**Data Science Development in Cloud Pak for Data**

Elena Lowery, WW Team, Data Science and IA

[elowery@us.ibm.com](mailto:name@ibm.com)

A close up of a logo

Description automatically generated

**Table of contents**

[Overview 1](#_Toc79589292)

[Required software, access, and files 1](#_Toc79589293)

[Part 1: Create a Project and Load Data 3](#_Toc79589294)

[Add Data and Connections to a Project 3](#_Toc79589295)

[Working with Data 6](#_Toc79589296)

[Write Data to a Project 13](#_Toc79589297)

[Write Data to a Database 14](#_Toc79589298)

[Part 2: Environment configuration 16](#_Toc79589299)

[Package management 21](#_Toc79589300)

[Part 3: Collaboration and Git Integration 26](#_Toc79589301)

[Option 1: Local collaboration 29](#_Toc79589302)

[Option 2: Git Collaboration for non-JupyterLab assets 30](#_Toc79589303)

[Option 3: Git Collaboration for JupyterLab assets 32](#_Toc79589304)

[Part 4: Importing Notebooks into JupyterLab 40](#_Toc79589305)

[Importing notebooks 40](#_Toc79589306)

[Appendix A: Getting a Git Repo Token 42](#_Toc79589307)

[Appendix B: Setting up a Git repo 44](#_Toc79589308)

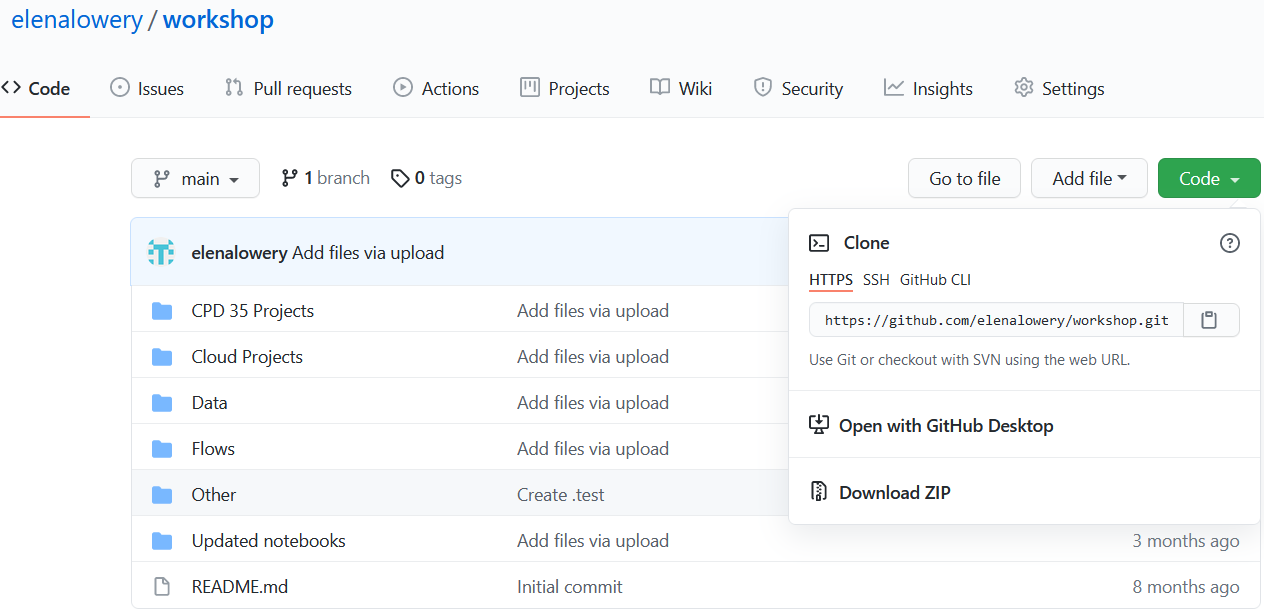
# Overview

In this lab you will learn how to complete the following tasks in **Cloud Pak for Data**:

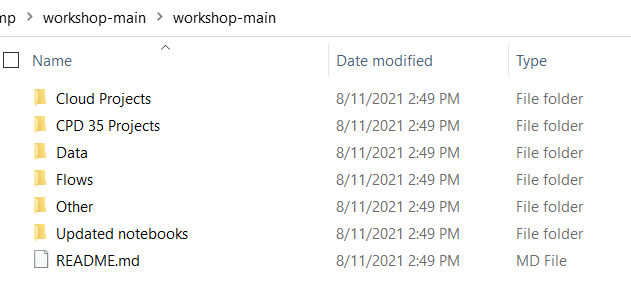
* Create a project
* Connect to data and import data
* Create notebooks
* Configure environments
* Collaborate with other users
* Integrate with Git
* Use JupyterLab

# Required software, access, and files

* To complete this lab, you will need access to a **Cloud Pak for Data** (CPD) cluster with **Watson Studio** and **WML**.
* Instructions in this lab are for CPD version 3.5.
* You will need a Git account (for example, a free account on [www.github.com](http://www.github.com)) to complete the Collaboration and Git integration as well as JupyterLab sections of the lab. You will also need a token for the git repo.
  + See **Appendix A** and **B** for Git setup instructions.
* You will also need to download and unzip this GitHub repository: https://github.com/elenalowery/workshop



* Unzip the files until you get to this directory structure:



In the lab we will refer to this folder as the git repo folder.

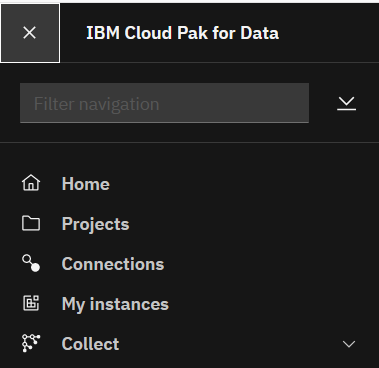
# Part 1: Create a Project and Load Data

In CPD a *project* is the container for all asset types created in the platform – notebooks, scripts, Modeler flows, RStudio applications, and other types of assets. A project also contains environments, which are runtime software and hardware configurations that can be associated with asset types like notebooks. Projects provide secure access to collaborators and can be used to store data or connections to data.

In this section you will learn the basics of working with projects.

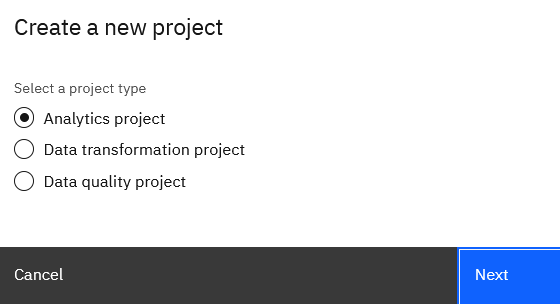
### Add Data and Connections to a Project

1. Log in to a **CPD cluster**. The instructor will provide a URL and userid/password.
2. From the main menu in the top left corner select **Projects**.

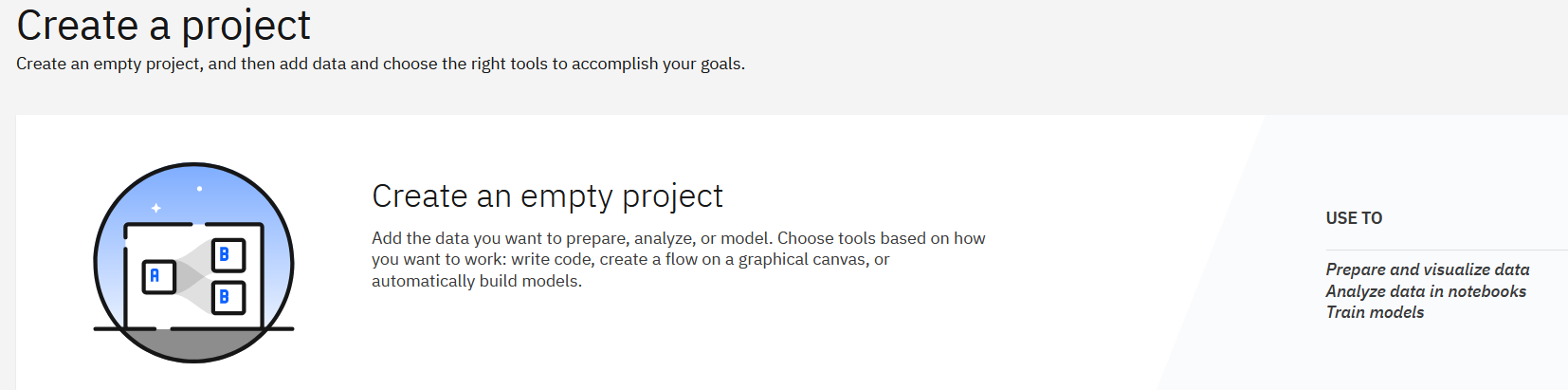


1. Click on the **New Project** button, then select the *Analytics project* radio button and click **Next**.

*Note: you may see a different list of projects. The options depend on the services that are installed in CPD.*



1. Click **Create an empty project**



1. Provide a project name, for example, *IBM Workshop*.

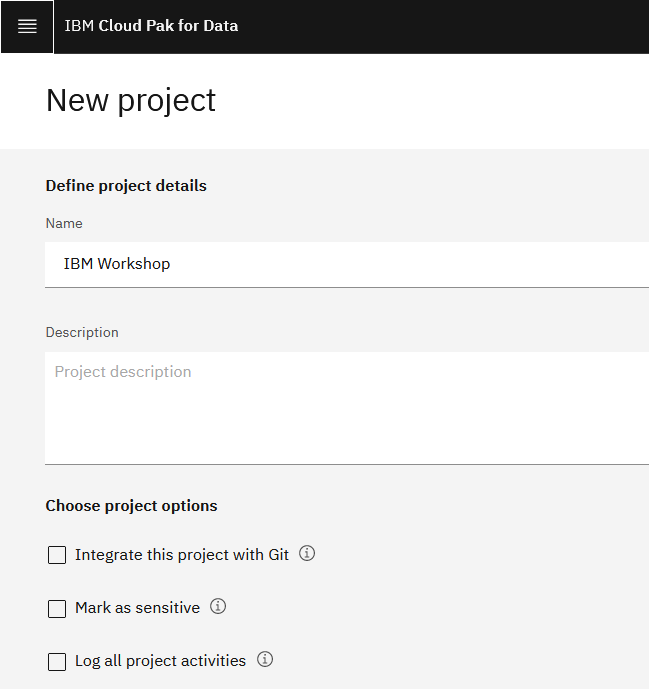
Review details of project options by clicking on the information icon:

* + *Git integration*
  + *Mark project as sensitive*
  + *Log all project activities.*

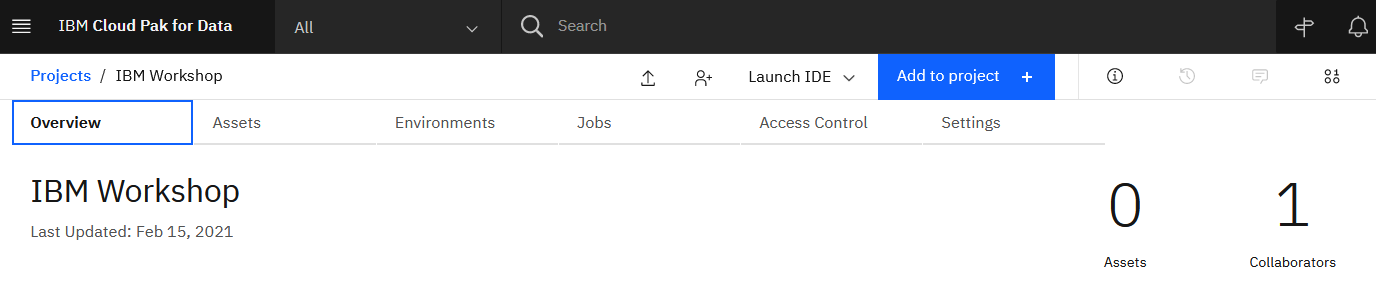
Note that Git integration can only be enabled during project creation. The other two options can be enabled/disabled in project **Settings** after the project has been created.

At this time we are not going to integrate the project with Git*.*

Click **Create**.

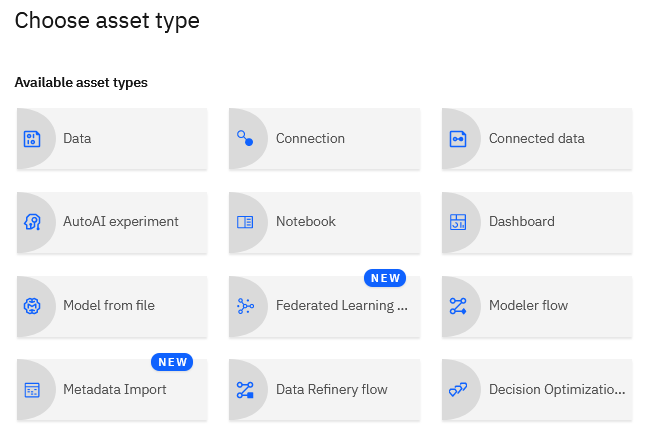


1. Notice the different tabs in the project. We will review them later in the lab.



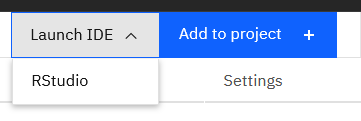
1. Click the **Add to project** button and review the various assets that can be added.

*Note: your screenshot may look different – it depends on the services installed in the cluster.*



In CPD JupyterLab and RStudio are integrated as IDEs. JupyterLab requires project integration with Git. When the project is configured with Git, we will see a **Launch IDE** option for JupyterLab. If you don’t see RStudio, that means that RStudio service was not installed in the CPD cluster.

We did not integrate this project with Git, and that’s why we don’t see the JupyterLab option.



### Working with Data

Since Cloud Pak for Data allows installing any Python or R library into development environments, a data scientist can use any option for accessing data. We recommend that data scientists use database drivers that are provided with the platform.

Supported data sources are listed in documentation: <https://www.ibm.com/docs/en/cloud-paks/cp-data/3.5.0?topic=data-supported-sources>

Cloud Pak for Data supports code generation for reading data from several types of data sources. If code generation is not supported, we can still take advantage of security accessing data sources credentials that are defined in *Project Connections*. List of data sources that support code generation: <https://www-ibmdocs.ak.origins.s81c.com/docs/en/cloud-paks/cp-data/3.5.0?topic=notebook-support-loading-data-in#file-types>

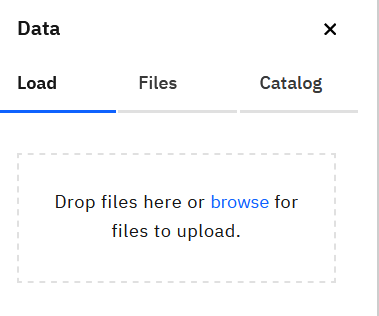
In the rest of this section you will learn how to load data and configure data sources.

First, we will import data (a csv file) into the project. When the file is loaded, it will be considered a project *Data Asset*.

When we use this option, data is stored in the project. It will be accessible to all collaborators that have project-level access, in other words, there is no file-level security. If security is a concern, then remote data sources (databases, object storage, Hadoop, etc.) should be used. In general, importing files should be used for experimentation and quick testing. Other file types, not just .csv, can be loaded into *Data Assets*.

1. Click **Add to Project -> Data**.

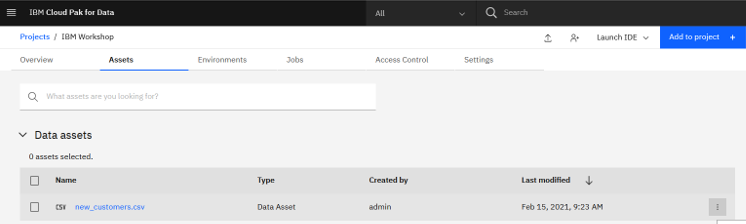
Select **Browse** in the right navigation panel



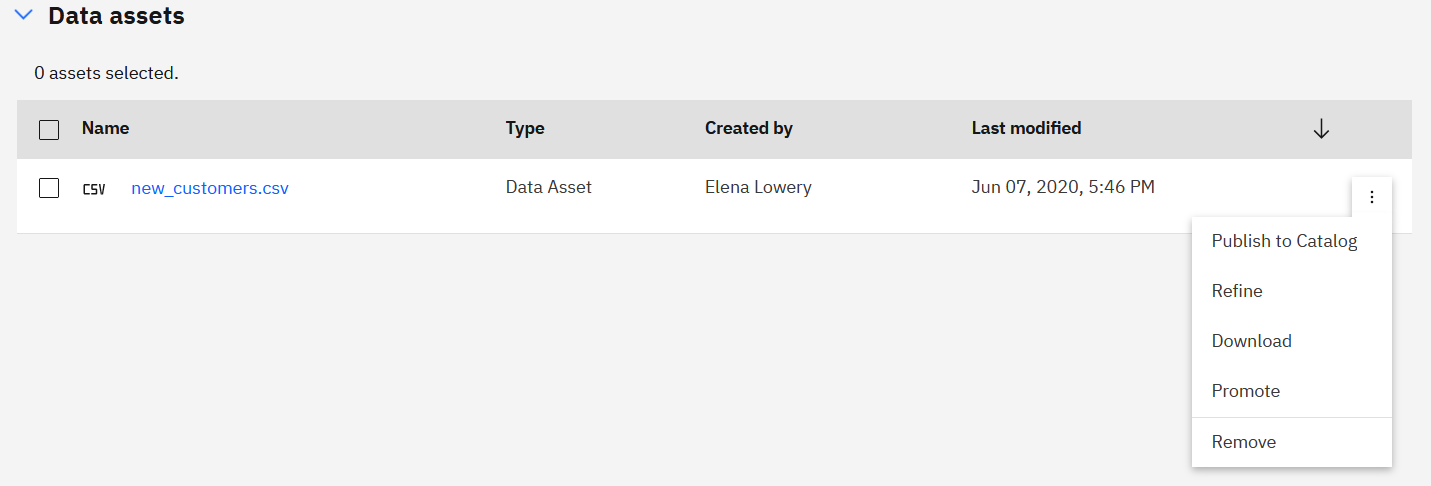
Browse to the *git repo/data* directory and load one of the data files, for example, *new\_customers.csv*.

1. Click on the **Assets** tab.

The uploaded file is displayed under **Data assets** section of the project.



Action items applicable to data assets can be displayed by clicking on the vertical ellipses in far-right column (also known as the *More* button).

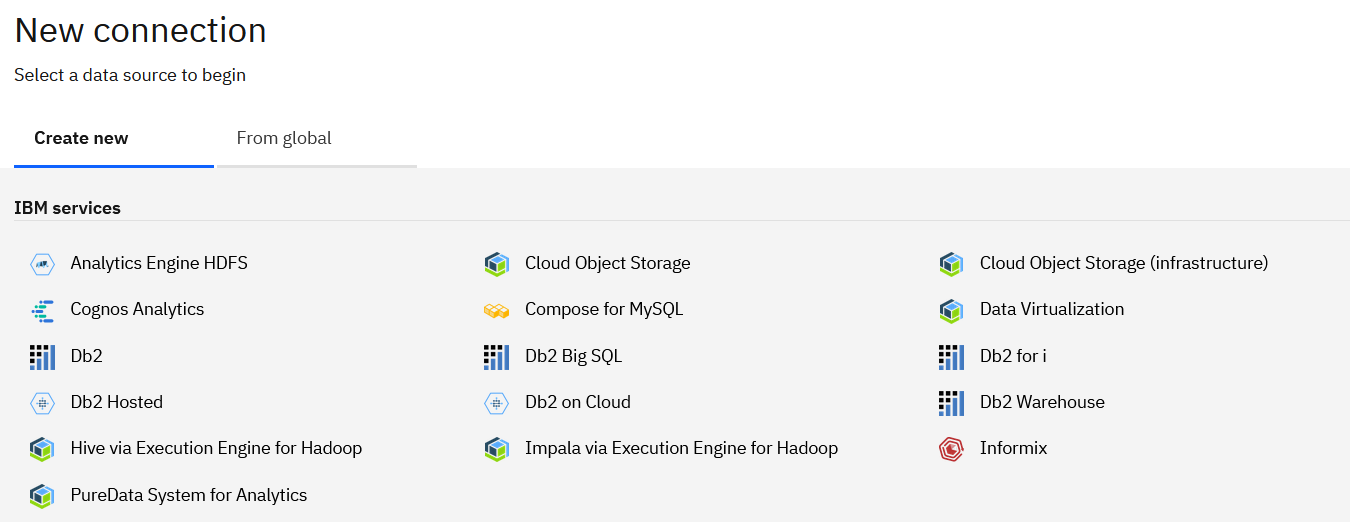


We will review some of these options later in the lab.

*Note: In this lab we explain how to work with csv files. It’s possible to load any file type into Data Assets. CPD supports code generation for csv, JSON, and Excel files. To access all file types from Notebooks or RStudio, you can use the project-lib library. More information about accessing files from Data Assets in documentation: https://www.ibm.com/support/knowledgecenter/en/SSQNUZ\_3.5.0/wsj/analyze-data/load-and-access-data.html*

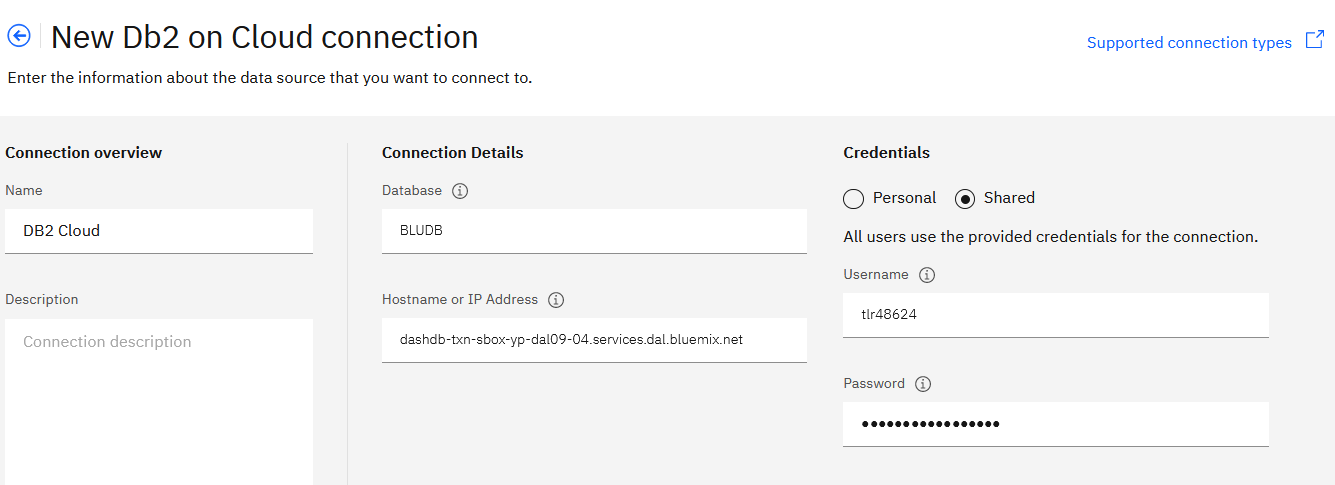
Next, we will create a connection to the database and a remote data asset. Your instructor will provide database information.

1. Click **Add to Project -> Connection.**
2. Select *DB2 Cloud*.



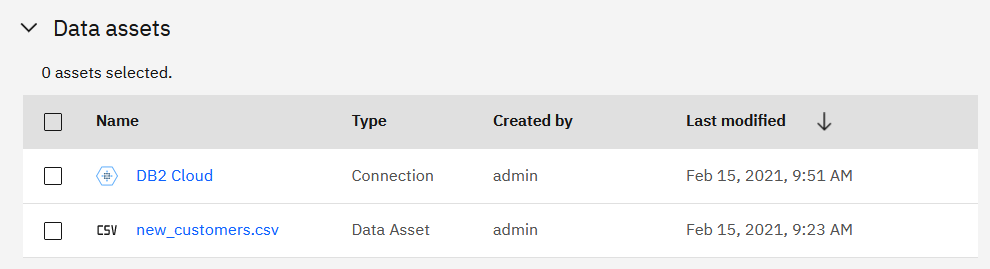
1. Enter connection information (provided by the instructor)

*Note: in this lab, we will use a* ***shared*** *credential. We recommend that you use a* ***personal*** *database credential in your environment.*



Click **Test**, and if the test was successful, click **Create**.

1. Notice that connection to *DB2 Cloud* is now saved under **Data Assets**.



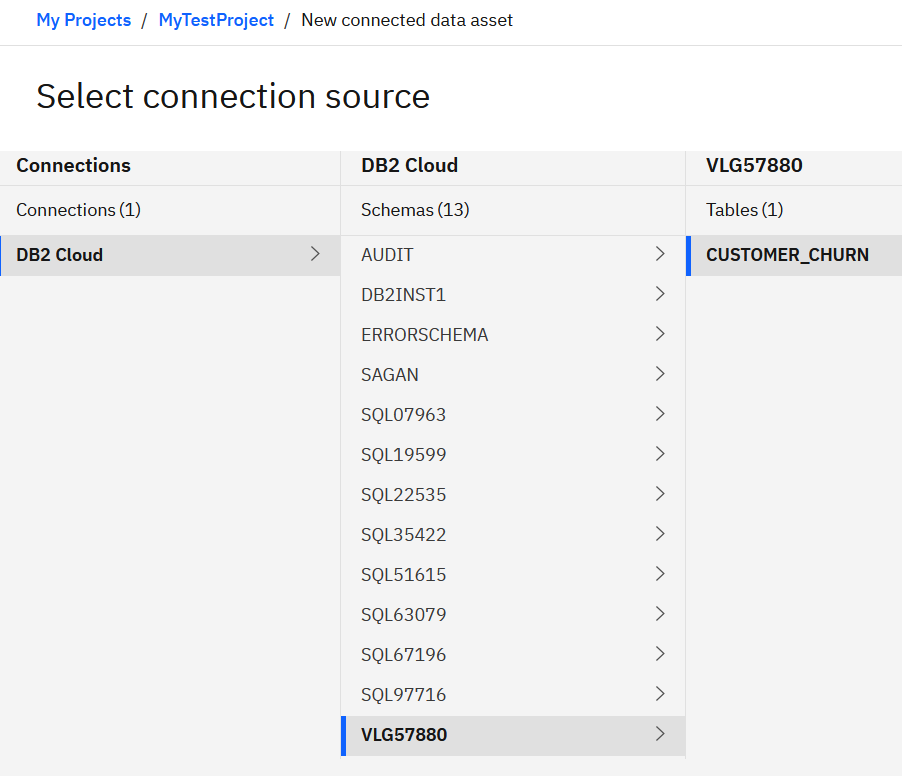
You can modify the connection by clicking on it. Additional actions for the connection are available by clicking the More icon.

*Note: Unlike the file that’s imported into the project, remote connections do not import data until requested by a user action (for example, a menu in an IBM tool or code in a notebook).*

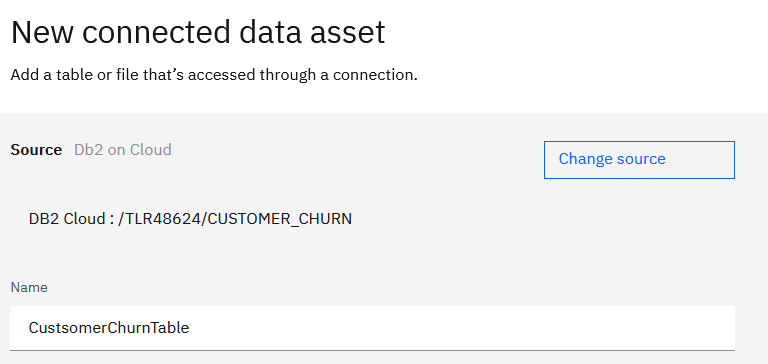
Next, we will add a table from the *DB2 Cloud* data source.

1. Click **Add to Project -> Connected Data.**
2. Click **Select source,** then click on *DB2 Cloud* connection, schema and table names provided by the instructor**.**

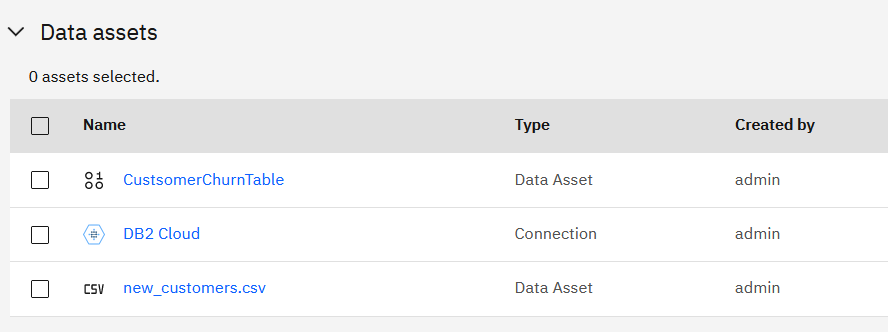
Click **Select**.



Provide the name that will be displayed in the project and click **Create**.



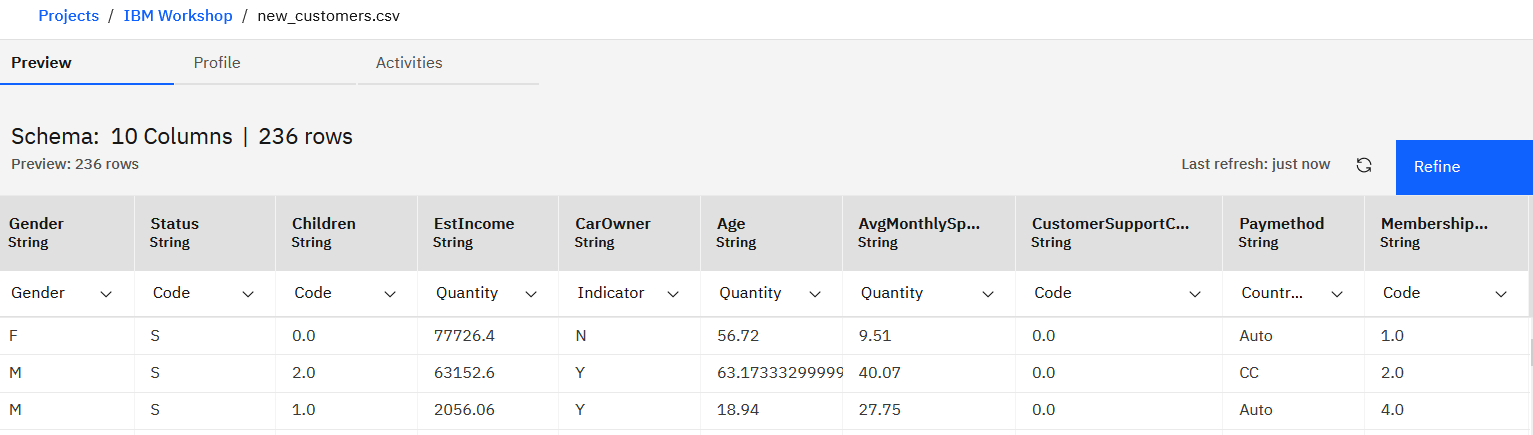
The table is now displayed under **Data Assets**.



In this lab we explained how each user can add files and connections to a project. CPD also supports adding *Platform Connections* and *Catalog Connections*, which can be reused by all users in all projects. Instructions for setting up these connections can be found in **Appendix C**.

Next, we will review some actions that are available for data assets.

1. Click on *new\_customers.csv*. A file preview opens.



The **Profile** function requires integration with **Watson Knowledge Catalog (WKC)**, which we are not covering in this lab.

The **Actions** tab will keep track of various events associated with the file. We can come back to this tab after we perform some actions.

The **Refine** button will open **Data Refinery**, a lightweight ETL tool that can be used to build ETL flows.

If you wish, you can click **Refine** and review some of the built-in functions. The functions that you apply to data are “preview only”, i.e. they don’t modify the original file. Notice that the steps that you apply are captured in the right panel. Once you finish applying the actions, the steps are saved in the *Refinery* *flow* can be configured to run as a batch job.

We are not providing a deep dive of **Data Refinery** in this lab, but if you have questions, please ask your instructor.

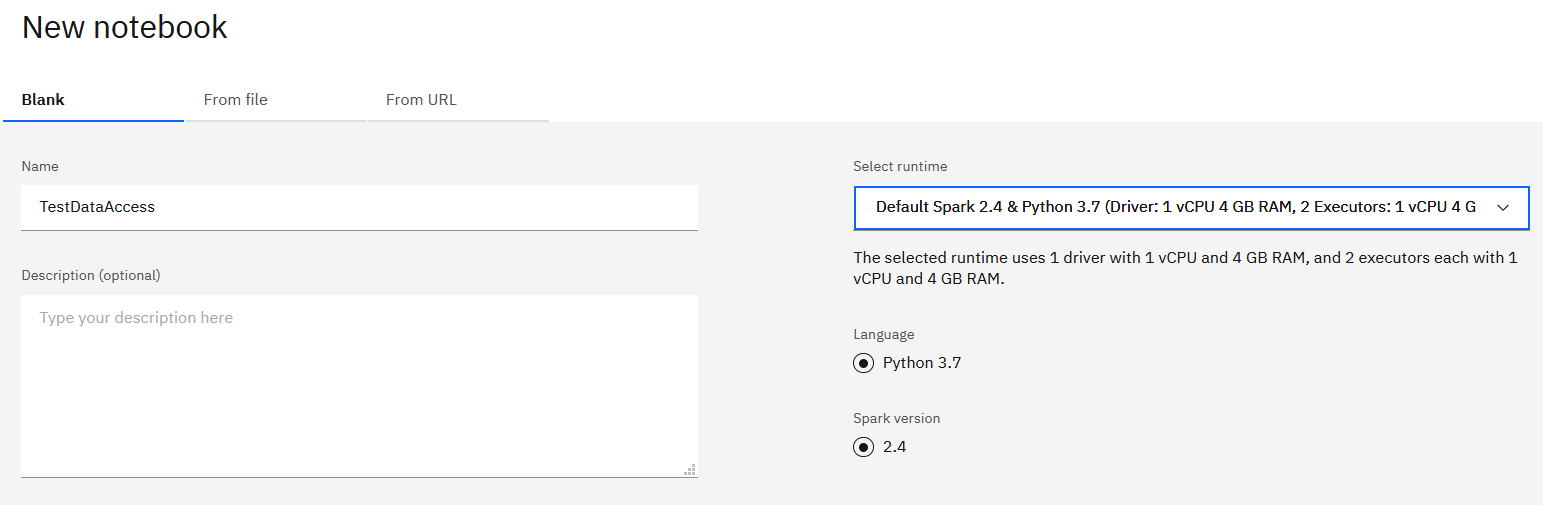
1. If you wish, repeat the same steps for database table (*CUSTOMER\_CHURN*).

Next, we will test code generation for reading data and writing output to the project.

1. Click **Add to Project -> Notebook**.

Give the notebook a name and select *Default Spark 2.4 and Python 3.7 environment*.

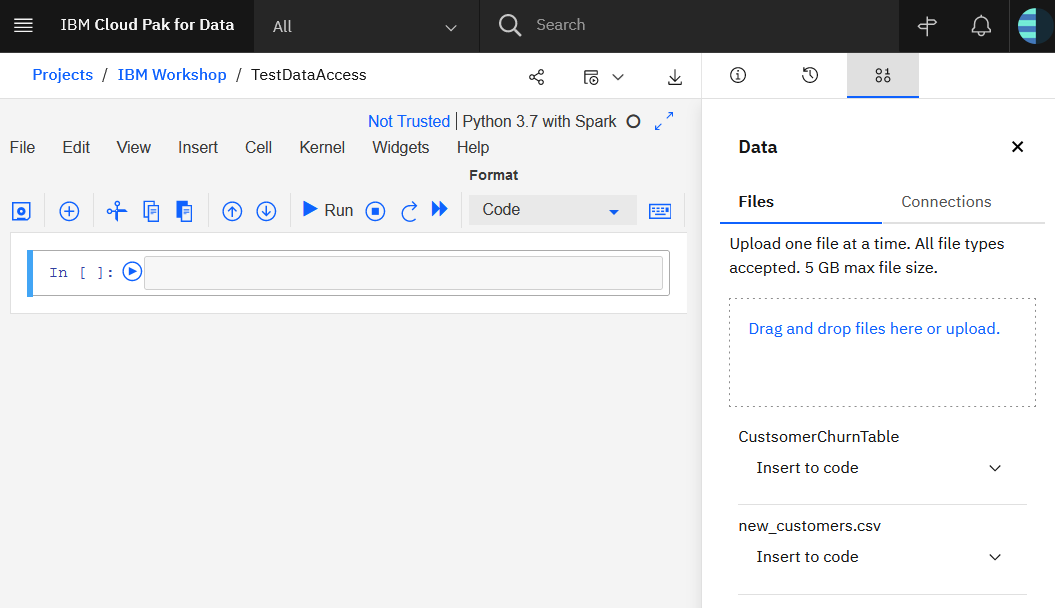
*Note: this environment will allow us to test both pandas and Spark data frame data import.*



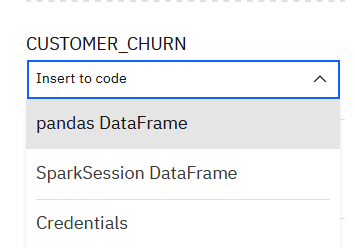
1. Click on the **Data Assets** icon in the top right corner.

Notice that code generation is available for all data assets that we have in the project.

Code generation is supported for data sources and types listed in documentation: <https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_3.5.0/wsj/analyze-data/data-load-support.html#file-types>

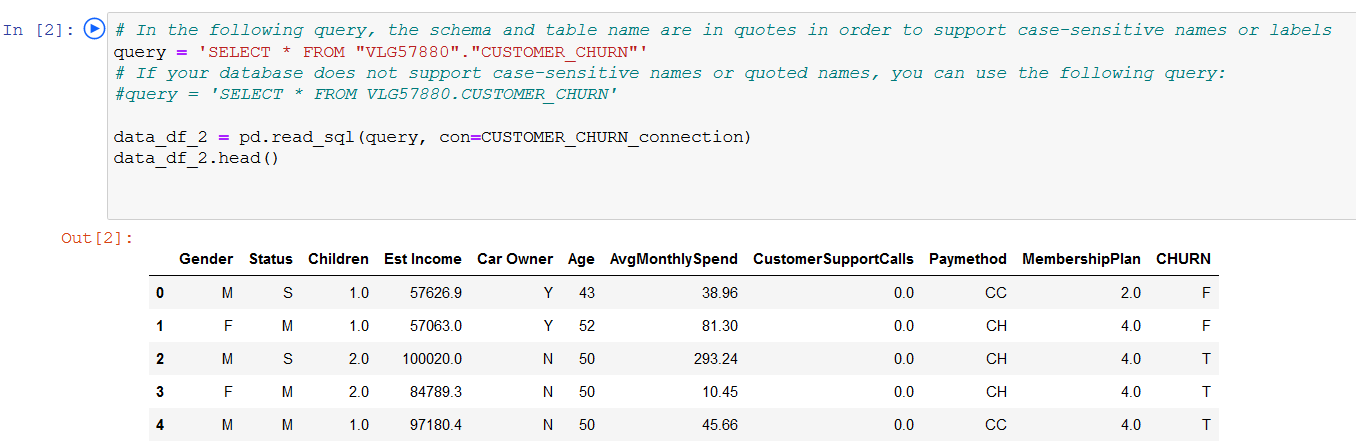


Click **Insert to Code** and test code generation for pandas and Spark for both *Customer Churn* table and the csv file.

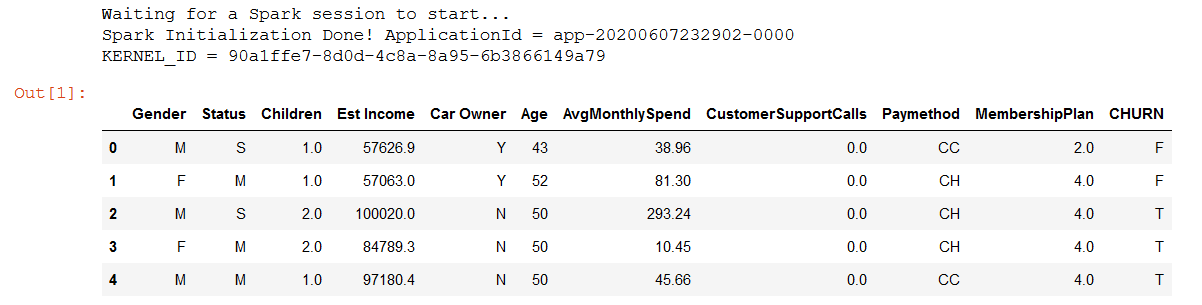


When testing, you should see output similar to the following screenshots.

**Pandas**



**Spark**



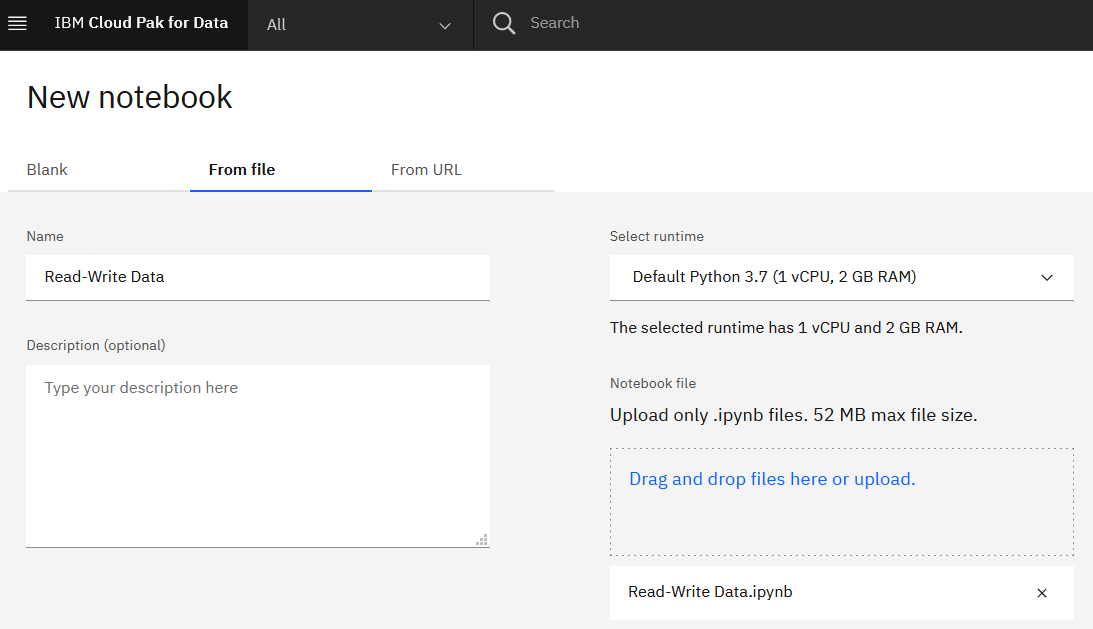
In some cases, you may need to write data (a csv file) to the project. Let’s review a notebook that provides sample code for this task.

### Write Data to a Project

1. Return to the project menu. Click **Add to Project -> Notebook**.
2. Click the **From File** tab. Navigate to *git repo/notebook* folder and select the *Read-Write Data* notebook.

Make sure *Default Python 3.7* is selected as the environment.

Click **Create notebook**.

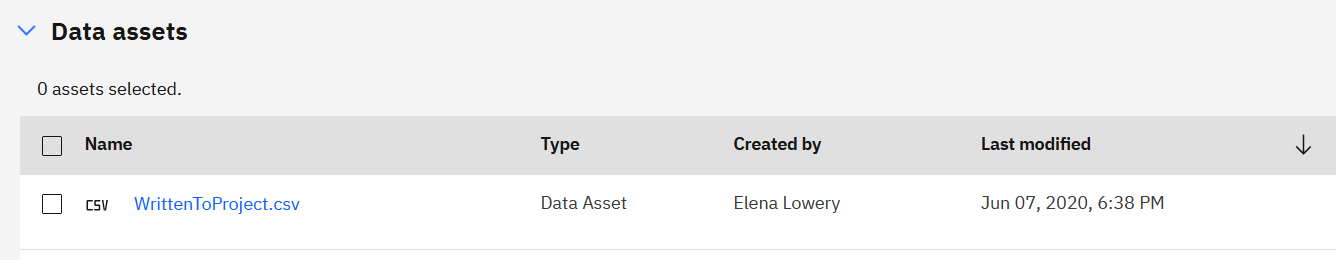


This notebook shows how to read and write files to **Data Assets** of the project. Once the file is written to **Data Assets**, we can perform several actions, for example:

* + Save the file in the Catalog
  + Upload the file to a Git repository
  + Download the file to local system.

Review the code in the notebook and run it. Let the instructor know if you have questions.

Notice that the written data asset is displayed in the project.



### Write Data to a Database

1. Return to the project menu. Click **Add to Project -> Notebook**.
2. Click the **From File** tab. Navigate to *git repo/notebook* folder and select the *Database\_Read\_Write* notebook.

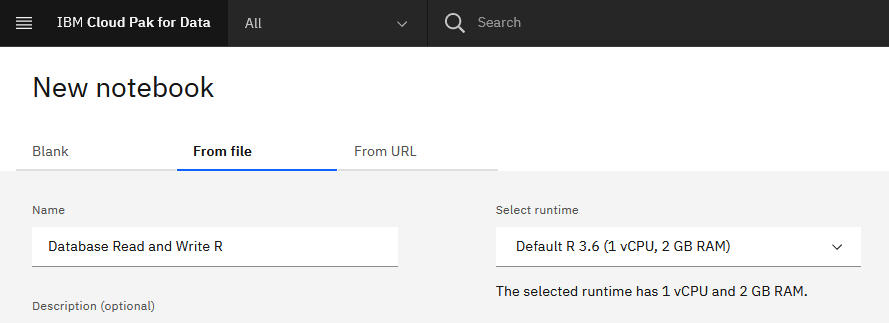
Make sure *Default Python 3.7* is selected as the environment.

Click **Create notebook**.

The notebook provides several examples of reading and writing data to a database. Run the notebook and let the instructor know if you have questions.

1. If you would like to review the R example, load and run the *Database\_Read\_Write\_R* notebook.

When loading the notebook, make sure to select R as the runtime environment.



# Part 2: Environment configuration

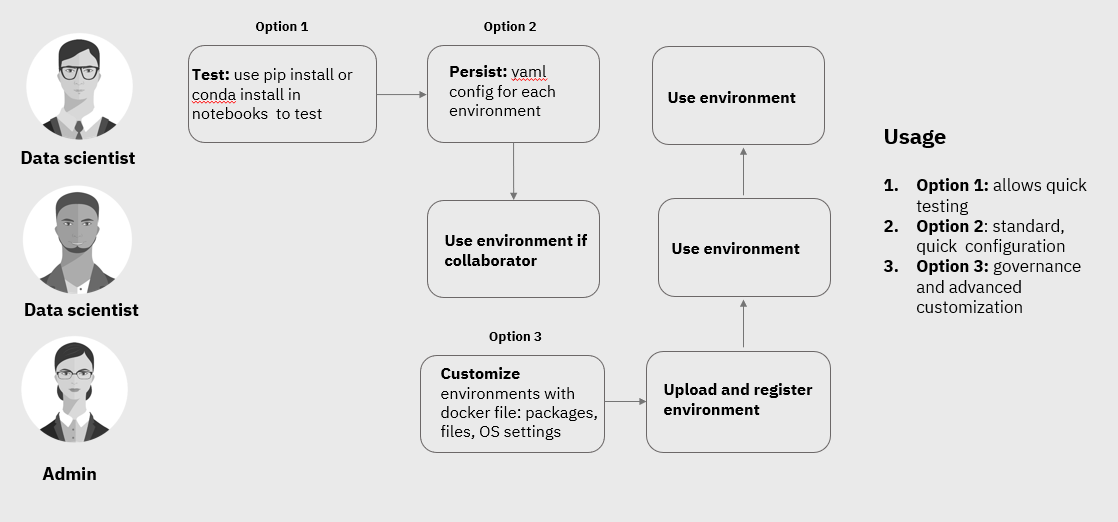
In this section you will learn how to configure environments for data science assets in CPD.

An *environment* is a runtime hardware and software configuration for data science assets (for example, a notebook or an RStudio application).

Environments can be customized by end users and by an administrator. When an environment is created by an administrator, it’s referred to as a *custom image*. This terminology is related to platform architecture, which is based on Kubernetes.

A *custom image* is registered at the platform level, which means that this environment will be available to all users of CPD. An environment customized by a data scientist can be used only by project collaborators, and its scope is limited to a project.

This diagram shows the sample workflow that can be used for environment customization.



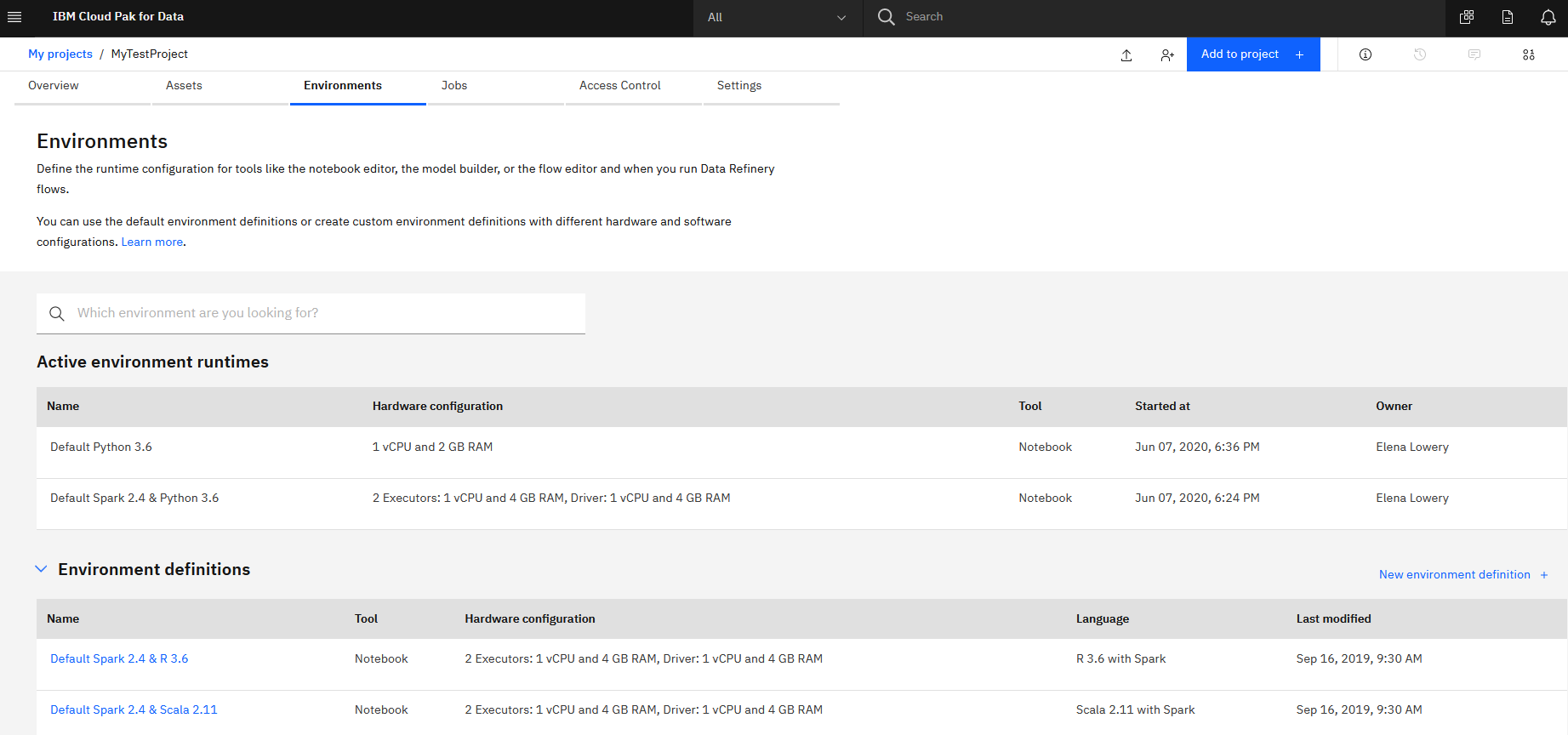
In this lab we will cover environment configuration by a data scientist. The instructions for custom image creation, which is done by an administrator, are available in product documentation.

First, let’s review the default environments and hardware configuration.

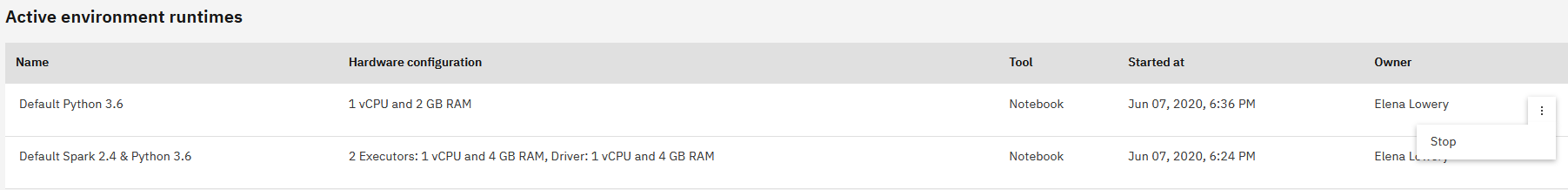
* Navigate to your project in CPD and click on the **Environments** tab.

In this view you can

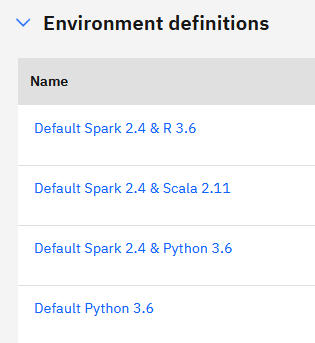
* + View which environments you started
  + Stop running environments
  + View environments that you can use
  + Drill down to environment definition
  + Create a new environment.



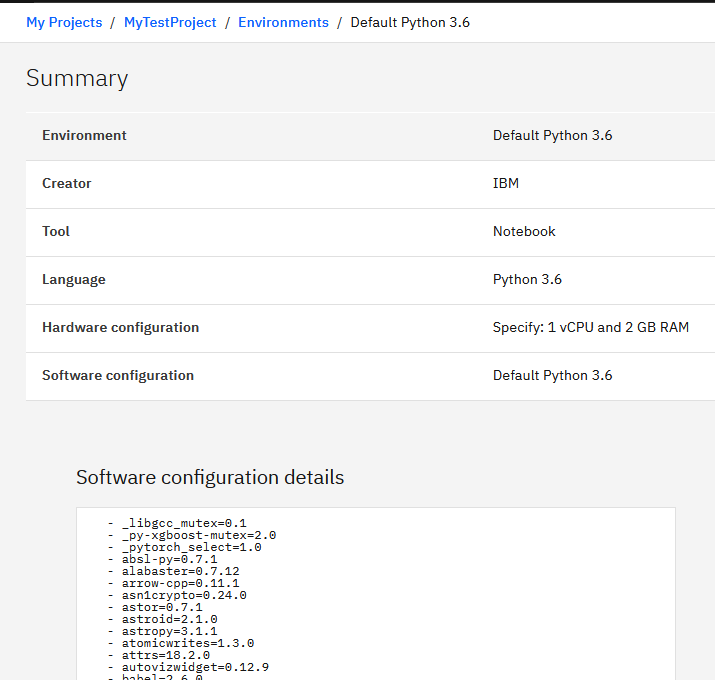
* If you wish, stop one of your environments. The stop menu is available for each environment.



* To view the environment configuration, click on the environment hyperlink in the **Environment Definitions** view.

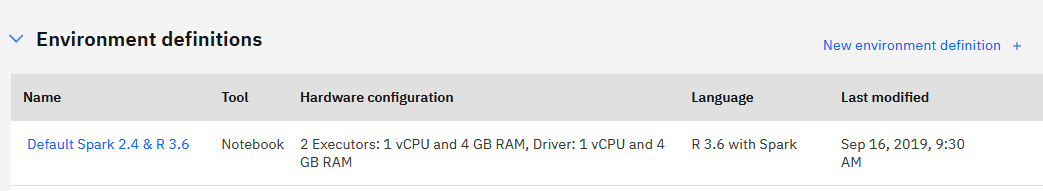


The environment detail page displays hardware and software configuration for the environment.

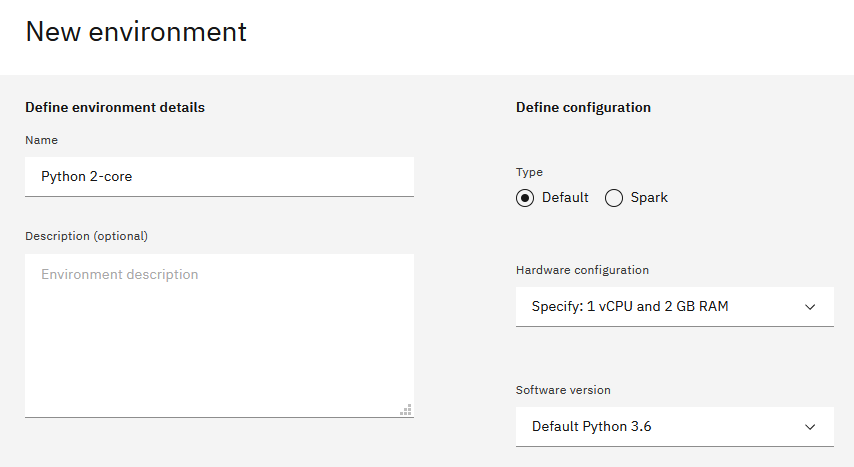


Return to the **Environments** tab.

* Click on **New environment definition**.



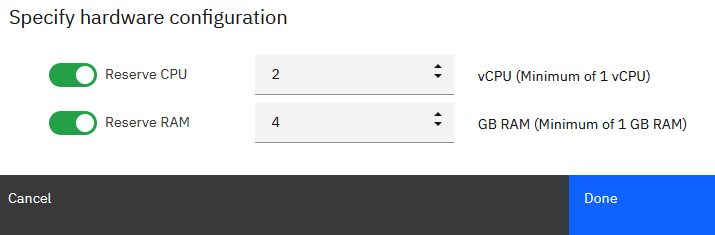
* Provide an environment name, for example, *Python 2-core*.



Notice that we have options for Spark and non-Spark environments. If you wish, click on the Spark radio button to review configurable options for Spark. While Spark configurations are pre-defined, we have flexible options for other environments.

*Note: If the cluster includes GPU systems, you will also see options for GPU environments.*

* Click on the **Hardware configuration** dropdown and select **Specify**.
* Enter *2 vCPU* and *4 GB RAM*, and click **Done**.

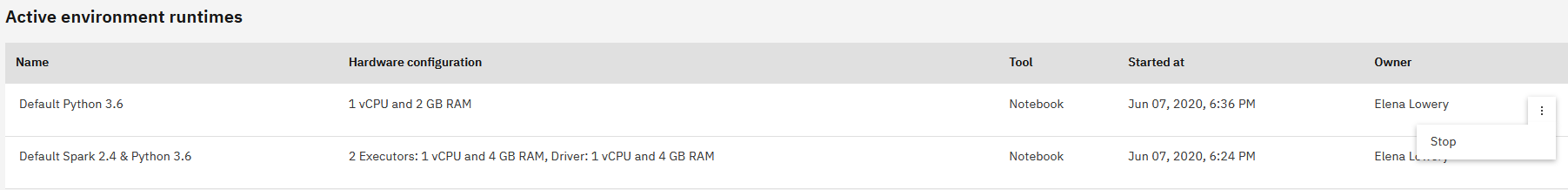


At this time the environment has not been provisioned yet, you created an environment definition. When you start this environment, it will be guaranteed 2 vCPU and 4 GB RAM However, in order to start this environment 2 vCPU and 4GB RAM must be free on any node of the cluster. If the requested resources are not available, a message will be displayed, and you will be asked to contact the administrator to free the resources.

The administrator can look at the cluster utilization and identify users/processes that are taking up the resources. As a user, you can review the environments that you have started by clicking on the **Environments** tab. You can stop each environment to free up resources.

Important points about environments:

* + Environments are started “per project per user”. If a user opens 3 notebooks in 3 different projects, 3 environments will be provisioned.
  + If a user opens 3 notebooks in the same project, and they use the same environment configuration (both hardware and packages), then 1 environment will be provisioned.
  + Notebook environments will automatically shut down after 18 hours of inactivity.

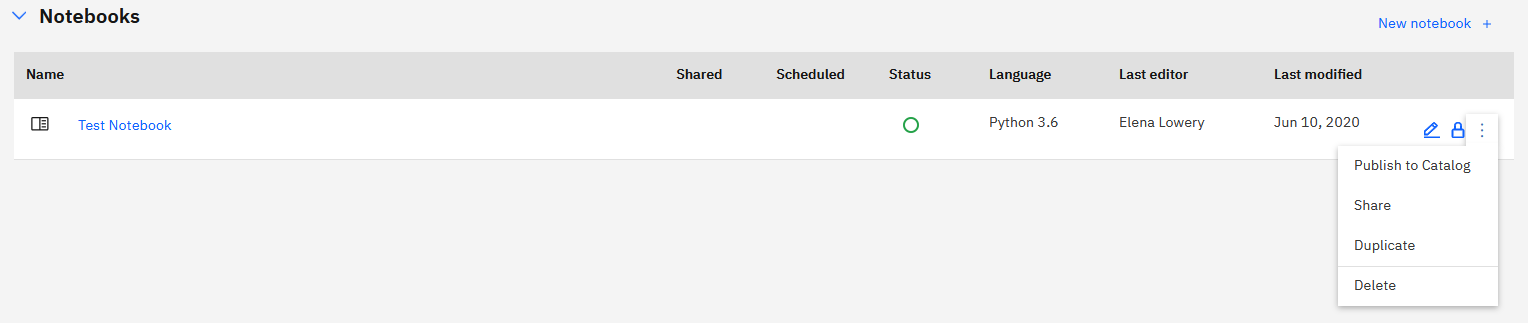


Some of the best practices when working with environments are:

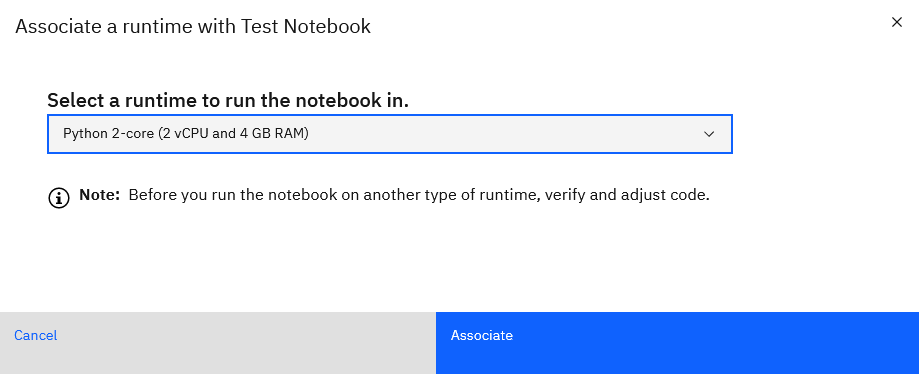
1. Do not request more resources than needed.
2. Do not request more resources than 75% of resources available on a single node of the cluster for Python and R environments (check with the administrator). For example, if a single node of the cluster is 16 vCPUs, do not define an environment that’s greater than 12 vCPUs.
3. Shut down the environments in projects after you’re done using them.

* Return to the project **Assets** view.
* Make sure the notebook is stopped.

When the notebook is running, you will see the green circle in the *Status* column and the **Change Environment** menu will not be available. You can stop the notebook in the **Environments** view.



* Click **Change Environment** and select the environment you created.



Next time you open the notebook in *Edit* mode, it will run in the 2-core vCPU. environment.

## Package management

While it’s possible to load packages using *!conda* and *!pip* commands in notebooks, installation of packages with these commands is not persisted. The packages will not be available to collaborators and even to the same user after environment restart.

We recommend that you use *!conda* and *!pip install* for testing, and after you verified that packages can be loaded, create an environment *customization*.

CPD supports two options for customization:

* *Option1:* customization on environment startup. A yaml file is used to specify channels and packages that should be loaded. The packages will be installed each time the environment starts, which will provide a consistent experience for each user. This option can be used by any user of the platform.
* *Option 2:* preload requirement libraries by creating a *custom image*. A custom image is usually created by an *administrator* who downloads the environment provided with CPD, updates it, and uploads it back to the platform.

Option 2 is often used when:

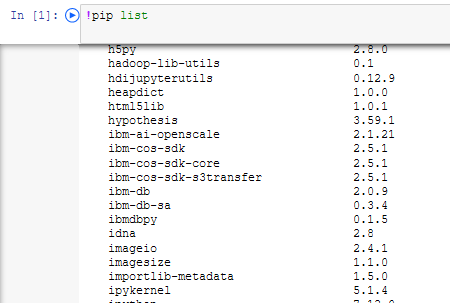
* There is a requirement to shorten startup time of the environment (i.e. loading libraries each time affects performance)
* In addition to Python and R libraries, system libraries must be loaded.

In this lab we will review *Option 1* – customization on environment startup.

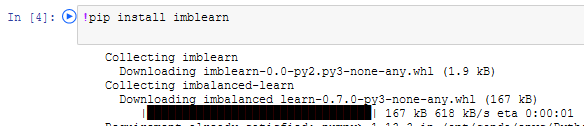
First, let’s test what happens if we use *pip* *install* in a notebook.

1. Create a notebook.
2. Run the *!pip* *list* command.

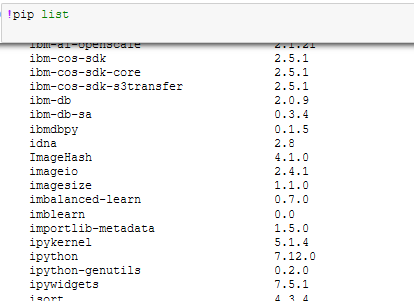
Notice that the *imblearn* library is not installed.



1. Install the *imblearn* library: *!pip install imblearn*



1. List the packages again to make sure that the library was installed.



1. Create another notebook and run *!pip list* command

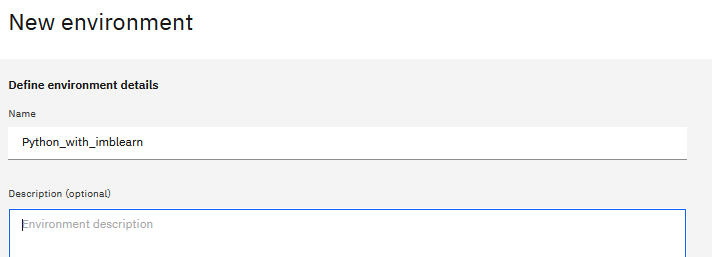
The *imblearn* library is installed because the 2nd notebook runs in the same environment.

1. Stop the environment in the **Environments** view.
2. Open one of the notebooks in *Edit* mode and run the *!pip list* command.

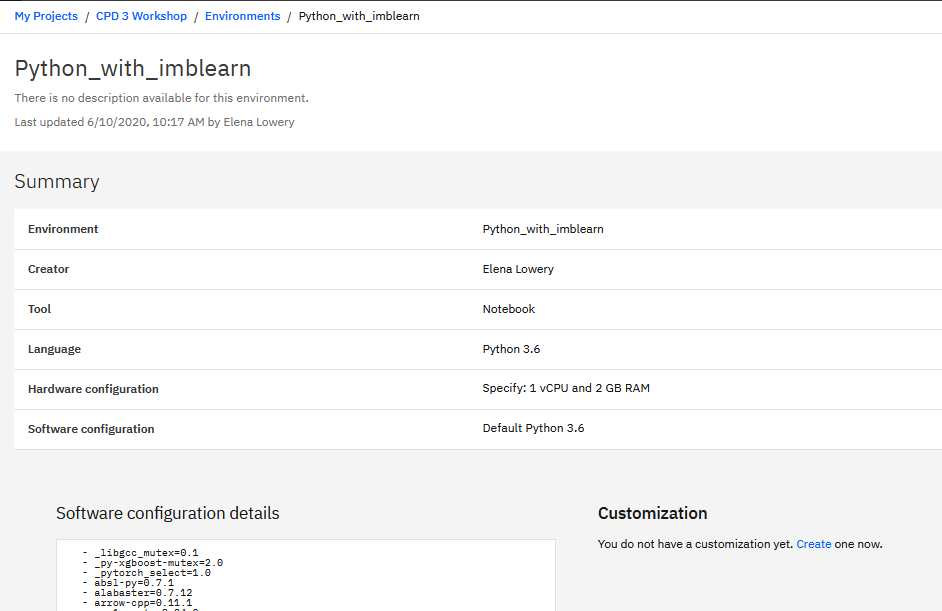
The *imblearn* library is not installed because the configuration that’s done with *pip* and *conda* commands is not persisted.

Next, we will explain how to customize the environments to load libraries on startup.

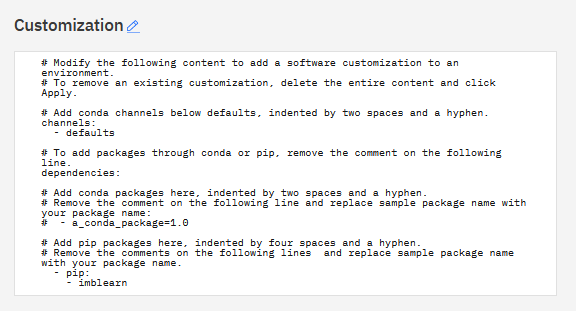
1. Return to the **Environments** tab and click **New environment definition**.
2. Provide the environment name and click **Create**.



1. Click **Create** under **Customization**.



1. Make the following changes to the yaml file:
   * Uncomment the *dependencies* line
   * Add *– imblearn* under pip
   * Make sure the indentation is exactly as described in the file. If indentation is not correct, you will get an error when the environment starts.



The customization screen is in the standard Conda *.yaml* You can find more details about customization details here: <https://www.ibm.com/docs/en/cloud-paks/cp-data/3.5.0?topic=environments-customization-options-conda-pip>

1. Associate the customized environment with one of your notebooks.
2. Open the notebook and verify that the library was installed.

You have finished learning about environment customization in CPD.

# Part 3: Collaboration and Git Integration

In this section you will learn about collaboration and Git integration.

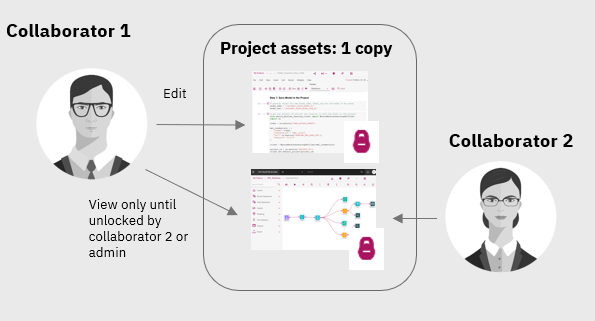
CPD supports three modes of collaboration:

* **Option 1:** Local collaboration (no Git)
* **Option 2**: Collaboration via Git for all assets with the exception of JupyterLab
* **Option 3:** JupyterLab collaboration with Git

Let’s review collaboration options in more detail.

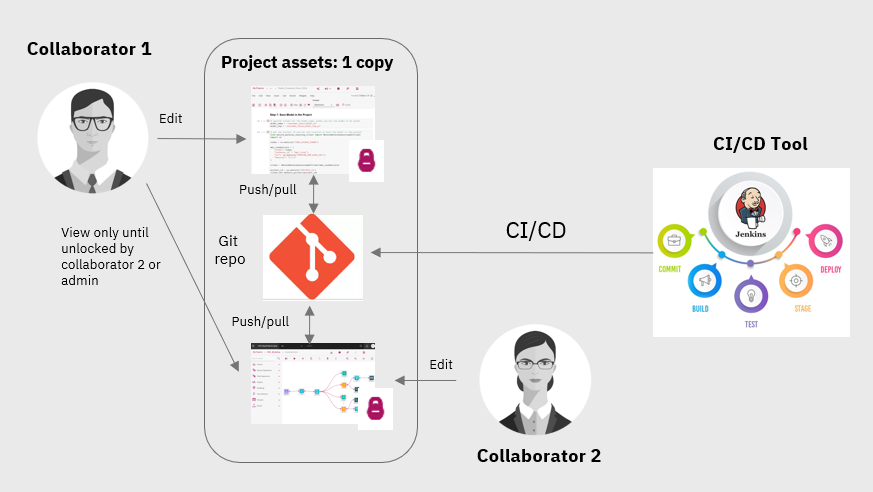
When we work in *local collaboration* mode, all collaborators work on one copy of assets in the project. When a user works on an asset, for example, a notebook, it becomes locked. Only the current user or the admin of the project can unlock the asset. Since only one version of the asset exists, changes are immediately available to all collaborators. Local collaboration mode is enabled when we create a project **without** a connection to Git.

**Figure 1: Option 1 – local collaboration**



The second option is *collaboration via Git for all assets with the exception of JupyterLab*. JupyterLab is an exception because it has its own Git integration that works differently than integration for other assets in the project. This option is enabled when we connect the project to a Git repo. Once the project is connected, we can use the **Pull** and the **Push and Pull** options to synchronize with the repo.

**Figure 2: Option 2 - Collaboration with Git**



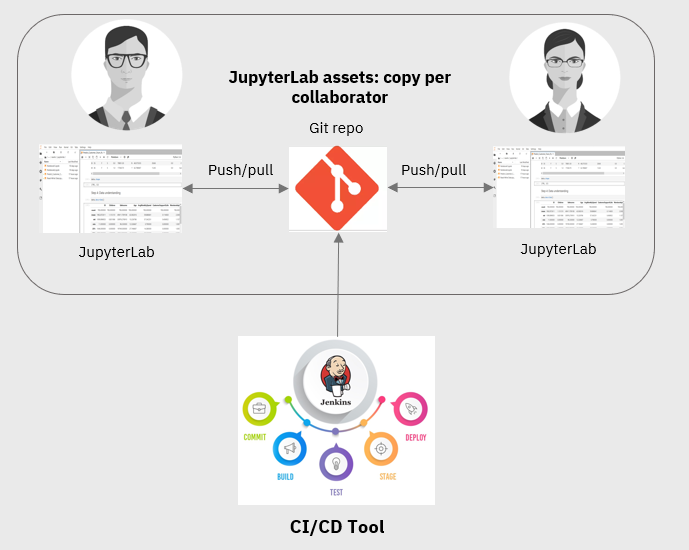
Since there is only one copy of the assets, *Pull and Push* and *Pull* options are not directly applicable to collaboration (because collaborators will see the changes immediately). However, these options are needed for 2 reasons:

* *Push* option will publish assets to a git repo. A git repo can be used by a CI/CD tool for automating deployment.
* *Pull* option is needed for getting assets from JupyterLab into the project (explained in the next option).

The third option is *collaboration with JupyterLab*. JupyterLab is an IDE that’s used for editing notebooks, and Git integration is provided by the JupyterLab Git extension.

While conceptually Git integration in JupyterLab is similar to Option 2, the UI and the steps are different.

**Figure 3: Option 3 - Collaboration in JupyterLab**



Unlike Jupyter Notebook environment, JupyterLab includes a file management component, which means that notebooks and data files can be stored in the JupyterLab IDE. We refer to these files as JupyterLab files. All other files in the project are called “project assets”.

Since a project may include not just JupyterLab assets, but also SPSS flows, data assets, RStudio files, and Data Refinery flows, there are two types of Git integration in a project with a JupyterLab – Option 2 for all project assets and Option 3 for JupyterLab files. Please note that only one notebook IDE – either Jupyter Notebook OR JupyterLab can be configured in one project.

While the details of Git integration may seem confusing, if the projects are organized by “asset type”, then the user doesn’t need to understand the complexity and the different options for integration.

Option 1, local collaboration, is a good option for any company that does not require Git as a version and collaboration management system. High availability is configured by default in a CPD cluster, and backup can be performed either manually or with scripts.

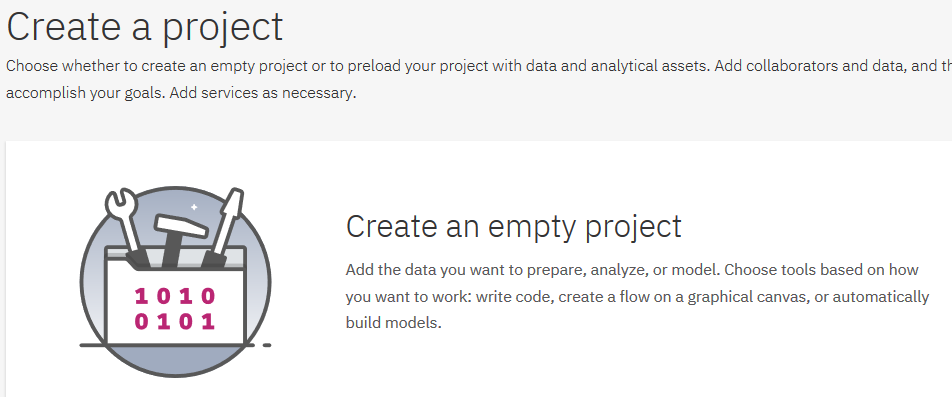
The customer can create a project just with JupyterLab IDE and keep all notebooks and data assets in JupyterLab for a consistent collaboration experience.

Finally, the “mixed asset” projects can be reserved for analytical assets that will eventually be deployed to production. These projects will require more involvement from an admin who can create best practices and a recommended workflow for working with Git.

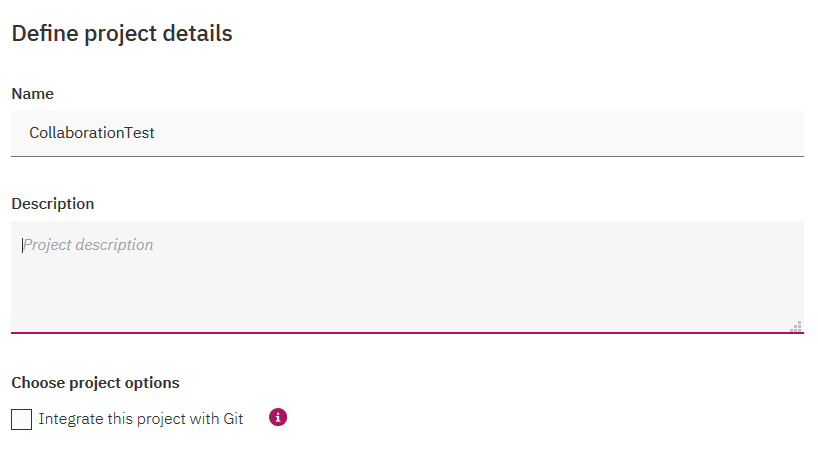
In this section of the lab we will review the three collaboration options. You will need to work with a colleague to test some aspects of collaboration. If you’re running this lab by yourself, create a 2nd userid.

## Option 1: Local collaboration

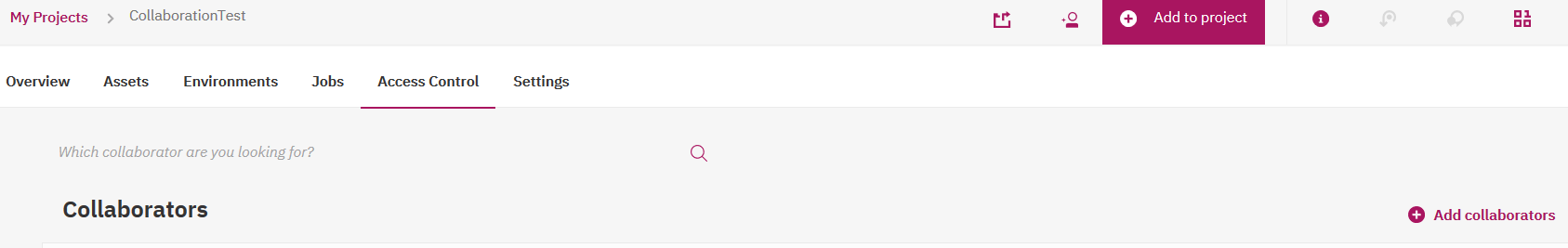
1. In CPD create a new *empty project*.



1. When providing project details, ***do not check*** the *Integrate this project with Git* option.

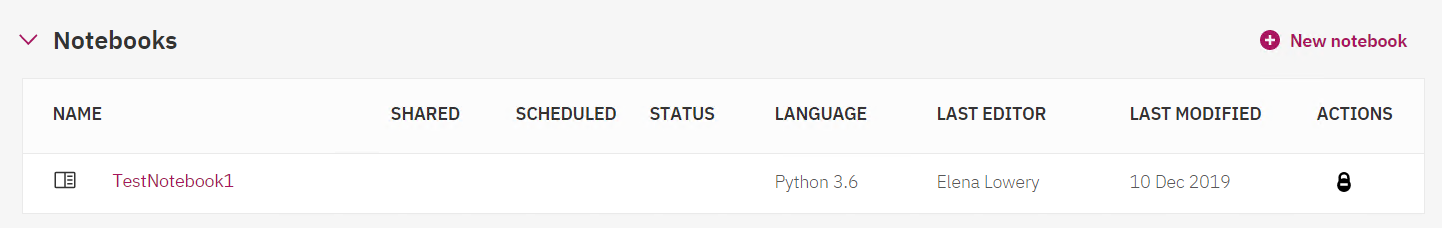


1. In the project click on the **Access Control** tab and add a collaborator – one of your colleagues or a 2nd userid. Give the user **Editor** role.



1. Create a notebook and add simple code, for example a markdown cell or a print statement.
2. Next, we’ll test the collaborator experience.
   * If you’re working with a colleague, take a look at the project from their browser.
   * If you’re using a second userid, log in to CPD with that userid in a different browser (i.e *Chrome* or *Firefox*), not a different tab in the same browser (browser may refresh and log in as 1 userid).

The collaborator will see the notebook in locked status. Notice that they can’t unlock it. After you unlock the notebook (click on the lock icon and select **Unlock**), ask the collaborator to save changes, close the notebook and unlock it.



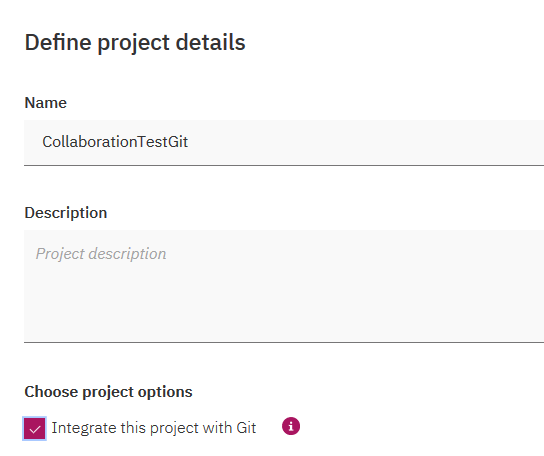
Open the notebook – you will see the changes made by the collaborator.

## Option 2: Git Collaboration for non-JupyterLab assets

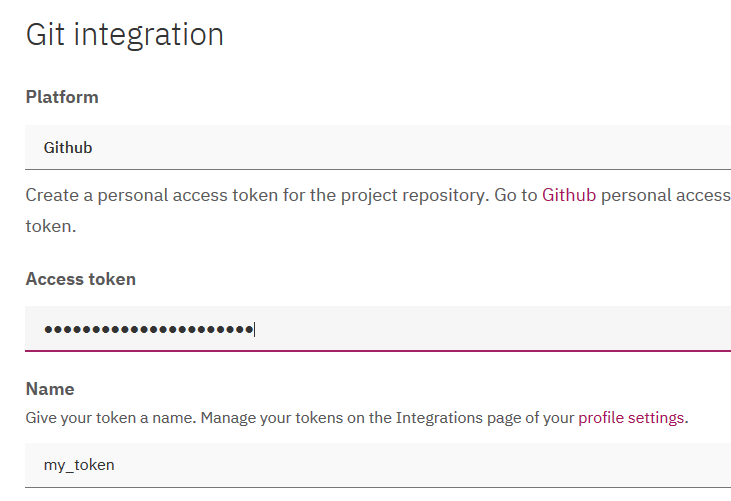
1. Create an empty repo in your Git application. Save the URL and the token.

*Note: If need instructions for completing these steps, see* ***Appendix A*** *and* ***B.***

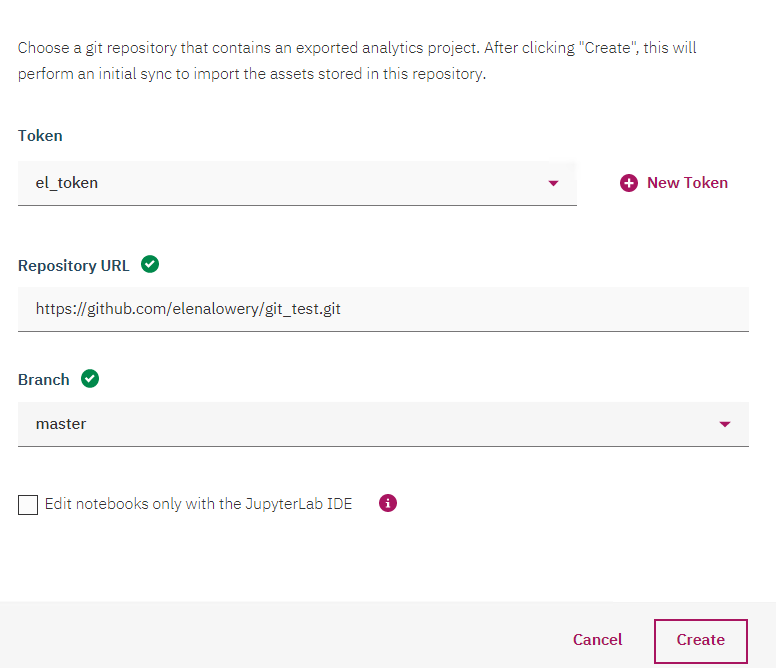
1. In CPD create a new *empty project*.
2. When providing project details, ***check*** the *Integrate this project with Git* option.



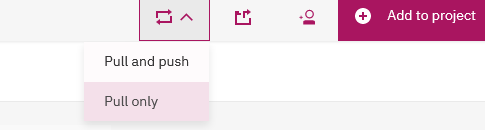
1. Click **New token**. Paste the token and give it a unique name.



1. Enter the *Repository URL* and select the *master* branch. **Do not check** the *Edit notebooks only with the JupyterLab IDE* checkbox.



1. In the project click on the **Access Control** tab and add a collaborator – one of your colleagues. Give the user **Editor** role.
2. Create 2 notebooks, *Notebook1* and *Notebook2* when logged in as your userid. Add simple code to both notebooks.
3. Ask your colleague to create *Notebook3* or a notebook with a unique name. Add simple code to this notebook.
   * Notice that all assets are visible to collaborators immediately
   * Only the user with *Admin* role on the project can push assets to the git repository (see the Git icon on the menu bar).



***Important Note:*** *While the Pull option is available, new assets will never be pulled from the Git repo for two reasons:*

* *The Git repo should never be modified from outside of Watson Studio. If the Git repo is modified either manually or through another tool, integration from Watson Studio will no longer work. The only exception to this rule is putting files into the JuptyerLab folder, which is explained in the next section.*
* *Changes are immediately available to all collaborators, so the collaborators don’t need to Push/Pull to the repo.*

## Option 3: Git Collaboration for JupyterLab assets

While JupyterLab is in a project, it has some differences compared to other assets that are a part of a Watson Studio project:

* *JupyterLab* maintains a separate file structure for each user. In other words, users **are not** working on one copy of a notebook when working in JupyterLab, which is the case for all other assets in Watson Studio.
* If you want to commit notebooks to Git, you need to place notebooks into a pre-created directory in *JupyterLab* (*project\_git\_repo/<repo\_name>/assets/jupyter\_lab*)
* Notebooks that are saved outside of that directory can’t be pushed to Git.
* Data assets can also be uploaded directly into Git. Uploading data assets into Git is not required (project data assets can still be used). If you want to commit data assets from JupyterLab to Git, you need to place them into a a pre-created directory in *JupyterLab* (*project\_data\_assets/data\_asset*).

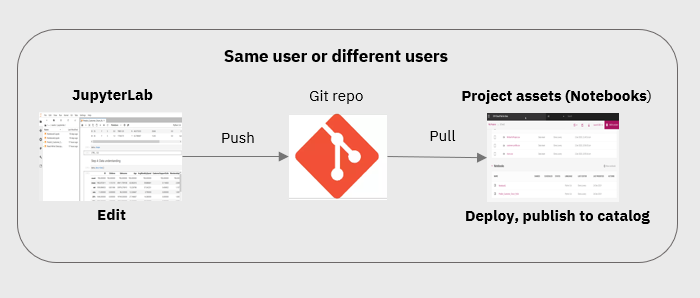
There are two types of Git integration in a project that has JupyterLab:

1. *JupyterLab* integration with Git via the JupyterLab Git extension
2. Project-level integration with Git.

The integration in *JupyterLab* is done with the Git extension (open source extension to JupyterLab). This integration is needed for collaboration. Collaborators won’t see notebooks unless they go through the Commit/Pull steps.

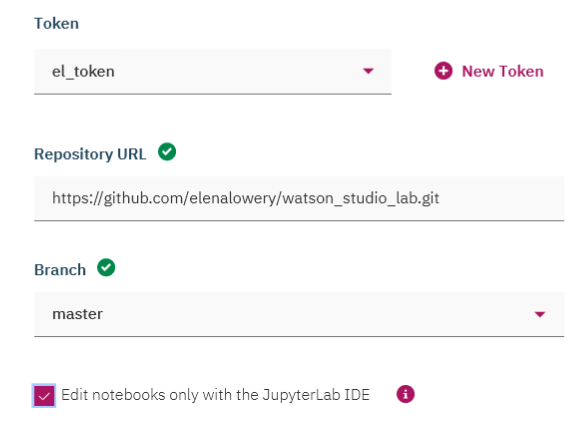
Project-level integration with Git is still needed for performing notebook-related tasks, such as scheduling a notebook run or publishing a notebook as a URL. In order to perform these tasks, the notebooks must become a “project assets” – they should be pushed to the Git repo from *JupyterLab* Git extension, and pulled from the same repo on the project level.

**Figure 4: Using Git to get assets from JupyterLab to Project.**



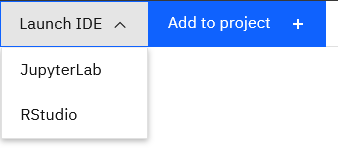
We will review these steps in the lab.

1. Create an empty repo in your Git application
   * Note the repo URL
   * You will be able to reuse the token that you created in the previous section.
2. In CPD create a new *empty project*.
3. When providing project details, ***check*** the *Integrate this project with Git* option.
4. Provide Git repo information.
5. Enter the *Repository URL* and select the *master* branch. **Check** the *Edit notebooks only with the JupyterLab IDE* checkbox.



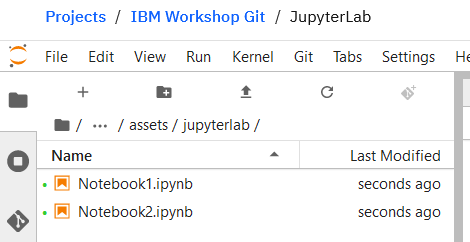
1. In the project click on the **Access Control** tab and add a collaborator – one of your colleagues. Give the user **Editor** role.

1. Notice that we now have an option to launch *JupyterLab* IDE. Launch the IDE.

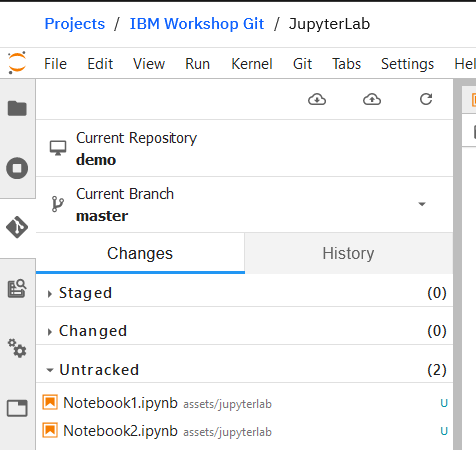


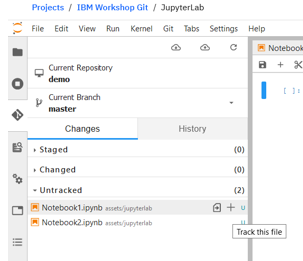
1. In *JupyterLab*, navigate to the *project\_git\_repo/<your\_repo>/assets/jupyterlab* directory and create 2 notebooks.

***Important note:*** *If you create notebooks in any other folder in JupyterLab, you will not be able to push them to the git repo.*

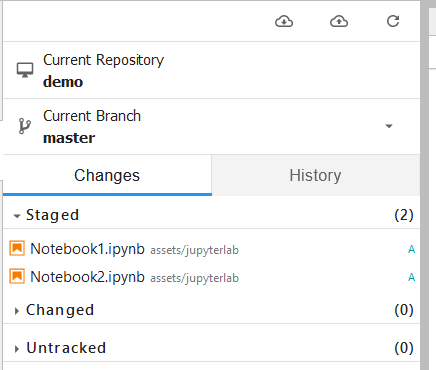


1. Ask your colleague to open the same project and launch *JupyterLab*. While they will see the git repo, they will not see any notebooks because you haven’t committed them yet.
2. In JupyterLab, click on the *Git* icon. Then click on the plus next to each notebook.



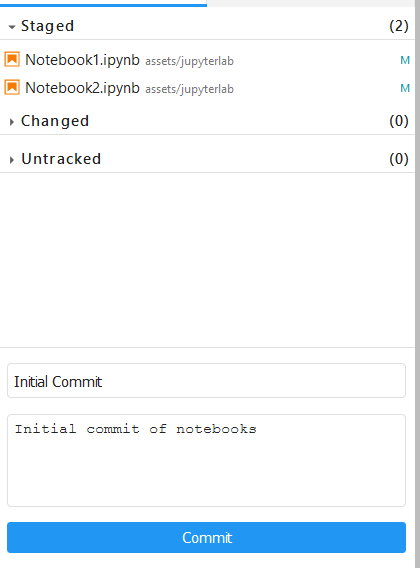


1. All assets are moved to the *Staged* section. You can remove the assets that you don’t want committed from *Staged* section by selecting them and then clicking the minus icon.

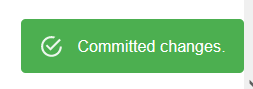


Let’s commit notebooks.

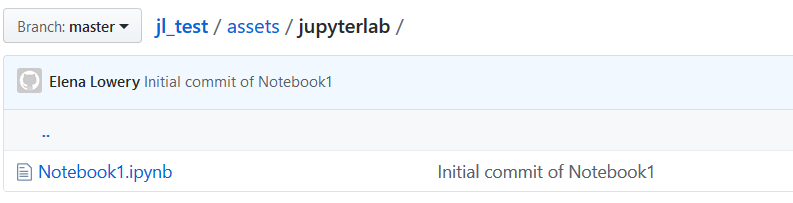
* Provide commit *Summary* message and click **Commit**.



* Click the *push* icon (cloud with an up arrow). Successful push notification is displayed in the right corner of JupyterLab.

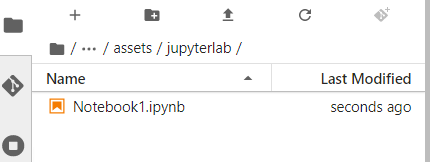


1. Check your git repo (in *Github*). You should now see the notebooks.



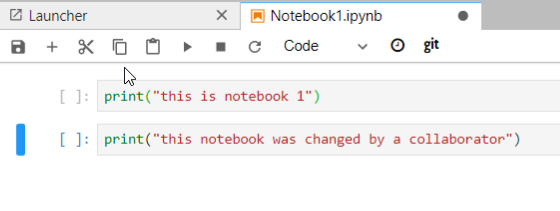
1. Ask your colleague to pull the changes into their project
   * Open the Git extension in the JupyterLab environment
   * Click the **Pull** icon (cloud with down arrow).

When they switch to the **File** view, they will see the notebook.

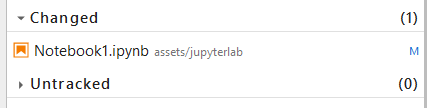


Ask your colleague to change the notebook.

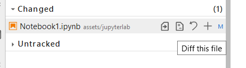
In our example, we added a 2nd cell with a *print* statement.

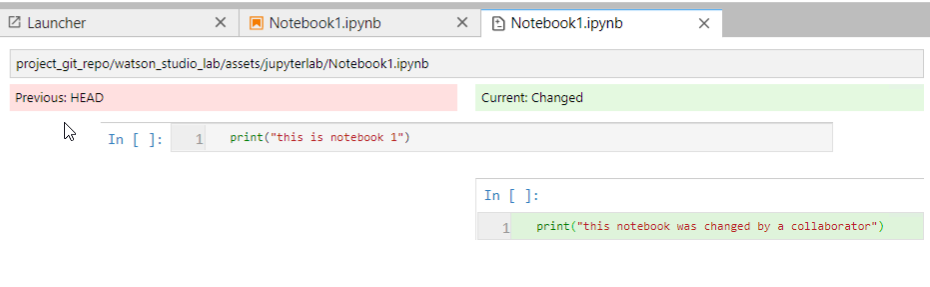


Ask your colleague to click on the Git icon. JupyterLab already determined that the notebook has changed.



The Git extension provides access to another extension, *nbdime*, that highlights differences between the versions of the notebook. To display this extension, highlight the notebook and click on the icon that appears when you hover over the highlighted notebook.

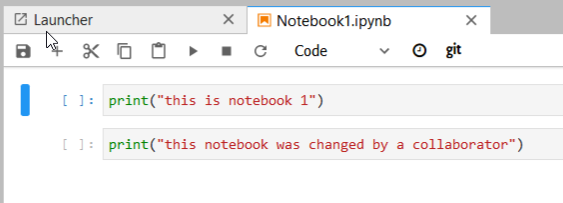




Ask your colleague to commit the change. They will need to follow the same process as we did in **Steps 10** and **11**.

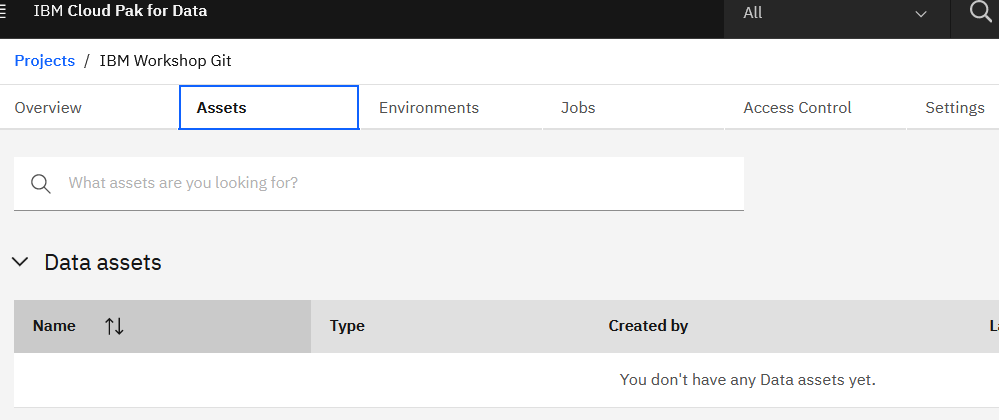
1. In your *JupyterLab* environment open Git extension and pull from Git. You should now see the changes that were made by your colleague.

*Note: If the notebook was open in your JupyterLab environment, you will have to close it and open it again.*

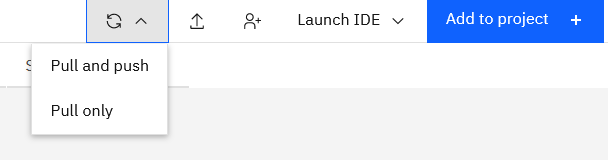


Next, we will pull the notebook into the project so that we can perform notebook functions such as publishing a notebook as a URL or configuring it to run as a batch job.

1. Navigate up to the **Project** view. Notice that there are no assets in the project (because notebooks are stored in *JupyterLab* IDE).

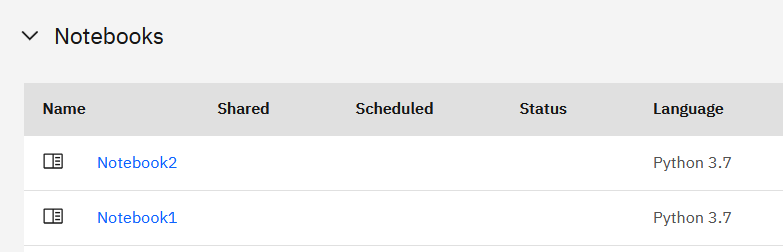


1. Click on the **Git** icon, then select **Pull only**.



1. Click **Sync**, then **View Project Assets**.

The notebooks are now displayed under project assets.



Notice that you can’t edit this notebook. This is by design – a project is always configured with a *JupyterLab IDE* OR *Jupyter Notebook IDE* (not both). If you want to make changes to this notebook, you will need to do it in *JupyterLab*, commit changes to the Git repo, and pull changes into the project.

Click on the notebook and notice that several action items are now available. We will test these options in the **Deployment** section of the workshop.

# Part 4: Importing Notebooks into JupyterLab

In this section you will learn how to import existing notebooks into JupyterLab.

## Importing notebooks

You may have noticed that JupyterLab does not provide *Create from existing* option for notebooks. We can get existing notebooks into JupyterLab by importing them into the git repo.

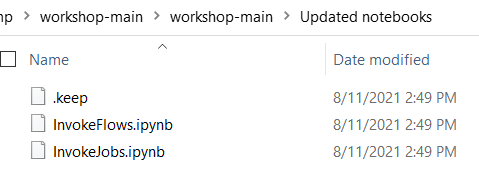
Notebooks and data files have to be placed in specific folders in the git repo. The easiest way to make sure that the directory structure is correct is to do a *push* from JupyterLab.

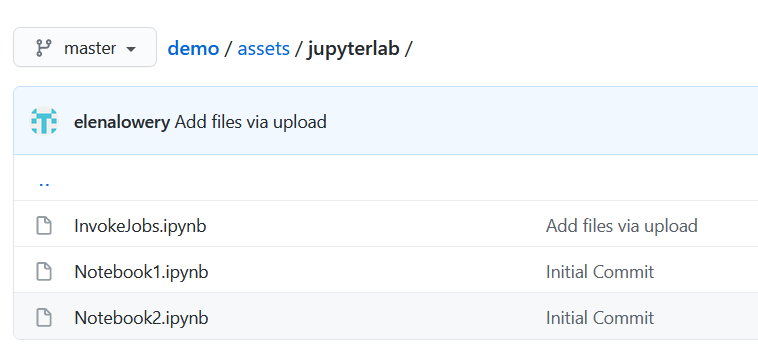
We already completed some of these steps in **Part 3**, let’s review them:

1. Create an empty git repo.
2. Associate a git repo with a Watson Studio project.
3. Create a simple notebook.
4. Commit changes to the repo - this will create the folder structure.

Next, we’ll complete the steps to import a notebook into JupyterLab.

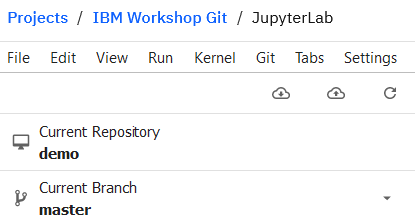
1. Open the git repo which you connected to your JupyterLab project.
2. Import the notebook *InvokeJobs from the git repo/Updated notebooks* folder (downloaded for this lab) into the *<your\_git\_repo>/assets/jupyterlab* folder in Git*.*



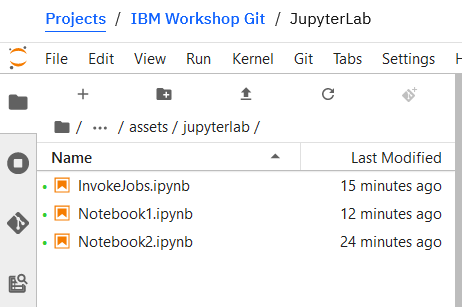


***Important note****: notebooks must be imported into this folder. If they are imported into a different folder, they will not be available in JupyterLab. Importing assets into other folders will also break git integration on the project level (as explained in Part 2).*

1. In CPD open your project, then open JupyterLab.
2. Open the git extension and click on the **Pull** icon.



1. After synchronization is done, click on the **File** icon. The notebook is now available in JupyterLab.

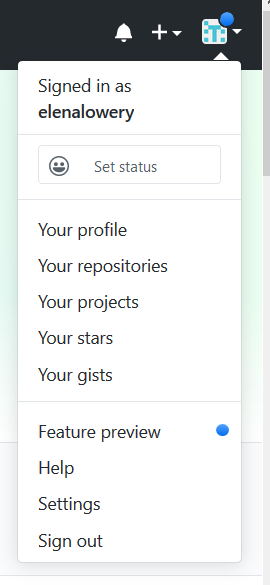


**You have completed the JupyterLab section of the lab.**

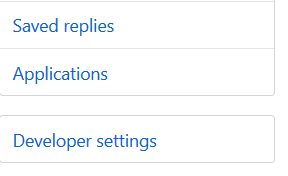
# Appendix A: Getting a Git Repo Token

*Note: The following instructions are specific to getting a token from a github.com account.*

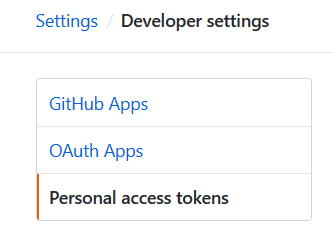
1. If you don’t already have a git account, set it up on *github.com* or another git site.
2. Create a git repo. If you need instructions – see **Appendix B.**
3. Click on the **Profile** icon (top right corner), then click on **Settings**.



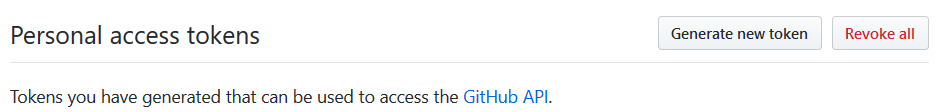
1. Click on **Developer Settings** (in the bottom of the left-side menu).



1. Click on **Personal access tokens**.

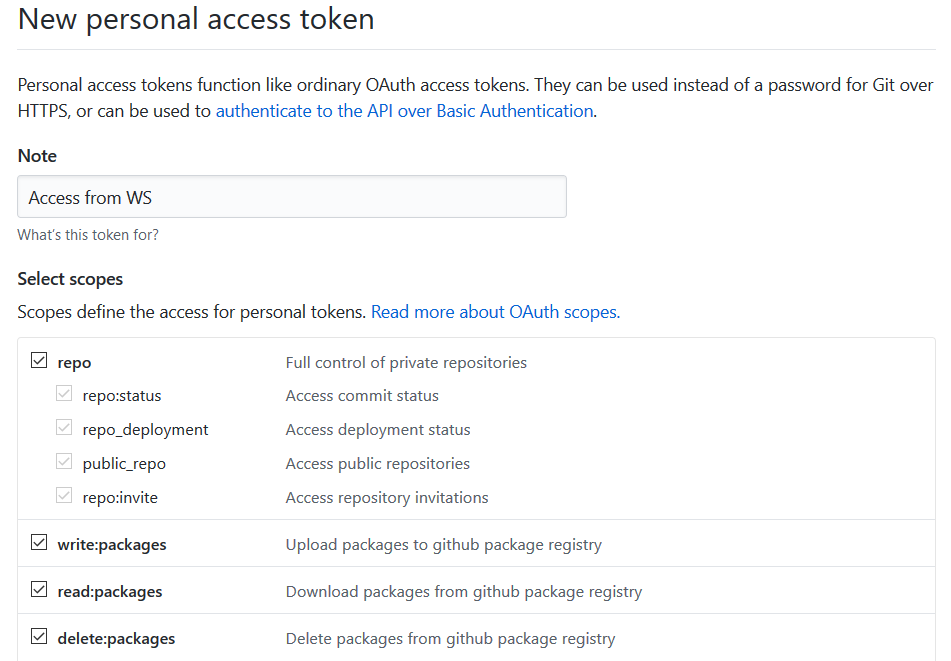


1. Click **Generate new token.**

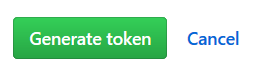


1. Provide a token name, for example, *Access from WS*.

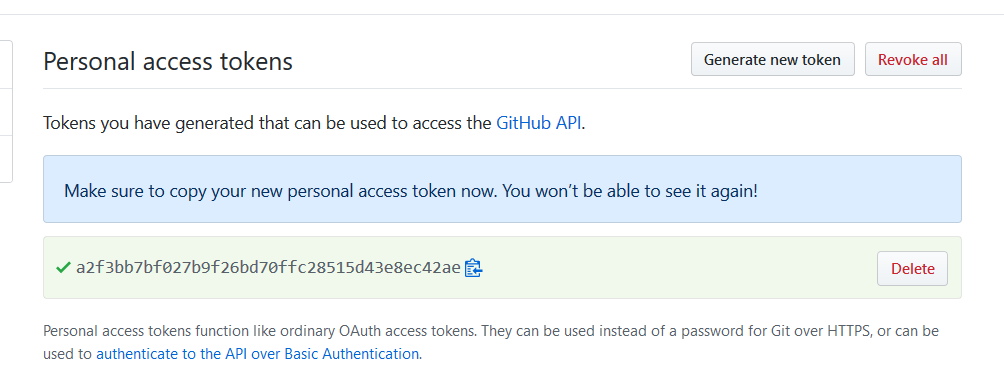
Check the *scopes* checkboxes – as a minimum, select the scopes listed in the screenshot.



1. Click Generate token.



1. Save the token in a notepad. You will use this value in Watson Studio.

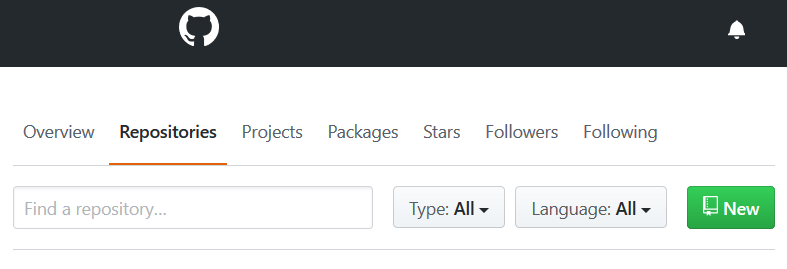


# Appendix B: Setting up a Git repo

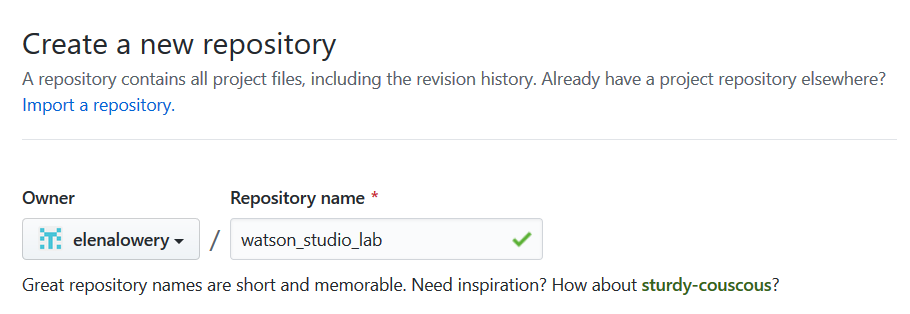
1. If you don’t already have a git account, set it up on *github.com* or another git site.

*Note: The following instructions are specific to getting a token from a github.com account.*

1. Log in to your git account.
2. Click on the **Repositories** tab (top right corner), then click **New**.



1. Provide *Repository name*. You can keep the default settings or you can change the security setting to *Private*.



1. Note the *https URL* – you will use it in Watson Studio.

