Governed MLOps Workshop

Continuous Integration (CI) / Continuous Delivery (CD) of AI Models

Document version: June 2023

DISCLAIMER

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."
Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

OpenShift is a trademark of Red Hat, Inc.

UNIX is a registered trademark of The Open Group in the United States and other countries.

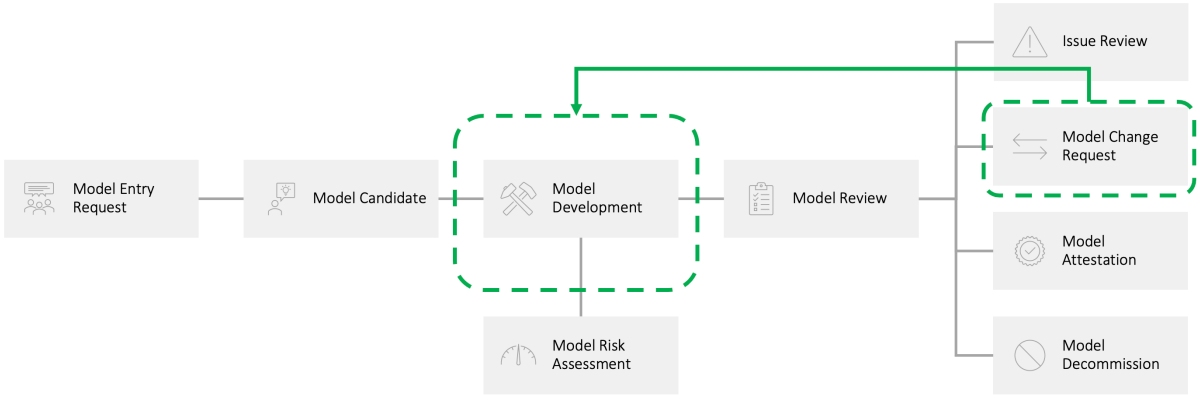
Table of Contents

Introduction

As organizations scale adoption of AI models in production, it becomes more important to automate the process for testing, validating, and promoting such models from dev (development) to uat (user acceptance testing, also known as pre-prod, quality assurance or staging) to prd (production) environments.

To enable such automation, it is important to be able to validate and monitor the performance of AI models (fairness, quality, drift, explainability) as well as automate the process of propagating the models and associated assets from one environment to another. In the previous modules, you have witnessed the birth of a model from an organizational perspective and you have learned how Watson OpenScale can be leveraged to monitor AI models and feed KPIs into the model governance workflow.

In this part of the workshop we augment scenario with a typical scenario for doing the initial model development and also the technical process of handling a model change request.



- During model development, data scientists explore multiple algorithms and techniques to train best performing AI model.
- Once satisfied with performance results, data science lead deploys the best performing model to a UAT deployment space.
- MLOps team configures Watson OpenScale to monitor and run validation tests against the model deployed in UAT space.
- A step in the workflow subsequently triggers a review of the model, which should lead to approval.
- Once model validation is approved, MLOps team propagates model from UAT to Prod.
- MLOps team configures Watson OpenScale to monitor the production model for fairness, quality and drift.

In this workshop, we cover two common approaches for implementing a governed MLOps methodology to enable the automation of propagating models from development through user acceptance testing (UAT) to production:

1. Propagating trained models from one environment to another.
2. Git based automation where data assets and source code are checked into a git repository, git integration is leveraged for code management, and automation is leveraged for testing and validation in UAT (PreProd) and production environments.

Choosing which of these approaches to implement is mostly dictated by the use case and the preferred method of governance that an organization chooses to adopt. In this workshop, we highlight both approaches and how they can be implemented using Cloud Pak for Data.

Before diving into the details of these approaches, it is helpful to quickly review the overall data science process and various tasks/activities performed by the data science team.

Following the approval of a model candidate, the data science team engages with business stakeholder to discuss the business problem to be addressed. After understanding and scoping the business problem, data scientists search and find data assets in the enterprise catalog that may be relevant and useful for training AI models to address the identified business problem.

Data scientists experiment with various data visualizations and summarizations to get a sound understanding of available data. This is because real-world data is often noisy, incomplete, and may contain wrong or missing values. Using such data as-is can lead to poor models and wrong predictions.

Once relevant data sets are identified, data scientists commonly apply “Feature Engineering” which is the task of defining and deriving new features from existing data features to train better-performing AI models. The feature engineering step includes aggregation and transformation of raw variables to create the features used in the analysis. Features in the original data may not have sufficient predictive influence and by deriving new features, data scientists train AI models that deliver better performance.

Afterwards, data scientists train machine learning models using the cleansed data prepared in the previous steps. Data scientists train several machine learning models, evaluate them using a holdout data set (data not used at training time) and select the best model or multiple models (ensemble) to be deployed in the next phase. Model building usually also includes a hyperparameter optimization step, which aims at selecting the best set of model hyperparameters (i.e. parameters of the model itself), which are set before training starts to further increase model performance

After data scientists build (train) an AI model that meets their performance criteria, they make that model available for other collaborators, such as software engineers, other data scientists, and business analysts, to validate (or quality test) the model before it gets deployed to production.

Once a model has gone through the iterations of development, build, and test, the Machine Learning Operations (or MLOps) team deploys the model into production. Deployment is the process of configuring an analytic asset for integration with other applications or access by business users, to serve production workload at scale. Two most popular types of deployment are:

- Online: a real time request/response deployment option. When this deployment option is used, models or functions are invoked with a REST API. A single row or multiple rows of data can be passed in with the REST request.
- Batch: a deployment option that reads and writes from/to a static data source. A batch deployment can be invoked with a REST API.

Keep in mind that in a hybrid multi-cloud world, development (dev), user-acceptance testing (uat), and production (prd) environment can be on-prem or in one of different cloud platforms. For example, the development environment can be hosted in a cloud platform but the production environment can be on-prem. Alternatively, the user acceptance testing environment can be on-prem while the production environment can be hosted on a public cloud platform.

For some background on the model development and deployment in the AI Lifecycle Management process, please review the white paper or check the following blogs:

- AI Model Lifecycle Management: Overview
- AI Model Lifecycle Management: Build
- AI Model Lifecycle Management: Deploy

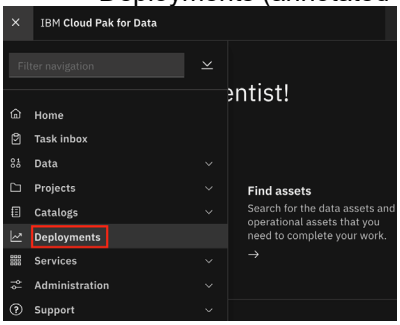
Additionally, please review the **Jenkins + IBM Cloud Pak for Data == Production-ready Delivery Pipelines for AI** blog to learn how to leverage cpdctl and Jenkins to design pipelines to enable automation in propagating AI models to production.

Method 1 – AI Model Propagation across Environments

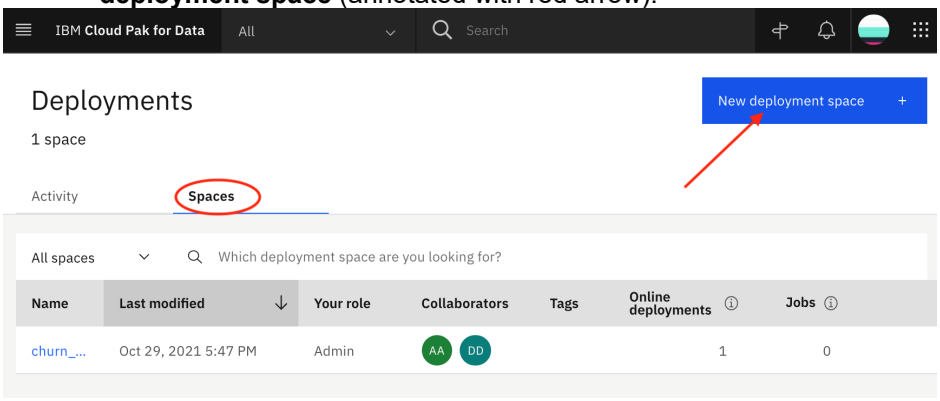
In this section, we illustrate the first approach leveraging cpdctl, a Cloud Pak for Data CLI (command line interface) tool, to automate the process of propagating trained models from one environment to another.

In practice, the environments can exist in the same Cloud Pak for Data cluster or in completely different Cloud Pak for Data clusters hosted on different cloud platforms. For this lab, we will use the same Cloud Pak for Data cluster and illustrate how to use the **cpdctl** tool to propagate assets from the quality assurance (QA, also known as user acceptance testing) deployment space to the production deployment space. The process is identical to how you'd propagate models from one cluster to another as the cpdctl tool is designed to handle the hybrid multi-cloud seamlessly.

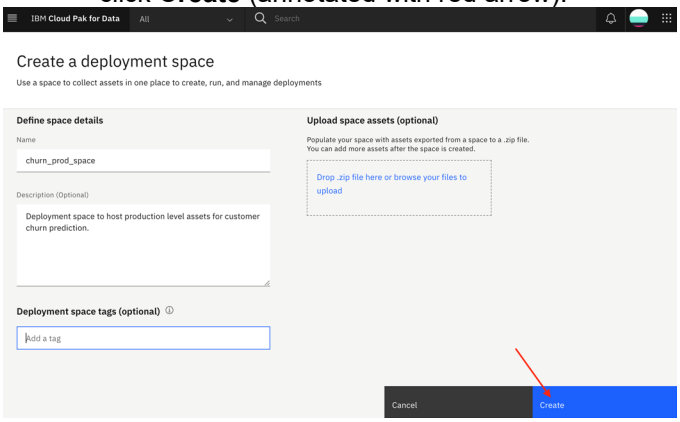
1. Log into Cloud Pak for Data as the **dslead** (lead data scientist) user.
2. Navigate to Deployments by clicking the Navigation menu (top left hamburger icon) and selecting Deployments (annotated with red rectangle).



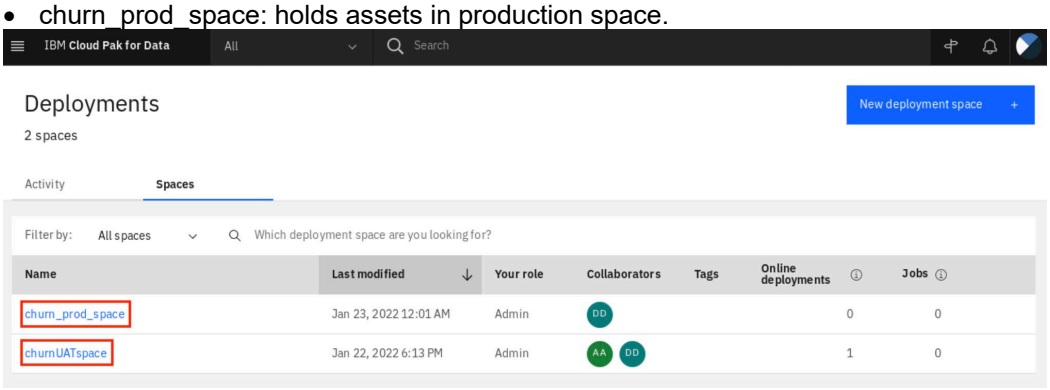
3. On the Deployments page, click on the **Spaces** tab (annotated with red oval) and click **New deployment space** (annotated with red arrow).



4. Provide a Name <churn_prod_space> and a Description (optional) for the deployment space and click **Create** (annotated with red arrow).

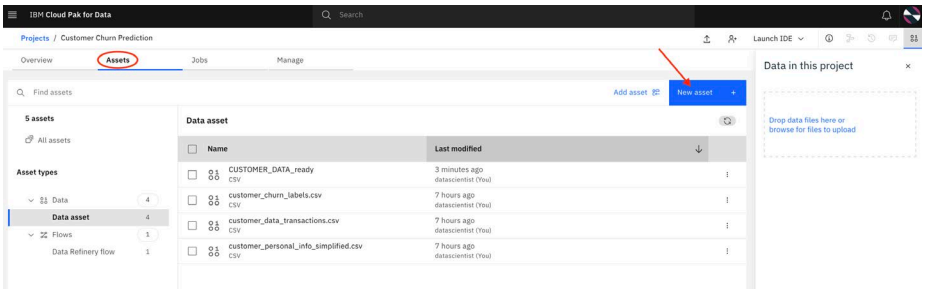


5. Validate you have two deployment spaces:
   - churnUATspace: holds assets in quality assurance (or UAT) space.
   - churn_prod_space: holds assets in production space.

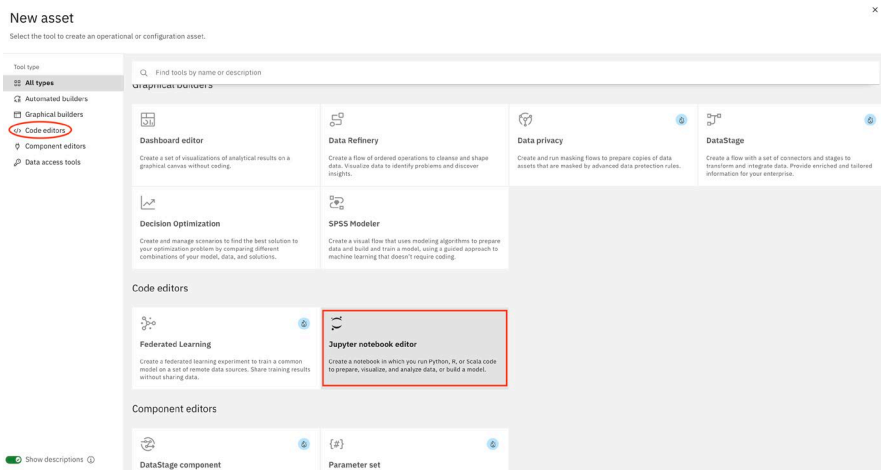


Next, you will run a notebook to leverage cpdctl to copy assets from churnUATspace to churn_prod_space.

6. Navigate back to your Customer Churn Prediction project by clicking the Navigation menu (top left hamburger icon), selecting **Projects ➔ All projects**, and then clicking on your Customer Churn Prediction project.
7. Click **Assets** tab (annotated with red oval), and click **New asset** + (annotated with red arrow).

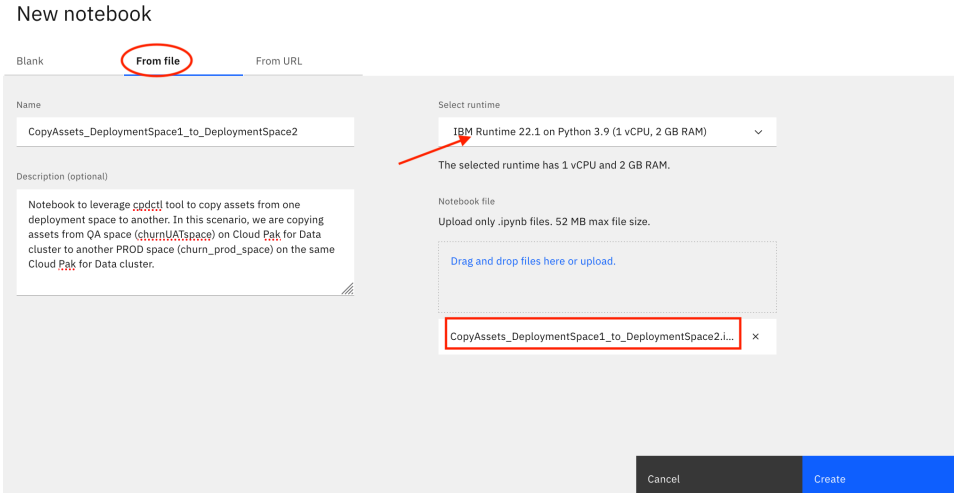


8. Scroll down and select the **Jupyter notebook editor** (annotated with red rectangle). Note that you can filter asset types by selecting the **Code editors** (annotated with red oval) to quickly find Jupyter notebook editor.



9. On the New notebook page, click **From file** tab (annotated with red oval) and click the **Drag and drop files here or upload** (

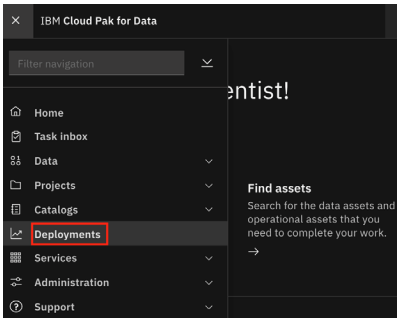
10. On the New notebook page, click **From file** tab (annotated with red oval) and then browse to find the [CopyAssets_DeploymentSpace1_to_DeploymentSpace2.ipynb] notebook file. Add a Description (optional) and click **Create**.
Verify the selected runtime is IBM Runtime 22.1 on Python 3.9 (annotated with red arrow).



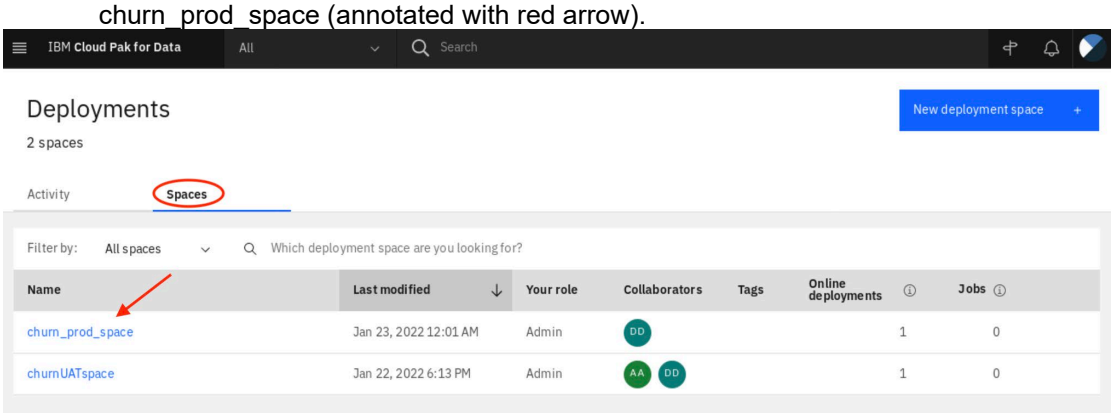
11. When the notebook loads in edit mode, execute the cells of the notebook step by step. Please read the instructions carefully as you execute the steps. You will need to modify the specific details about the source and target deployment spaces and model names.
    - **SOURCE_DEPLOYMENT_SPACE_NAME: churnUATspace**
    - **TARGET_DEPLOYMENT_SPACE_NAME: churn_prod_space**
    - **SOURCE_MODEL_NAME: 'Churn Model'**
    - **TARGET_DEPLOYMENT_NAME: 'ChurnPredictionProd'**

12. After you execute all the steps in the notebook, navigate back to your prod deployment space, **churn_prod_space,** to verify that all assets have been copied and a new model created.

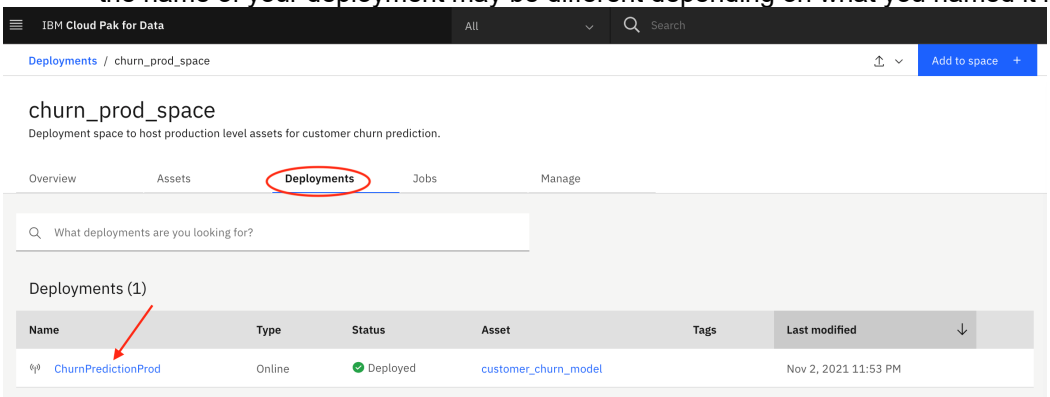
Navigate to Deployments by clicking the Navigation menu (top left hamburger icon).



13. On the Deployments page, click the Spaces tab (annotated with red oval) and select the churn_prod_space (annotated with red arrow).



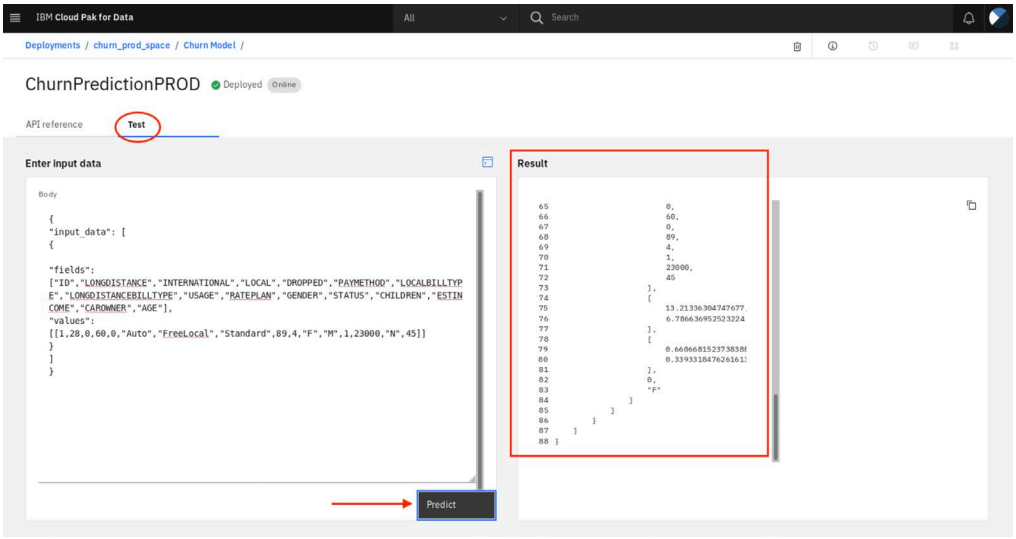
14. On the prod deployment space page, select the **Deployments** tab (annotated with red oval) and click on the production deployment, **ChurnPredictionProd** (annotated with red arrow). Note that the name of your deployment may be different depending on what you named it in the notebook.



15. On the deployed model page, click the **Test** tab (annotated with red oval), provide a sample dataset to score in the Body field and click **Predict** (annotated with red arrow). Use the following sample data as an example and note the prediction output shown in the Result section (annotated with red rectangle). Alternatively, you can copy/paste from the CICD Payload Sample boxnote.

Feel free to edit/change values and re-run the prediction to see how different features can impact the prediction.

```
{
"input_data": [
{

"fields":["ID","LONGDISTANCE","INTERNATIONAL","LOCAL","DROPPED","PAYMETHOD","LOCALBILLTYPE","LONGDISTANCEBILLTYPE","USAGE","RATEPLAN","GENDER","STATUS","CHILDREN","ESTINCOME","CAROWNER","AGE"],
"values":[[1,28,0,60,0,"Auto","FreeLocal","Standard",89,4,"F","M",1,23000,"N",45]]
}
]
}
```



At this point you've seen how to run a notebook leveraging cpdctl tool to propagate assets from one deployment space to another. You can create and schedule a job to run periodically and execute this sample notebook.

Optional As a bonus activity, configure Watson OpenScale to monitor this production churn prediction model. To do so, follow the steps in the previous module and adapt it to work with this model.

Method 2 – Git Based Flow

In this part of the workshop, we are going to use Git based automation where data assets and source code are checked into a git repository for source code management, and automation is leveraged for validation and deployment in UAT and production environments. Effectively we are propagating source code only and train and deploy models in each of the environments leveraging notebooks and jobs which are triggered from a Git repository.

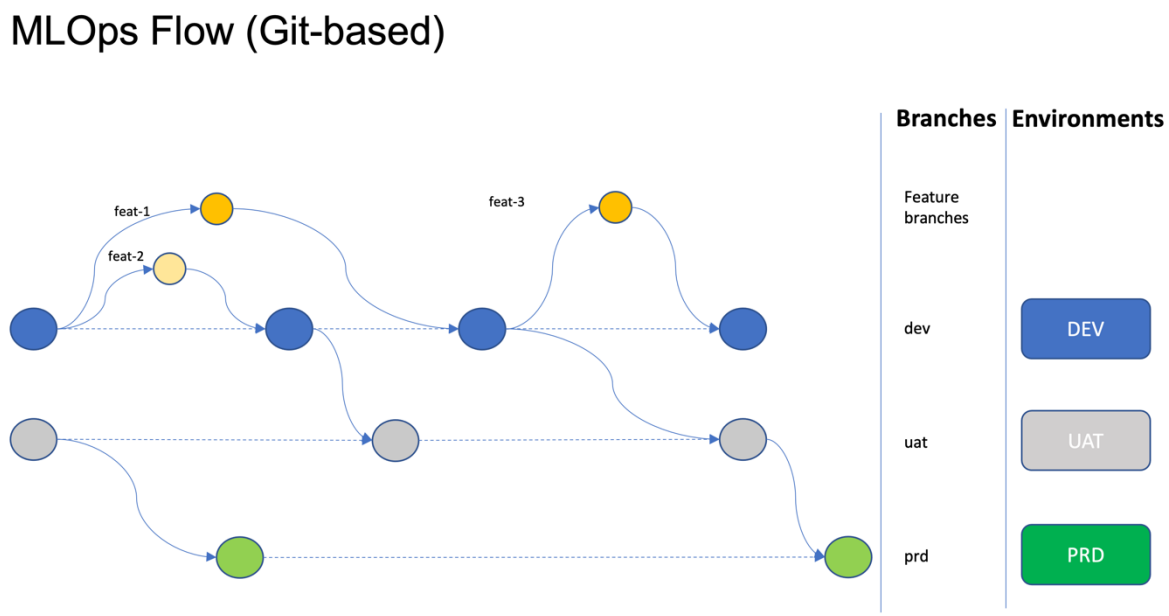
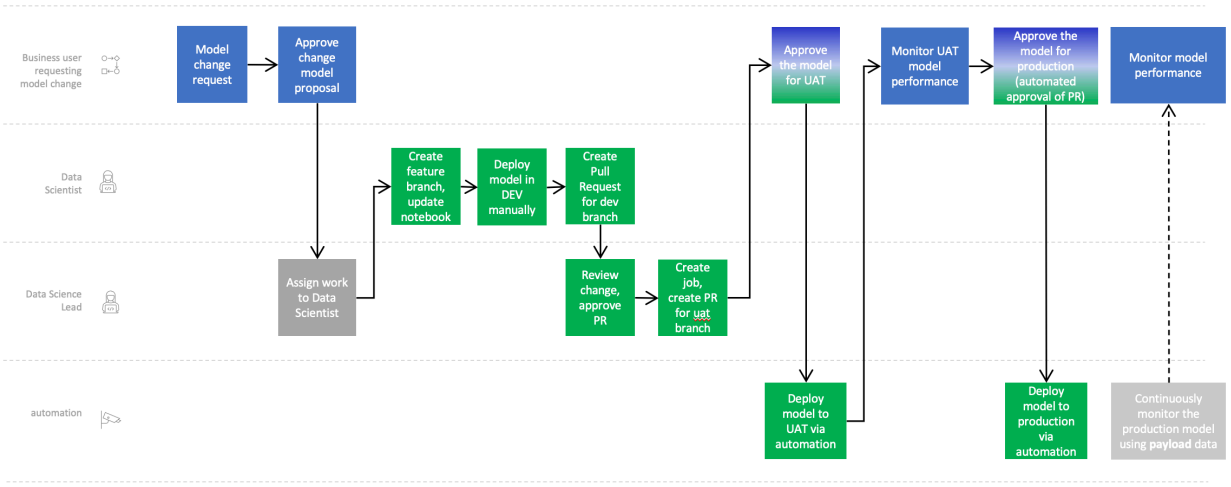


Figure 1: Git based deployment flow

Figure 1 illustrates a development and deployment process of AI models following a governed MLOps methodology applied through Git integration. Typically, the enterprise would have separate clusters or namespaces (depending on isolation needs) to support the various stages of development (training / developing), validation (evaluation/pre-production) and deploying AI (production) models.

Figure 1 depicts three git branches: **dev** (development), **uat** (user-acceptance testing), and **prd** (production) that correspond to the DEV, UAT, and PRD clusters. Data scientists mainly operate in the development cluster and interact with the git **dev** branch. Data scientists leverage the DEV cluster for experimentation and exploration where they collaborate with other ***data scientists***, ***data engineers***, and ***business SMEs*** to identify the right data sets and train the best performing AI models. As denoted, there are typically multiple branches off the **dev** git branch to add features for improving the AI model.

Model change with automation flow



The remainder of this chapter will leads through the various activities marked green in the above picture. Boxes which are marked blue and green are manual processes in this workshop but could be automated using the OpenPages workflow that was covered earlier.

A data scientist works off an existing model that is already in production but no longer meets the model KPI thresholds. The data scientist creates a feature branch from the dev branch, reviews the notebook and updates it to improve the model performance. Once satisfied with the machine learning model that delivers best performance, the data scientist checks the code and assets into the git dev branch via a pull request. The data science project lead, who owns the dev git branch, approves the submitted pull requests and tests the notebook and model.

After review (and there could be a few back-and-forth interactions), the lead data scientist would create a pull request (sometimes also referred to as merge request) to propagate the assets (notebooks) to the uat git branch for testing in the UAT environment, which typically references different data stores than the DEV environment.

In the workshop, the approval of the pull request to merge the change with the uat branch is done manually, but as depicted it could also be triggered by an approval step in OpenPages.

Deployment of the assets in the UAT environment (from the uat branch) is done via automation, also known as GitOps. A general policy in many organizations mandates that deployment of applications, which includes data science assets, is always fully automated without human intervention. First, this helps to streamline the processes but moreover, it reduces the risk of failing installations because the exact same process is executed in multiple stages before it reaches production.

Automation pulls the assets from the uat git branch into the UAT cluster to retrain the machine learning model and run validation testing against such model. The data used for validation is different from the data used for training and initial testing of the model. Validation is executed on the data assets in the uat branch.

Once UAT validation tests are concluded, the final assets (code and data assets) are checked into production git branch via a pull request. The MLOps team lead reviews and approves the

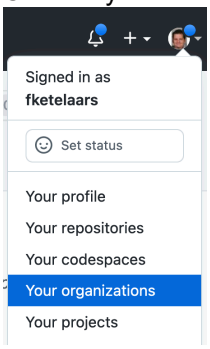
pull request to propagate the assets into the production git branch. Automation then picks up the assets from the production git branch and pushes those into the production cluster where the code is run to re-train the AI model and validate the performance. Assuming all performance checks meet expected targets, the model is deployed in the production cluster and is ready for inferencing at scale.

Preparation of the Git repository

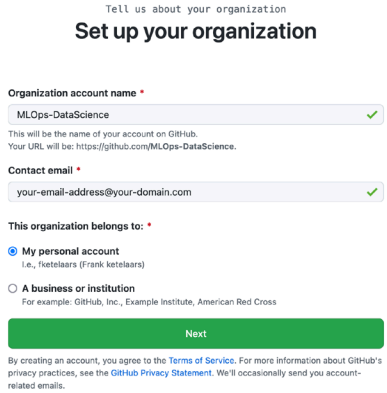
We have already prepared a Git repository template that consists of 3 branches: prd (production), uat (UAT) and dev (development): https://github.com/CP4DModelOps/mlops-churn-prediction-46. The repository also includes GitHub Actions to automate the deployment of models into the uat and prd environments.

In this exercise you will create a repository in your own GitHub organization.

1. Go to https://github.com and login.
2. Click on your user at the right top of the screen and go to Your Organizations



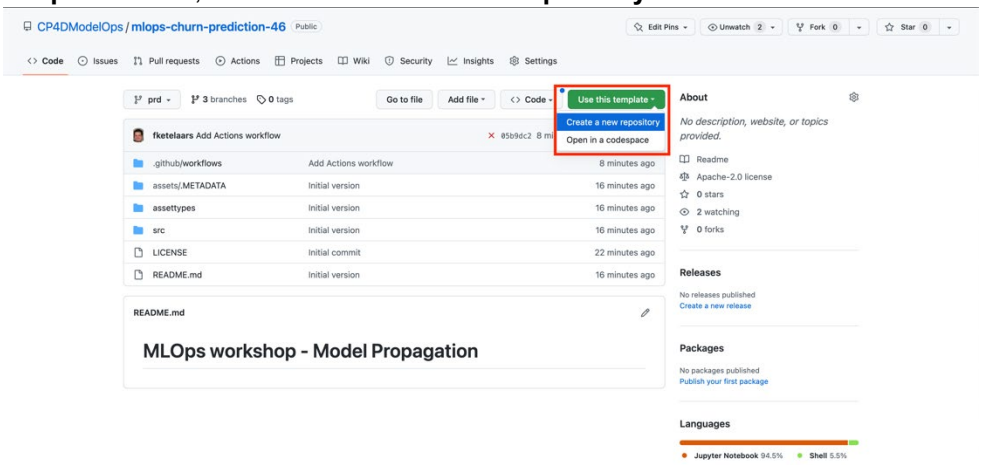
3. Create a new organization and select the Free plan
4. Enter the “Organization account name”, for example “MLOps-<your_full_name>”. The name must be unique so you may have to be creative. Also enter your e-mail address.



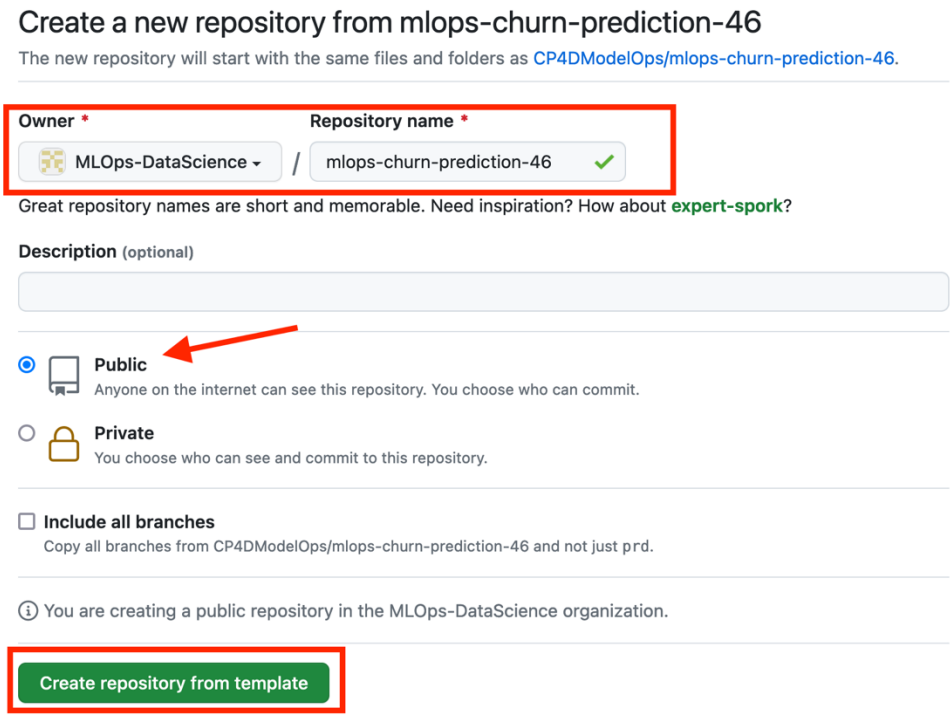
5. On the next screen, just click “Complete Setup” and then “Submit”

Now you have a new github.com organization which you can use to create the training repository:

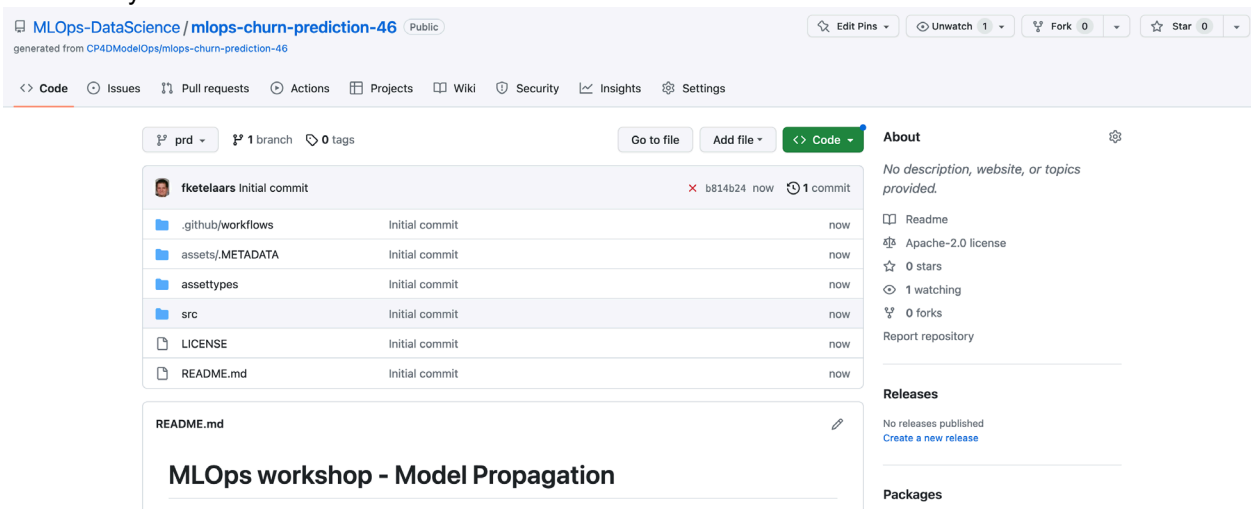
1. Go to https://github.com/CP4DModelOps/mlops-churn-prediction-46 and click the **Use this template** button; then select **Create a new repository**.



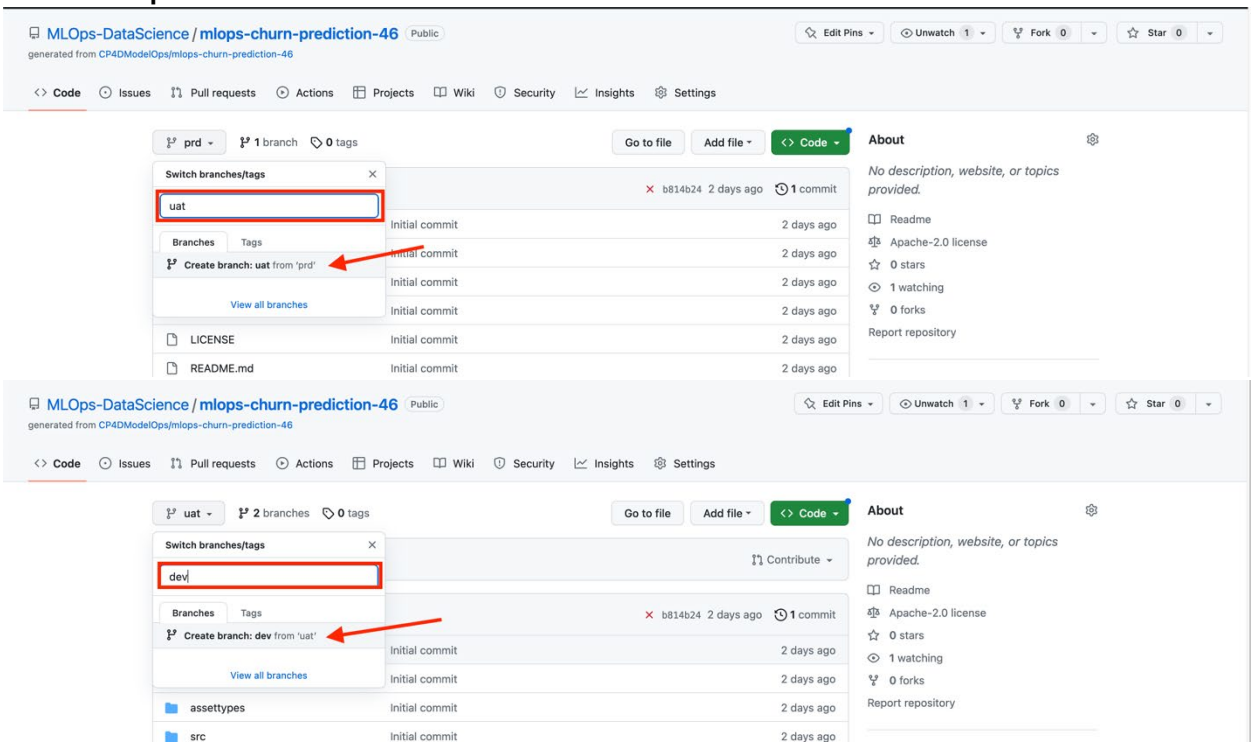
2. Select the organization you just created from the list and **Create repository from template**.



3. GitHub will generate the repository and then show it. You will notice that the repository has 1 branch only.



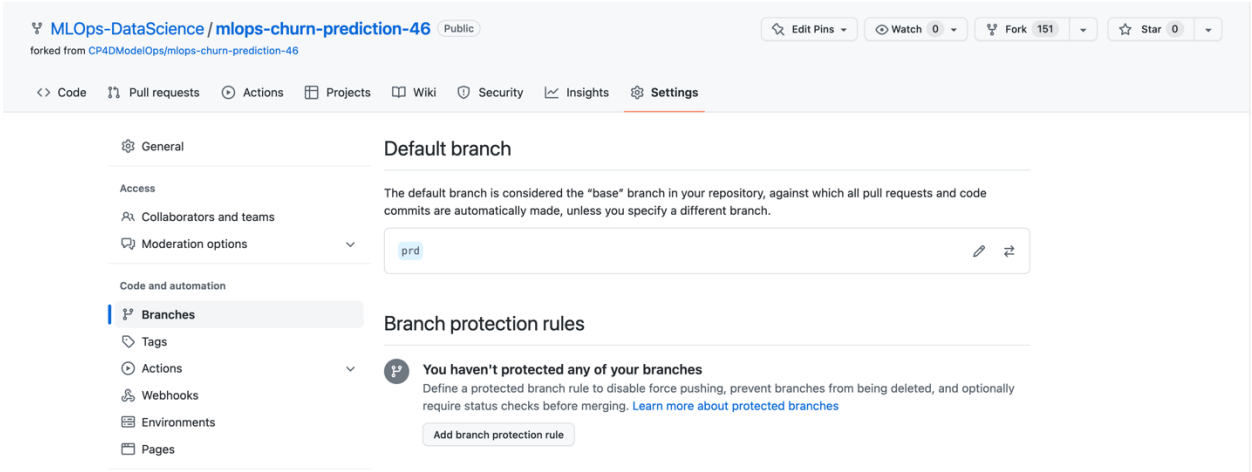
4. Click on the **prd** branch button and create the other two branches: **uat** and **dev**.



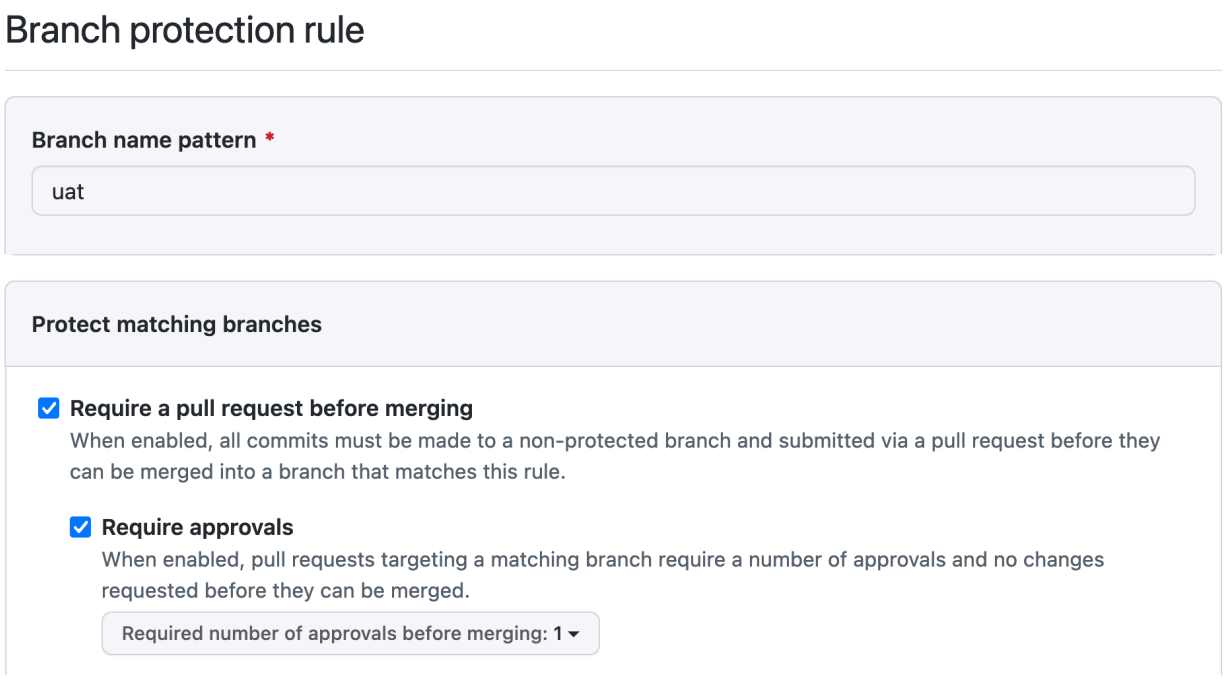
We will now set up a branch protection rule to simulate a “real” scenario where pull requests must be approved before being merged.

1. Click on Settings and then Branches on the left-hand side. You will see that there are no branch protection rules have been set up yet. To protect the uat and prd branches we will create these

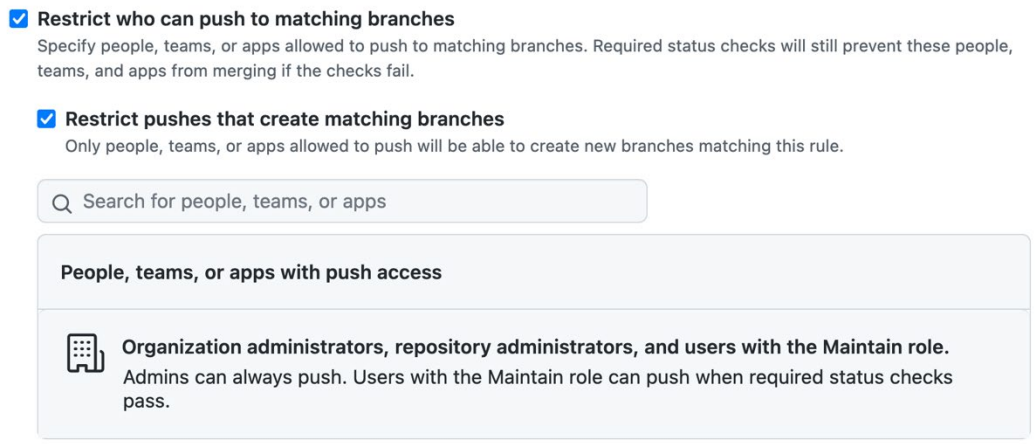
rules.



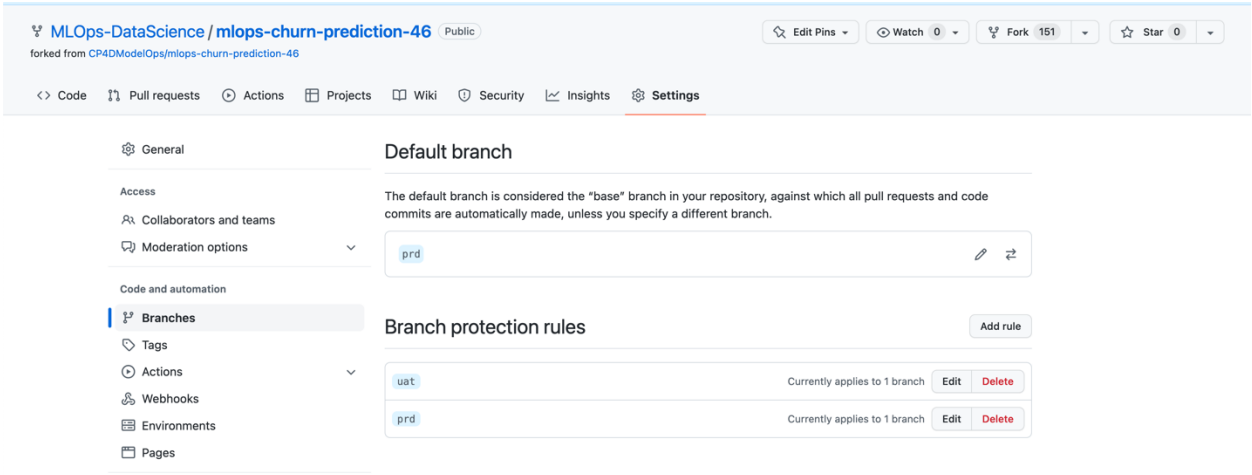
2. Click on **Add branch protection rule** to create a new rule for the **uat** branch. Enter **uat** for the Branch name pattern and select the "Require a pull request before merging" checkbox.



also scroll down and select the “Restrict who can push to matching branches” checkbox:



3. Scroll down and click the Create button.
4. Repeat the above steps for the **prd** branch.
5. When done you should see the following branch protection rules:



As your organization only has a single user (yourself) and you are the administrator, we cannot truly set up a formal approval process in which data scientists only have write access to the repository and the MLOps staff can approve requests. In an actual implementation, the production code could also reside in an upstream repository with different permissions. Setting up the GitHub topology with different repositories and teams is beyond the scope of this workshop.

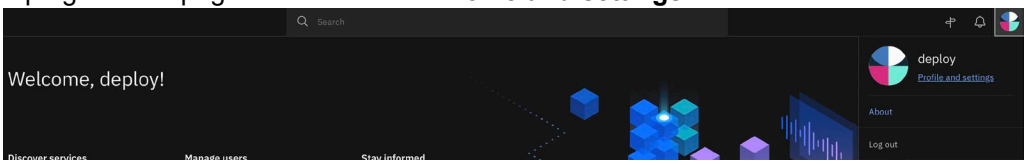
Setting up GitHub Actions for automation

The GitHub repository already includes a GitHub Actions pipeline to automate deployment of models when a Jupyter notebook is pushed to the **uat** or **prd** branches. To ensure the pipeline can connect to your cluster you need set secrets and environment variables for GitHub Actions.

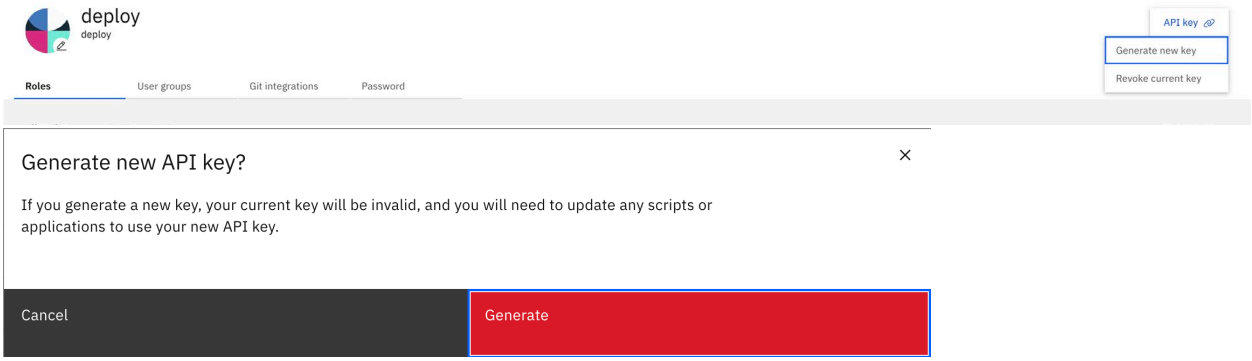
Generate API key for deployment

When deploying the models, a Cloud Pak for Data user with sufficient permissions must authenticate to the cluster. For this workshop, we choose the **deploy** user to ensure the user has the ability to create deployment spaces and run notebooks. In a production implementation, you could authenticate using different users for UAT and Production, or implement a whole different authentication process.

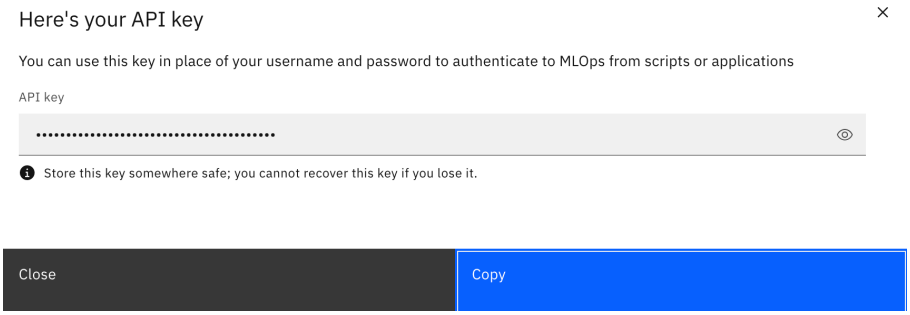
1. Go to Cloud Pak for Data and log on as the **deploy** user. When logged on, click on the icon at the top right of the page and then select **Profile and settings**.



2. Once on the **deploy** profile and settings page, click **API key** at the top right of the page, then click **Generate new key** to generate a new API key and click the red **Generate** button.



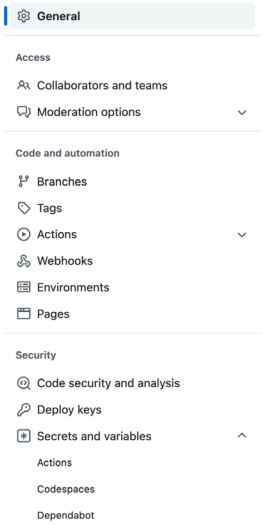
3. Cloud Pak for Data will generate a new API key and revoke any old key that existed. Copy the key and keep it in a document. You will need it in the next step and potentially in some other places as well.



Configure credentials for GitHub Actions

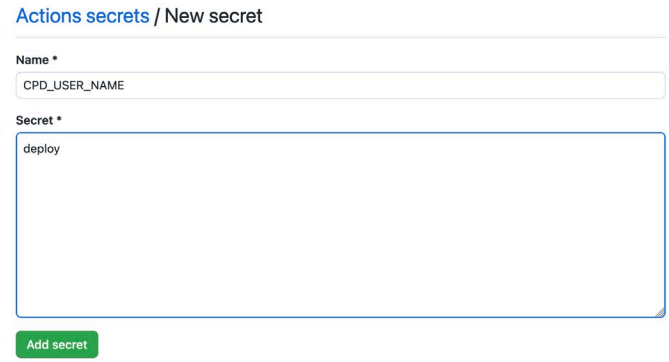
1. While on the main page of your repository, click the **Settings** link. A navigation menu will appear on the left hand side of the page.

2. Expand the **Secrets and variables** menu option and click **Actions**

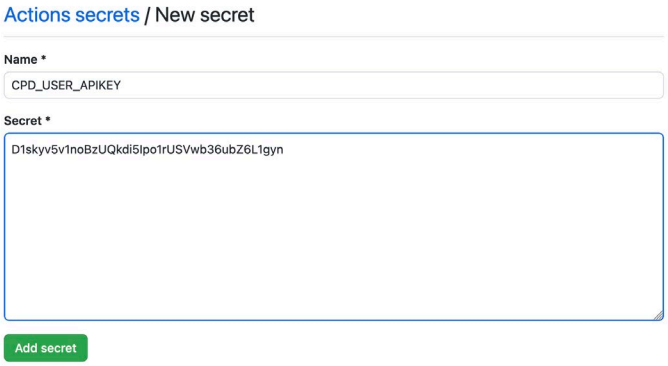


3. **You will create 2 Actions secrets that hold the user and password to connect to Cloud Pak for Data. These will be passed to the automation script as environment variables.**

4. Click the **New repository secret** button on the top right of the page and add the **CPD_USER_NAME** secret. Below you will see how the **deploy** value is set for the **CPD_USER_NAME** secret. You can choose any user that has the permissions to log in to Cloud Pak for Data and create deployment spaces.

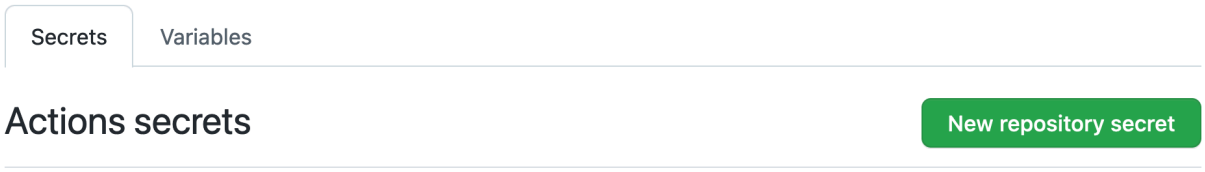


5. Next, create the **CPD_USER_APIKEY** secret.

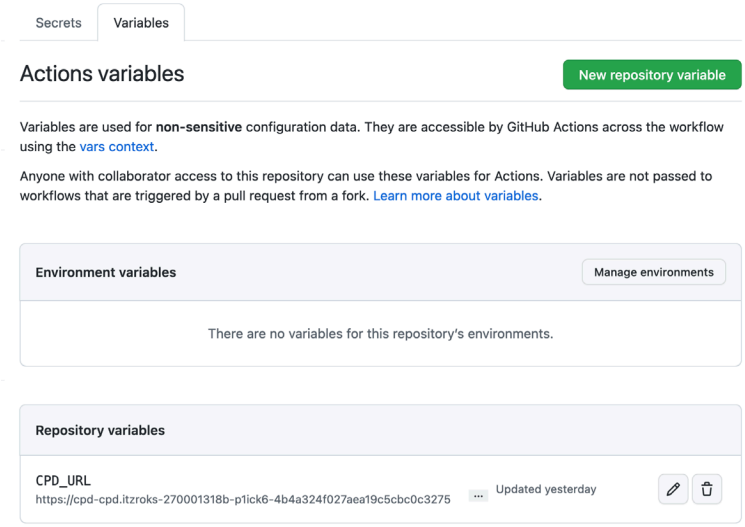


6. API key: TY9MNZqSi84BoToRsudq5pNlM1rzQRDxqRkuZhTE [remove]

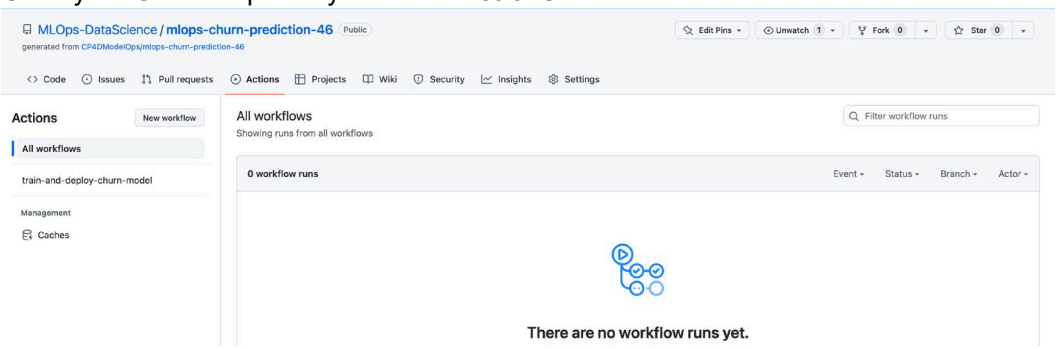
7. The Cloud Pak for Data URL is not considered a secret, hence we will add this as an environment variable to the GitHub repository. Click on the **Variables** tab at the top of the page.



8. Similar to how you have added the secrets, now add the **CPD_URL** environment variable and set this to the Cloud Pak for Data URL.

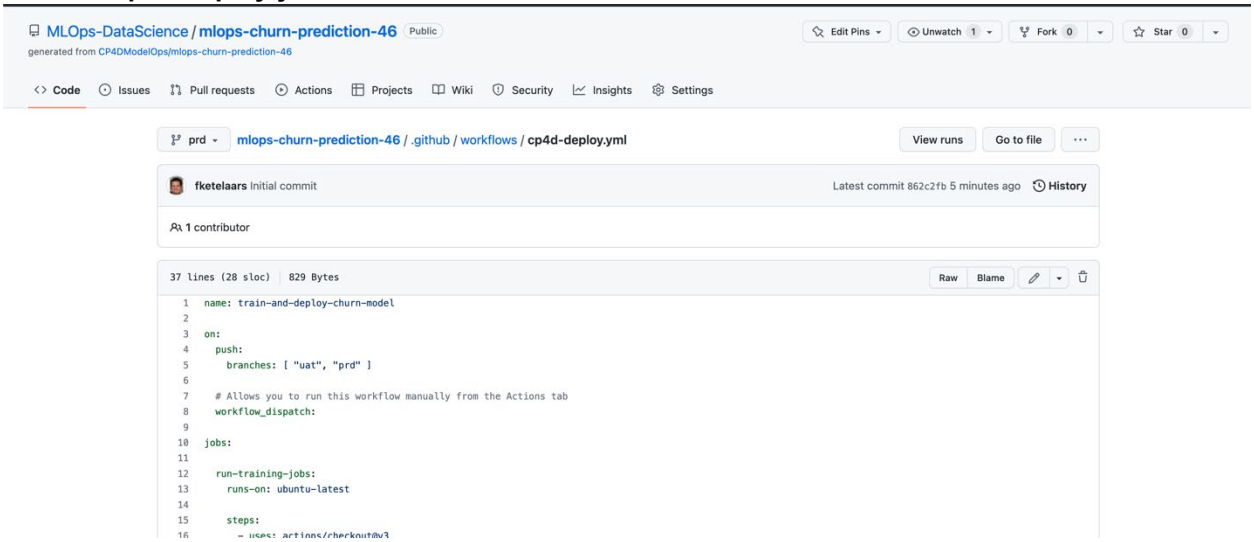


9. Go to your GitHub repository and click **Actions** in the header.



You will see the workflows on the left hand side. Click the **train-and-deploy-churn-model** workflow. The YAML file that defines the workflow appears at the top.

10. Click the **cp4d-deploy.yml** link to view the source of the workflow.



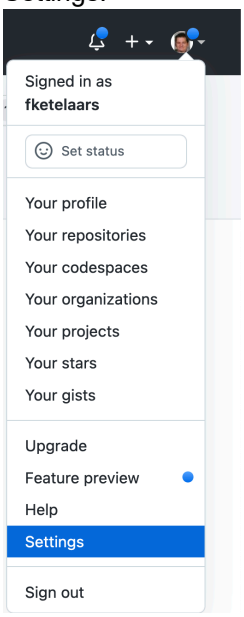
Observe that the workflow consists of a single job, **run-training-jobs**. This job does the following:

a. Checkout the repository that holds the notebook and workflow. This creates a local copy of the repository.
b. Change the flag of the scripts to make them runnable, just in case.
c. Download **cpdctl**. This is the Cloud Pak for Data command line tool to handle projects and assets.
d. Set up the **cpdctl** credentials. This will create a $HOME/.cpdctl directory and store the CP4D user name, API key and URL.
e. Run the deployment job(s). This script checks for any jobs starting with “deploy” and runs them. In a further step the data science lead creates the **deploy-customer-churn-model** job, which then gets executed by this script whenever a change is pushed to the **uat** or **prd** branches.

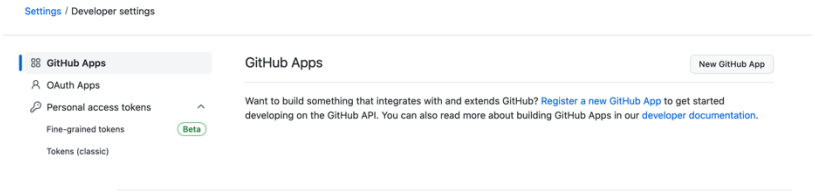
Obtain Git token

Now that you have finished creating your repository and setting up your GitHub Actions credentials, we can start setting up Watson Studio with Git integration. Watson Studio needs to be given write access to your repository via a token.

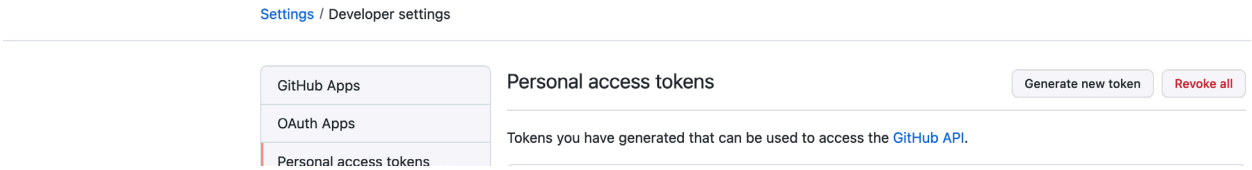
1. Go to your GitHub repository and click on your user at the right top of the page and select Settings.



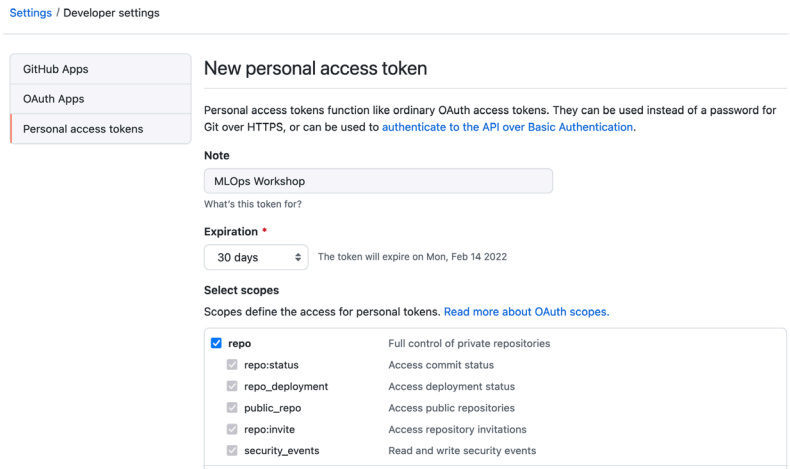
2. Scroll down until you see **Developer settings** in the left sidebar and click on it.
3. Click on Personal access tokens, then select Tokens (classic)



4. Click on **Generate new token** and then select **Generate new token (classic)**

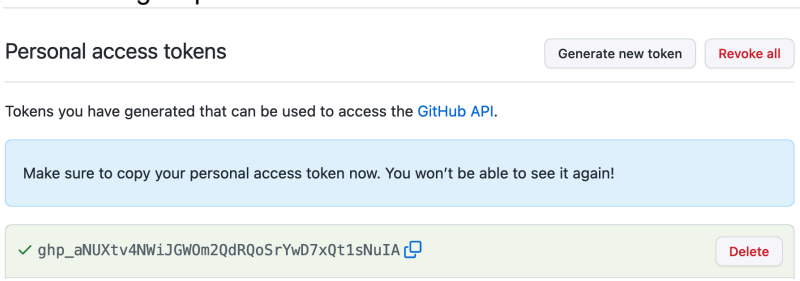


5. Enter the name of the token and ensure the **repo** box is checked.



Scroll down and create the new token.

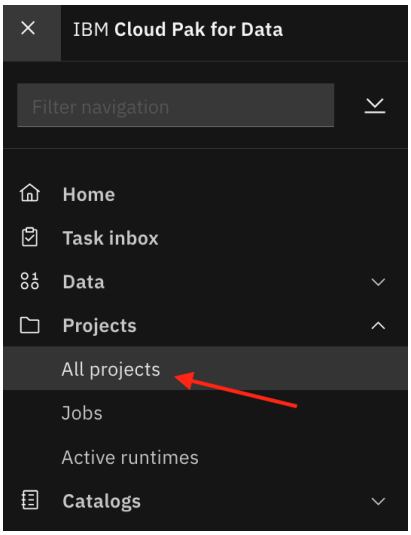
6. The token is displayed only once; make sure you copy it. You will need it **multiple times** during the following steps.



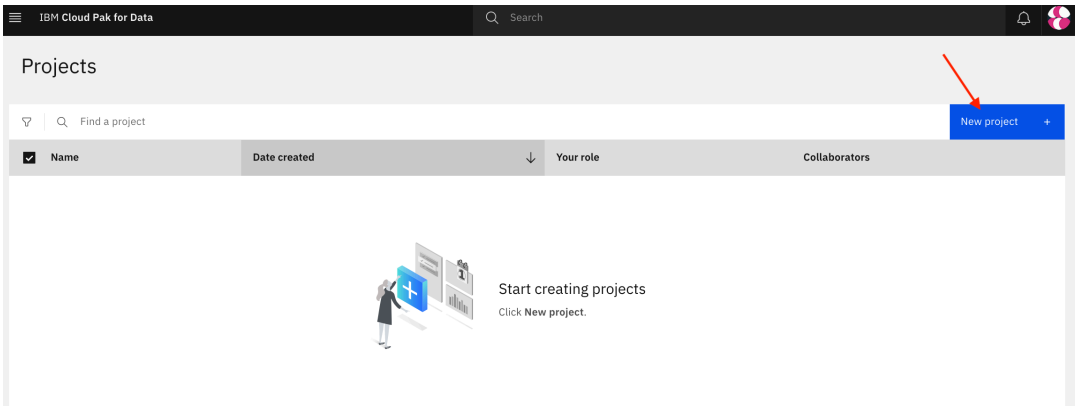
Set up Watson Studio project

In IBM Cloud Pak for data, a project is how you organize your resources to achieve a particular goal. A project allows for high-level isolation, enabling users to package their project assets independently for different use cases or departments. Your project resources can include data, collaborators, scripts, and analytic assets like notebooks and models.

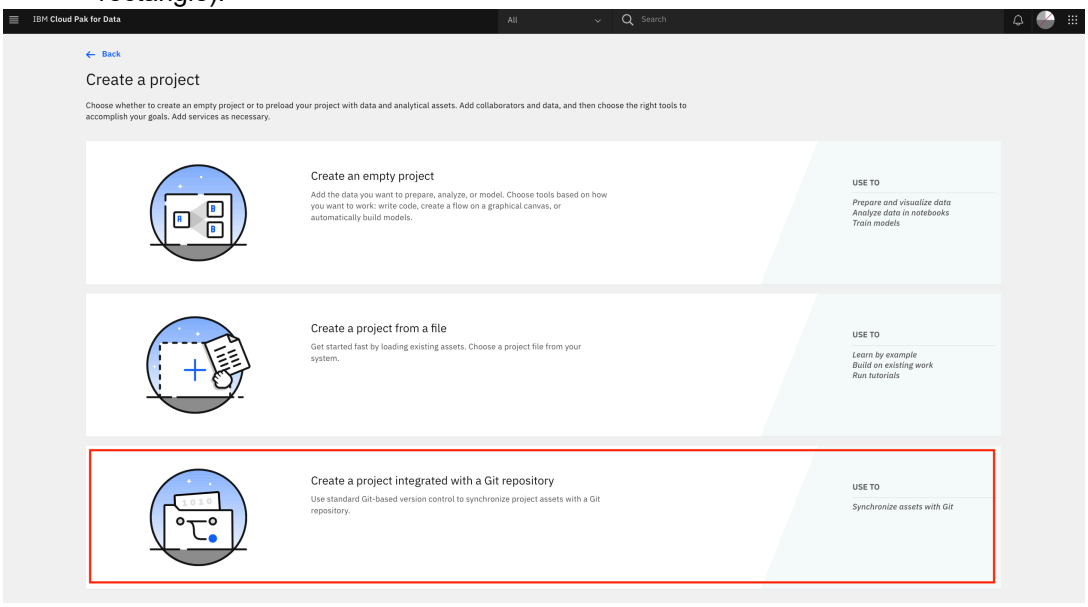
1- Log into Cloud Pak for Data as ***dslead*** user.
2- Select All projects by clicking on the Navigation menu (top left hamburger icon) and selecting **Projects → All projects** (annotated with red arrow).



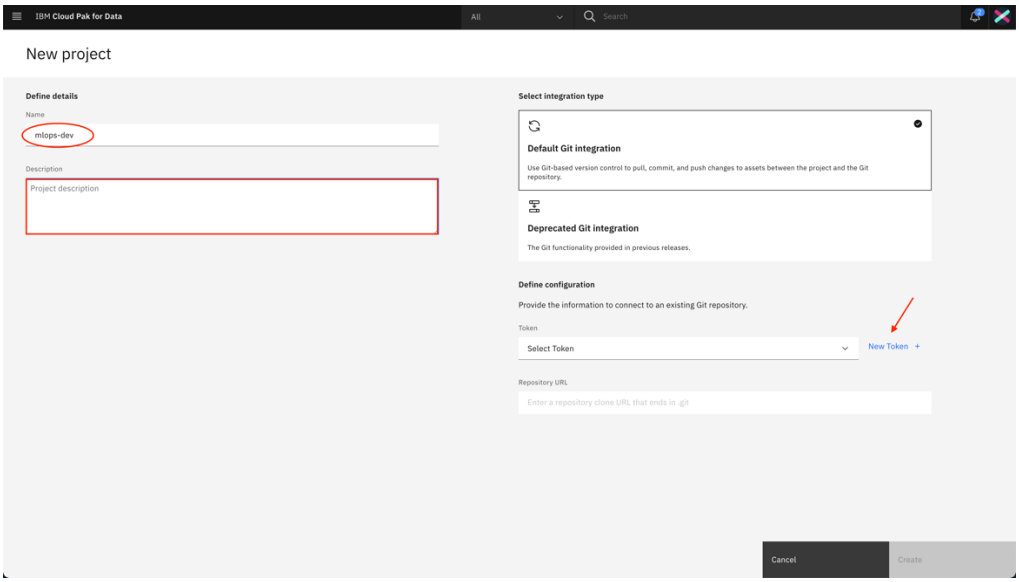
3- Click on **New project** (annotated with red arrow) to create a new project.



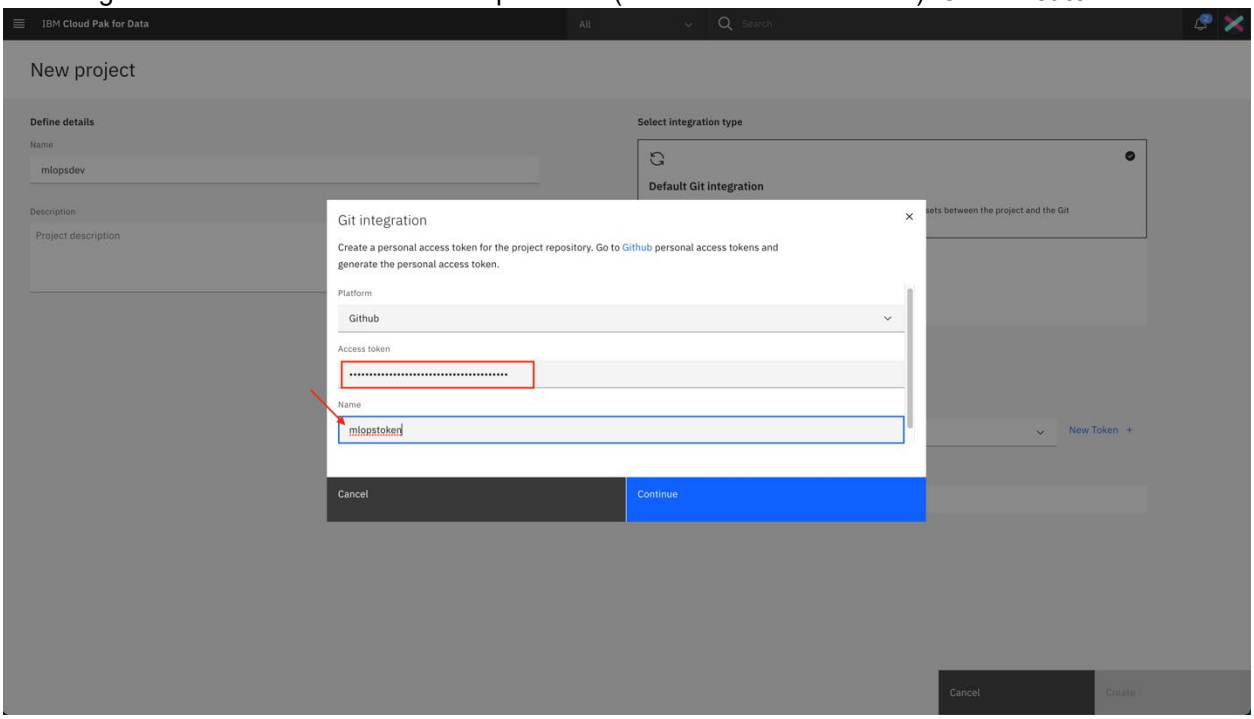
4- Select the **Create a project integrated with a Git repository** option (annotated with red rectangle).



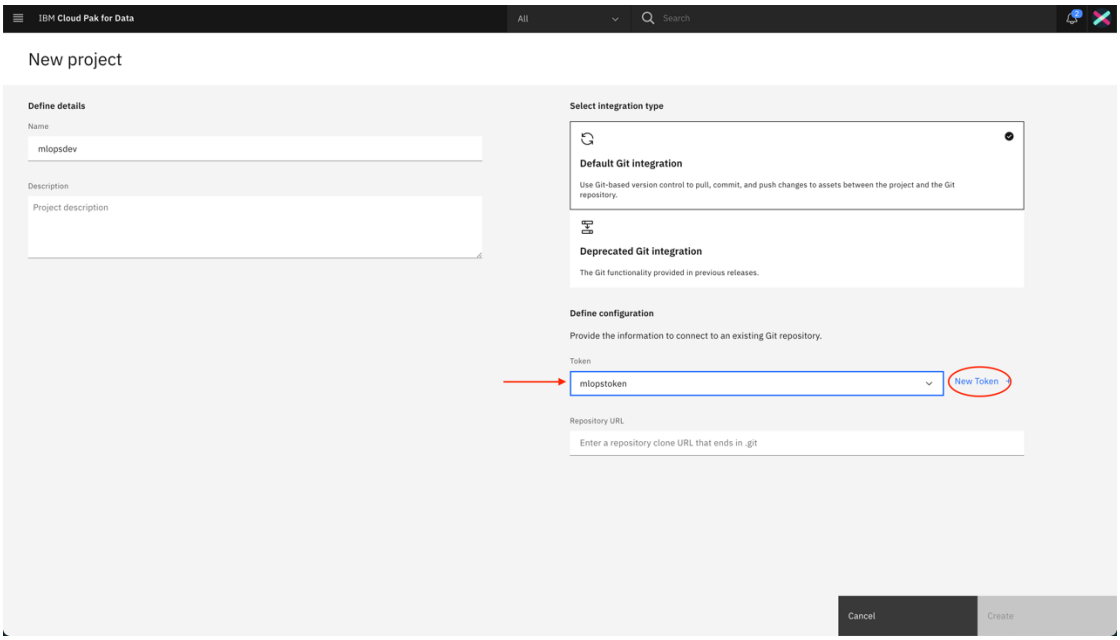
5- On the **New project** page, provide a Name (mlops-dev) and Description (optional) for the project. For integration type, select the Default Git integration. You will need to upload the GitHub token you generated before token to connect to your Git repository. Click the New Token link (annotated with red arrow).



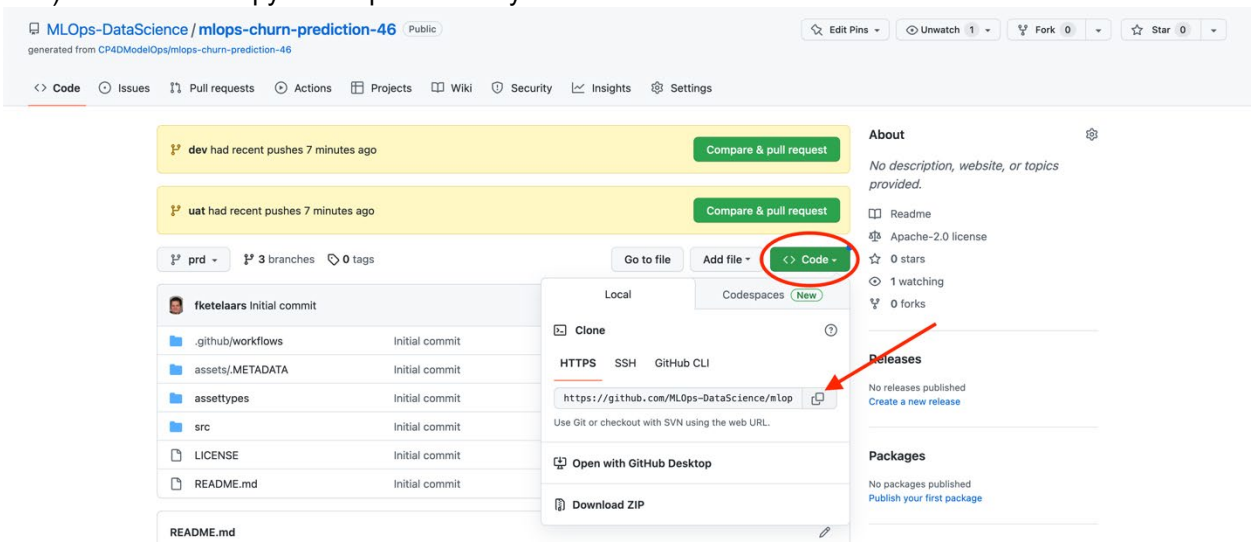
6- On the Git integration pop-up, provide the git access token (annotated with red rectangle) and give it a name for reference – mlopstoken - (annotated with red arrow). Click **Create**.



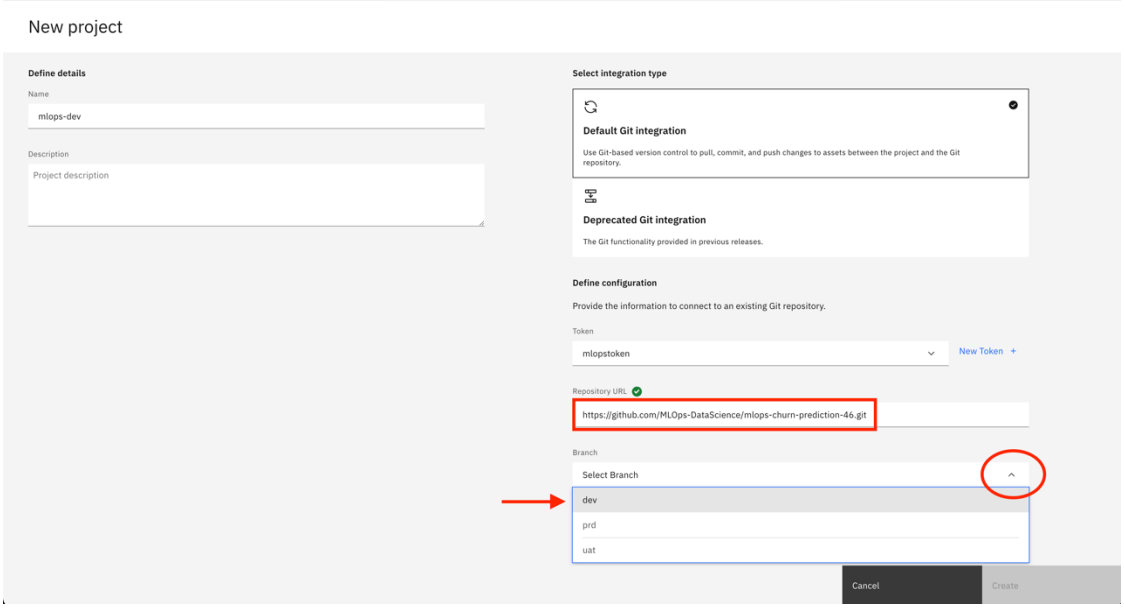
7- Next click the token drop down (annotated with red oval) and select the token you created – mlopstoken - (annotated with red arrow).



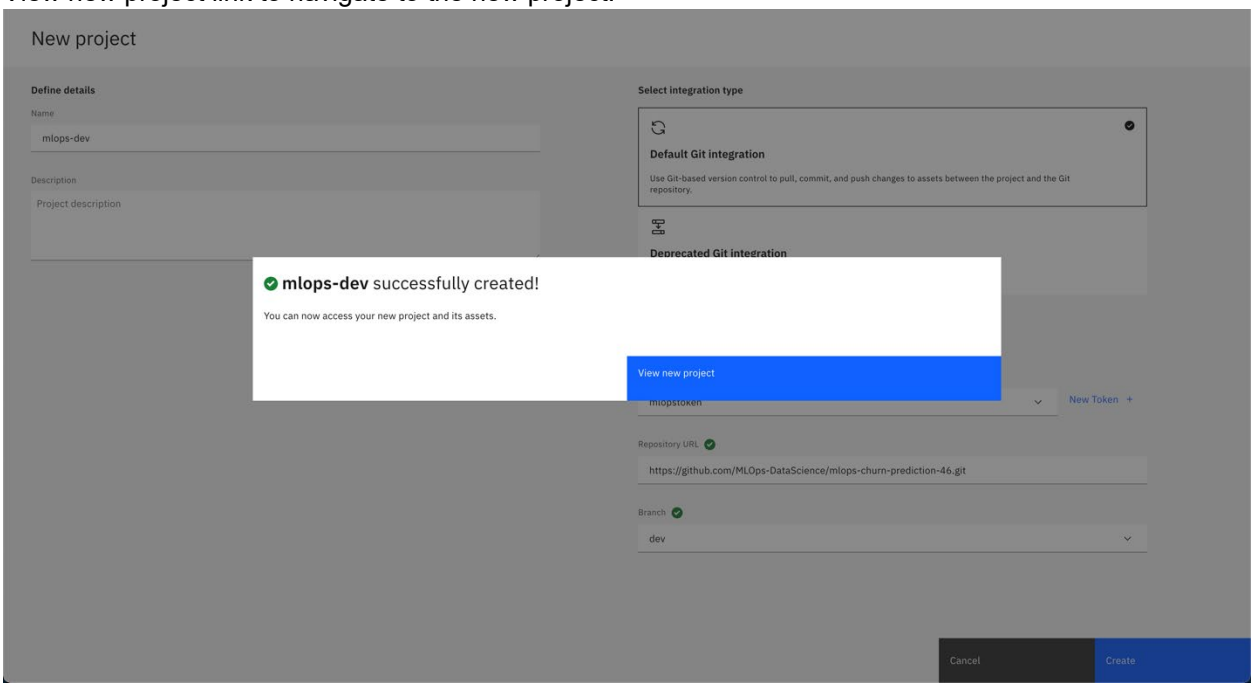
8- Next step is to provide the GitHub repository and branch to associate with your project. The rest of the instructions in this module reference the following repository **https://github.com/MLOps-DataScience/mlops-churn-prediction-46** but you will reference your own repository you generated from the upstream repository. For the Repository URL field, provide HTTPS reference for your Git repository. You can copy this by clicking on the Code button (annotated with a red oval) and then the copy button pointed to by the red arrow.



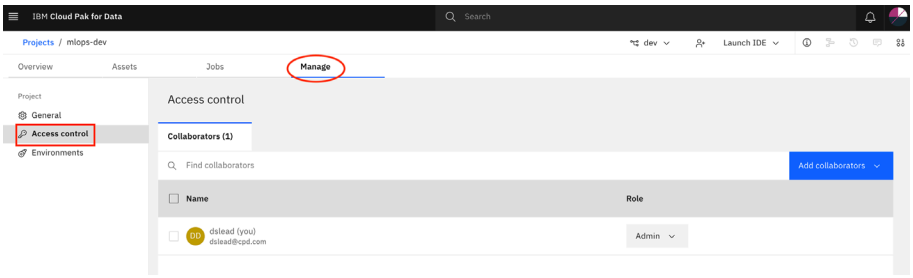
9- Go back to your Create project window and paste the URL. Once you provide that, the Branch field should become available. Click the drop-down menu (annotated with red oval) and select the **dev** branch (annotated with red arrow). When your New project looks like the following, click **Create** to create the new project.



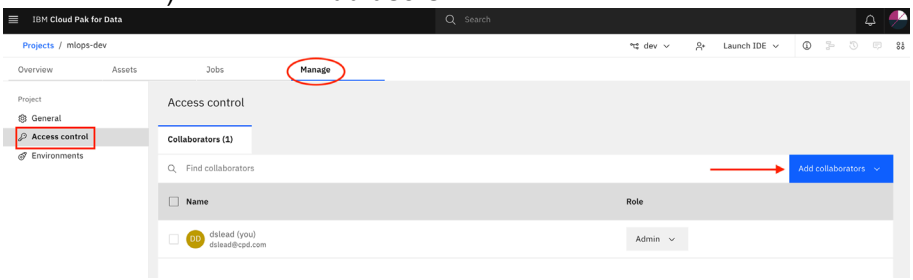
10- You should see a pop-up message stating that the project is created successfully. Click on the View new project link to navigate to the new project.



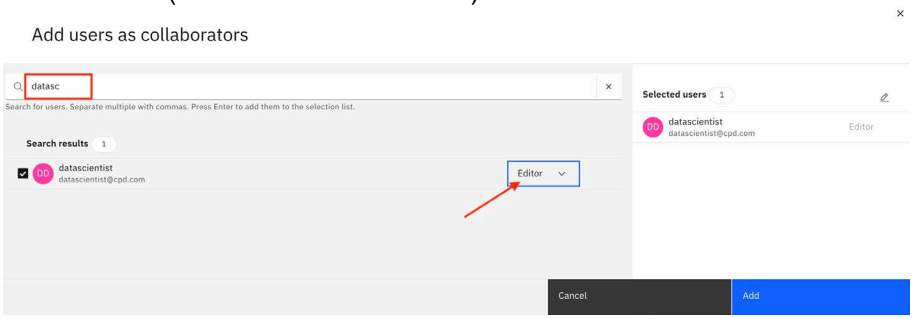
11- Take a few minutes to review the project information. The Overview tab provides general information about the project like Assets, Project history, and Storage used. Click the **Manage** tab (annotated with red oval) and select **Access control** (annotated with red rectangle) to review which users have access to the project. In this case, it should have the **dslead** user as the only collaborator in this project.



You can add other users or user groups and assign them the relevant permissions to collaborate on this project. Typically, a data science project involves multiple users and roles who collaborate on developing and evaluating AI models.
Add the **datascientist** user as an Editor to the project. Click **Add collaborators** (indicated by the arrow) and select **Add users**.



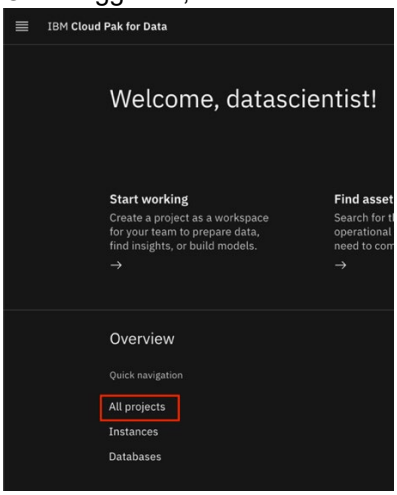
Enter **datascientist** in the field marked by the red rectangle and then select the appropriate user. Specify role as Editor (annotated with red arrow) and click **Add**.



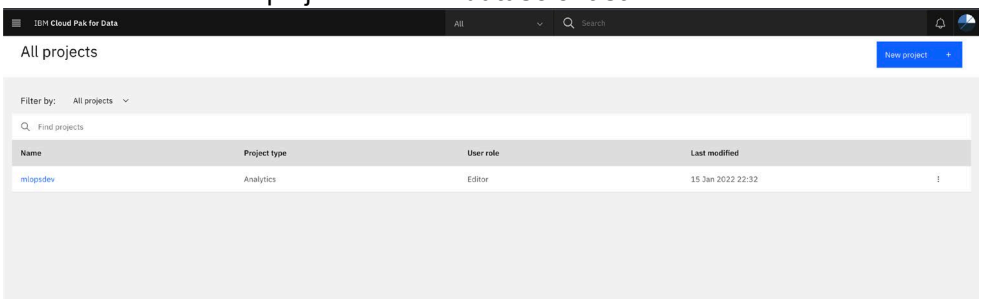
Click the **Assets** tab (annotated with red oval) to review the assets associated with the project. You will not see any assets in this view; notebooks and other data assets can be found from the JupyterLab interface.

12- Now that the project has been created, the **datascientist** user will start the JupyterLab IDE to work on a new version of the churn prediction model.
Log out from Cloud Pak for Data and login again using the **datascientist** user.

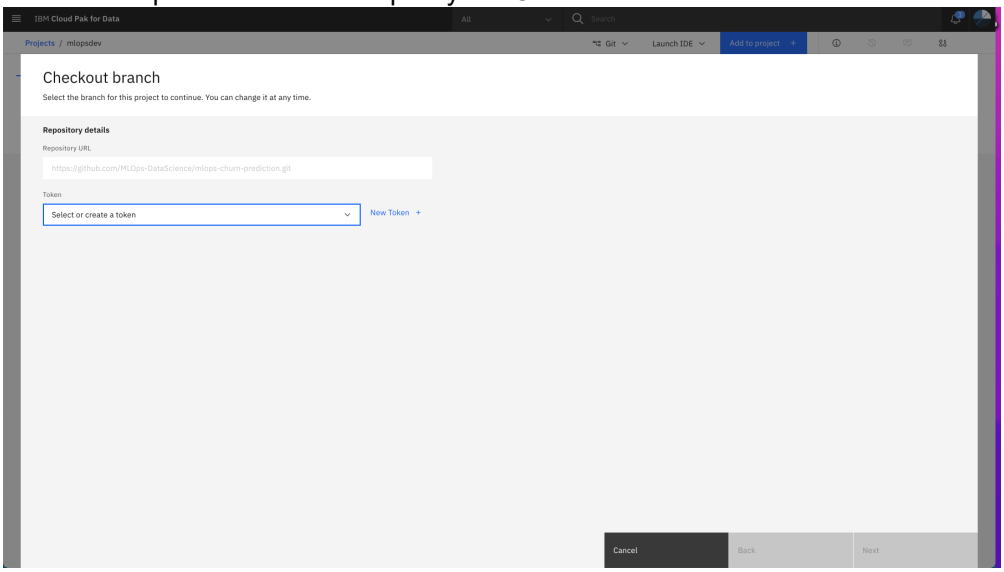
13- Once logged in, click on the All Projects link at the starting page.



14- You will see the list of projects that the **datascientist** user has access to.

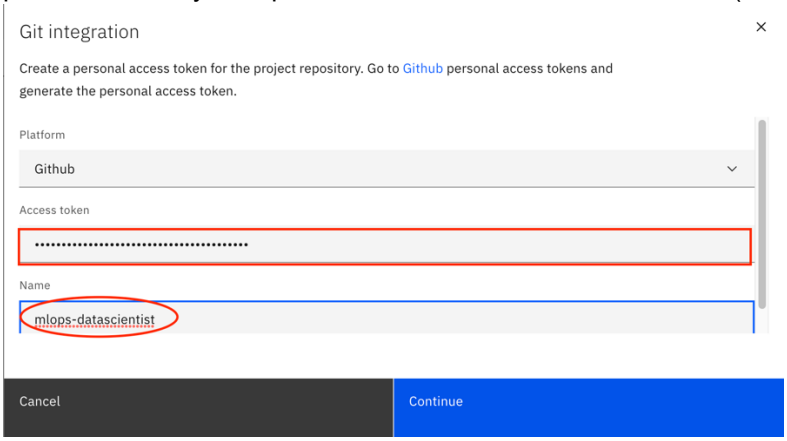


15- Open the project by clicking on its name (**mlops-dev**). Because the project is connected to a Git repository and the **datascientist** user did not open this project before, you will be asked to checkout a specific branch and specify the Git token.

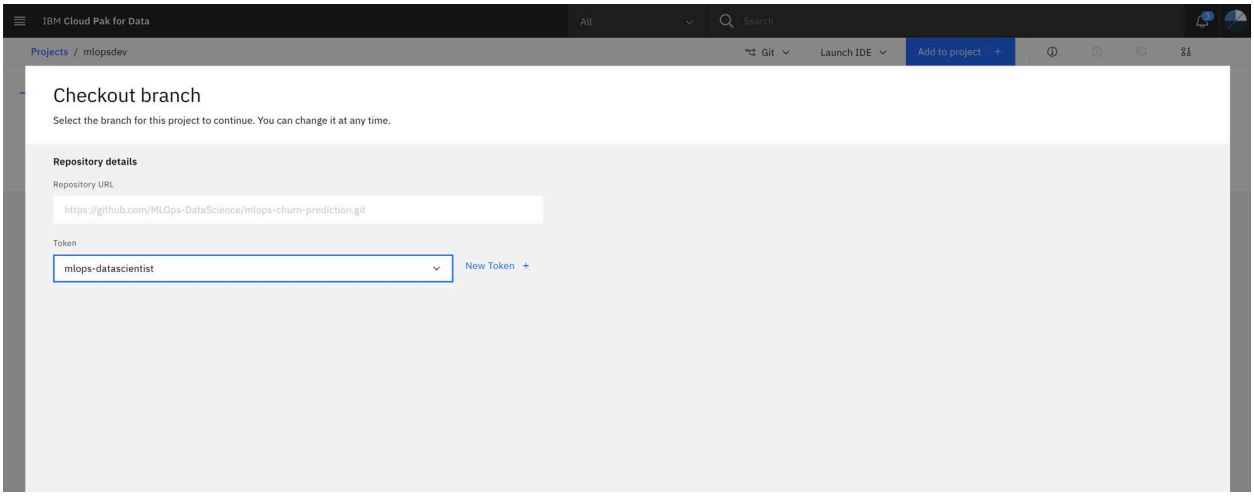


16- For convenience, we are going to use the same Git token that was used by the dslead user. Normally, every user would have their own credentials to login to the Git repository and tokens would not be shared between project members. Click the “New token” link and create a token with a new name, in the below case this is **mlops-datascientist** but you can pick any name. Also

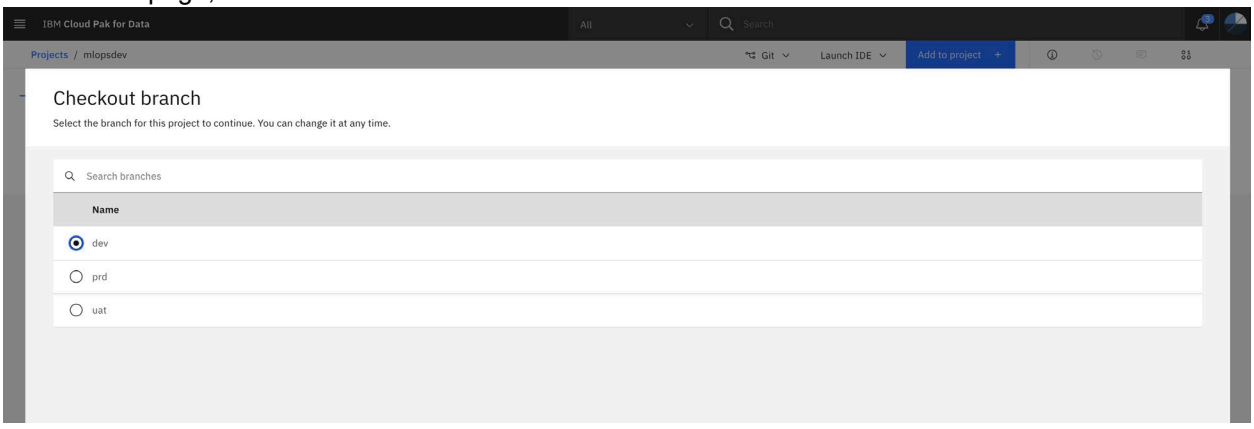
paste the token you captured before in the Access token field (indicated by the red box).



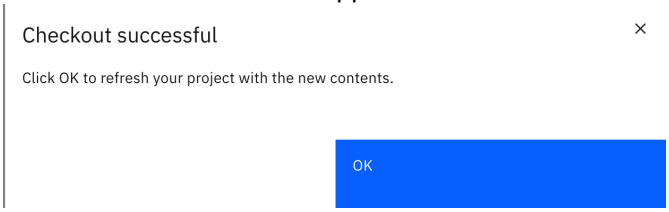
17- Now, you will return to the page to checkout the branch, select the token you just created and click Next.



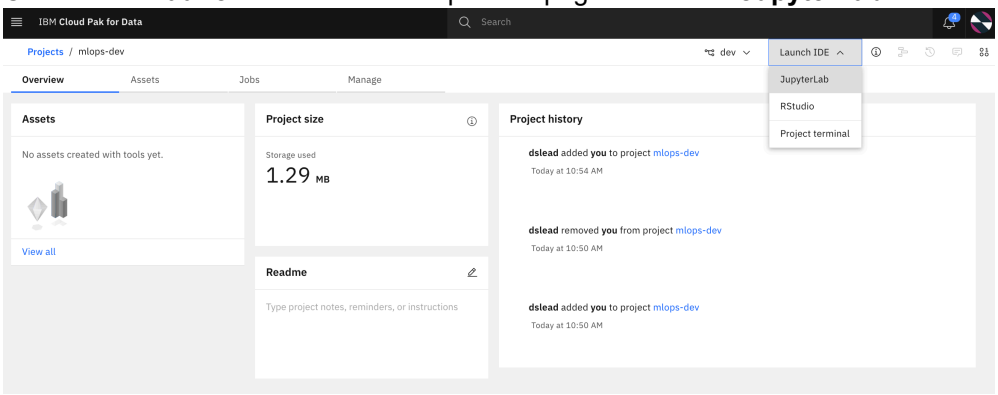
18- On the next page, select the **dev** branch and click the Select button.



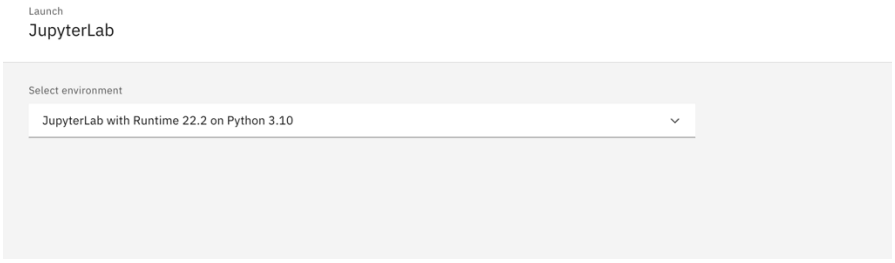
A confirmation window will appear.



19- Click on the **Launch IDE** link at the top of the page and select **JupyterLab**.



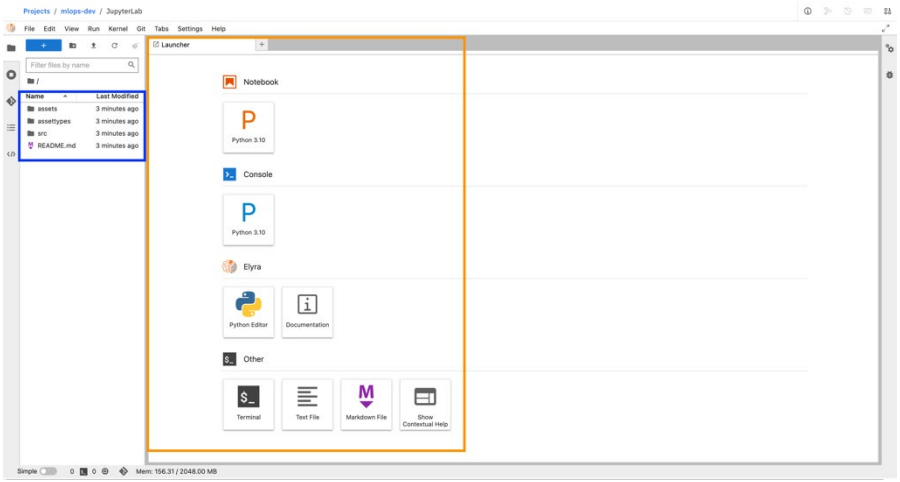
20- On the panel that shows up, select **JupyterLab with IBM Runtime 22.2 on Python 3.10** and then click **Launch**.



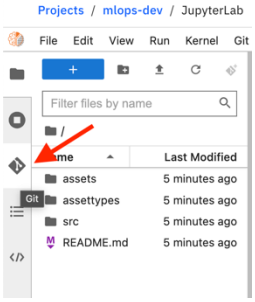
21- Once JupyterLab has started, take some time to look at the different areas of the page that is shown. On the far-left side (marked with a red box) you can browse the folder contents, view the active kernels and other tabs and view the Git repository state such as current branch and any changes that have already been done.
Also on the left-hand side (annotated with a blue box) you see the “root” folder of the repository with the **src** folder. This is the folder where the input files and notebooks are held.

WARNING: Do not store any files in the **assets** or **assettypes** folders; these are for internal use of Watson Studio.

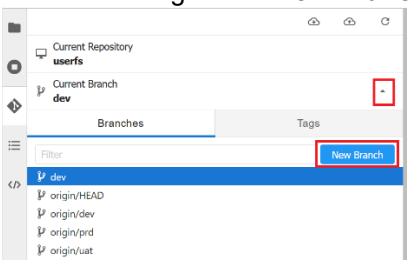
The middle part (marked by an amber box) is used to create new notebooks, open a Python console, a shell terminal and a few other applications.



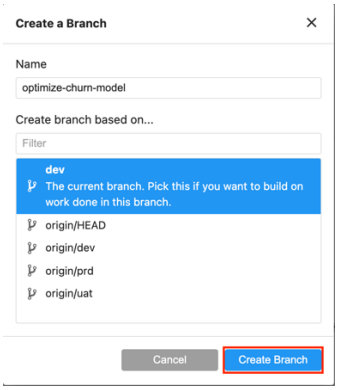
22- In this exercise, we are going to make a change to an existing notebook. First, we will make sure we make the changes in a “feature” branch so that can validate the changes without changing the notebook that may be used by other team members. Click on the “Git” icon at the far left as indicated by the red arrow.



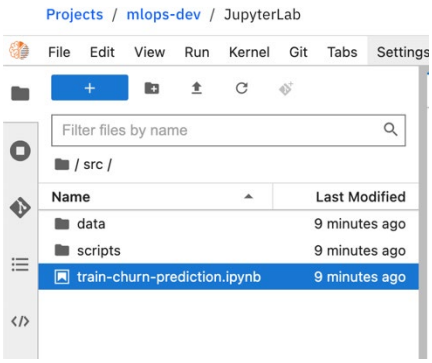
You will see your current branch (dev). Create a new branch by expanding the **dev** branch and clicking on the **New Branch** button.



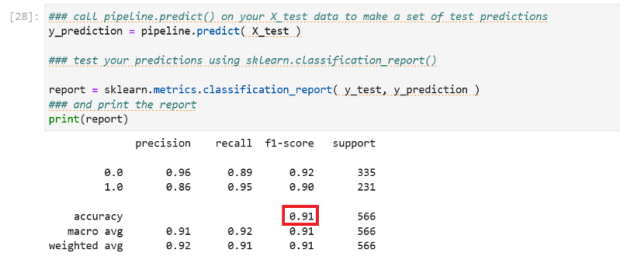
A new window appears in which you can enter the name of the new branch. In our example we have chosen for the **optimize-churn-model** name. Once you have entered a name, click the **Create Branch** button in the window, as annotated by the red box.



23- Navigate to the sources folder by clicking the folder icon at the top left, then double-click the **src** folder and open the **train-churn-prediction.ipynb** notebook by double-clicking it.



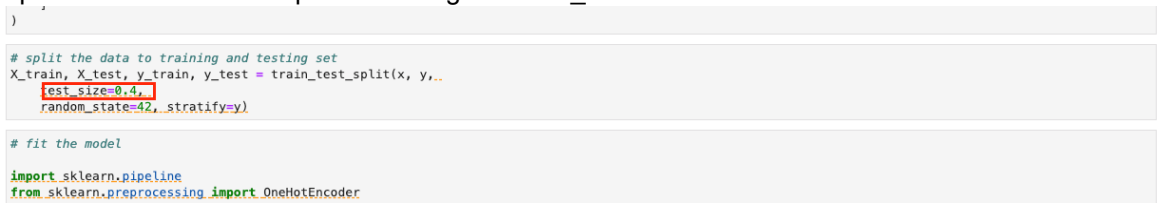
24- Scroll down to the section above **Evaluate** and notice the accuracy of the model. This is currently approximately 0.91.



```
[28]: ### call pipeline.predict() on your X_test data to make a set of test predictions
      y_prediction = pipeline.predict( X_test )

      ### test your predictions using sklearn.classification_report()

      report = sklearn.metrics.classification_report( y_test, y_prediction )
      ### and print the report
      print(report)
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.89      0.92       335
         1.0       0.86      0.95      0.90       231

    accuracy                           0.91       566
   macro avg       0.91      0.92      0.91       566
weighted avg       0.92      0.91      0.91       566
```

25- Scroll up to where the data is split and change the test_size from 0.4 to 0.2.

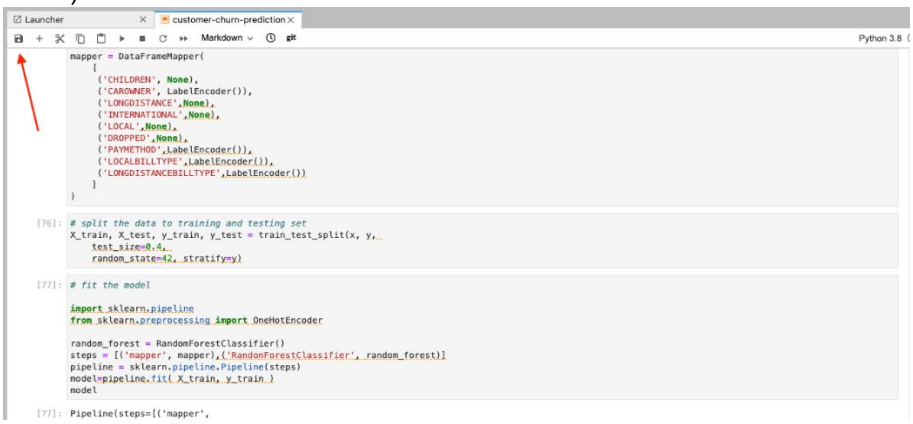


```
)
```

```
[76]: # split the data to training and testing set
      X_train, X_test, y_train, y_test = train_test_split(x, y,
          test_size=0.4,
          random_state=42, stratify=y)
```

```
[77]: # fit the model

      import sklearn.pipeline
      from sklearn.preprocessing import OneHotEncoder
```

26- Save the notebook by clicking the disk at the top left of the notebook (as indicated by the red arrow).



27- Run the notebook in its entirety by selecting **Run → Run all cells** from the top menu. You will find that all cells in the notebook will be run and when they have run, a sequence number will appear between the square brackets left of each of the cells.

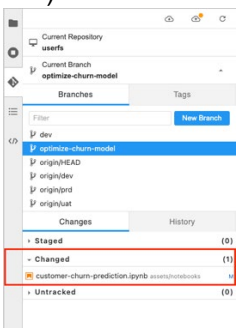
28- Scroll down to determine the accuracy of the model, which should be around 0.94 (although your results may differ slightly).

```
### call pipeline.predict() on your X_test data to make a set of test predictions
y_prediction = pipeline.predict( X_test )

### test your predictions using sklearn.classification_report()

report = sklearn.metrics.classification_report( y_test, y_prediction )
### and print the report
print(report)
```

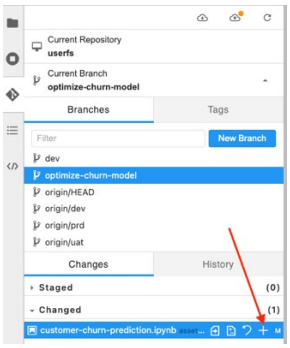
```
              precision    recall  f1-score   support

         0.0       0.98      0.92      0.95       168
         1.0       0.89      0.97      0.93       115

    accuracy                           0.94       283
   macro avg       0.93      0.95      0.94       283
weighted avg       0.94      0.94      0.94       283
```

29- Assume that you're happy with these results and you wish to promote the model to the **dev** environment. Wait for the entire notebook to finish running, this will take about 3-5 minutes to complete. Save the notebook again and go to the "Git" panel. Remember you have been there before when you browsed the Git branches.

30- As you can see your changed notebook appears in the list of "Changed" files (marked by the red box).

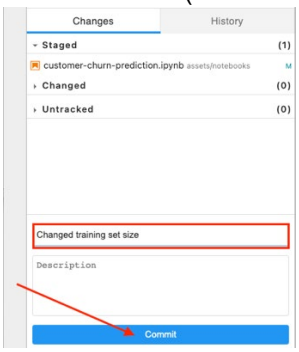


31- Click on the notebook and then on the "+"-sign (as marked by the red arrow) to stage the change for commit.

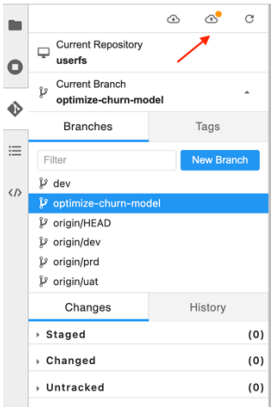


The notebook will now appear in the list of “Staged” files.

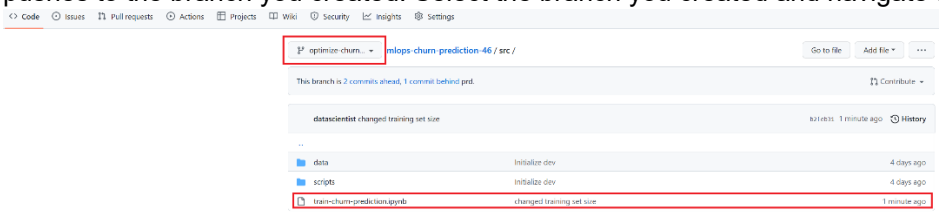
32- Enter a meaningful message in the “Summary” field (annotated by the red box) and click the Commit button (indicated by the red arrow).



33- The change is still only local to JupyterLab and to make it available in the Git repository, we need to synchronize. Click on the cloud button annotated by the red arrow to push the changes to the remote repository.

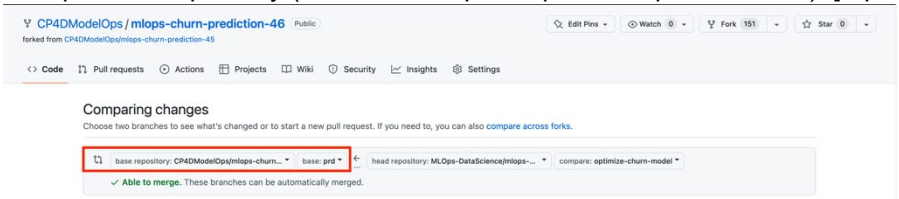


34- You can now go to your repository on GitHub, refresh the page and see that there were recent pushes to the branch you created. Select the branch you created and navigate to the **src** folder.

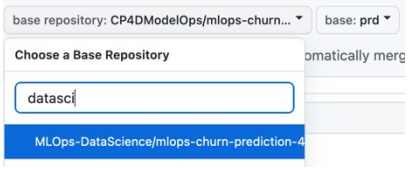


35- As you want the data science lead to review your changes, create a pull request. Click on **Pull Requests** in the top menu and then the **Compare & pull request** button to request your changes to be merged with the **dev** branch. Please pay attention to ensure you are creating a pull request for **your own** repository. You will find that GitHub assumes you want to create a pull request to

the upstream repository (CP4DModelOps/mlops-churn-prediction-46). [replace]

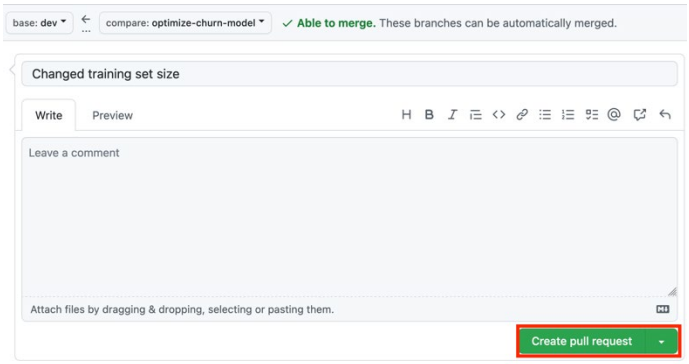


Because you want to merge the changes into the **dev** branch of your own repository, click on the "base repository" drop-down box as indicated by the red box and select your own repository.



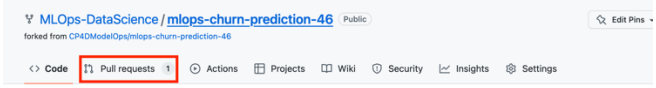
The upstream repository will disappear and now you can select the **dev** branch by clicking the drop-down box.

36- GitHub indicates that it is able to merge the changes automatically (Able to merge). Click the **Create pull request** button (annotated with a red box) to request the changes to be merged into the dev branch.

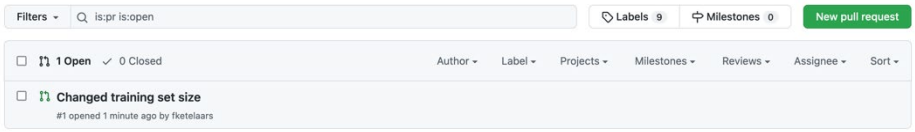


37- Now, assume that the Data Scientist lead has received an e-mail that a pull request is ready to be reviewed. You will take the role of the lead data scientist now.

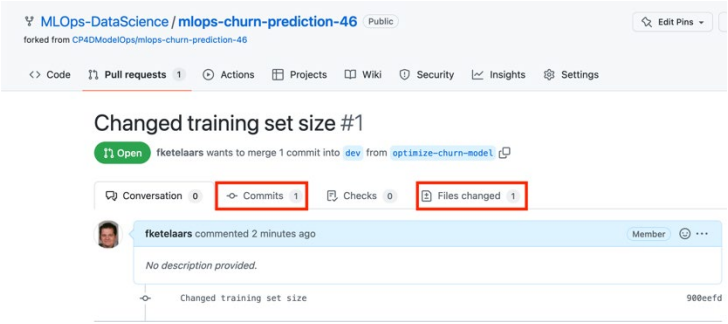
38- Click on the **Pull requests** link at the top (as indicated by the red box).



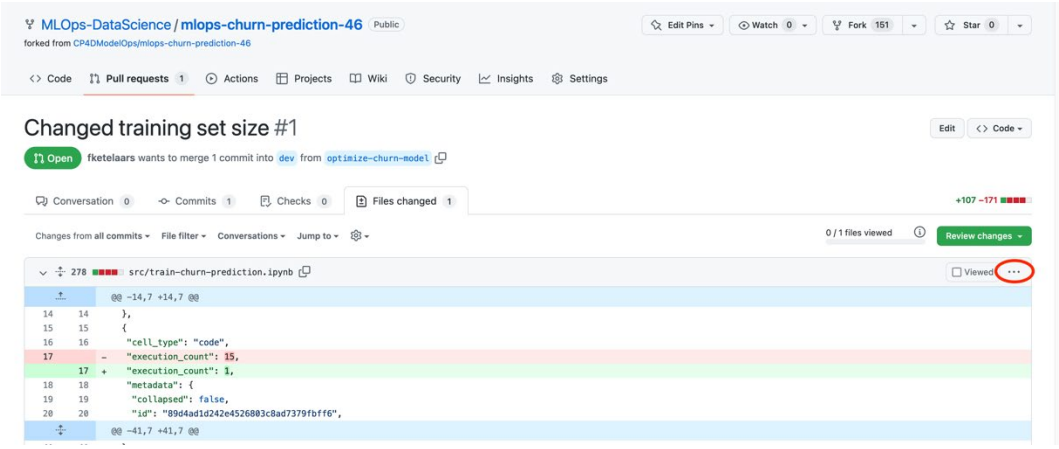
You will see a list of all (one) pull requests.



39- Select the pull request you created as the data scientist user by clicking the title. You will see that the request consists of 1 commit and only 1 file was changed (indicated by the red boxes).



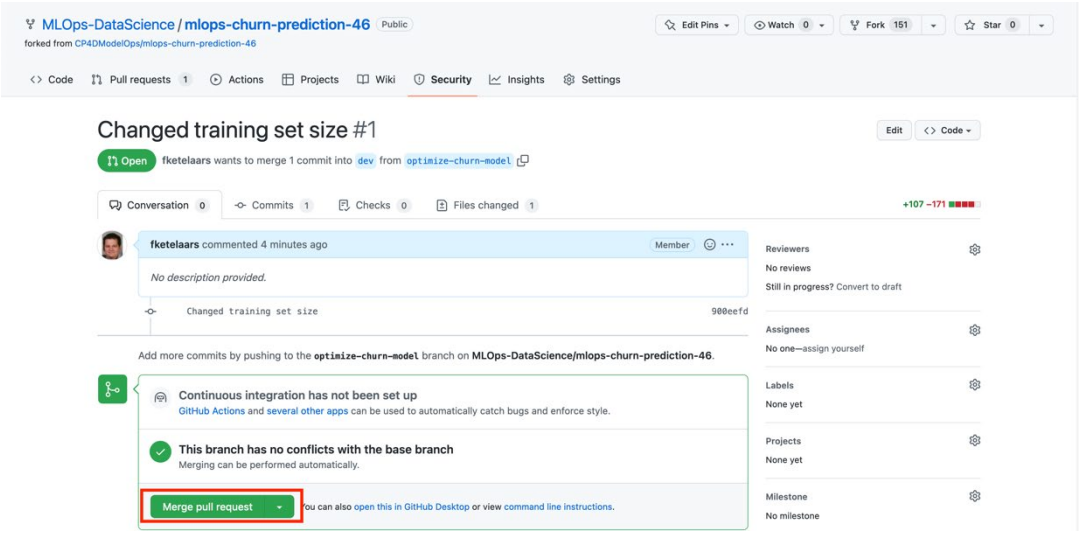
40- Click on the **Files changed** box to see the details of what was changed



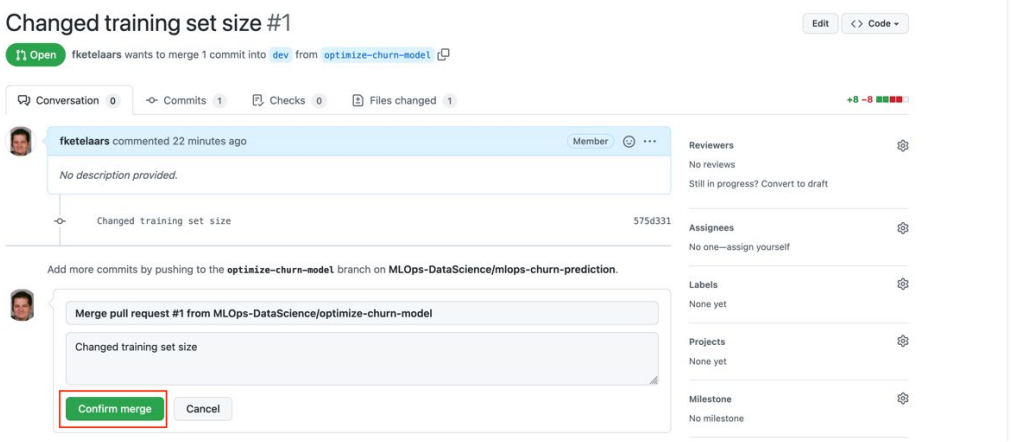
The file is not nicely formatted as a notebook but it is quite easy to identify the changes that have been made. If wanted you can click on the ellipses indicated by the red oval and select **View file** to display the file in a notebook format.

41- The data scientist lead is happy with the changes and will now approve and merge the changes to the **dev** branch. As we only have one user for this repository, we cannot approve our own pull request. Normally, the lead data scientist would use the **Review changes** button to approve the request. Click the browser's "back" button to go to the pull request and click on the **Conversation** tab.

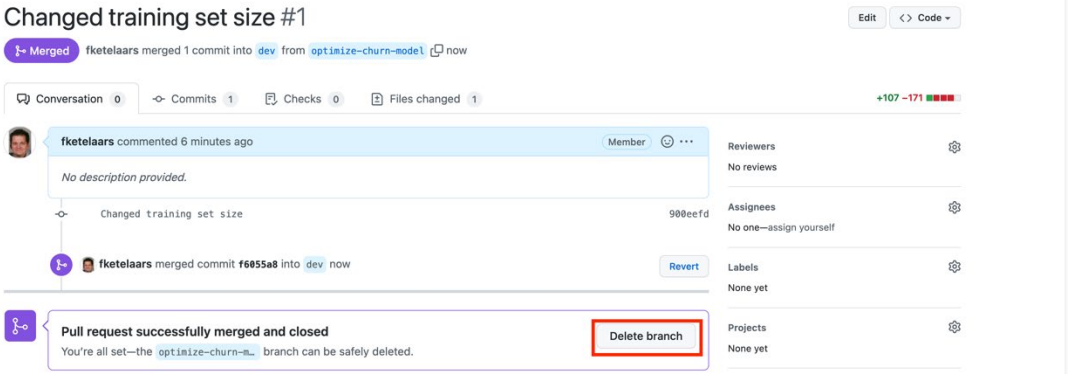
42- Click the **Merge pull request** button to merge the changes with the **dev** branch.



Then, click **Confirm merge**.

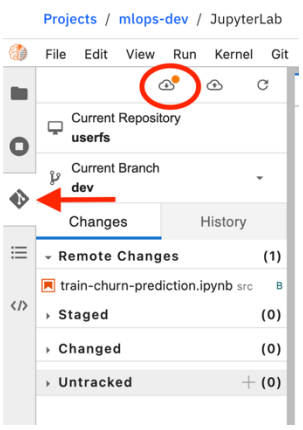


GitHub confirms that the pull request has been successfully merged and closed. As a good practice to keep the repository clean, delete the branch by clicking the **Delete branch** button.



43- Go to your Cloud Pak for Data window. Log off as the datascientist user and log in as the data scientist lead (**dslead** user) and navigate to the **mlops-dev** project. Open it and start JupyterLab like you did before.

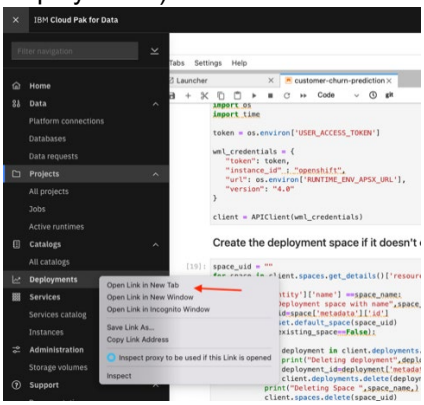
44- Go to the Git space in JupyterLab as indicated by the arrow. You will notice that the cloud icon with the down arrow (marked by the red oval) has an amber dot next to it. This indicates that there are changes in the remote repository that can be pulled.



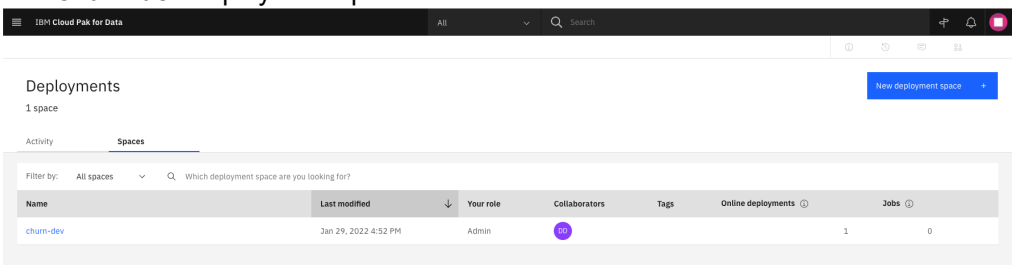
Click the cloud icon. The changes will be pulled and the amber dot disappears. The latest commit that was done by the datascientist user has now been downloaded.

45- Navigate to the **src → train-churn-prediction.ipynb** and review the change that was done by the datascientist user (verify you see the change in test-size). Run the entire notebook as the current **dslead** user by clicking **Run → Run All Cells**. The data science lead notices that the accuracy of the model has improved and is happy with the change.

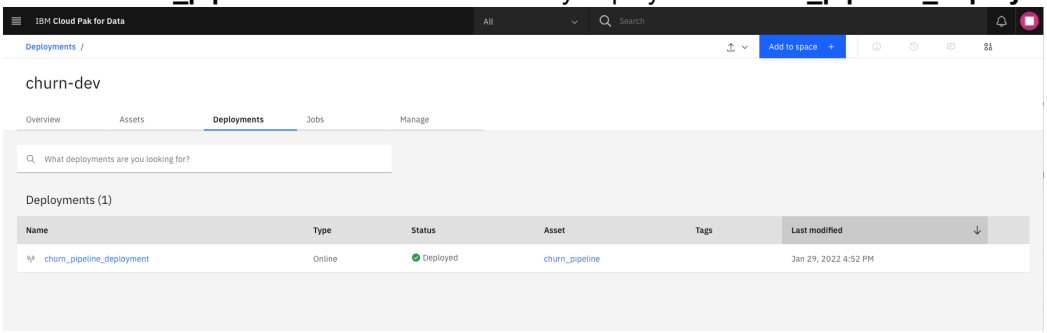
46- Also scroll down to the bottom of the notebook and observe that the notebook was successfully deployed to the **churn-dev** deployment space. You can double check this by clicking on the hamburger menu at the top left and then open the **Deployments** page in a new tab (right click on Deployments).



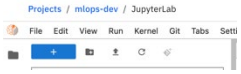
The **churn-dev** deployment space will be listed.



47- Click on the **churn-dev** deployment space and navigate to the **Deployments** tab. You can see that the **churn_pipeline** model was successfully deployed as **churn_pipeline_deployment**.

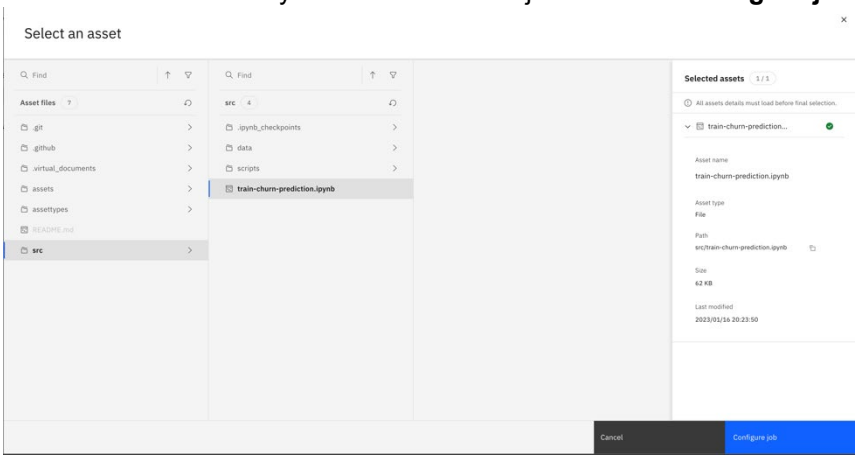


48- As the data scientist only did the analysis and is not responsible for deployment, the lead data scientist now creates a job that will take care of automated deployment in the subsequent environments, **uat** and **prd**. Go back to the JupyterLab tab and click on the **mlops-dev** breadcrumb.

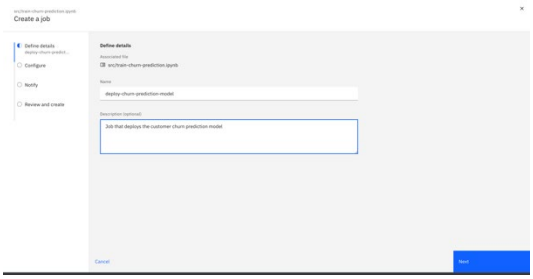


49- Now that you have returned to the project, click on the **Assets** link and then on **View local branch**.

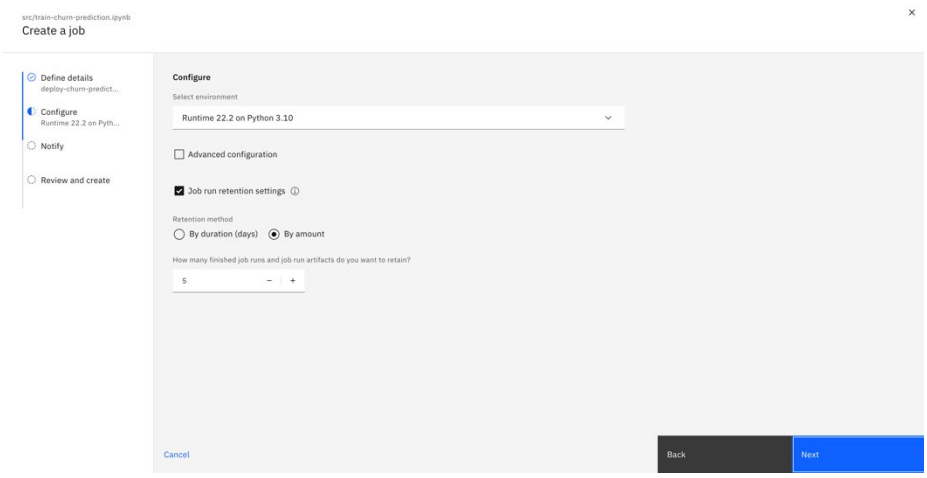
50- We will create a new job by clicking the **New code job** button at the top right. Then, navigate to the notebook for which you want to create a job and click **Configure job**.



51- Choose the name for your job. To ensure the job is picked up by the GitHub Actions automation, it must start with “deploy”. Enter a meaningful name for your job, for example **deploy-churn-prediction-model** and optionally type a description of the job (Job that deploys the customer churn prediction model). Then click **Next**.



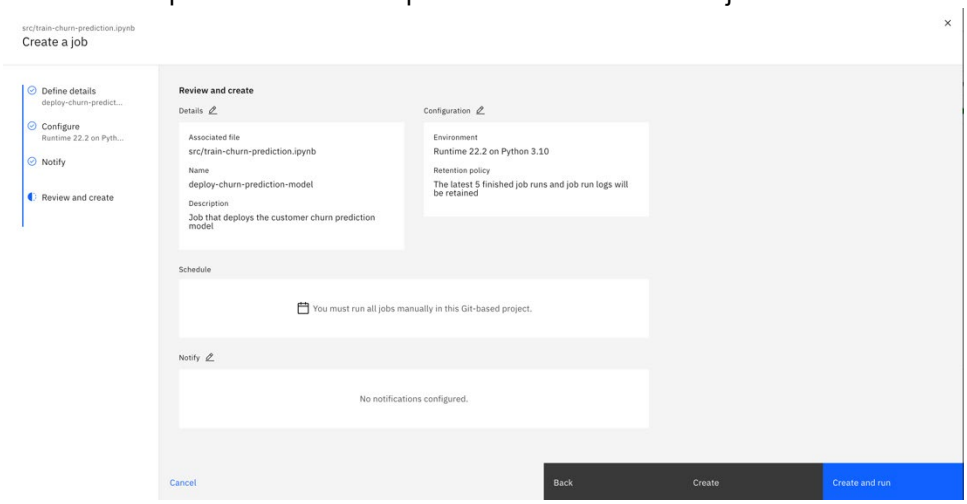
52- Choose the **IBM Runtime 22.2 on Python 3.10** environment.



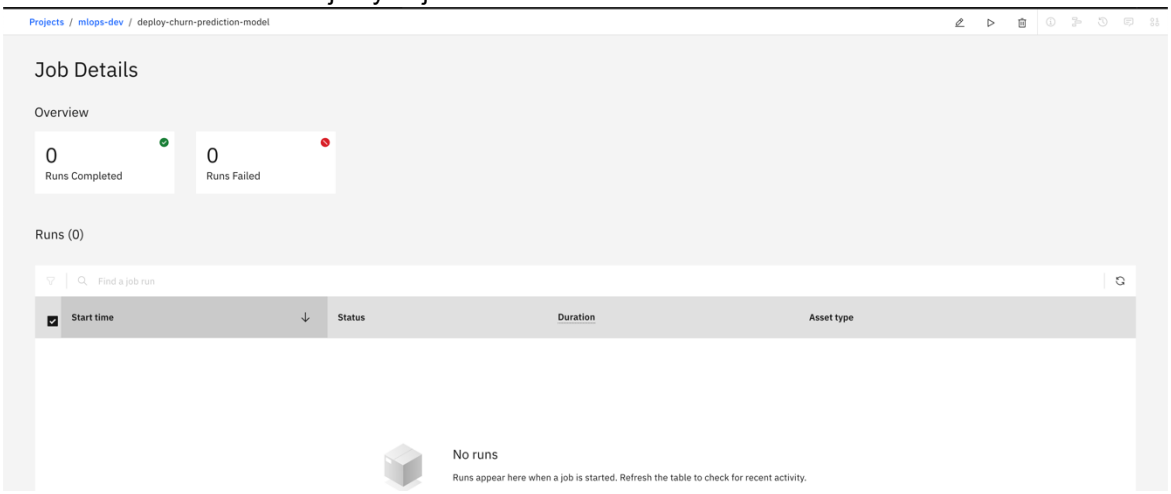
Optionally, you can open the Advanced configuration to pass additional environment variables to the job. The GitHub Actions flow will pass the branch as an environment variable to the job to specify the environment (deployment space) the model must be deployed into; there is no need specify any environment v

Click **Next** to continue with the notifications. we will skip notifications for this job. Click **Next** again.

53- In the final step we will review the parameters and create the job.



There is no need to run the job at this moment but you can choose either **Create** or **Create and run** as you prefer. You will proceeed to the **Job Details** page where you can view job runs and other information about the job you just created.

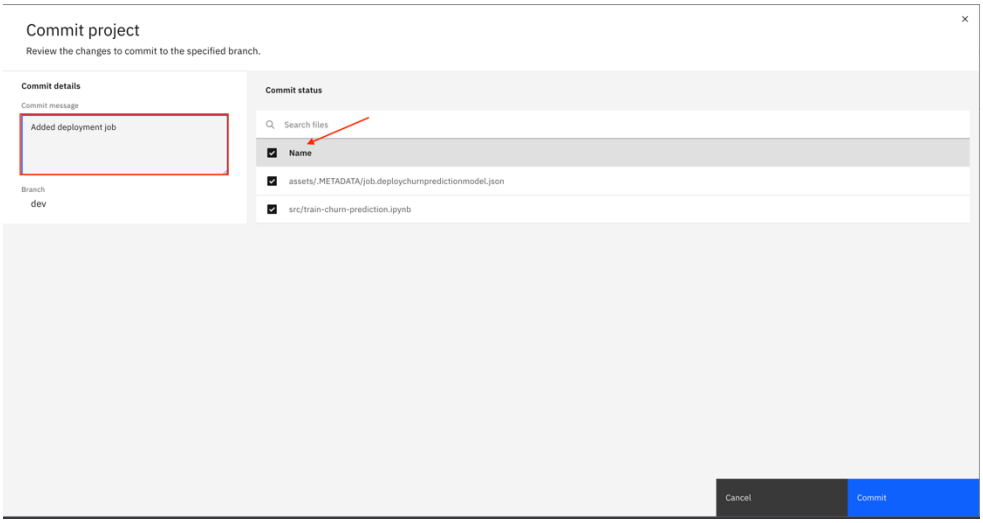


54- The lead data scientist now wants to promote the notebook and job to the next stage. First, the new job definition (which is effectively a JSON file) must be committed to the **dev** branch. Click on the breadcrumb of the **mlops-dev** project.

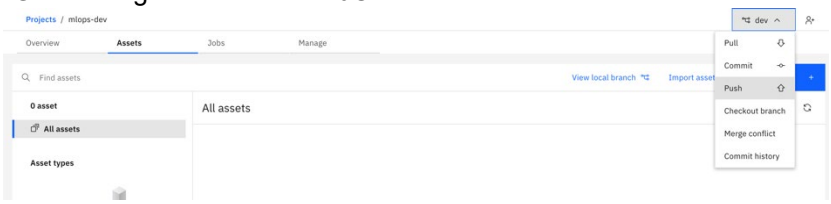
55- Now we will commit the changes to the **dev** branch. Click on the "Git" icon at the top and open the drop-down menu, then click **Commit**.

56- A new page will open where you can select the files to be included in the commit and a message indicating what was changed. Select all files by checking the checkbox at the top of the list as indicated by the arrow and enter a meaningful message (Added deployment job). Then click

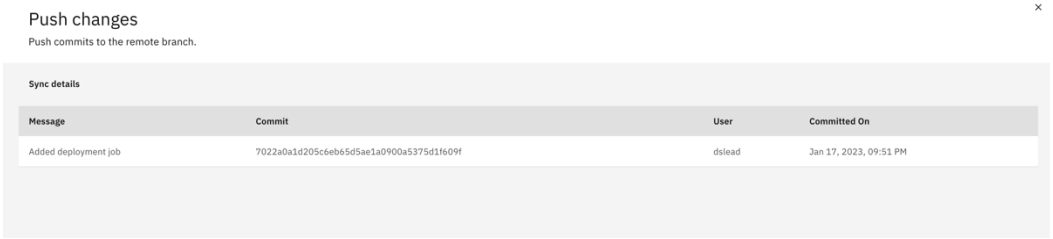
Commit.



57- The changes have not yet been synchronized with the Git repository. For this you must click the "Git" icon again and select **Push**.

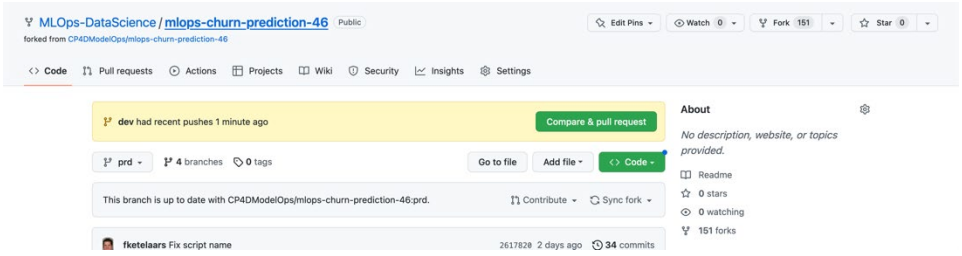


All commits you have done will be listed. In this case you have only performed 1 commit action so the list is pretty short.

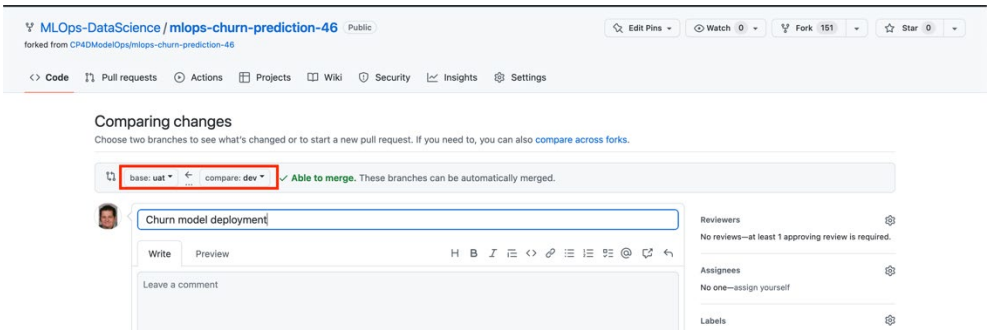


Confirm that your commit is listed and click the **Push** button.

58- Go to the GitHub repository in your browser. You will notice a message indicating that the **dev** branch had recent pushes.

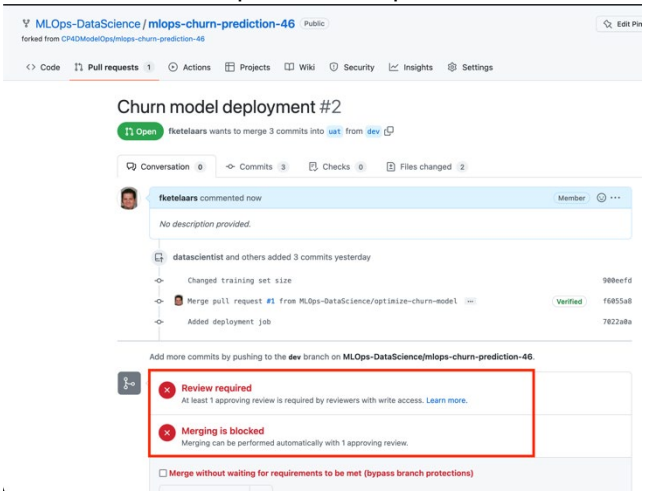


59- As before you can now create a pull request to merge changes with the **uat** branch. Please note that this would normally be done by the lead data scientist. Click the **Compare & pull request** button, change the left repository to your own repository and select the **uat** branch. Your screen should look something like below.

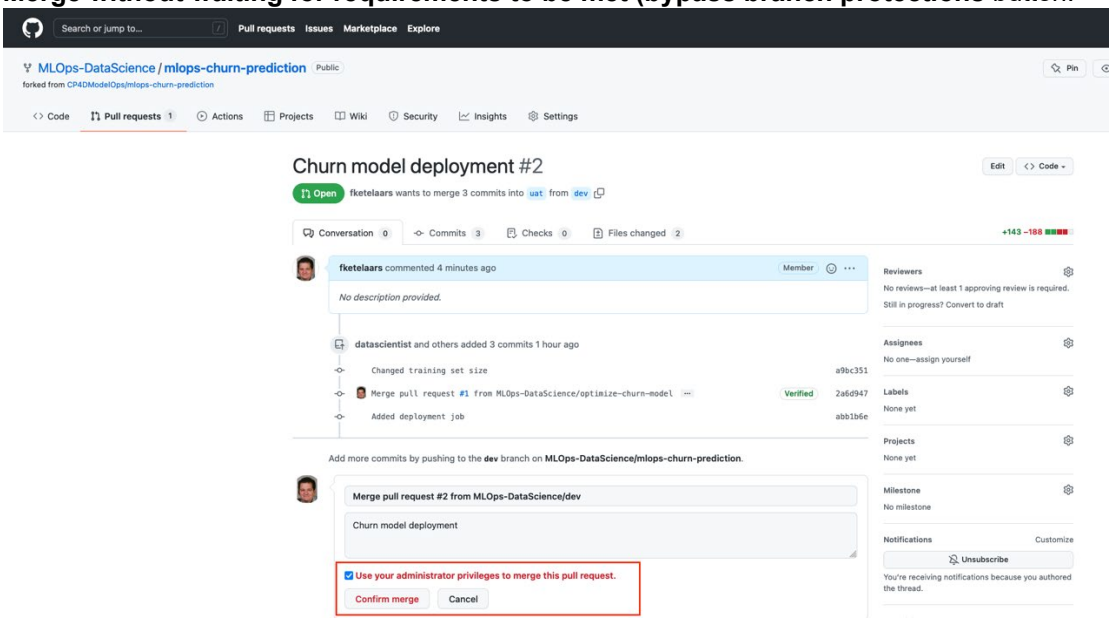


Enter a meaningful description (Churn model deployment) of the pull request and optionally write a longer explanation of the changes that will be merged. Then click **Create pull request**.

60- You will observe that the merging is blocked because it requires at least 1 approving review. This is because we set up the branch protection rules for the **uat** branch.

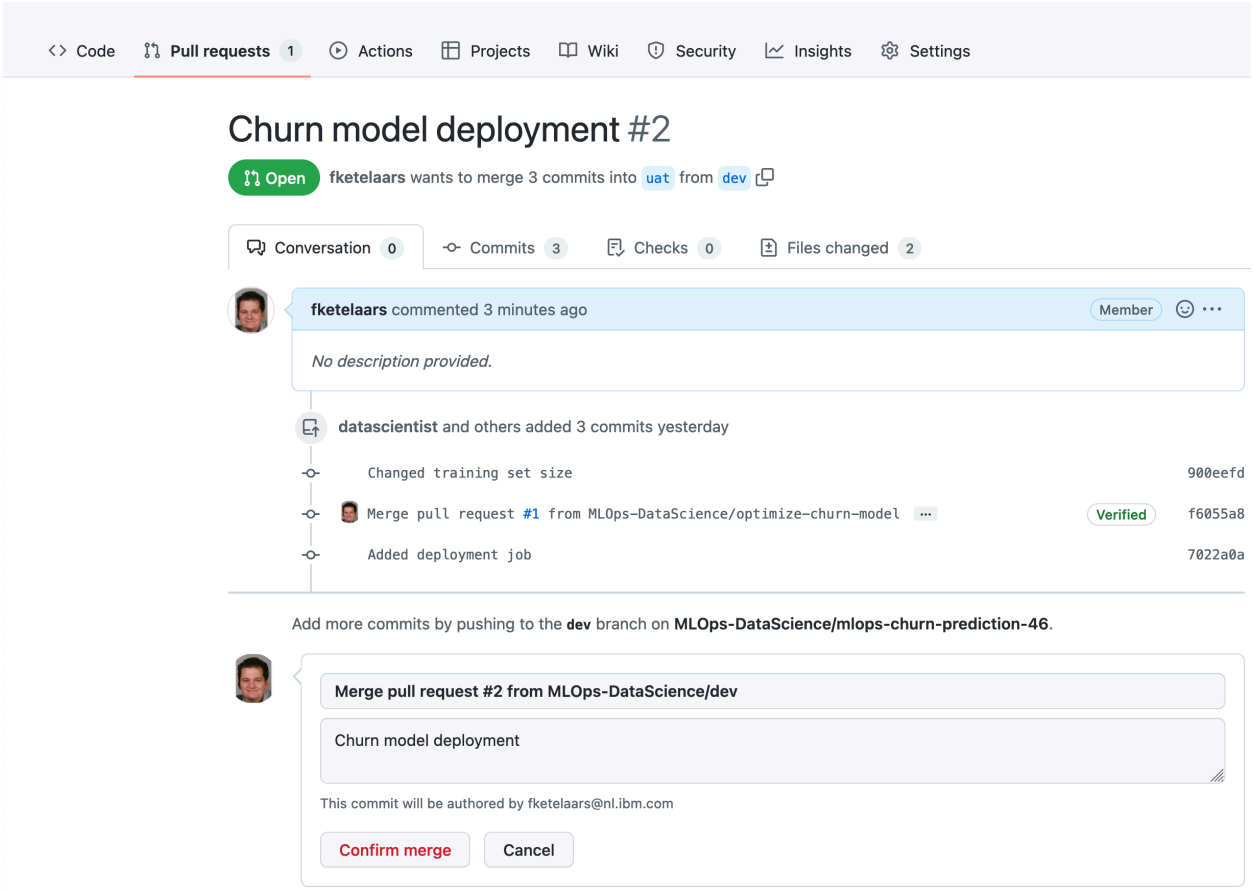


61- As we only have a single user in the GitHub repository, a formal approval process cannot be demonstrated. For now, we will force the merge by using the administrator privileges. Check the **Merge without waiting for requirements to be met (bypass branch protections** button.



Make sure the checkbox is ticked and then click the **Merge pull request** button, followed by

Confirm merge.

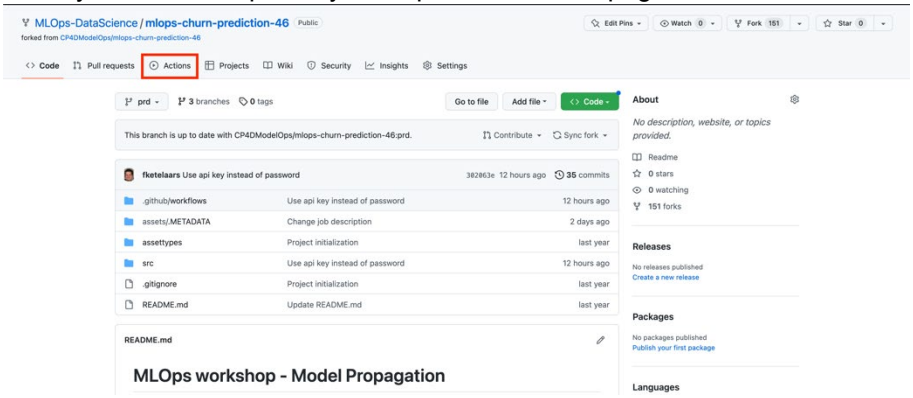


62- The changes are merged, and you will be asked if you want to delete the **dev** branch. In our scenario, we would keep the dev branch in place because other team members will be working in that branch. In a real implementation, you can prevent unintentional deletion of certain branches by setting authorizations in GitHub.

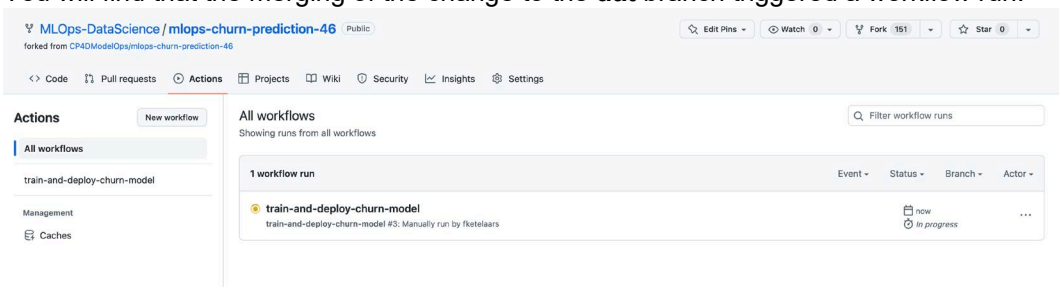
Review pipeline run

Because we have set up GitHub Actions for the repository, the pushing of the changes to the uat branch triggers the run of a workflow, effectively the Continuous Deployment part of the exercise.

1- Go to your GitHub repository and open the Actions page

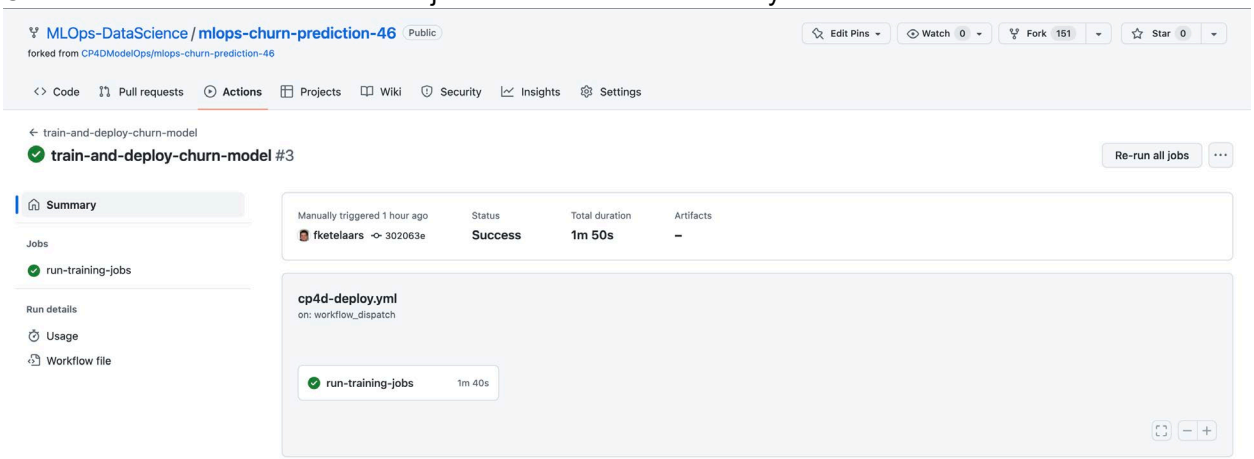


2- You will find that the merging of the change to the **uat** branch triggered a workflow run.

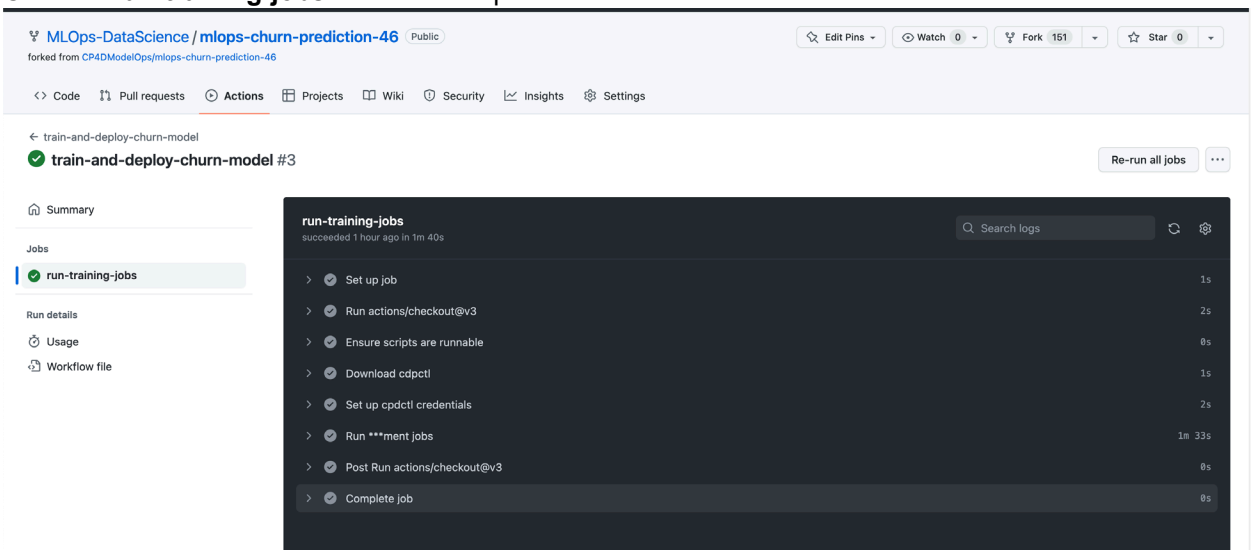


The workflow will finish in just a few minutes so by the time you get to this page, it may already be complete.

3- Click on the workflow run to see the jobs. In this case there is only one.



4- Click the **run-training-jobs** to see the steps it executed.



5- Open the **Run …** step and observe that a temporary deployment space is created (mlops-churn-prediction-46-uat-20230118093253 in the example below). The Git repository holding the notebook and job definition is uploaded as a "code package" to the deployment space and then the notebook is run.

The notebook creates the **churn-uat** deployment space if it doesn't exist yet and then deploys the model as an online deployment.

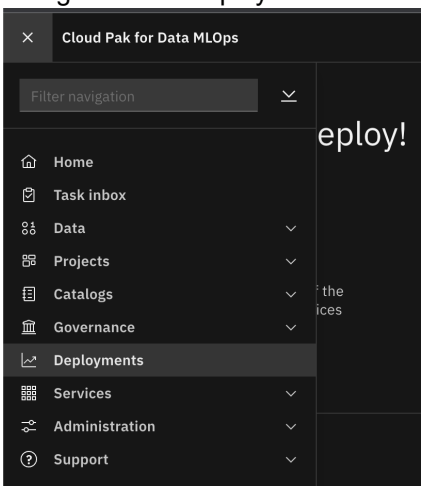
When the notebook has completed successfully, the temporary deployment space is automatically removed; when failing the space is not removed so one can check the output of the notebook cells.

Check the deployed model(s)

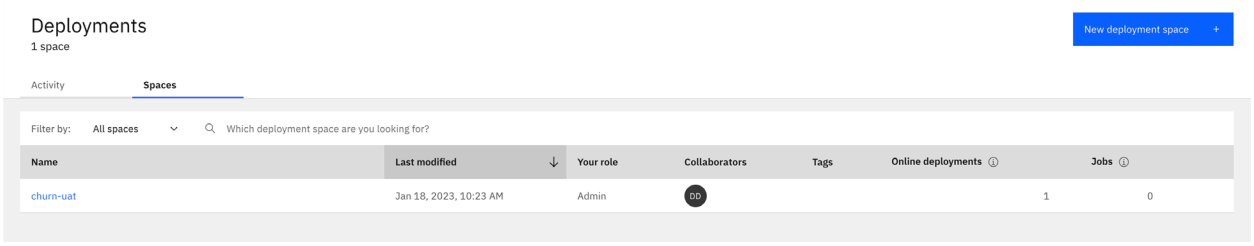
In the previous section, the pipeline has deployed the model into the **churn-uat** deployment space. We will now check that the model is there.

1- Login to Cloud Pak for Data as the **deploy** user; this is the service account user that was used in the automation. In normal circumstances you would not have to login with this user so the below steps are mainly to confirm that the automation has worked.

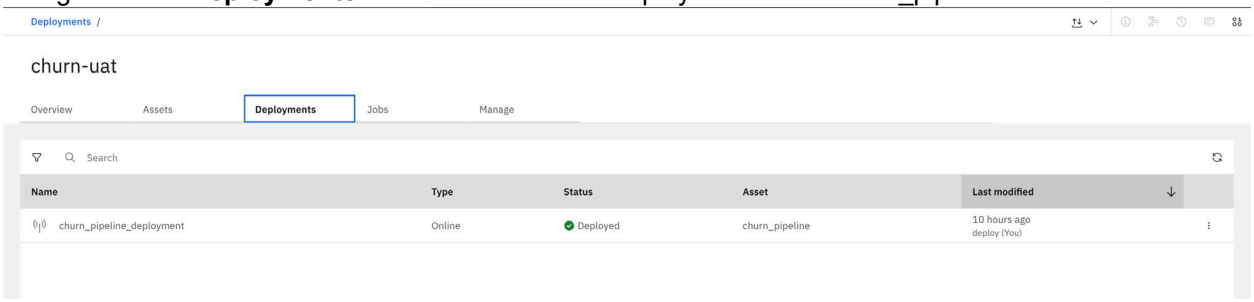
2- Navigate to the deployments via the hamburger menu.



3- Observe that there a deployment space called **churn-uat.** Click on the link to drill down into the space.



4- Navigate to the **Deployments** tab to see the online deployment of the churn_pipeline model.



5- The space and deployed model has been created using the GitHub Actions automation. The job log is kept as part of the pipeline run and once the model has been deployed, the project could be deleted.

This concludes the Git portion of the lab. Optionally you can continue to push the changes to the **prd** branch using the same method as above and then also deploy the model in the churn-prd deployment space.

Summary

To quickly re-cap, we've illustrated two common approached for implemented CI/CD for AI models using cpdctl utility as well as a git-based approach using git actions. In practice, and depending on the organization's preferences, one or the other or a custom combination of both approaches can deliver CI/CD functionality.