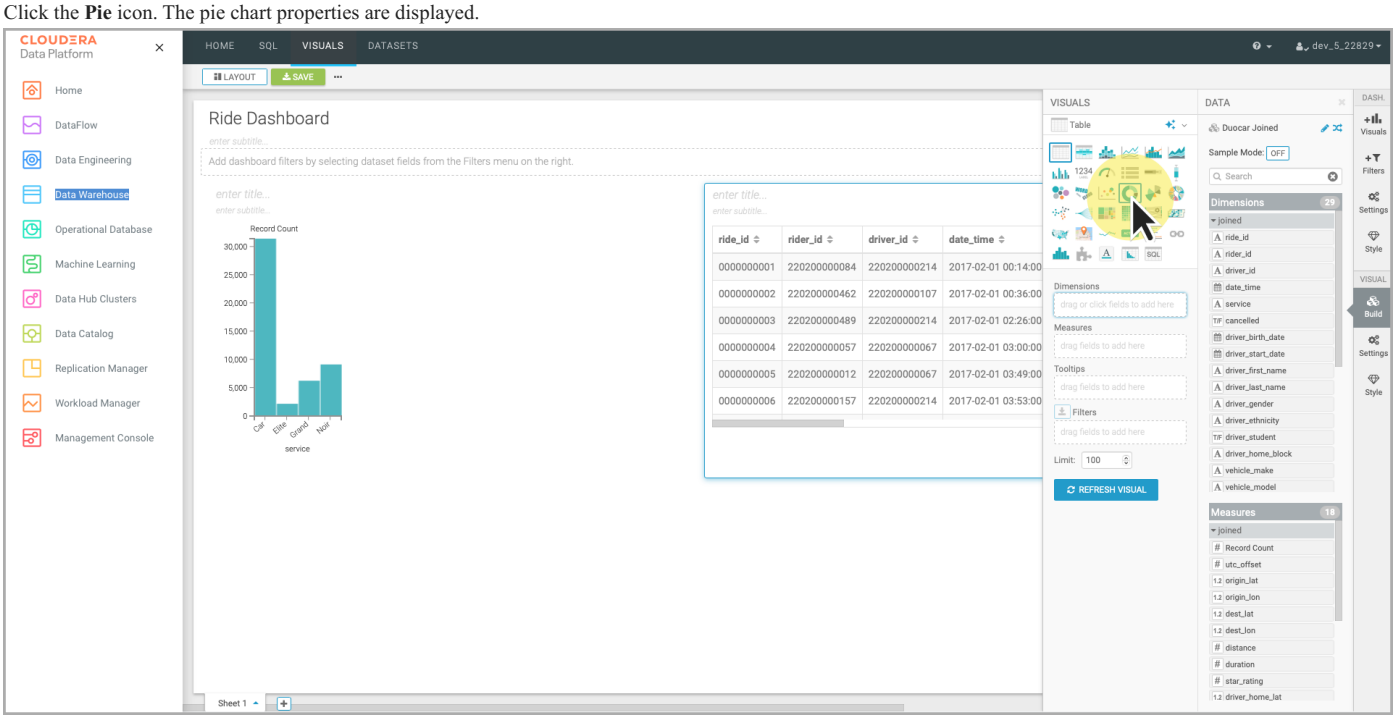
7. Click **Visuals** in the **DASH.** menu on the right.



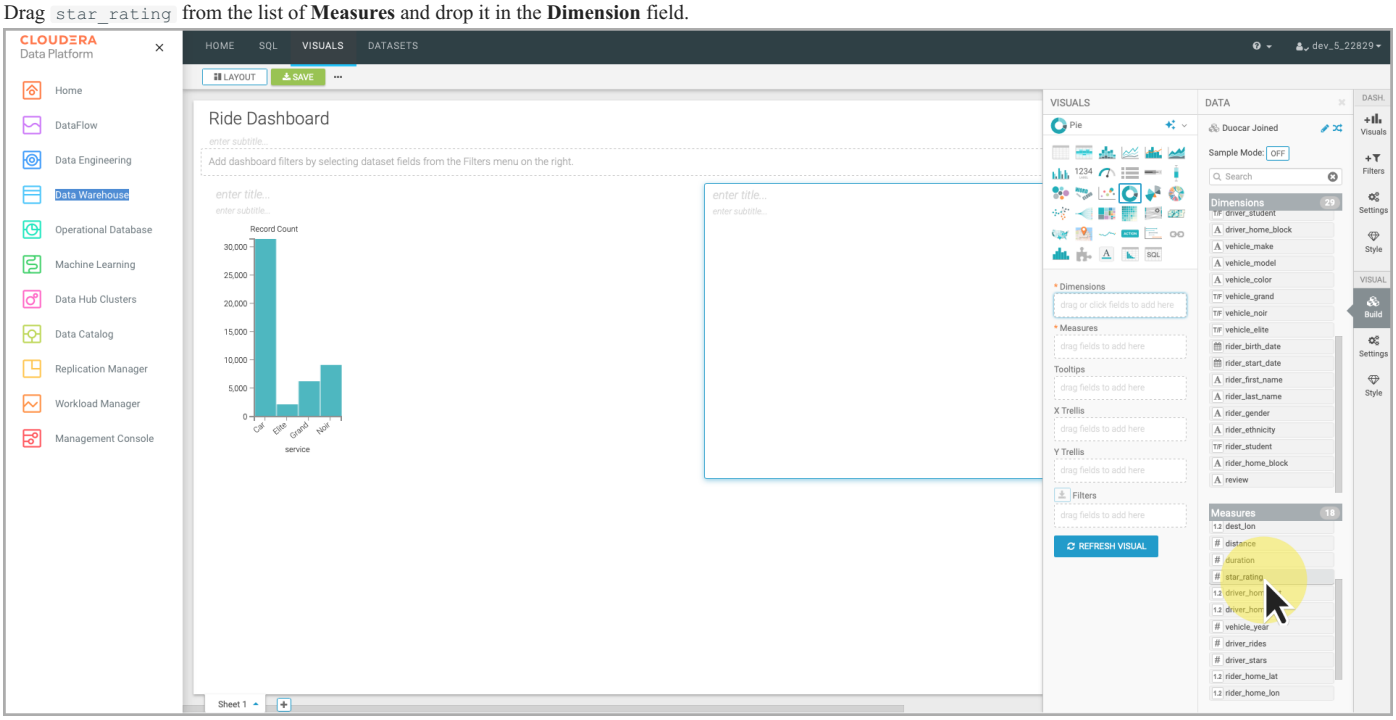
8. Click the **New Visual** button.



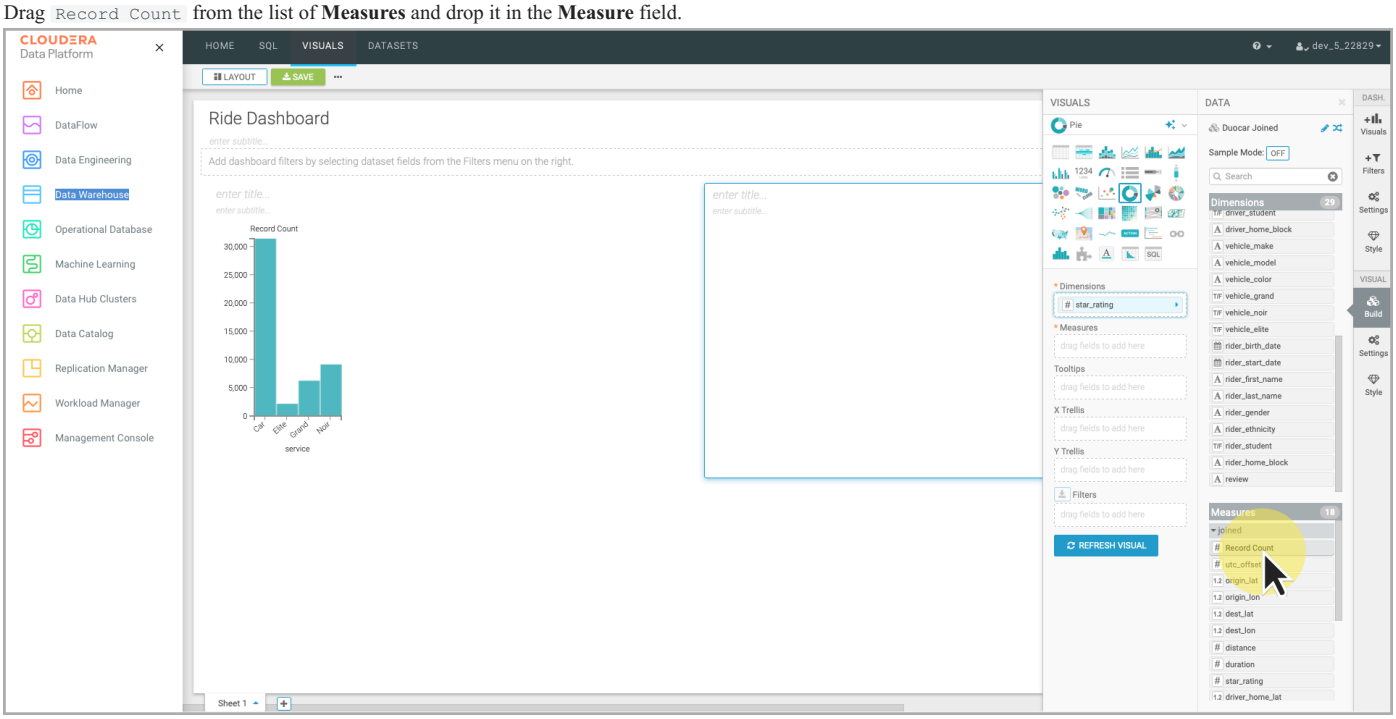
9. Click the **Pie** icon. The pie chart properties are displayed.



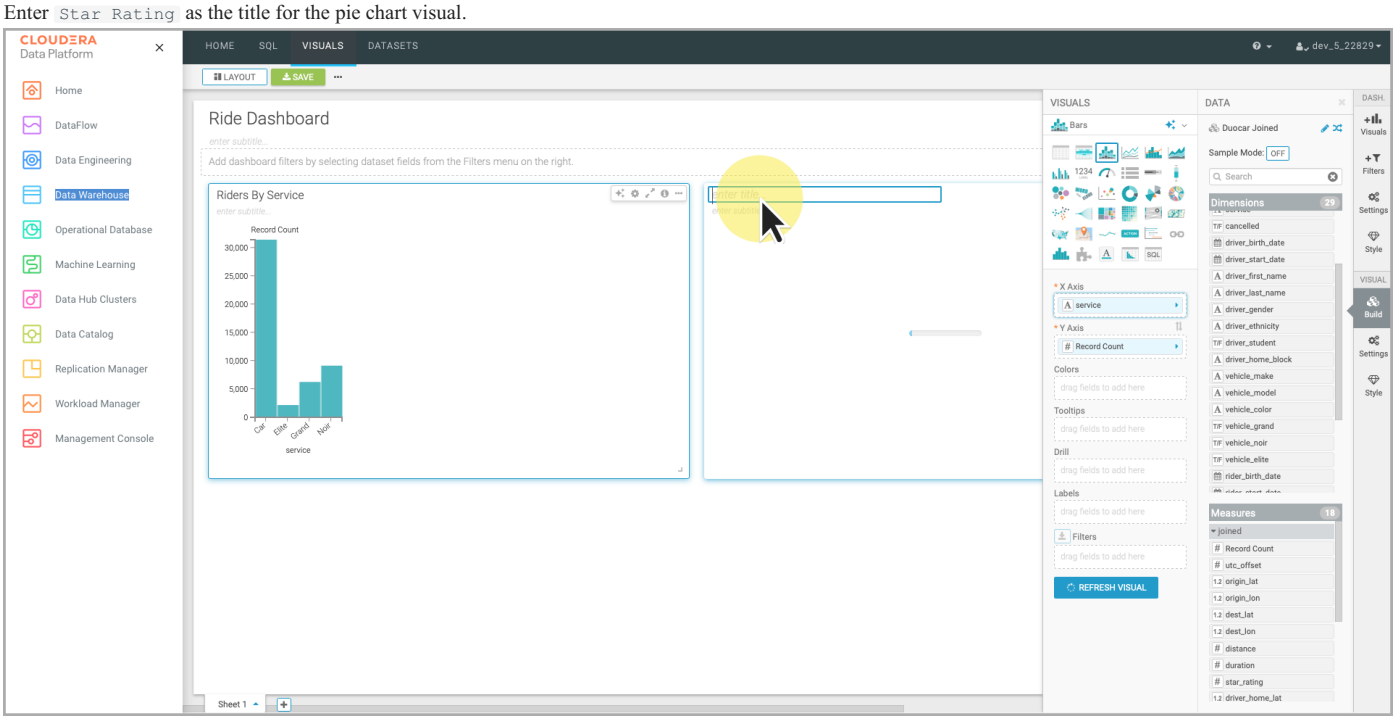
10. Drag `star_rating` from the list of **Measures** and drop it in the **Dimension** field.



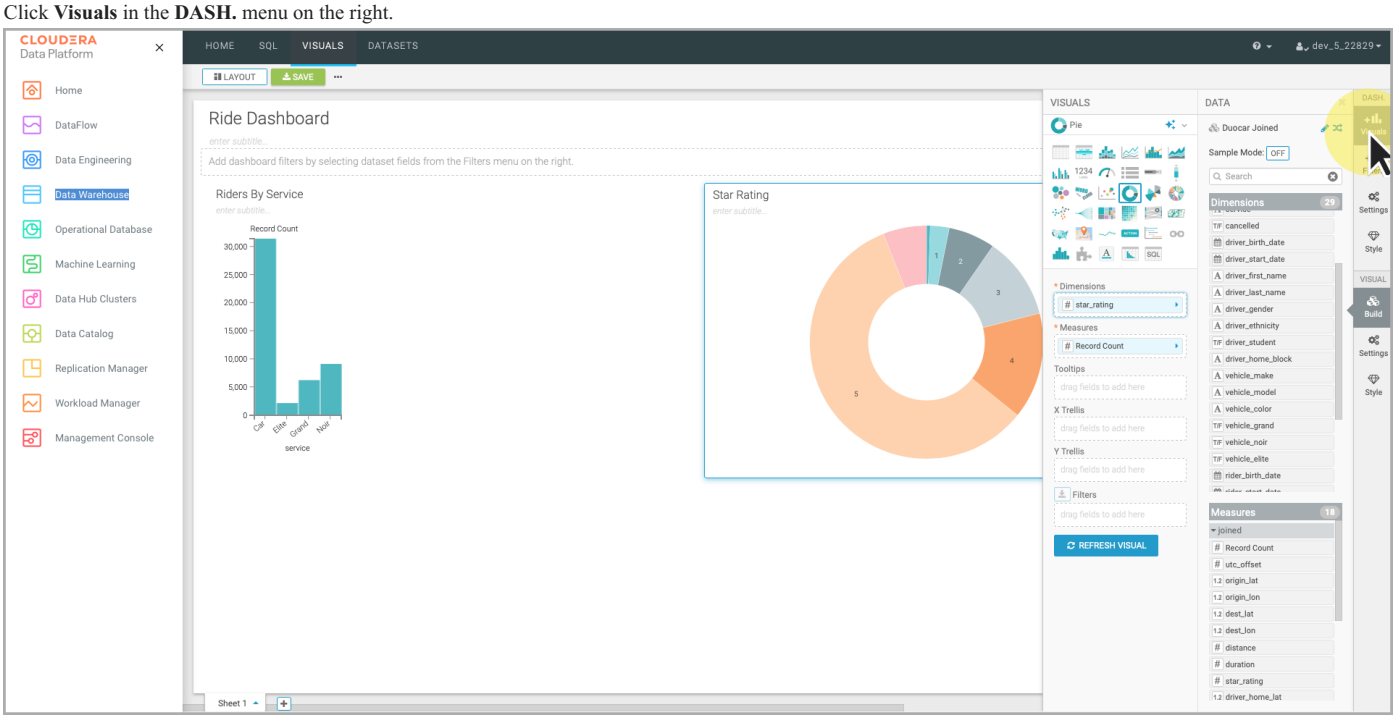
11. Drag `Record Count` from the list of **Measures** and drop it in the **Measure** field.



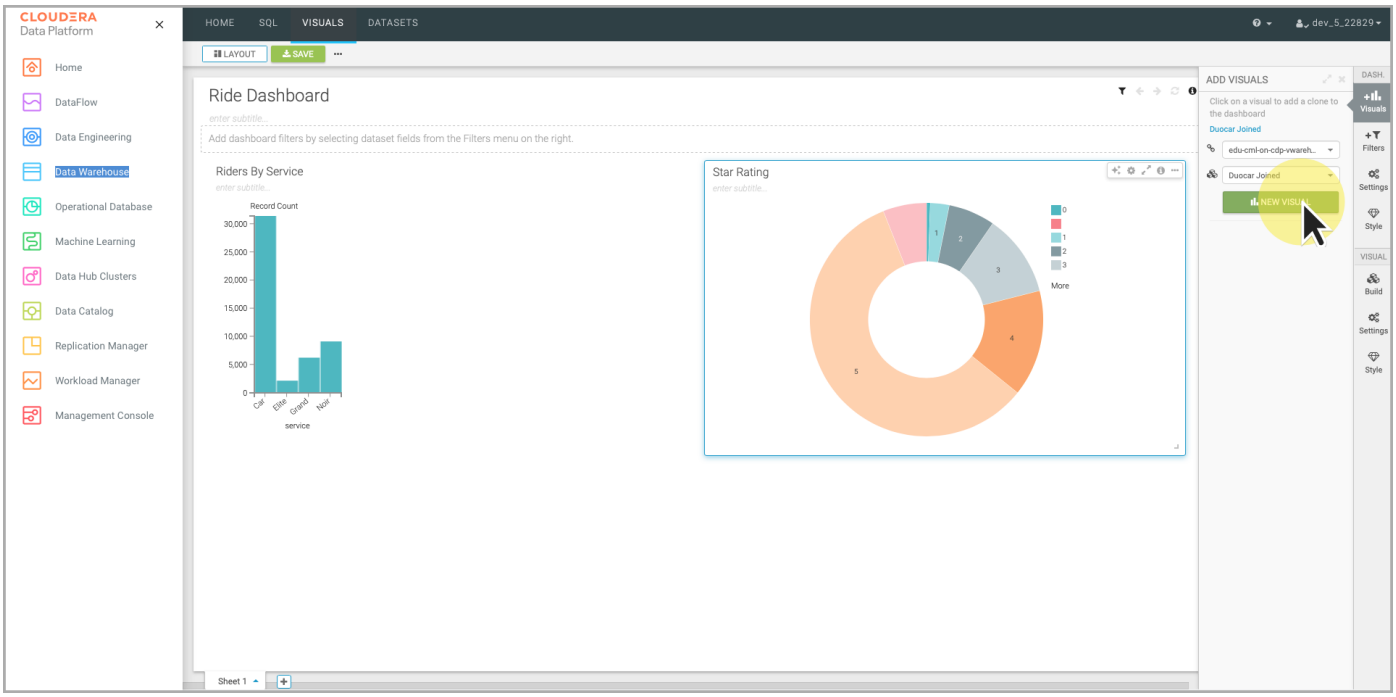
12. Enter `Star Rating` as the title for the pie chart visual.



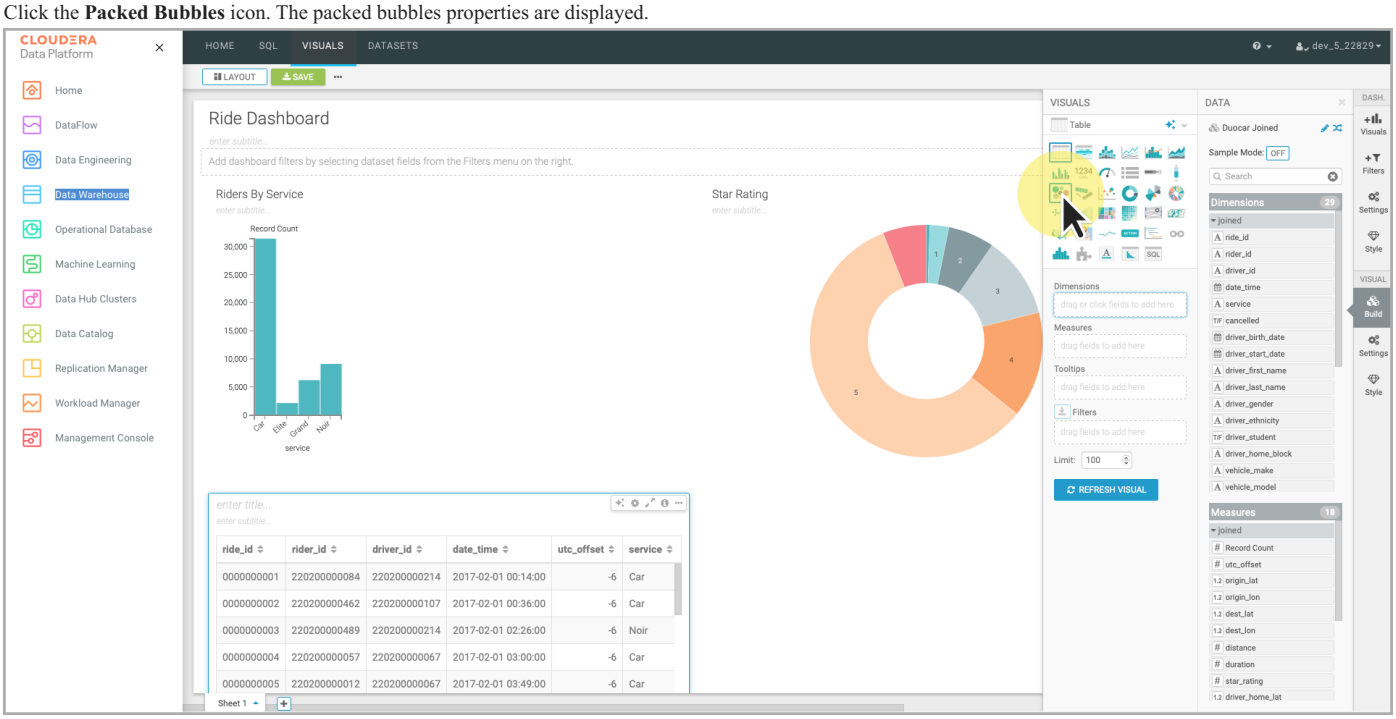
13. Click **Visuals** in the **DASH.** menu on the right.



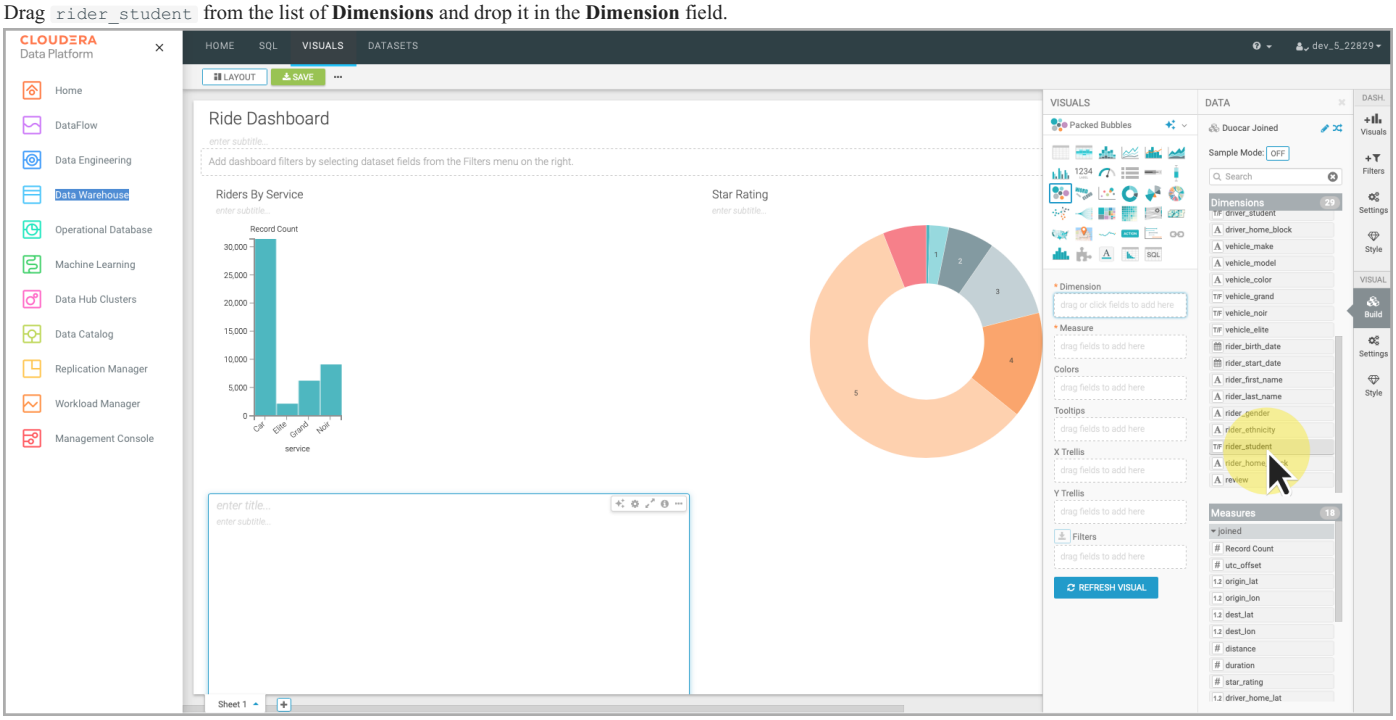
14. Click the **New Visual** button.



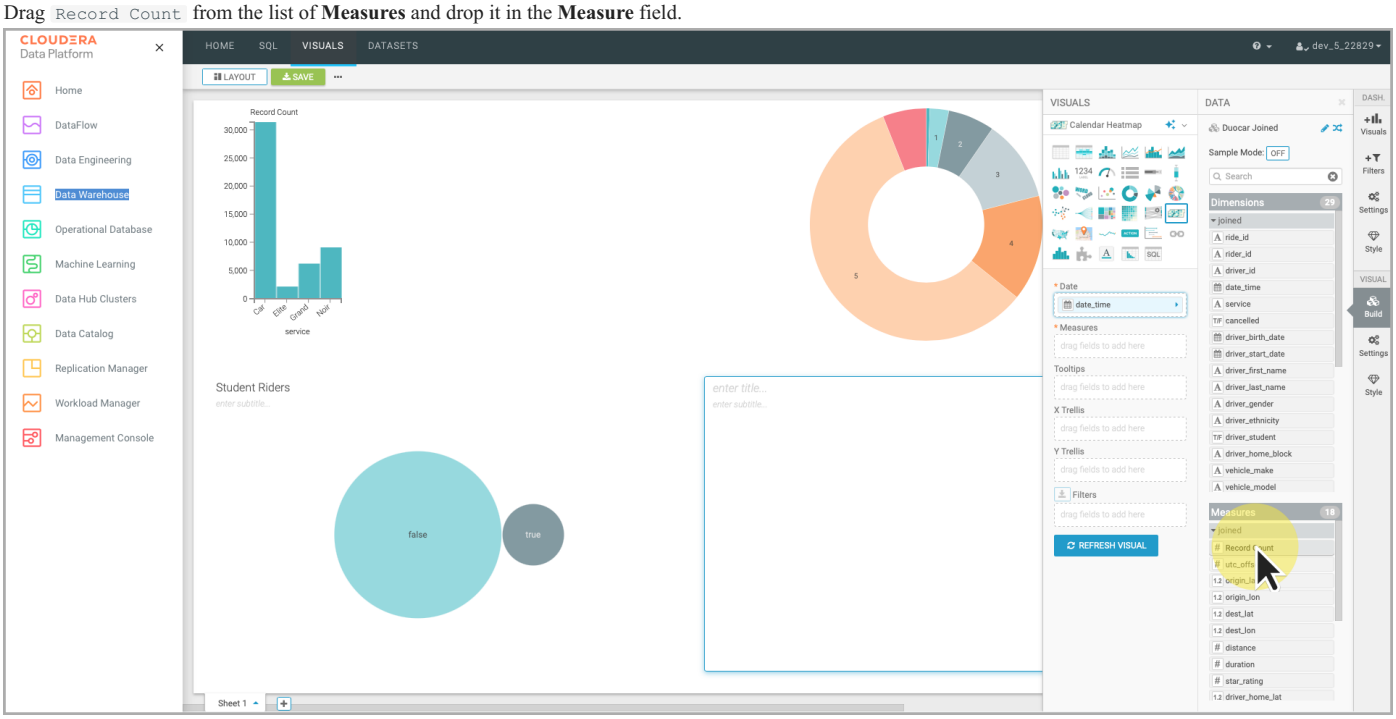
15. Click the **Packed Bubbles** icon. The packed bubbles properties are displayed.



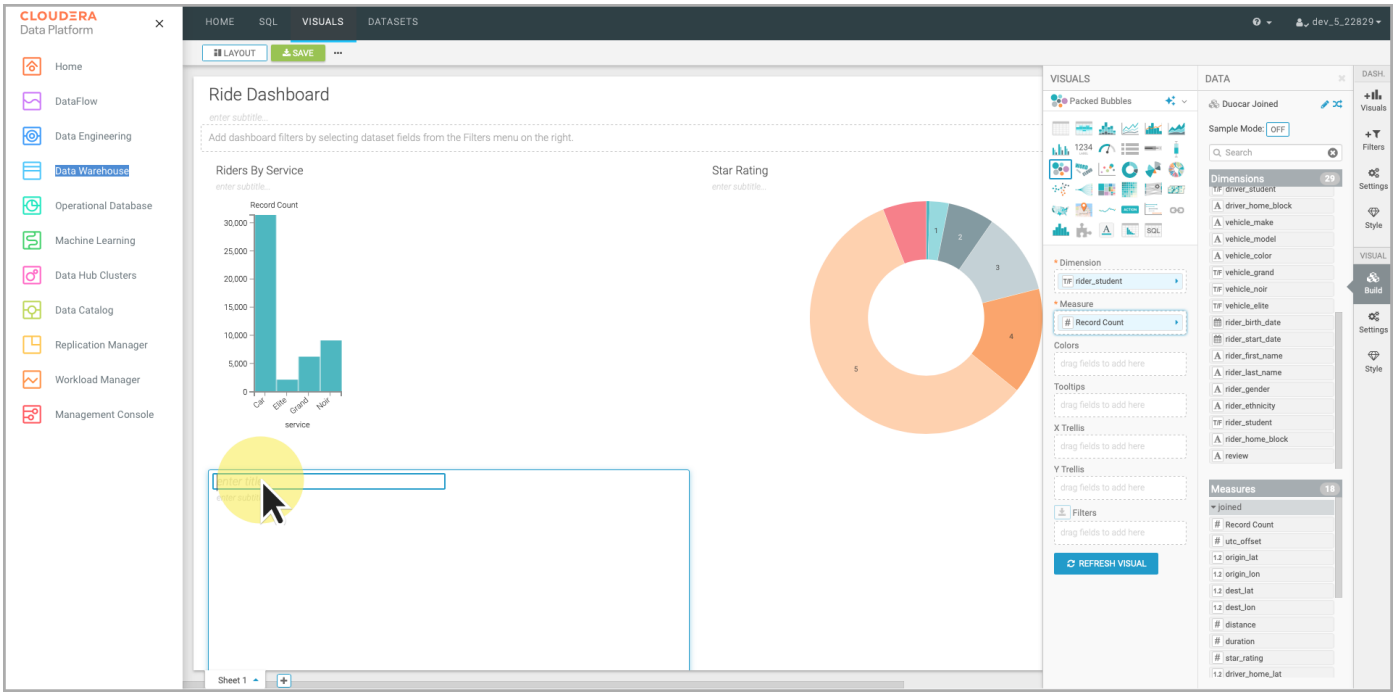
16. Drag `rider_student` from the list of **Dimensions** and drop it in the **Dimension** field.



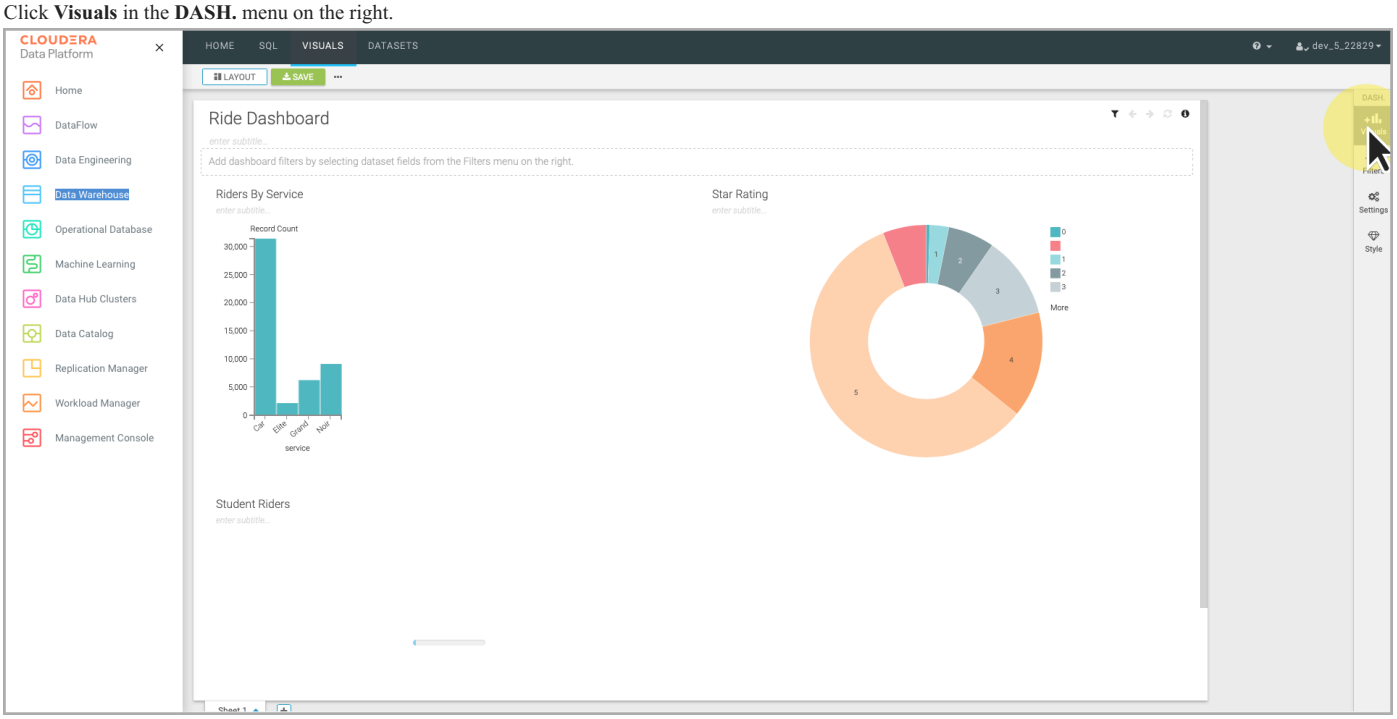
17. Drag `Record Count` from the list of **Measures** and drop it in the **Measure** field.



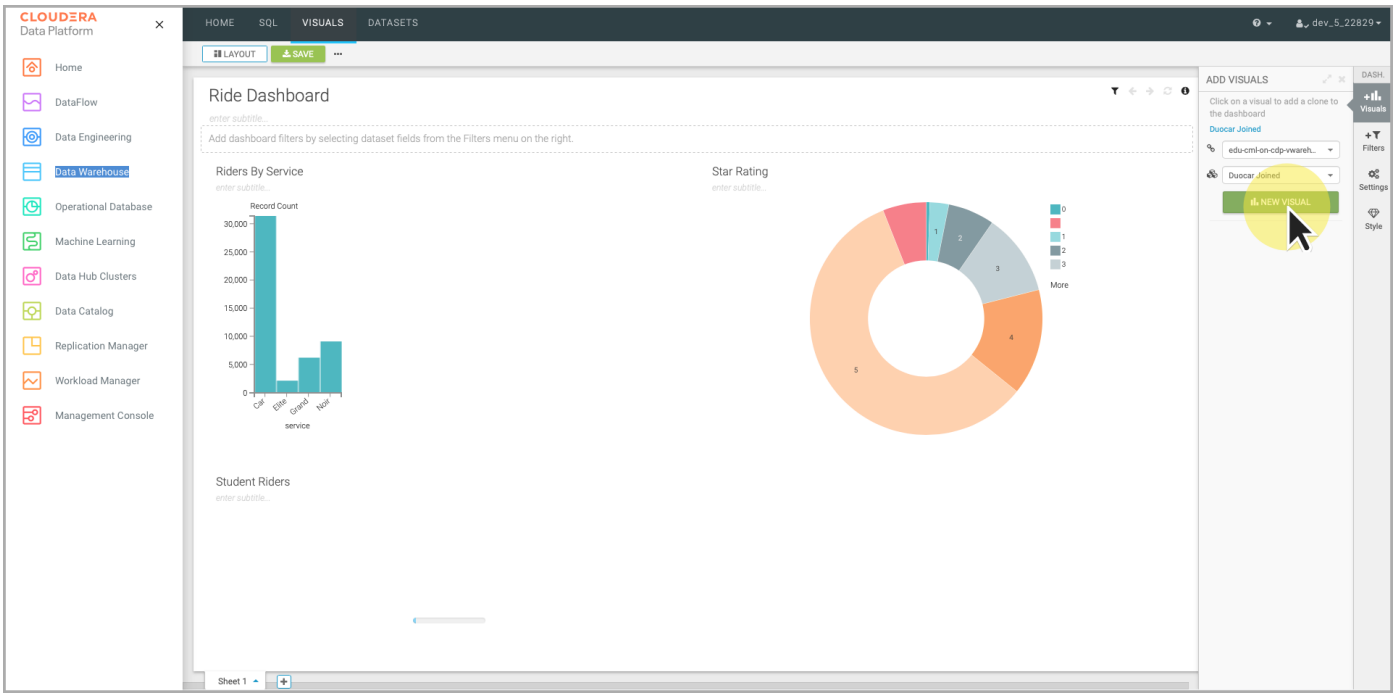
18. Enter `Student Riders` as the title for the packed bubbles visual.



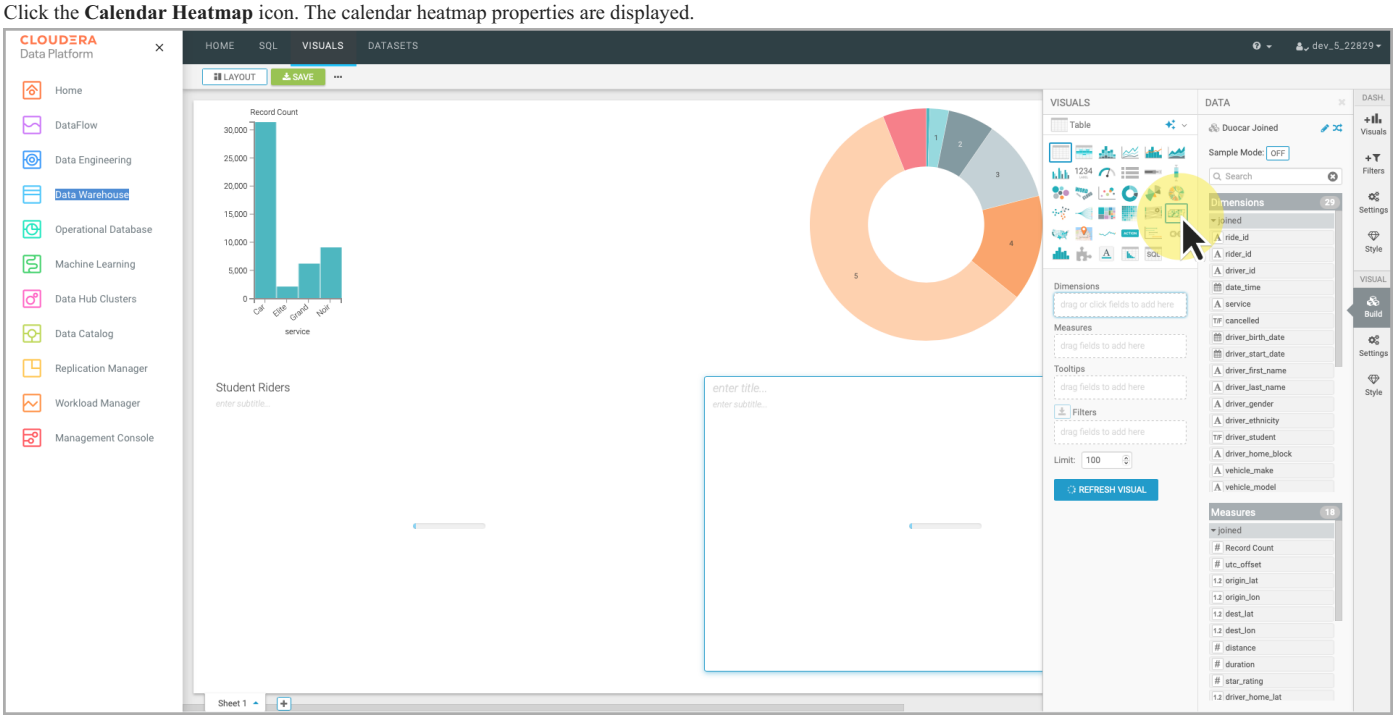
19. Click **Visuals** in the **DASH.** menu on the right.



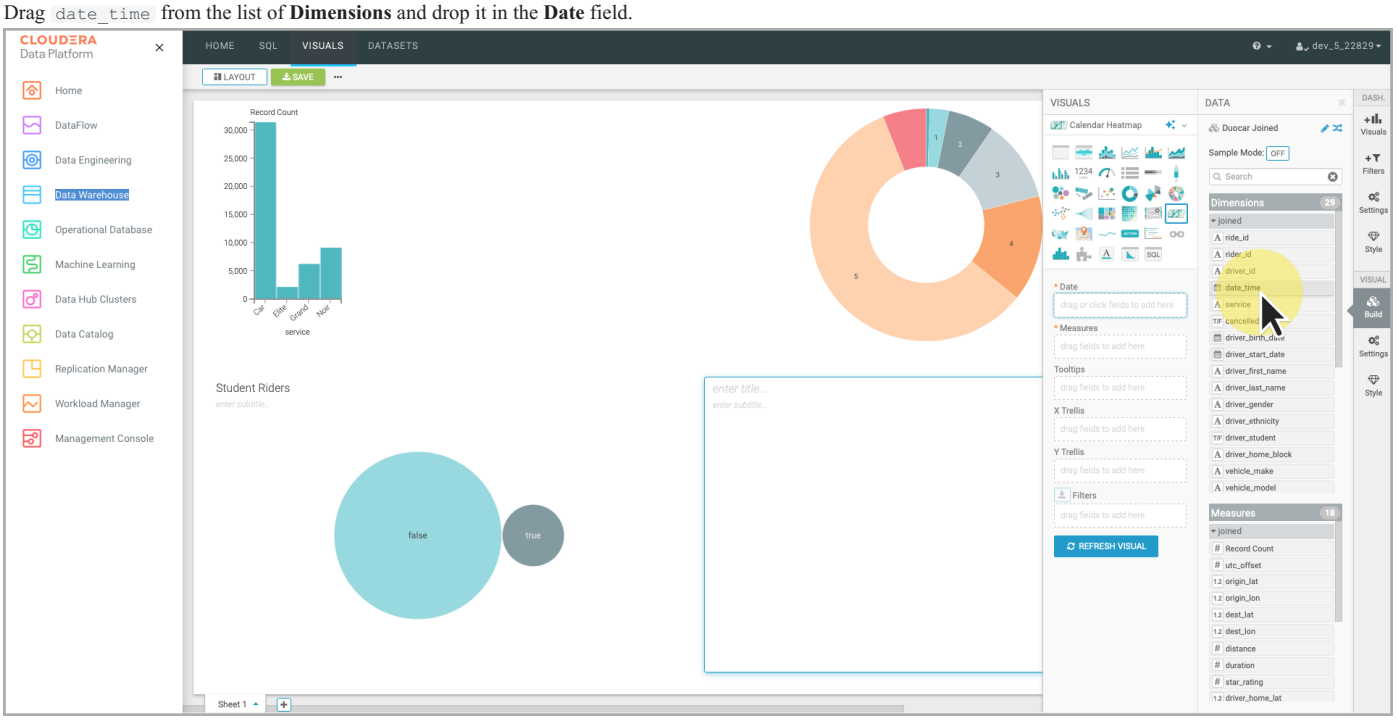
20. Click the **New Visual** button.



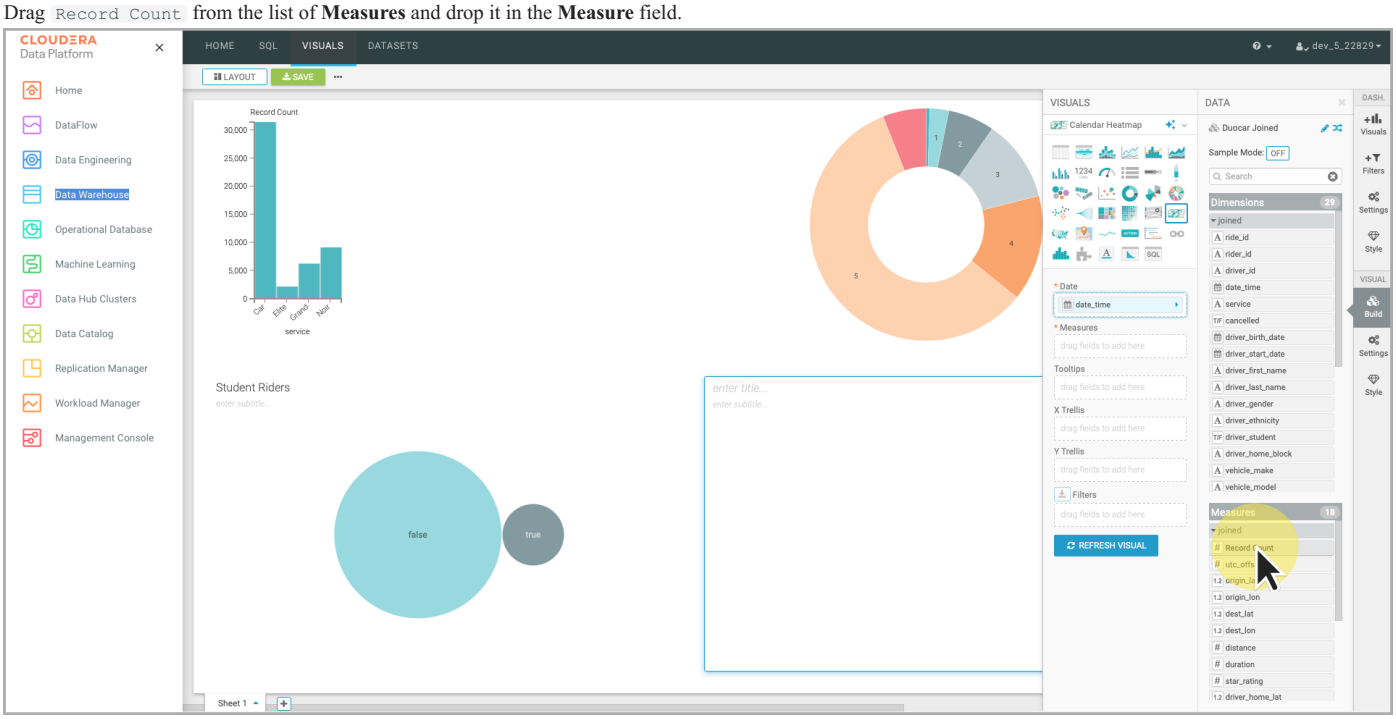
21. Click the **Calendar Heatmap** icon. The calendar heatmap properties are displayed.



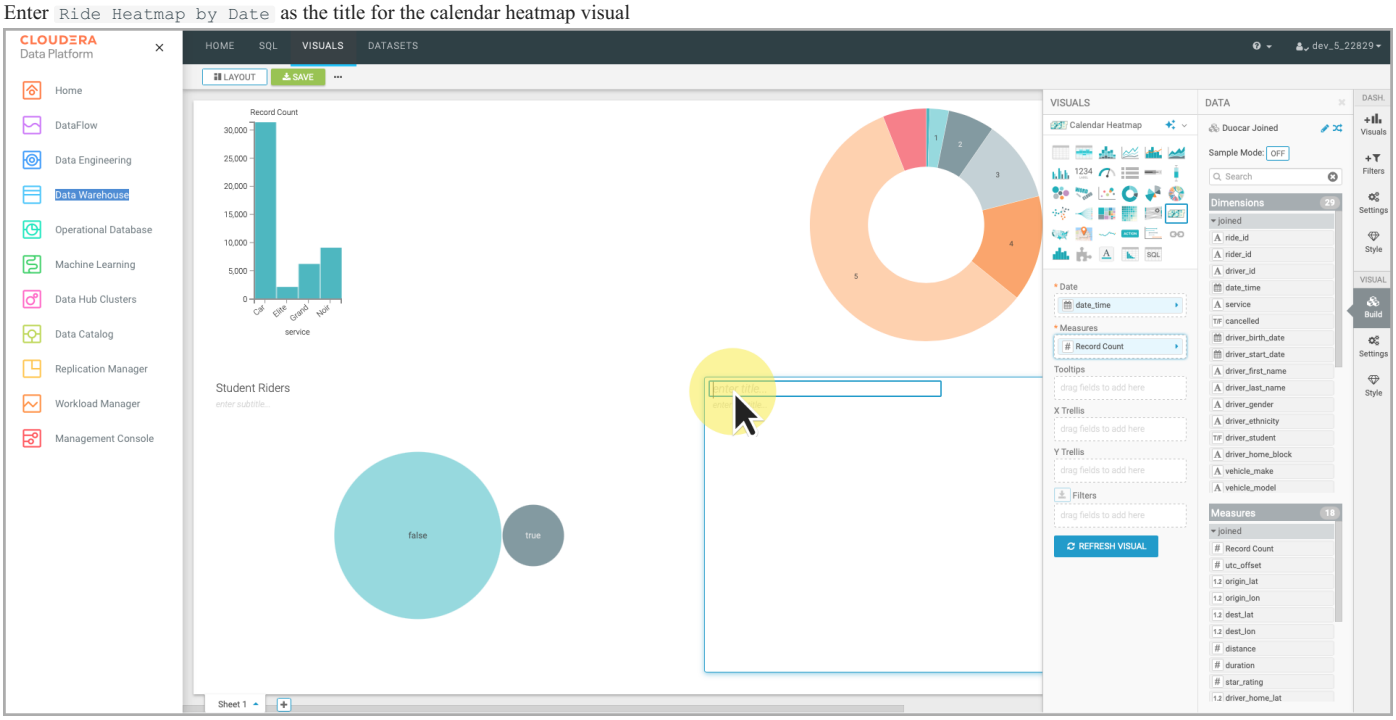
22. Drag `date_time` from the list of **Dimensions** and drop it in the **Date** field.



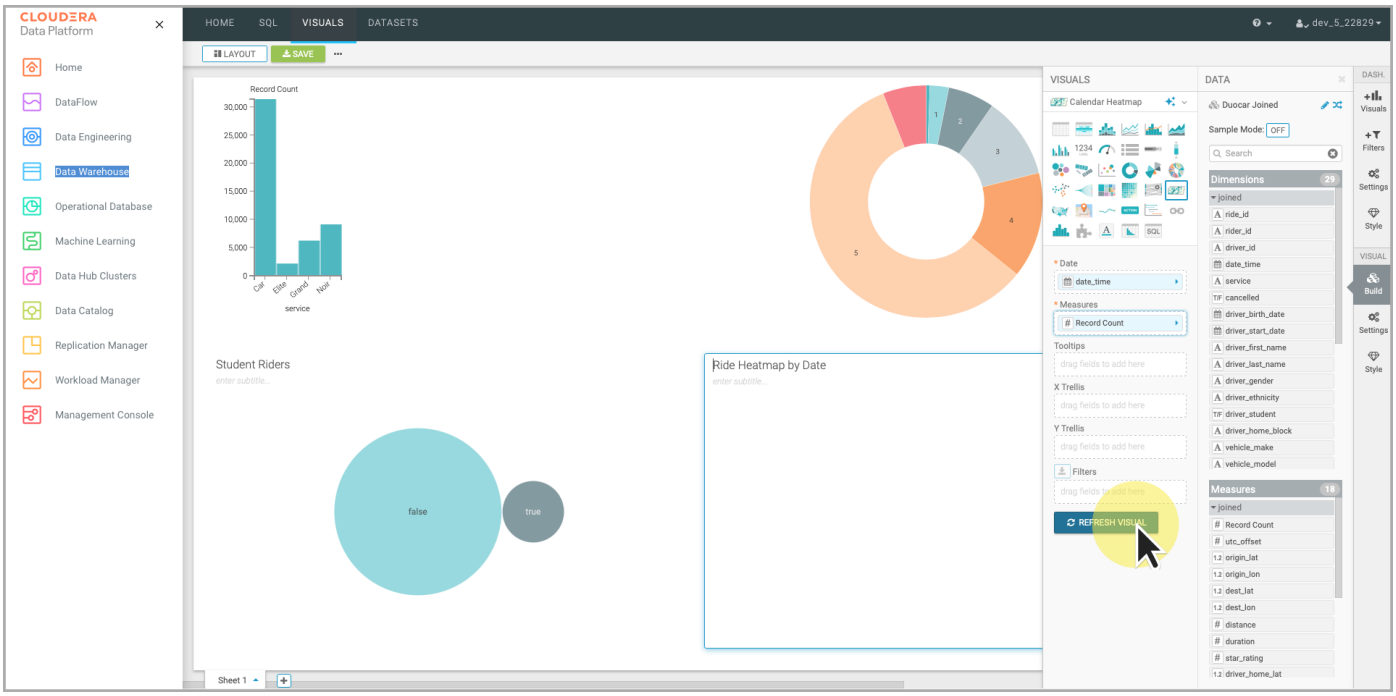
23. Drag `Record Count` from the list of **Measures** and drop it in the **Measure** field.



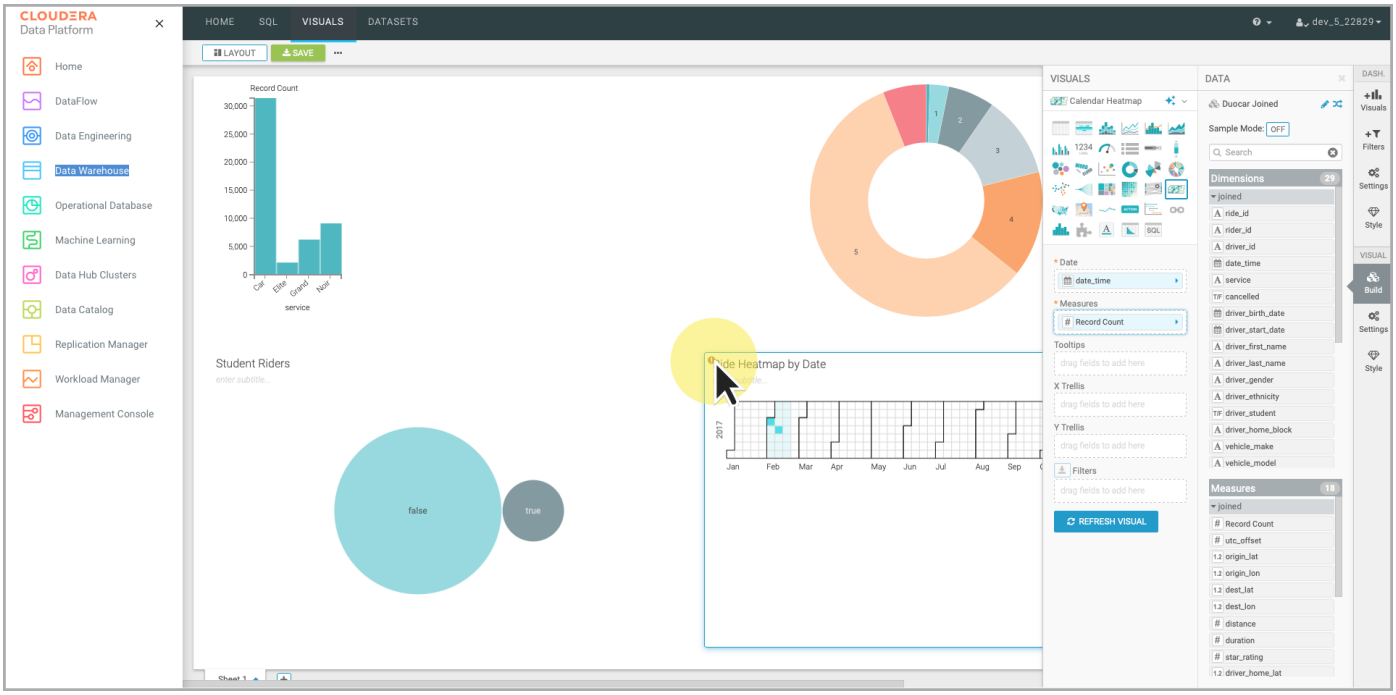
24. Enter `Ride Heatmap by Date` as the title for the calendar heatmap visual



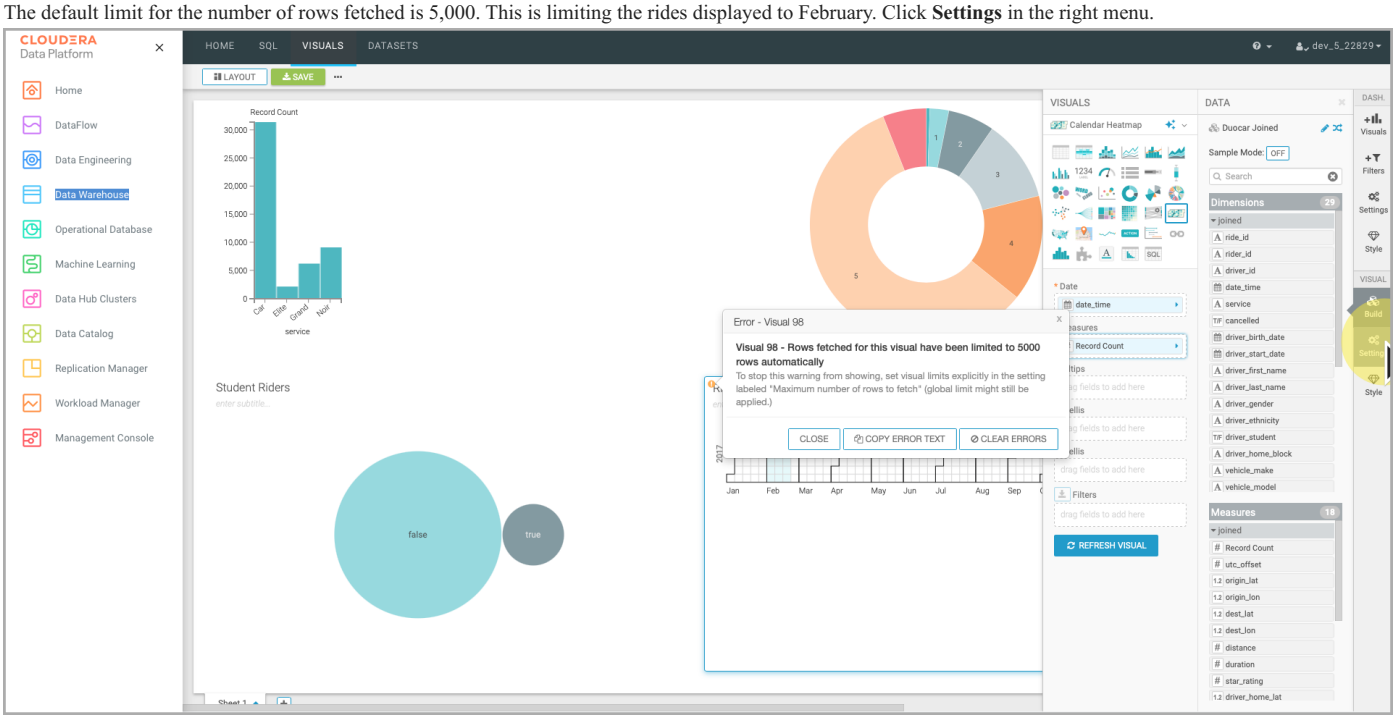
25. Click **Refresh Visual**.



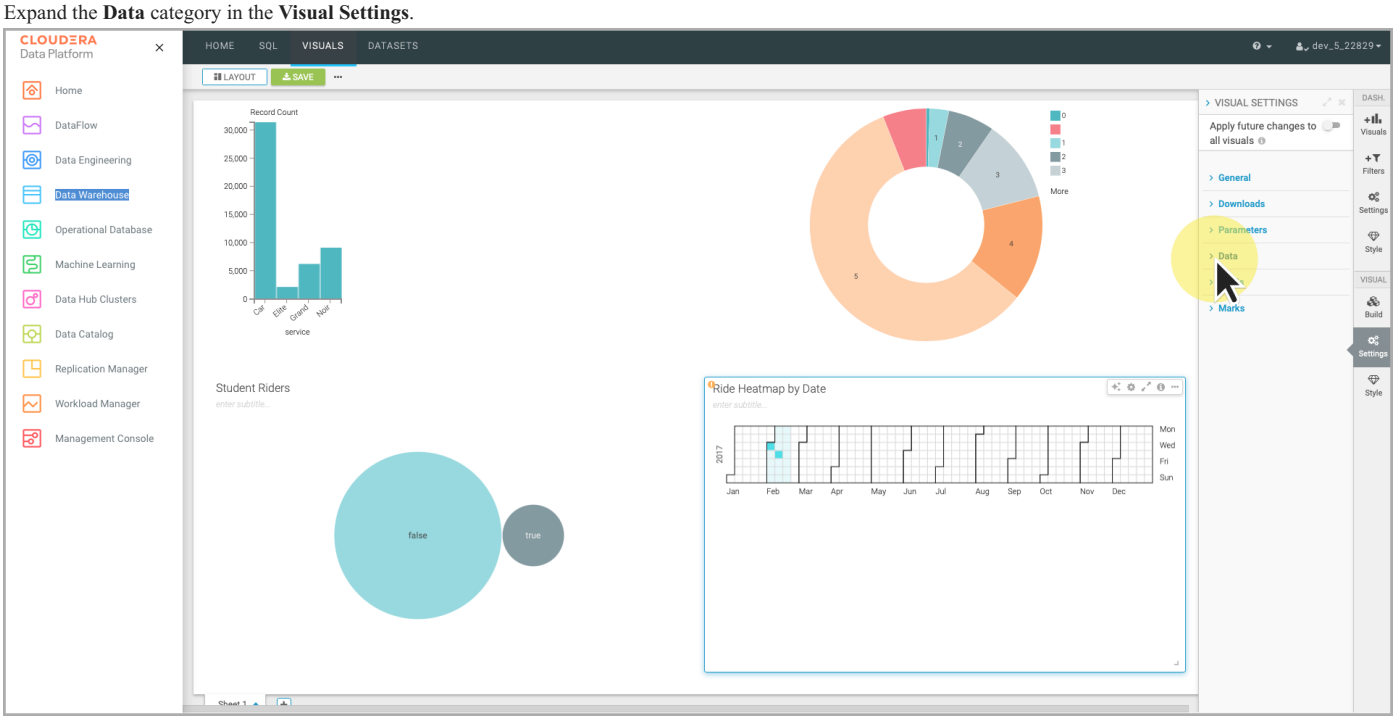
26. Notice, there is an ⓘ icon displayed on the visual and only February has rides. Click the ⓘ icon.



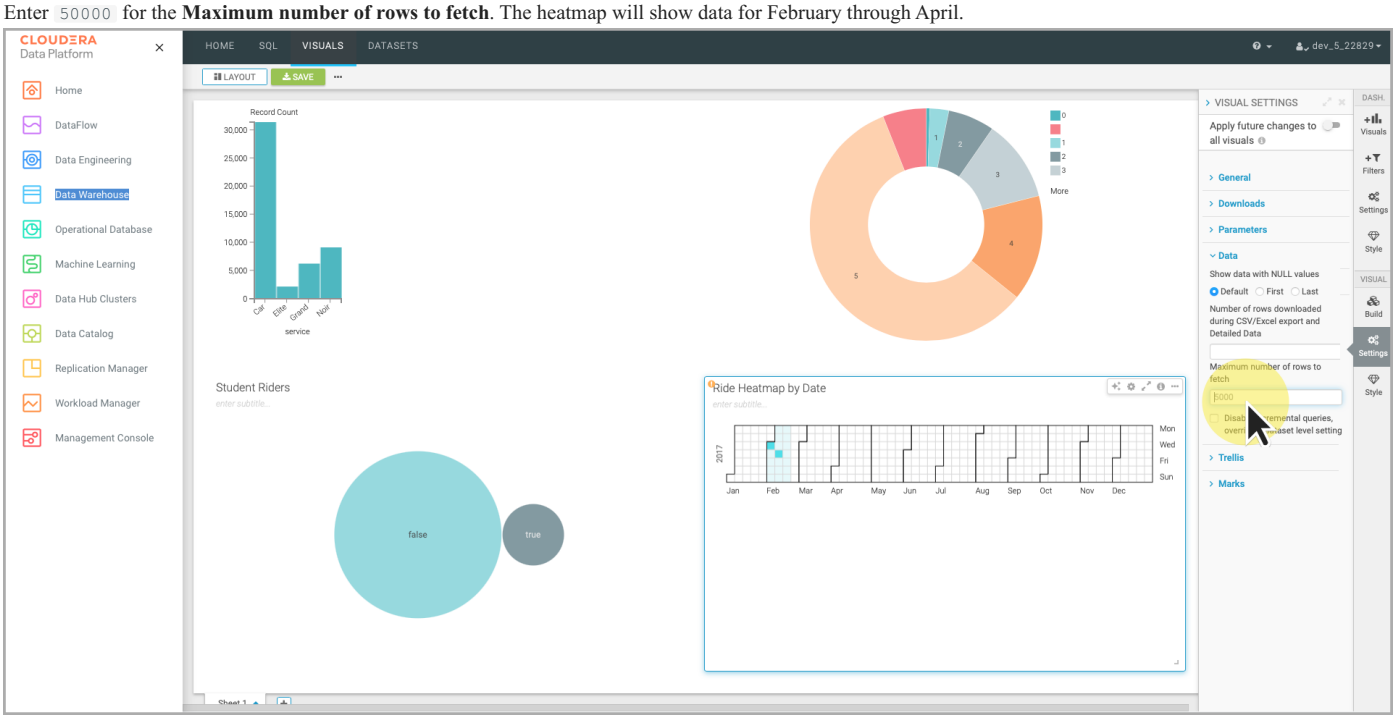
27. The default limit for the number of rows fetched is 5,000. This is limiting the rides displayed to February. Click **Settings** in the right menu.



28. Expand the **Data** category in the **Visual Settings**.



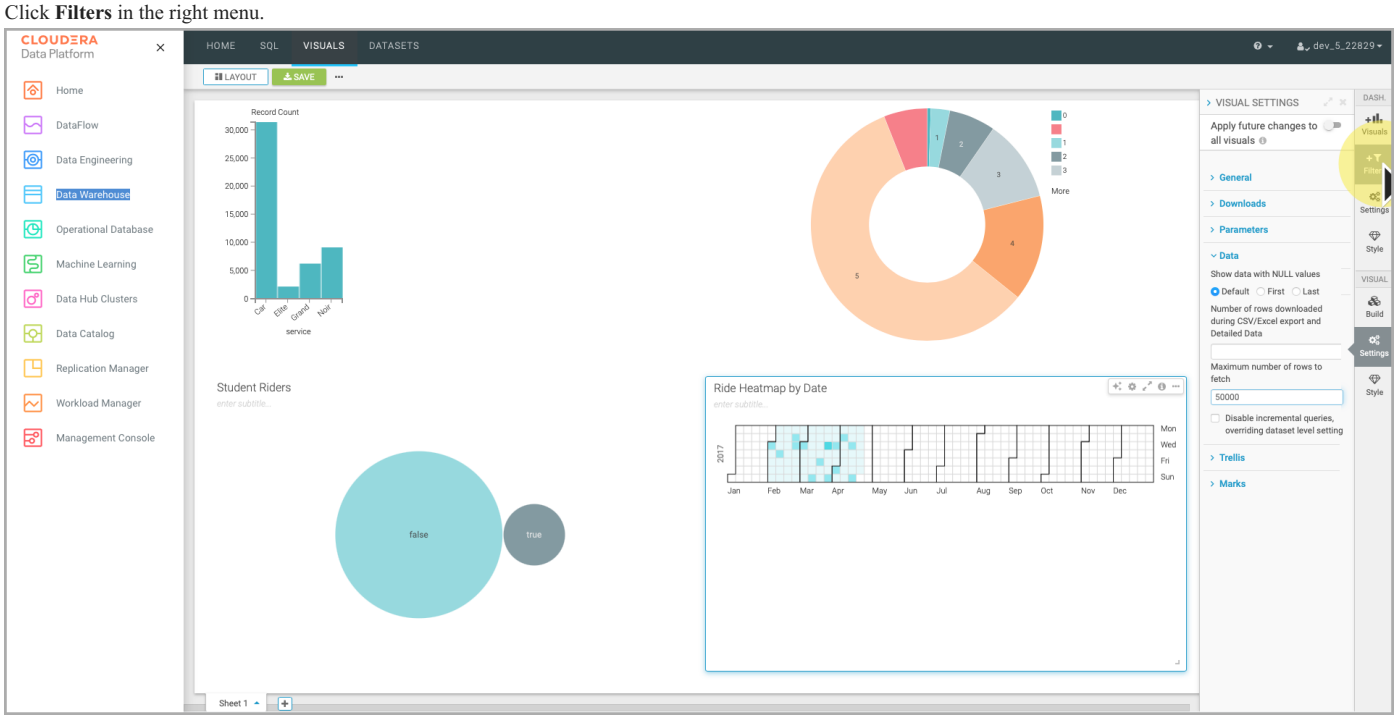
29. Enter `50000` for the **Maximum number of rows to fetch**. The heatmap will show data for February through April.



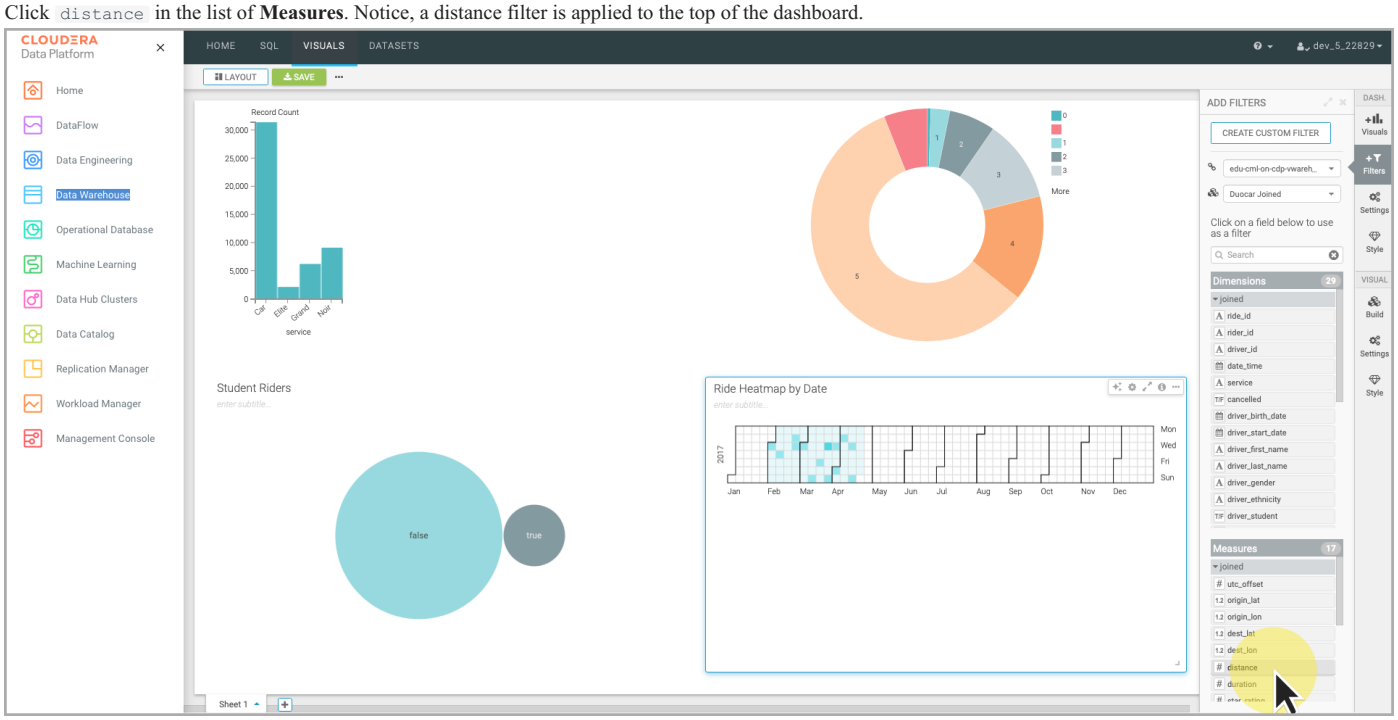
Create Filters

Displaying the data is interesting but it is static. In order to explore the data and find the answers to questions like, "What service do students prefer?", filters can be add to the dashboard to make it interactive.

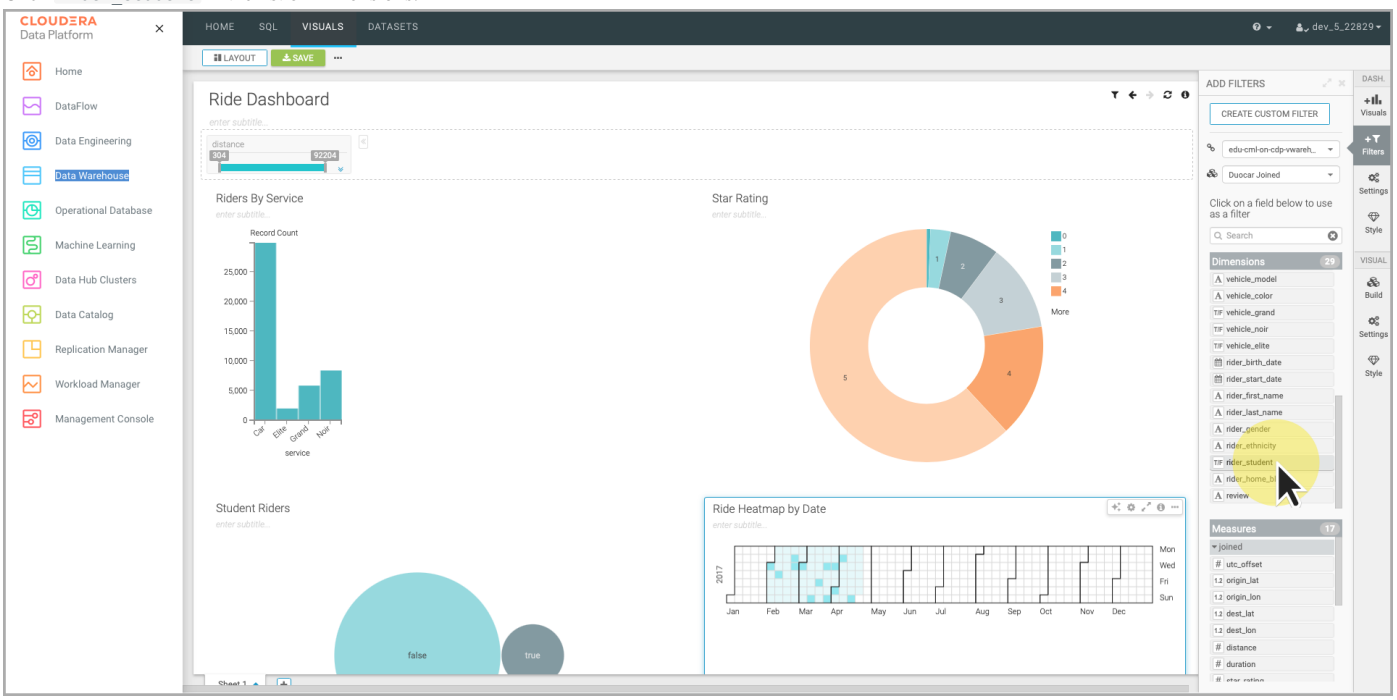
1. Click **Filters** in the right menu.



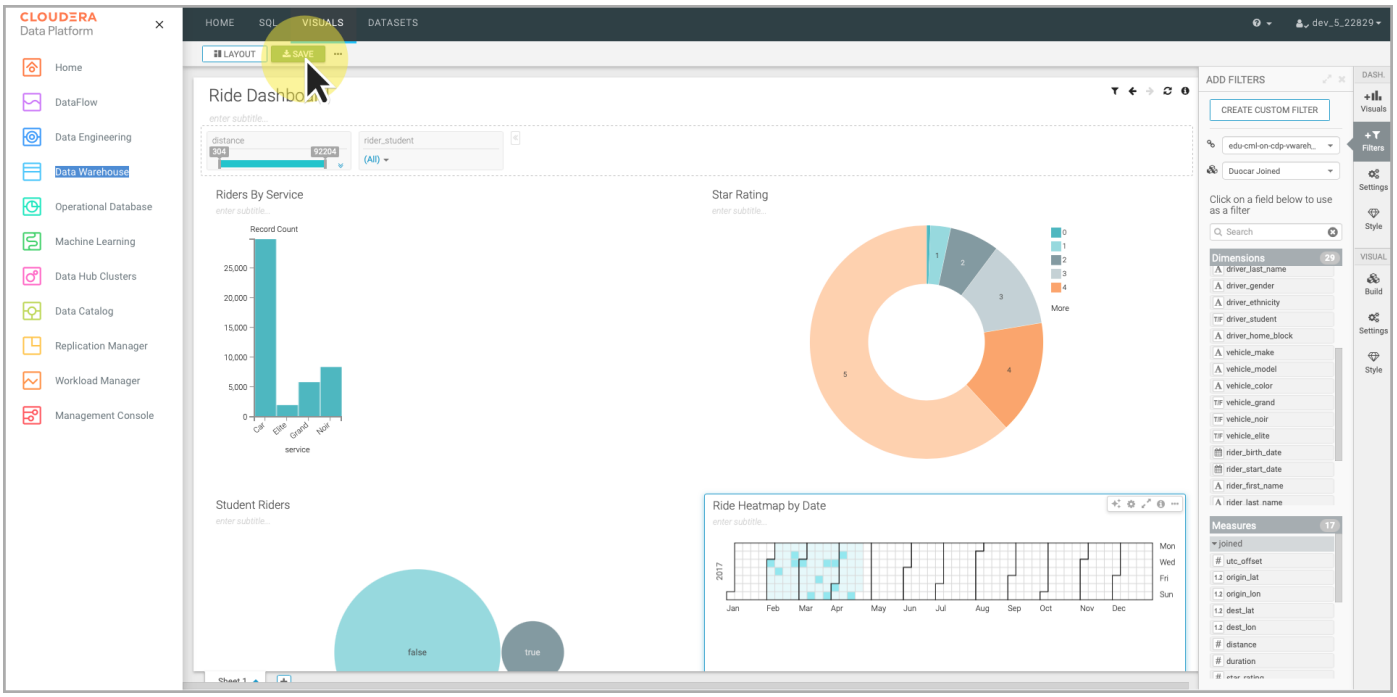
2. Click `distance` in the list of **Measures**. Notice, a distance filter is applied to the top of the dashboard.



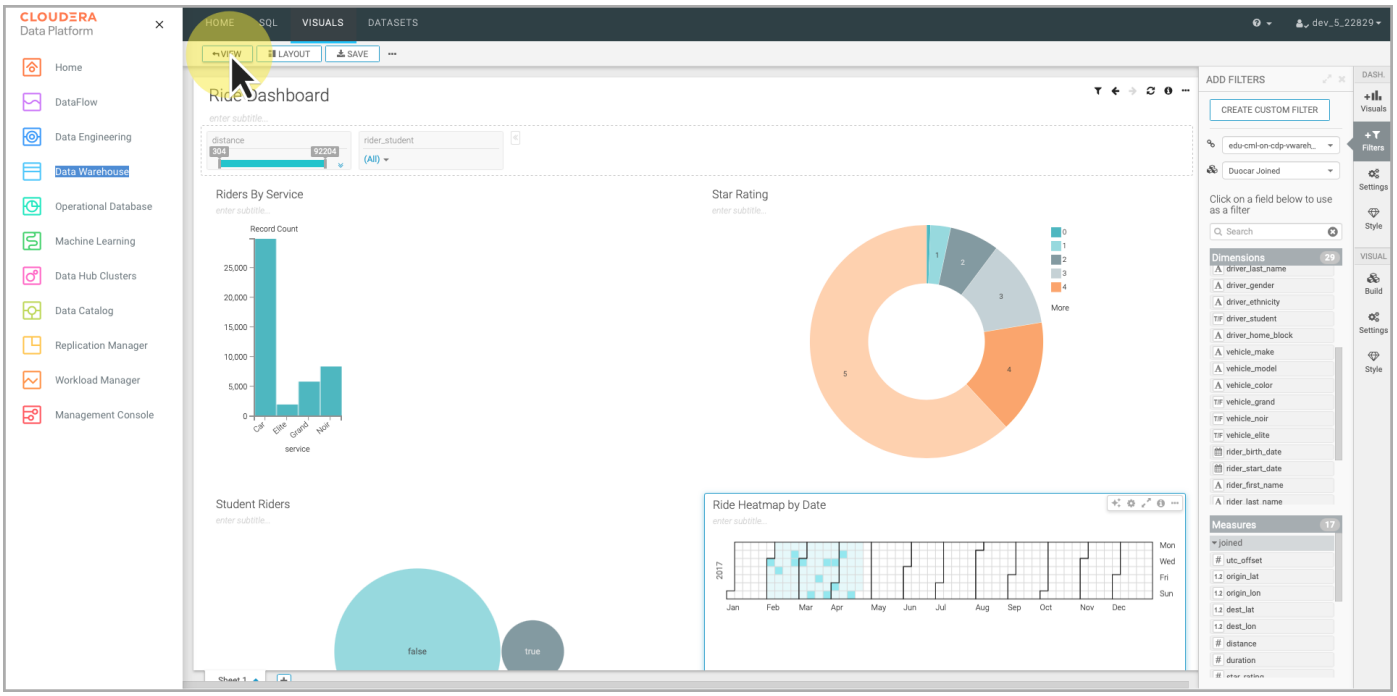
3. Click `rider_student` in the list of **Dimensions**.



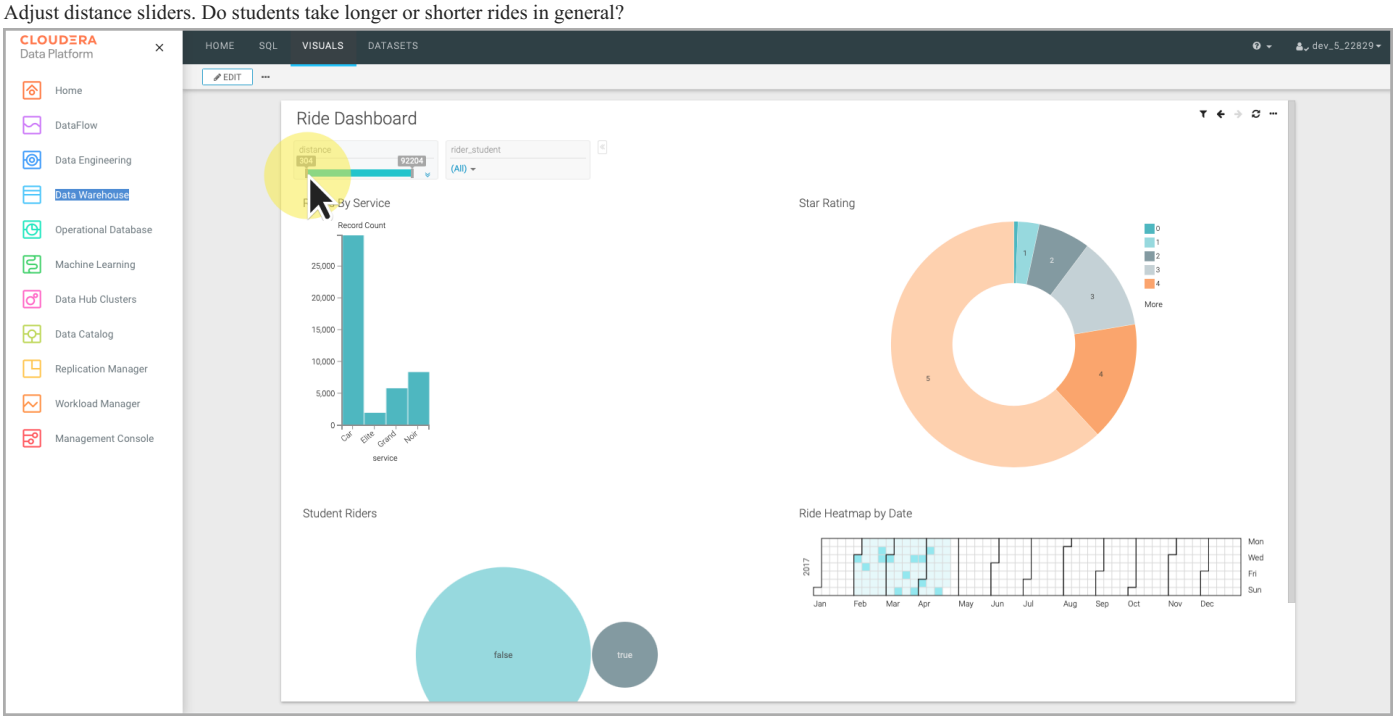
4. Click **Save** to save the dashboard.



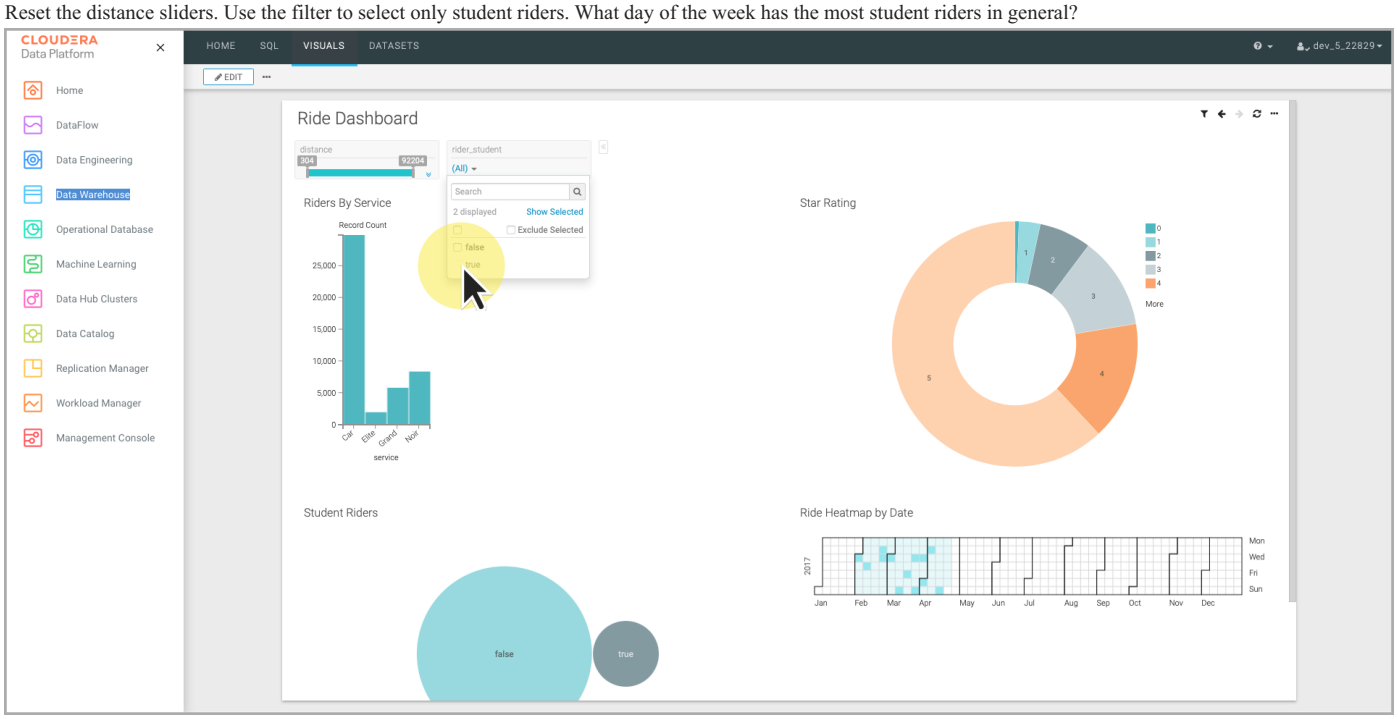
5. Click **View** to view the dashboard.



6. Adjust distance sliders. Do students take longer or shorter rides in general?

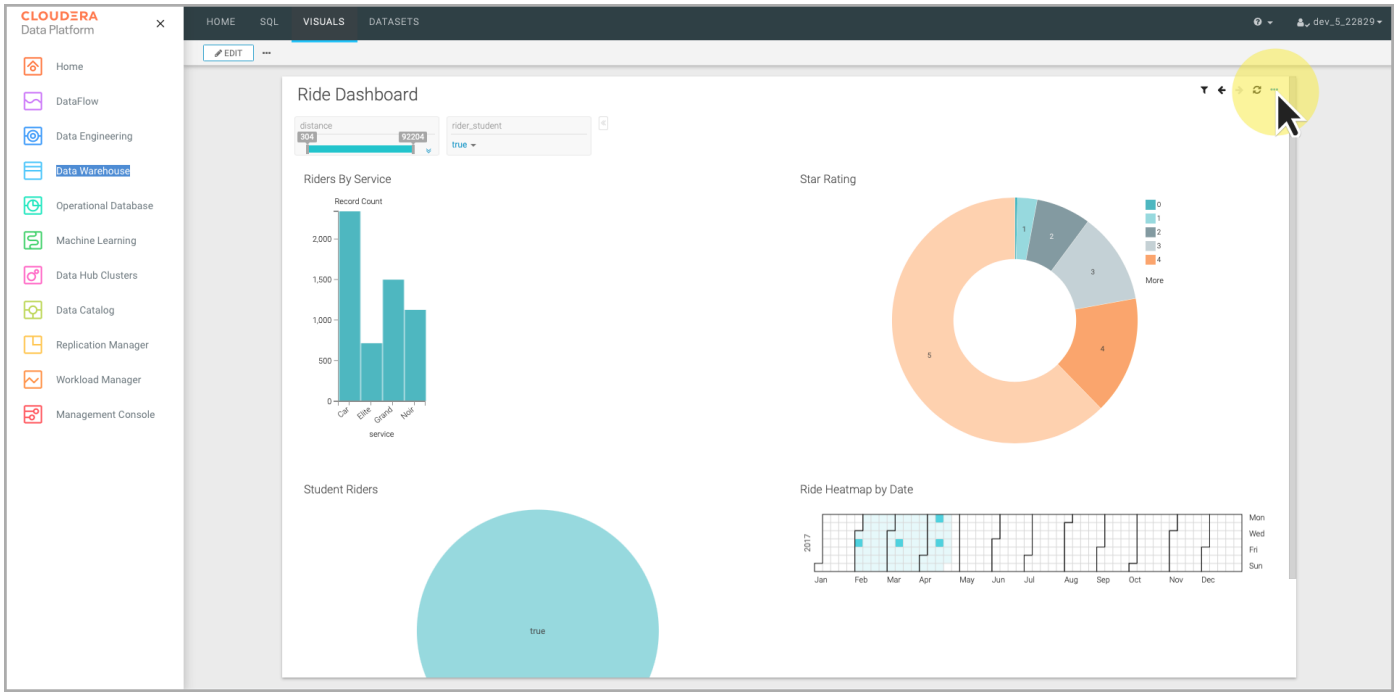


7. Reset the distance sliders. Use the filter to select only student riders. What day of the week has the most student riders in general?

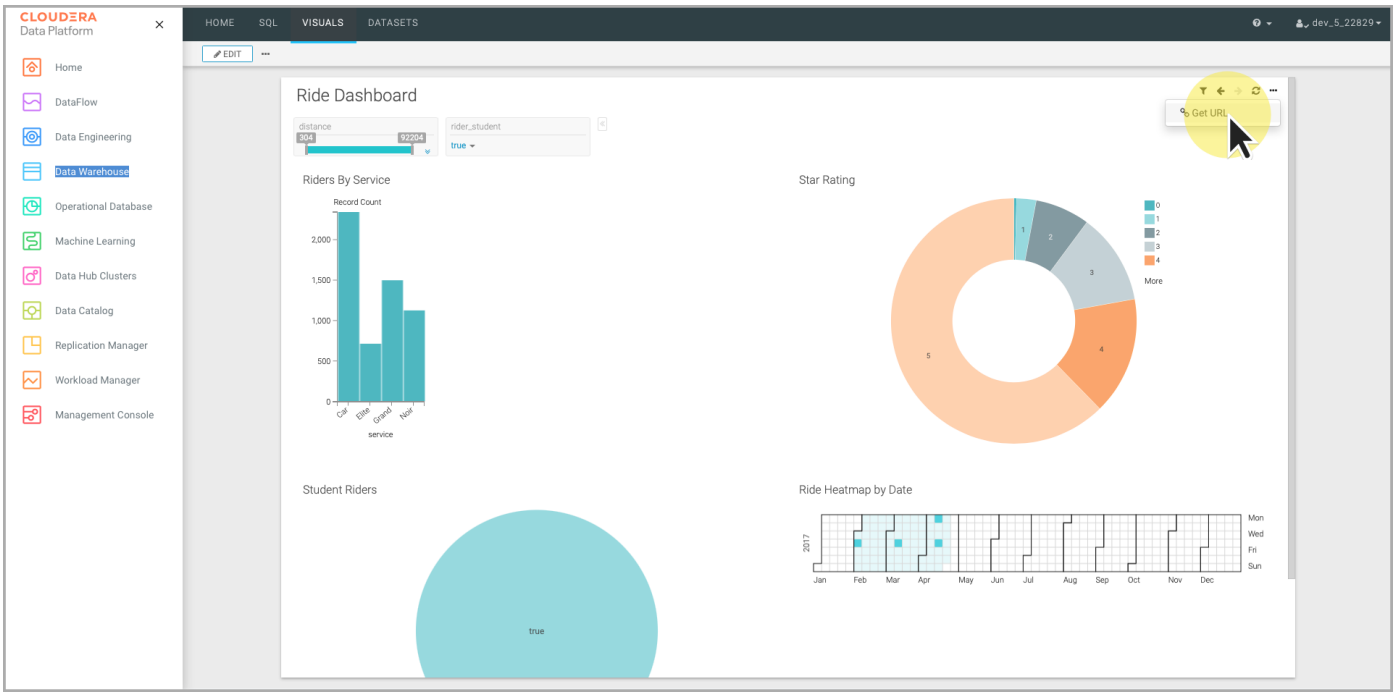


Share the Dashboard

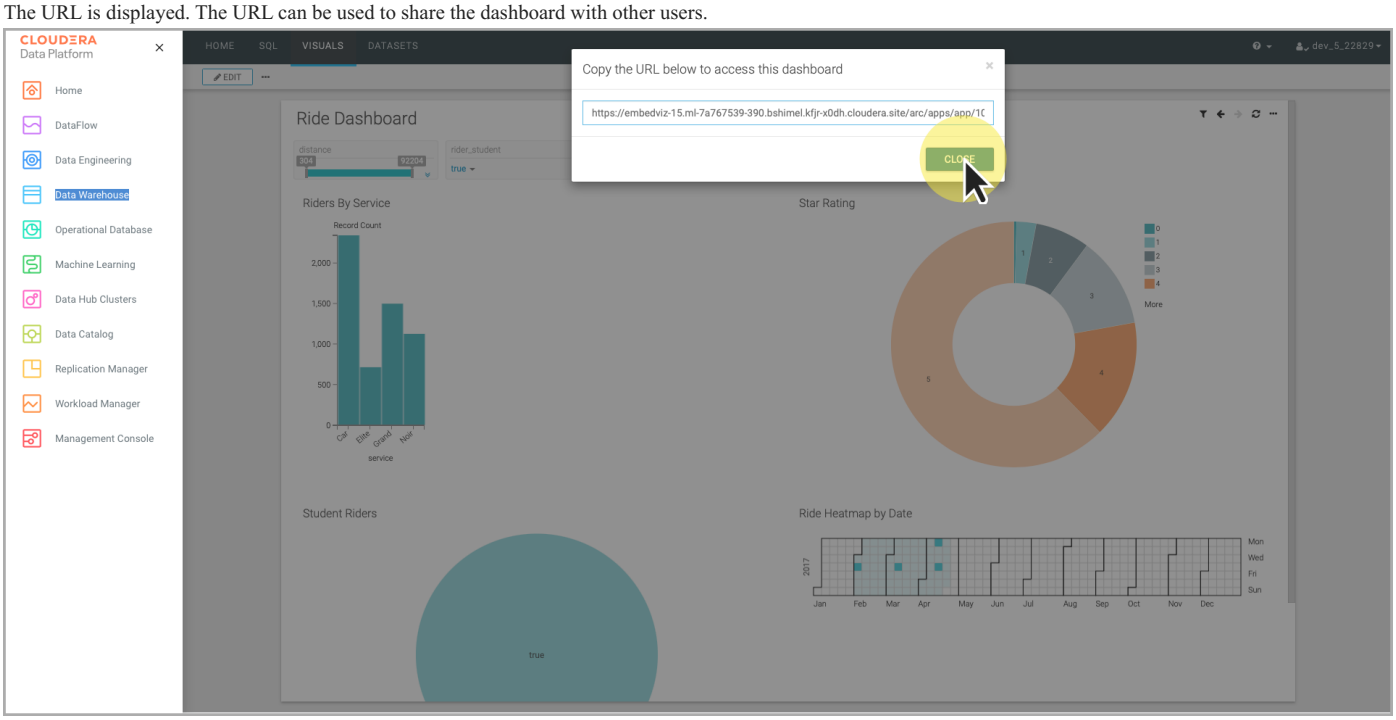
1. Click the ••• menu in the upper right corner of the dashboard.



2. Click **Get URL**.



3. The URL is displayed. The URL can be used to share the dashboard with other users.



End of Exercise

Experiment Tracking

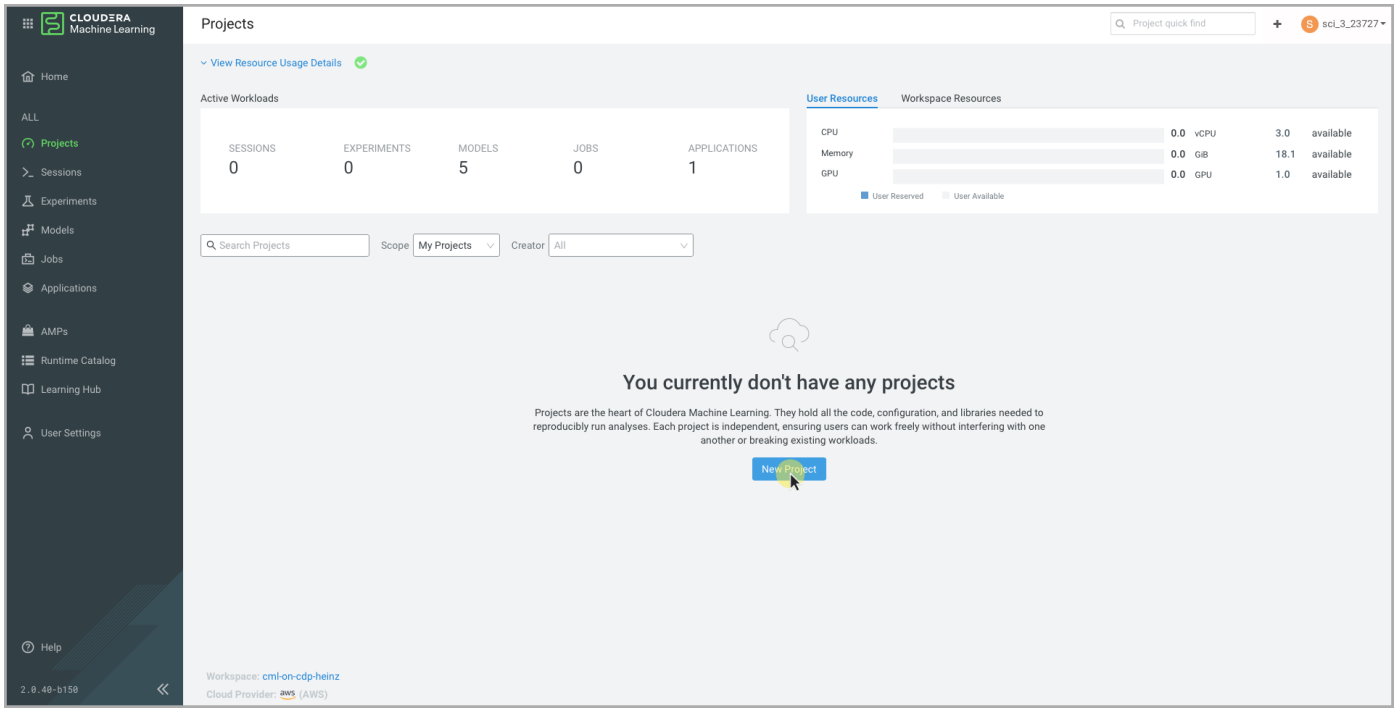
Tracking experiments is critical for knowing what parameters were used to generate a model and how the model is performing. CML projects provide experiment tracking based on MLflow to make this task easier.

In this exercise, you will:

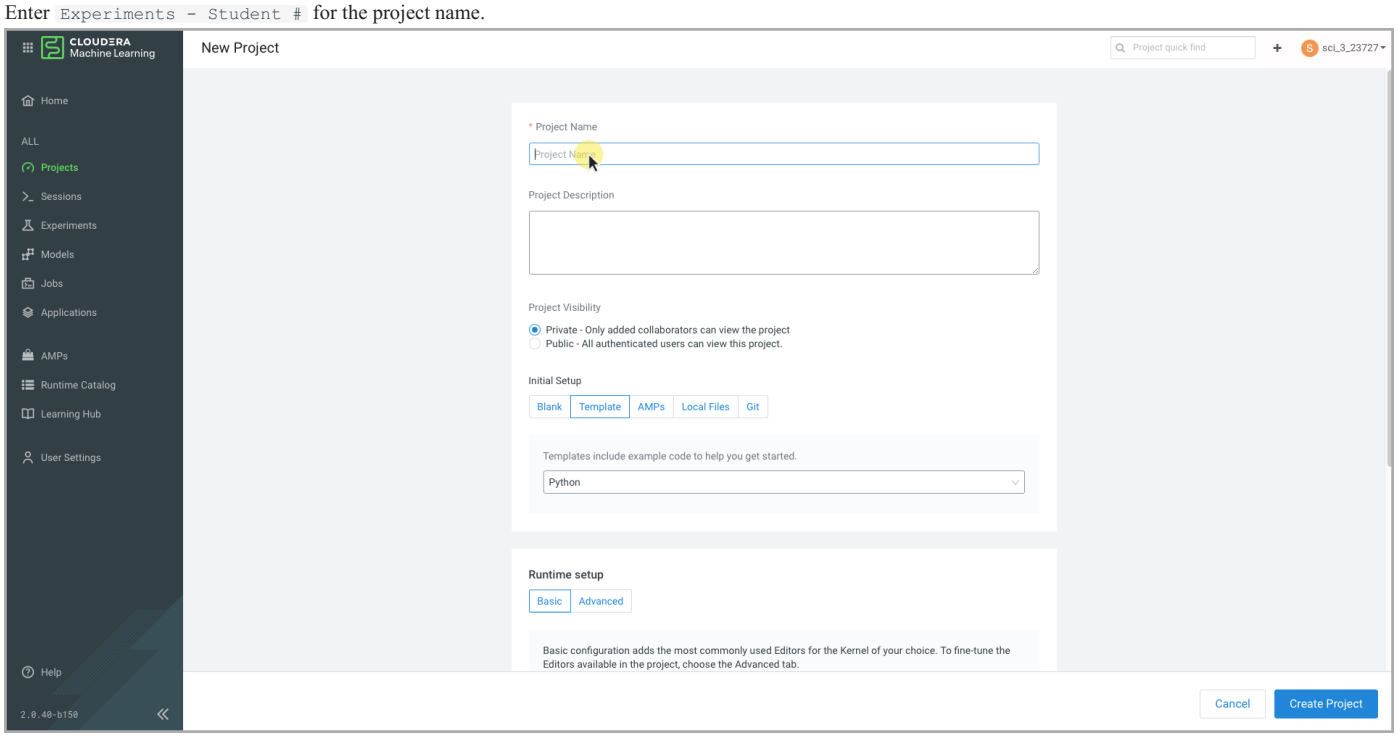
- use a Git repository to create your project,
- use the CML experiments feature to track a experiment, and
- commit the changes back to git.

Create a New Project from Github

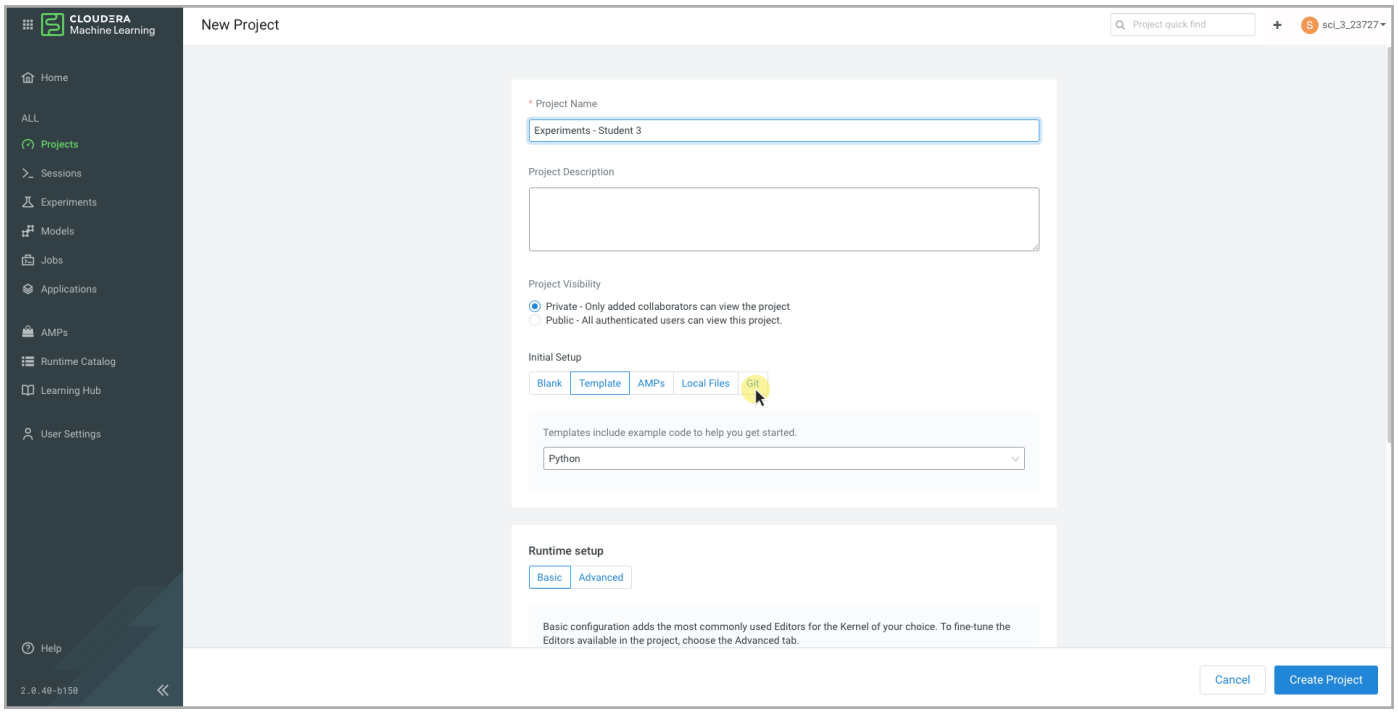
1. Navigate to your workspace.
2. Click the **New Project** button.



3. Enter `Experiments - Student #` for the project name.

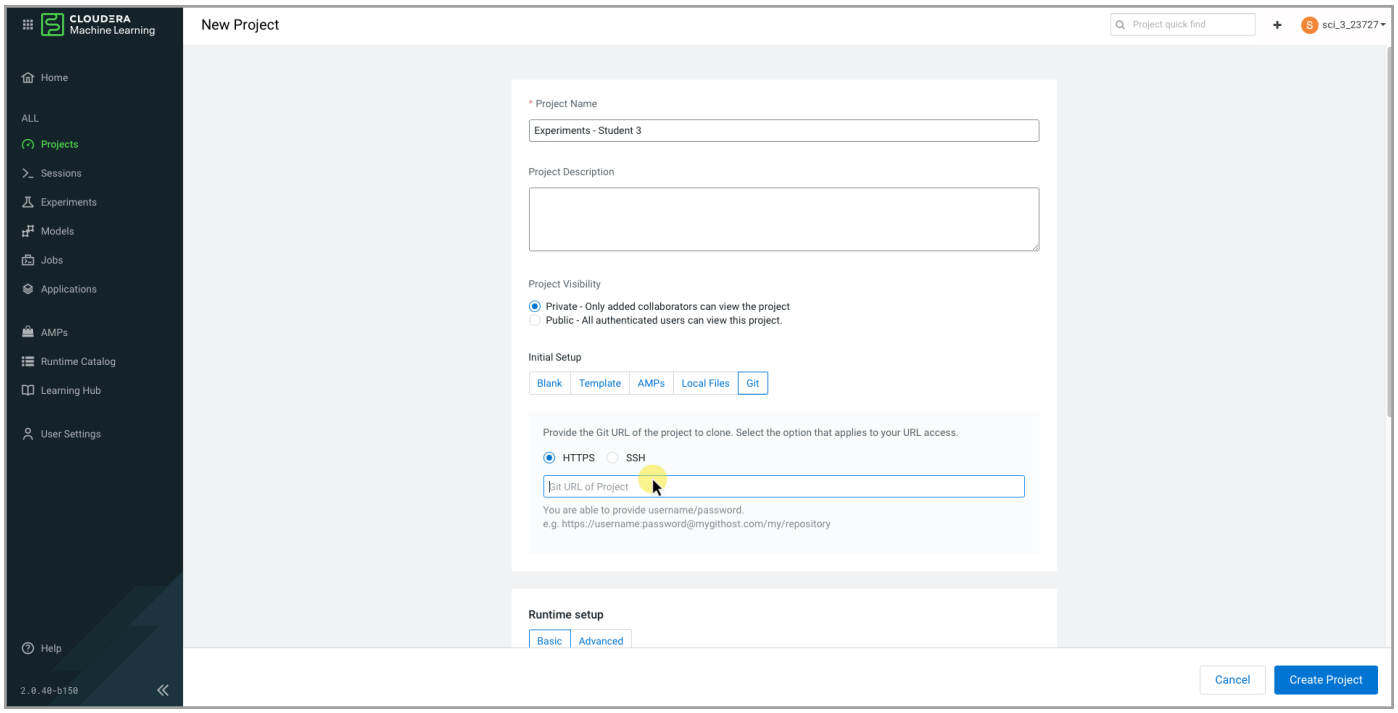


4. Click **Git** under **Initial Setup**.

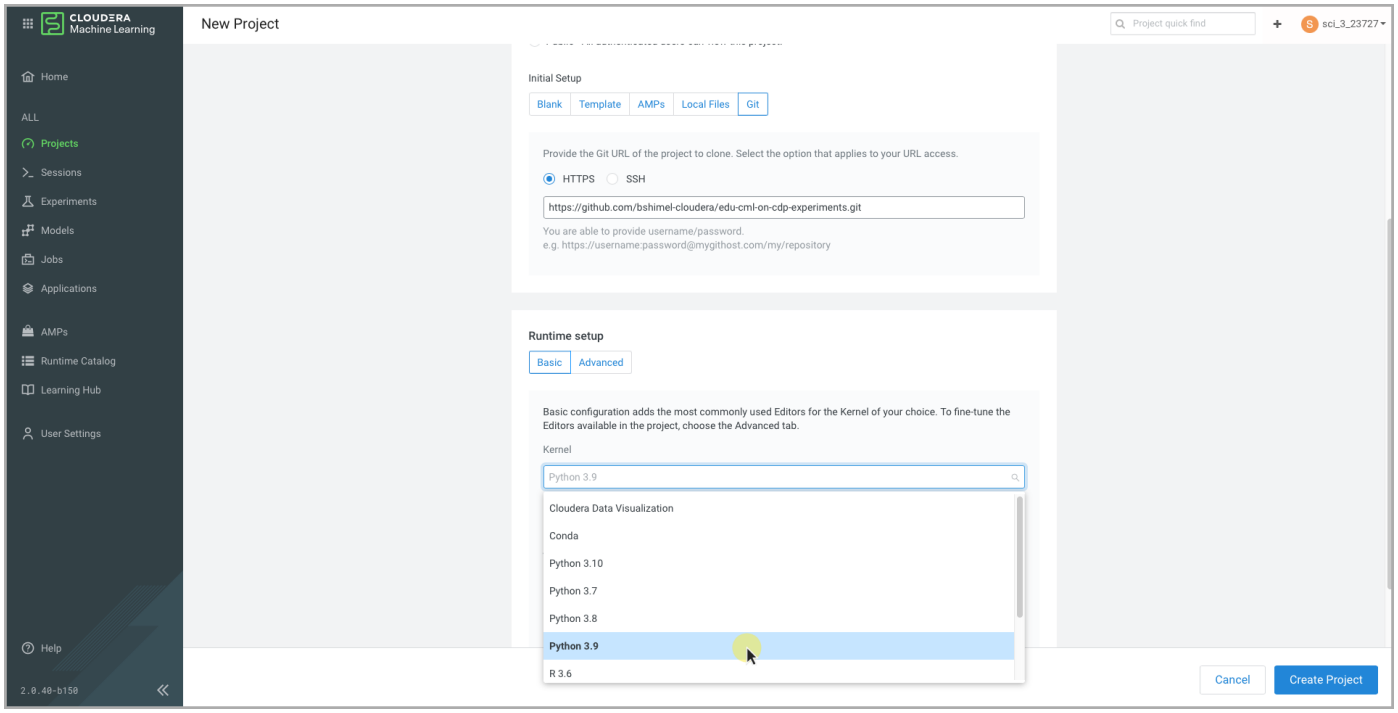


5. Enter the following for the **Git URL**:

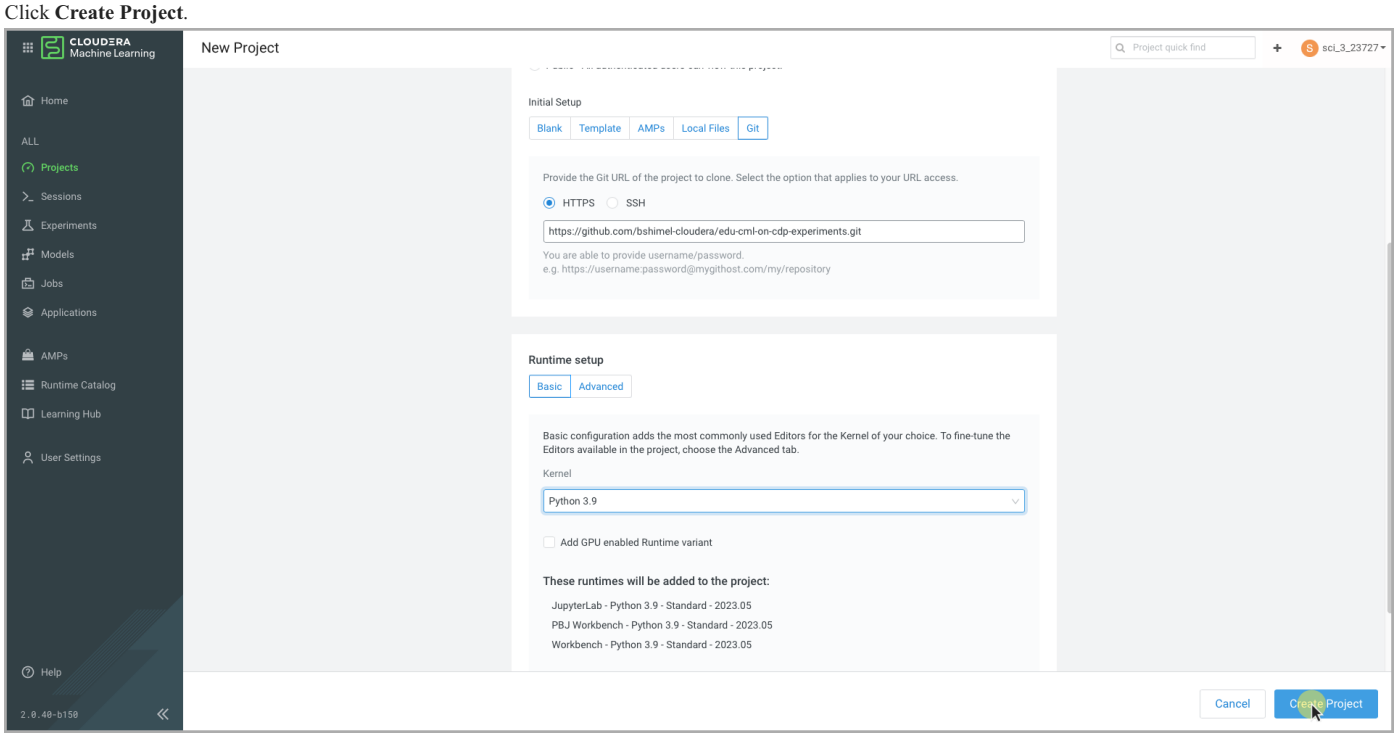
```
https://github.com/bshimel-cloudera/edu-cml-on-cdp-experiments
```



6. Select **Python 3.9** for the **Kernel**.



7. Click **Create Project**.



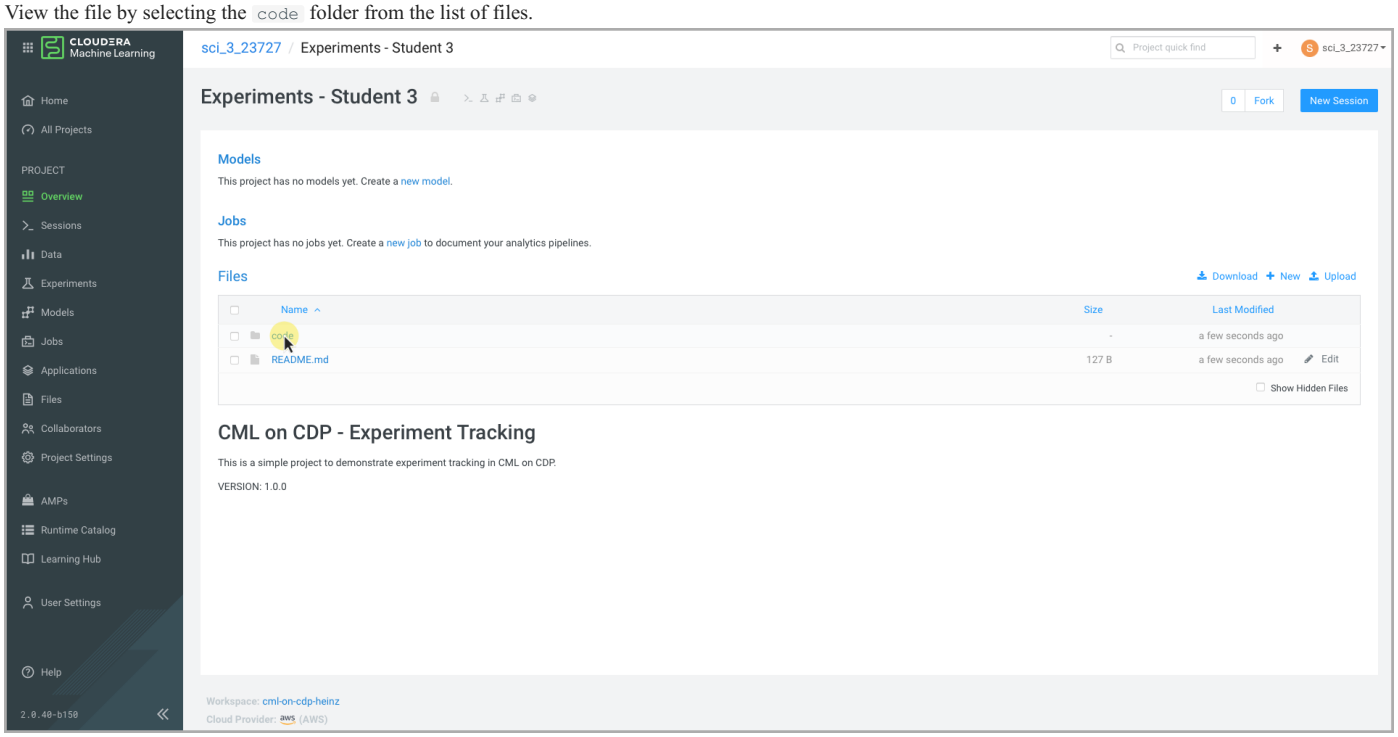
CML will now checkout the content from Github and add it to the project.

View Code and Run Job

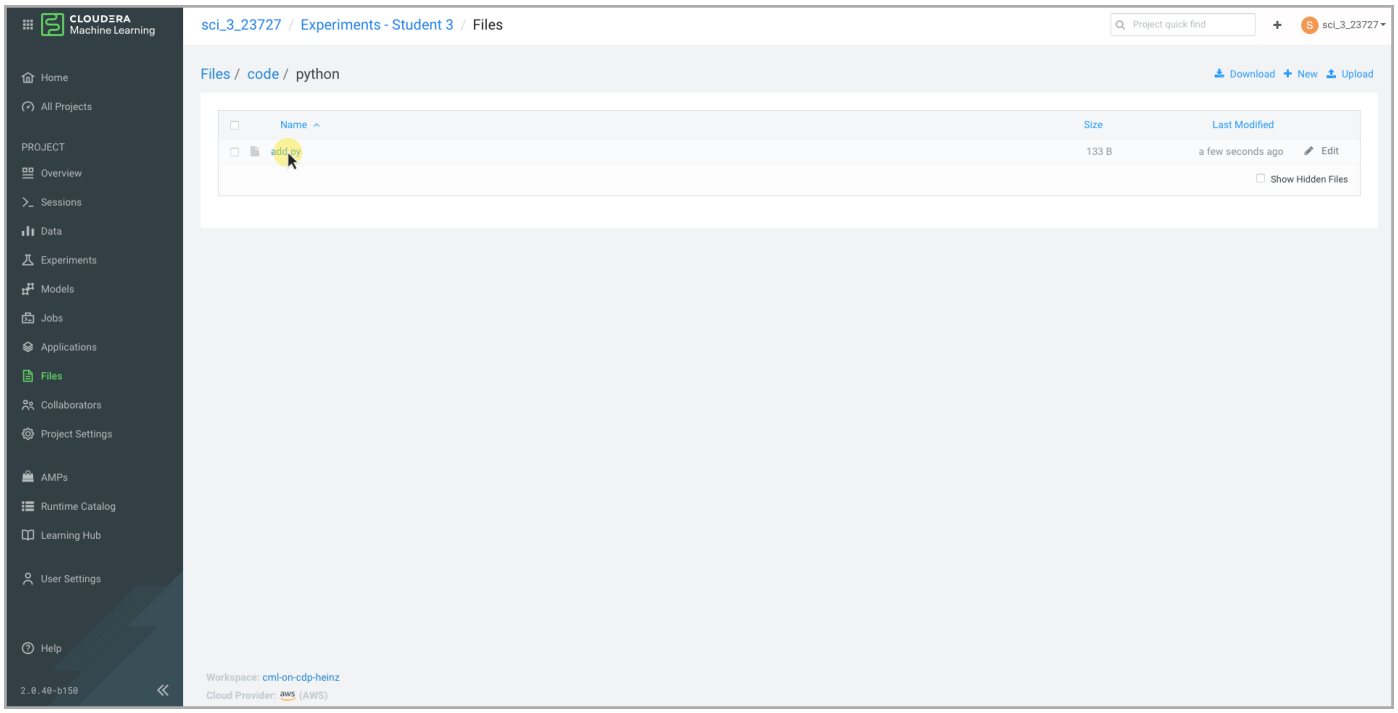
The next step is to look at the code and then run it. Since this exercise is focused on the use of Experiments feature, the code is very simple. There is a single file, `add.py`. It takes a series of numbers and calculates the sum. In this example, the numbers passed as arguments to the `add.py` program are acting as our

parameters, the sum() function is acting as our machine learning model, and the `total` is serving as the prediction or model output. Obviously, the "model" in this case will be 100% accurate, but it will allow the exploration of the Experiment tracking features of CML.

1. View the file by selecting the `code` folder from the list of files.

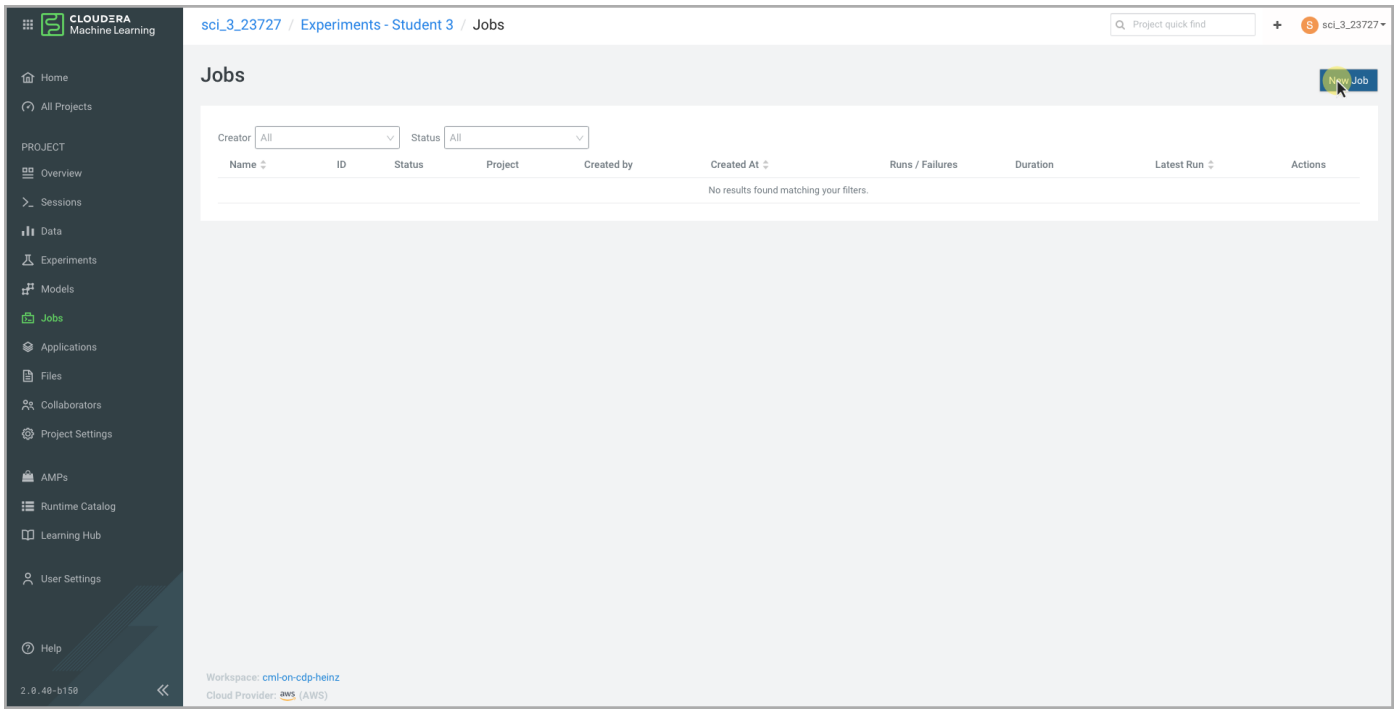


2. Click on the `python` folder, and then click on the `add.py` file.

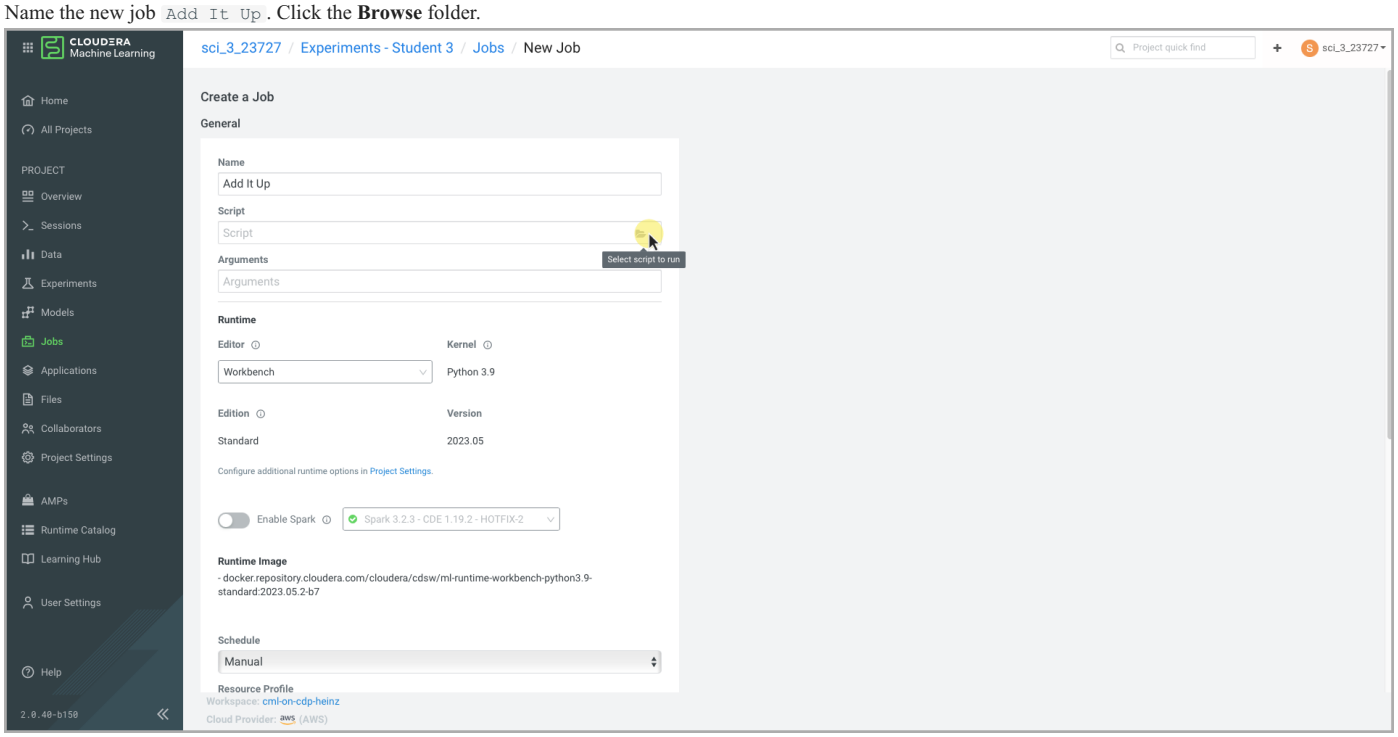


3. You can see that it is a very simple, three-line program. Click **Jobs** in the left-side menu.

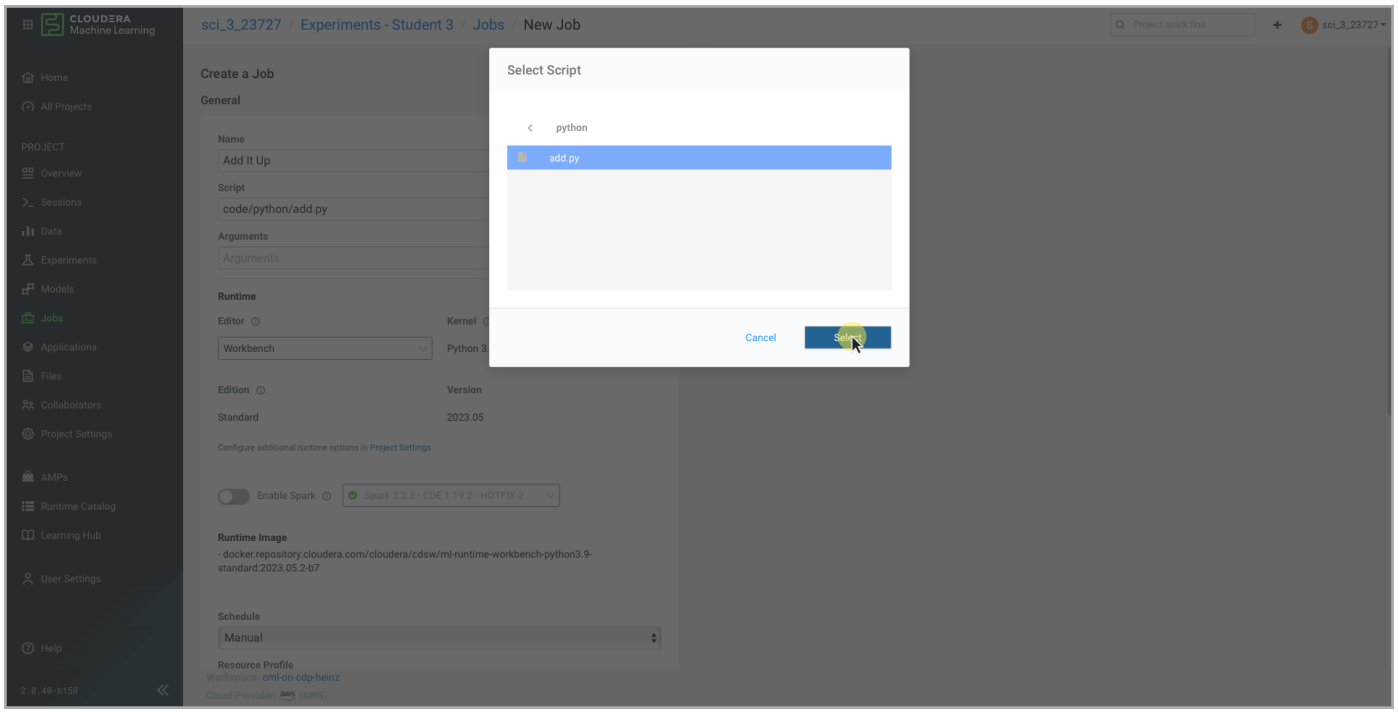
4. Click **New Job**.



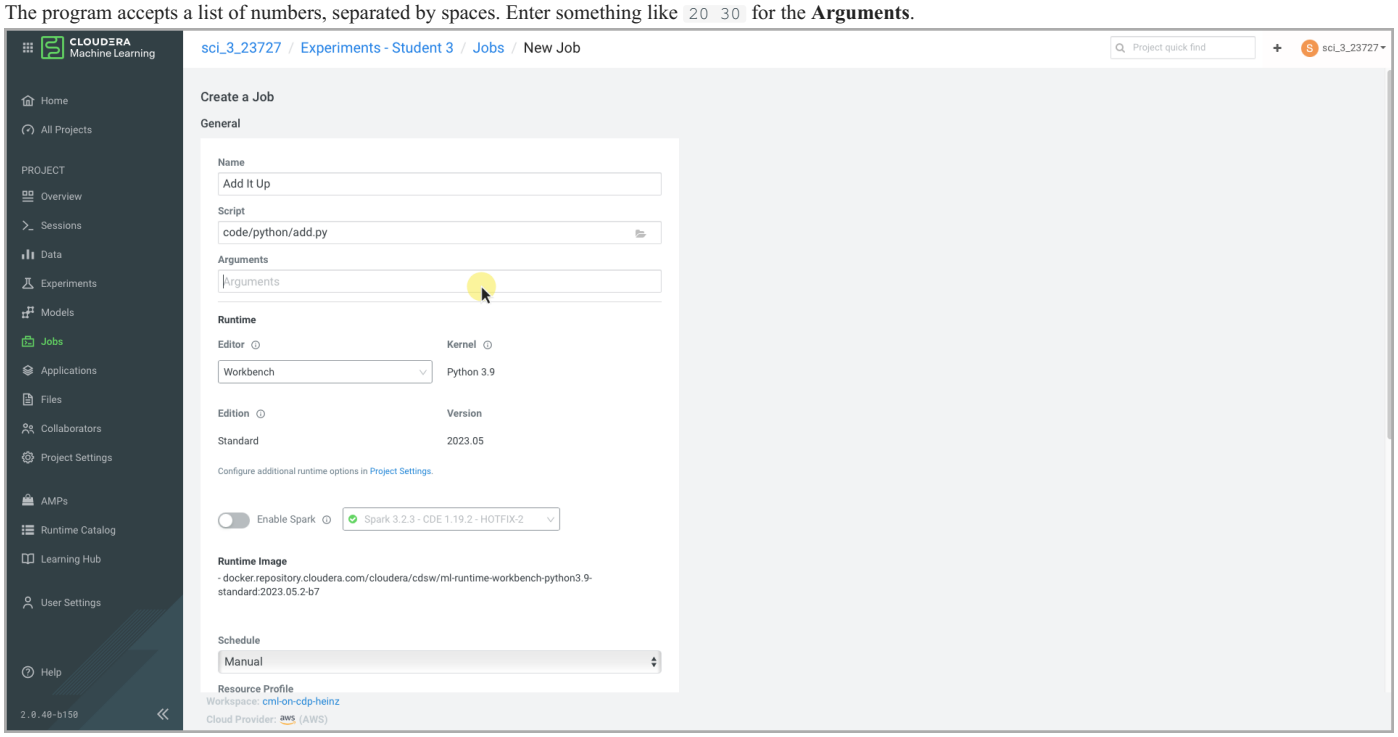
5. Name the new job `Add It Up`. Click the **Browse** folder.



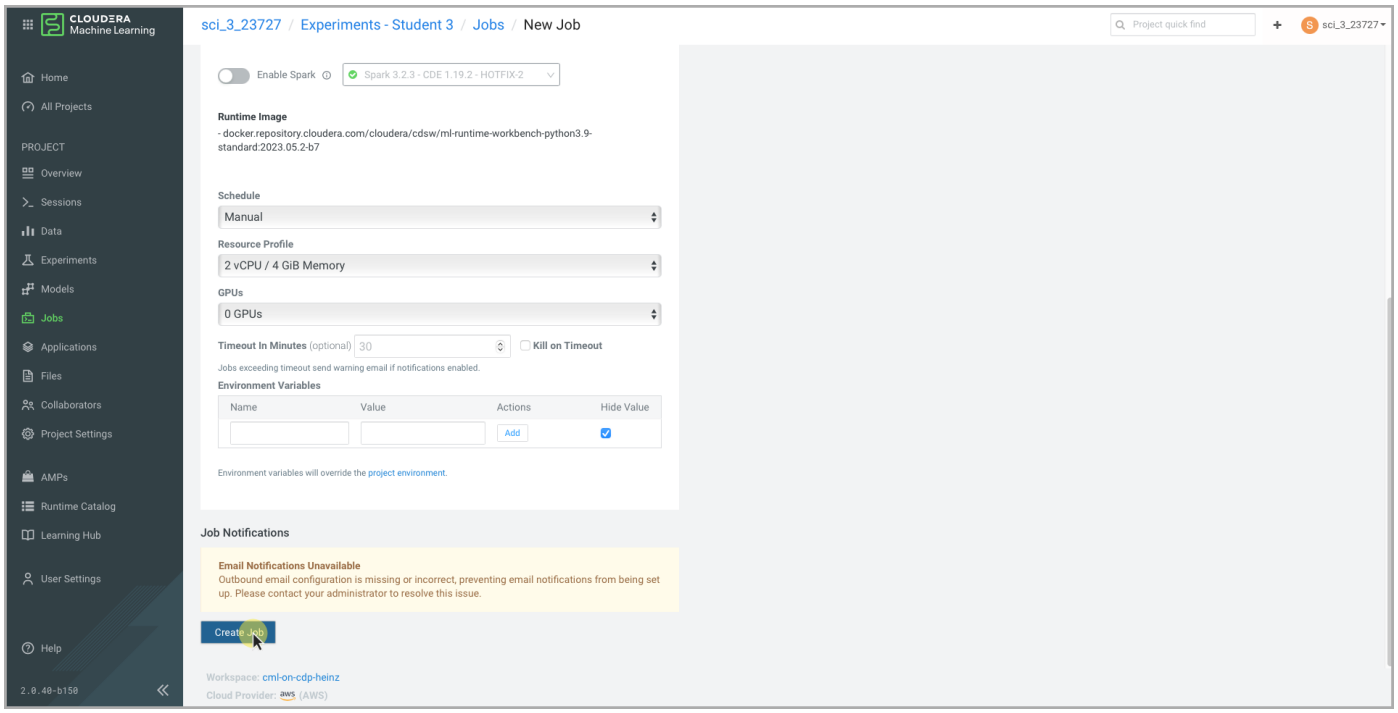
6. Select the `add.py` file.



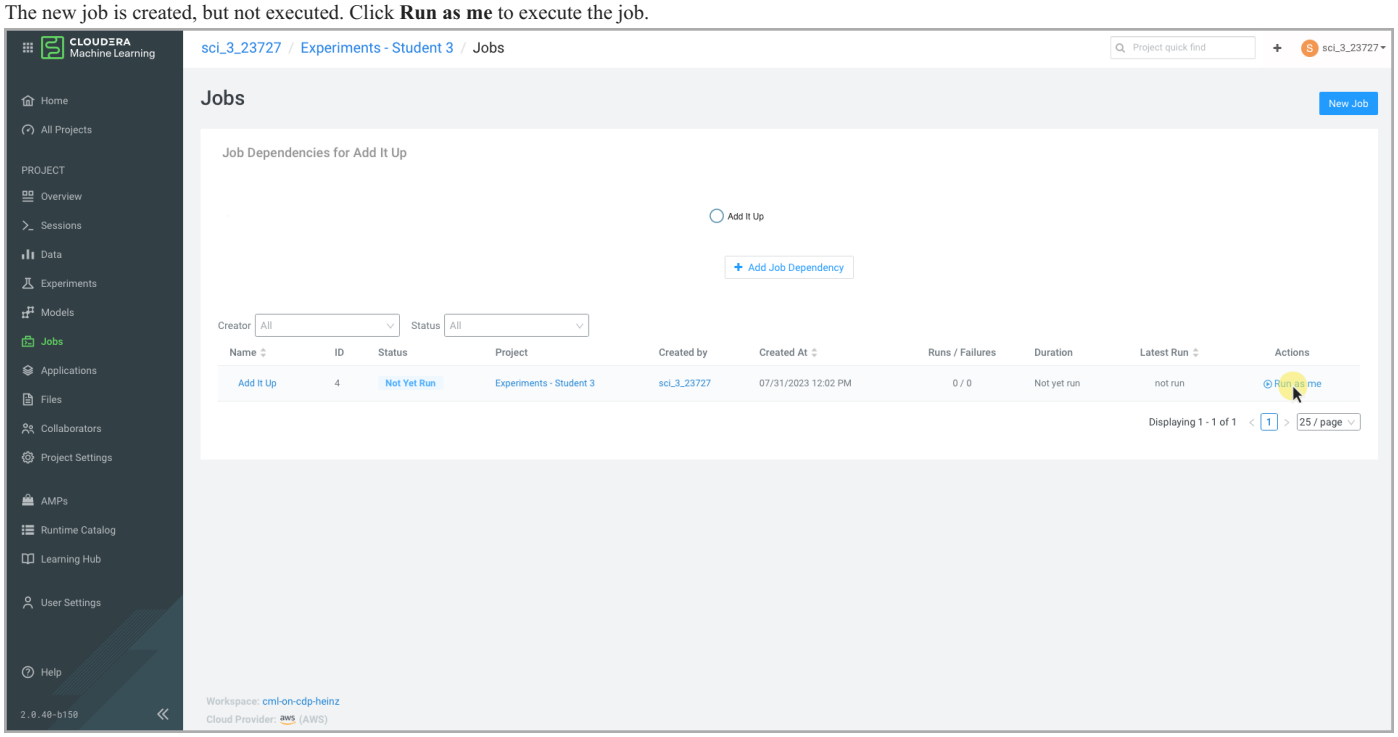
7. The program accepts a list of numbers, separated by spaces. Enter something like `20 30` for the **Arguments**.



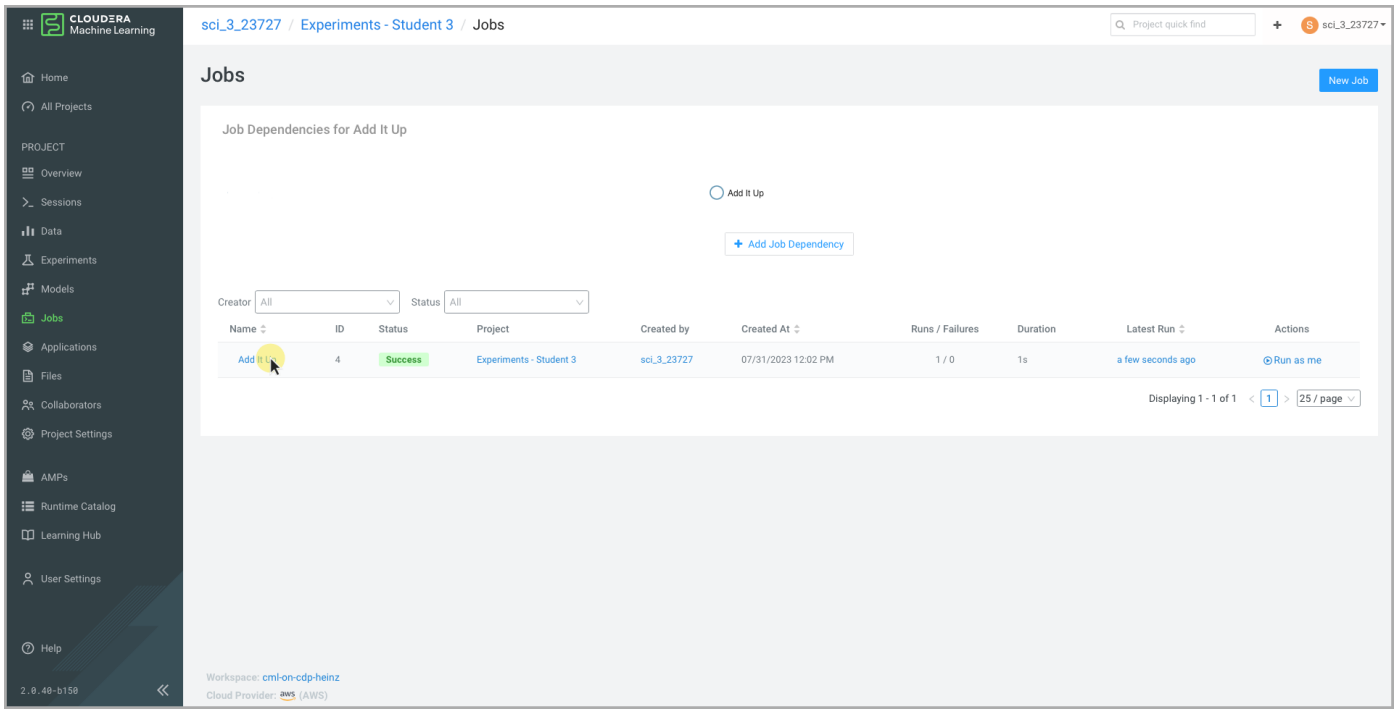
8. Click **Create Job**.



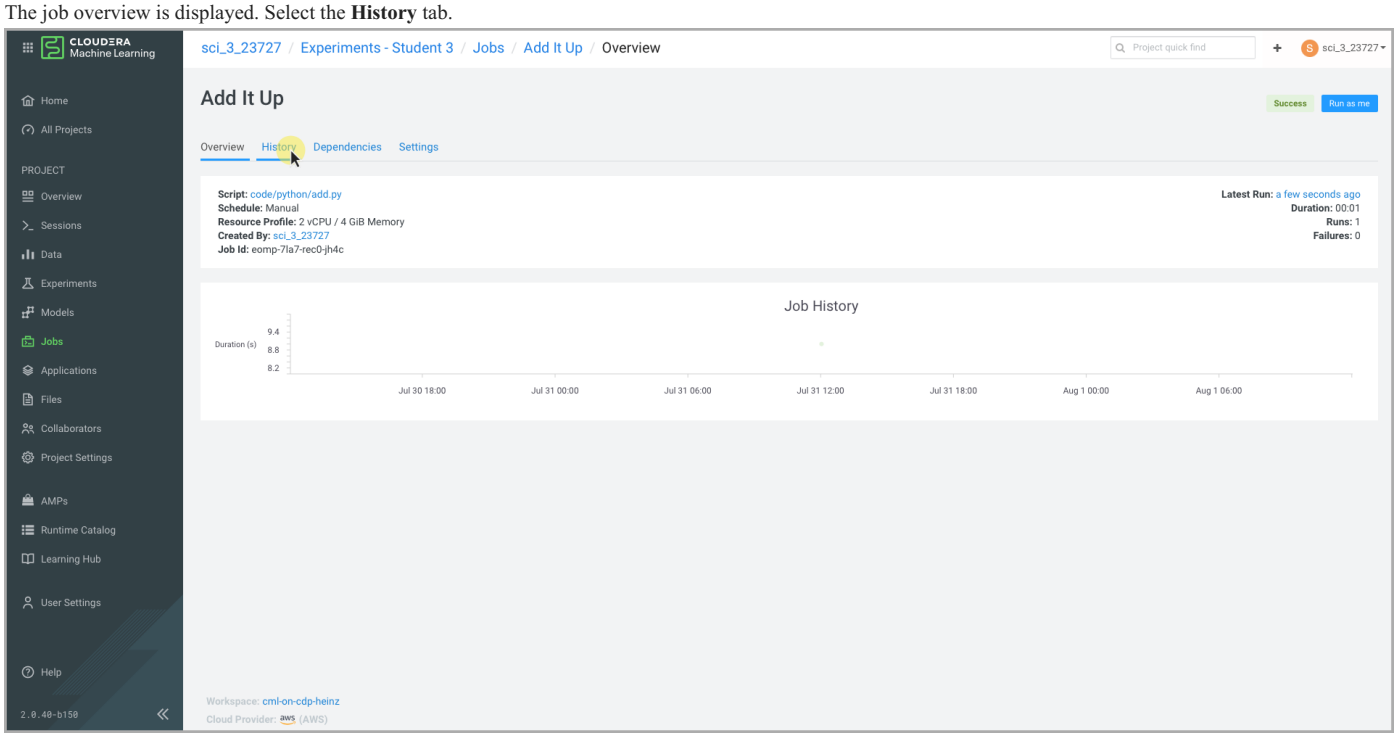
9. The new job is created, but not executed. Click **Run as me** to execute the job.



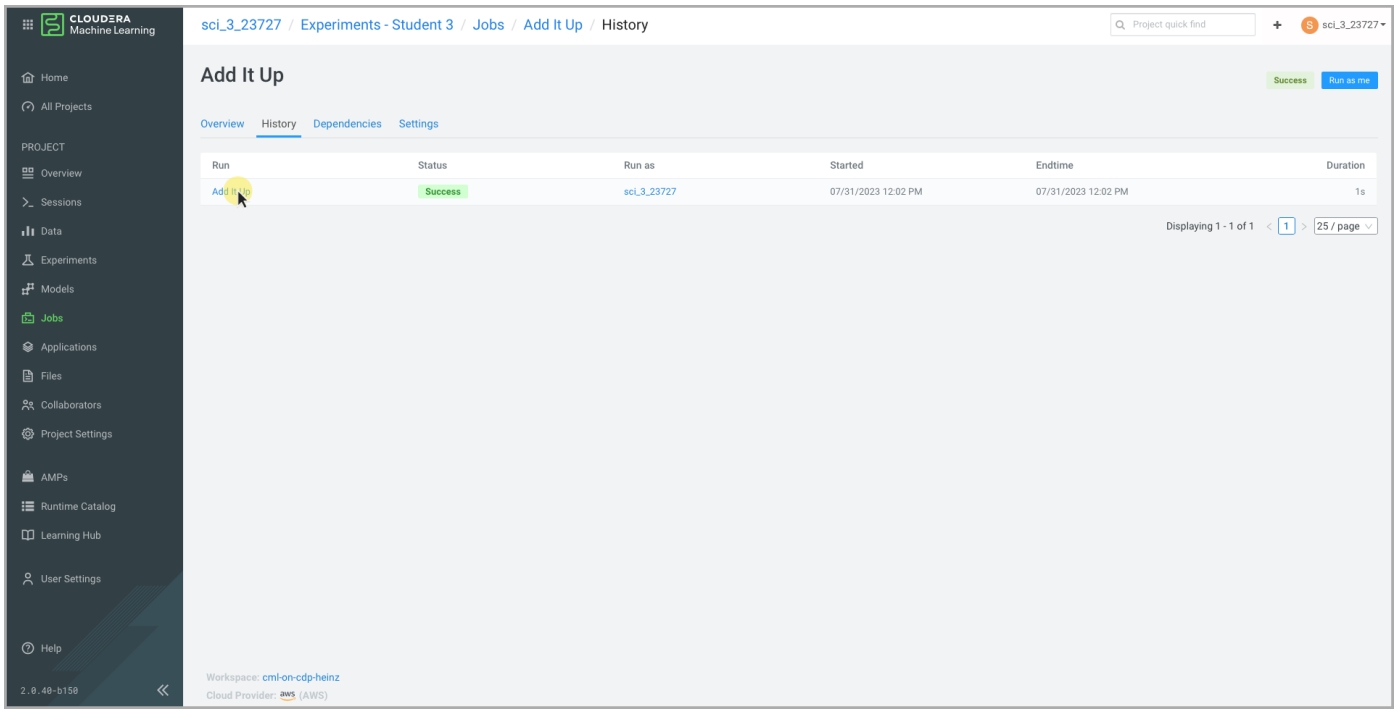
10. Once the **Status** has changed to `Success`, click on the job name.



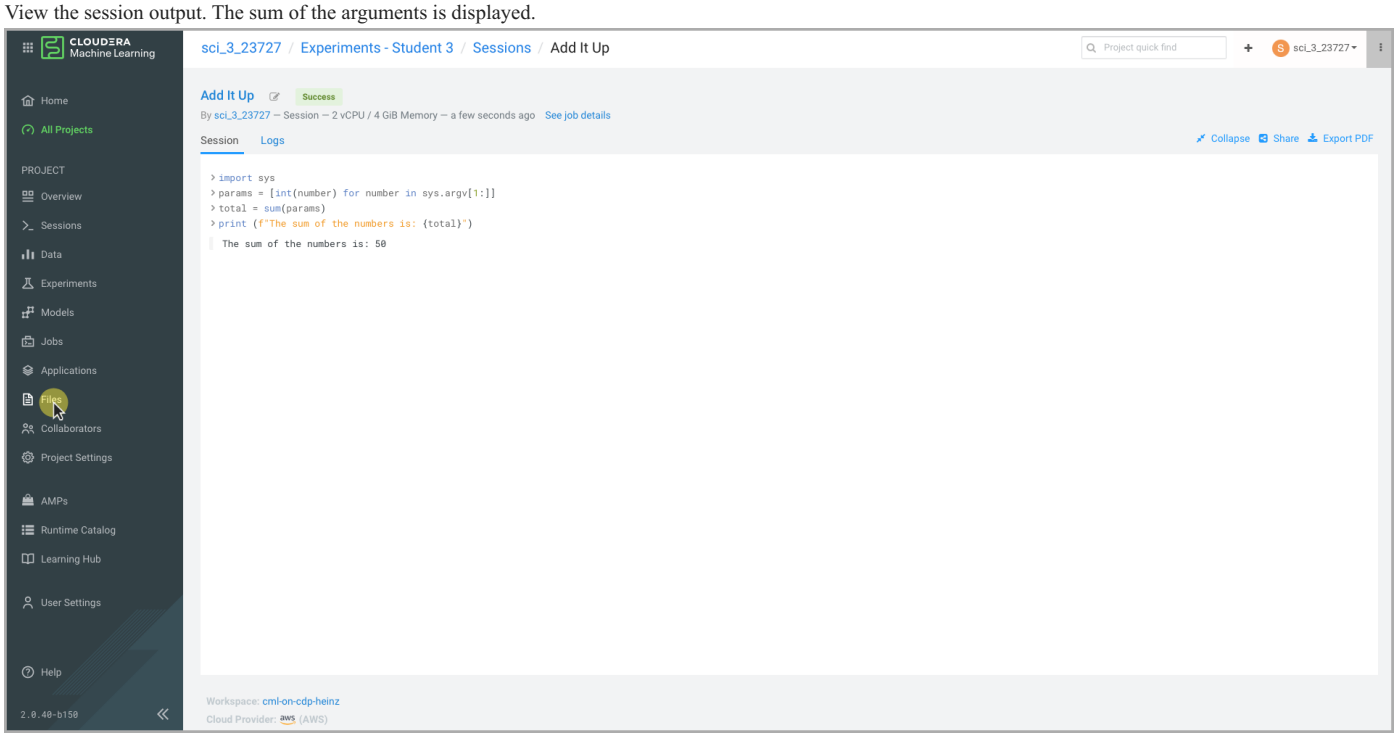
11. The job overview is displayed. Select the **History** tab.



12. A list of job runs is displayed. There should only be one job run at this point. Click on the run.

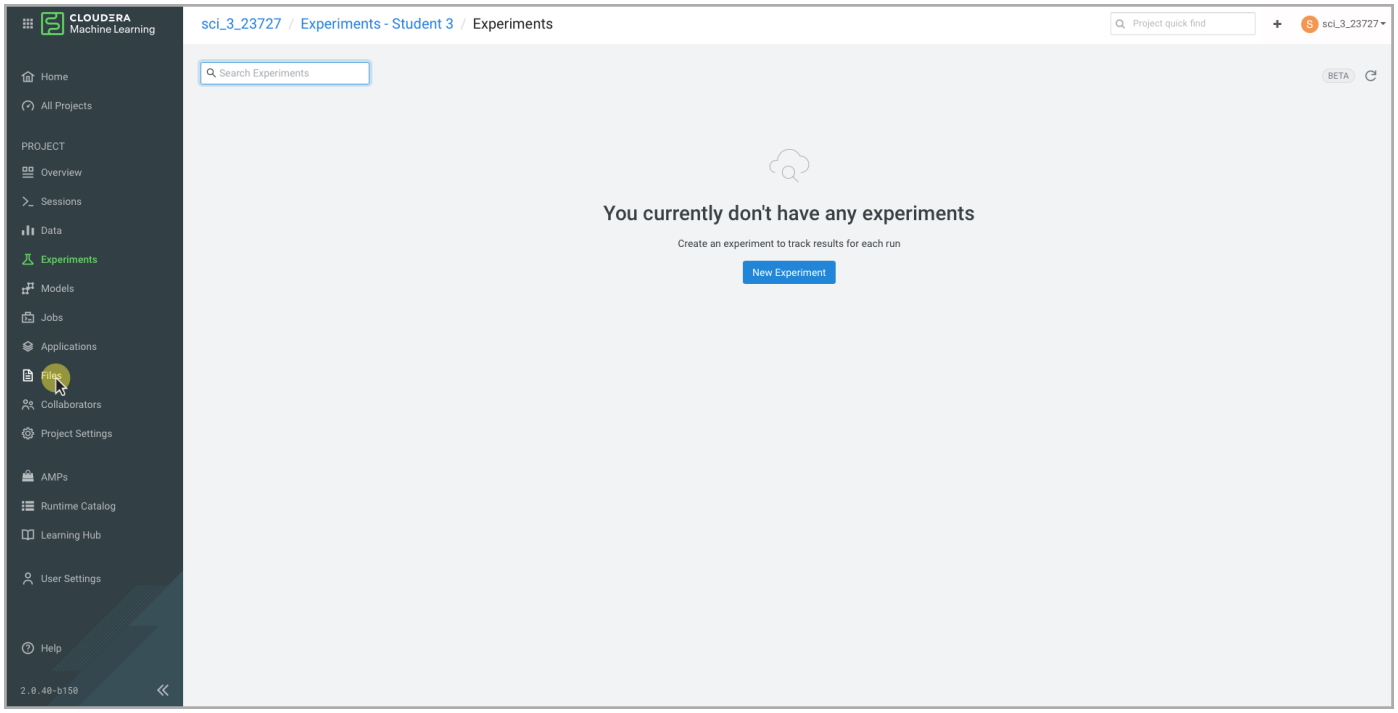


13. View the session output. The sum of the arguments is displayed.

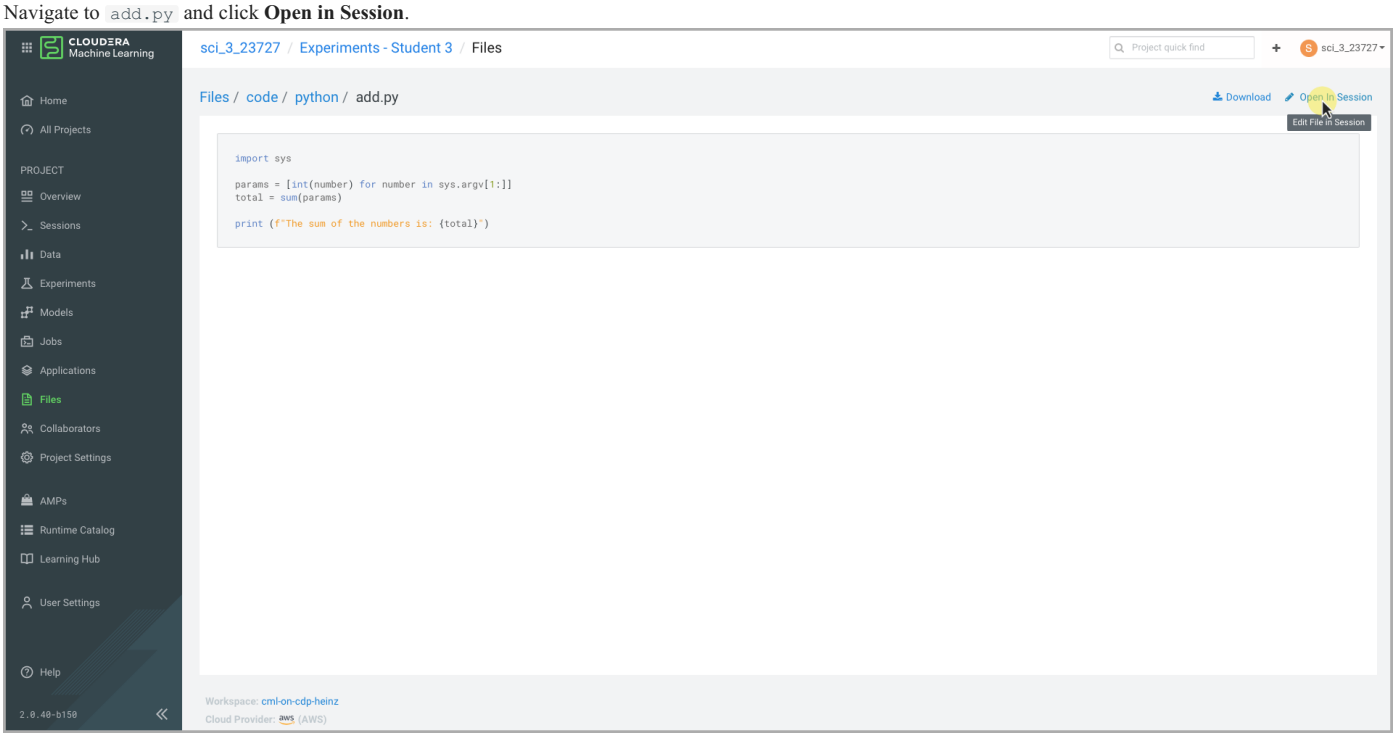


Creating an Experiment

1. Click **Files** in the left-side menu.



2. Navigate to `add.py` and click **Open in Session**.



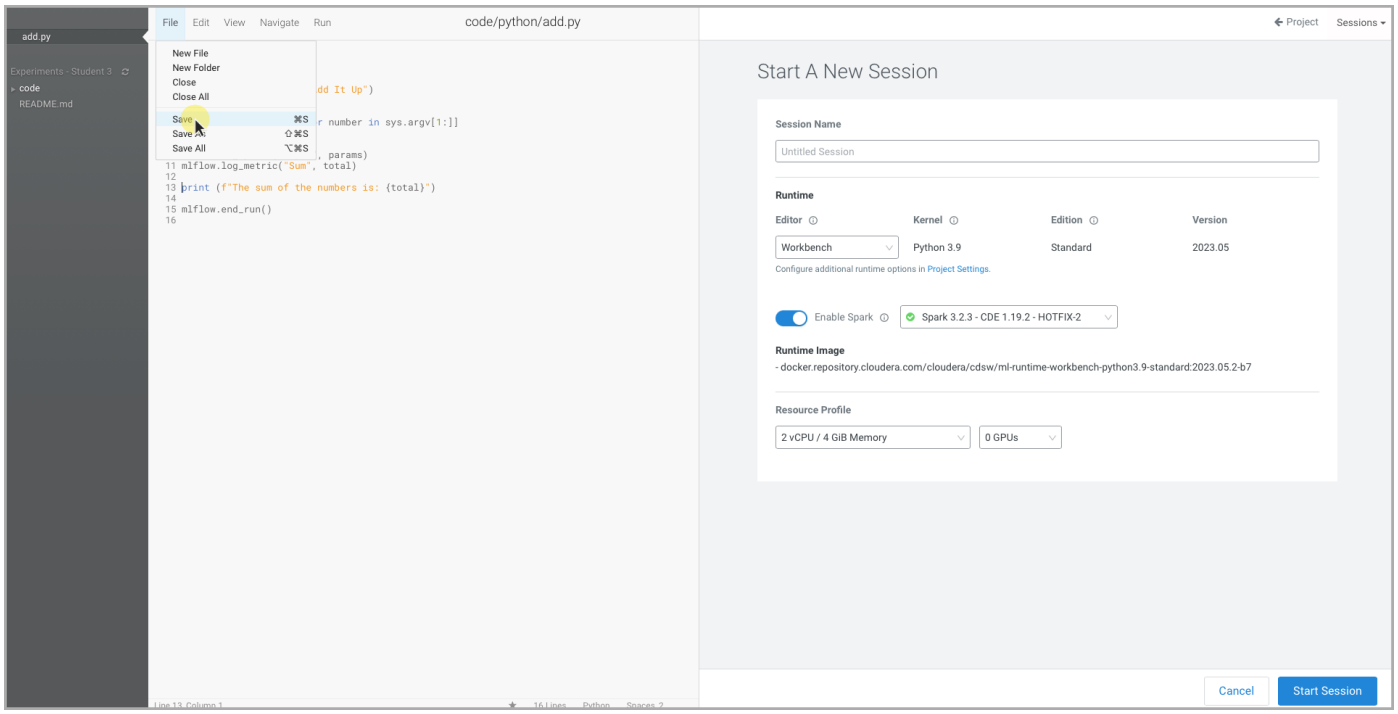
3. The file is opened in the Workspace file editor. An actual session is not needed to edit the file.
   Edit the file to look as follows:

```
import sys
import mlflow

mlflow.set_experiment("Add It Up")
mlflow.start_run()

params = [int(number) for number in sys.argv[1:]]
total = sum(params)

mlflow.log_param("Input", params)
mlflow.log_metric("Sum", total)

print (f"The sum of the numbers is: {total}")

mlflow.end_run()
```

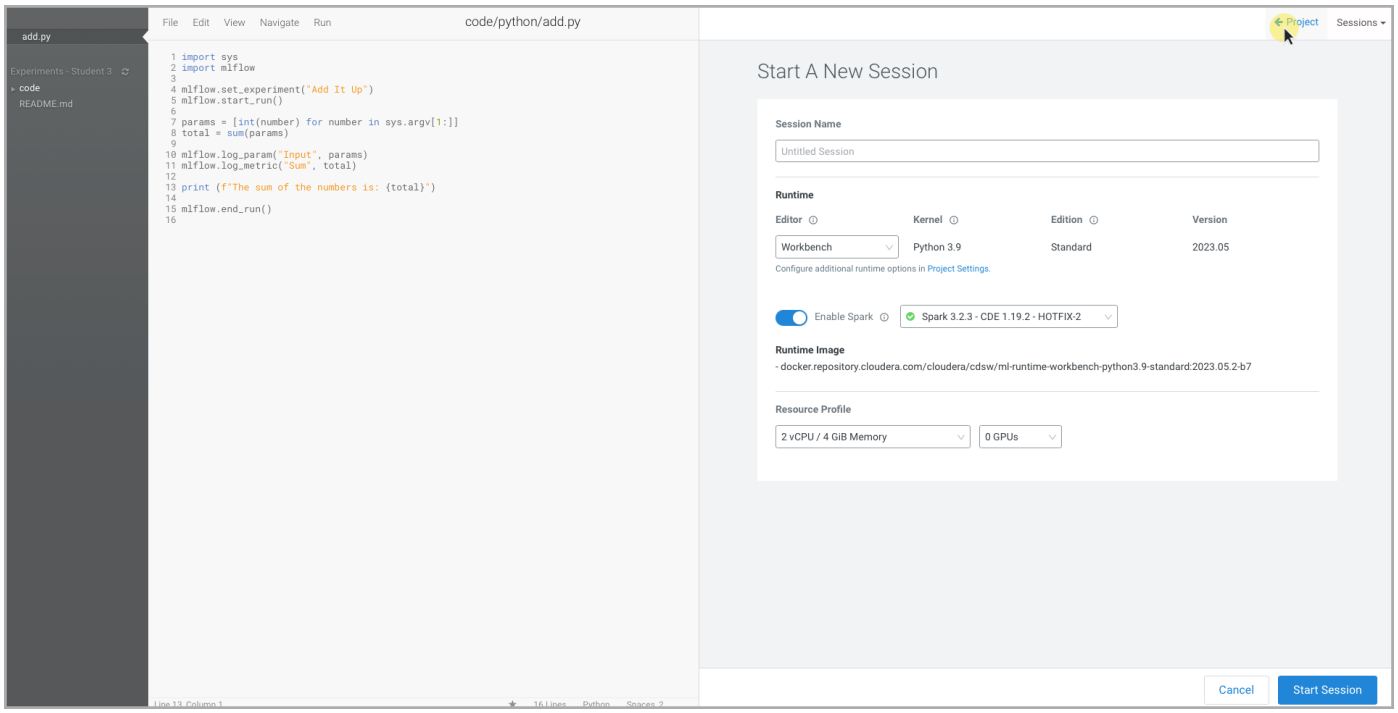
   Here is a brief description of the added lines:

- `import mlflow` - This line imports the mlflow module.
- `mlflow.start_run()` - MLflow can track multiple runs. In this case, only one run is tracked per execution. This line marks the start of the run.
- `mlflow.set_experiment("Add It Up")` - This line specifies which experiment to use. If the experiment does not exist, it will be created.
- `mlflow.log_param("Input", params)` - This line tracks the input parameters.
- `mlflow.log_metric("Sum", total)` - This line tracks the output metrics.
- `mlflow.end_run()` - This line marks the end of the run.

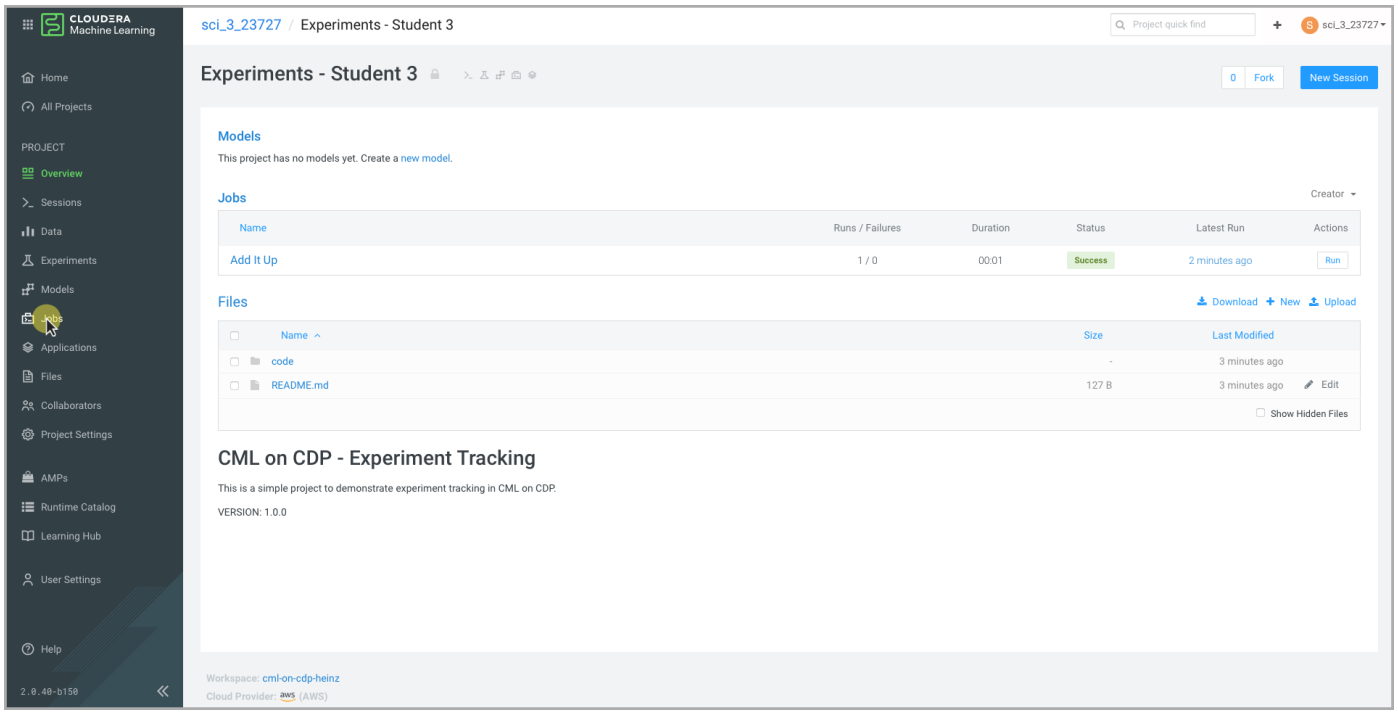
Click **File** and then **Save**.



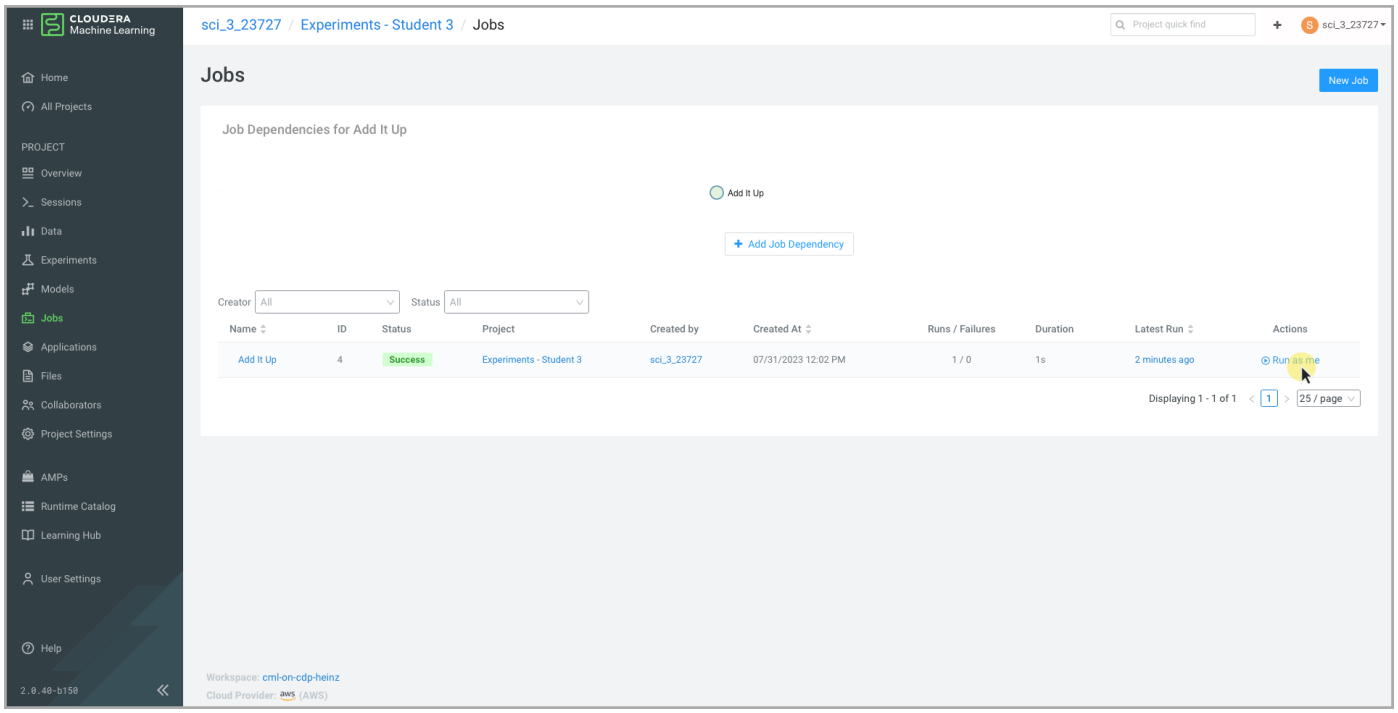
4. Click **Project** to return to the project.



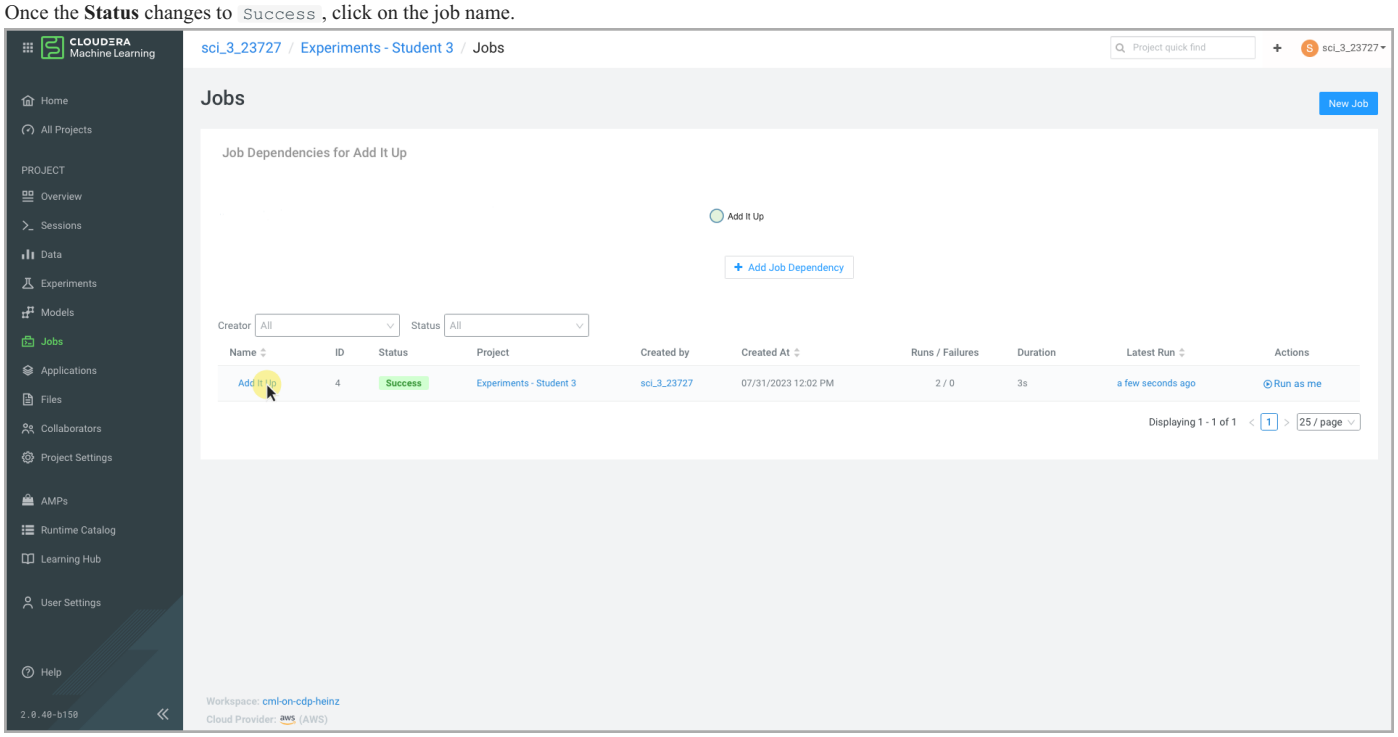
5. Click **Jobs** in the left-side menu.



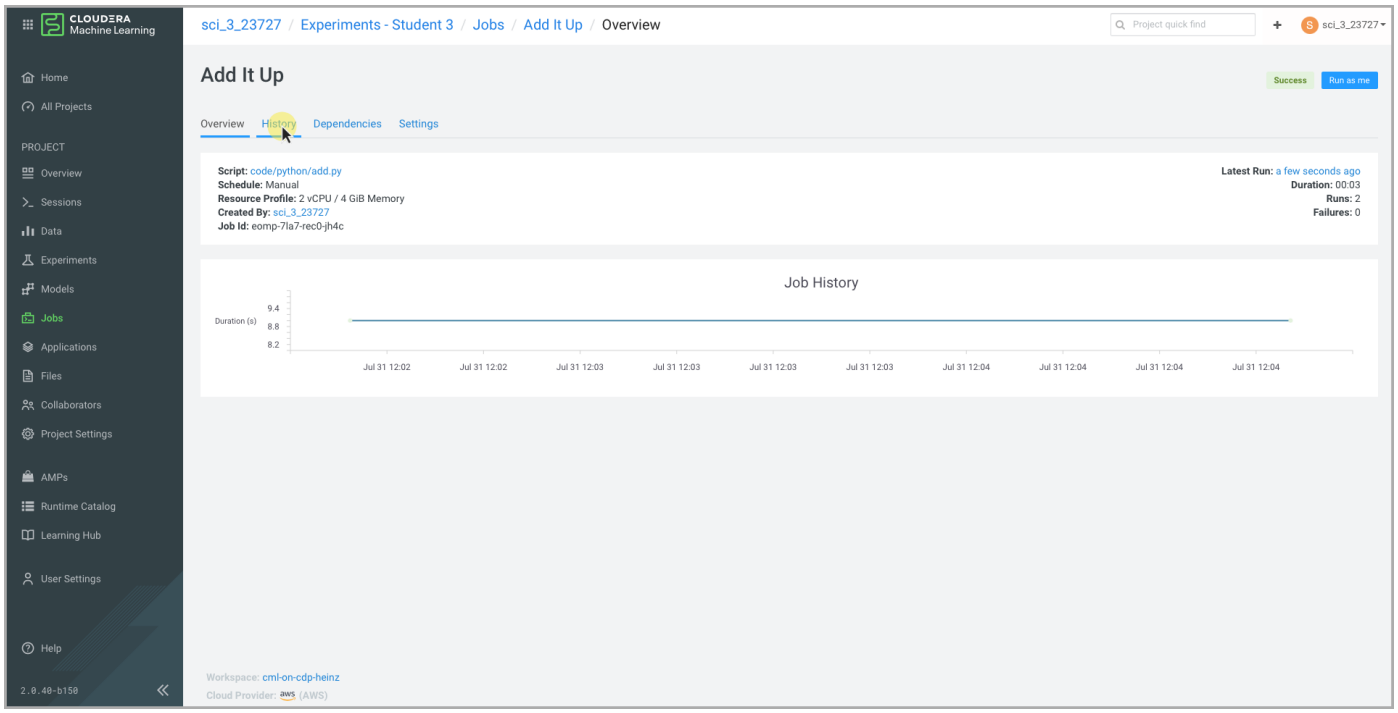
6. Click **Run as me** to execute the job again.



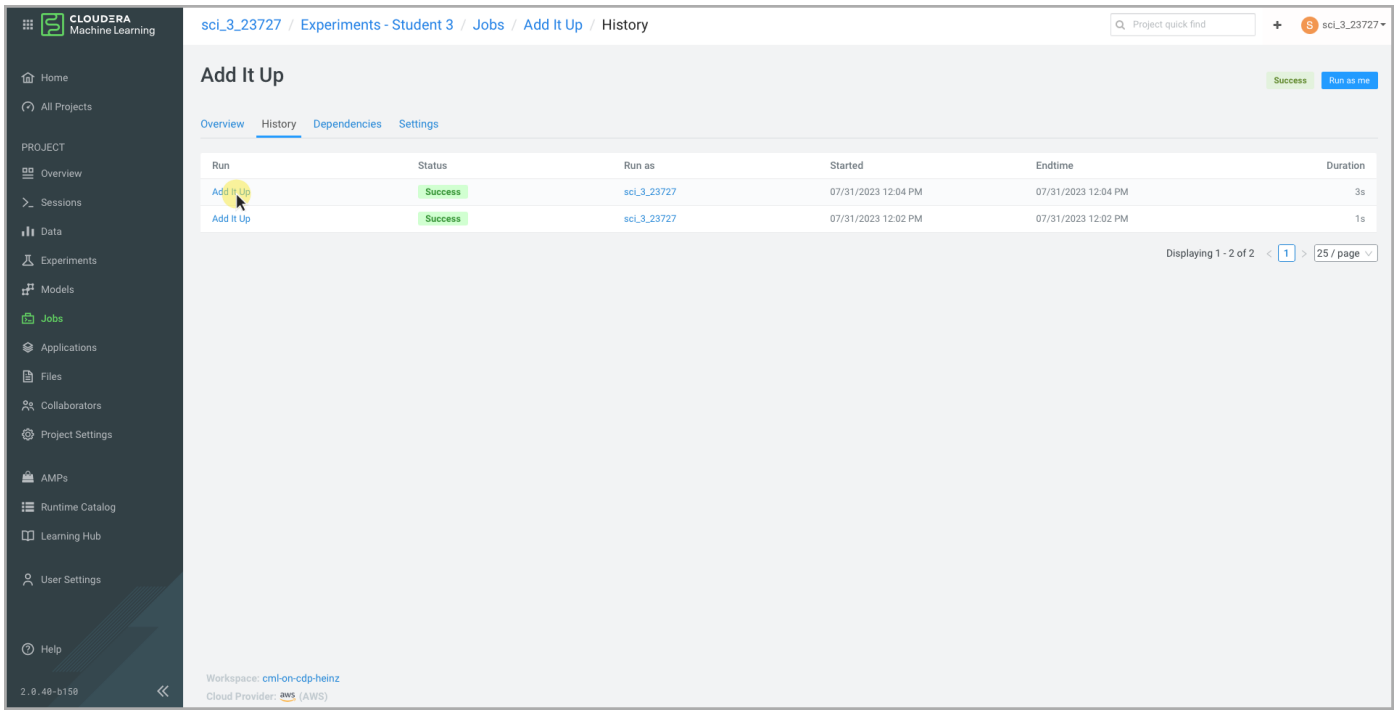
7. Once the **Status** changes to `Success`, click on the job name.



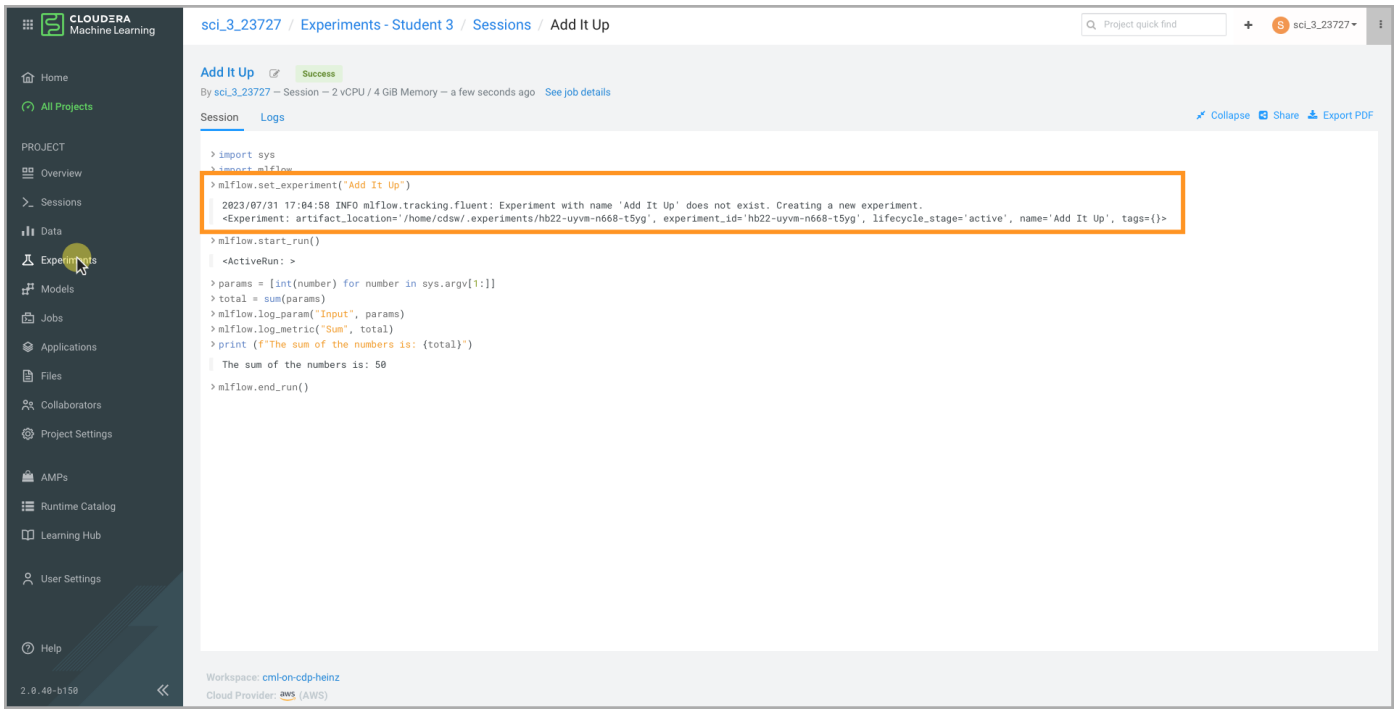
8. Click on the **History** tab.



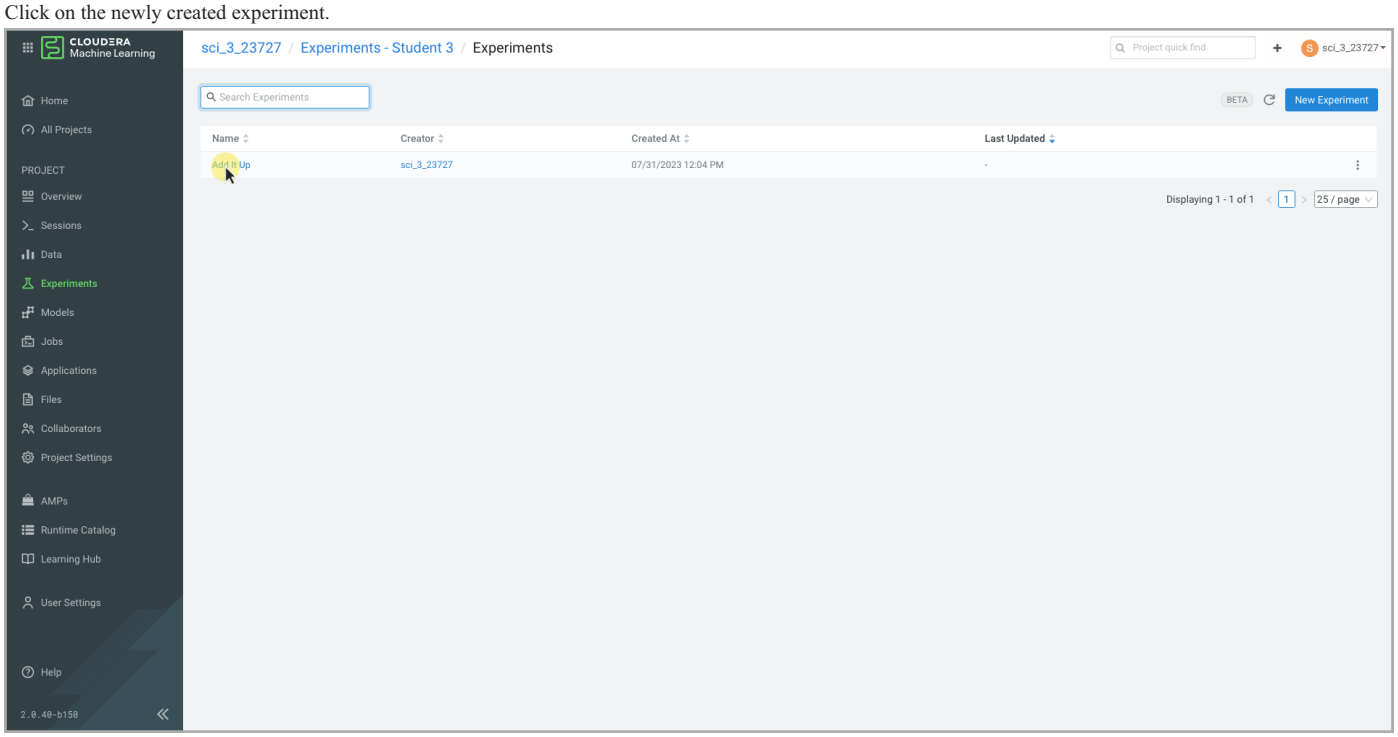
9. Click on the latest run.



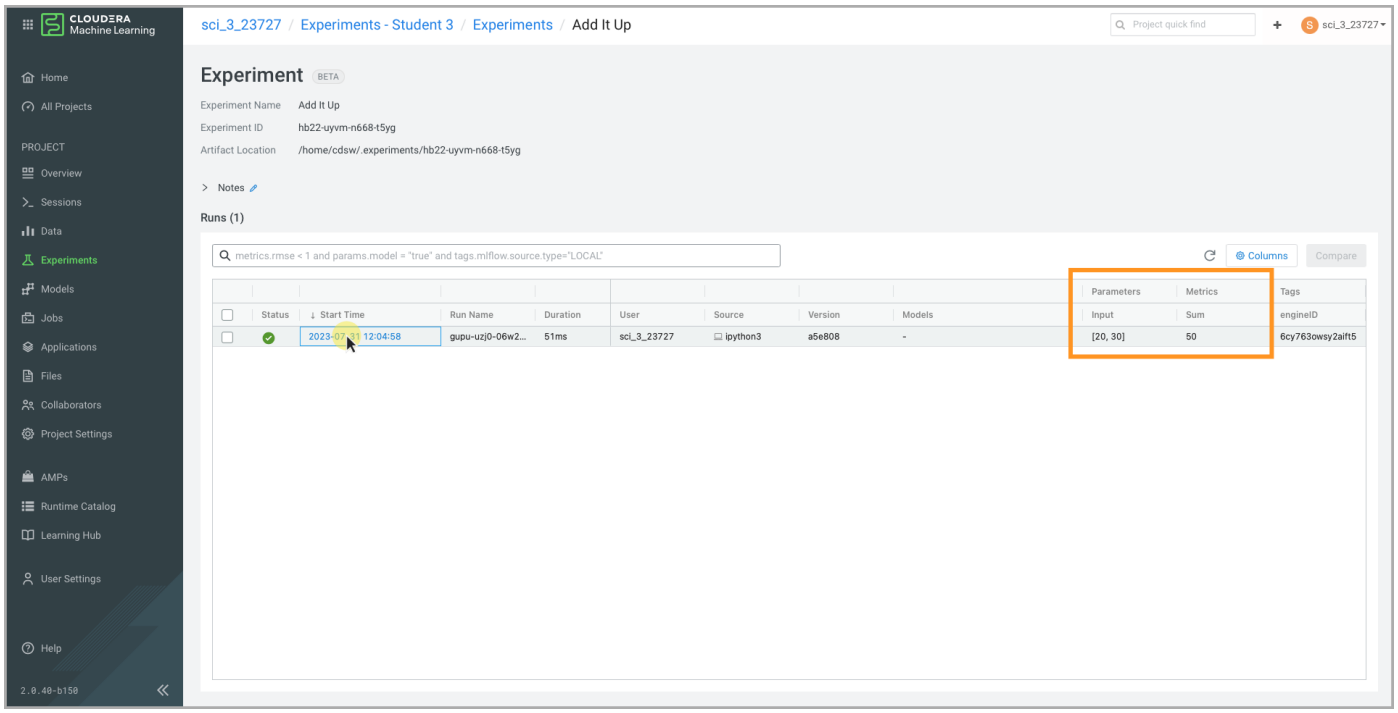
10. In the output, a message is displayed to notify you that a new experiment was created. Click **Experiments** in the left-side menu.



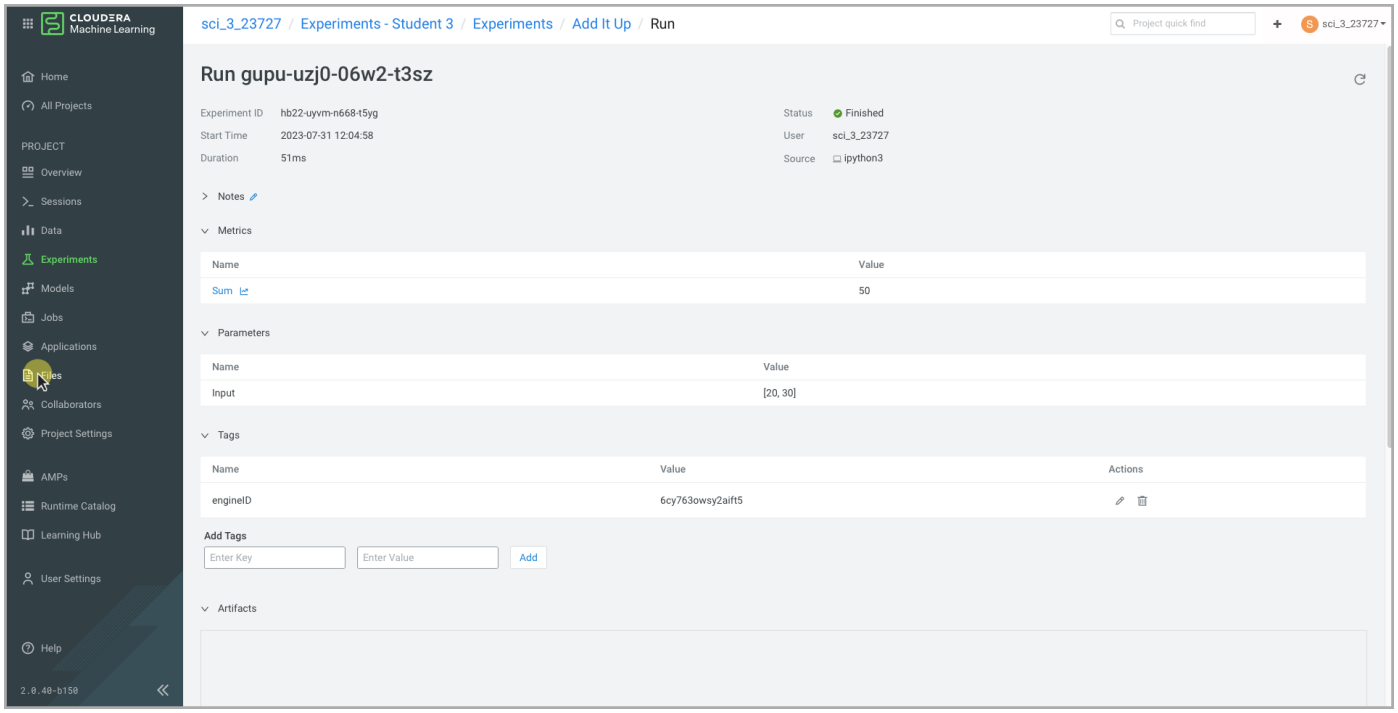
11. Click on the newly created experiment.



12. Information about the experiment and a list of runs is displayed. The parameters and metrics for the run are displayed. Click on the run.



13. A more detailed view of the run is displayed. Notice that in addition to the parameters and metrics, a run can have more information like notes, tags, and artifacts. Click **Files** in the left-side menu.



14. Navigate to `add.py` and click **Open in Session**. Add the following code to create a new metric called `count` and a new artifact, the `result.txt` file:

```
import sys
import mlflow

mlflow.set_experiment("Add It Up")
mlflow.start_run()

params = [int(number) for number in sys.argv[1:]]
count = len(params)
total = sum(params)

with open("result.txt", "w") as output_file:
    output_file.write(f"Input: {params},")
    output_file.write(f"Count: {count},")
    output_file.write(f"Sum: {total}\n")
    output_file.close()

mlflow.log_param("Input", params)
mlflow.log_metric("Count", count)
mlflow.log_metric("Sum", total)

mlflow.log_artifact("result.txt")

print (f"The sum of the numbers is: {total}")

mlflow.end_run()
```

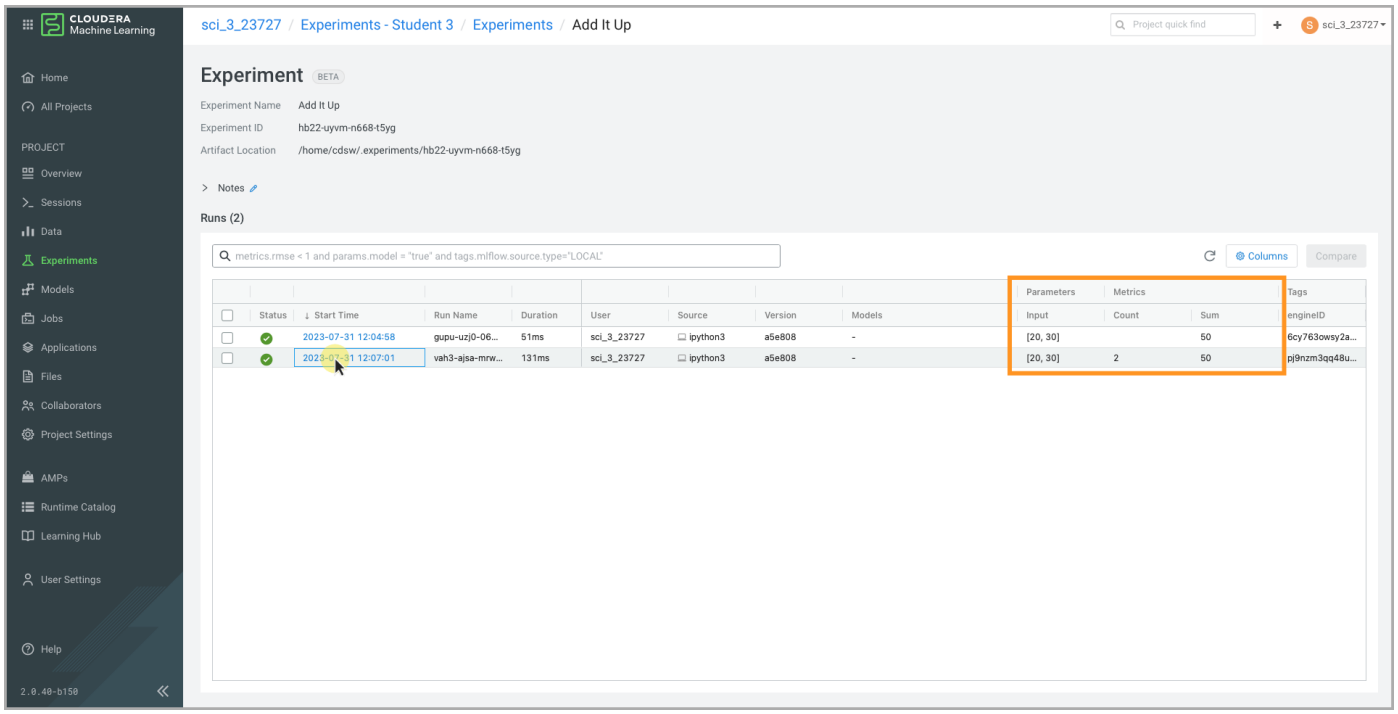
Here is a brief description of the added lines: * `count = len(params)` - This is a new metric for tracking the number of numbers to sum.

- `with open("result.txt", "w") ...` - This line and the following four lines create a file (artifact) with the results of the run.
- `mlflow.log_metric("Count", count)` - This line tracks the new count metric.
- `mlflow.log_artifact("result.txt")` - This line tracks the new artifact.

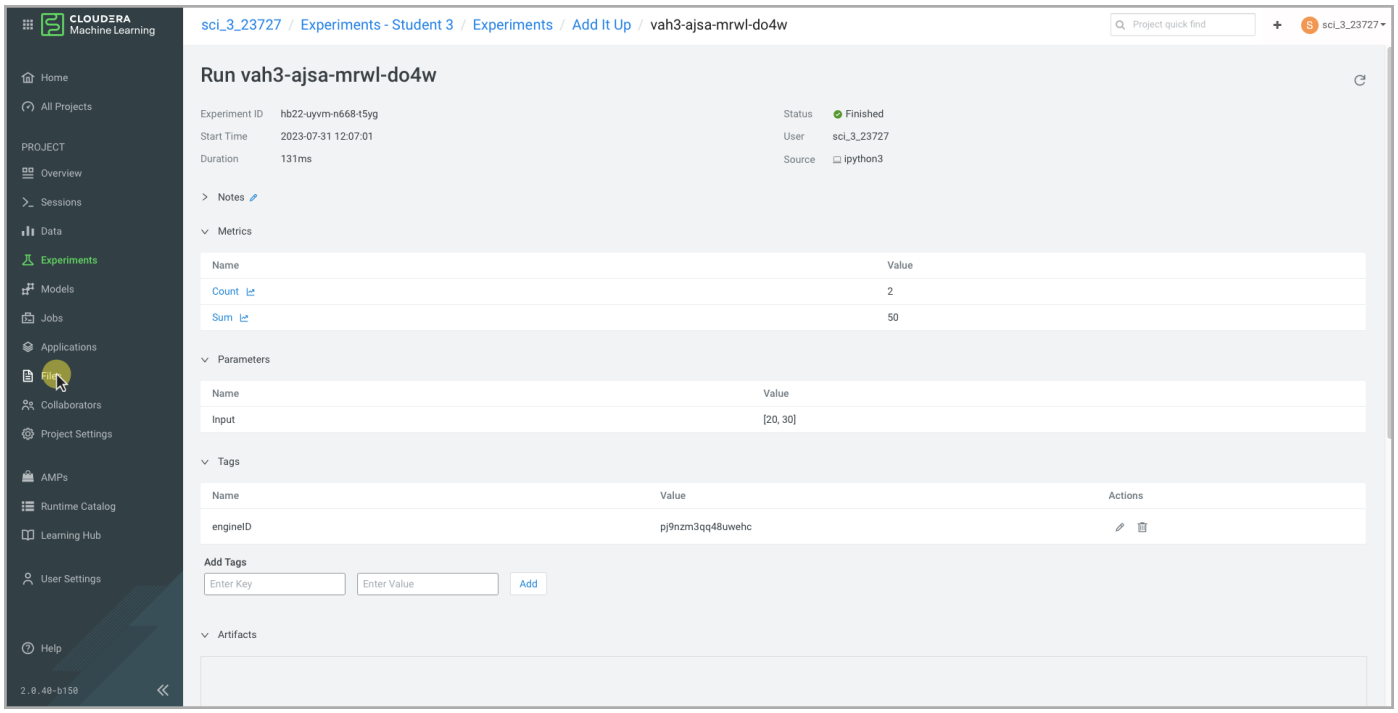
Save the file and click **Project** to return to the project.

15. Click **Jobs** in the left-side menu and click **Run as me** on the job.
16. Click **Experiments** in the left-side menu and click the experiment in the list.

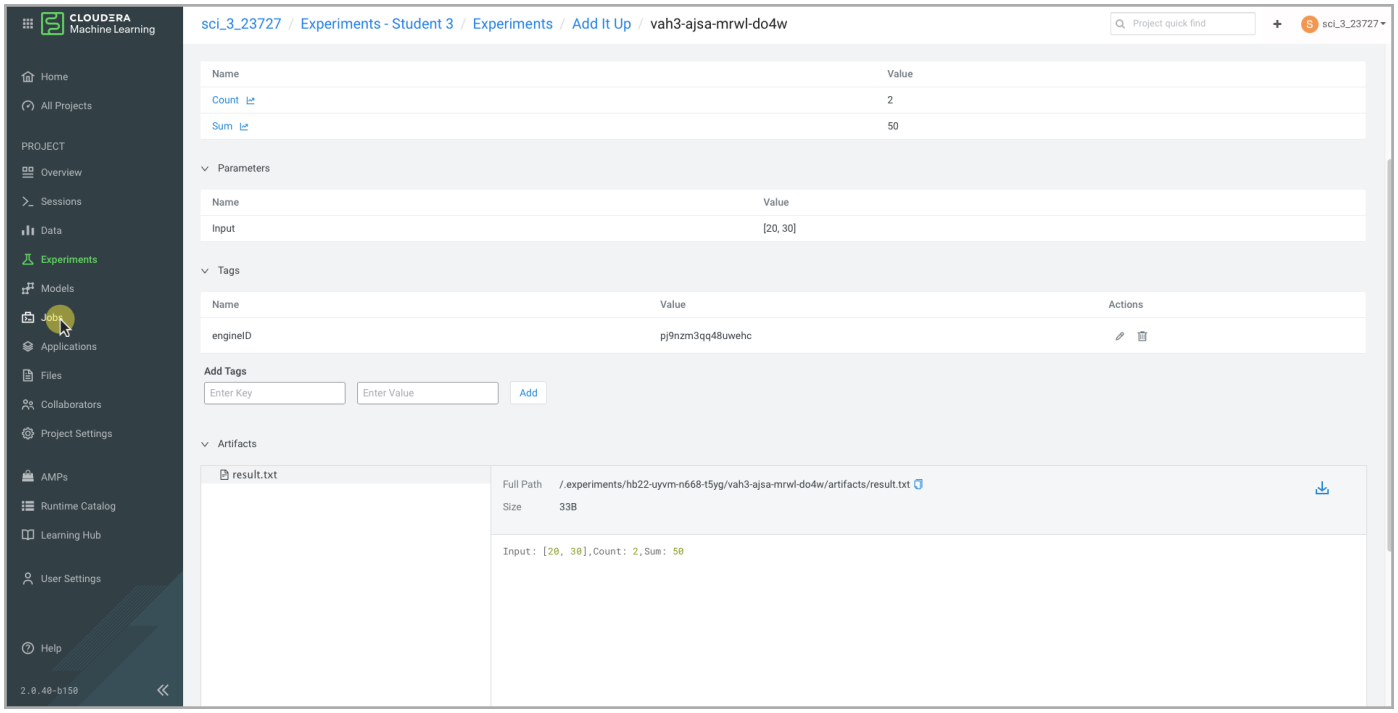
17. Notice, the list of runs includes the new **Count** metric. Click on the latest run.



18. The run details includes the new metric too.

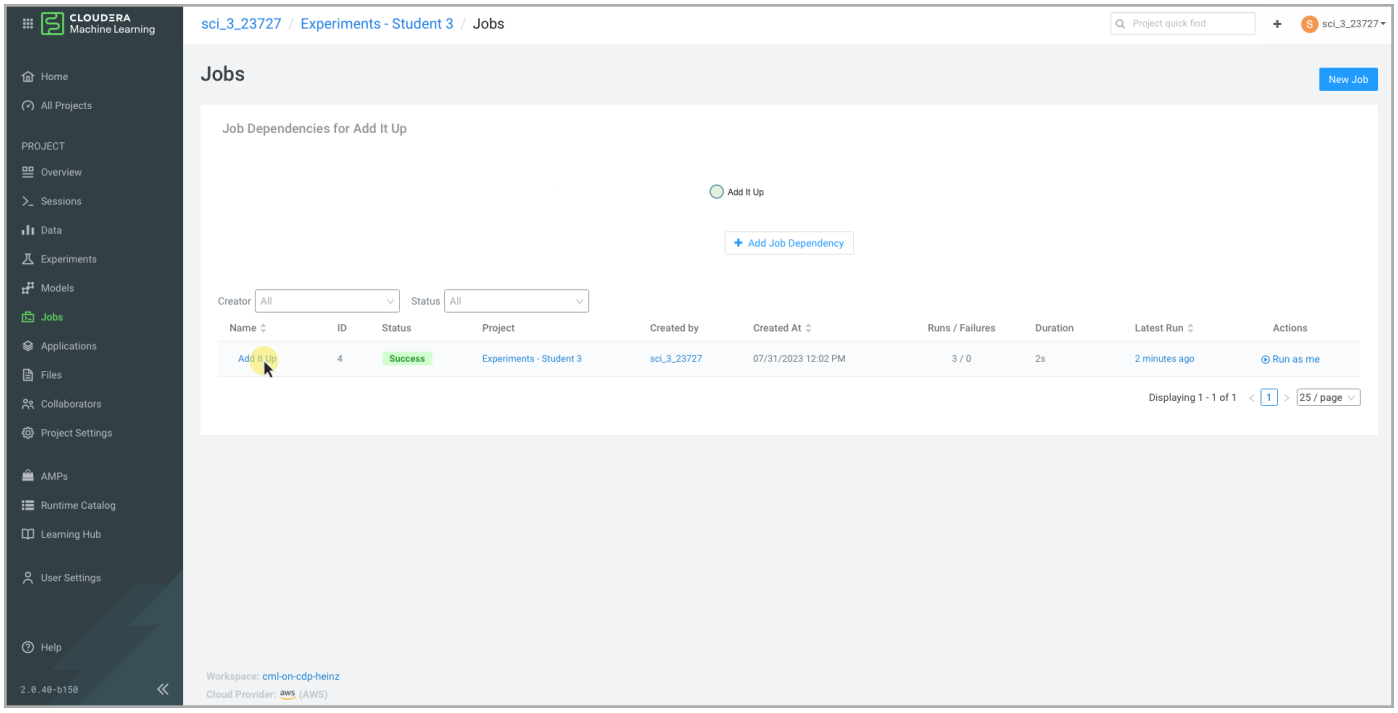


19. Scroll down, if needed, and view the Artifacts. If the job recently completed, the Artifacts may not have updated. Wait a minute and refresh the page, if you do not see the `result.txt` file in the list of Artifacts.

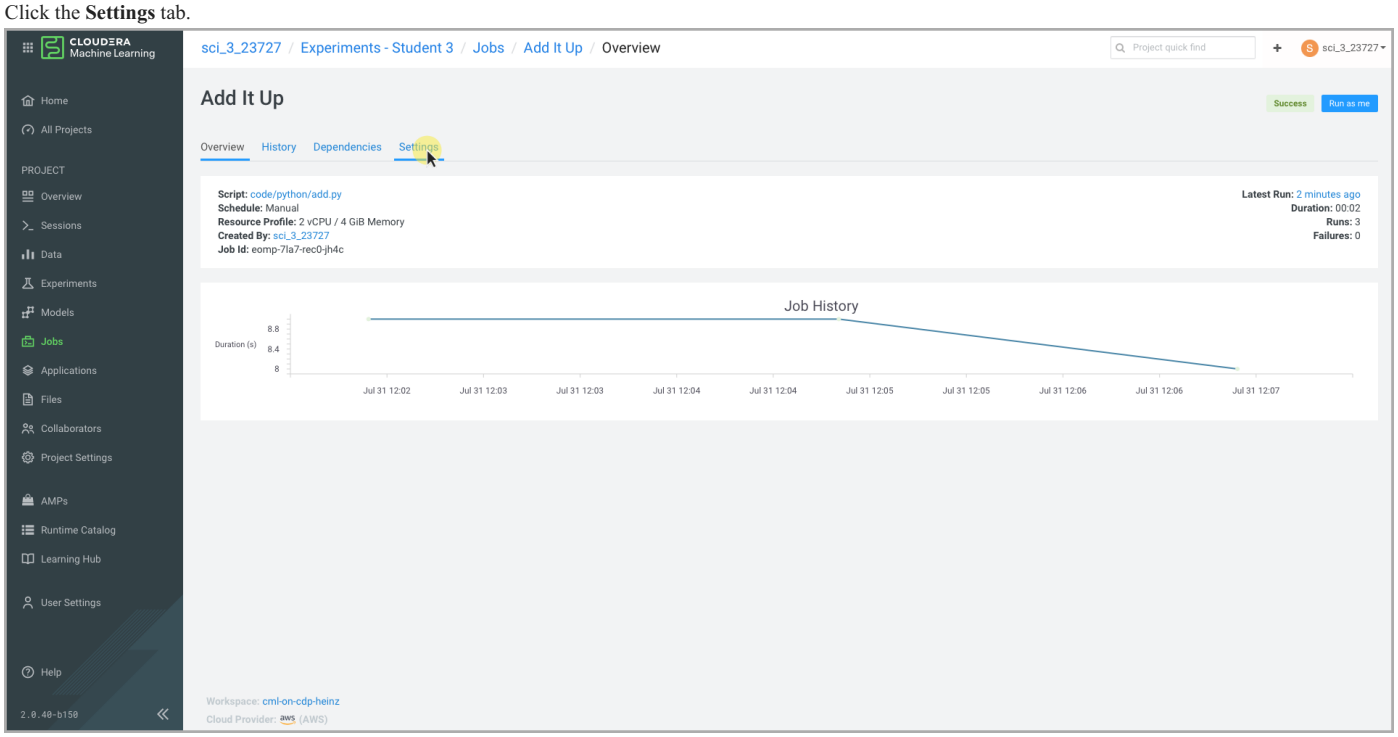


Change the Input and Compare Runs

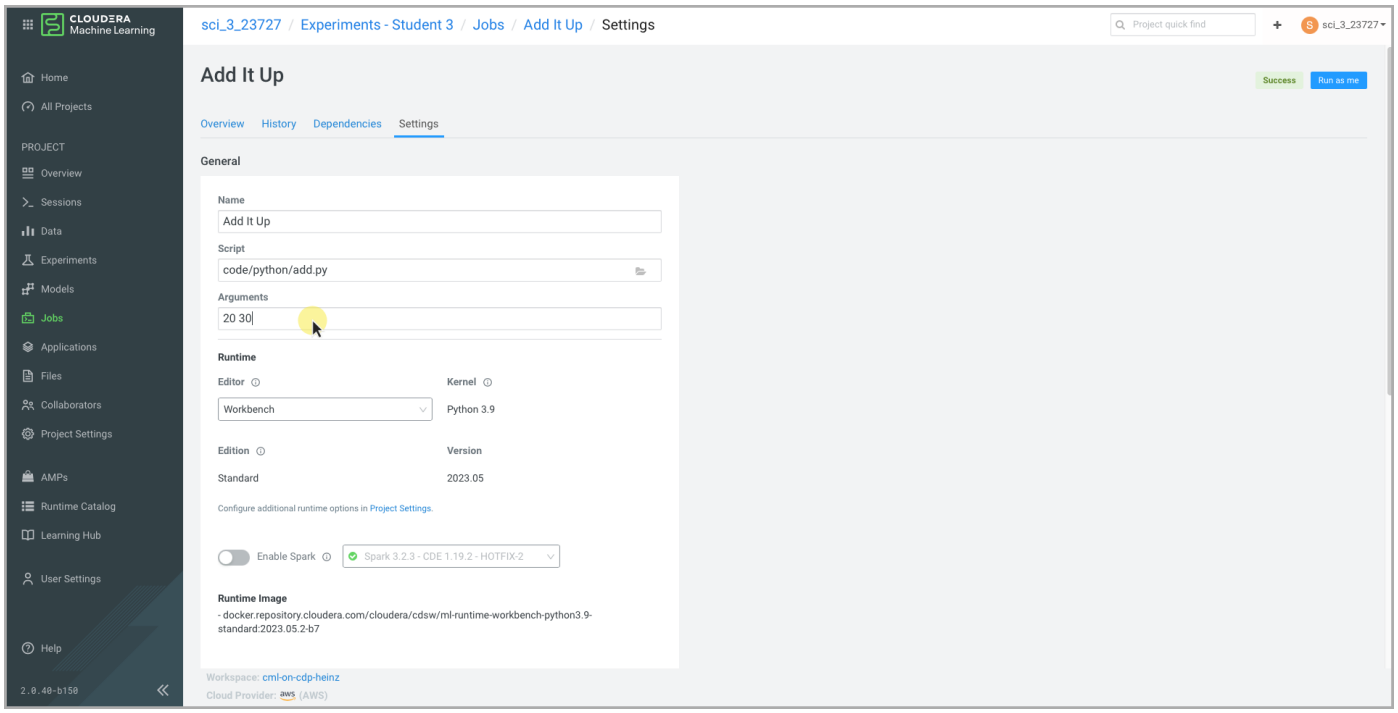
1. Click **Jobs** in the left-side menu and click the `Add It Up` job in the list.



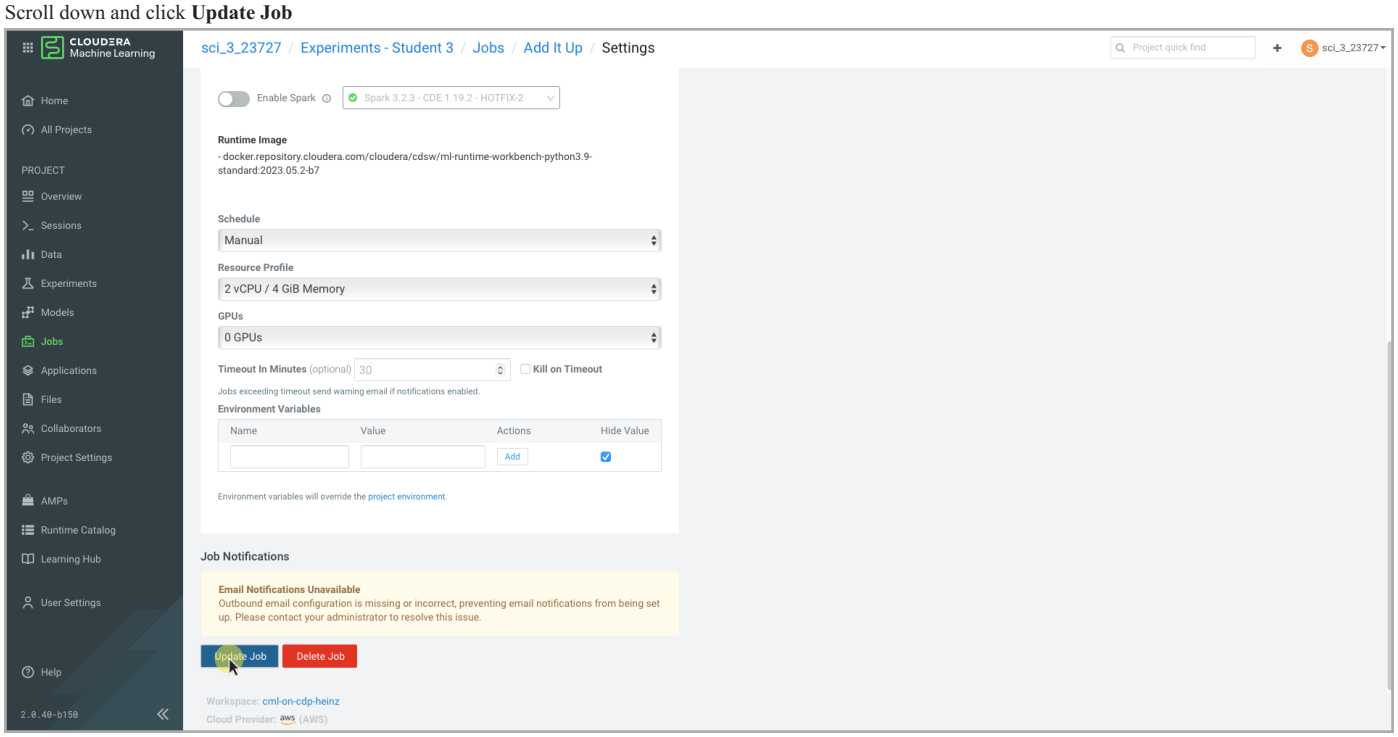
2. Click the **Settings** tab.



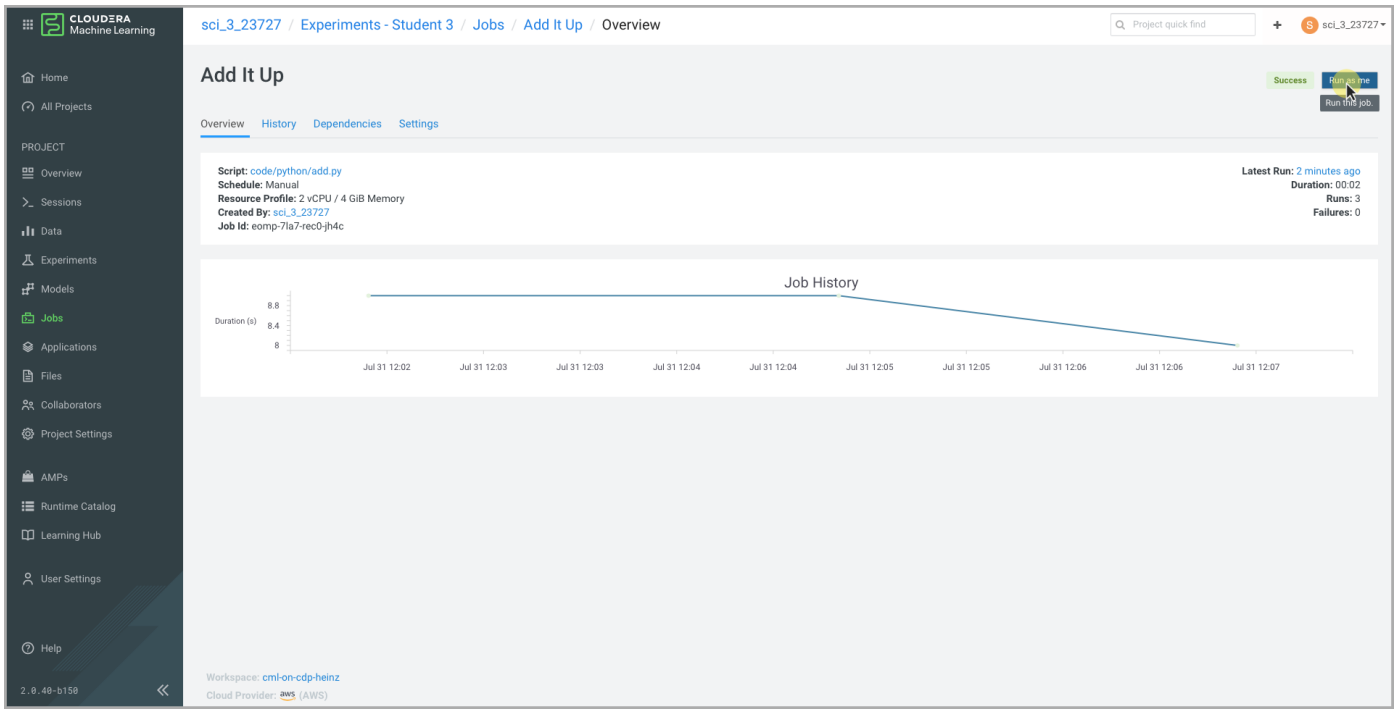
3. Edit the Arguments to be something different, like `20 30 40`.



4. Scroll down and click **Update Job**

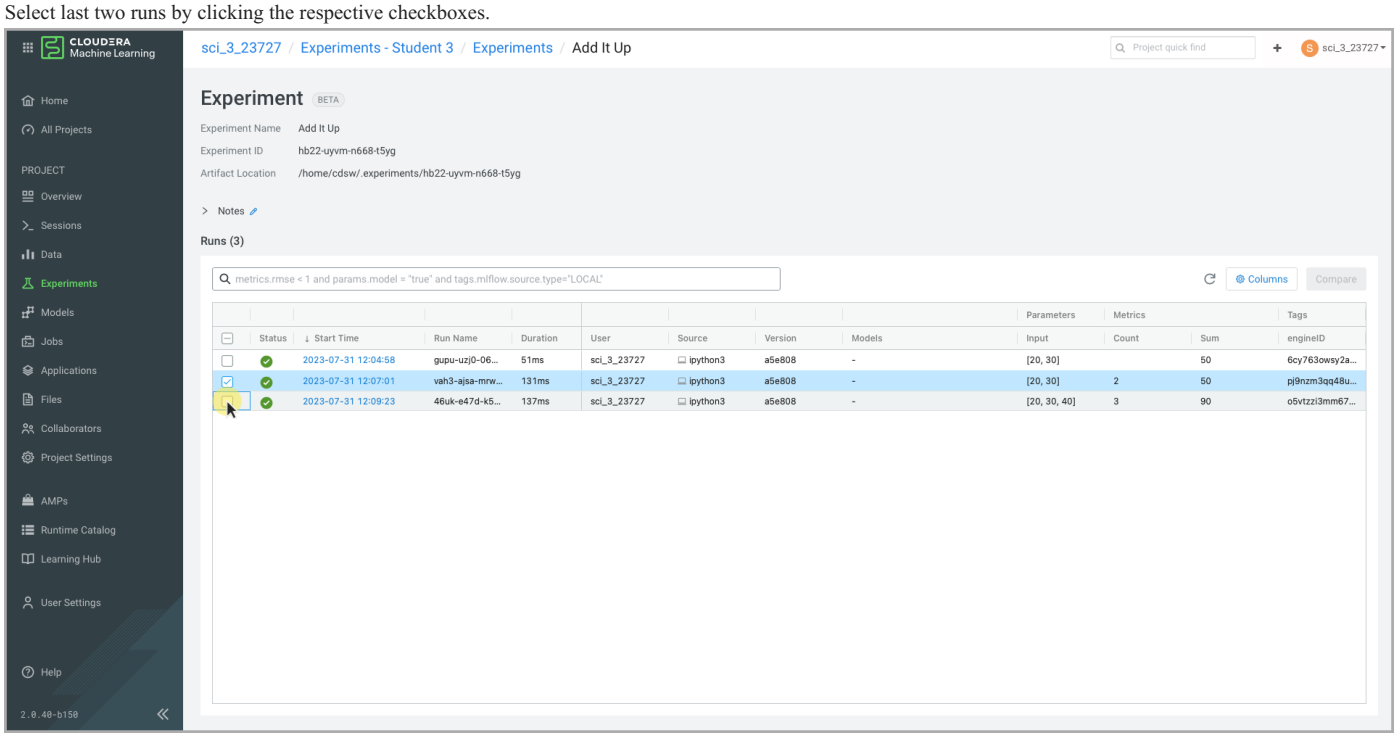


5. Click **Run as me**.

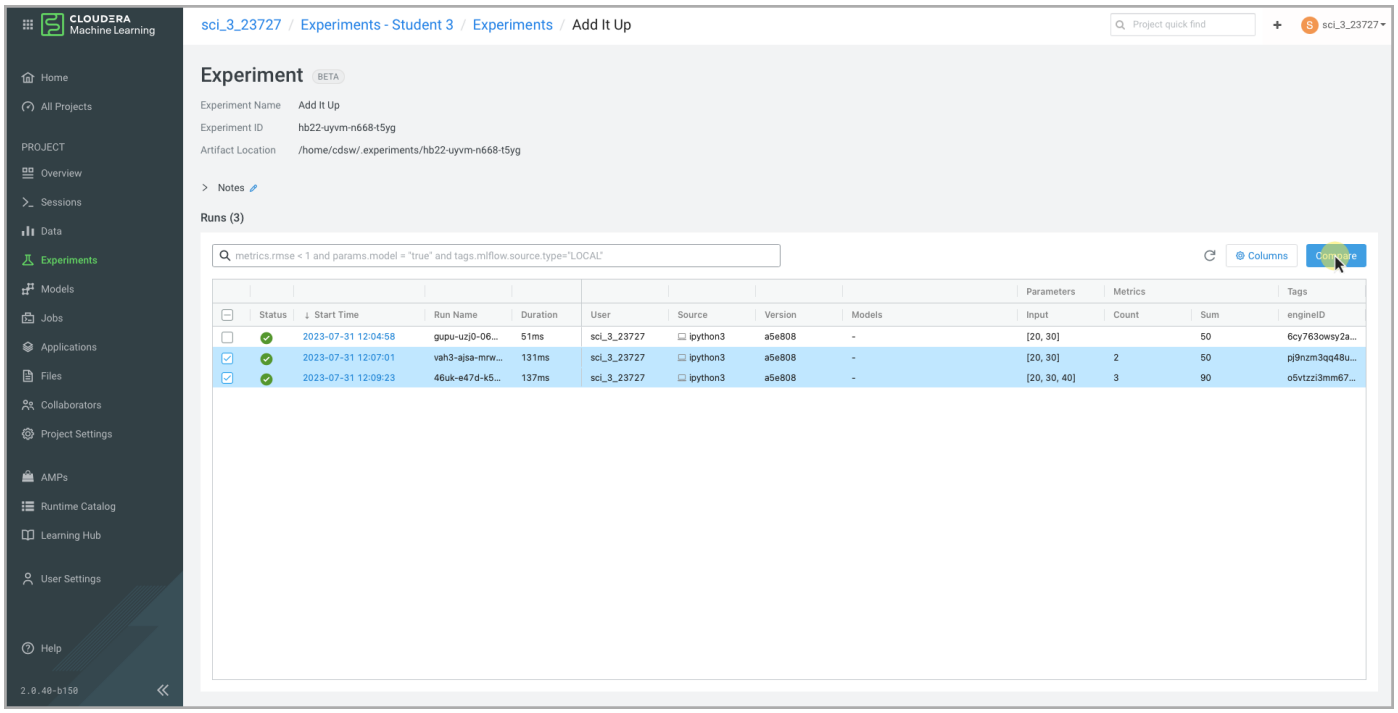


6. Click **Experiments** in the left-side menu and click on the `Add It Up` experiment.

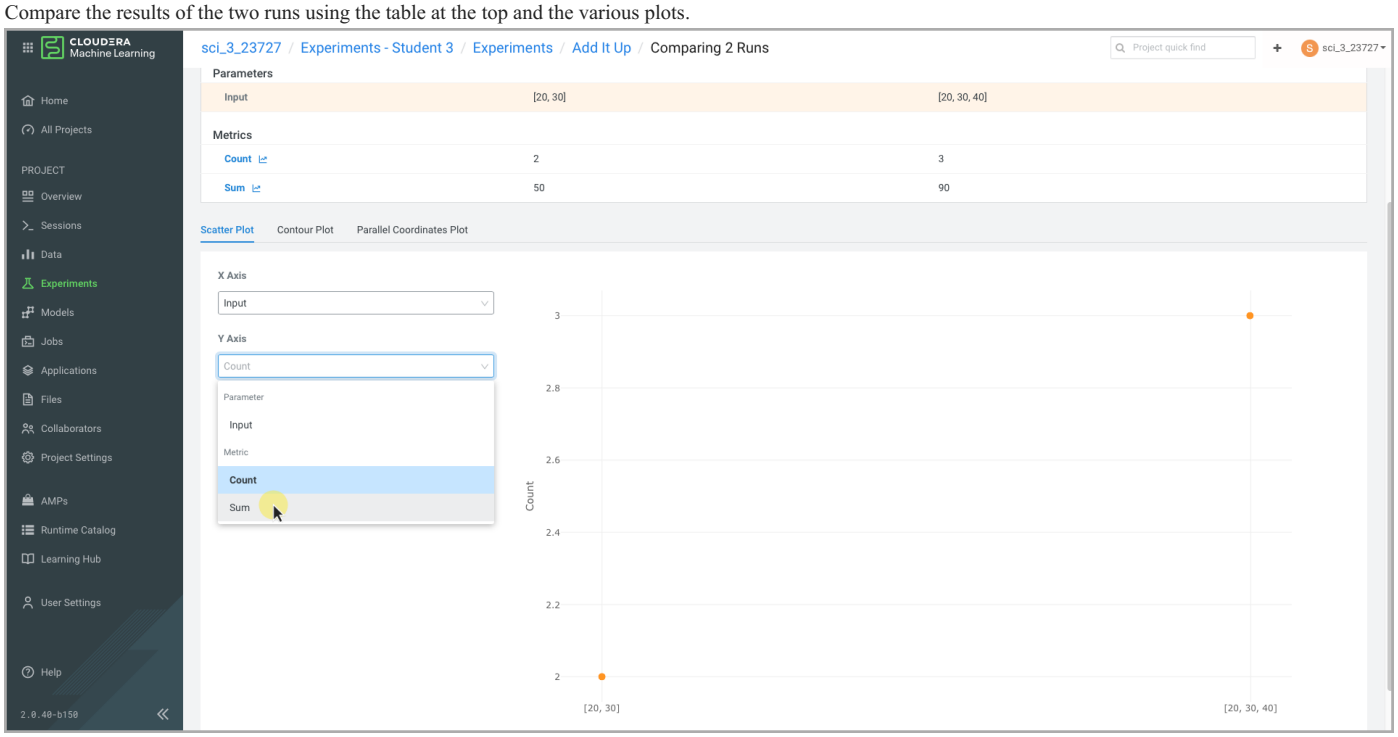
7. Select last two runs by clicking the respective checkboxes.



8. Click **Compare**.



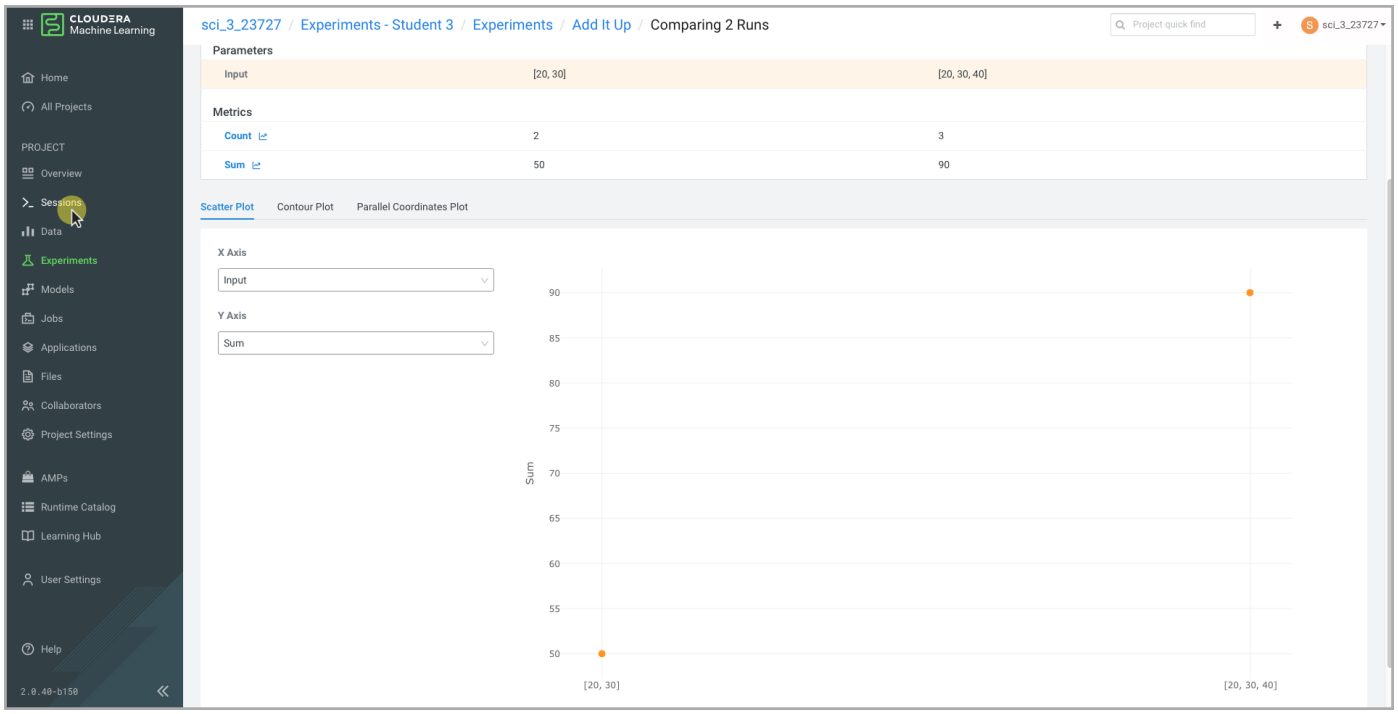
9. Compare the results of the two runs using the table at the top and the various plots.



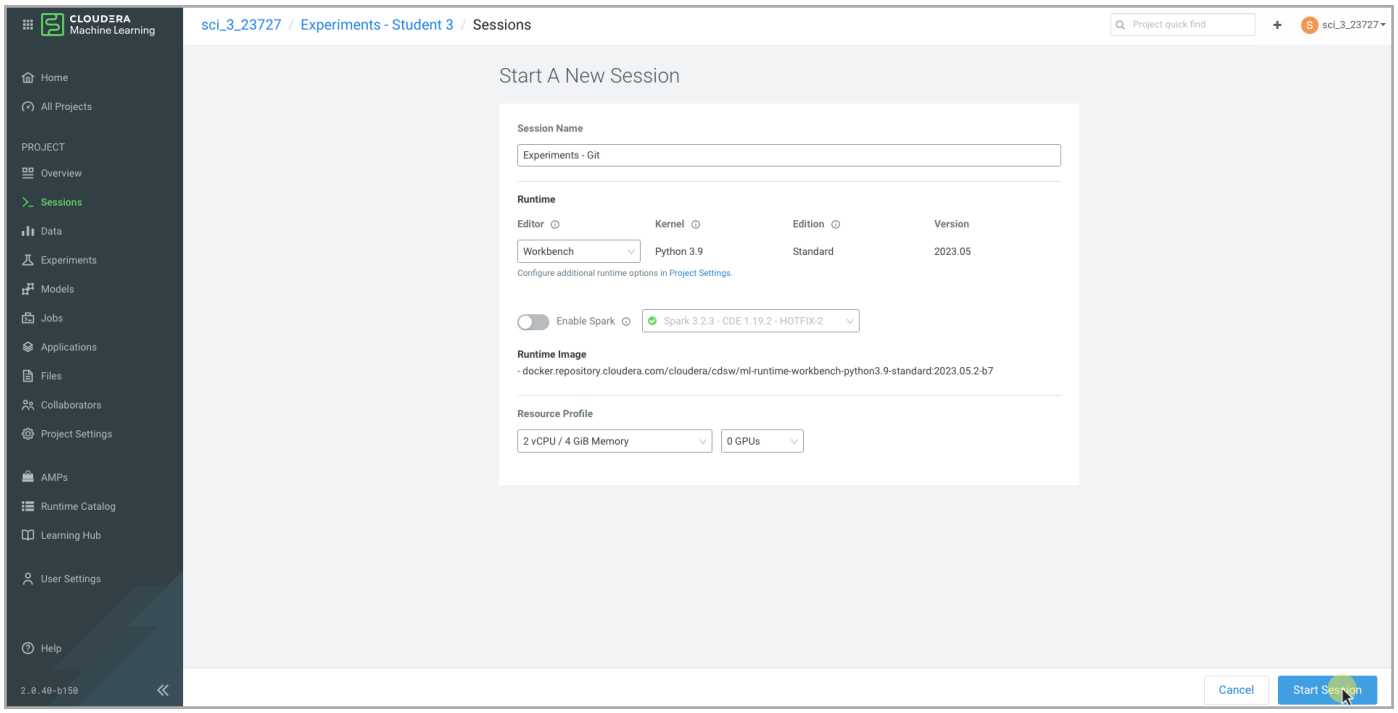
Commit the Changes to Git

Now that you have modified the `add.py` code to track metrics, you will commit the changes to git so they won't be lost.

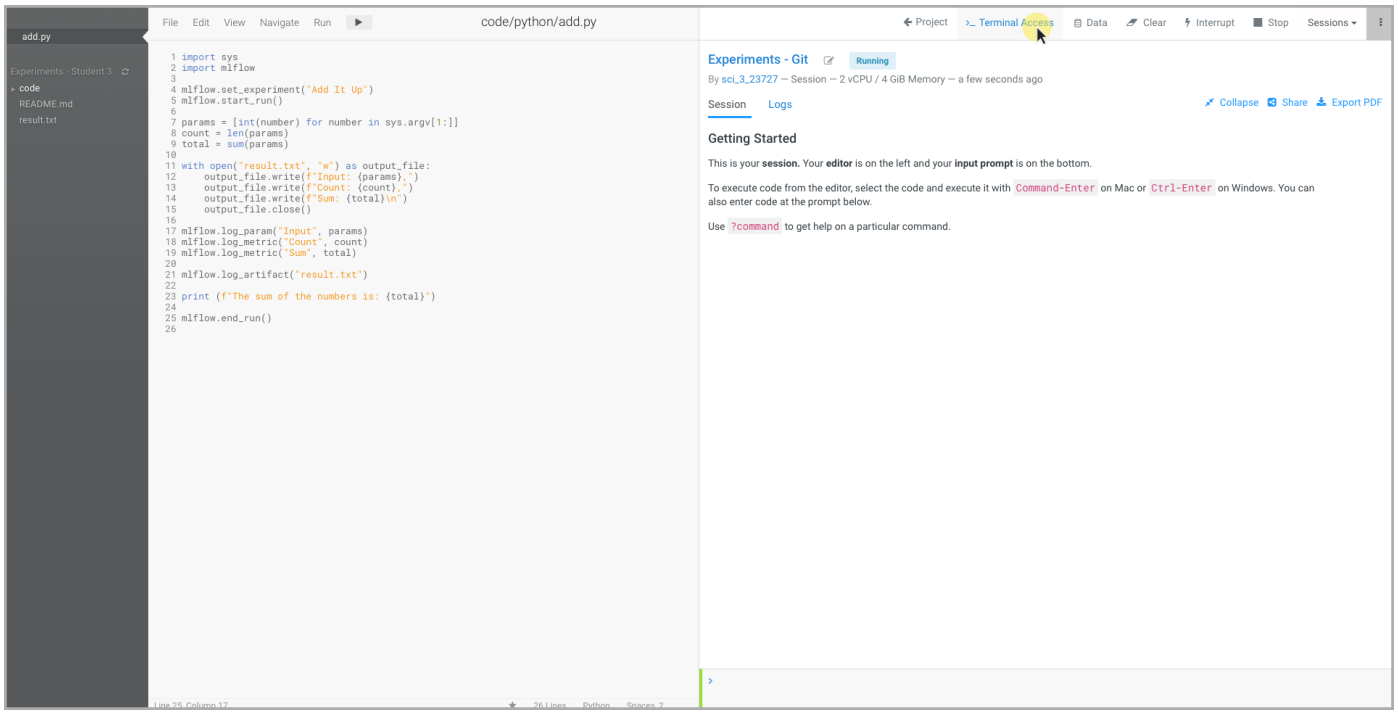
1. Click **Sessions** in the left-side menu and click **New Session**.



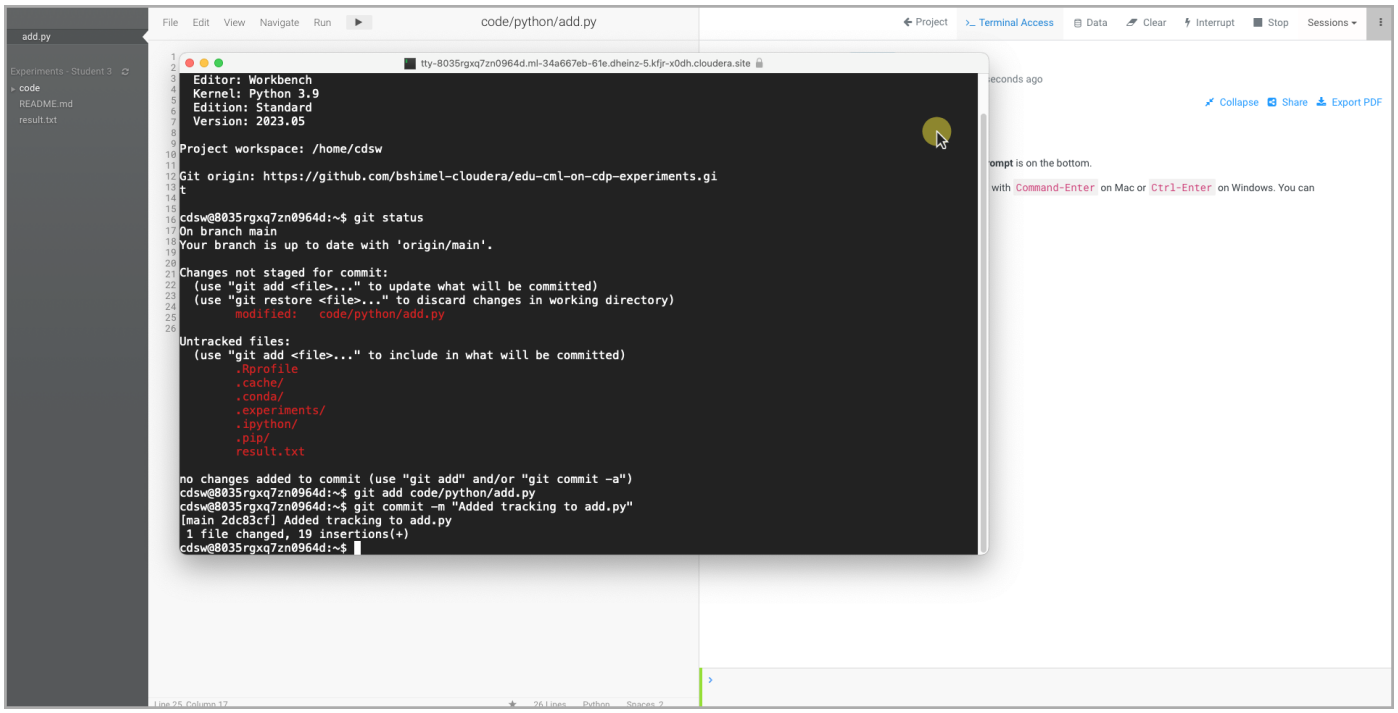
2. Enter a session name and click **Start Session**.



3. Once the session has started, click the **Terminal Access** button.



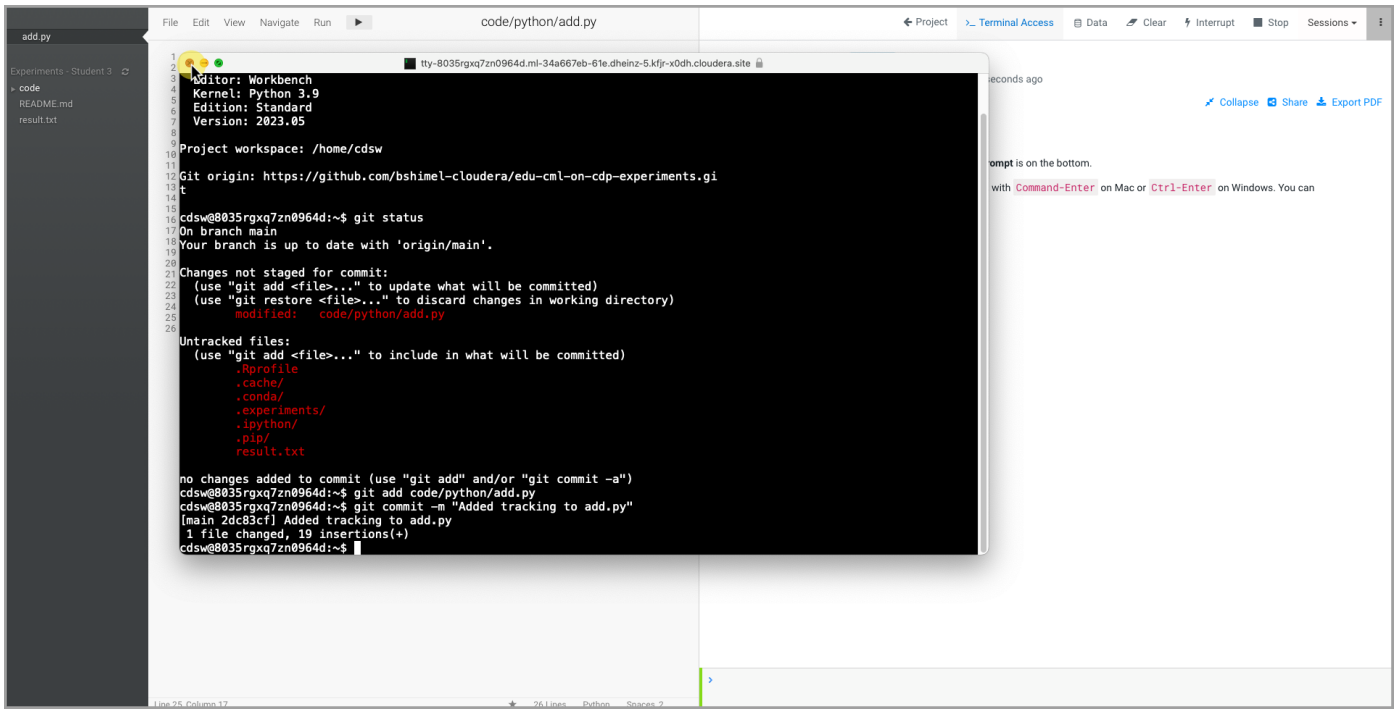
4. Enter `git status` and press **Enter**. You will see `add.py` has been modified.



5. Add the file to git and commit the changes.

```
git add code/python/add.py
```

```
git commit -m "Added tracking to add.py"
```



Your file changes have been committed. Normally, you would push the changes back to a remote repository. However, in this case, you do not have permission to commit to the remote repository.

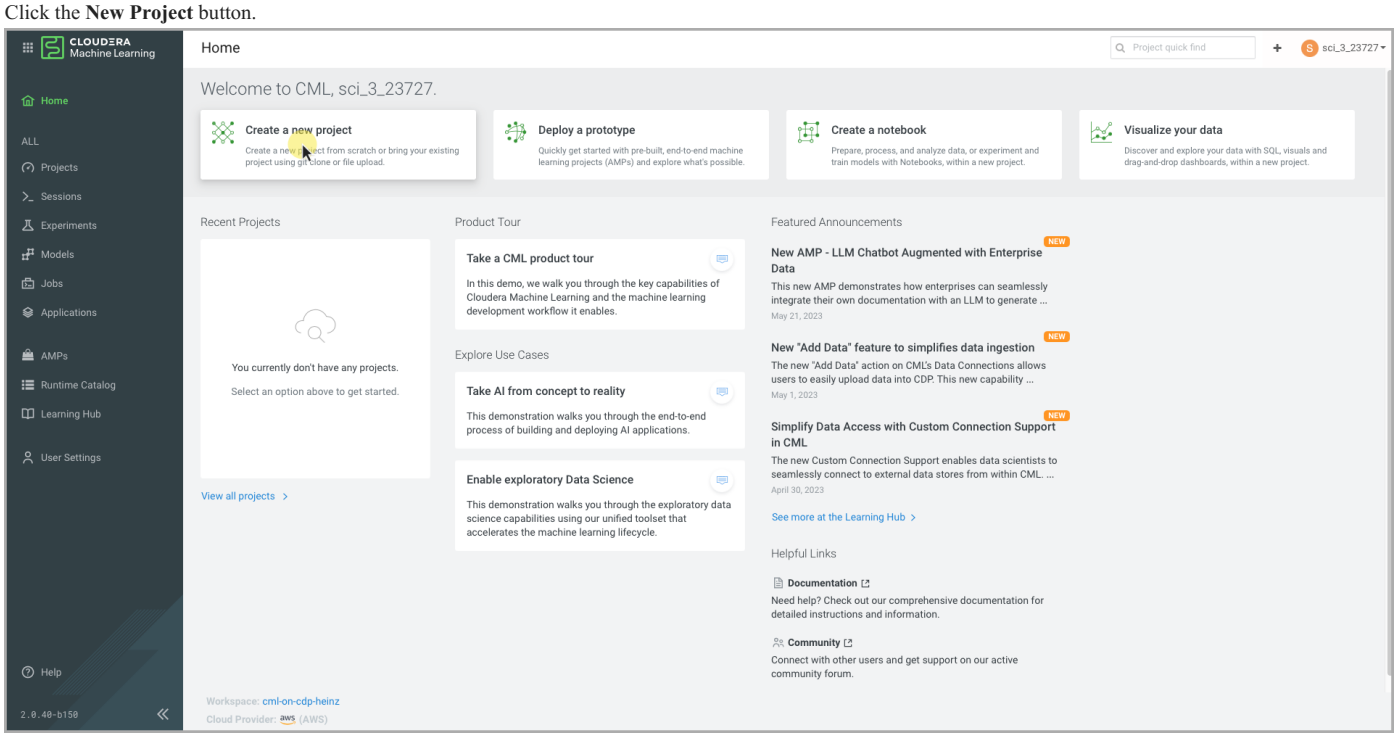
End of Exercise

Using Workbench for Lecture and Exercises

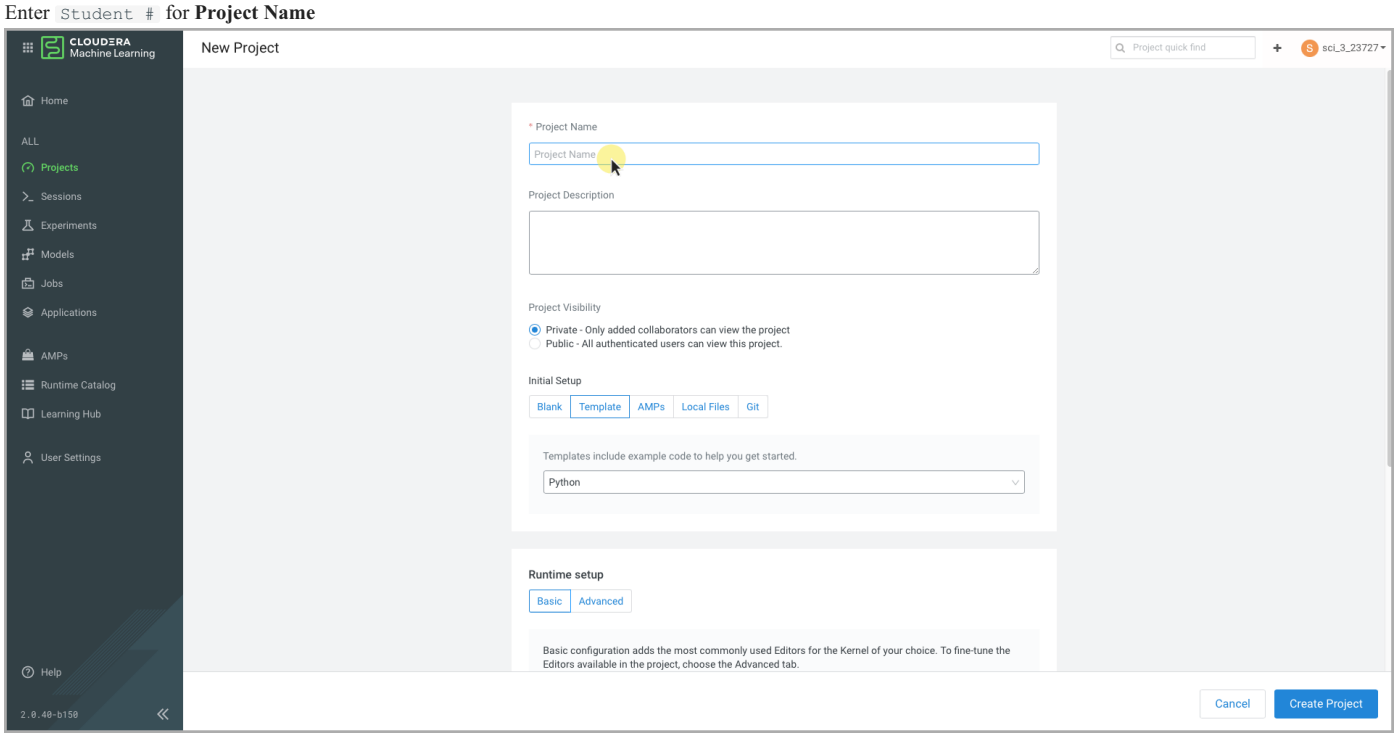
The next several lectures and exercises will be completed using the CML Workbench. The files for the lectures and exercises are provided as an AMP. In this brief exercise, you will create a new project based on the AMP.

1. Navigate to your Cloudera Machine Learning workspace.

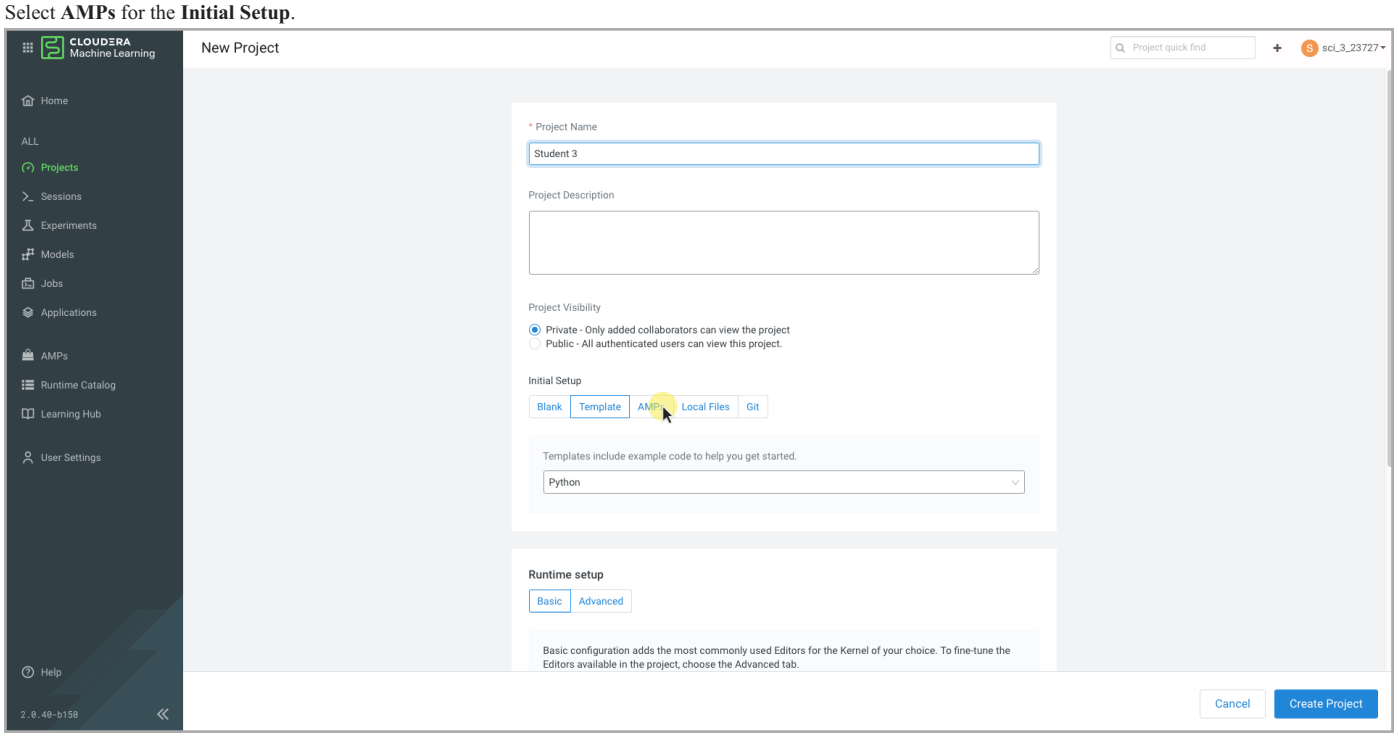
2. Click the **New Project** button.



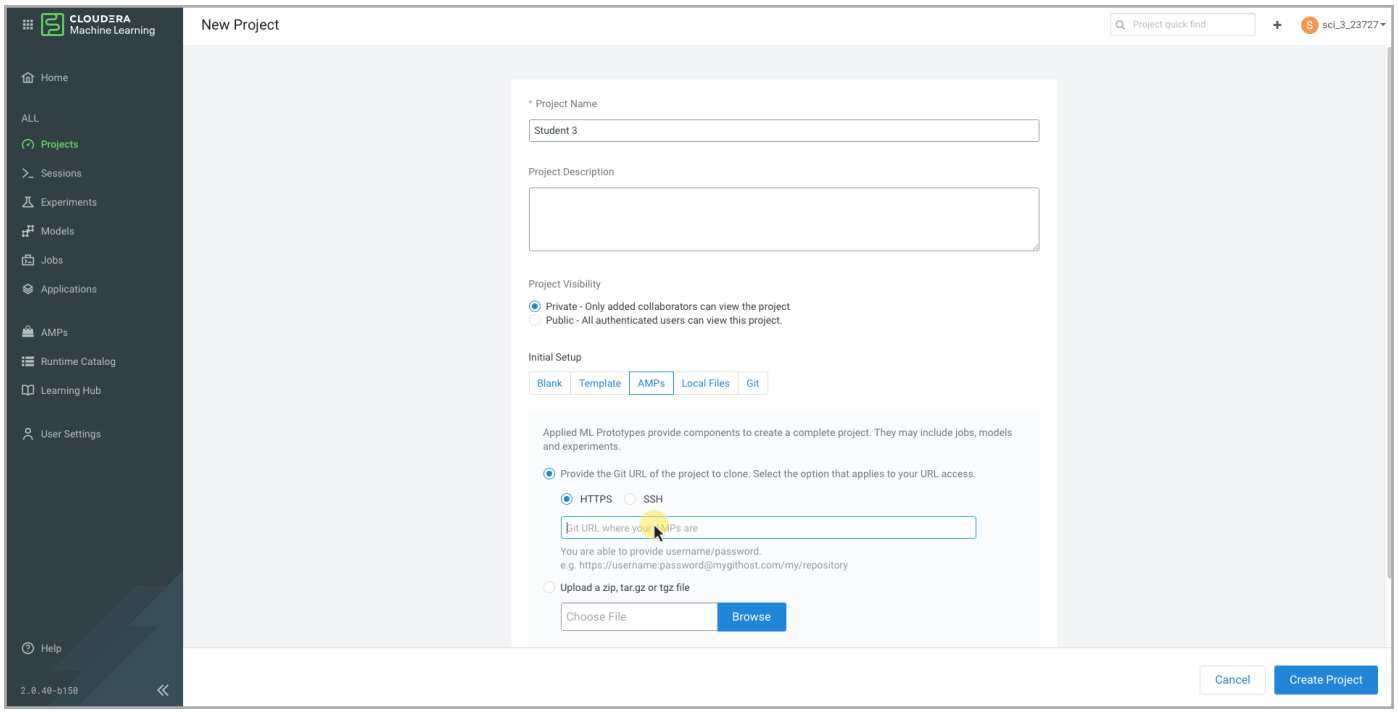
3. Enter `Student #` for **Project Name**



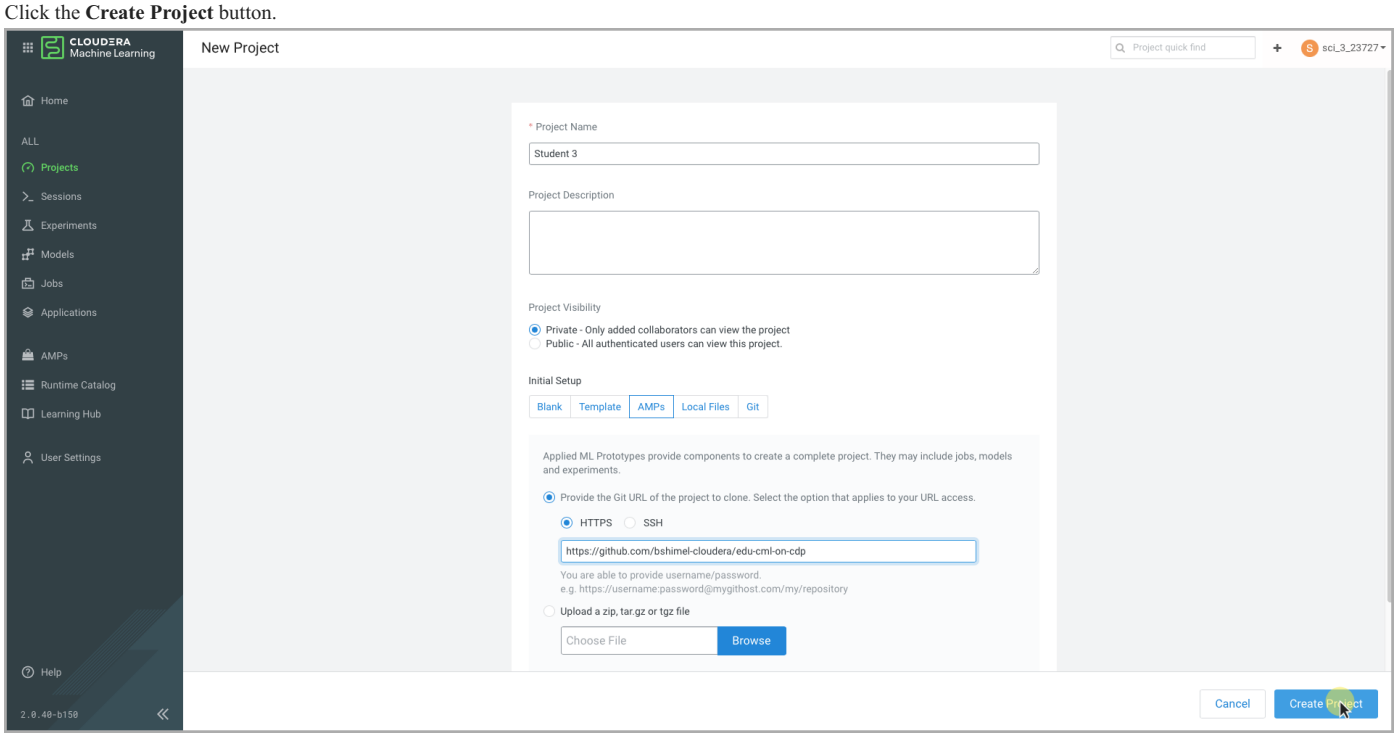
4. Select **AMPs** for the **Initial Setup**.



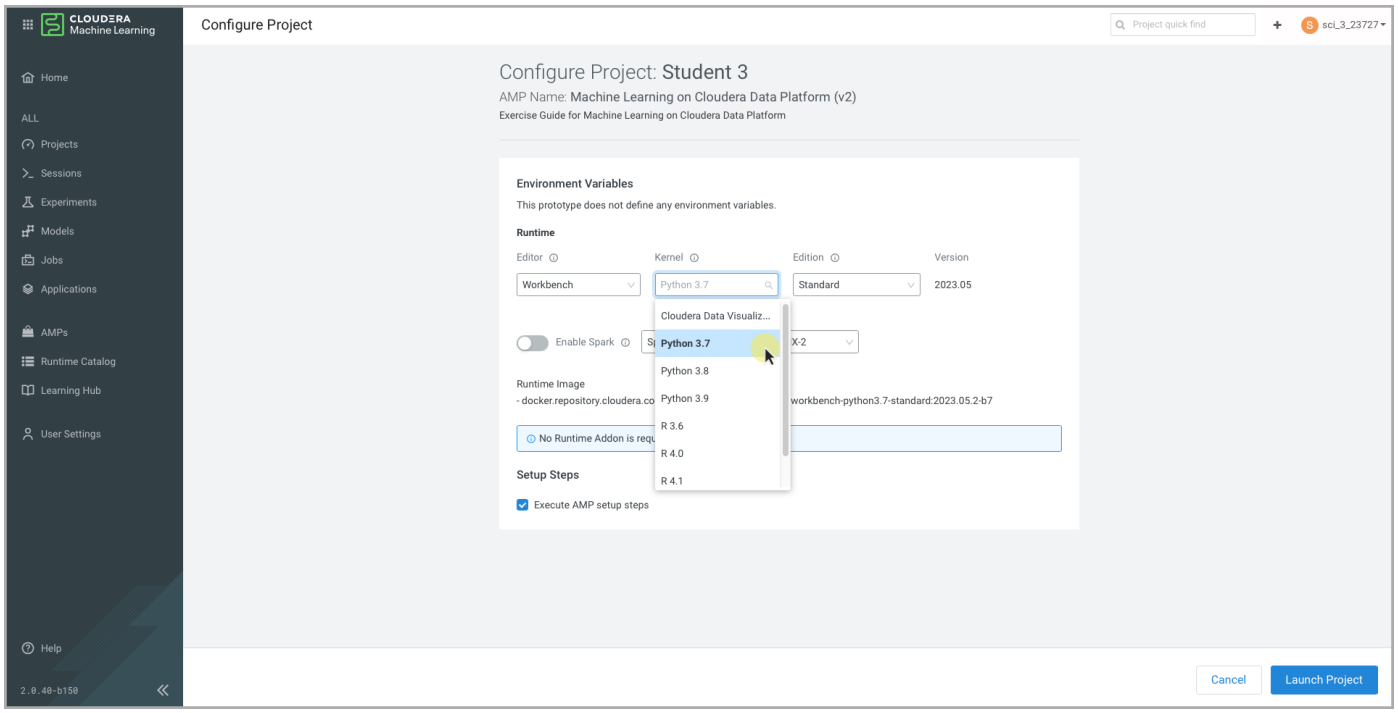
5. Enter `https://github.com/bshimel-cloudera/edu-cml-on-cdp` into the **Provide Git URL of your AMPs** field.



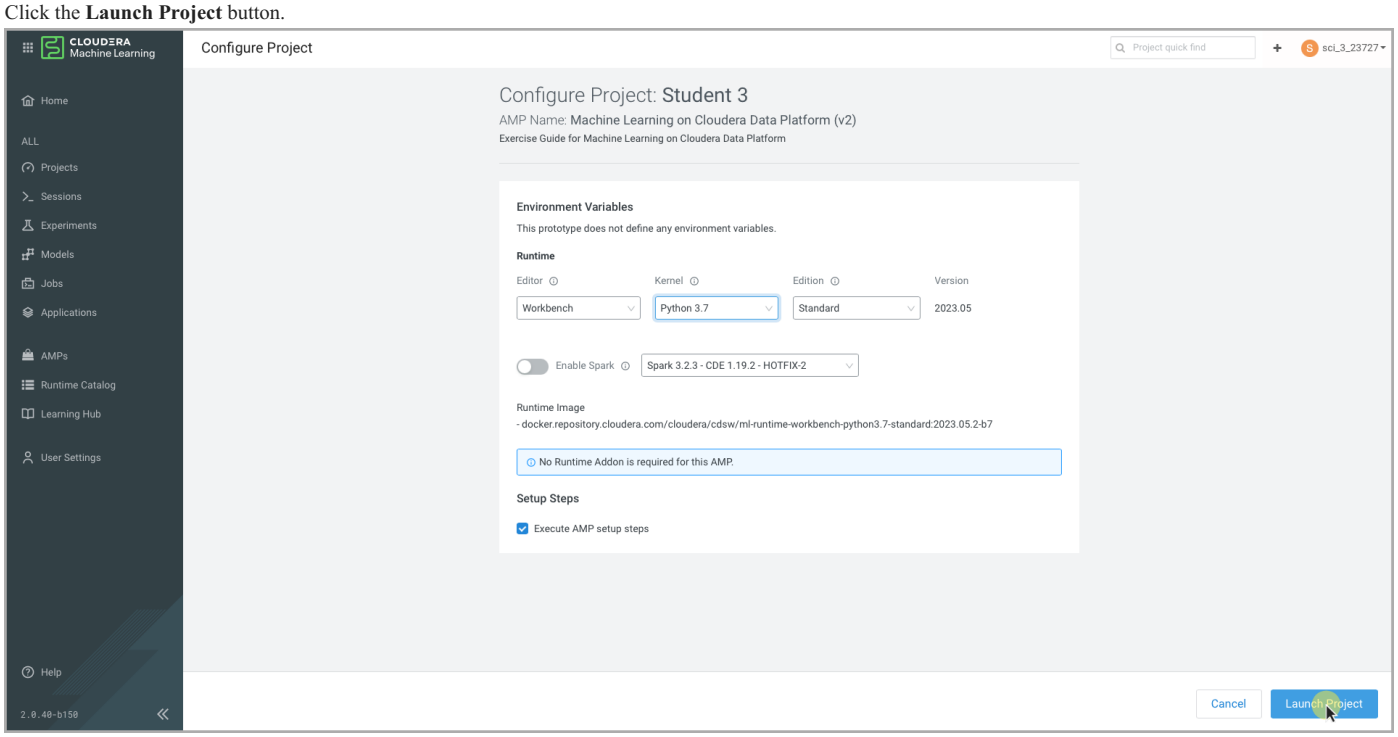
6. Click the **Create Project** button.



7. Select `Python 3.7` as the **Kernel**, if it is not already selected.

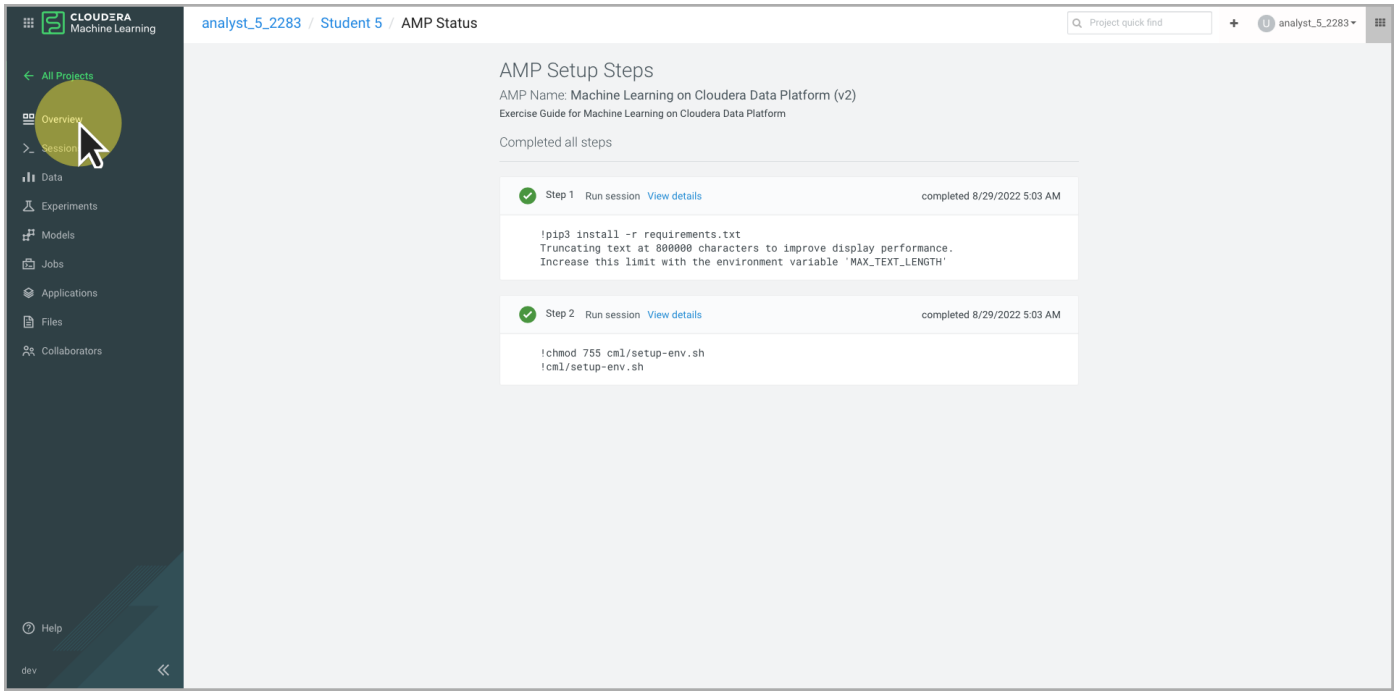


8. Click the **Launch Project** button.

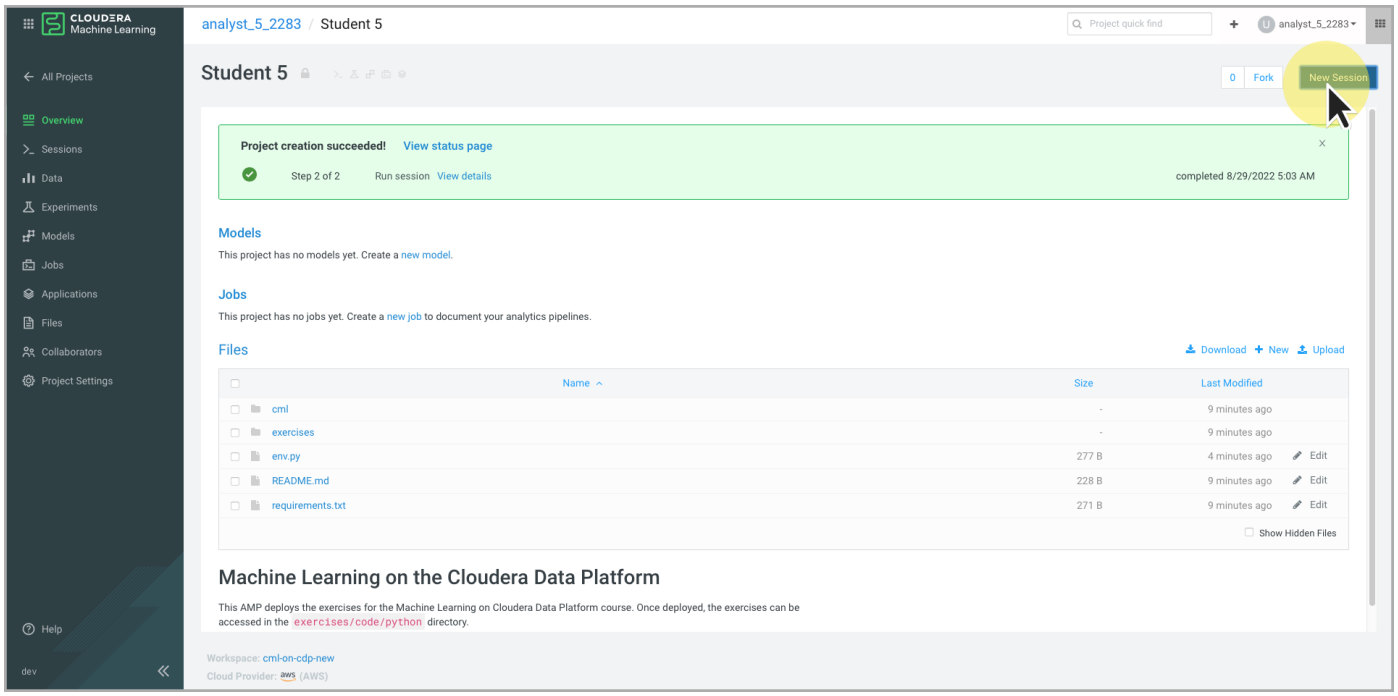


9. Wait for AMP Setup to Complete. The AMP is installing the Python modules and exercise files. The process should take about 5 minutes to complete.

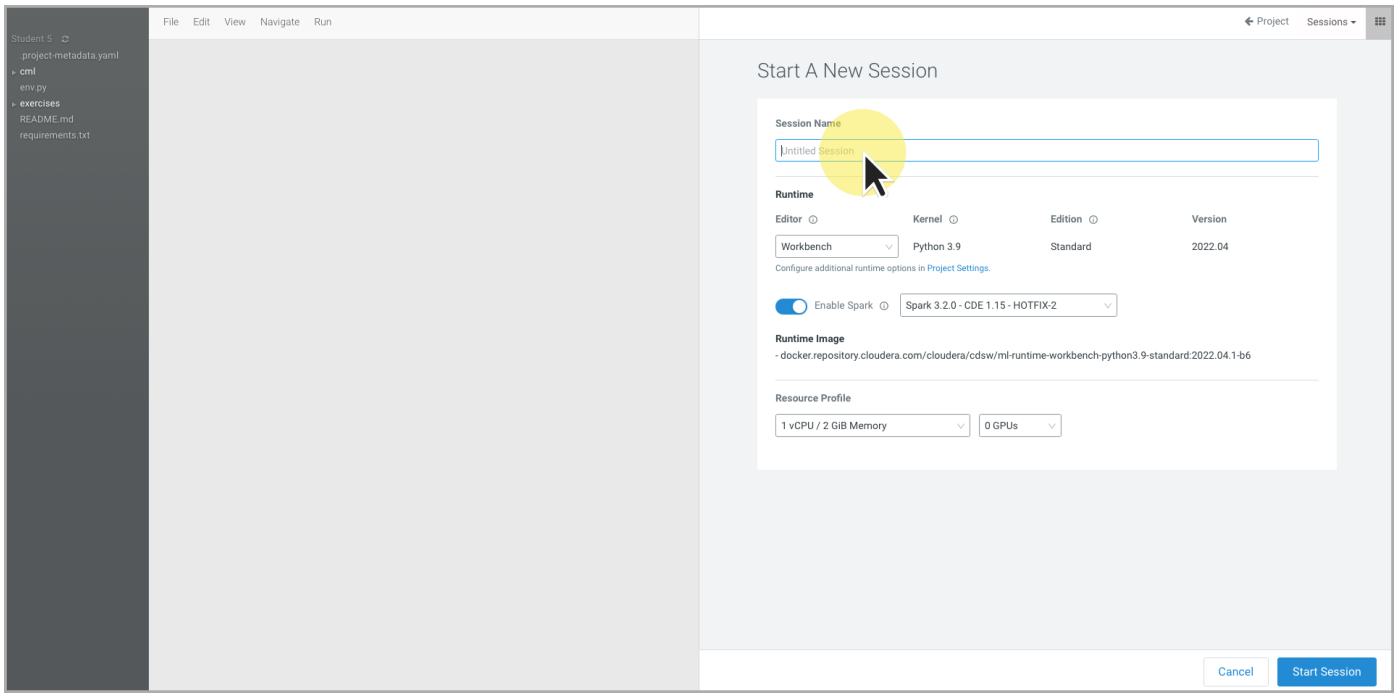
10. Select **Overview** from the project menu on the left.



11. Click the **New Session** button.



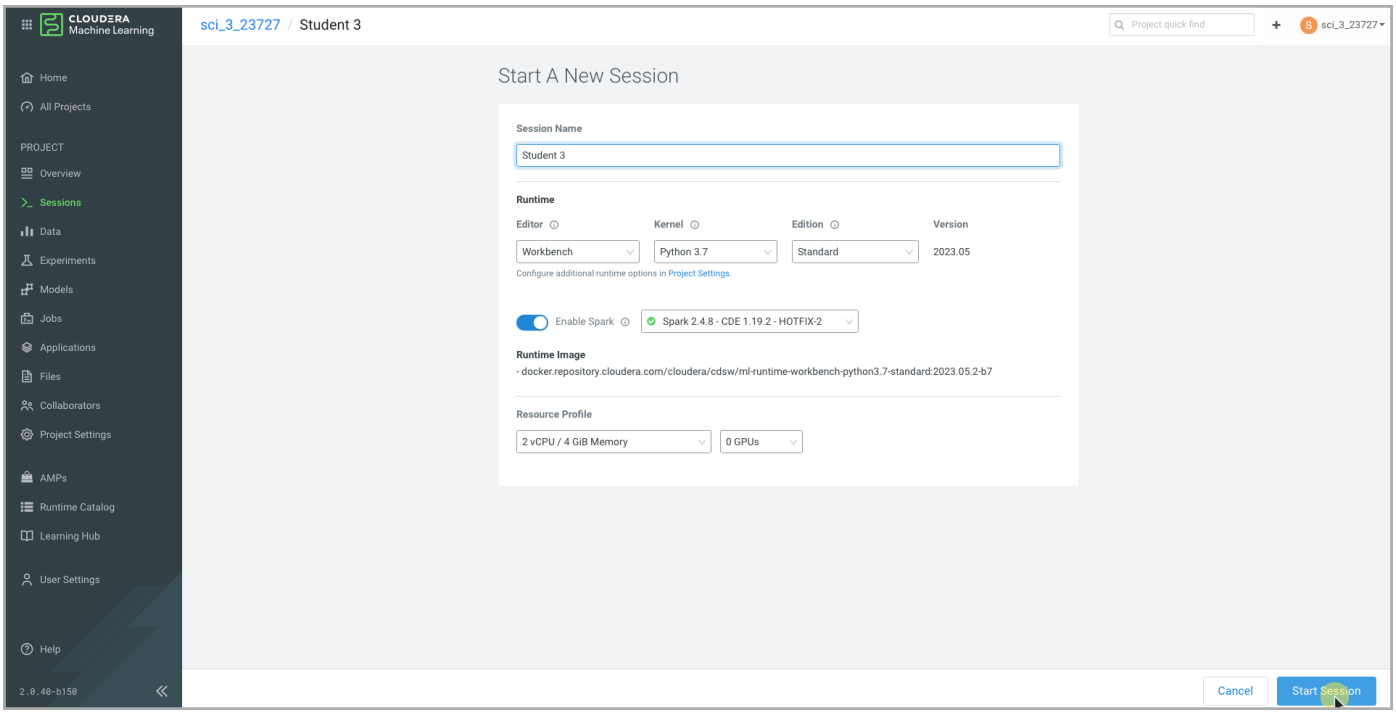
12. Enter `Student #` for **Session Name**.



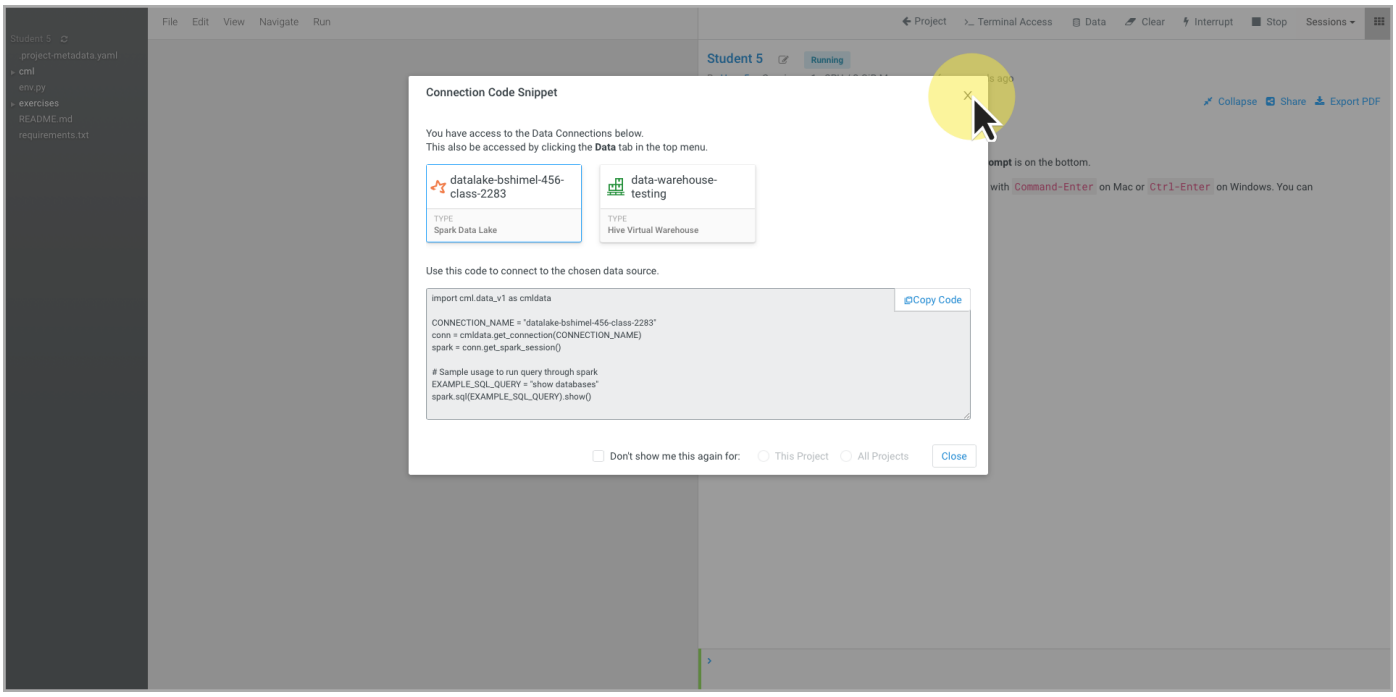
Info

The workbench exercises require Python 3.7 and Spark 2.

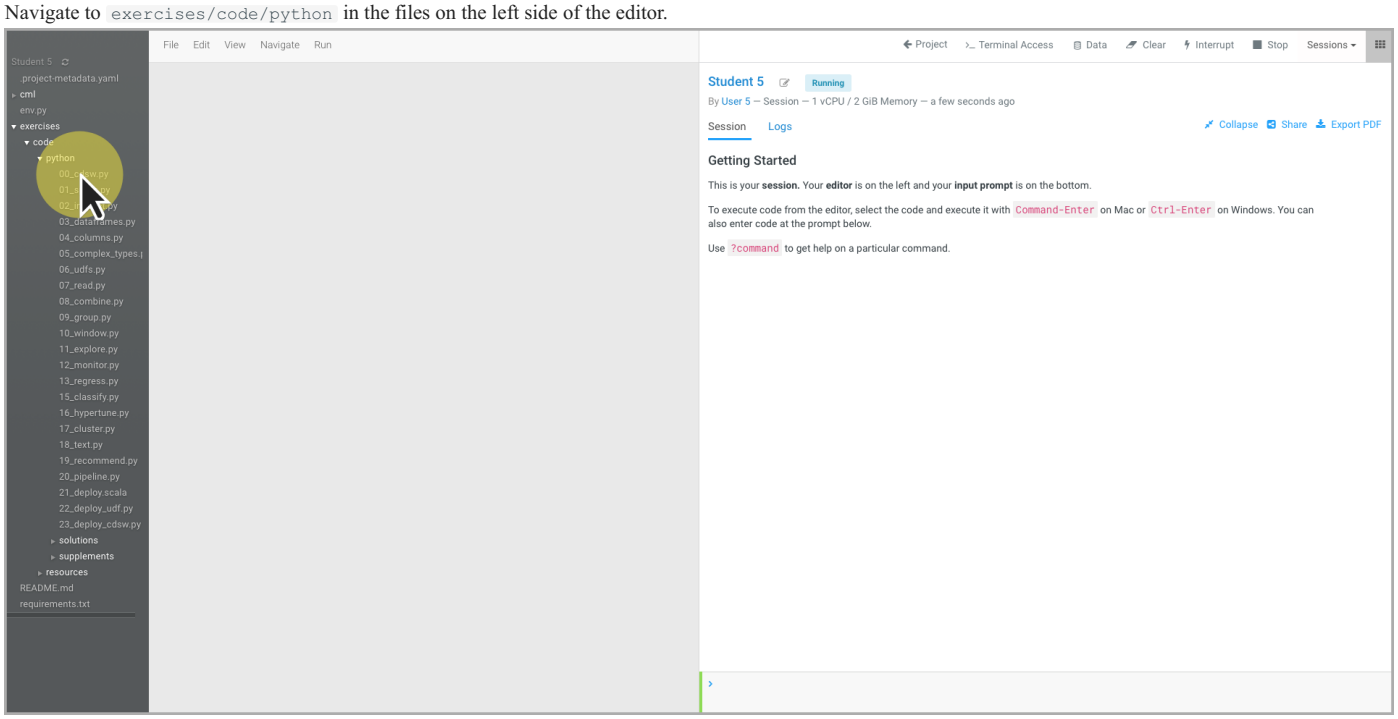
1. Click the **Start Session** button.



2. Close the **Connection Code Snippet** dialog.

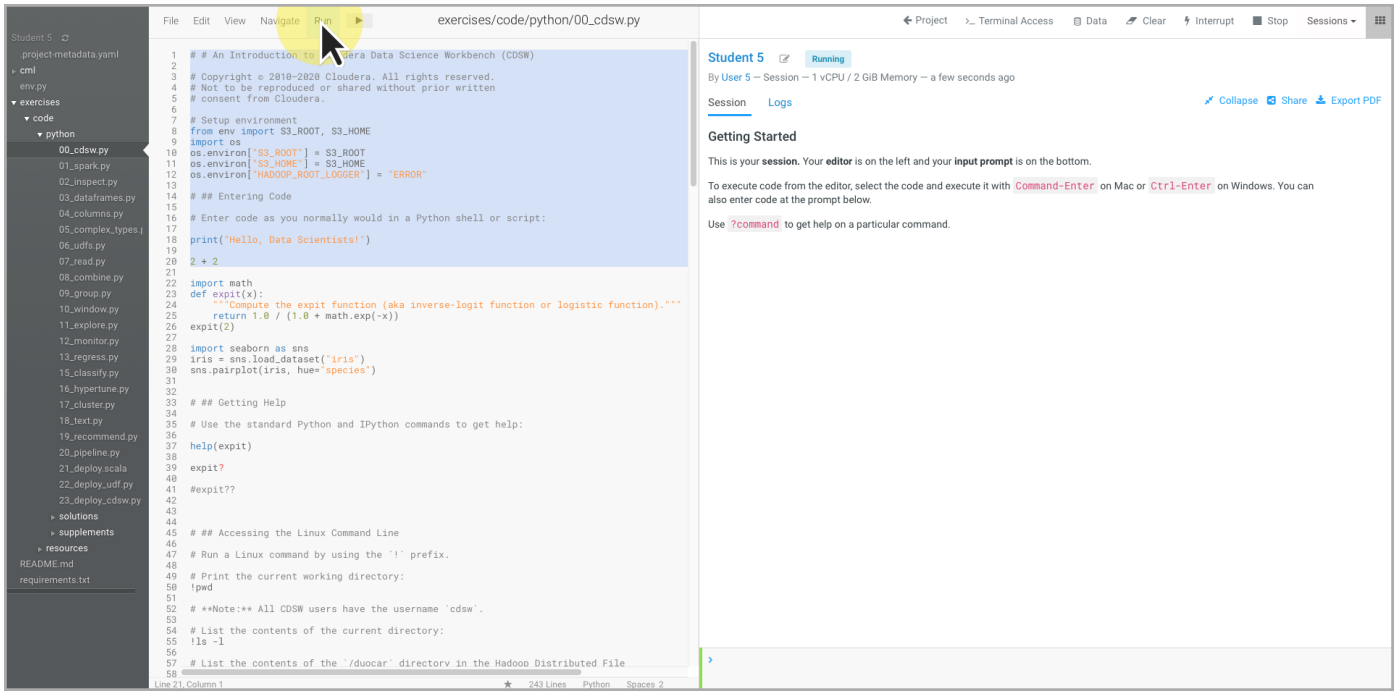


3. Navigate to `exercises/code/python` in the files on the left side of the editor.

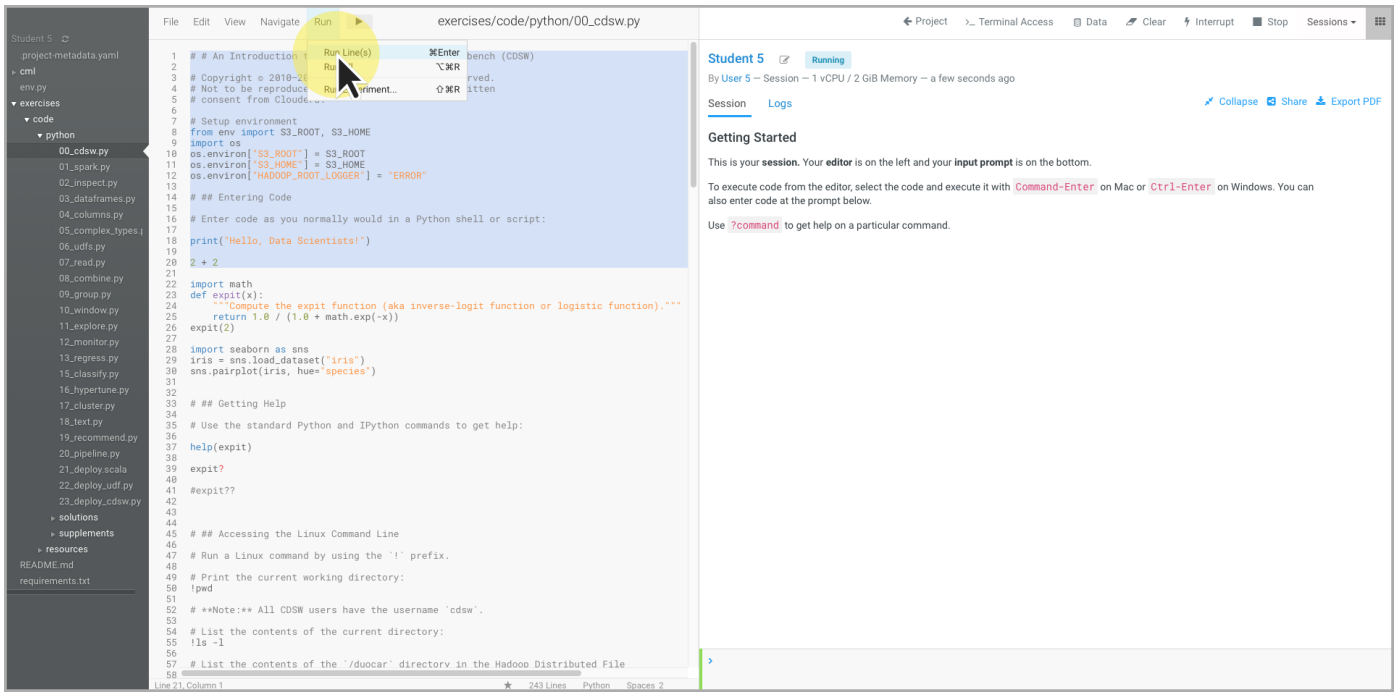


4. Click on `cml.py` to open the file in the editor.

5. Highlight rows 1 - 20.



6. Select **Run / Run Lines** from them menu.



The output from the first twenty lines should be displayed on the right.

End of Exercise

Autoscaling, Performance, and GPU Settings

The time-saving potential of using GPUs for complex and large tasks is massive, setting up these environments and tasks such as wrangling NVIDIA drivers, managing CUDA versions and deploying custom engines for your specific project needs can be time consuming and challenging. To make these processes simpler — and to get data scientists working on ML use cases faster — CML made it simple to configure and leverage NVIDIA GPUs natively.

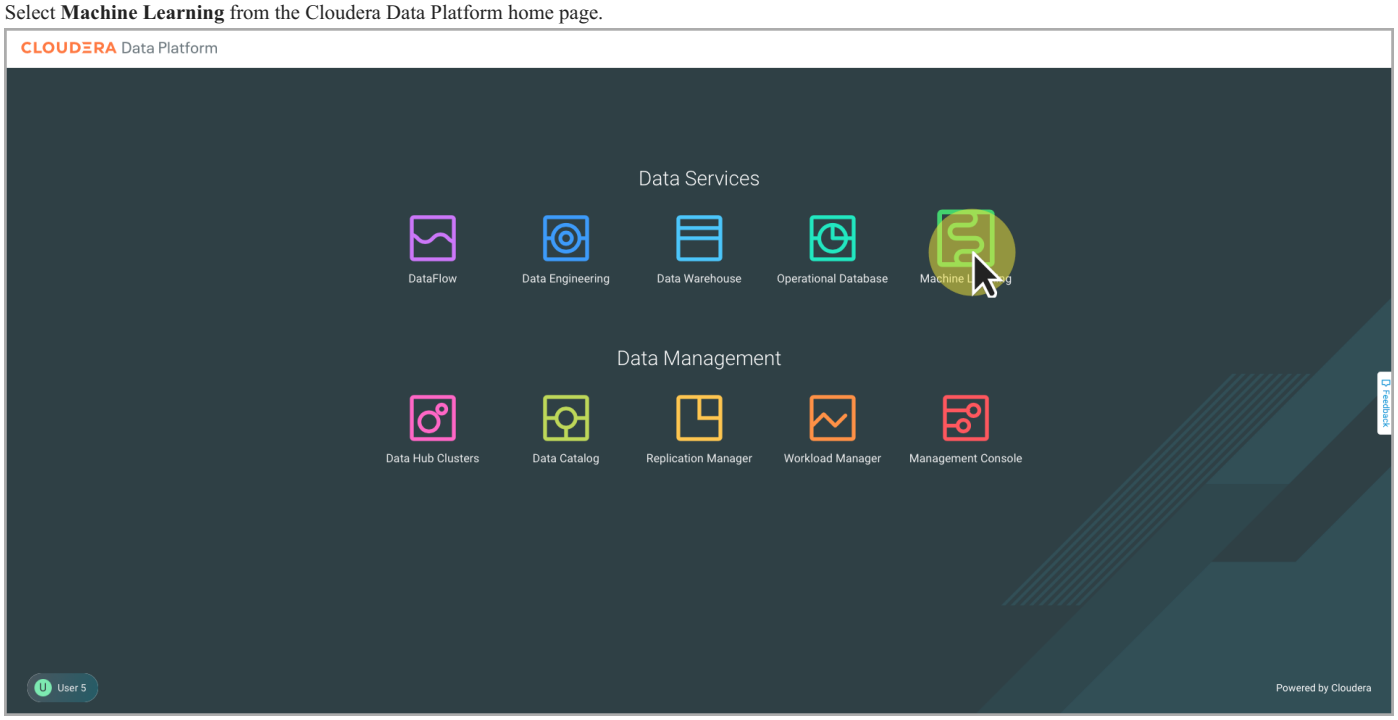
In this exercise, you will use a Computer Vision Image Classification example and train a deep learning model to classify fashion items leveraging the Fashion MNIST Dataset. The main focus of the exercise is to demonstrate how to use the GPU and auto scaling features of CML. If you would like to know more about the Computer Vision Image Classification example, you can view the project on Github.

In this exercise, you will:

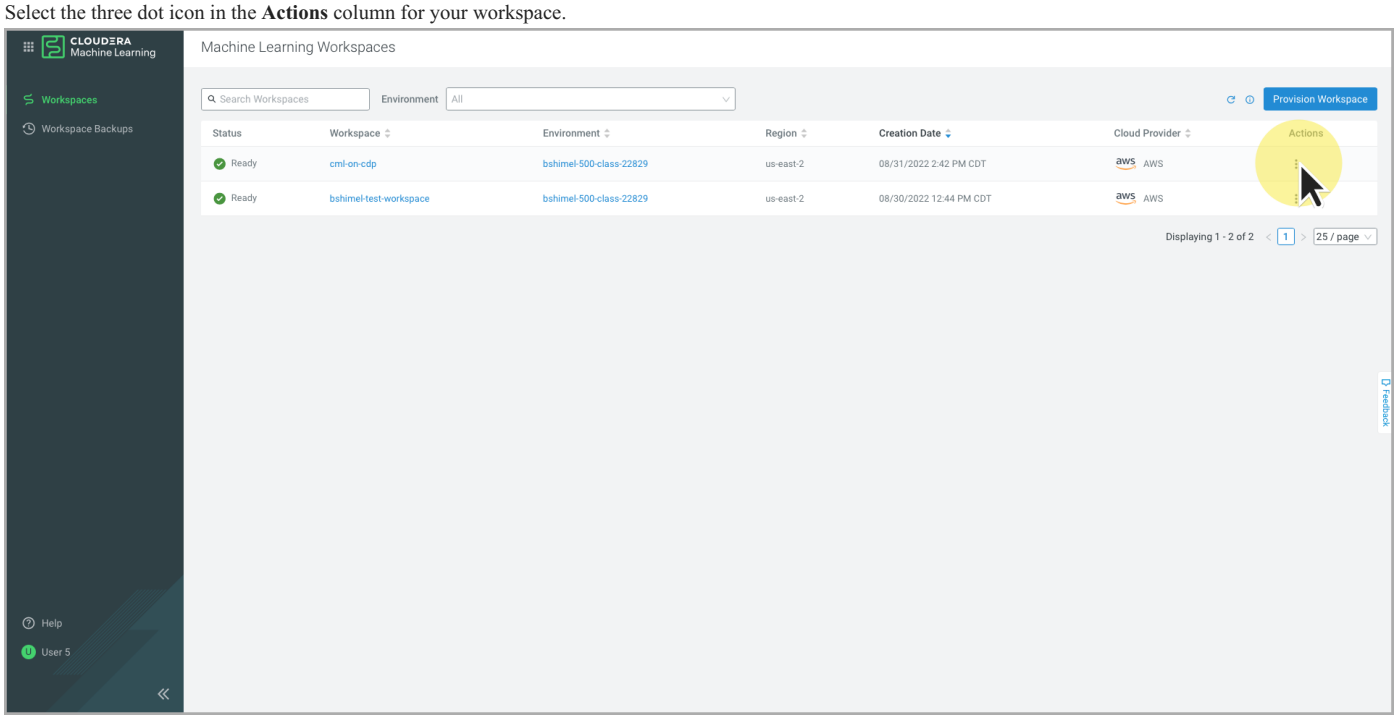
- Run an image classification without a GPU,
- Run and image classification example with a GPU, and
- Examine the difference in performance.

View Workspace Details and Allocated GPUs

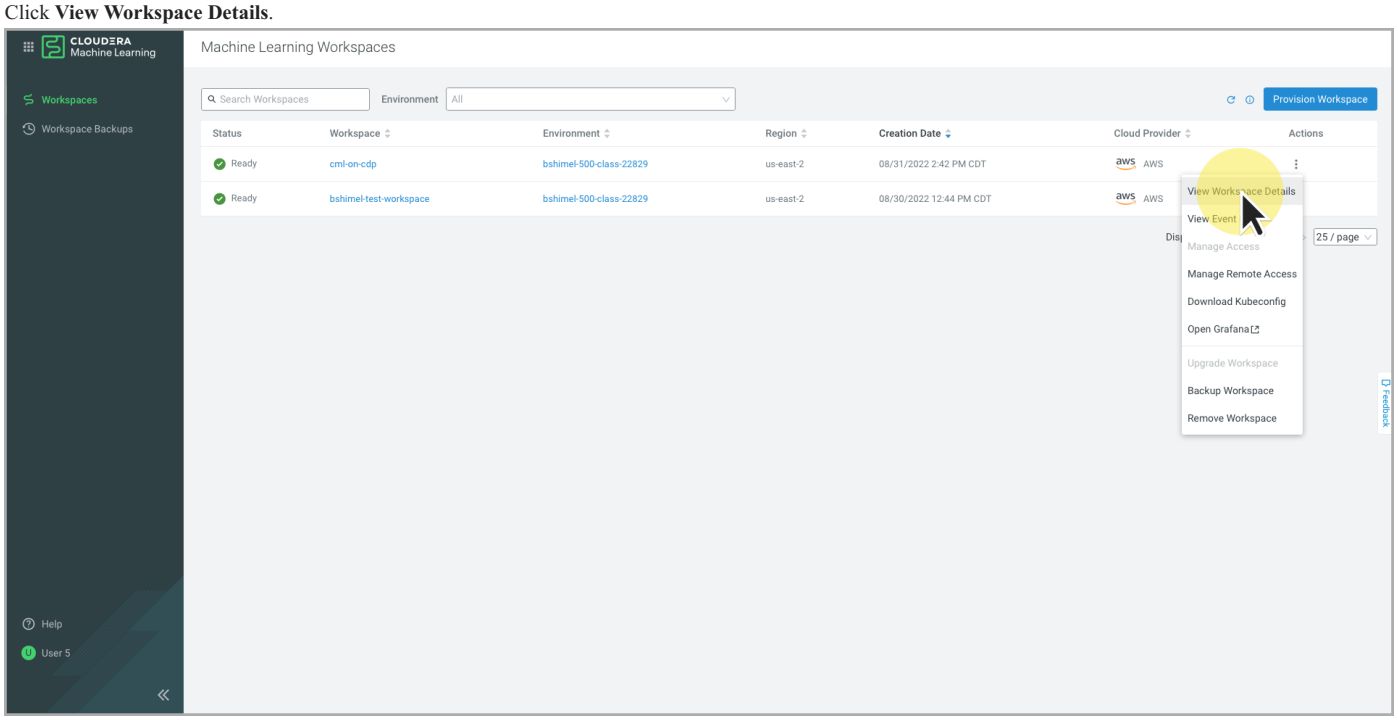
1. Select **Machine Learning** from the Cloudera Data Platform home page.



2. Select the three dot icon in the **Actions** column for your workspace.

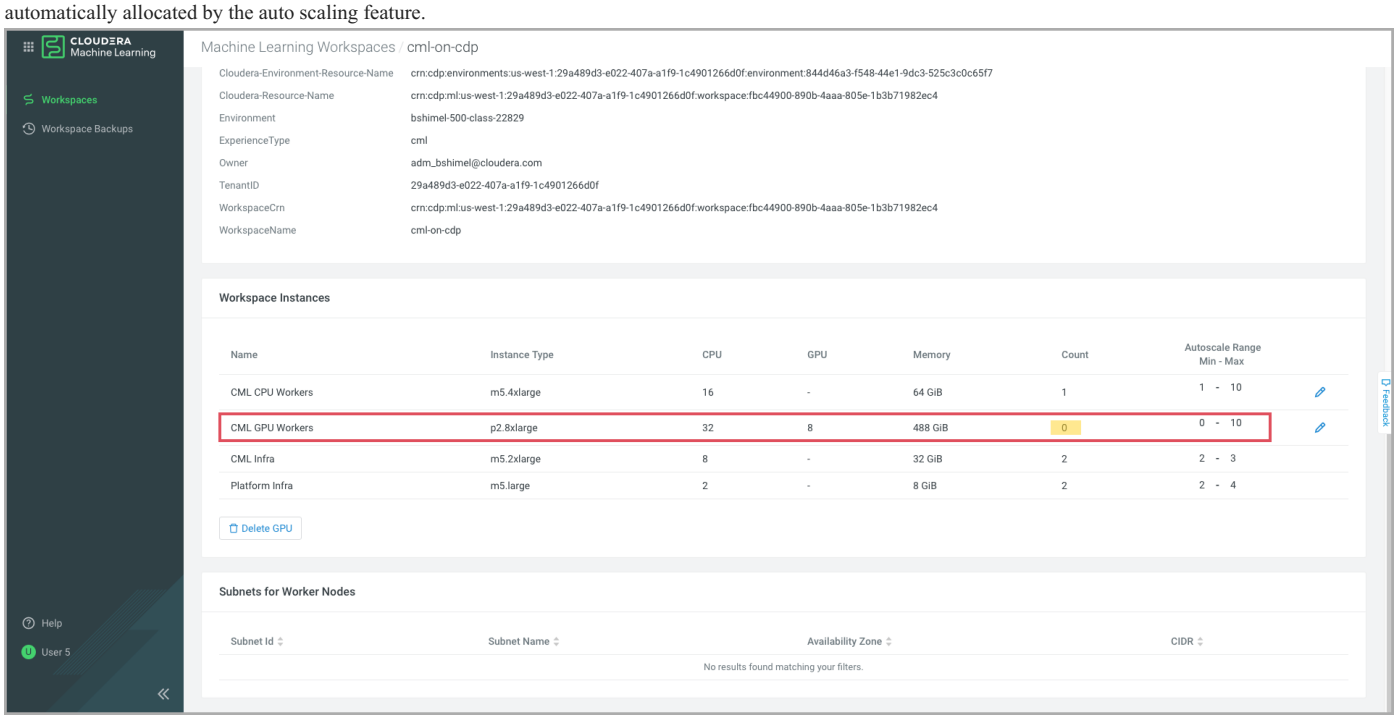


3. Click **View Workspace Details**.



4. Scroll down to view **Workspace Instances**. In the example below, there are no **CML GPU Workers** as indicated by the zero in the Count program. The **Autoscale Range** indicates that there could be zero to ten instances. Your workspace may show different values. For example, if someone else already requested a session with a GPU, you may have a one or greater value in the **Count** column. In the example below, the **Instance Type** for the CML GPU Worker is an AWS p2.8xlarge. This is a fairly beefy instance with 8 GPU cores. Therefore, if one instance is running, it can support up to eight sessions with one GPU before another instance will be

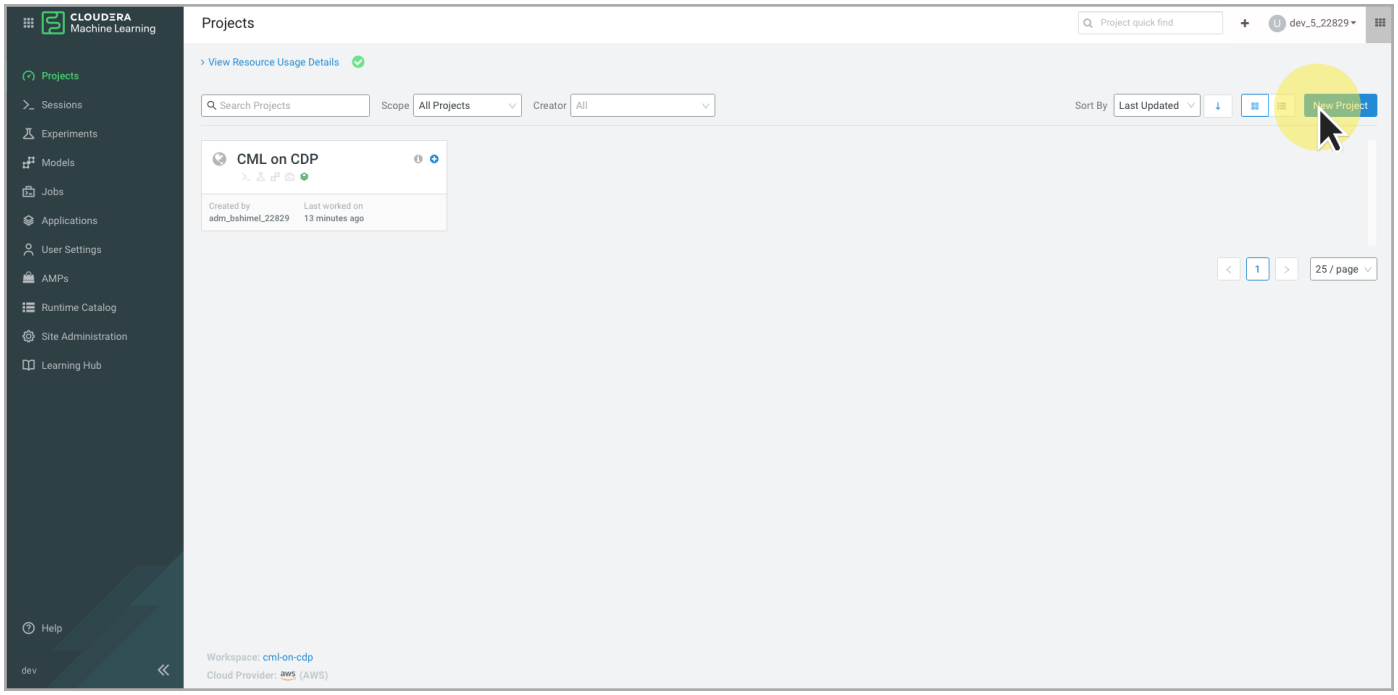
automatically allocated by the auto scaling feature.



5. Leave the Workspace Details tab open in your browser.

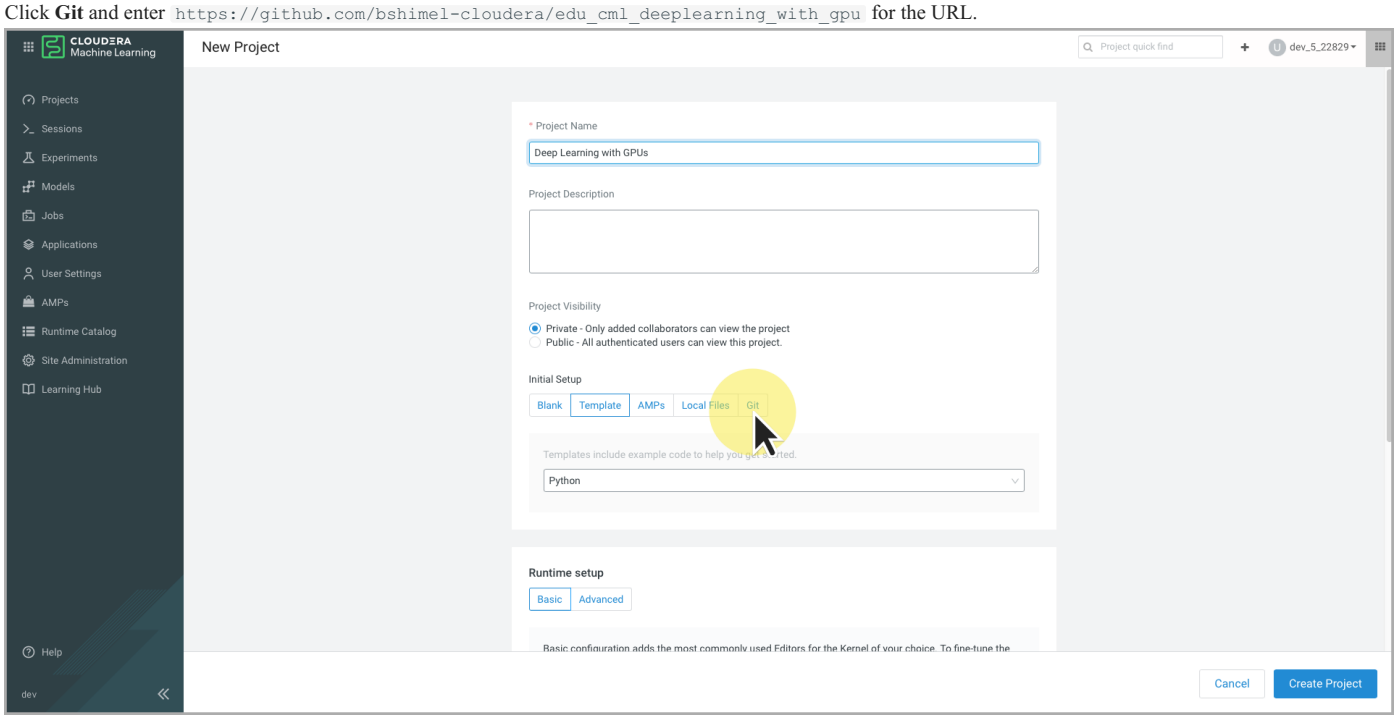
Create a New Project

1. Open a new browser tab and navigate to your CML workspace.
2. Click the **New Project** button.



3. Enter `Deep Learning with GPUs` for the Name.

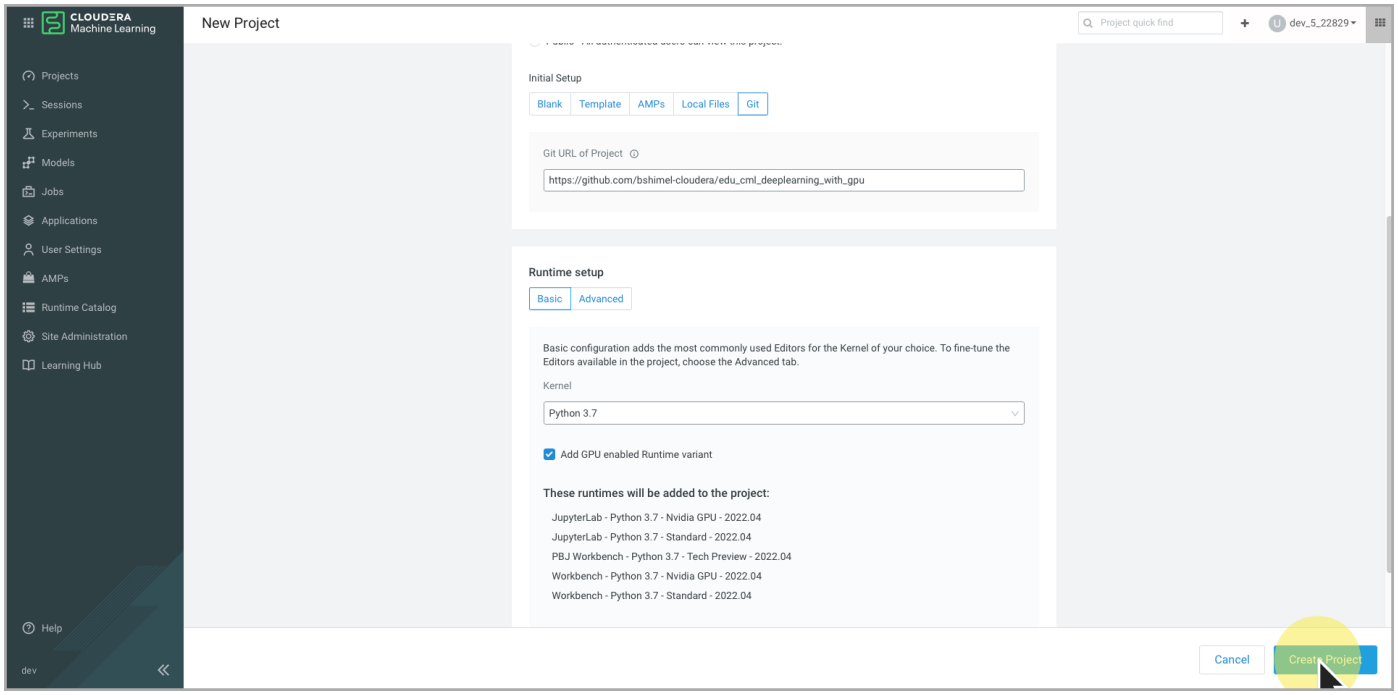
4. Click **Git** and enter `https://github.com/bshimel-cloudera/edu_cml_deeplearning_with_gpu` for the URL.



5. Select **Python 3.7** as the **Kernel**.

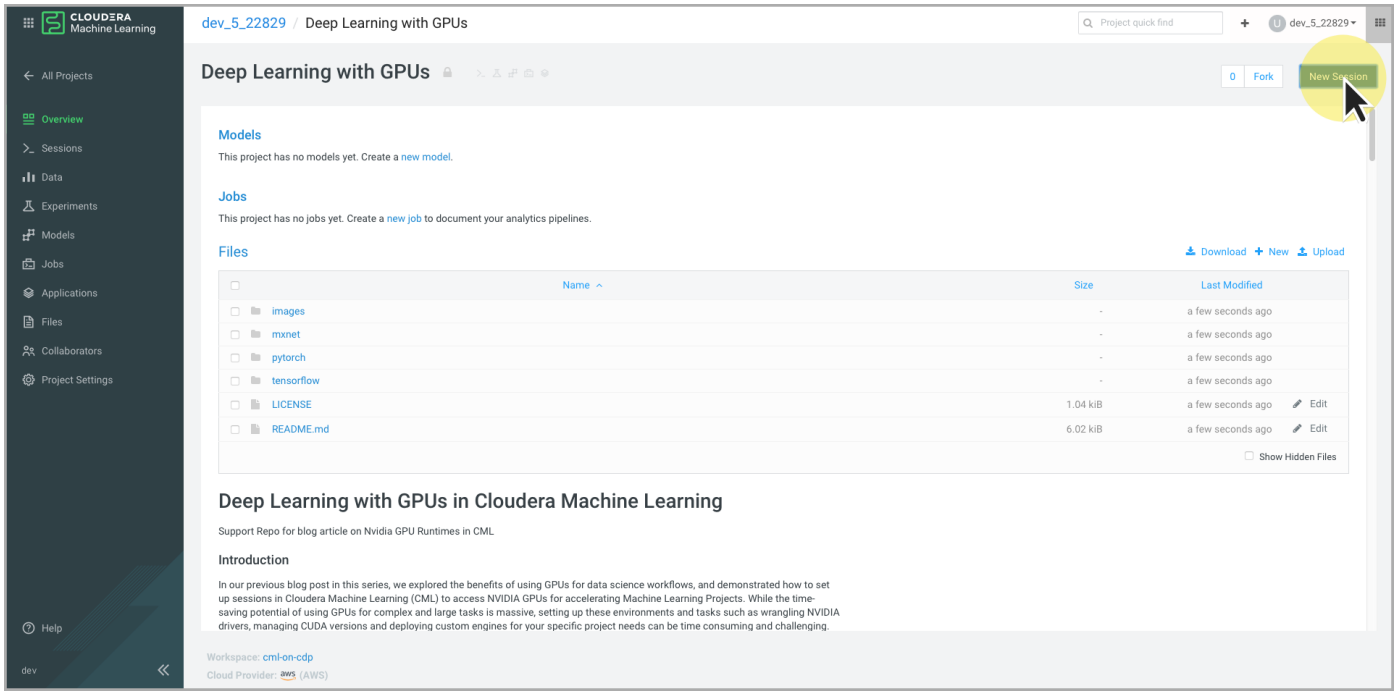
6. Check **Add GPU enabled Runtime variant**. This is extremely important. It tells CML to use the runtime with the GPU drivers.

7. Click **Create Project**

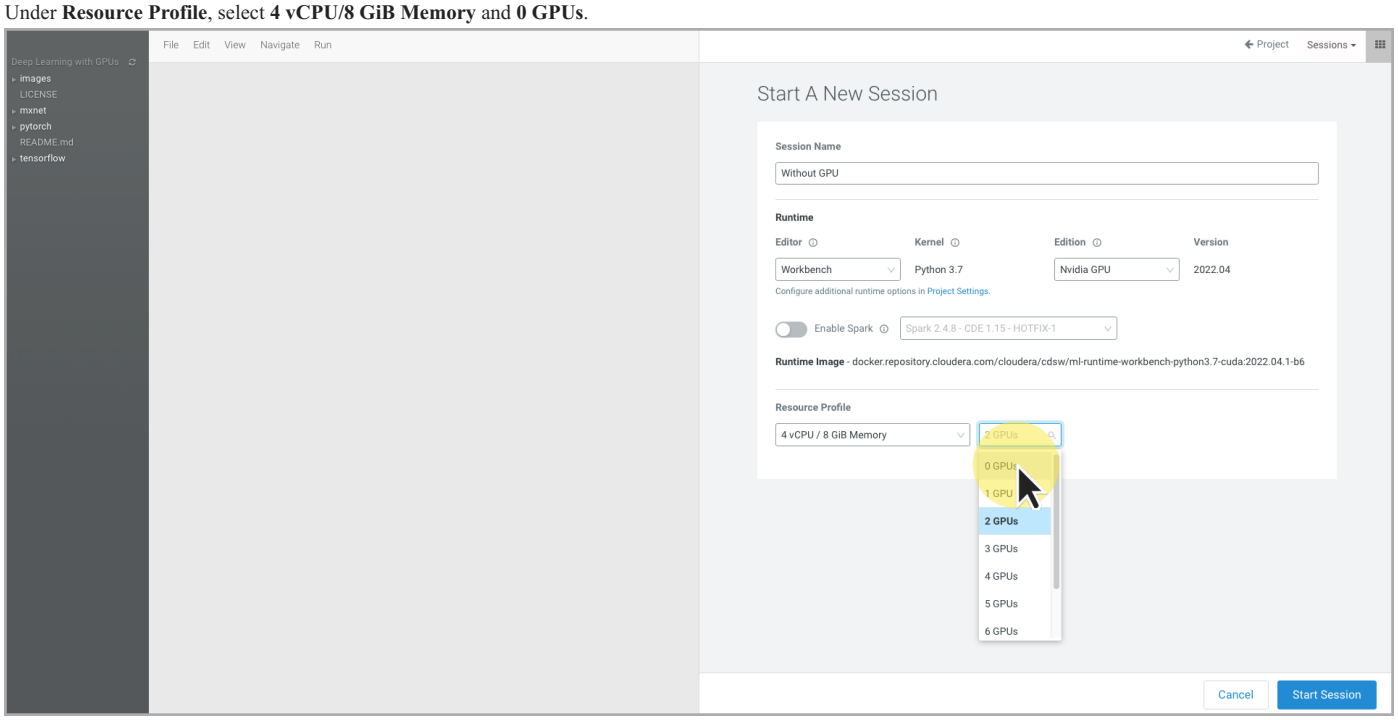


Create a Session without a GPU

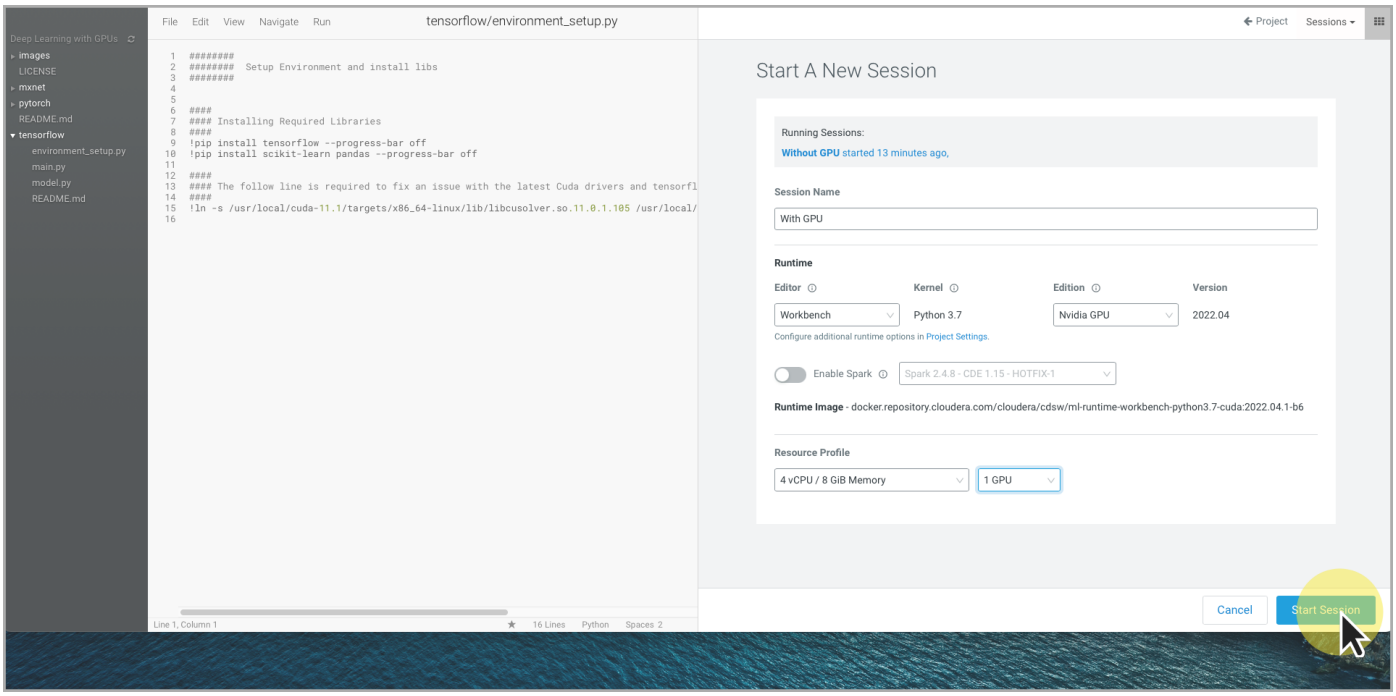
1. Click the **New Session** button.



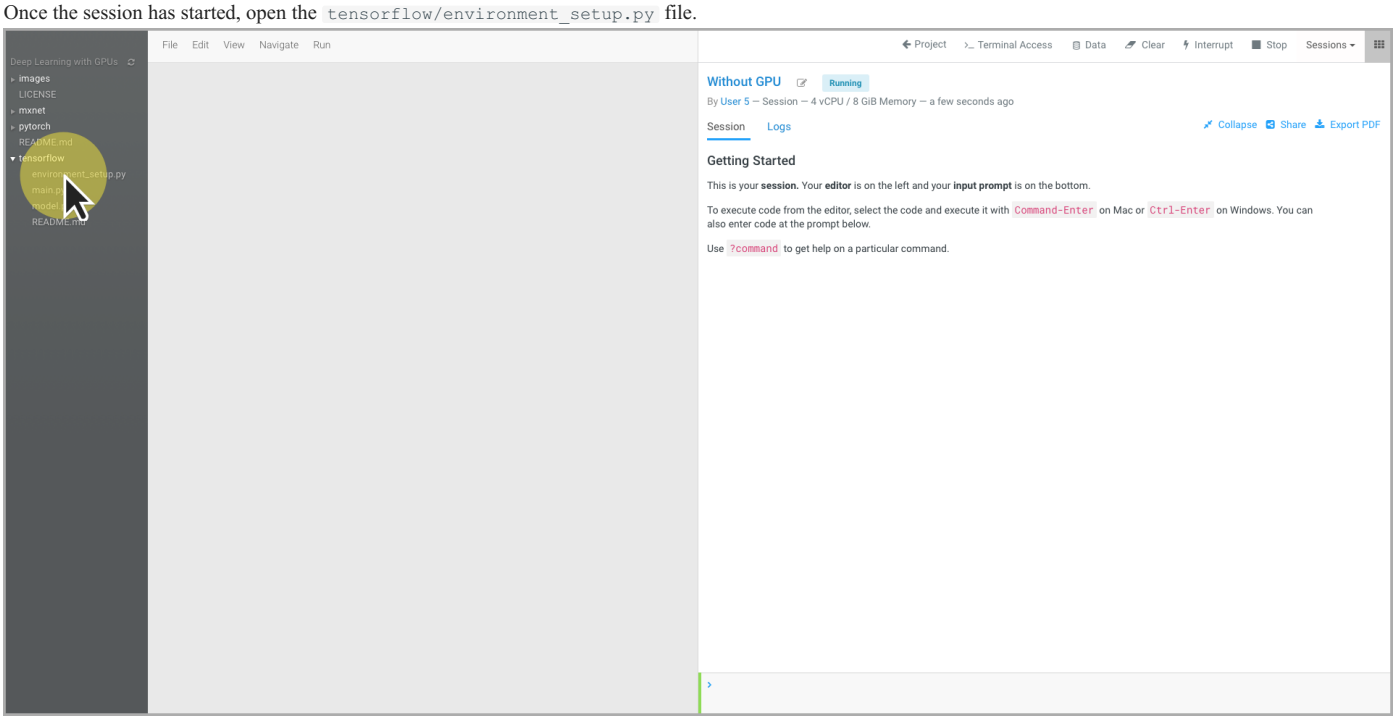
2. Enter **Without GPU** for the **Session Name**.
3. Under **Runtime**, select **Workbench**, **Nvidia GPU** and verify **Version** is **2023.05**. Even though you will not be using a GPU in this example, you want the runtime with the Nvidia libraries for TensorFlow to work properly.
4. Under **Resource Profile**, select **4 vCPU/8 GiB Memory** and **0 GPUs**.



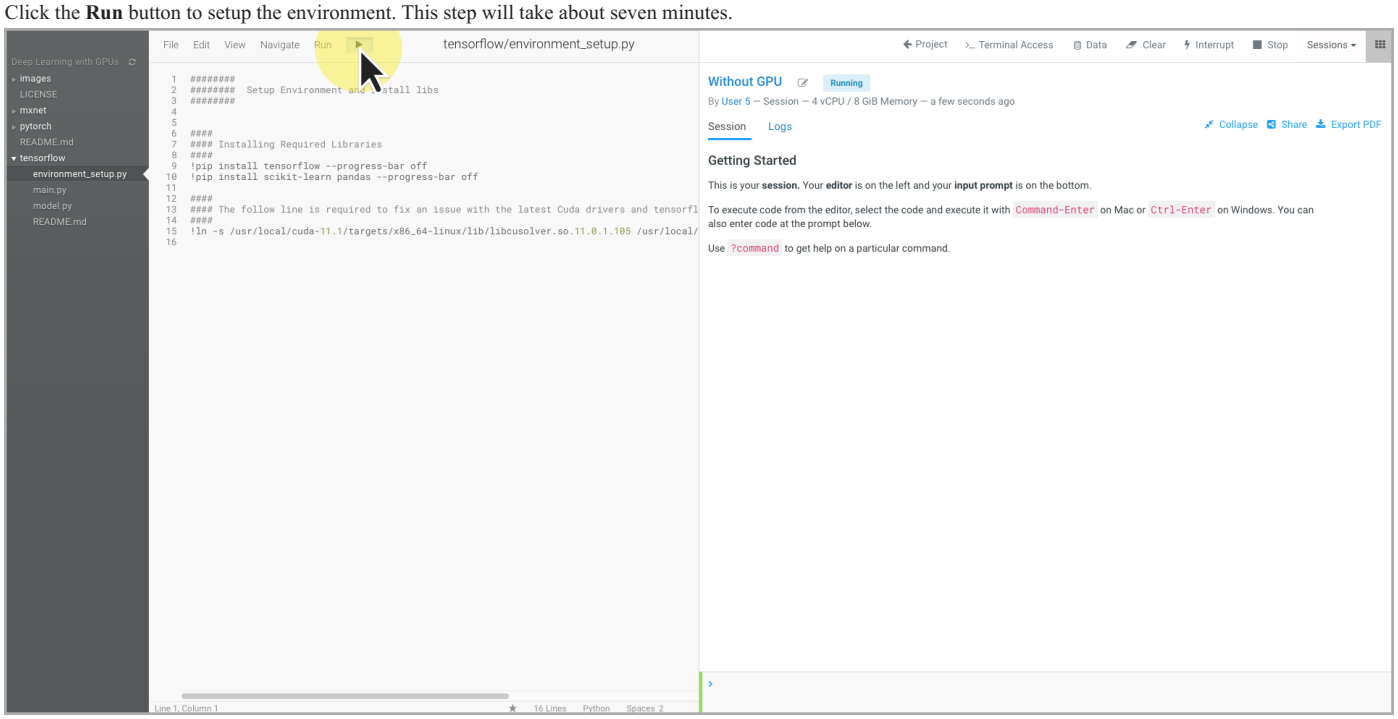
5. Click the **Start Session** button.



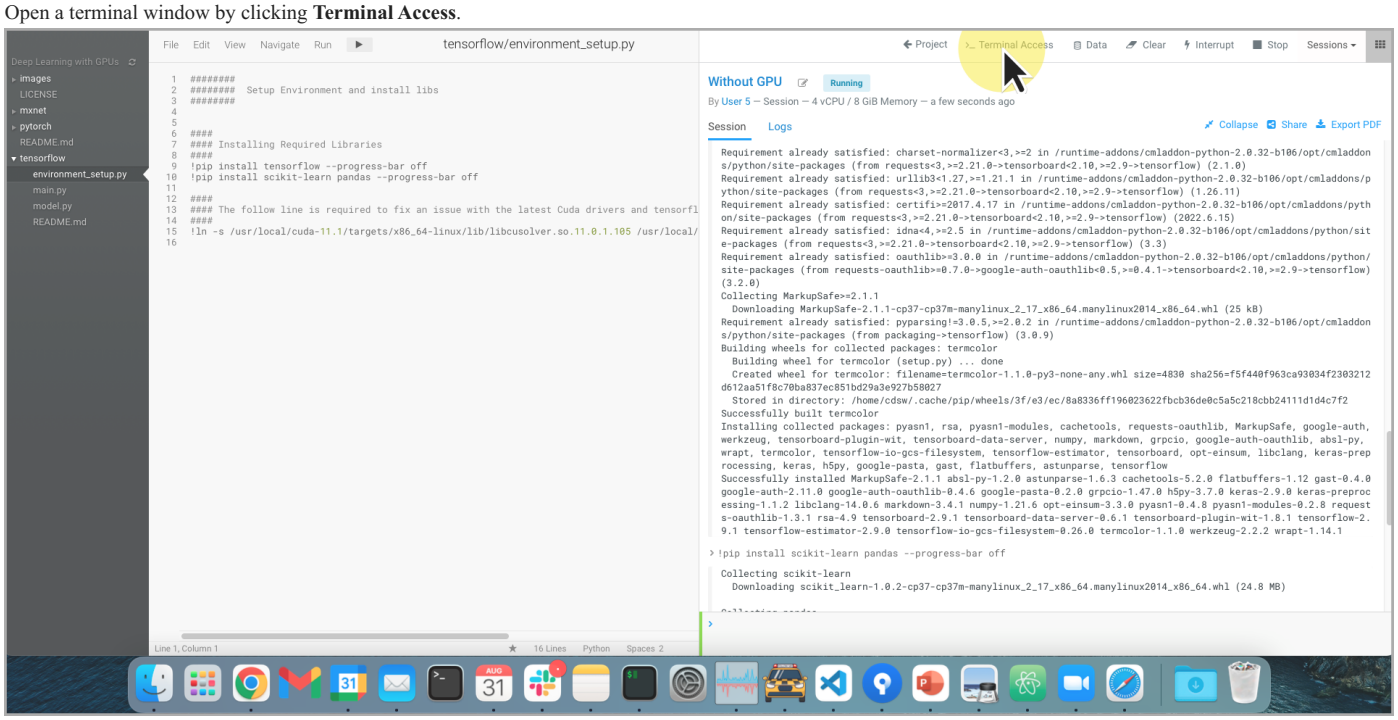
6. Once the session has started, open the `tensorflow/environment_setup.py` file.



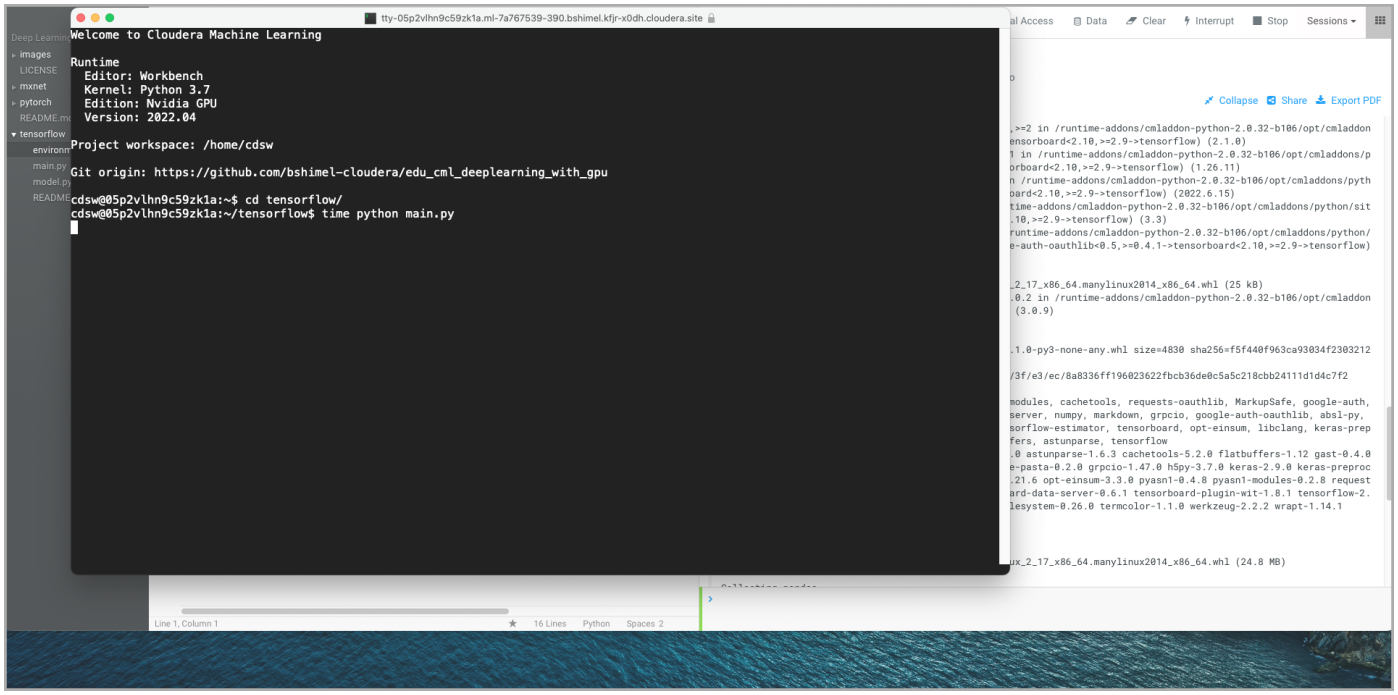
7. Click the **Run** button to setup the environment. This step will take about seven minutes.



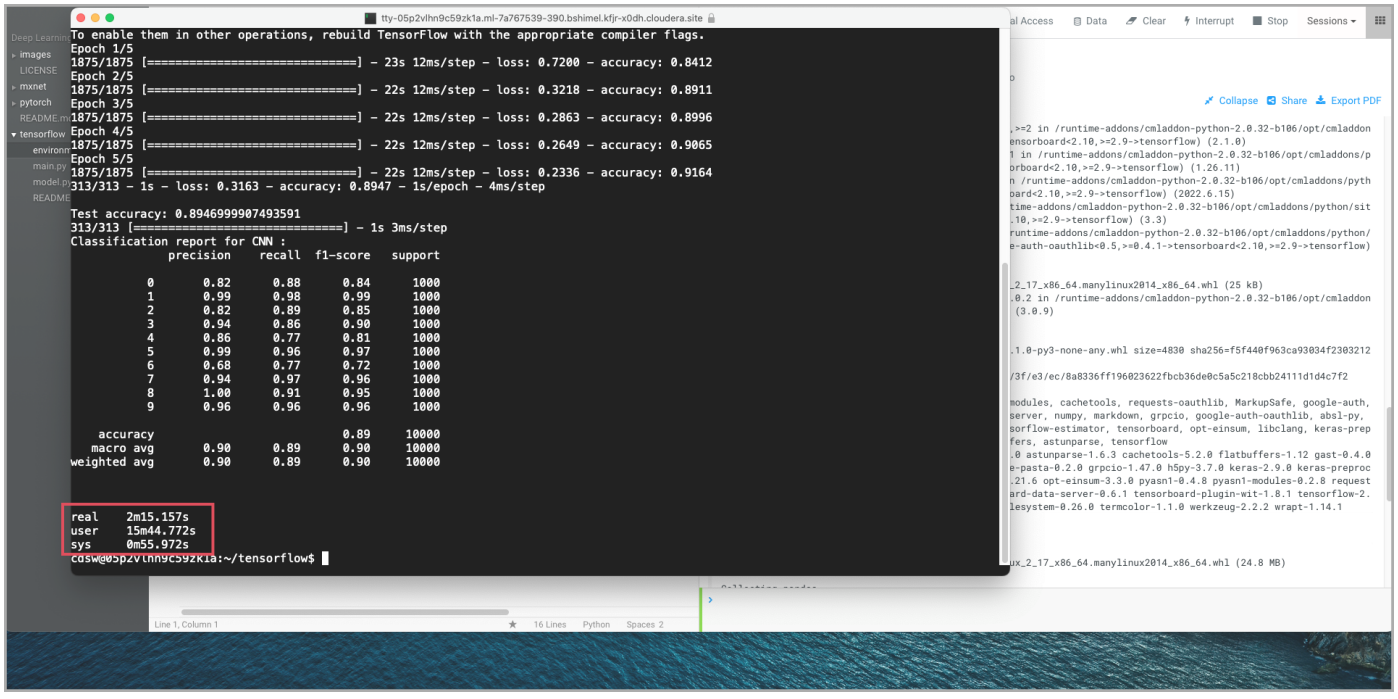
8. Open a terminal window by clicking **Terminal Access**.



9. Enter `cd tensorflow` to change the tensorflow directory. Then, enter `time python main.py`. This will run the image classification program and record the time to run the program.



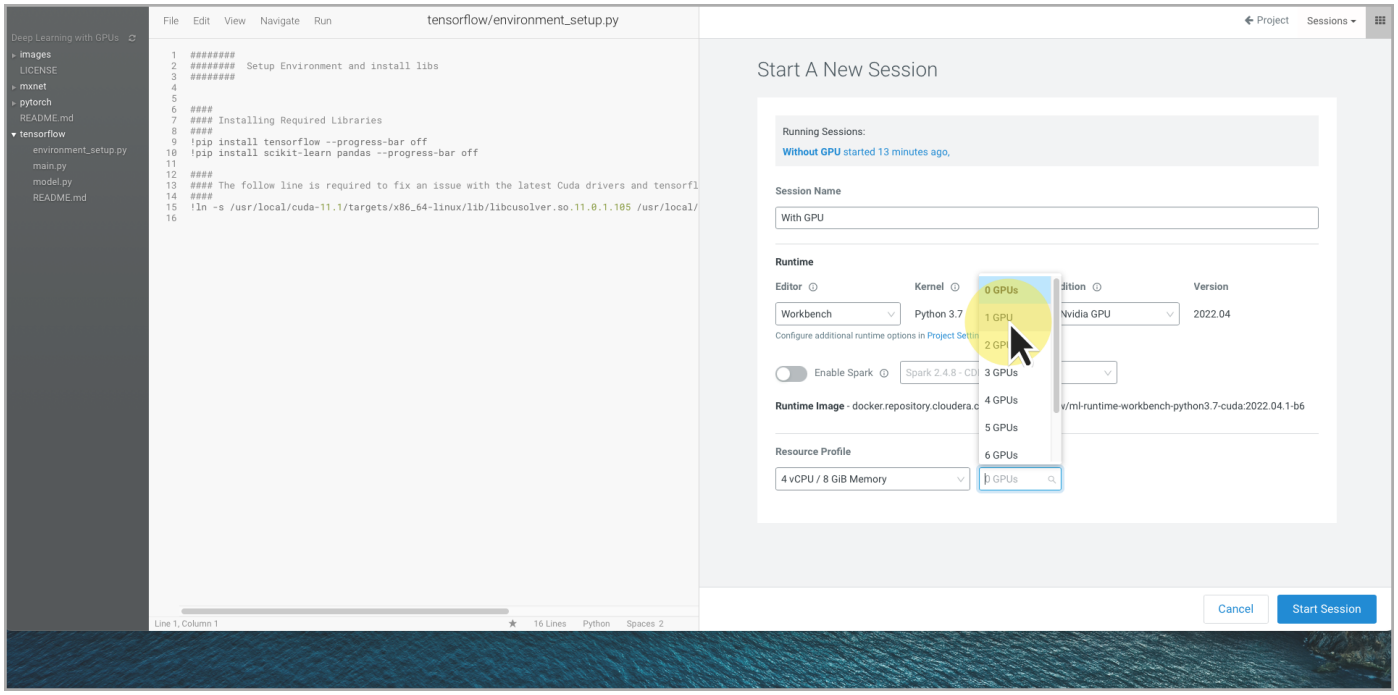
10. When the program finishes, note the real, user, and sys time required to run the program.



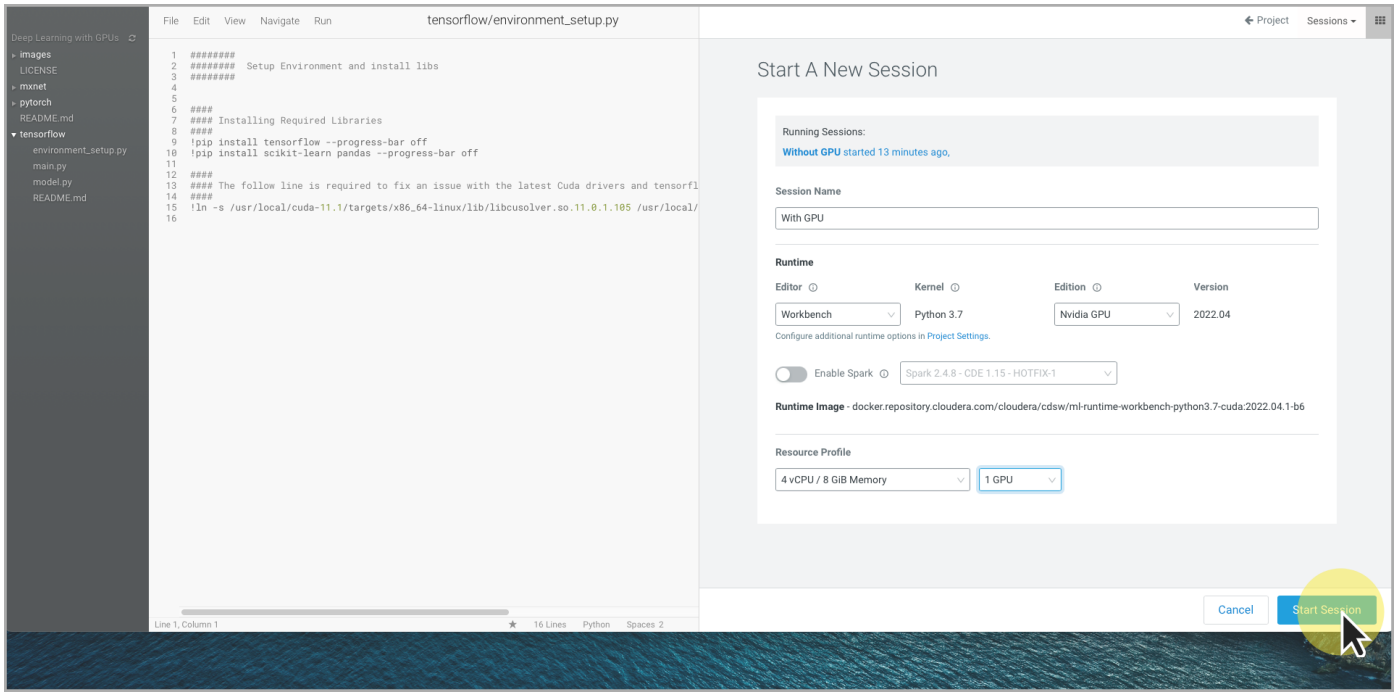
Create a Session with a GPU

1. Close the terminal and go back to the project.
2. Create a new session titled `With GPU`.

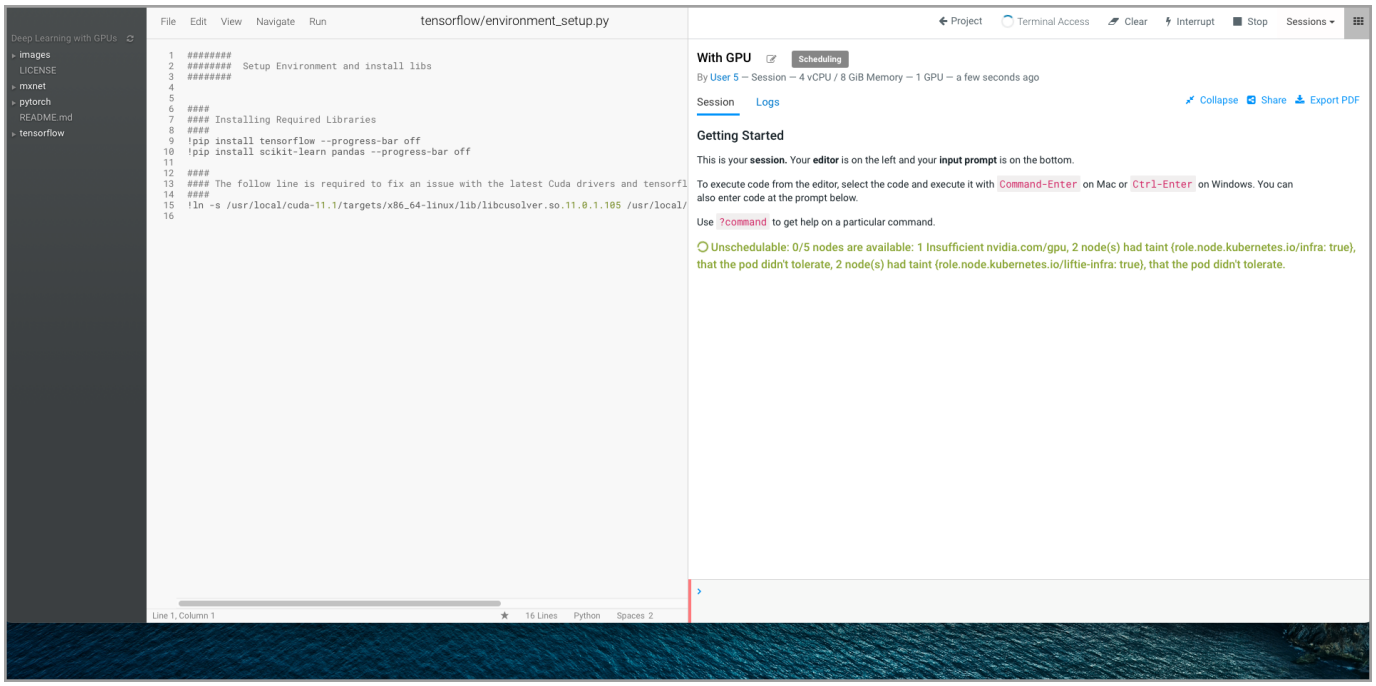
3. Select **4 vCPU/8 GiB Memory** and **1 GPU**. It might be tempting to select more than one GPU, but do not. The example is not configured to take advantage of more than one GPU.



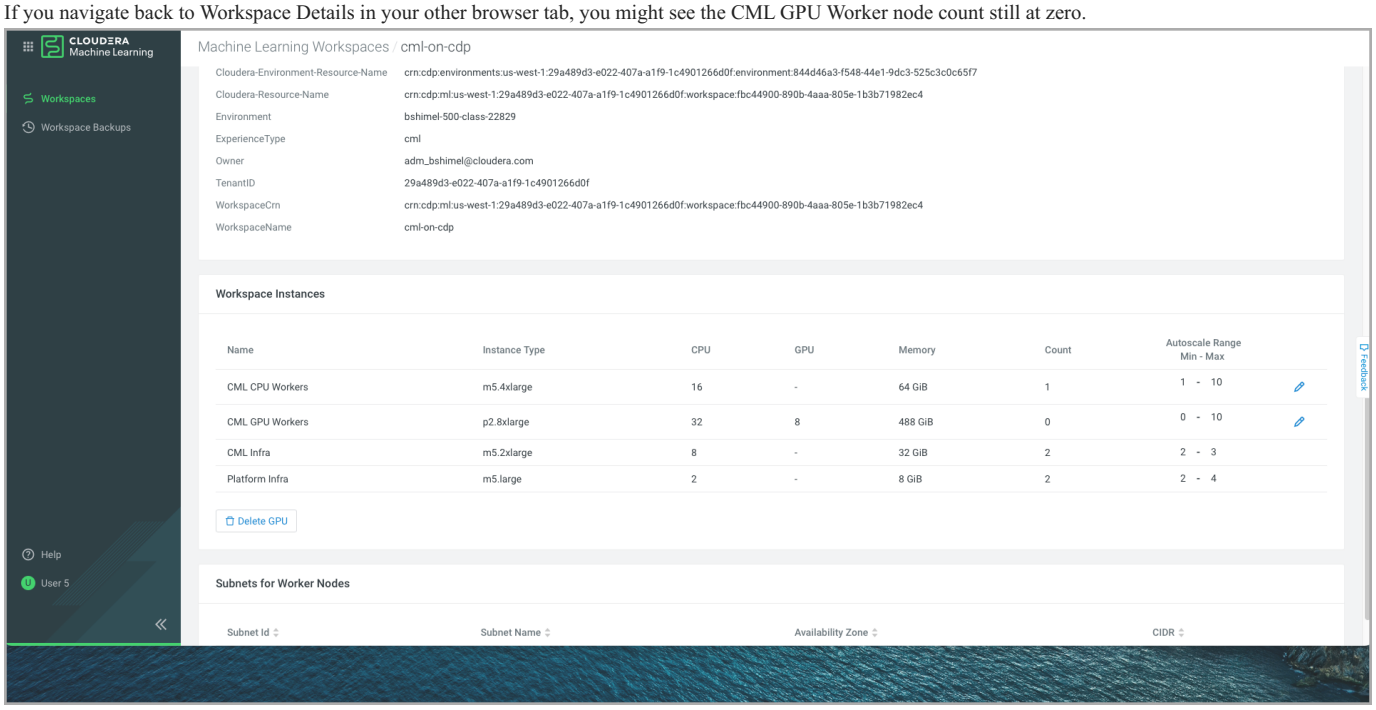
4. Click the **Start Session** button.



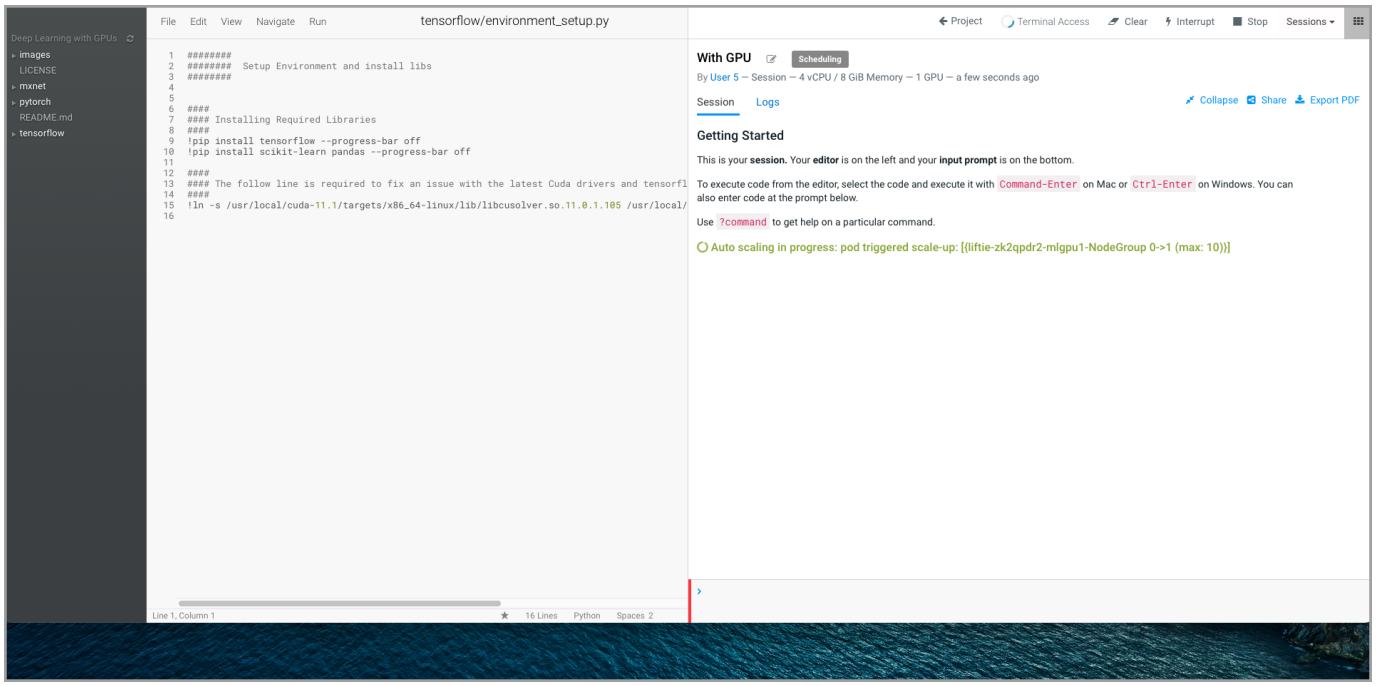
It is hard to predict what will be seen in a shared environment. If a CML GPU Worker node is not available, you might see a message about *Insufficient nvidia* resources or pod *taint*.



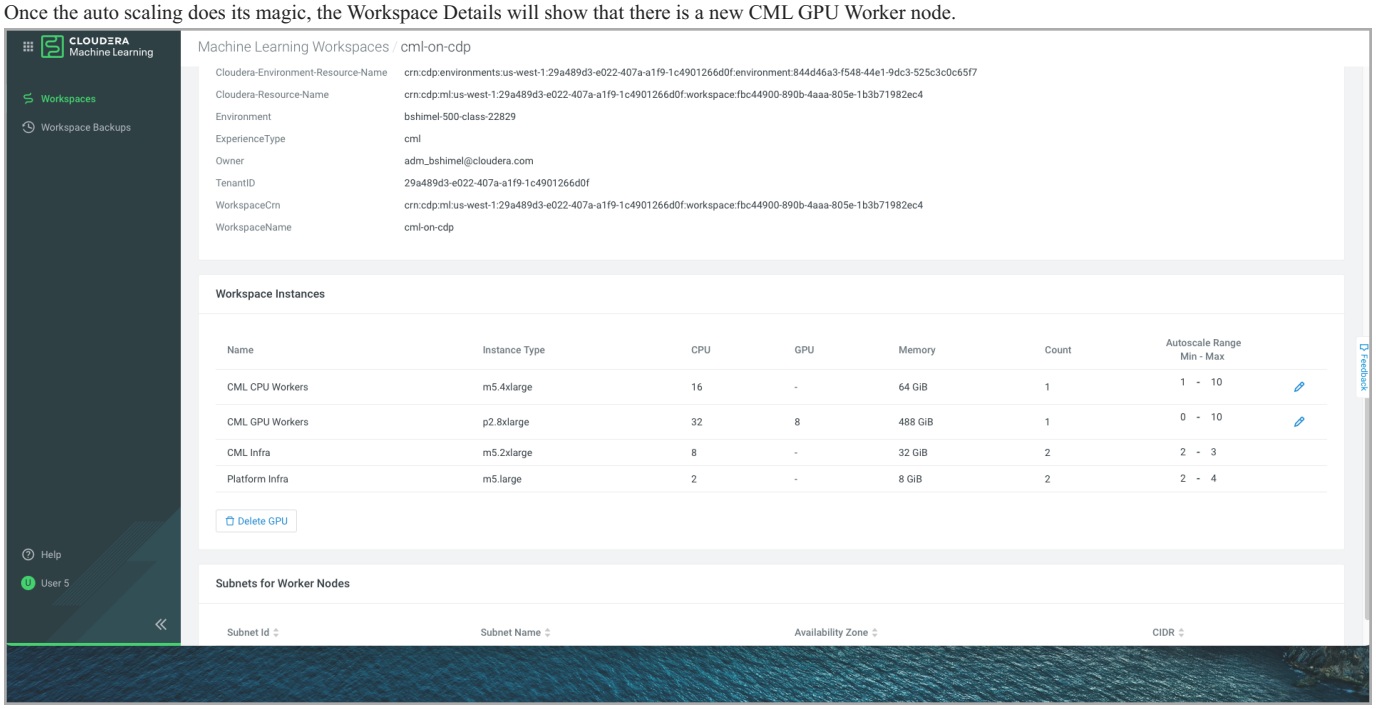
If you navigate back to Workspace Details in your other browser tab, you might see the CML GPU Worker node count still at zero.



If this the case, the auto scaling will take over and start up a CML GPU Worker node. You will see a message like the following.

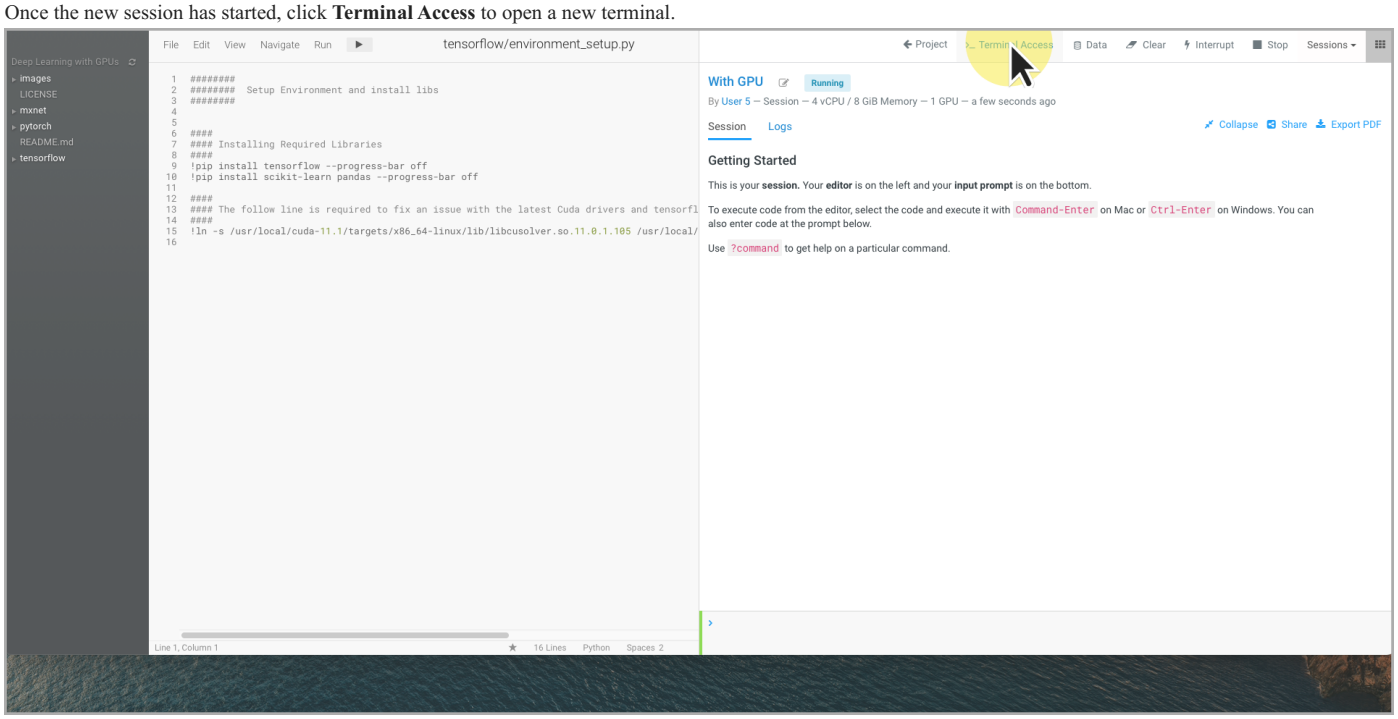


Once the auto scaling does its magic, the Workspace Details will show that there is a new CML GPU Worker node.

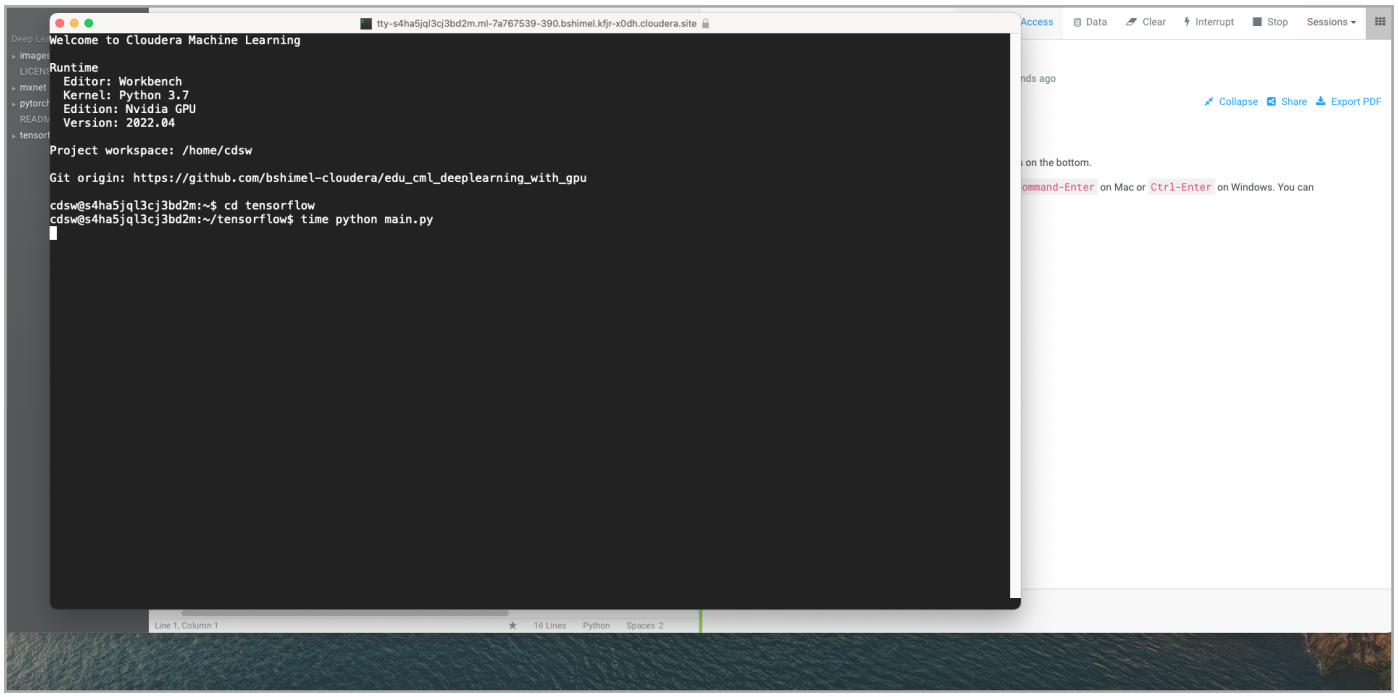


The whole process of auto scaling and adding a new worker node can take awhile (like ten minutes). Be patient, it is worth the wait.

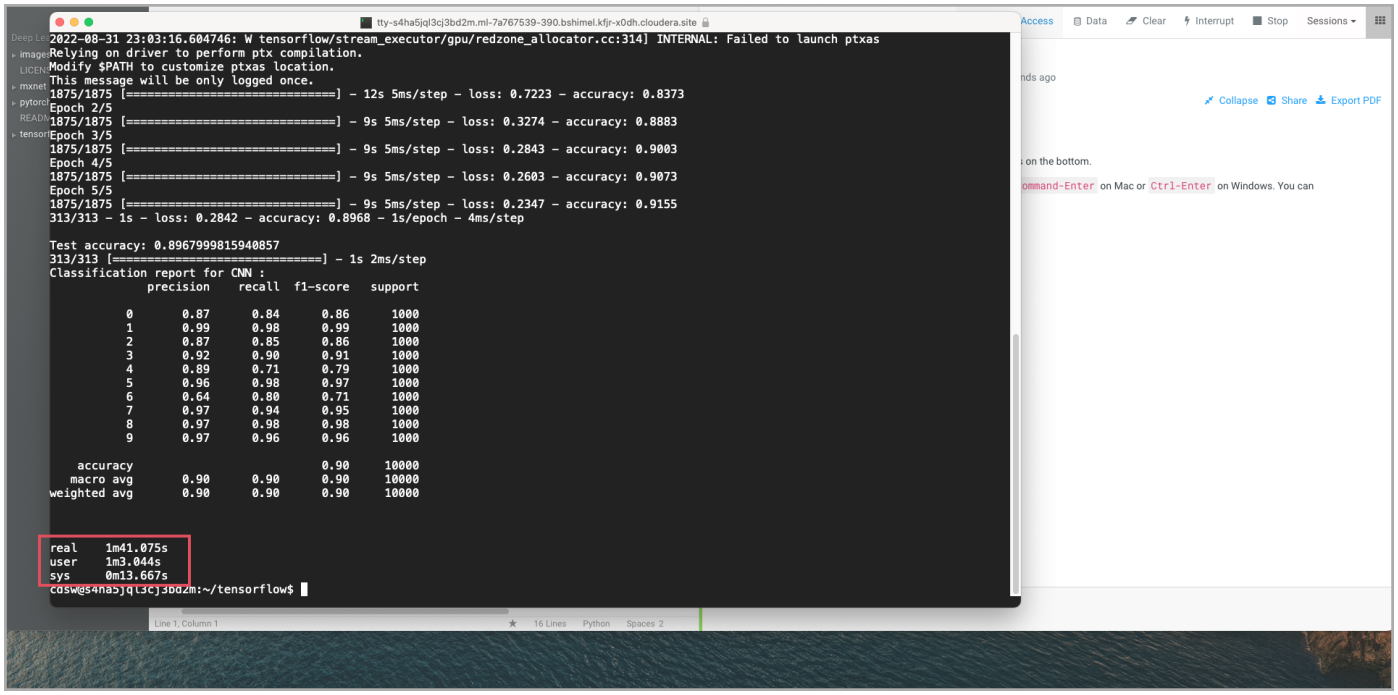
1. Once the new session has started, click **Terminal Access** to open a new terminal.



2. Enter `cd tensorflow`, then enter `time python main.py`.



3. Compare the new runtime using the GPU with your previous runtime without the GPU. The real runtime with a GPU should be about half of the real runtime without a GPU. Discuss the results with your instructor and classmates. Why is the user runtime so much greater for the non-GPU than the real runtime?



End of Exercise

Continuous Model Monitoring with Evidently

Monitoring a deployed machine learning model is critical to ensuring model performance over time. CML's model metrics combined with Evidently, an open source tool for evaluating, testing, and monitoring machine learning models, is a powerful tool to combat model drift.

In this exercise, you will:

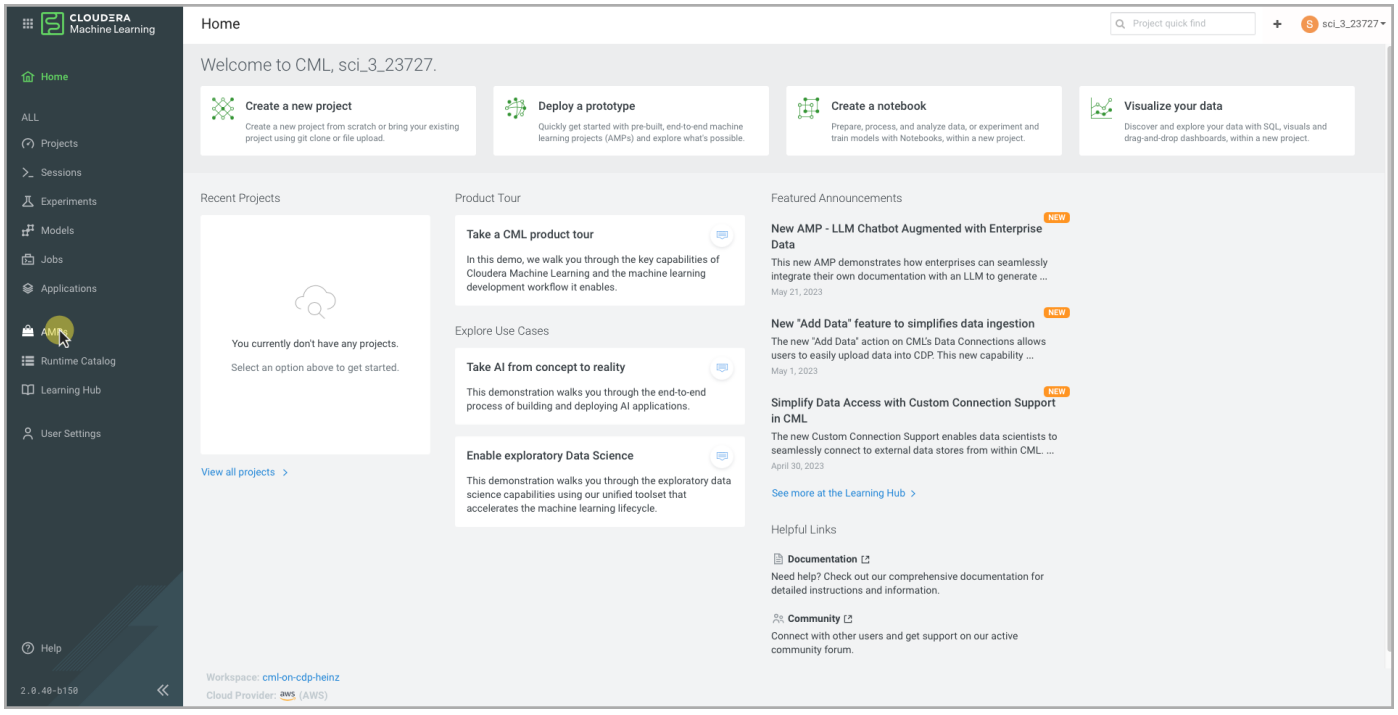
- Start the Continuous Model Monitoring AMP
- Launch the Price Regressor Monitoring dashboard
- Explore drift and variations in the model performance
- Identify the file that creates the Evidently dashboard
- Experiment with Evidently reports

Warning

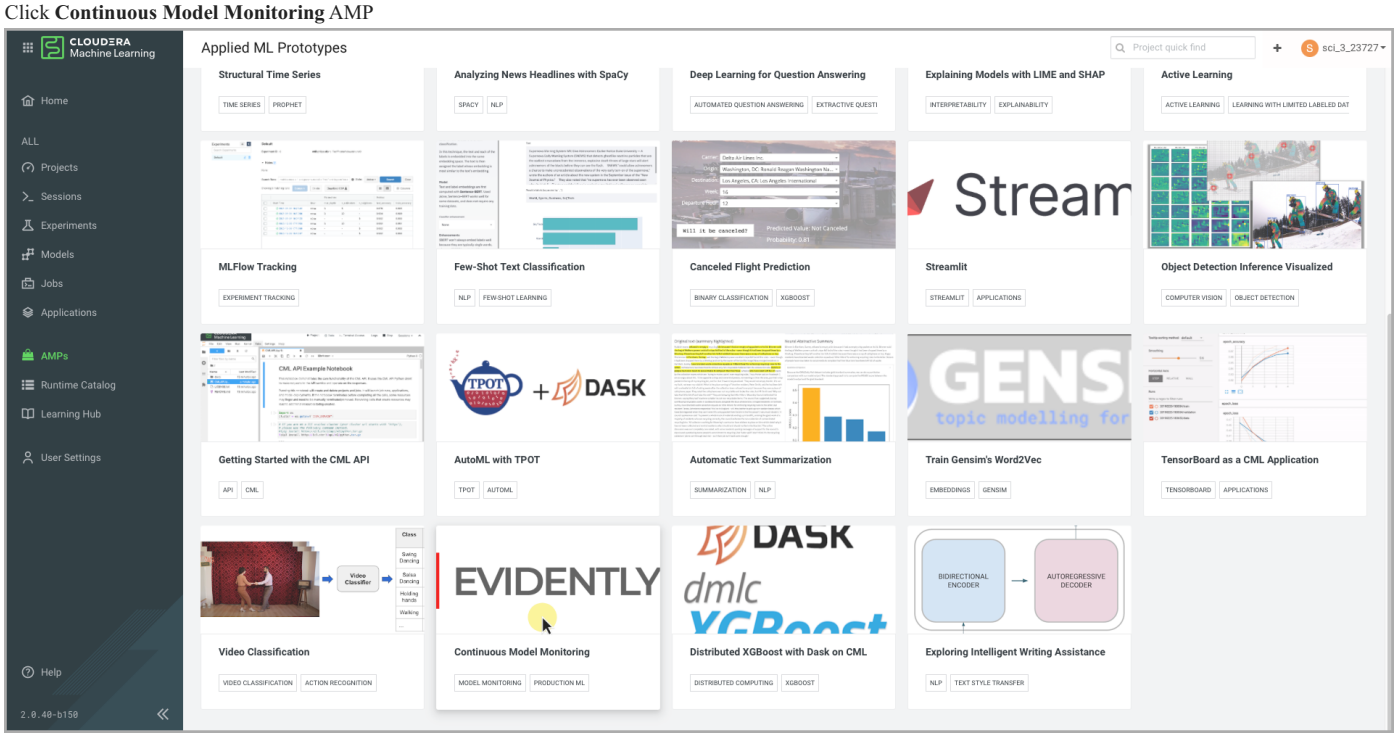
The initial deployment of the AMP takes approximately 30 minutes.

Start the Continuous Model Monitoring AMP

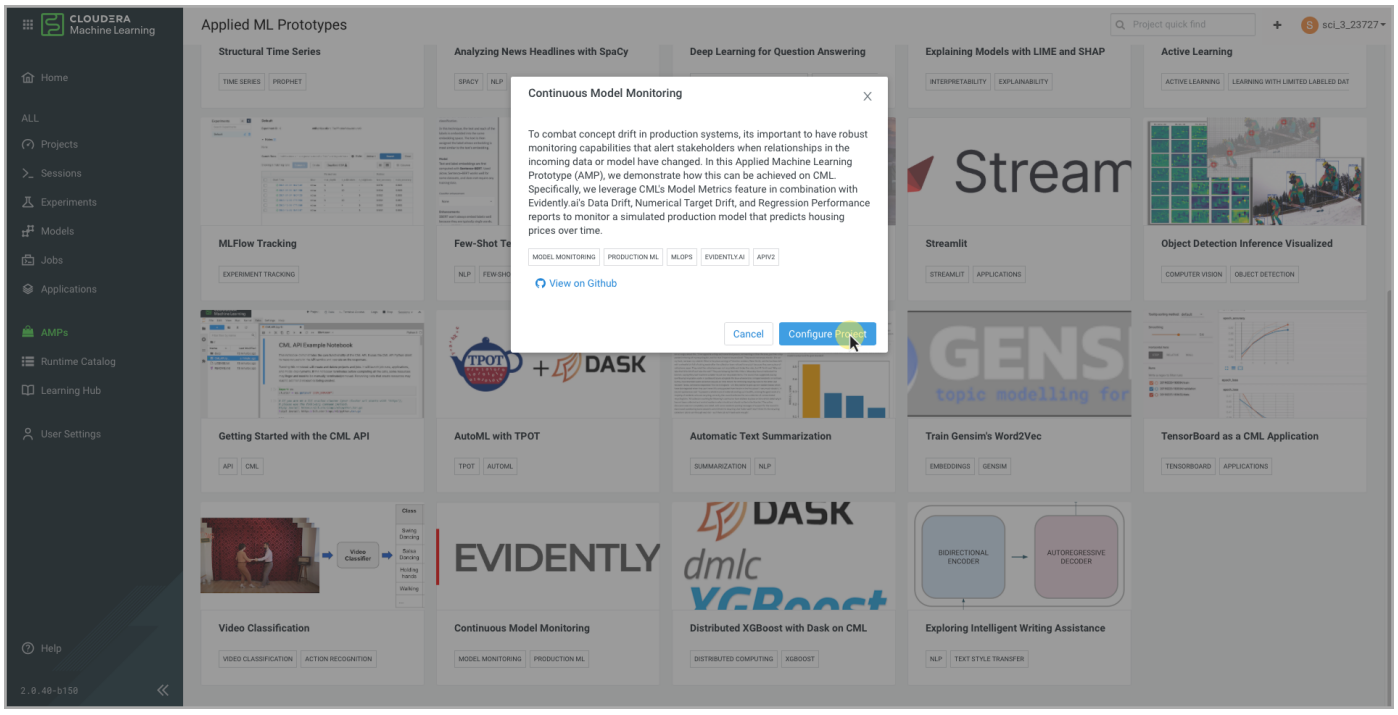
1. Click **AMPs**



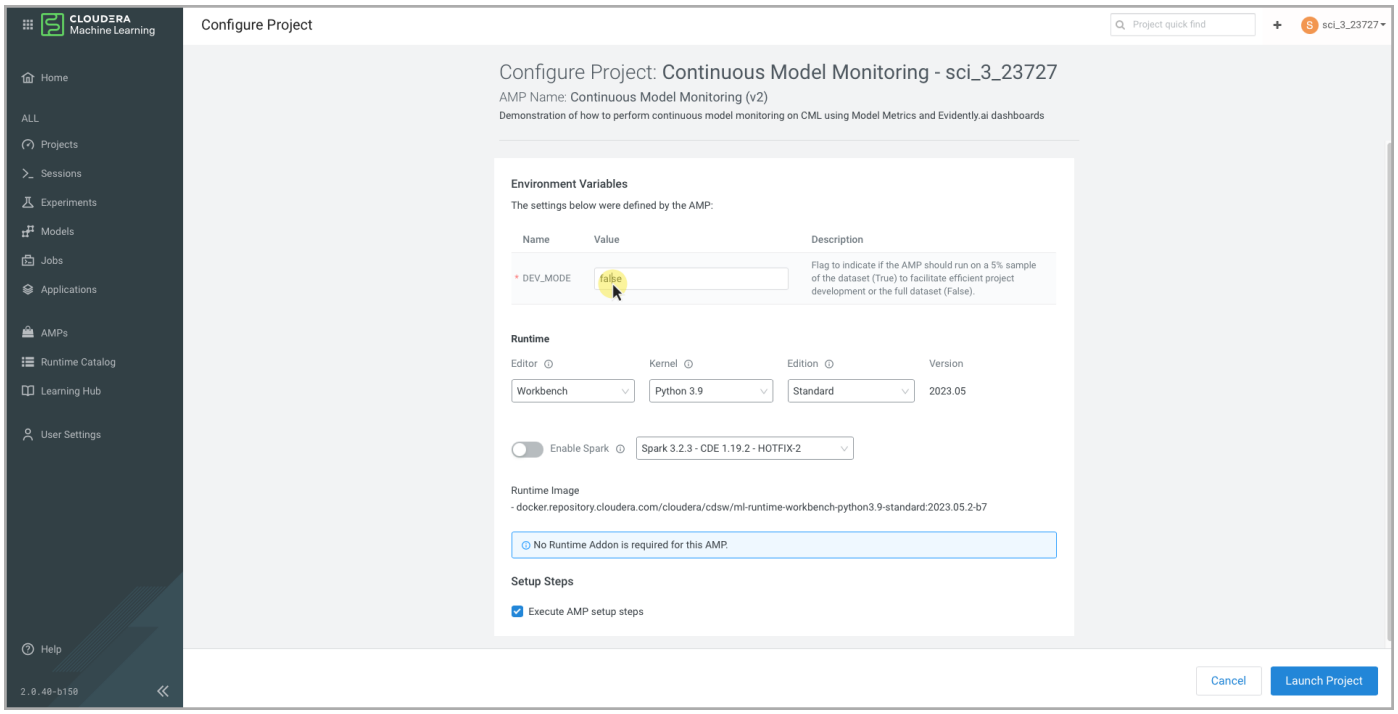
2. Click **Continuous Model Monitoring** AMP



3. Click **Configure Project**



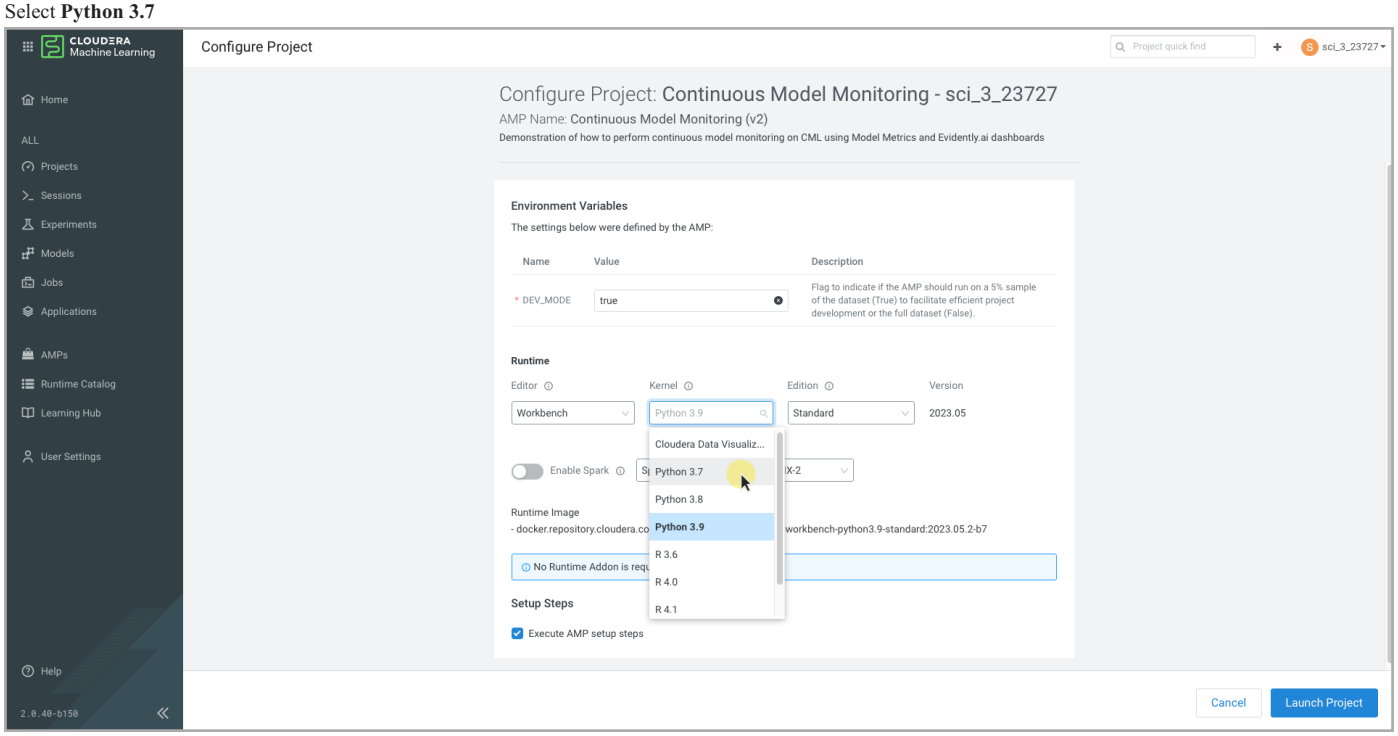
4. Set **Dev Mode** to `True`



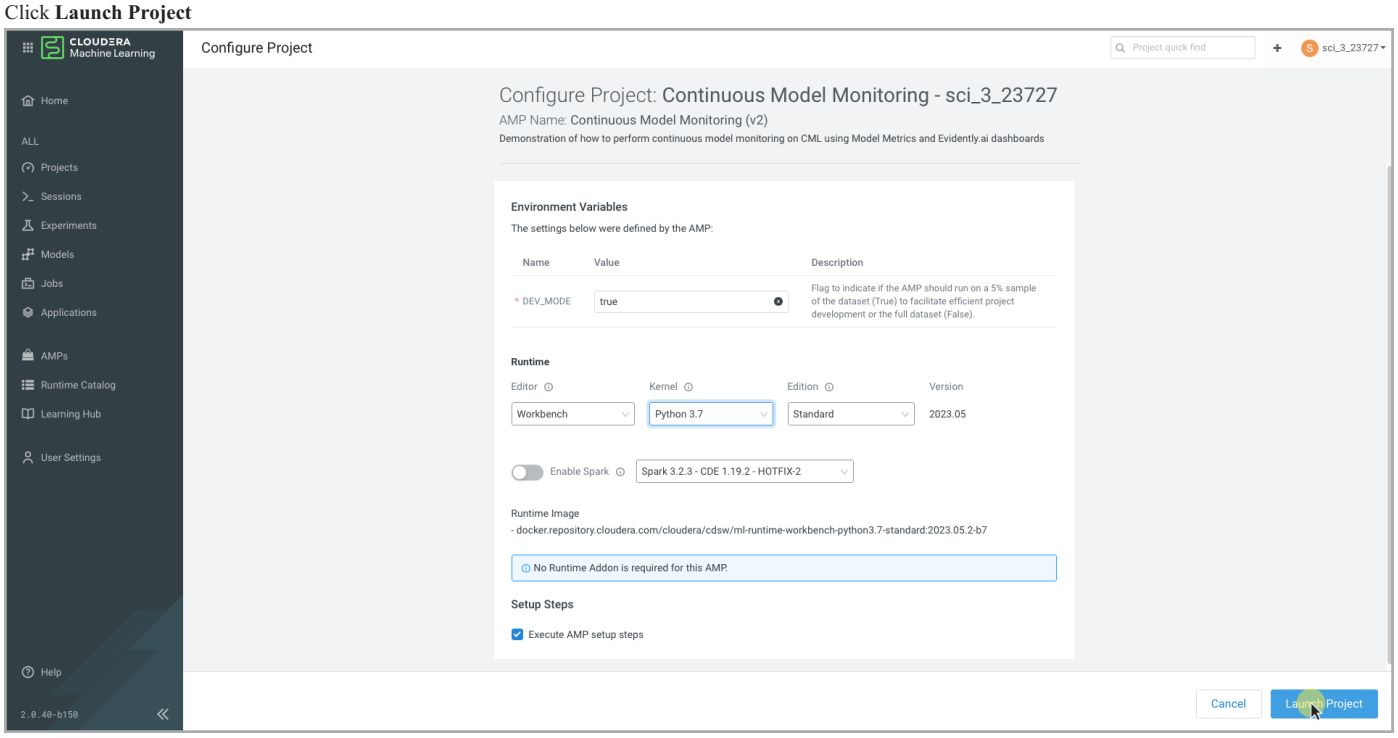
Warn

You must select Python 3.7 or the AMP will fail.

1. Select **Python 3.7**

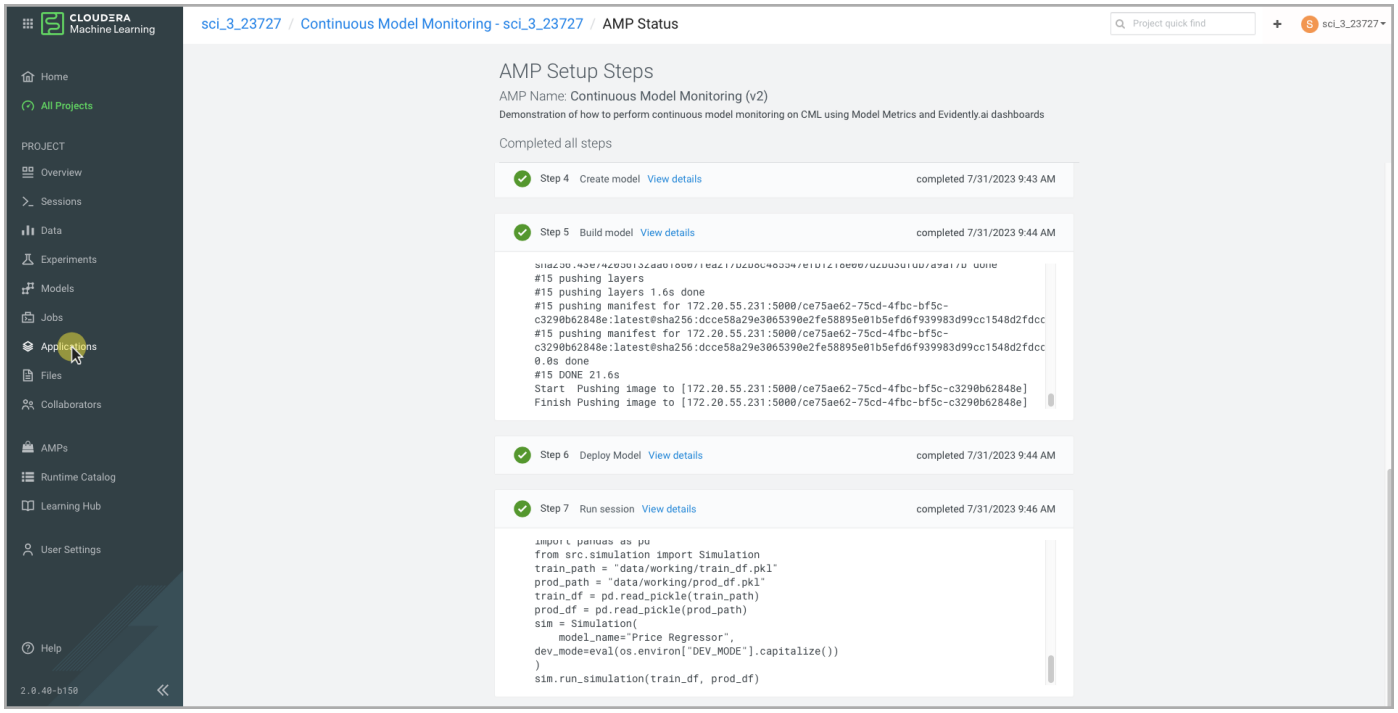


2. Click **Launch Project**

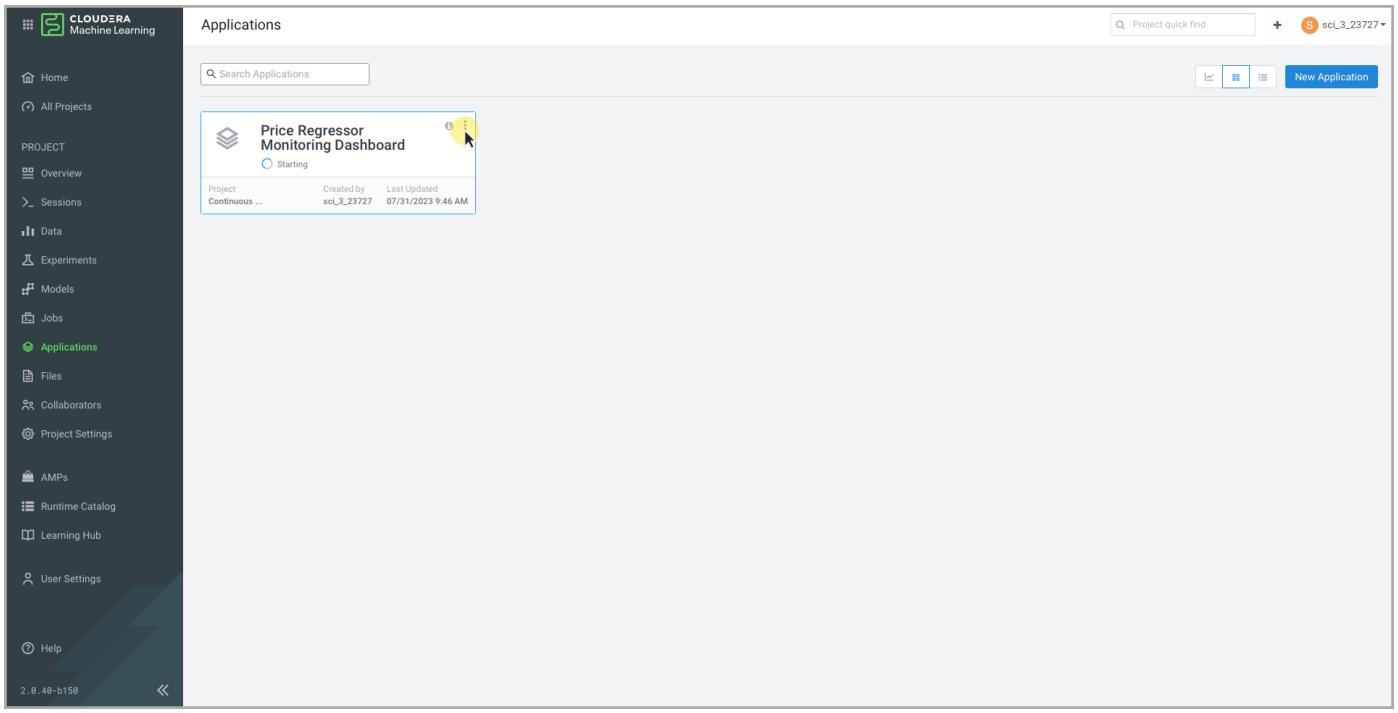


Launch the Price Regressor Monitoring dashboard

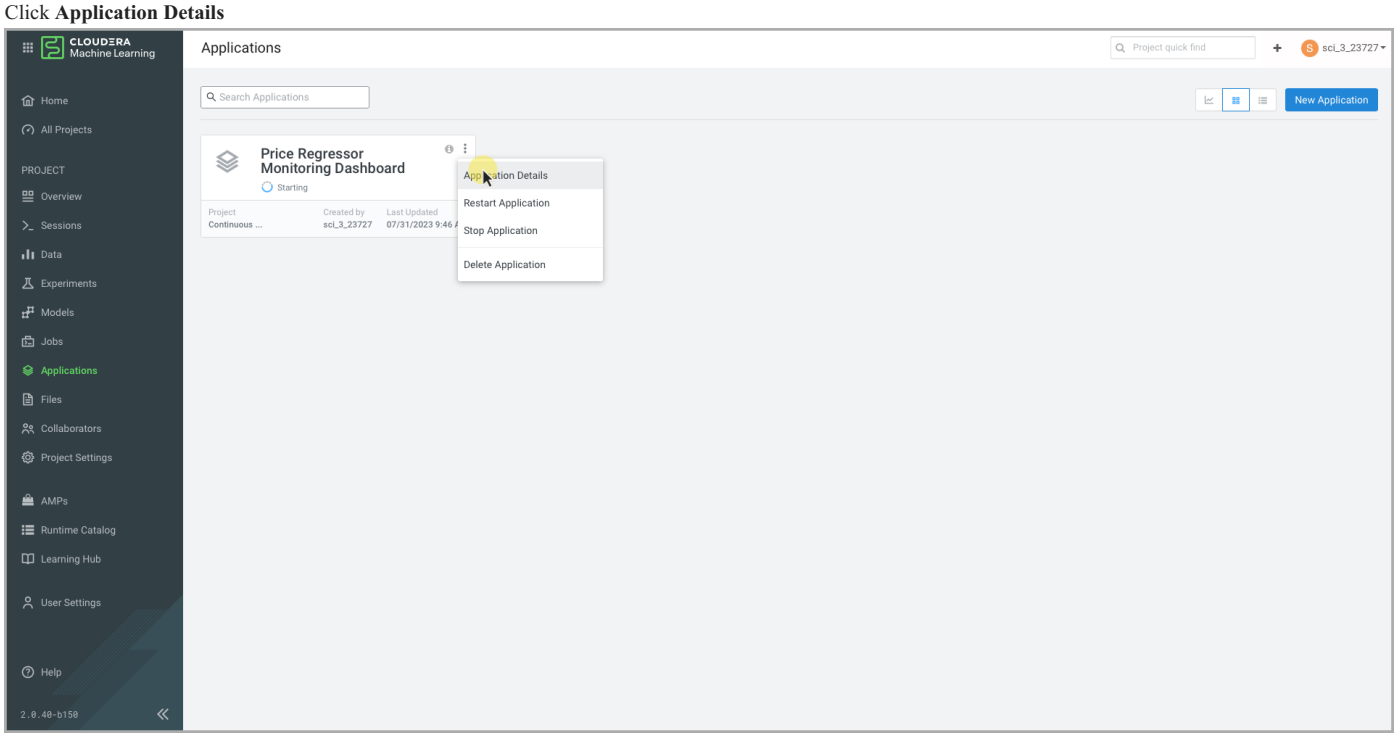
1. Click **Applications** in the left-side menu



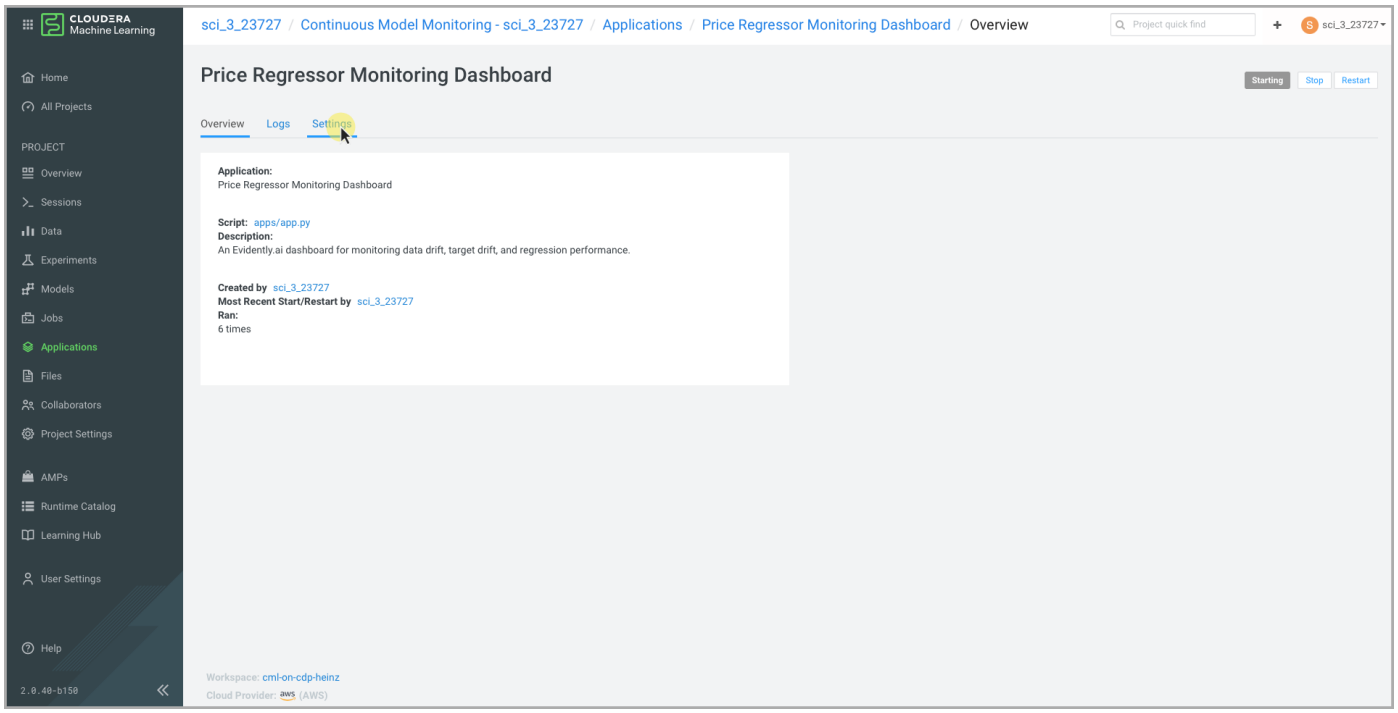
2. Click the application ⋮ menu



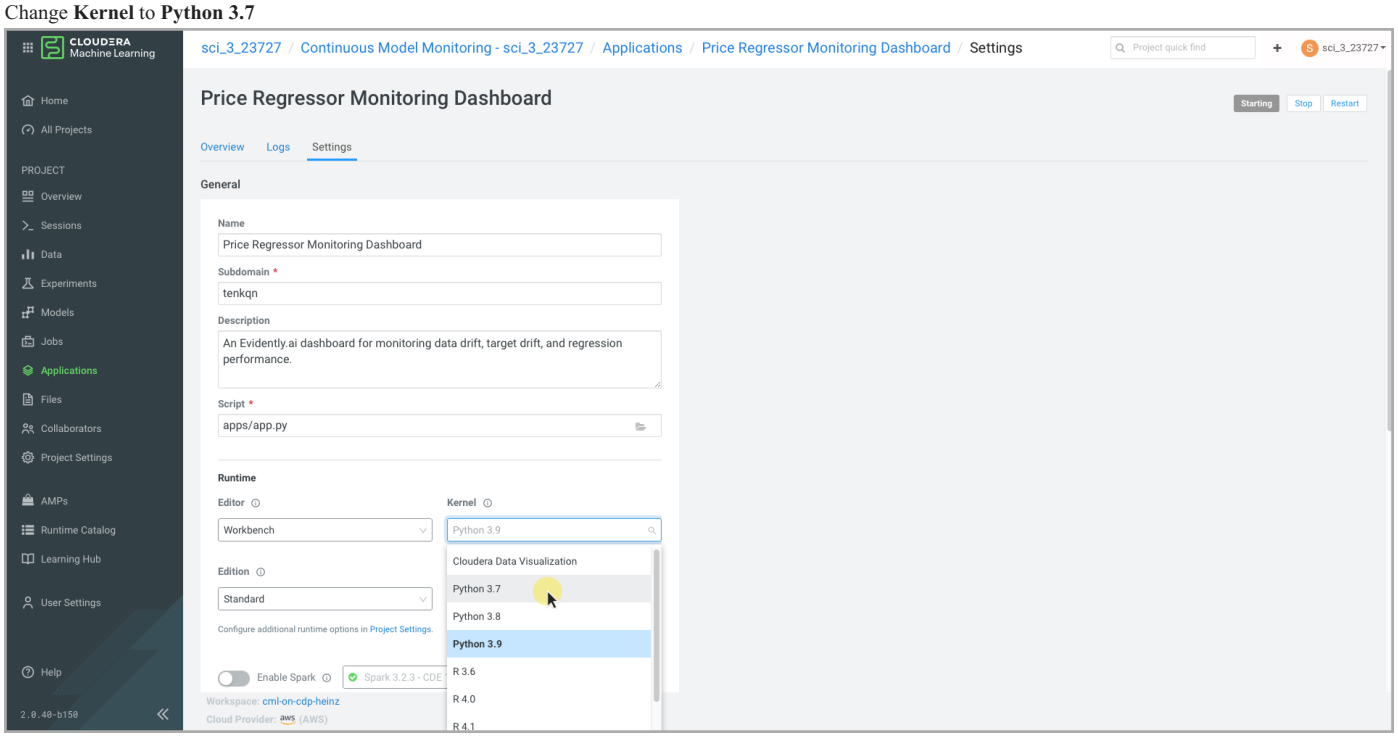
3. Click **Application Details**



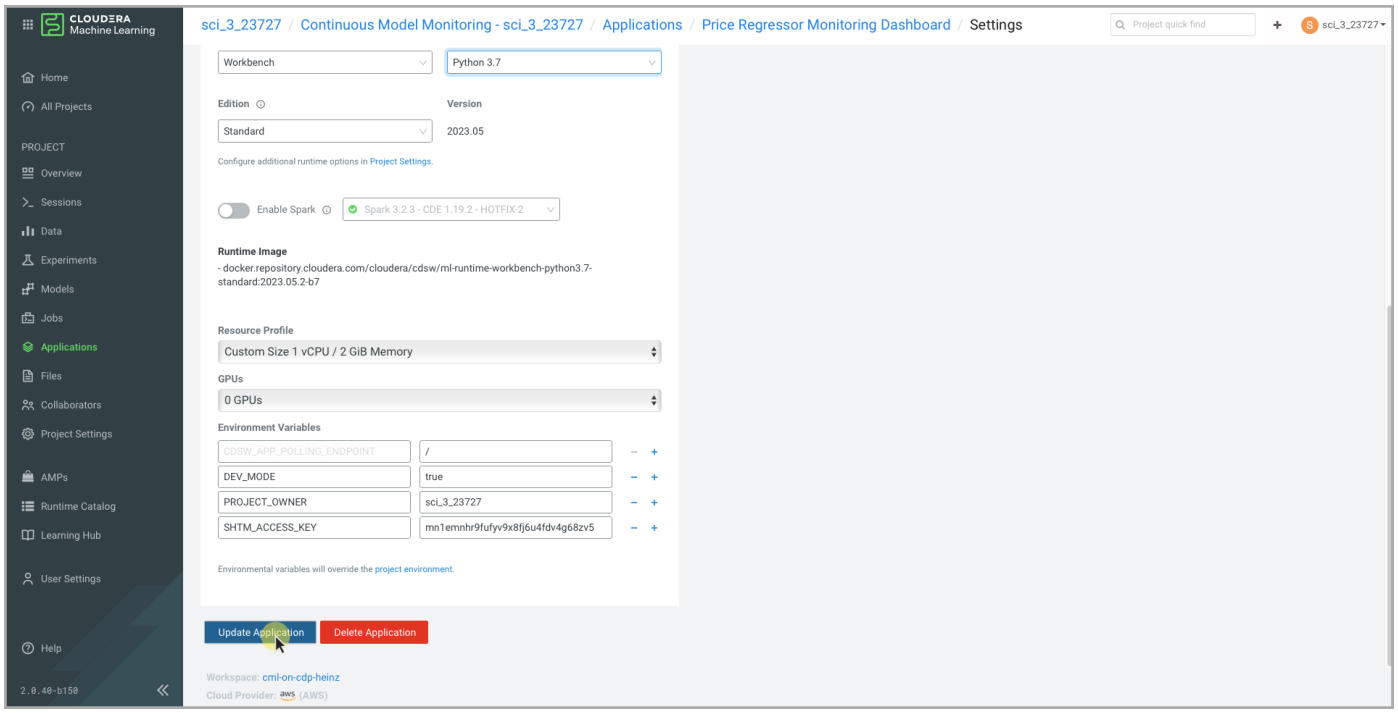
4. Click **Settings**



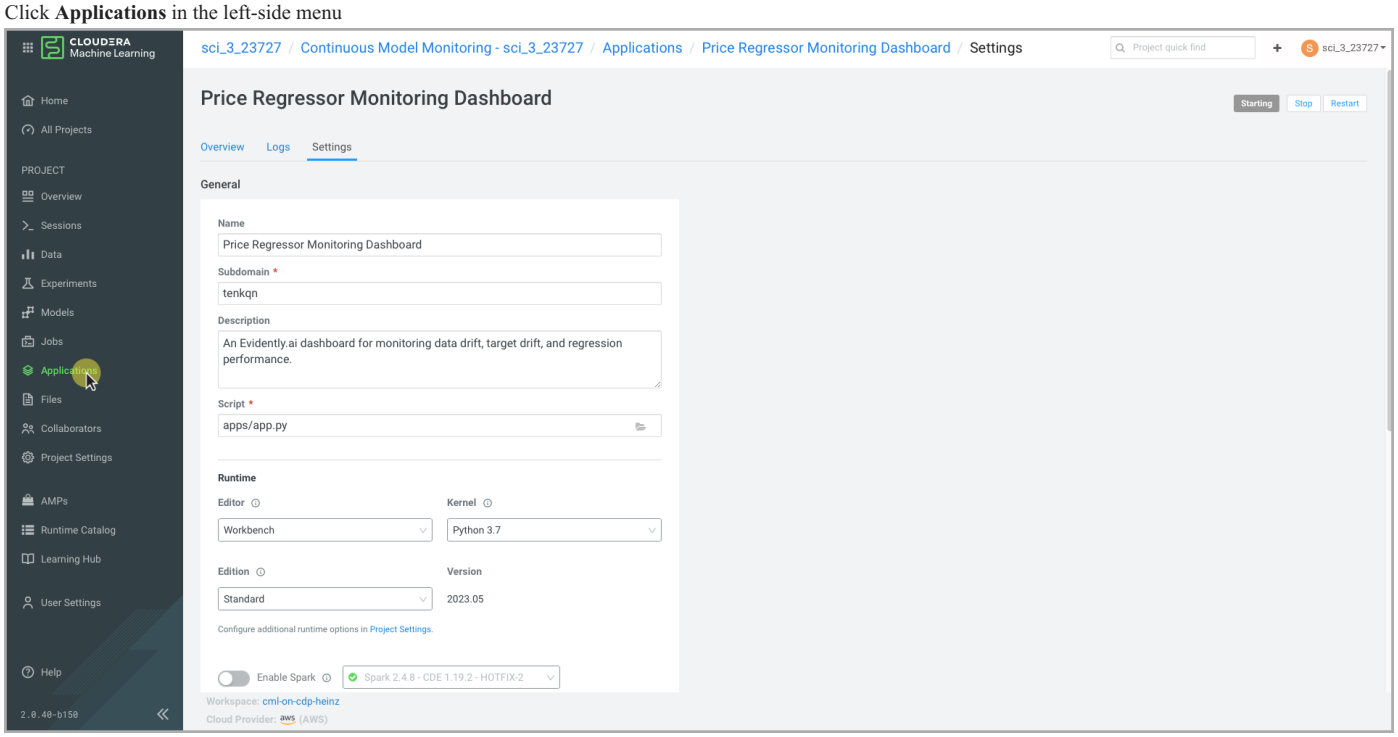
5. Change **Kernel** to **Python 3.7**



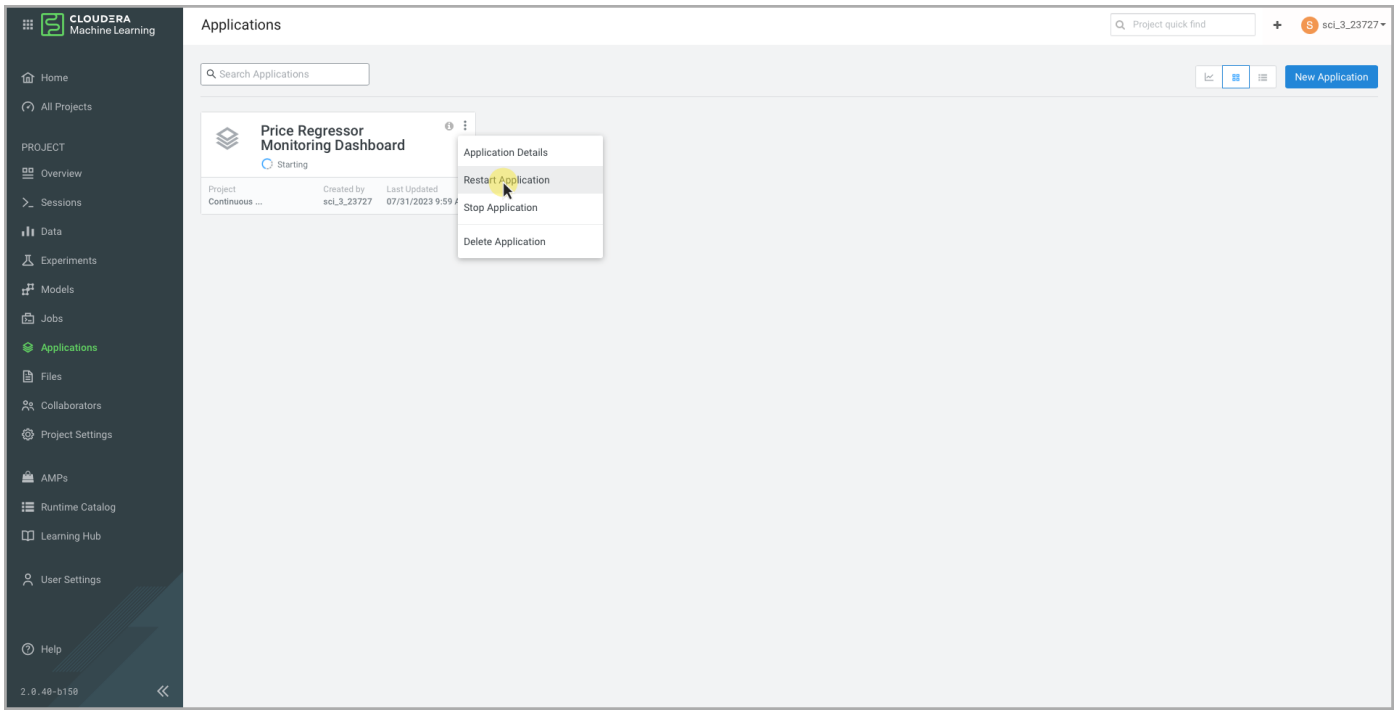
6. Click **Update Application** button



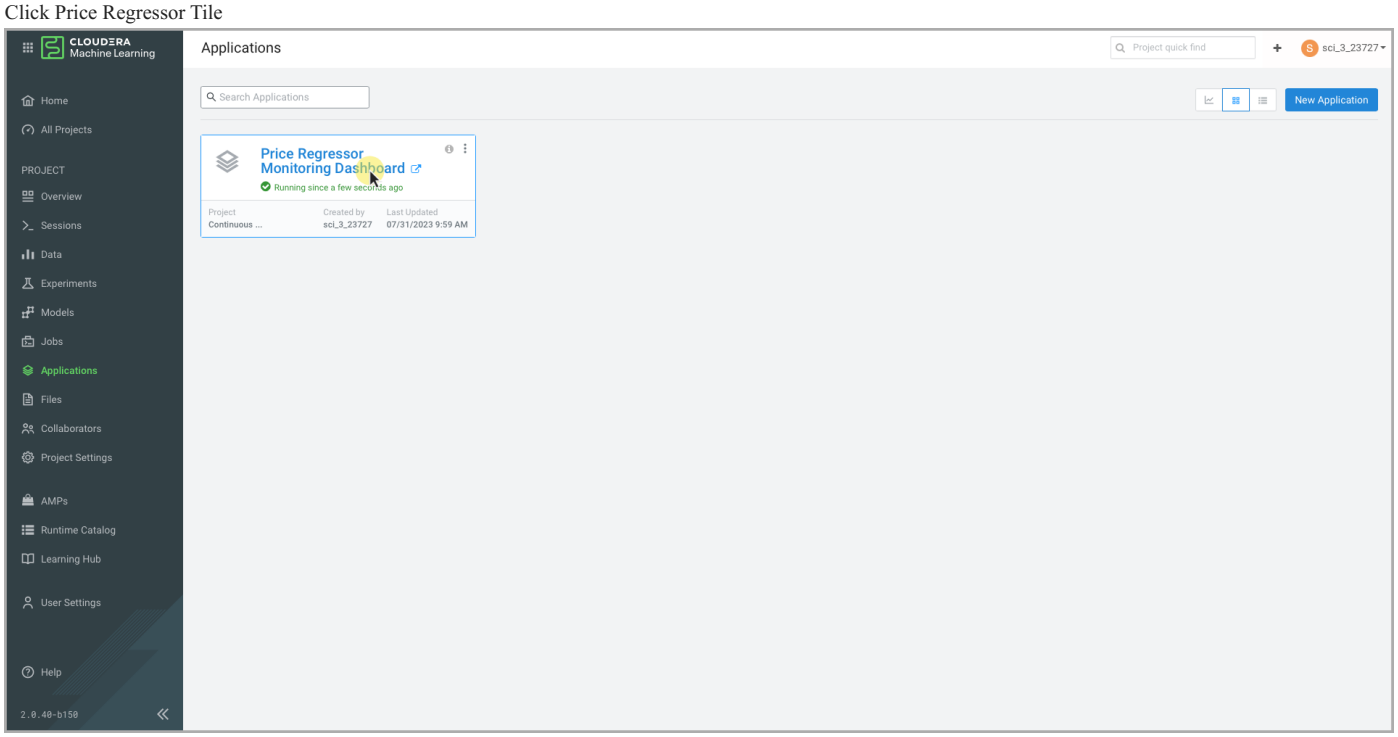
7. Click **Applications** in the left-side menu



8. Click the application ⋮ menu, and select **Restart Application**

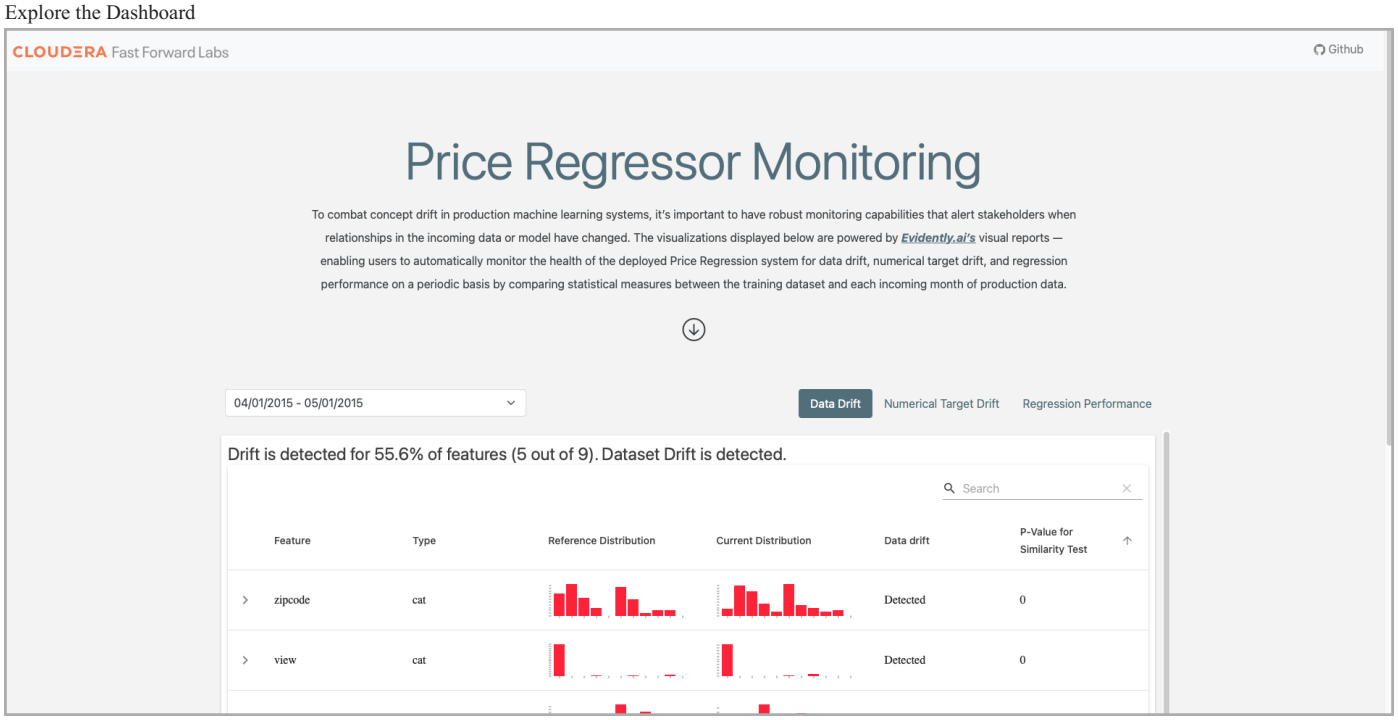


9. Click Price Regressor Tile



Explore drift and variations in the model performance

1. Explore the Dashboard



2. Explore the three tabs at different dates

- Is data monotonously drifting?
- Is there Numerical Target Drift at any point?
- Are there any significant variations in the model performance?

Identify the file that creates the Evidently dashboard

Project Structure

```
18.
├── LICENSE
├── README.md
├── .project-metadata.yaml              # declarative specification for AMP logic
├── apps
│   ├── reports                         # folder to collect monitoring reports
│   └── app.py                          # Flask app to serve monitoring reports
├── cdsw-build.sh                       # build script for model endpoint
├── data                                # directory to hold raw and working data artifacts
├── requirements.txt
├── scripts
│   ├── install_dependencies.py         # commands to install python package dependencies
│   ├── predict.py                      # inference script that utilizes cdsw.model_metrics
│   ├── prepare_data.py                 # splits raw data into training and production sets
│   ├── simulate.py                     # script that runs simulated production logic
│   └── train.py                        # build and train an sklearn pipeline for regression
├── setup.py
└── src
    ├── __init__.py
    ├── api.py                          # utility class for working with CML APIv2
    ├── inference.py                    # utility class for concurrent model requests
    ├── simulation.py                   # utility class for simulation logic
    └── utils.py                        # various utility functions
```

The Continuous Model Monitoring AMP project structure is above.

1. Select **Models** from the project menu, you see the Price Regressor model is deployed.
2. Click on the **Price Regressor** model.
3. Explore the model's properties. What file is the model based on? Which function is called?
4. Click the **Test** button to test the model. View the results.
5. Click **Files** in the project menu. Navigate to the `scripts/predict.py` file. Click the **Open in Workbench** button.
6. Go to line 87 and examine the `cdsw.track_metric` call that is used to store the prediction metrics in the CML PostgreSQL metrics database.

> **Note**
>
> Your CML workspace must have the **Enable Model Metrics** option selected when it is provisioned in order to track metrics.

The `src\simulation.py` file is the main simulation routine and mimics a production monitoring use case.

This simulation assumes the Price Regressor model has already been deployed, and accepts that model name as input. The .run_simulation() method operates the main logic of this class. Namely, it:

- Scores all training data against the deployed model so we can query metrics from the Model Metrics database for evaluation
- Initializes a simulation clock, which is just a list of date ranges from the prod_df to iterate over. These batches mimic the cadence upon which new data "arrives" in a production setting.
- For each simulation clock date_range, we:
- Query the prod_df for newly *listed* records and score them using deployed model
- Query the prod_df for newly *sold* records and add ground truths to metric store
- Query the metric store for those newly *sold* records and generate new Evidently report
- Redeploy the hosted Application to surface the new monitoring report

1. Navigate to `src/simulation.py` and open the file in the Workbench Editor.
2. Examine the `query_model_metrics` method on line 340. This method reads the metrics stored in the CML model metric database.
3. Examine the `build_evidently_reports` method on line 404. This method builds the reports and saves them as HTML to be served by the Price Regressor Monitoring Dashboard application.
4. Go to line 186 and examine how the simulation deploys the updated Price Regressor Monitoring Dashboard application upon simulation completion. This is an example of using the CML API to deploy an application. More details can be seen in the `src/api.py` file.

End of Exercise