End-to-end machine learning operations (MLOps) with Azure Machine Learning

This document serves as my personal tutorial guide, detailing the steps I executed to implement a complete Machine Learning Operations (MLOps) pipeline for a diabetes prediction model. The workflow covers my approach to environment setup, model training, and continuous delivery (CI/CD) on Azure Machine Learning (AML). In this learning path I have completed 7 challenges to complete the task.

Challenge 0: Convert a notebook to production code

My Understanding:

In this challenge I converted a Jupyter notebook to production ready code. I understood that when I store these scripts then it is easier to automate the code execution so I can parameterize scripts to easily reuse the code for retraining.

In this challenge I have created a new public repo by navigating to https://github.com/MicrosoftLearning/mslearn-mlops and selecting use this template feature.

And in that in the experimentation folder, I found a Jupyter Notebook that trains a Classification model and the data used by this notebook is in the experimentation/data. And in the src/model folder I found train.py that is used for training

Challenge 1: Create an Azure Machine Learning job

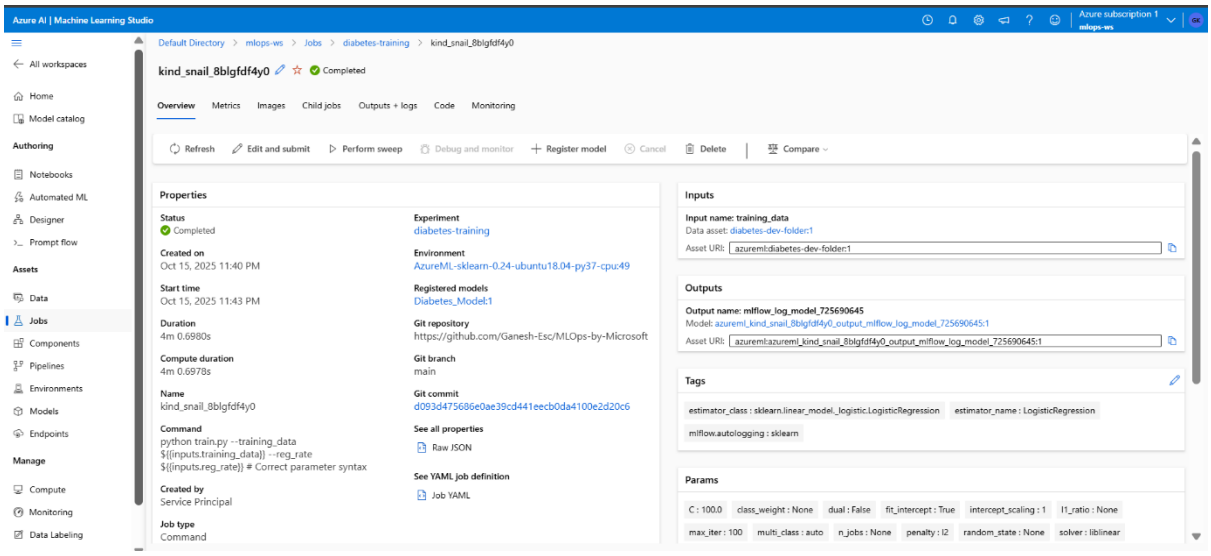
My Understanding:

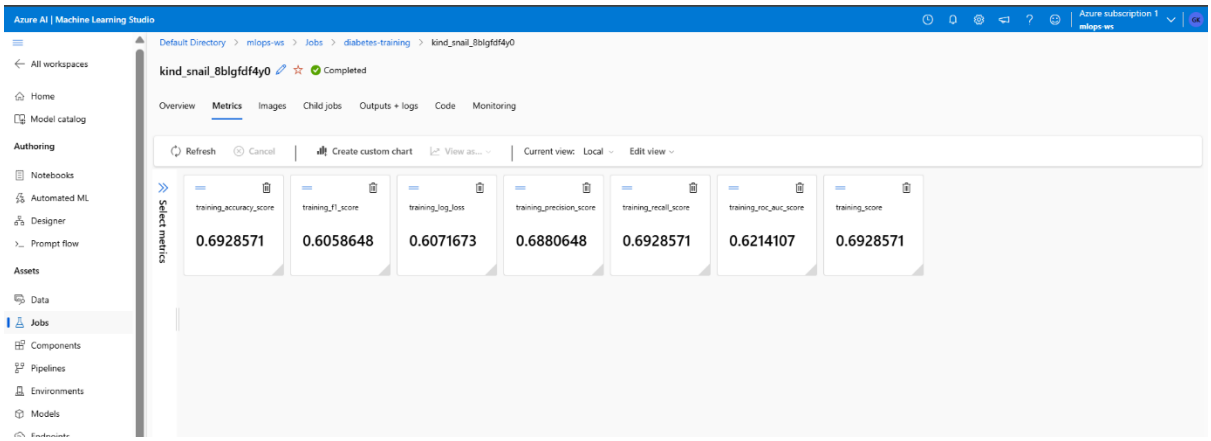
In this challenge I understood that to automate machine learning workflows I can define machine learning tasks in scripts. And to execute those workflows which have Python scripts, I must use Azure Machine Learning jobs. And Azure Machine Learning jobs store all metadata of a workflow, including input parameters and output metrics. By running scripts as jobs, it's easier to track and manage machine learning models

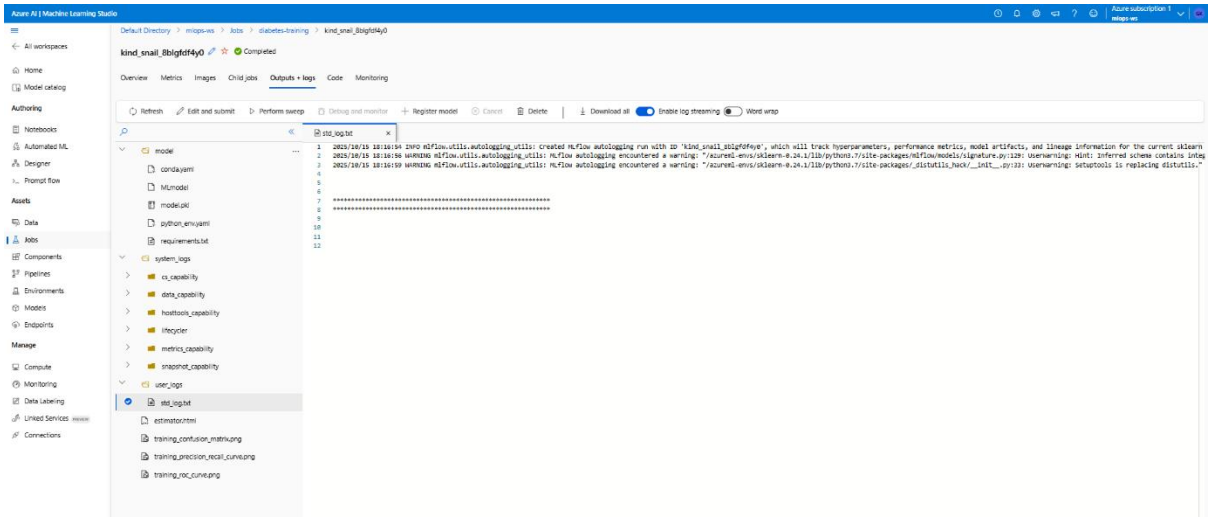
In this challenge, I found a YAML file to define the job. And I created an Azure Machine Learning workspace and a compute instance and ran job using CLI (v2).

Result: A successfully completed job in the Azure Machine Learning workspace. The job should contain all input parameters and output metrics for the model you trained.

Azure Machine Learning Job details:







Challenge 2: Trigger the Azure Machine Learning job with GitHub Actions

My Understanding:

By using CLI I can run Machine Learning job from anywhere. Using a platform like GitHub will allow us to automate Azure Machine Learning jobs. And to trigger the job to run, I can use GitHub Actions.

In the. github**/**workflows folder, I found the 02-manual-trigger.yml file. This file defines a GitHub Action which can be manually triggered. The workflow checks out the repo onto the runner, installs the Azure Machine Learning extension for the CLI (v2), and logs in to Azure using the AZURE_CREDENTIALS secret.

I created a service principal, using the Cloud Shell in the Azure portal, which has contributor access to the resource group I created.

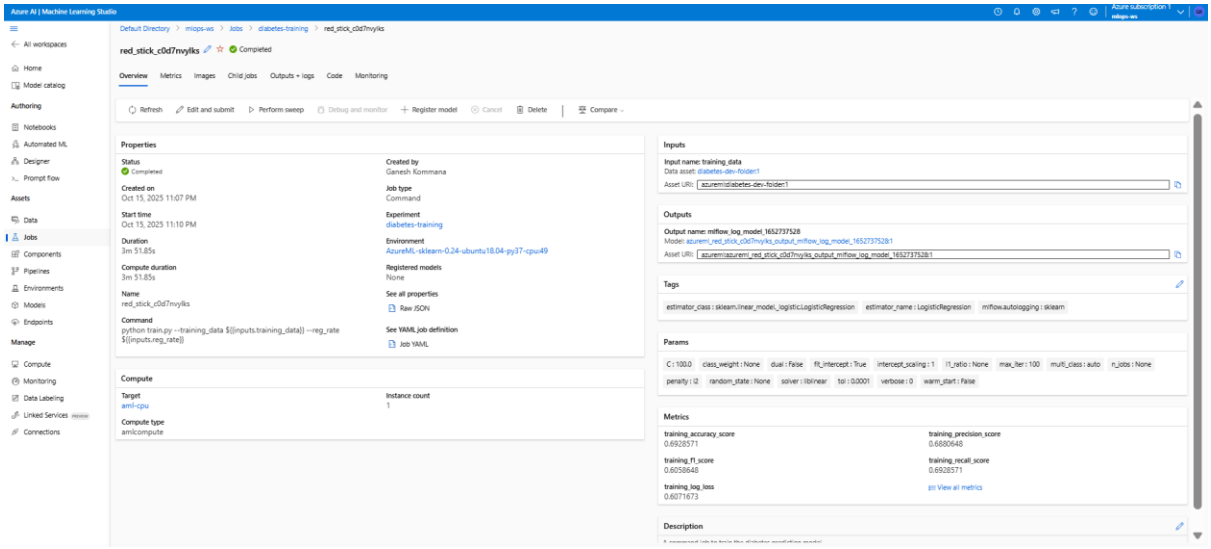
Created a GitHub secret in your repository and named it AZURE_CREDENTIALS and copy and paste the output of the service principal to the **Value** field of the secret.

Result:

A successfully completed Action in your GitHub repo, triggered manually in GitHub.

A step in the Action should have submitted a job to the Azure Machine Learning workspace.

A successfully completed Azure Machine Learning job, shown in the Azure Machine Learning workspace.



Challenge 3: Trigger GitHub Actions with feature-based development

My Understanding:

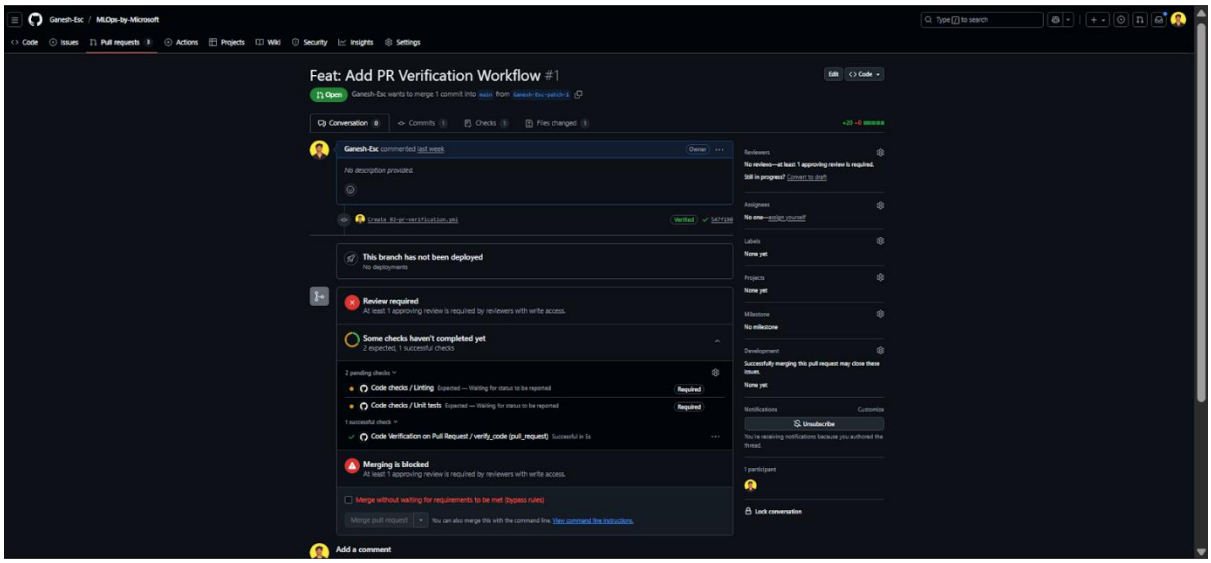
Triggering a workflow by pushing directly to the repo is **not** considered a best practice. Preferably, I must review any changes before you build them with GitHub Actions.

In this challenge, I created a GitHub Actions workflow which is triggered by the creation of a pull request. And also created a **branch protection rule** to block any direct pushes to the **main** branch.

To trigger the workflow, I did the following:

- Created a branch in the repo.
- Make a change and push it. For example, I changed the hyperparameter value.
- Created a pull request merge the new branch with the main.

Result:



Challenge 4: Work with linting and unit testing

My understanding:

Code quality can be assessed in two ways: linting and unit testing. Use linting to check for any stylistic errors and unit testing to verify your functions.

In this challenge I found that there are files in the test folder that perform linting and unit testing on the scripts. The flake8 lints the code to check for stylistic errors. The test_train.py performs unit tests on the code to check whether functions behave as expected.

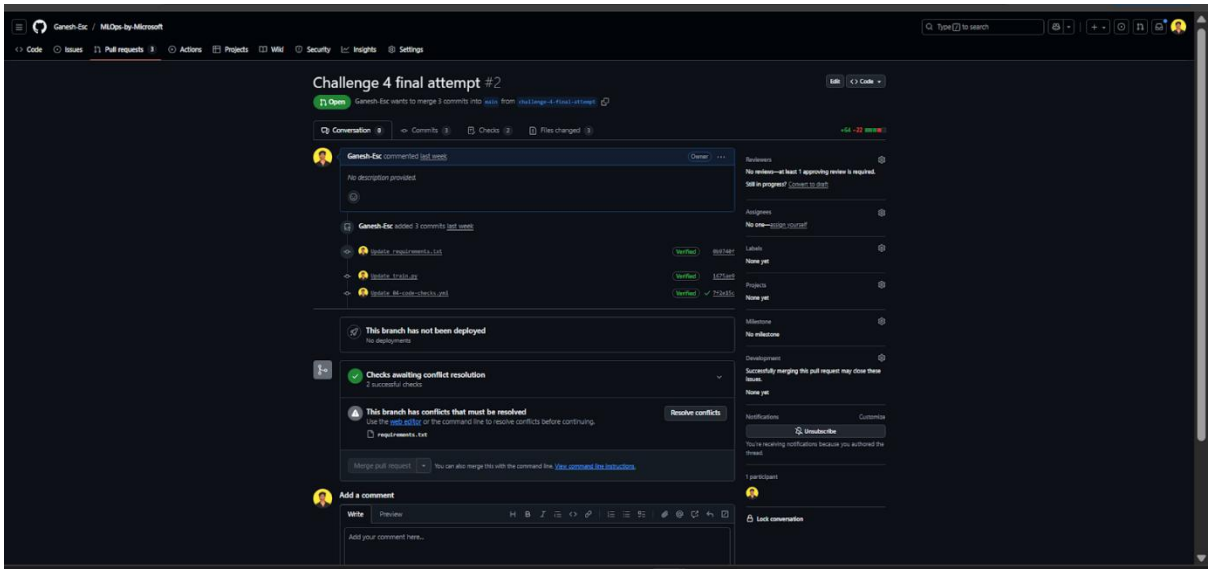
To achieve this, I triggered the code checks manually in the Actions tab of Github repository.

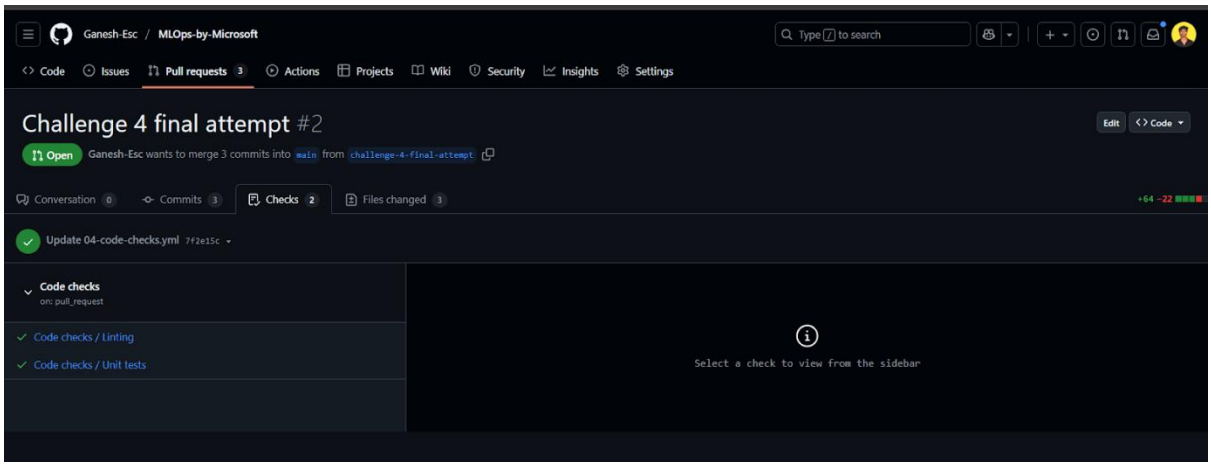
And to trigger the workflow, I did the following:

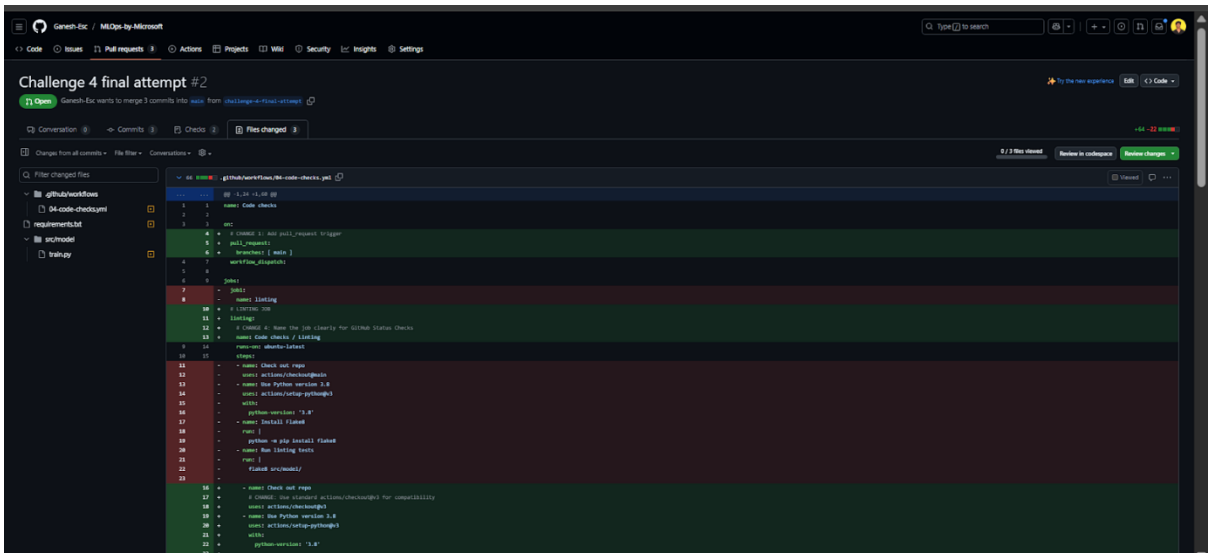
Made a change and pushed it. And created a pull request, showing the integrated code checks.

Result:

Both the **Linting** and **Unit tests** checks are completed successfully without any errors. The successful checks should be shown in a newly created pull request.


Challenge 4 final attempt #2
Open
Ganesh-Esc wants to merge 3 commits into main from challenge-4-final-attempt
Conversation
Commits
Checks
Files changed
Ganesh-Esc commented last week
No description provided.
Ganesh-Esc added 3 commits last week
Update requirements.txt
Update train.py
Update 04-code-checks.yml
This branch has not been deployed
No deployments
Checks awaiting conflict resolution
2 successful checks
This branch has conflicts that must be resolved
Use the web editor or the command line to resolve conflicts before continuing.
requirements.txt
Resolve conflicts
Merge pull request
You can also merge this with the command line. View command line instructions.
Add a comment
Write
Preview
Add your comment here...
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation


Ganesh-Esc / MLOps-by-Microsoft
Type / to search
Code
Issues
Pull requests 3
Actions
Projects
Wiki
Security
Insights
Settings
Challenge 4 final attempt #2
Open
Ganesh-Esc wants to merge 3 commits into main from challenge-4-final-attempt
Edit
Code
Conversation 0
Commits 3
Checks 2
Files changed 3
+64 −22
Update 04-code-checks.yml 7f2e15c
Code checks
on: pull_request
Code checks / Linting
Code checks / Unit tests
Select a check to view from the sidebar


Challenge 4 final attempt #2
Open
Files changed 3
Filter changed files
.github/workflows
04-code-checks.yml
requirements.txt
src/model
train.py
.github/workflows/04-code-checks.yml
name: Code checks
on:
pull_request:
branches: [main]
workflow_dispatch:
jobs:
linting:
name: Code checks / Linting
runs-on: ubuntu-latest
steps:
- name: Check out repo
uses: actions/checkout@main
- name: Use Python version 3.8
uses: actions/setup-python@v3
with:
python-version: '3.8'
- name: Install flake8
run: |
python -m pip install flake8
- name: Run linting tests
run: |
flake8 src/model/

Challenge 5: Work with environments

My Understanding:

There are many advantages to using environments in Machine learning projects. When we have separate environments for development, staging, and production, we can more easily control access to resources.

It's better to use environments to isolate workloads and control the deployment of the model.

In this Challenge
I created a development and production environment and added an approval check to the production environment

Removed the global repo AZURE_CREDENTIALS secret, so that each environment will only be able to use its own secret and added the **AZURE_CREDENTIALS** secret to each environment that contains the service principal output.

And then Created a new data asset in the workspace with the following configuration:

Name: *diabetes-prod-folder*

Path: The **data** folder in the **production** folder which contains a larger CSV file to train the model. The path should point to the folder, not to the specific file.

Also Created one GitHub Actions workflow, triggered by changes being pushed to the main branch, with two jobs:

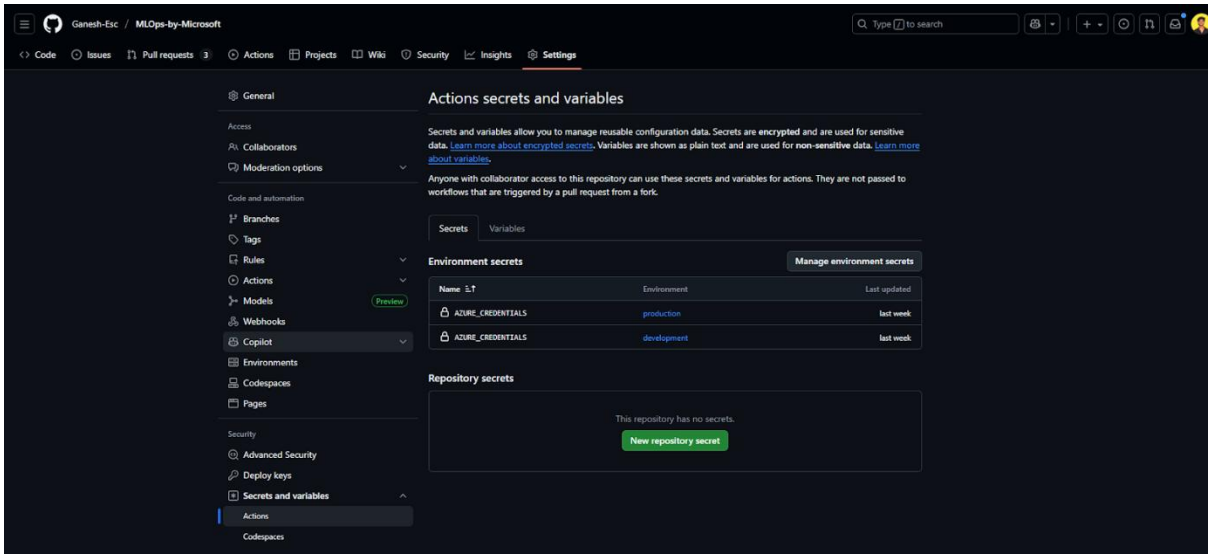
The **experiment** job that trains the model using the *diabetes-dev-folder* dataset in the **development environment**.

The **production** job that trains the model in the **production environment**, using the production data (the *diabetes-prod-folder* data asset as input).

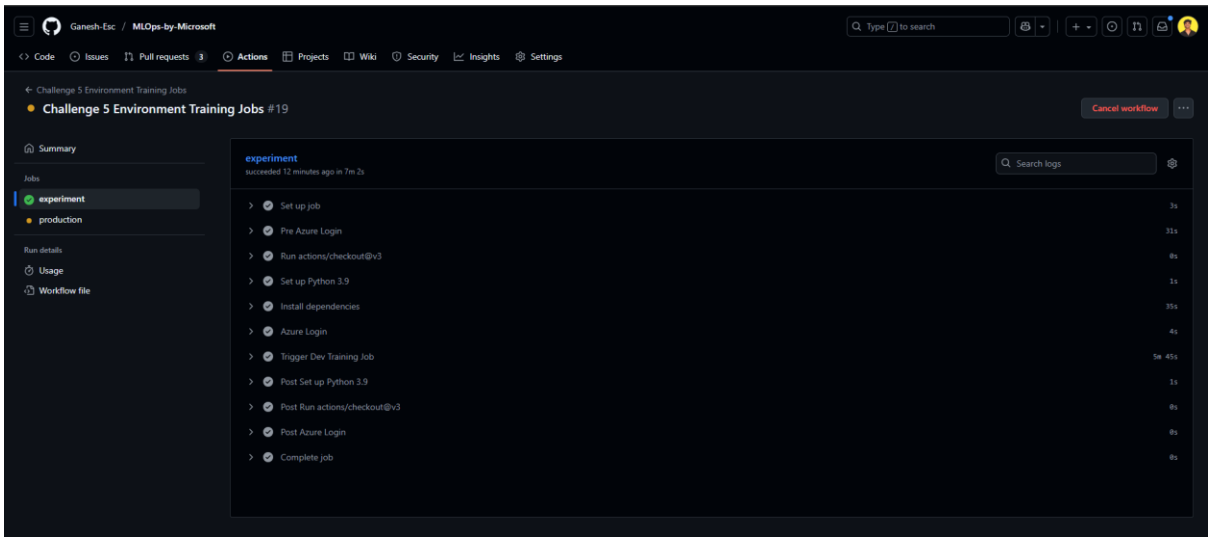
Added a condition that the **production** job is only allowed to run when the **experiment** job ran *successfully*. Success means that the Azure Machine Learning job ran successfully too.

Result:

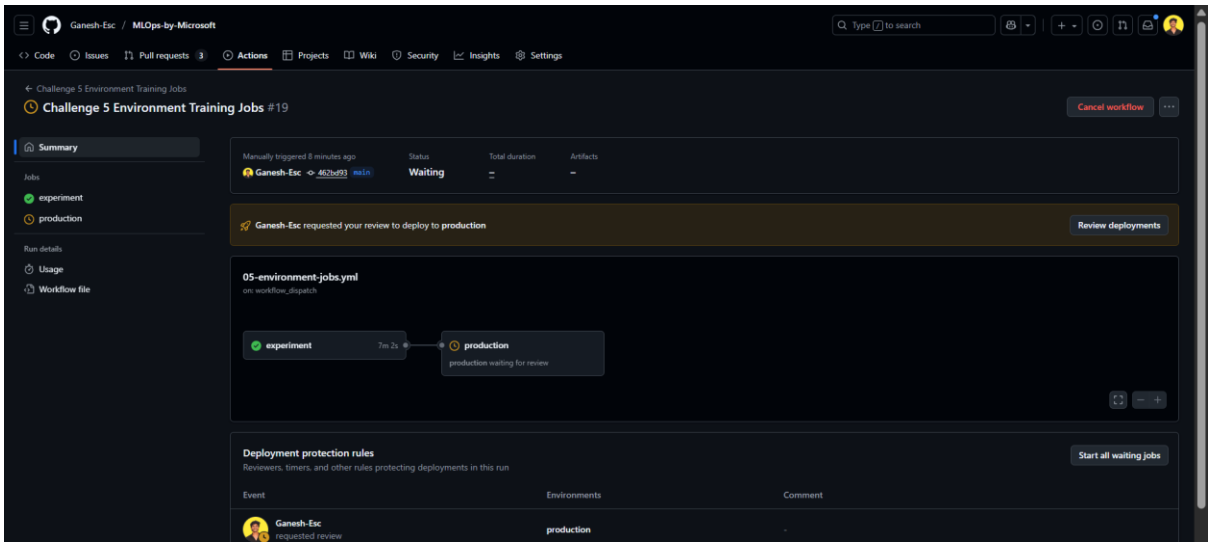
Show the environment secrets in the settings.

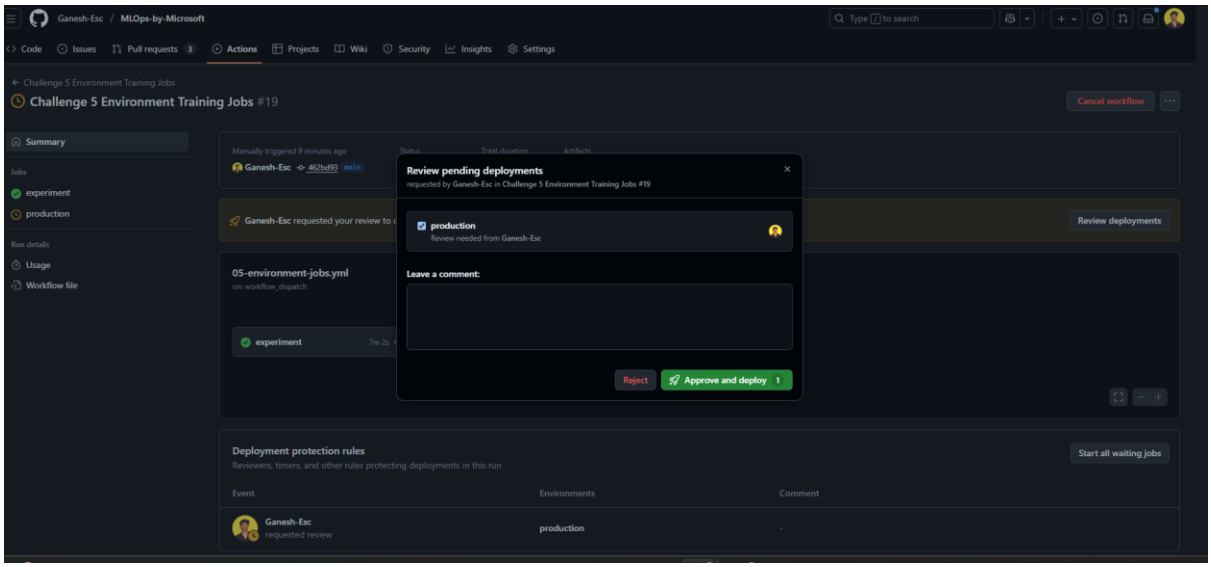


A successfully completed Actions workflow that contains two jobs. The production job needs the experimentation job to be successful to run.

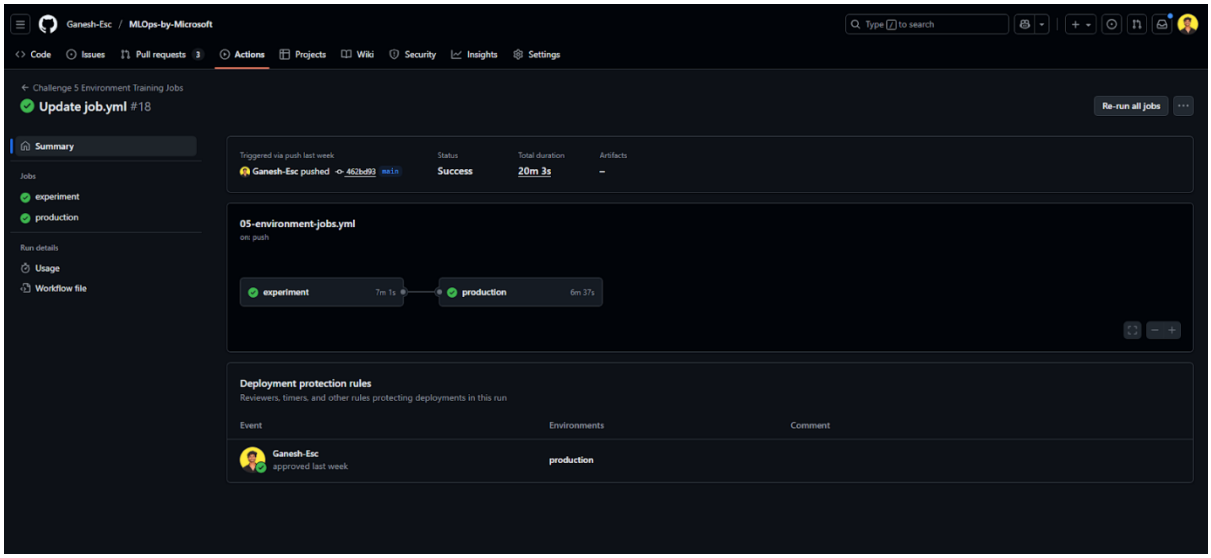


Show that the workflow required an approval before running the production workload.

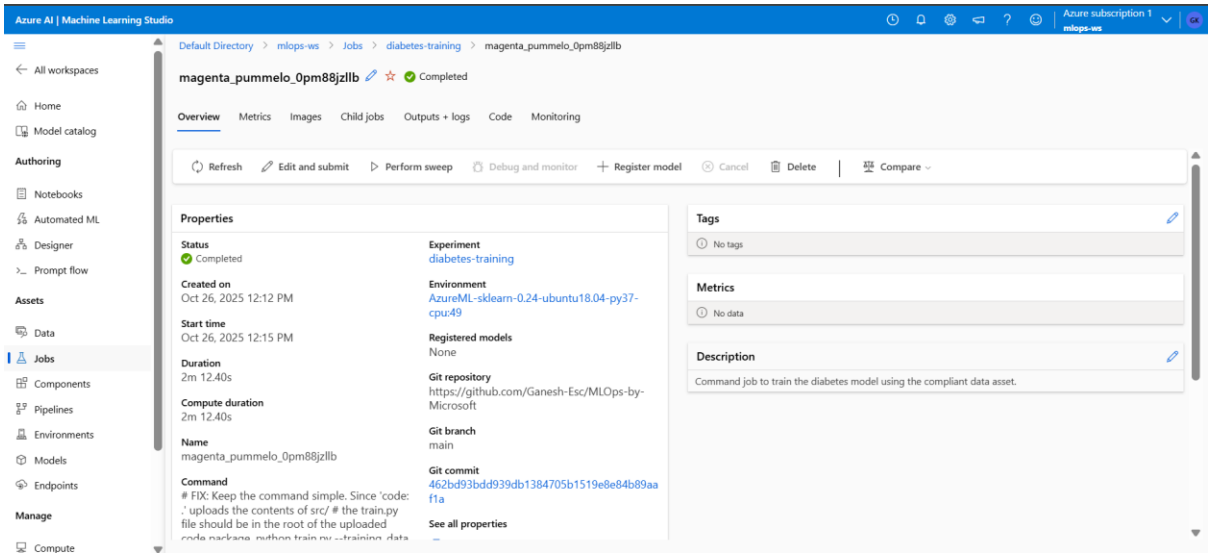


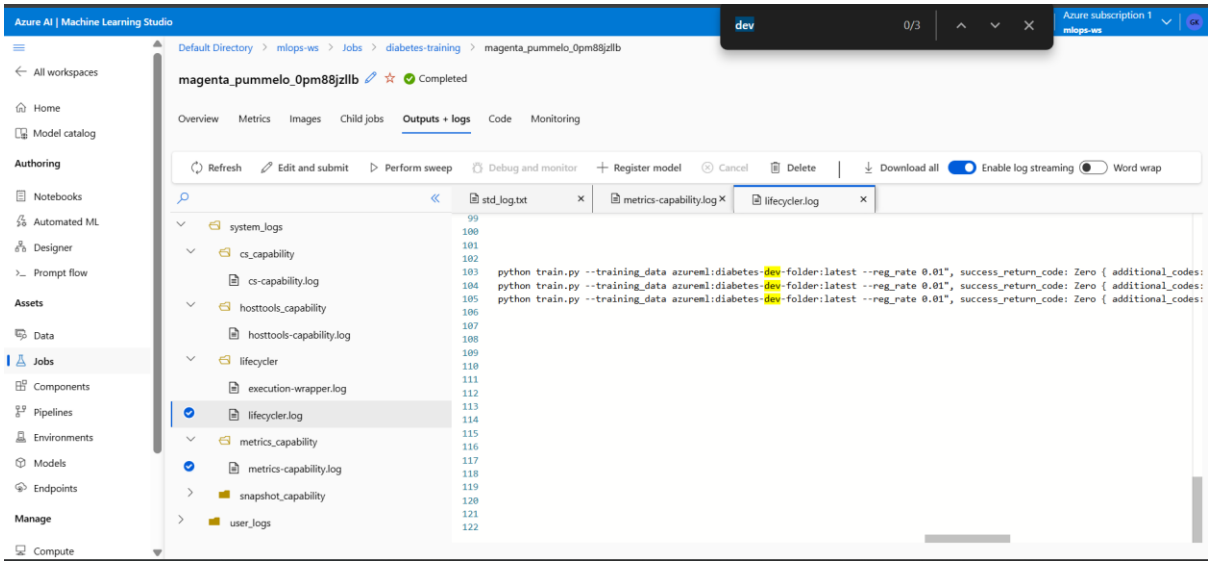


Show two successful Azure Machine Learning jobs, one trained with the *diabetes-dev-folder* as input and the other with the *diabetes-prod-folder* as input.

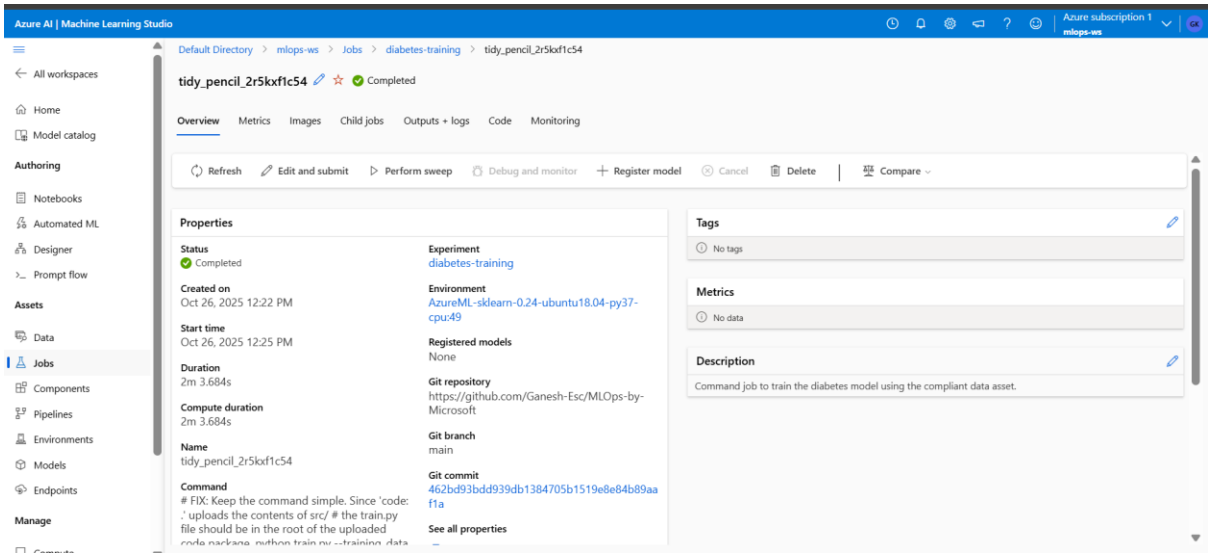


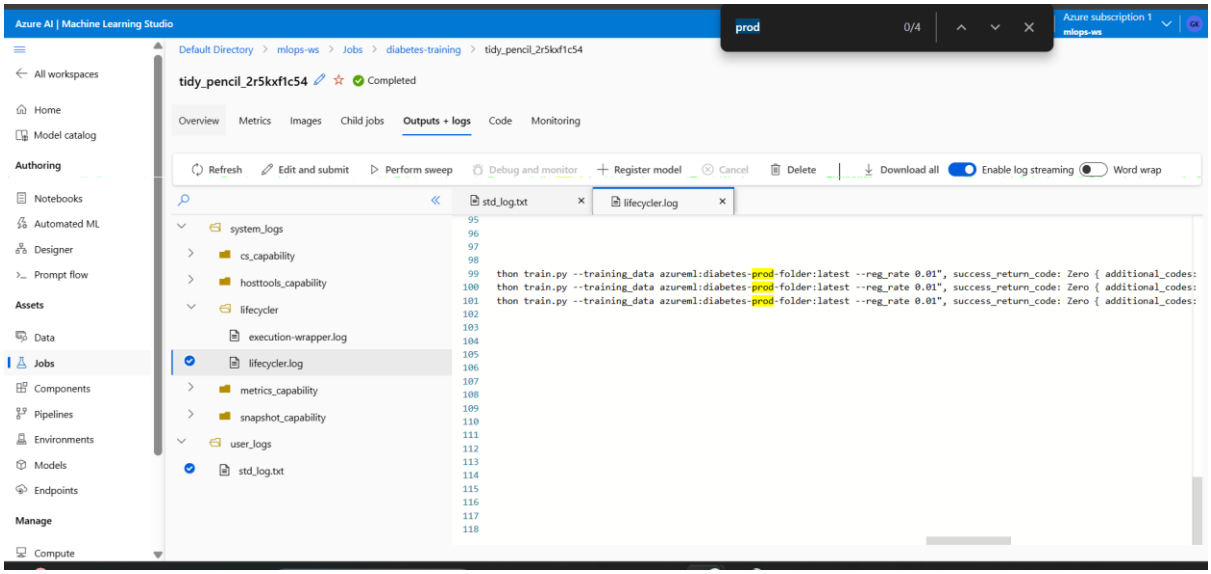
Development Job details:





Production Job Details:





Challenge 6: Deploy and test the model

My Understanding:

To get value from a model, we must deploy it. And we can deploy a model to a managed online or batch endpoint

In this challenge, I

Registered the model from the production job output in the Azure Machine Learning Studio.

Created a GitHub Actions workflow which deploys the latest version of the registered model. And the workflow should create an endpoint and deploy your model to the endpoint using the CLI (v2).

Result:

A model registered in the Azure Machine Learning workspace.

A successfully completed Action in your GitHub repo that deploys the model to a managed online endpoint.