Contents

POC: MLOPS

Problem Statement

Calculate the progression of the diabetes of 442 patients, given 10 attributes that contribute to the predictions.

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline

It is a linear regression problem.

Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of n_samples (i.e., the sum of squares of each column totals 1).

Data Set Characteristics:

Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, total serum cholesterol • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, total cholesterol / HDL • s5 ltg, possibly log of serum triglycerides level • s6 glu, blood sugar level

Capabilities of the Pipeline

This pipeline contains steps to

- Convert the experimentation code to the azure-ml production code
- Creates and manages the azure-ml connection with azure devops.

This pipeline will:

trigger CI (continuous Integration) on each code commit, to validate the code for code quality, and runs unit tests and linting tests, and will trigger the CD (Continuous Deployment) to train, score, evaluate and register the model in .pkl format to the azure storage account.

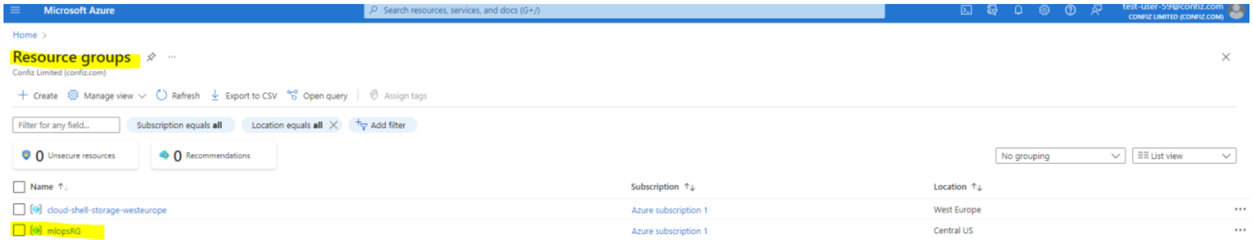
This pipeline does the Batch Scoring over a given set of batches and produces the results and saves them into the prediction.csv file into the azure-ml storage account.

Setting Up Azure Machine Leaning Workspace

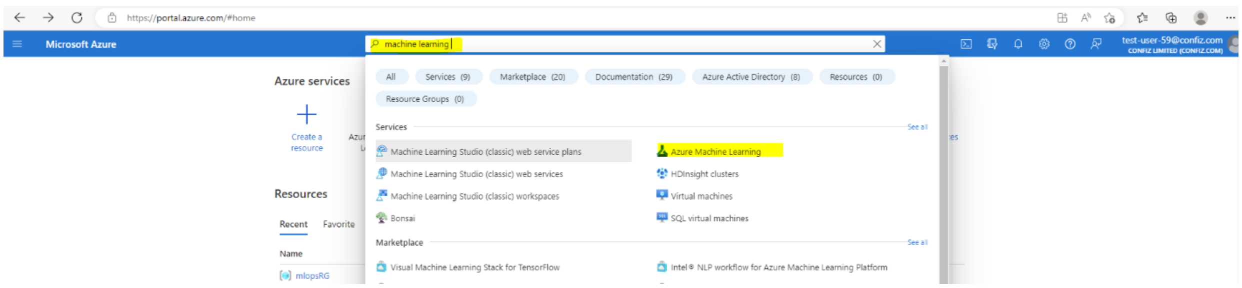
Open the Azure: https://portal.azure.com

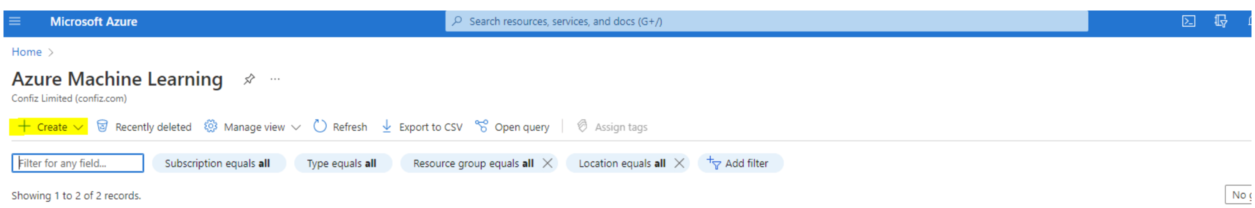
Register for a subscription of a free account, create your free account and access the portal home page.

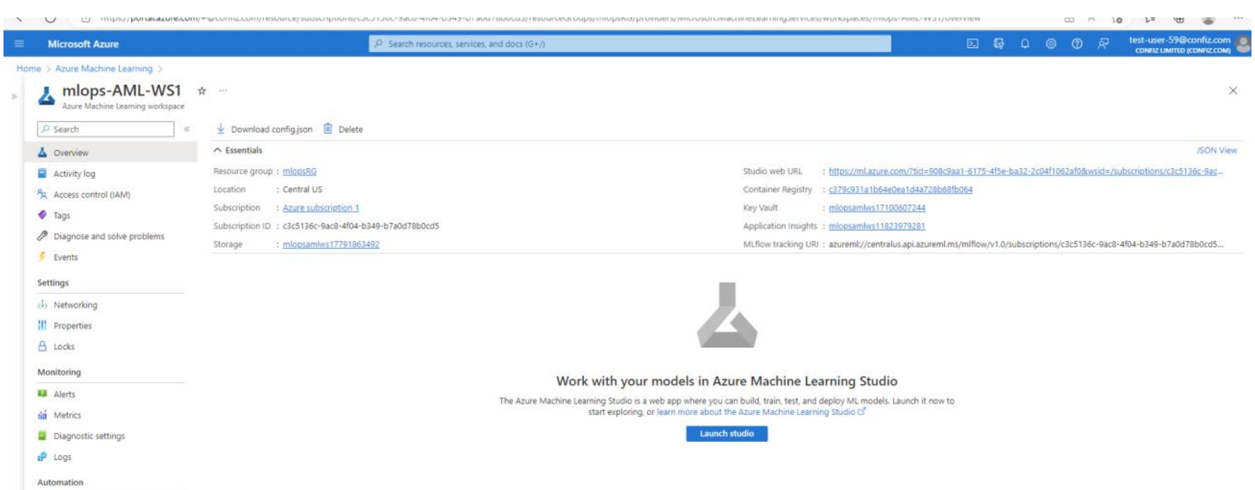
Create a resource group named “mlopsRG”, with this subscription, with the highlighted icon on the screen.



Create an “Azure Machine Learning Workspace” the same way named “mlops-AML-WS1”.





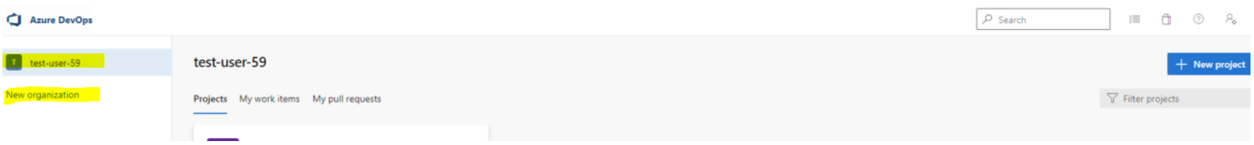


Setting Up Azure DevOps:

[MLOpsPython/getting_started.md at master · microsoft/MLOpsPython (github.com)]

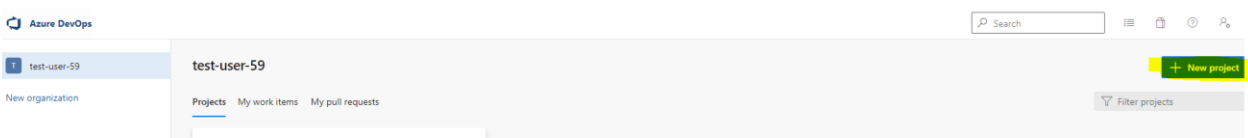
Create an azure devops organization

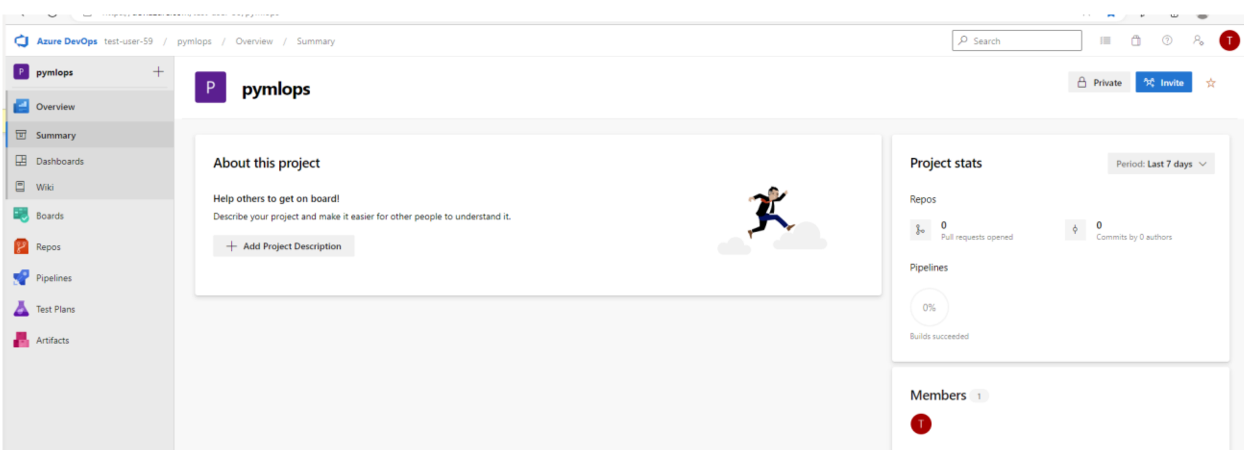
https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/create-organization?view=azure-devops



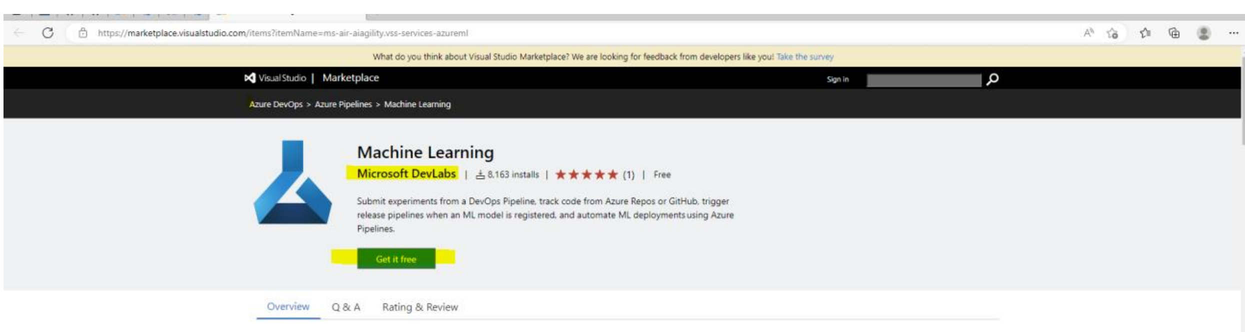
If you already have an organization then create a new project

https://docs.microsoft.com/en-us/azure/devops/organizations/projects/create-project?view=azure-devops



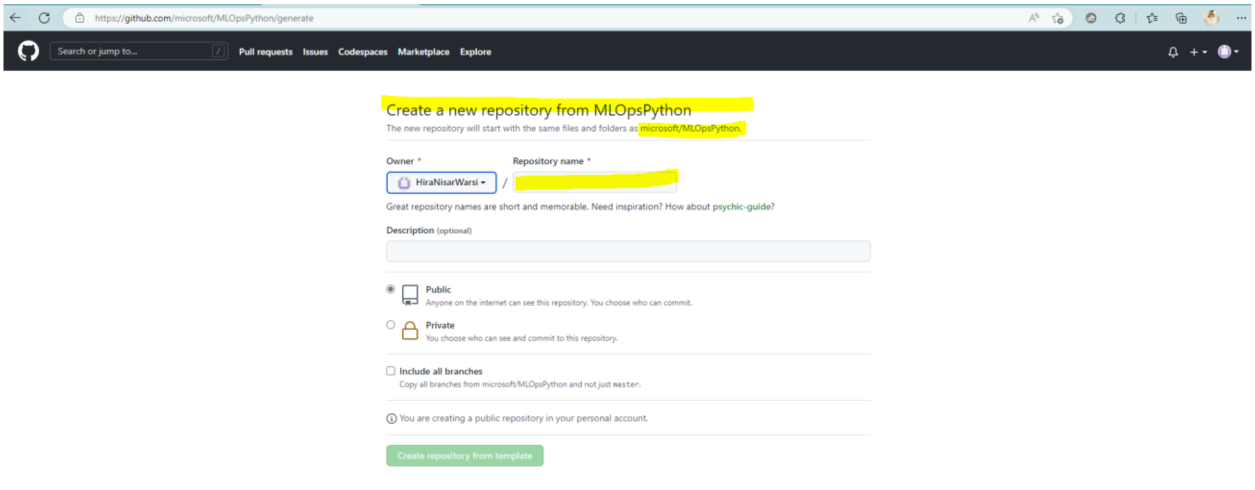


Install azure machine learning extension to azure devops organization, by clicking "get it free".



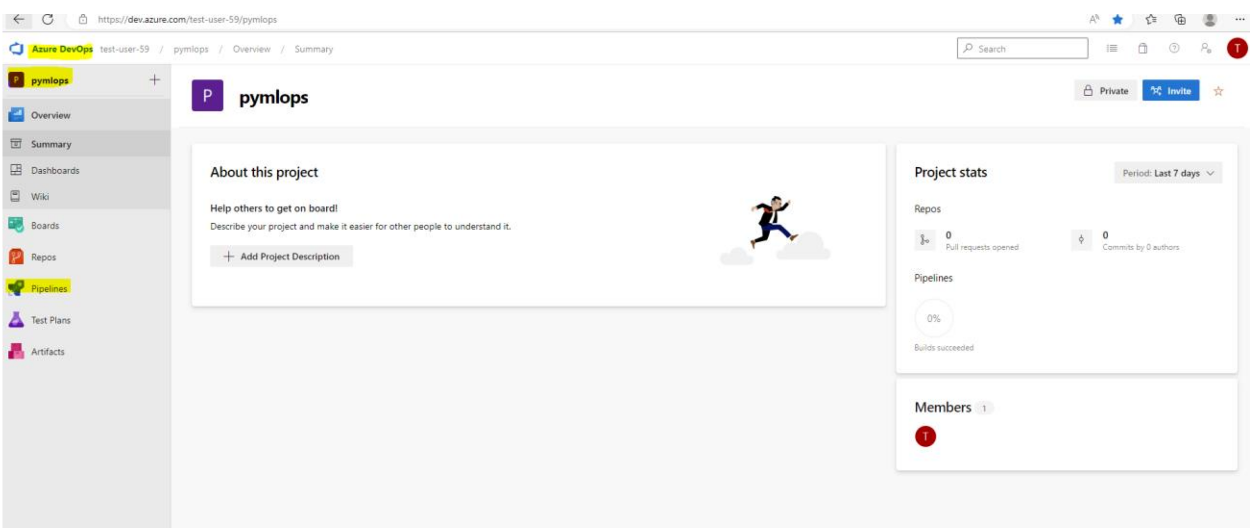
https://marketplace.visualstudio.com/items?itemName=ms-air-aiagility.vss-services-azureml

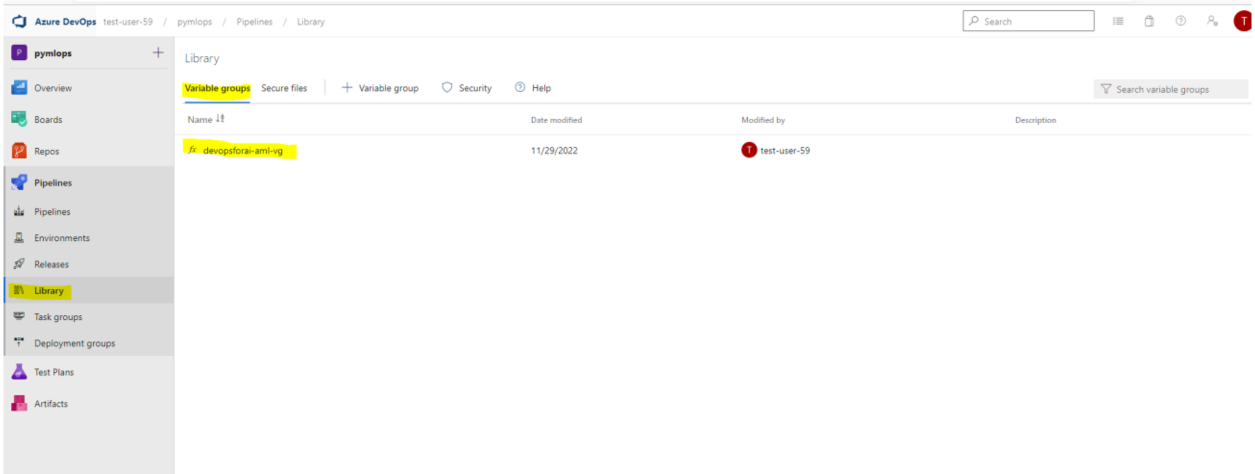
Get the code by using the *repository template* https://github.com/microsoft/MLOpsPython/generate



Create a variable group for your pipeline in azure devops.

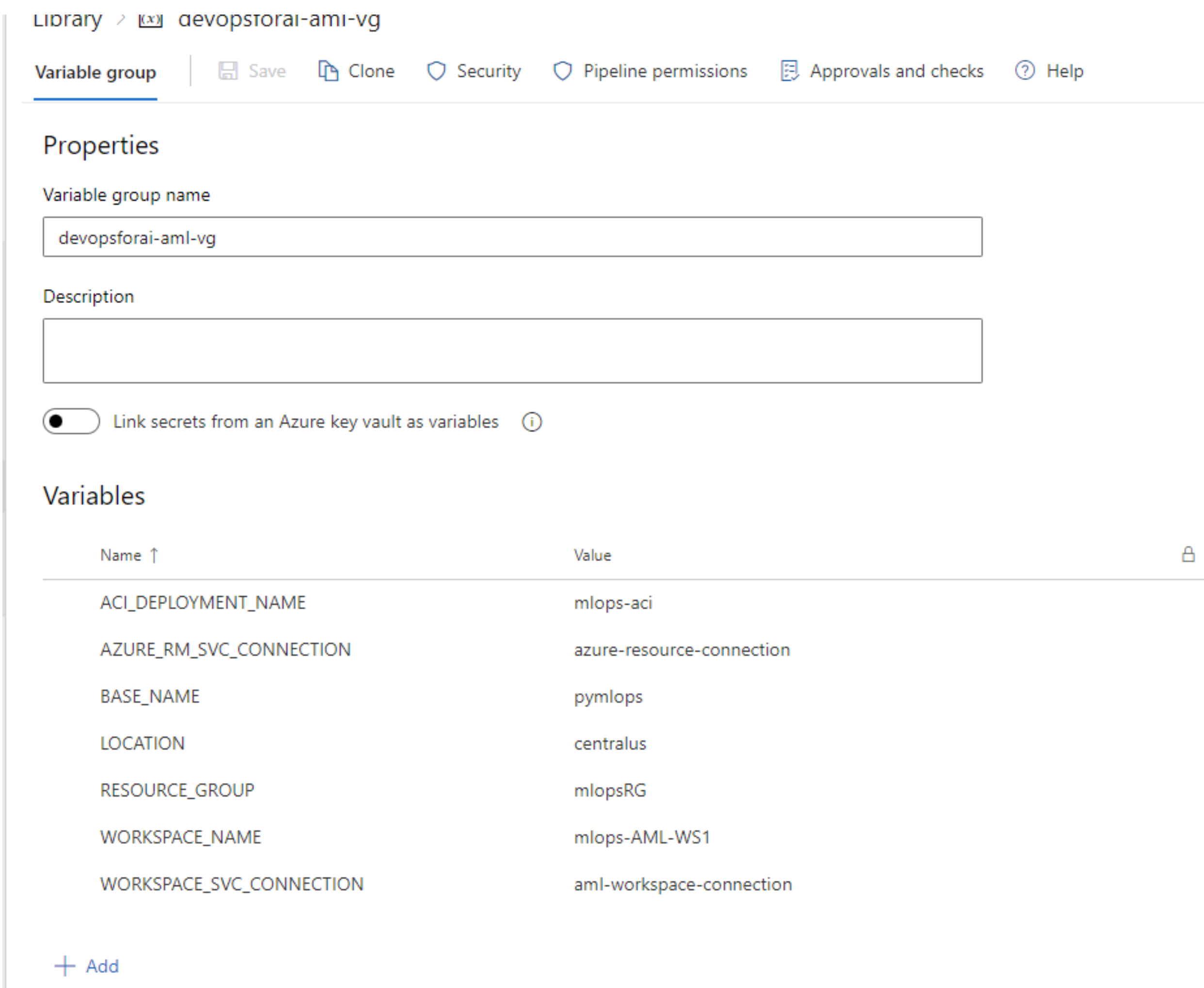
- In azure devops, navigate to ***library*** in the ***pipelines*** section, and create a variable group named **"devopsforai-aml-vg"**. The YAML pipeline definitions in this repository refer to this variable group by name.



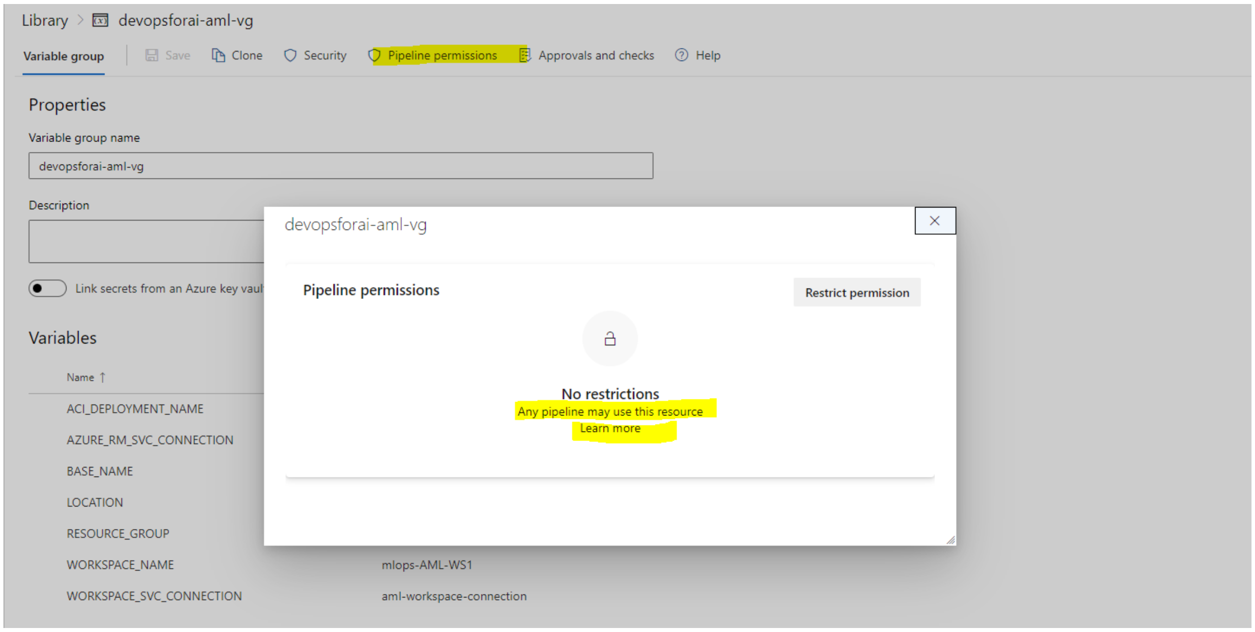


- The variable group should contain the following required variables.

Variable Name	Suggested Value	Short description
BASE_NAME	[your project name]	Unique naming prefix for created resources - max 10 chars, letters and numbers only
LOCATION	centralus	Azure location, no spaces. You can list all the region codes by running `az account list-locations -o table` in the Azure CLI
RESOURCE_GROUP	mlops-RG	Azure Resource Group name
WORKSPACE_NAME	mlops-AML-WS	Azure ML Workspace name
AZURE_RM_SVC_CONNECTION	azure-resource-connection	Azure Resource Manager Service Connection name
WORKSPACE_SVC_CONNECTION	aml-workspace-connection	Azure ML Workspace Service Connection name
ACI_DEPLOYMENT_NAME	mlops-aci	Azure Container Instances name

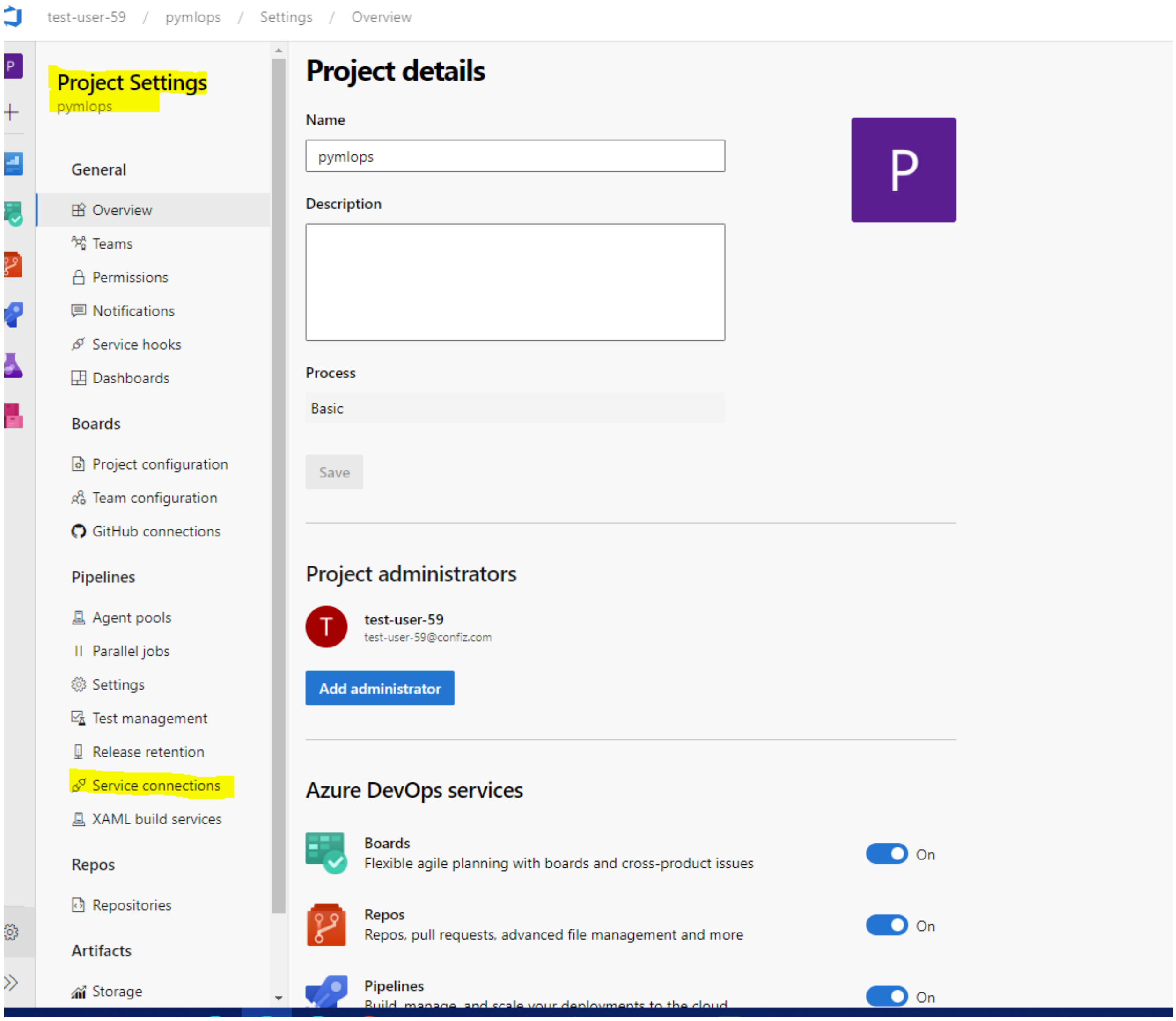


- Make sure you select the **Allow access to all pipelines** checkbox in the variable group configuration. To do this, first **Save** the variable group, then click **Pipeline Permissions**, then the button with 3 vertical dots, and then **Open access** button

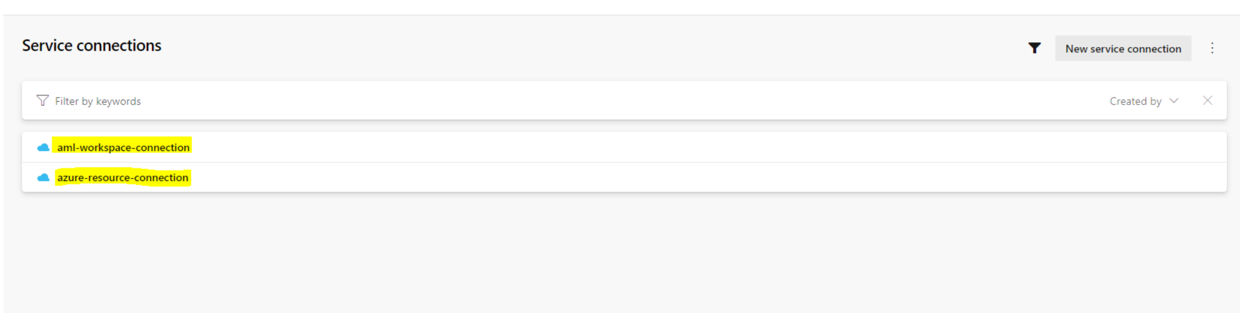


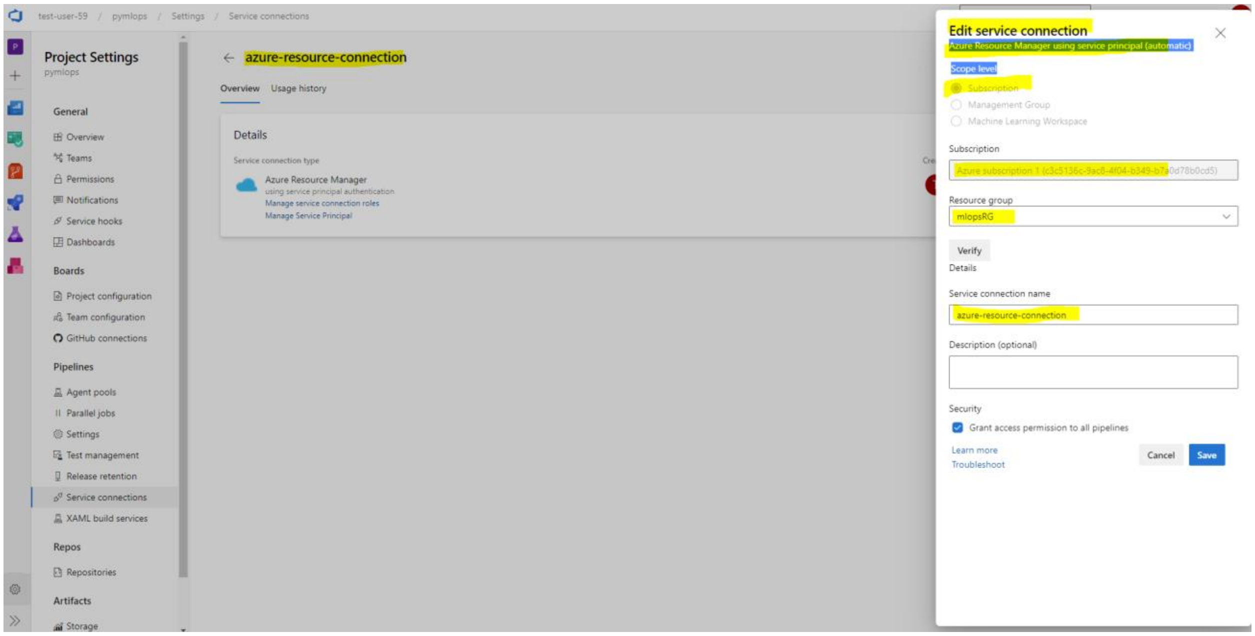
Create an Azure DevOps Service Connection for the Azure Resource Manager

The IaC provisioning pipeline requires an **Azure Resource Manager** service connection. To create one, in Azure DevOps select **Project Settings**, then **Service Connections**, and create a new one, where:



- Type is **Azure Resource Manager**
- Authentication method is **Service principal (automatic)**
- Scope level is **Subscription**
- Leave **Resource Group** empty after selecting your subscription in the dropdown
- Use the same **Service Connection Name** that you used in the variable group you created
- Select **Grant access permission to all pipelines**

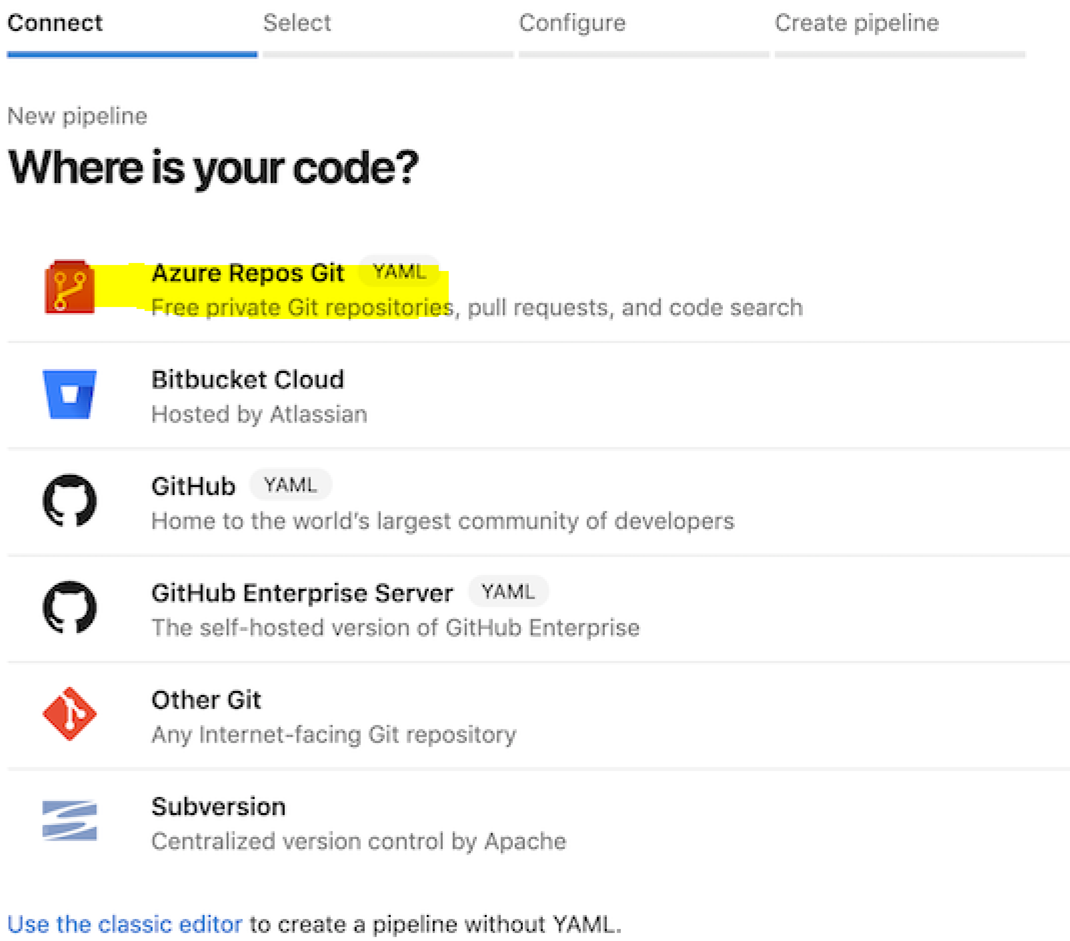


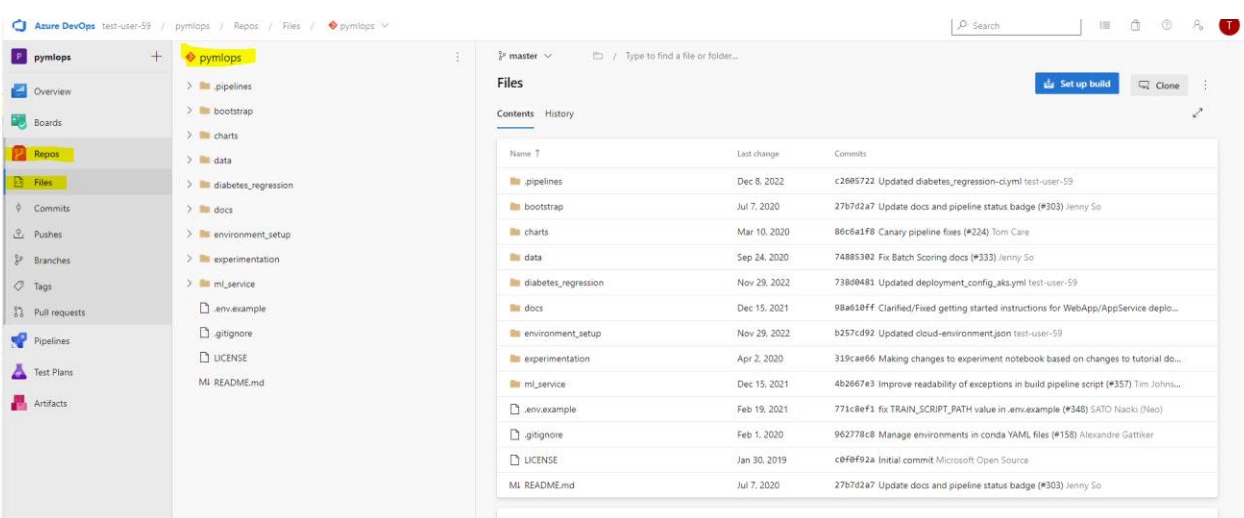


Create the IaC pipeline

Follow: https://github.com/microsoft/MLOpsPython/blob/master/docs/getting_started.md#create-the-iac-pipeline

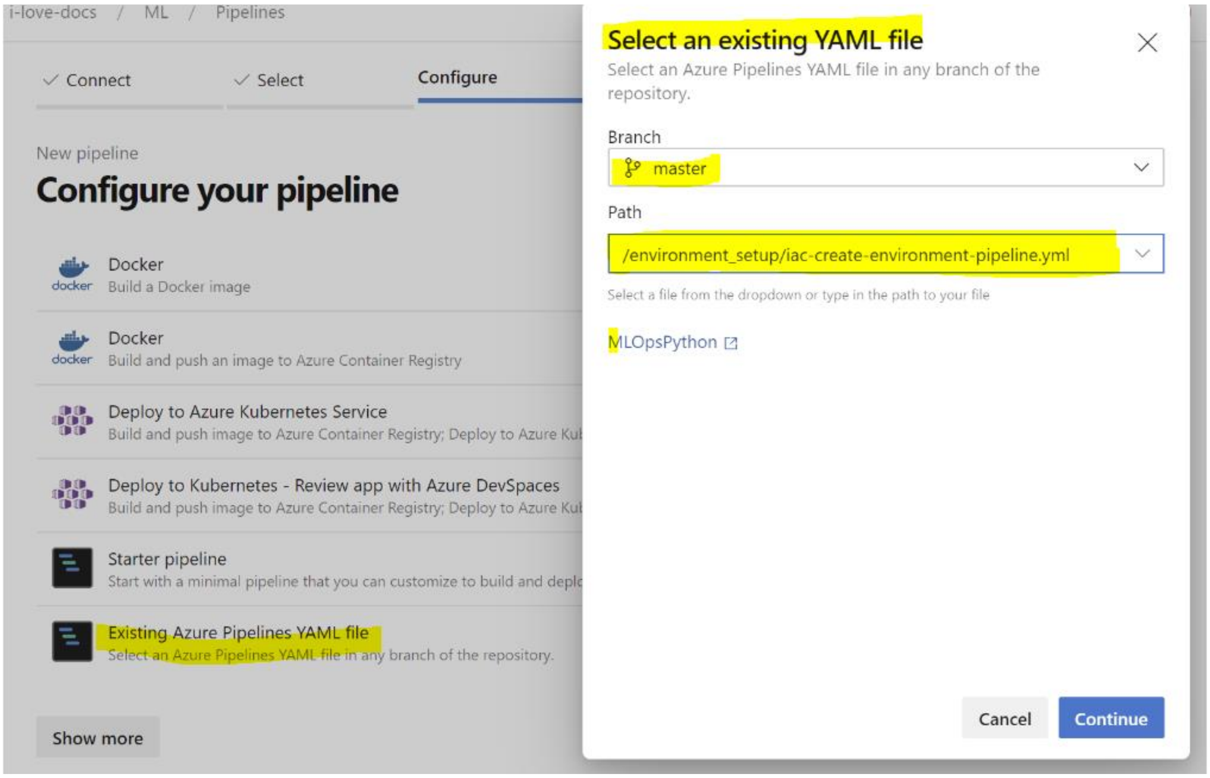
In your Azure DevOps project, create a build pipeline from your forked repository:



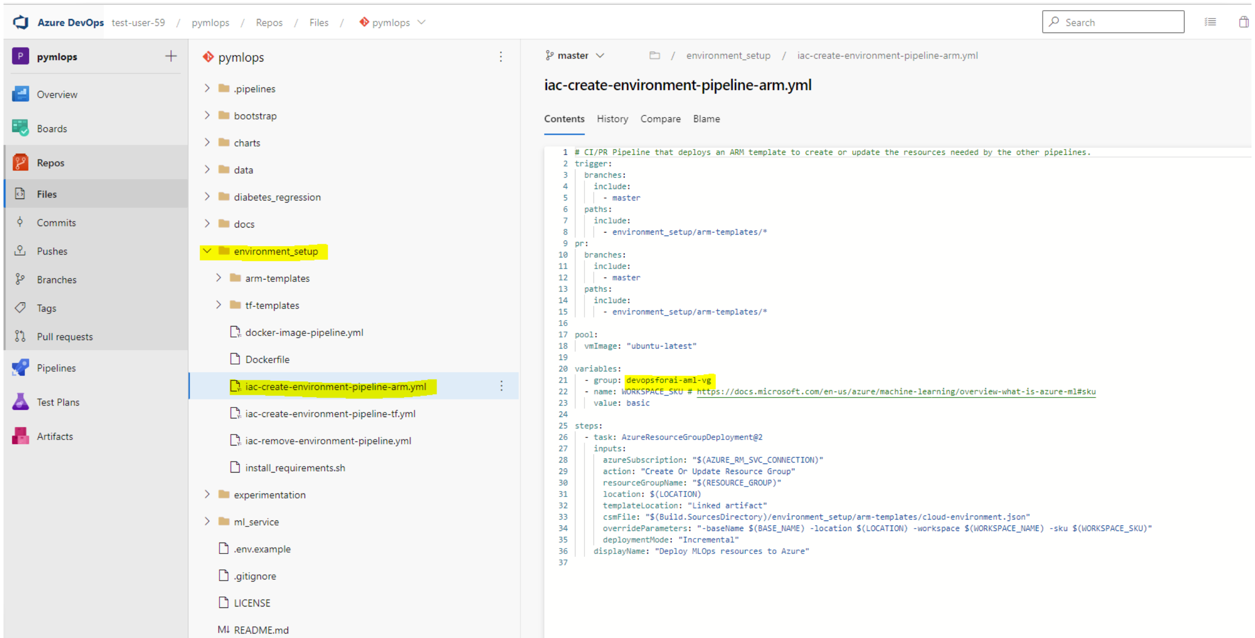


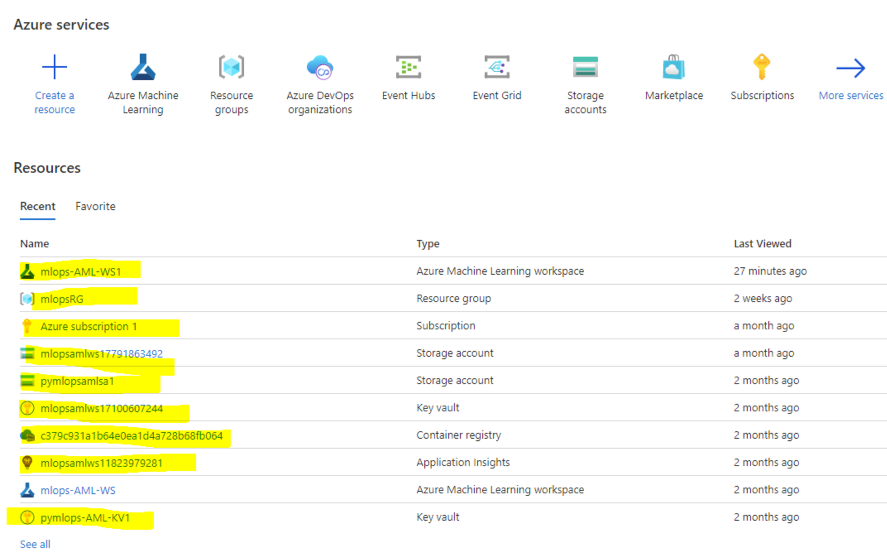
If you are using GitHub, after picking the option above, you'll be asked to authorize GitHub and select the repo you forked. Then you'll have to select your forked repository on GitHub under the **Repository Access** section and click **Approve and Install.**

After the above, and when you're redirected back to Azure DevOps, select the Existing Azure Pipelines YAML file option and set the path to ***/environment_setup/iac-create-environment-pipeline-arm.yml*** or to ***/environment_setup/iac-create-environment-pipeline-tf.yml***, depending on if you want to deploy your infrastructure using ARM templates or Terraform:



This will create the resources from the variable group (azure devops/pipelines/library) to azure machine learning workspace in Azure.



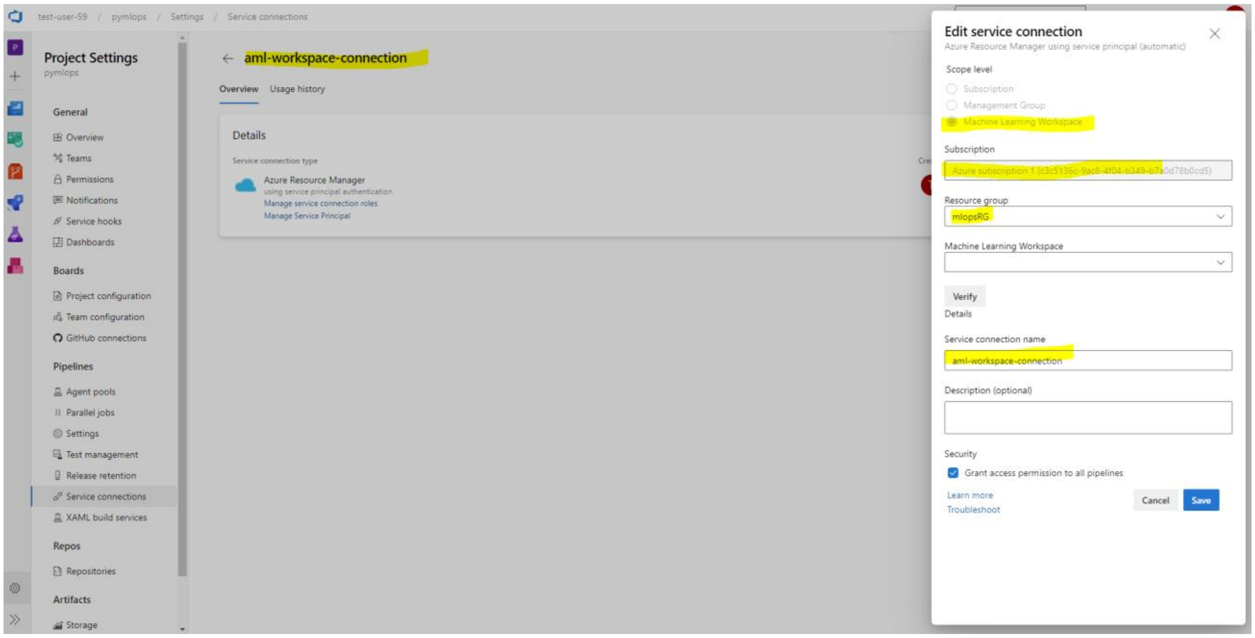


Note: If you have other errors, one good thing to check is what you used in the variable names. If you end up running the pipeline multiple times, you may also run into errors and need to delete the Azure services and re-run the pipeline -- this should include a resource group, a KeyVault, a Storage Account, a Container Registry, an Application Insights and a Machine Learning workspace.

Create an Azure DevOps Service Connection for the Azure ML Workspace

At this point, you should have an Azure ML Workspace created. Similar to the Azure Resource Manager service connection, you need to create an additional one for the Azure ML Workspace.

Create a new service connection to your Azure ML Workspace using the Machine Learning Extension instructions to enable executing the Azure ML training pipeline. The connection name needs to match WORKSPACE_SVC_CONNECTION that you set in the variable group above (e.g., 'aml-workspace-connection').



Note: Similar to the Azure Resource Manager service connection you created earlier, creating a service connection with Azure Machine Learning workspace scope requires 'Owner' or 'User Access Administrator' permissions on the Workspace. You'll need sufficient permission to register an application with your Azure AD tenant, or you can get the ID and secret of a service principal from your Azure AD Administrator. That principal must have Contributor permissions on the Azure ML Workspace.

Set up Build, Release Trigger, and Release Multi-Stage Pipelines

Now that you've provisioned all the required Azure resources and service connections, you can set up the pipelines for training **(Continuous Integration - CI)** and deploying **(Continuous Deployment - CD)** your machine learning model to production. Additionally, you can set up a pipeline for batch scoring.

- **Model CI, training, evaluation, and registration** - triggered on code changes to master branch on GitHub. Runs linting, unit tests, code coverage, and publishes and runs the training pipeline. If a new model is registered after evaluation, it creates a build artifact containing the JSON metadata of the model. Definition: diabetes_regression-ci.yml.
- **Release deployment** - consumes the artifact of the previous pipeline and deploys a model to either Azure Container Instances (ACI), Azure Kubernetes Service (AKS), or Azure App Service environments. See Further Exploration for other deployment types.

  Definition: diabetes_regression-cd.yml.

Note: Edit the pipeline definition to remove unused stages. For example, if you're deploying to Azure Container Instances and Azure Kubernetes Service only, you'll need to delete the unused Deploy_Webapp stage.

- **Batch Scoring Code Continuous Integration** - consumes the artifact of the model training pipeline. Runs linting, unit tests, code coverage, publishes a batch scoring pipeline, and invokes the published batch scoring pipeline to score a model.

These pipelines use a Docker container on the Azure Pipelines agents to accomplish the pipeline steps. The container image ***mcr.microsoft.com/mlops/python:latest*** is built with this Dockerfile and has all the necessary dependencies installed for MLOpsPython and ***diabetes_regression***. This image is an example of a custom Docker image with a pre-baked environment. The environment is guaranteed to be the same on any building agent, VM, or local machine. *In your project, you'll want to build your own Docker image that only contains the dependencies and tools required for your use case. Your image will probably be smaller and faster, and it will be maintained by your team.*

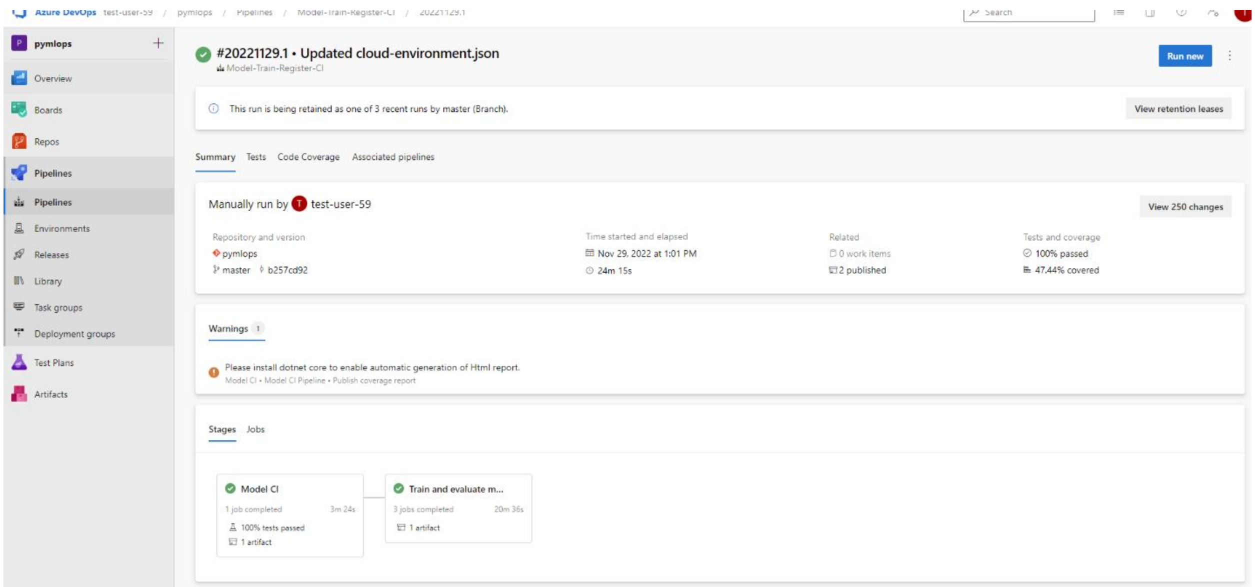
Set up the Model CI, training, evaluation, and registration pipeline

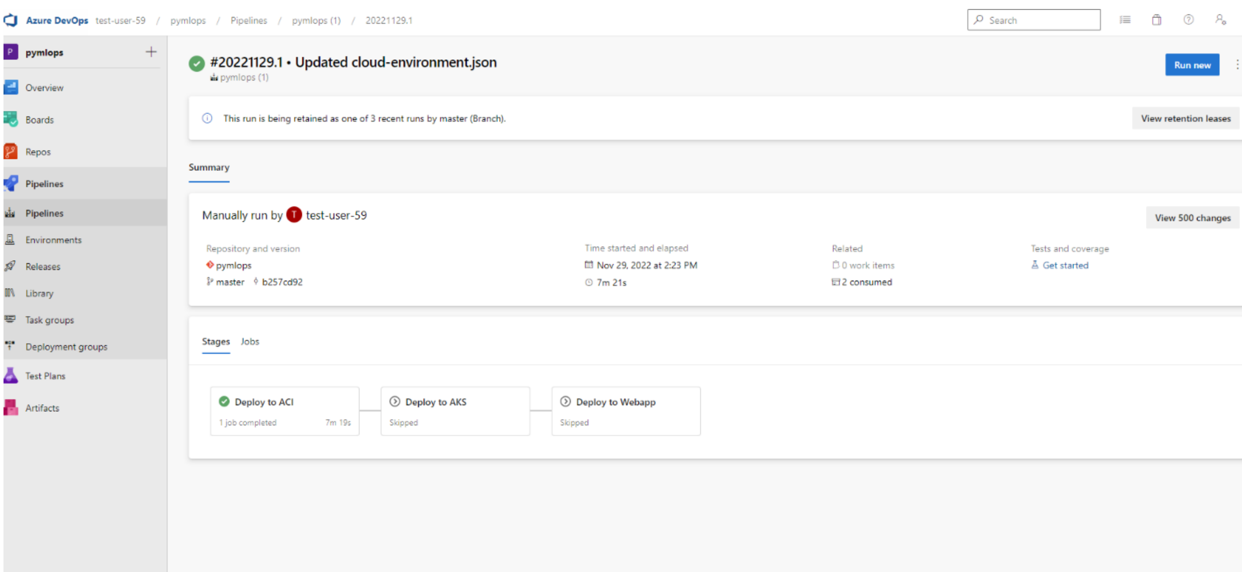
In your Azure DevOps project, create and run a new build pipeline based on the ***./pipelines/diabetes_regression-ci.yml*** pipeline definition in your forked repository.

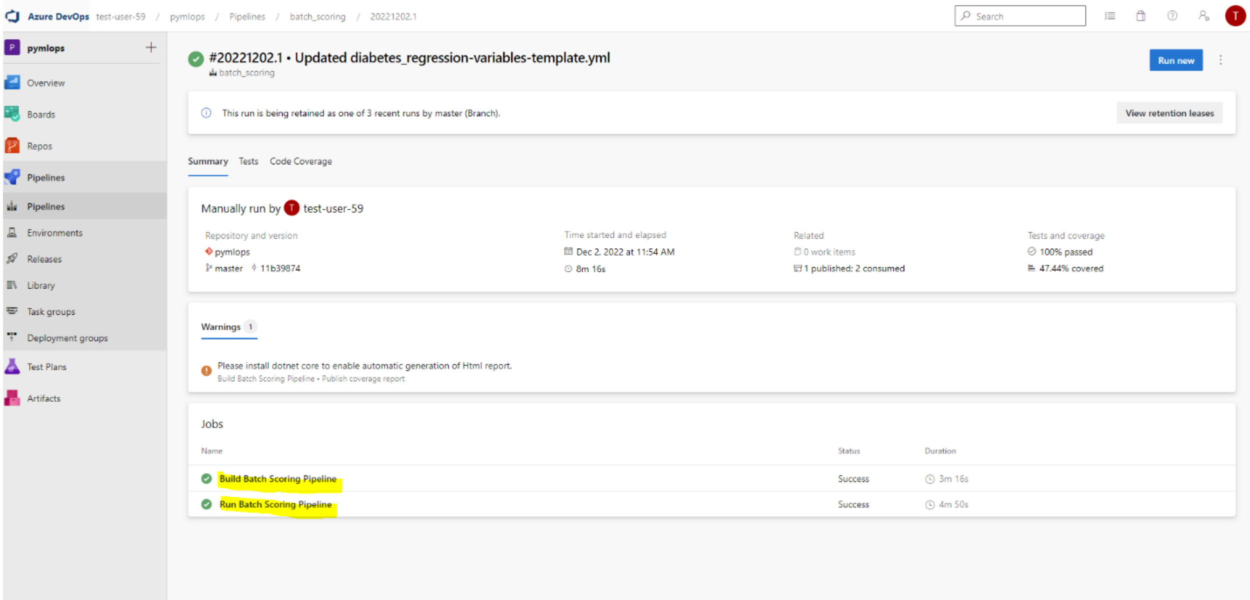
If you plan to use the release deployment pipeline (in the next section), you will need to rename this pipeline to Model-Train-Register-CI.

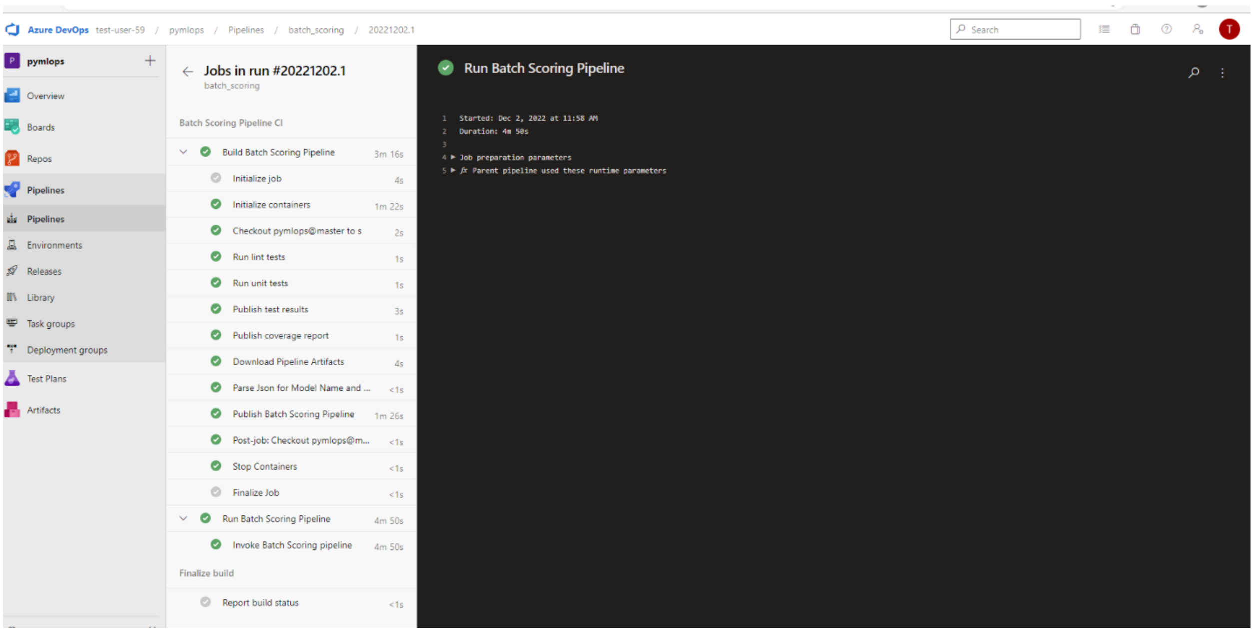
Note: To rename your pipeline, after you saved it, click Pipelines on the left menu on Azure DevOps, then All to see all the pipelines, then click the menu with the 3 vertical dots that appears when you hover the name of the new pipeline, and click it to pick "Rename/move pipeline".

Start a run of the pipeline if you haven't already, and once the pipeline is finished, check the execution result. Note that the run can take 20 minutes, with time mostly spent in Trigger ML Training Pipeline > Invoke ML Pipeline step. You can track the execution of the AML pipeline by opening the AML Workspace user interface. Screenshots are below:

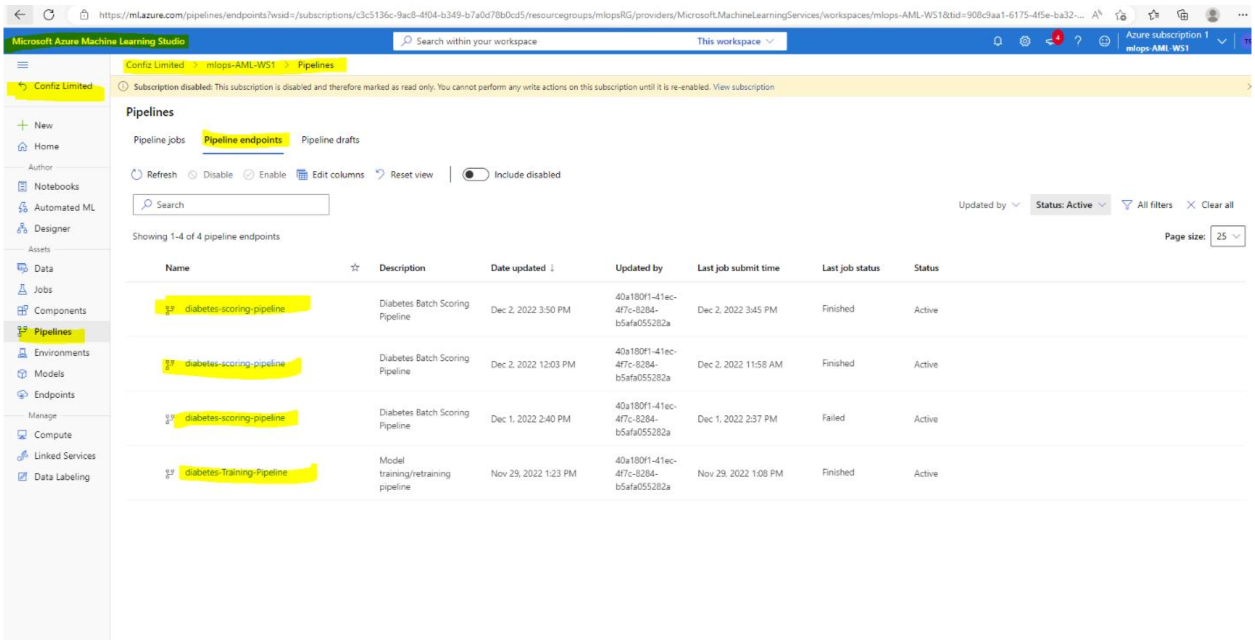

Azure DevOps test-user-59 / pymlops / Pipelines / Model-Train-Register-CI / 20221129.1
pymlops
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Deployment groups
Test Plans
Artifacts
#20221129.1 • Updated cloud-environment.json
Model-Train-Register-CI
Run new
This run is being retained as one of 3 recent runs by master (Branch).
View retention leases
Summary
Tests
Code Coverage
Associated pipelines
Manually run by test-user-59
View 250 changes
Repository and version
pymlops
master
b257cd92
Time started and elapsed
Nov 29, 2022 at 1:01 PM
24m 15s
Related
0 work items
2 published
Tests and coverage
100% passed
47.44% covered
Warnings 1
Please install dotnet core to enable automatic generation of Html report.
Model CI • Model CI Pipeline • Publish coverage report
Stages
Jobs
Model CI
1 job completed
3m 24s
100% tests passed
1 artifact
Train and evaluate m...
3 jobs completed
20m 36s
1 artifact


Azure DevOps test-user-59 / pymlops / Pipelines / pymlops (1) / 20221129.1
Search
pymlops
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Deployment groups
Test Plans
Artifacts
#20221129.1 • Updated cloud-environment.json
pymlops (1)
Run new
This run is being retained as one of 3 recent runs by master (Branch).
View retention leases
Summary
Manually run by test-user-59
View 500 changes
Repository and version
pymlops
master
b257cd92
Time started and elapsed
Nov 29, 2022 at 2:23 PM
7m 21s
Related
0 work items
2 consumed
Tests and coverage
Get started
Stages
Jobs
Deploy to ACI
1 job completed
7m 19s
Deploy to AKS
Skipped
Deploy to Webapp
Skipped



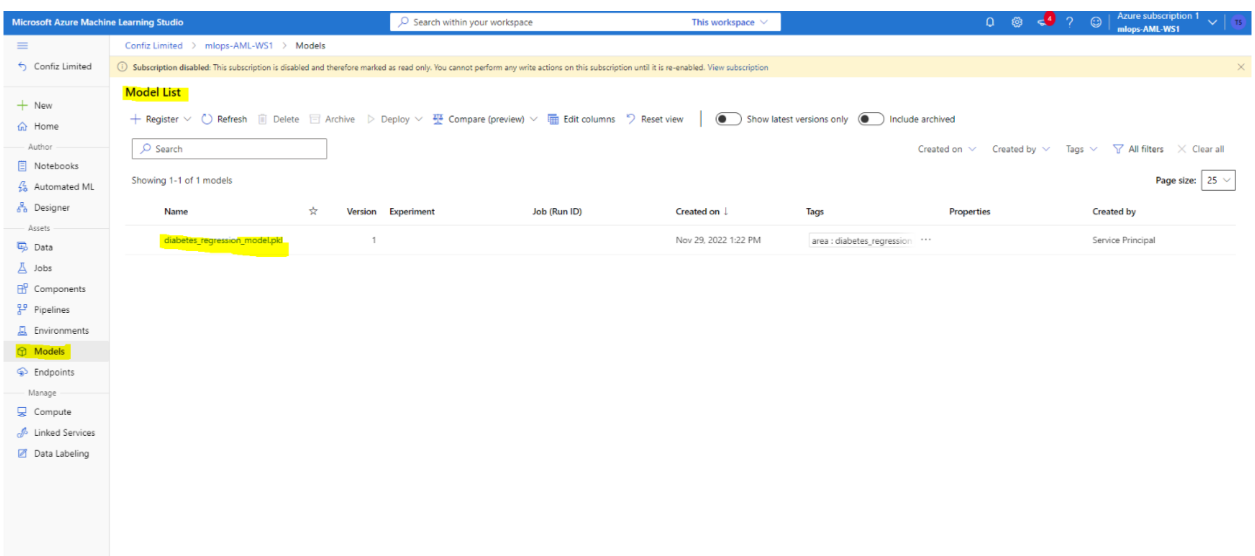


Also check the published training pipeline in your newly created AML workspace in Azure Machine Learning Studio:



Great, you now have the build pipeline for training set up which automatically triggers every time there's a change in the master branch!

After the pipeline is finished, you'll also see a new model in the AML Workspace model registry section:



To disable the automatic trigger of the training pipeline, change the `auto-trigger-training` variable as listed in the `.pipelines\diabetes_regression-ci.yml` pipeline to `false`. You can also override the variable at runtime execution of the pipeline.
The pipeline stages are summarized below:

Model CI

- Linting (code quality analysis)
- Unit tests and code coverage analysis
- Build and publish *ML Training Pipeline* in an *ML Workspace*

Train model

- Determine the ID of the *ML Training Pipeline* published in the previous stage.
- Trigger the *ML Training Pipeline* and wait for it to be completed.
    - This is an **agentless** job. The CI pipeline can wait for ML pipeline completion for hours or even days without using agent resources.
- Determine if a new model was registered by the *ML Training Pipeline*.
    - If the model evaluation step of the AML Pipeline determines that the new model doesn't perform any better than the previous one, the new model won't register, and the *ML Training Pipeline* will be **canceled**. In this case, you'll see a message in the 'Train Model' job under the 'Determine if evaluation succeeded and new model is registered' step saying '**Model was not registered for this run.**'
    - See evaluate_model.py for the evaluation logic. This is a simplified test that just looks at MSE to decide whether or not to register a new model. A more realistic verification would also do some error analysis and verify the inferences/error distribution against a test dataset, for example.
    - **Note**: *while it's possible to do an Evaluation Step as part of the ADO pipeline, this evaluation is logically part of the work done by Data Scientists, and as such the recommendation is that this step is done as part of the AML Pipeline and not ADO pipelines.*
    - Additional Variables and Configuration for configuring this and other behavior.

Create pipeline artifact

- Get the info about the registered model
- Create an Azure DevOps pipeline artifact called model that contains a model.json file containing the model information, for example:

Set up the Release Deployment and/or Batch Scoring pipelines

PRE-REQUISITES

To use these pipelines:

1. Follow the steps to set up the Model CI, training, evaluation, and registration pipeline.
2. You **must** rename your model CI/train/eval/register pipeline to Model-Train-Register-CI.

These pipelines rely on the model CI pipeline and reference it by name.

If you would like to change the name of your model CI pipeline, you must edit this section of yml for the CD and batch scoring pipeline, where it says `source: Model-Train-Register-CI` to use your own name.

```
trigger: none
resources:
  containers:
  - container: mlops
    image: mcr.microsoft.com/mlops/python:latest
  pipelines:
  - pipeline: model-train-ci
    source: Model-Train-Register-CI # Name of the triggering pipeline
    trigger:
      branches:
        include:
        - master
```

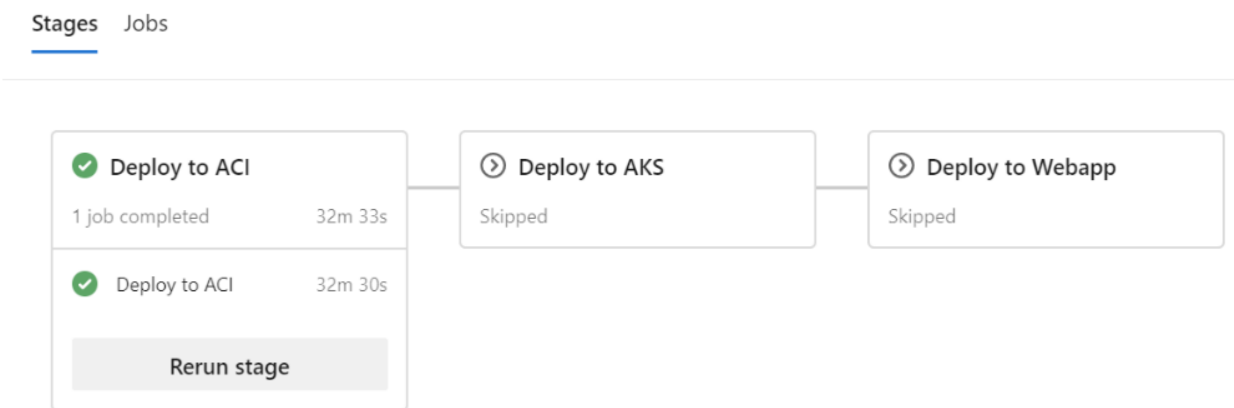
The release deployment and batch scoring pipelines have the following behaviors:

- The pipeline will **automatically trigger** on completion of the Model-Train-Register-CI pipeline for the master branch.
- The pipeline will default to using the latest successful build of the Model-Train-Register-CI pipeline. It will deploy the model produced by that build.
- You can specify a Model-Train-Register-CI build ID when running the pipeline manually. You can find this in the url of the build, and the model registered from that build will also be tagged with the build ID. This is useful to skip model training and registration and deploy/score a model successfully registered by a Model-Train-Register-CI build.
    - For example, if you navigate to a specific run of your CI pipeline, the URL should be something like https://dev.azure.com/yourOrgName/yourProjectName/_build/results?buildId=653&view=results. **653** is the build ID in this case.

Set up the Release Deployment pipeline

In your Azure DevOps project, create and run a new **build** pipeline based on the ./pipelines/diabetes_regression-cd.yml pipeline definition in your forked repository. It is recommended you rename this pipeline to something like Model-Deploy-CD for clarity.
Note: *While Azure DevOps supports both Build and Release pipelines, when using YAML you don't usually need to use Release pipelines. This repository assumes the usage only of Build pipelines.*

Your first run will use the latest model created by the Model-Train-Register-CI pipeline.
Once the pipeline is finished, check the execution result:



Set up the Batch Scoring pipeline

https://github.com/microsoft/MLOpsPython/blob/master/docs/getting_started.md#set-up-the-batch-scoring-pipeline

Data Drift Monitoring script

Open the “Notebooks” tab in the workspace.
use the following script:

```
import json
import pandas as pd
import numpy as np
import requests
import zipfile
import io
import plotly.offline as py #working offline
import plotly.graph_objs as go
from evidently.pipeline.column_mapping import ColumnMapping
from evidently.report import Report
from evidently.metric_preset import DataDriftPreset
import mlflow
import mlflow.sklearn
from mlflow.tracking import MlflowClient

from sklearn.datasets import load_diabetes

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')


sample_data = load_diabetes()
raw_data = pd.DataFrame(
```

```
    data=sample_data.data,
    columns=sample_data.feature_names)
raw_data.head()


raw_data['date'] = pd.date_range(start='1/1/2021', periods=len(raw_data), freq='D')
raw_data.head()

raw_data = raw_data.set_index('date')
raw_data.head()

data_columns = ColumnMapping()
data_columns.numerical_features = ['age', 'sex', 'bmi', 'bp', 's1']


#set reference dates
reference_dates = ('2021-01-01','2021-01-28')

#set experiment batches dates
experiment_batches = [
    ('2021-02-01','2021-02-28'),
    ('2021-03-01','2021-03-31'),
    ('2021-04-01','2021-04-30'),
    ('2021-05-01','2021-05-31'),
    ('2021-06-01','2021-06-30'),
    ('2021-07-01','2021-07-31'),
]

data_drift_report = Report(metrics=[DataDriftPreset()])
data_drift_report.run(reference_data=raw_data[:100], current_data=raw_data[100:200], column_mapping=data_c
olumns)
report = data_drift_report.as_dict()


report["metrics"][1]["result"]["drift_by_columns"]


#evaluate data drift with Evidently Profile
def detect_dataset_drift(reference, production, column_mapping, get_ratio=False):
    """
    Returns True if Data Drift is detected, else returns False.
    If get_ratio is True, returns the share of drifted features.
    The Data Drift detection depends on the confidence level and the threshold.
    For each individual feature Data Drift is detected with the selected confidence (default value is 0.95
).
    Data Drift for the dataset is detected if share of the drifted features is above the selected threshol
d (default value is 0.5).
    """
```

```
    
    data_drift_report = Report(metrics=[DataDriftPreset()])
    data_drift_report.run(reference_data=reference, current_data=production, column_mapping=column_mapping
)
    report = data_drift_report.as_dict()
    
    if get_ratio:
        return report["metrics"][0]["result"]["drift_share"]
    else:
        return report["metrics"][0]["result"]["dataset_drift"]



#evaluate data drift with Evidently Profile
def detect_features_drift(reference, production, column_mapping, get_scores=False):
    """
    Returns True if Data Drift is detected, else returns False. 
    If get_scores is True, returns scores value (like p-value) for each feature.
    The Data Drift detection depends on the confidence level and the threshold.
    For each individual feature Data Drift is detected with the selected confidence (default value is 0.95
).
    """
    
    data_drift_report = Report(metrics=[DataDriftPreset()])
    data_drift_report.run(reference_data=reference, current_data=production, column_mapping=column_mapping
)
    report = data_drift_report.as_dict()
    
    drifts = []
    num_features = column_mapping.numerical_features if column_mapping.numerical_features else []
    cat_features = column_mapping.categorical_features if column_mapping.categorical_features else []
    for feature in num_features + cat_features:
        drift_score = report["metrics"][1]["result"]["drift_by_columns"][feature]["drift_score"]
        if get_scores:
            drifts.append((feature, drift_score))
        else:
            drifts.append((feature, report["metrics"][1]["result"]["drift_by_columns"][feature]["drift_det
ected"]))
            
    return drifts



features_historical_drift = []

for date in experiment_batches:
    drifts = detect_features_drift(raw_data.loc[reference_dates[0]:reference_dates[1]], 
                           raw_data.loc[date[0]:date[1]], 
                           column_mapping=data_columns)
    
```

```
    features_historical_drift.append([x[1] for x in drifts])

features_historical_drift_frame = pd.DataFrame(features_historical_drift,
                                               columns = data_columns.numerical_features)


fig = go.Figure(data=go.Heatmap(
                   z = features_historical_drift_frame.astype(int).transpose(),
                   x = [x[1] for x in experiment_batches],
                   y = data_columns.numerical_features,
                   hoverongaps = False,
                   xgap = 1,
                   ygap = 1,
                   zmin = 0,
                   zmax = 1,
                   showscale = False,
                   colorscale = 'Bluered'
))

fig.update_xaxes(side="top")

fig.update_layout(
    xaxis_title = "Timestamp",
    yaxis_title = "Feature Drift"
)

fig.show()


 features_historical_drift_pvalues = []


for date in experiment_batches:
    drifts = detect_features_drift(raw_data.loc[reference_dates[0]:reference_dates[1]],
                                   raw_data.loc[date[0]:date[1]],
                                   column_mapping=data_columns,
                                   get_scores=True)

    features_historical_drift_pvalues.append([x[1] for x in drifts])

features_historical_drift_pvalues_frame = pd.DataFrame(features_historical_drift_pvalues,
                                                       columns = data_columns.numerical_features)



fig = go.Figure(data=go.Heatmap(
                   z = features_historical_drift_pvalues_frame.transpose(),
                   x = [x[1] for x in experiment_batches],
                   y = features_historical_drift_pvalues_frame.columns,
                   hoverongaps = False,
                   xgap = 1,
```

```
                   ygap = 1,
                   zmin = 0,
                   zmax = 1,
                   colorscale = 'reds_r'
                   )
               )

fig.update_xaxes(side="top")

fig.update_layout(
    xaxis_title = "Timestamp",
    yaxis_title = "p-value"
)

fig.show()
```

Dataset Drift

```
dataset_historical_drift = []

for date in experiment_batches:
    dataset_historical_drift.append(detect_dataset_drift(raw_data.loc[reference_dates[0]:reference_dates[1
]],
                                   raw_data.loc[date[0]:date[1]],
                                   column_mapping=data_columns))


fig = go.Figure(data=go.Heatmap(
                   z = [[1 if x == True else 0 for x in dataset_historical_drift]],
                   x = [x[1] for x in experiment_batches],
                   y = [''],
                   hoverongaps = False,
                   xgap = 1,
                   ygap = 1,
                   zmin = 0,
                   zmax = 1,
                   colorscale = 'Bluered',
                   showscale = False
                   )
               )
fig.update_xaxes(side="top")
fig.update_layout(
    xaxis_title = "Timestamp",
    yaxis_title = "Dataset Drift"
)
fig.show()


dataset_historical_drift_ratio = []
```

```
for date in experiment_batches:
    dataset_historical_drift_ratio.append(detect_dataset_drift(raw_data.loc[reference_dates[0]:reference_d
ates[1]], 
                                           raw_data.loc[date[0]:date[1]],
                                           column_mapping=data_columns,
                                           get_ratio=True))



fig = go.Figure(data=go.Heatmap(
                   z = [dataset_historical_drift_ratio],
                   x = [x[1] for x in experiment_batches],
                   y = [''],
                   hoverongaps = False,
                   xgap = 1,
                   ygap = 1,
                   zmin = 0.5,
                   zmax = 1,
                   colorscale = 'reds'
                  )
                )
fig.update_xaxes(side="top")
fig.update_layout(
    xaxis_title = "Timestamp",
    yaxis_title = "Dataset Drift"
)
fig.show()
```

Log Dataset Drift in MLFlow

```
#log into MLflow
client = MlflowClient()

#set experiment
mlflow.set_experiment('Dataset Drift Analysis with Evidently')

#start new run
for date in experiment_batches:
    with mlflow.start_run() as run: 
        
        # Log parameters
        mlflow.log_param("begin", date[0])
        mlflow.log_param("end", date[1])

        # Log metrics
        metric = detect_dataset_drift(raw_data.loc[reference_dates[0]:reference_dates[1]], 
                                           raw_data.loc[date[0]:date[1]],
```

```
                                  column_mapping=data_columns,
                                  get_ratio=True)

        mlflow.log_metric('dataset drift', metric)

        print(run.info)
```