MLOps: When DevOps meets Machine Learning

Andrea Afify (813466), Emanuele Marnati (812503), Alessandro Risaro (825113)

Table of contents

MLOps

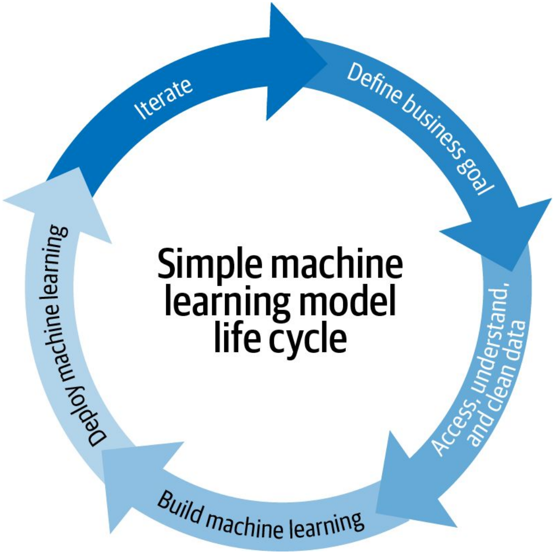
Aims and Needs

A Short **Introduction**

When you are looking for **machine learning solutions to real problems**, you may encounter **several difficulties**.

As a matter of fact, during the implementation of these models, it is necessary to go through **several steps** that represent the life cycle of a machine learning model:

- **Collect and process raw data.**
- **Analyze the data.**
- **Process the data for training.**
- **Construct, train and test the model.**
- **Validate and tune the model.**
- **Deploy and monitor the model.**



Why MLOps?

Hopefully, it’s clear just how **work-intensive** the entire process can get, especially since it will most likely **need to be repeated multiple times**.

- **The entire cycle is time consuming**: updating the model on new data patterns and trends, it is a problem which can take up hours of manual labor that can be better spent elsewhere.

- **Cost Increase**: worsening the overall maintenance costs because the costs for deployed machine learning models are added on top of the costs for the software application utilizing the services of the models.

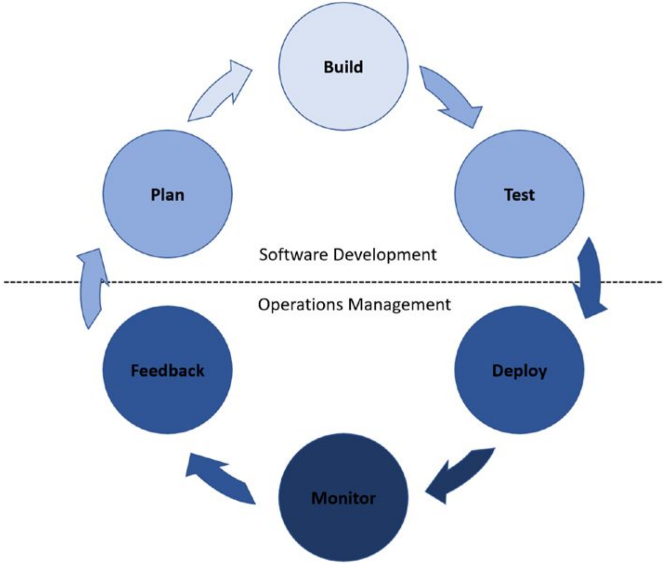
A possible solution: **MLOps**

MLOps, which can be thought of as the **intersection between machine learning and DevOps practices**, tries to solve the previous problems.

As a matter of fact, DevOps, or developmental operations, refers to a set of practices that combines the work processes of software developers with those of operational teams to create a common set of practices that functions as a hybrid of the two roles. In this way:

1. the developmental cycle of software is expedited, and **continuous delivery** of software products is ensured.

2. **Total costs also go down** because maintenance costs are reduced as a result of the increase in efficiency of the workflow in maintaining the software applications.



MLOps: **Three different setups**

We have three main setups of **machine learning solutions**:

1. **Manual implementation.**

2. **Continuous model delivery.**

3. **Continuous integration/continuous delivery of pipelines.**

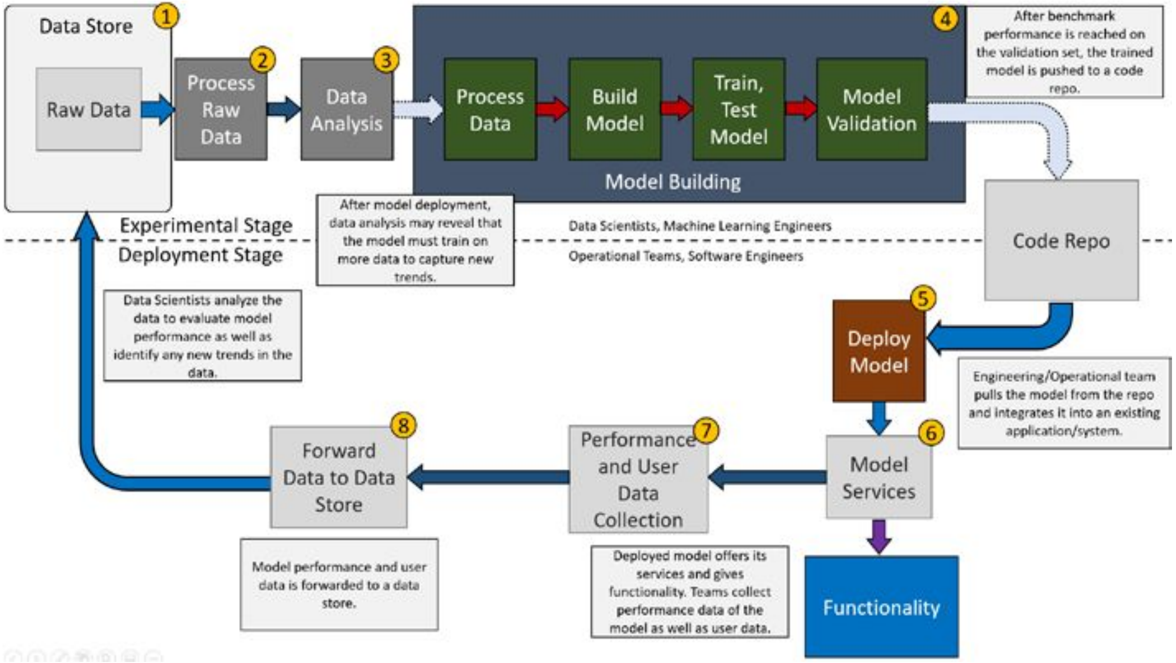
Manual Implementation

Refers to a setup where **there are no MLOps principles applied** and everything is manually implemented.

The steps discussed above in the creation of a machine learning model are **all manually performed**.

Software engineering teams must **manually integrate the models into the application**, and **operational** teams must help **ensure all functionality is preserved** along with collecting data and performance metrics of the model.

How it works!



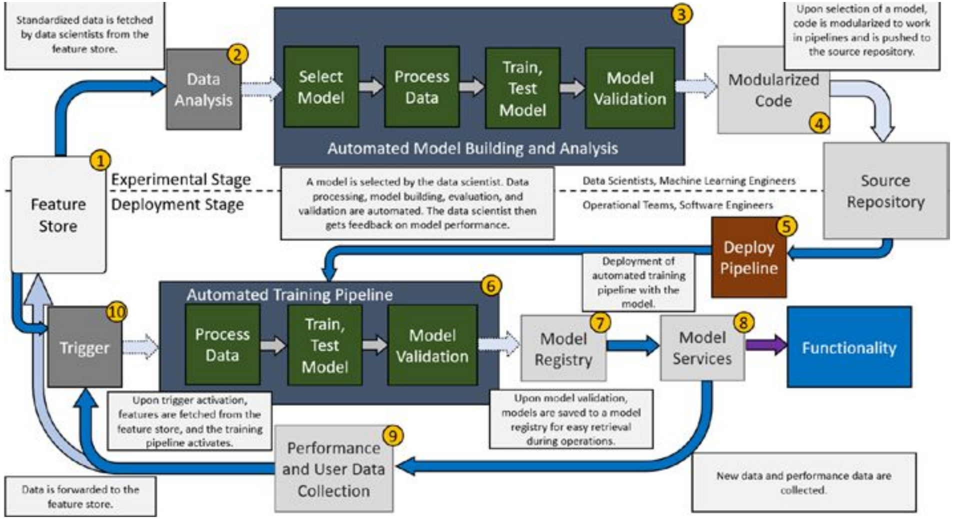
Data Store
Raw Data
Process Raw Data
Data Analysis
Process Data
Build Model
Train, Test Model
Model Validation
Model Building
After benchmark performance is reached on the validation set, the trained model is pushed to a code repo.
Code Repo
Experimental Stage
Deployment Stage
After model deployment, data analysis may reveal that the model must train on more data to capture new trends.
Data Scientists, Machine Learning Engineers
Operational Teams, Software Engineers
Data Scientists analyze the data to evaluate model performance as well as identify any new trends in the data.
Deploy Model
Engineering/Operational team pulls the model from the repo and integrates it into an existing application/system.
Forward Data to Data Store
Performance and User Data Collection
Model Services
Model performance and user data is forwarded to a data store.
Deployed model offers its services and gives functionality. Teams collect performance data of the model as well as user data.
Functionality

Continuous Model Delivery

Here, we see the emergence of **pipelines** to allow for **automation** of the machine learning side of the process.

The main feature of this type of setup is that the deployed model has pipelines established to **continuously train it on new data, even after deployment**.

How it works!



Automation of the **experimental stage**, or the **model development stage**, also emerges along with **modularization of code** to allow for further automation in the subsequent steps.

In this setup, **continuous delivery** refers to **expedited development and deployment** of new machine learning models.

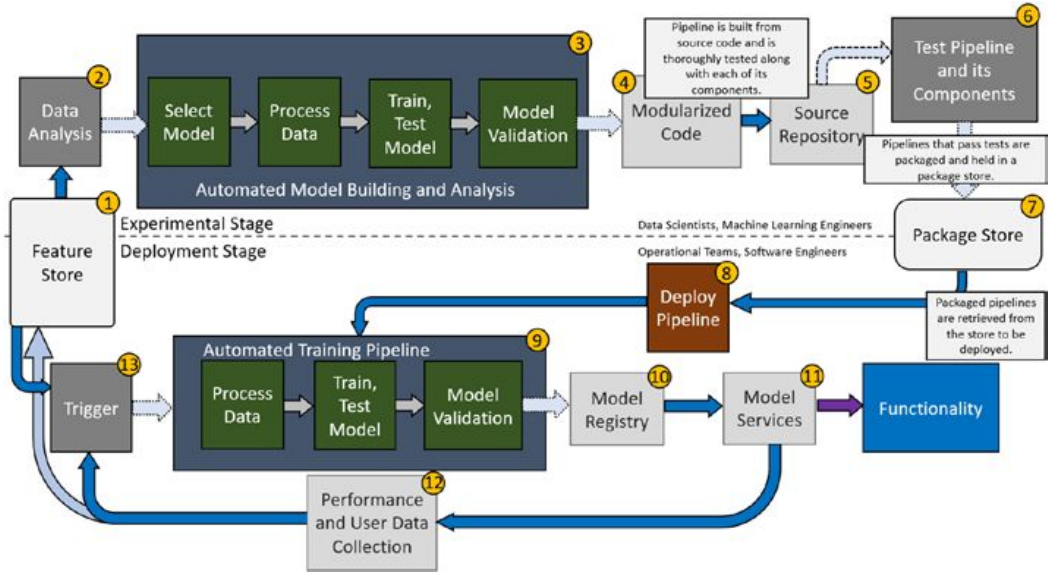
With the barriers to rapid deployment lifted (the tediousness of manual work in the experimental stage) by automation, **models can now be created or updated at a much faster pace**.

Continuous Integration & Delivery of Pipeline

It refers to a setup where pipelines in the experimental stage are **thoroughly tested in an automated process** to make sure all components work as intended.

From there, **pipelines are packaged and deployed**, where **deployment teams** deploy the pipeline to a **test environment**, handle additional testing to ensure both **compatibility** and **functionality**, and then deploy it to the production environment.

How it works!



In this setup, pipelines can now be created and deployed at a **quick pace**, allowing for teams to **continuously create new pipelines** built around the latest in machine learning architectures without any of the resource barriers associated with **manual testing and integration**.

MLOps vs DevOps vs Agile

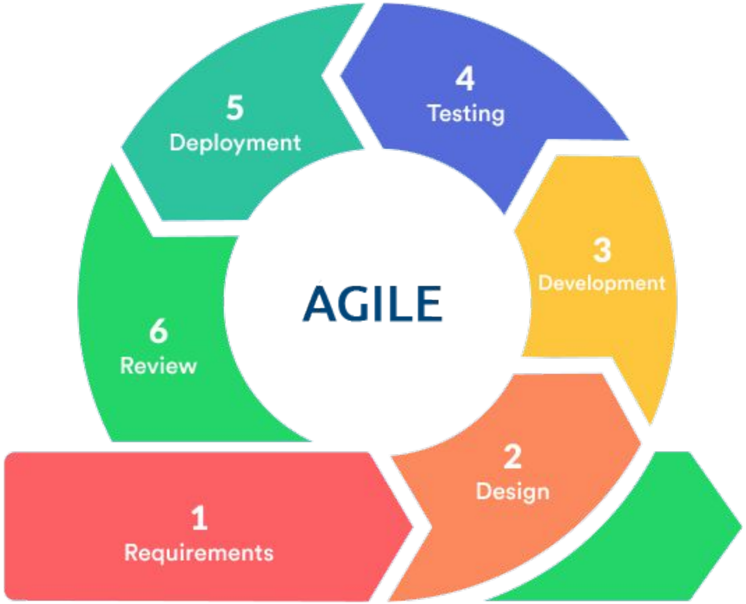
Comparison and Issues

What is **Agile** methodology?

Agile is an **iterative approach** to project management and software development that helps teams **deliver value to their customers faster** and with fewer headaches.

Instead of betting everything on a "big bang" launch, an agile team delivers work in **small, but consumable, increments**.

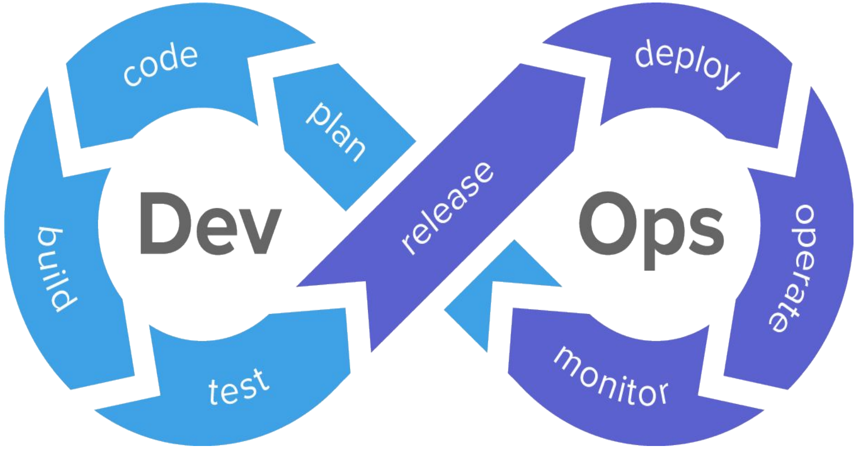
Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for **responding to change quickly.**



From Agile to **DevOPS**

DevOps is an approach to software development that enables teams to **build, test, and release software faster and more reliably by incorporating agile principles and practices**, such as **increased automation and improved collaboration between development and operations teams**.

Development, testing, and deployment occur in both agile and DevOps. Yet traditional agile stops short of operations, which is an integral part of DevOps



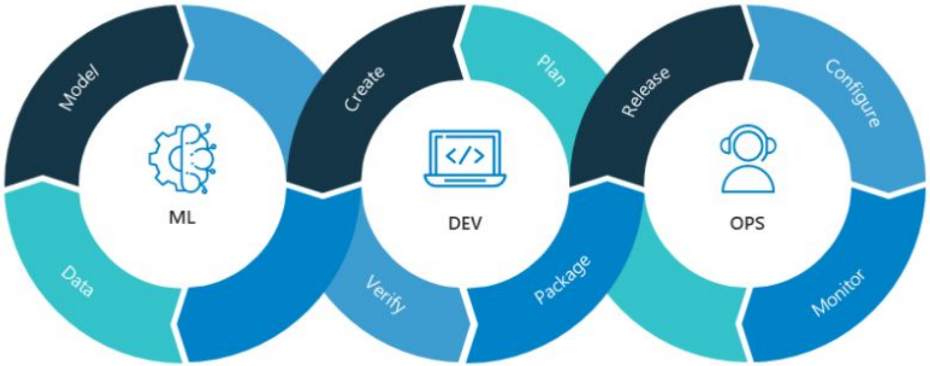
What would happen if we applied **DevOps** principles to **Machine Learning?**

MLOps

MLOps is a set of practices that **apply DevOps principles to machine learning problems** to ensure the creation of **machine learning products** that you can verify, trust, and maintain in the long term.

There are many **shared elements** between DevOps and MLOps such as:

- **Code and component testing**
- **Automation**
- **Continuous integration of code**
- **Continuous delivery into production**
- **Cross-Team collaboration**
- **Integration of feedbacks loops**



MLOps vs DevOps: **Main Differences**

	DevOps	MLOps
Goal	Automate software quality assurance checks and feedback loops	Support the Machine Learning Lifecycle with automated quality checks
Componentes	Continuous integration + Continuous Delivery	Continuous training + Continuous Validation
Deployment cycles	Frequent iterations	Long, continuous and resource hungry (re)training cycles
Team composition	SWD + QA Engineer + Devops	Data Scientist + ML Engineer + Data Engineer
Main deliverable	Code, executable	Model, Data, Metadata, Training parameters
Quality assurance	Code Tests	Data and Model Validation

MLOps: Issues and Challenges

The addition of **Data** and **Machine Learning models**, which an MLOps process must address compared to a DevOPS process, has caused some **issues** and **challenges** :

- **Model Drift**
- **Data Drift**

MODEL DRIFT

A deployed model is based on a definition of **what the business case needs**. These needs may **evolve**.

For example, if you were detecting credit card fraud and the business evolved its thinking it will need a rethink or retraining of the model.

DATA DRIFT

This is when you train a model on the demographics of a set of users and now you are observing that **the population it is being utilized on doesn't match the same demographic**.

Other examples of data drift is based on **change of the data** due to **seasonality,** changes in **consumer preferences,** the **addiction of new products** ecc. **MLops** can listen to these changes and trigger **automated retraining of the models**

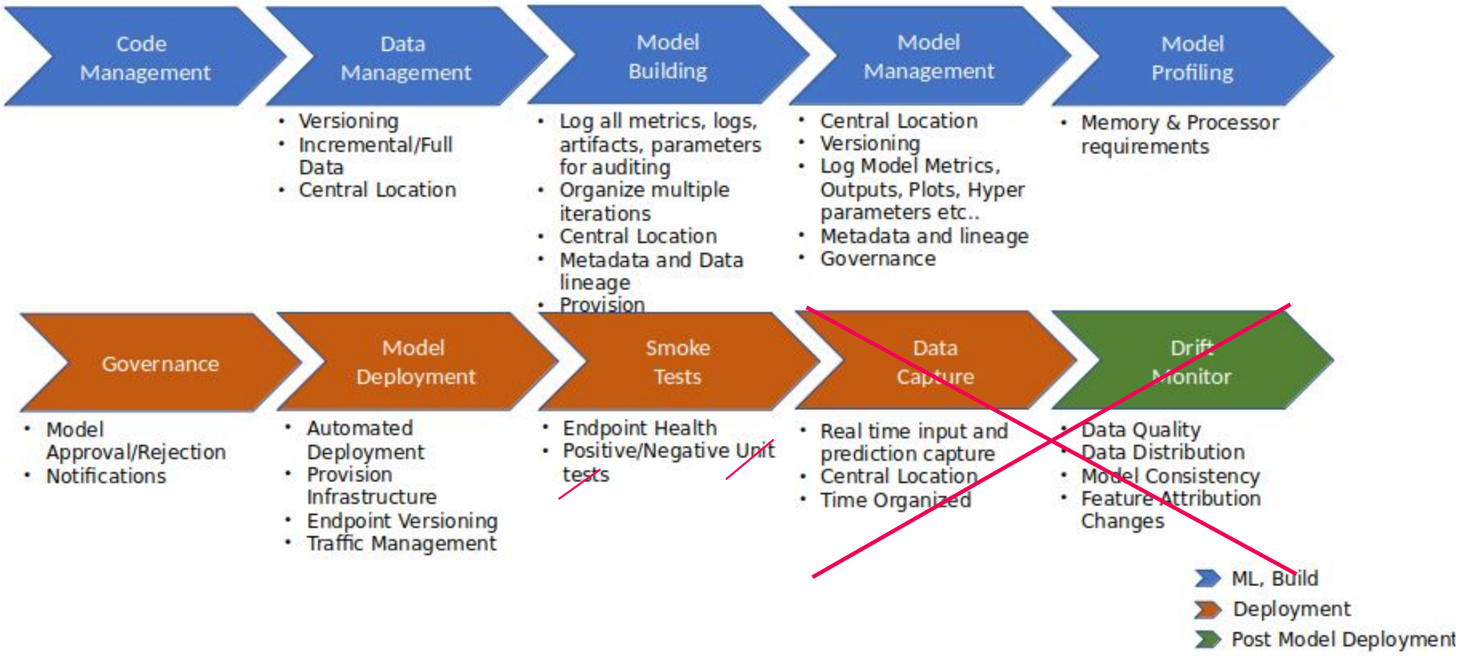
MLOps Models

How to build a fully automated training and deployment pipeline on Azure

The **models** of our demo

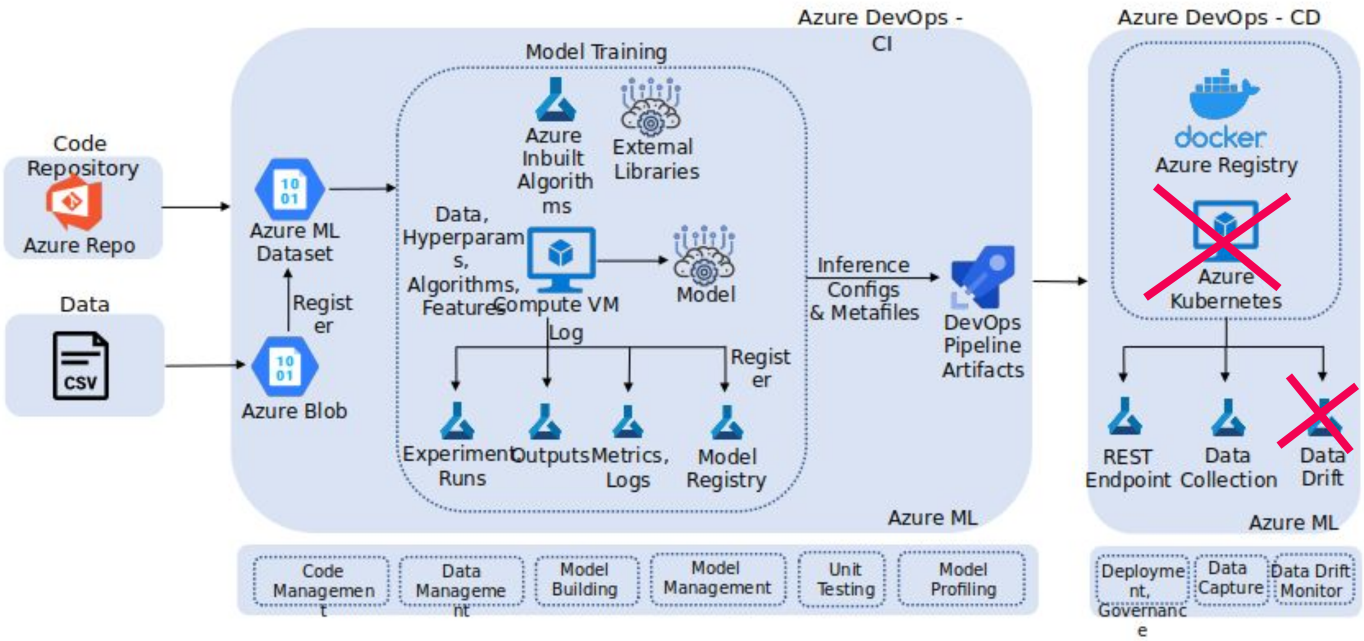
MLOps (ML Operations)

- MLOps is an end-to-end life cycle management of an ML Project



Our Hands-On Flow

Azure MLOps + DevOps

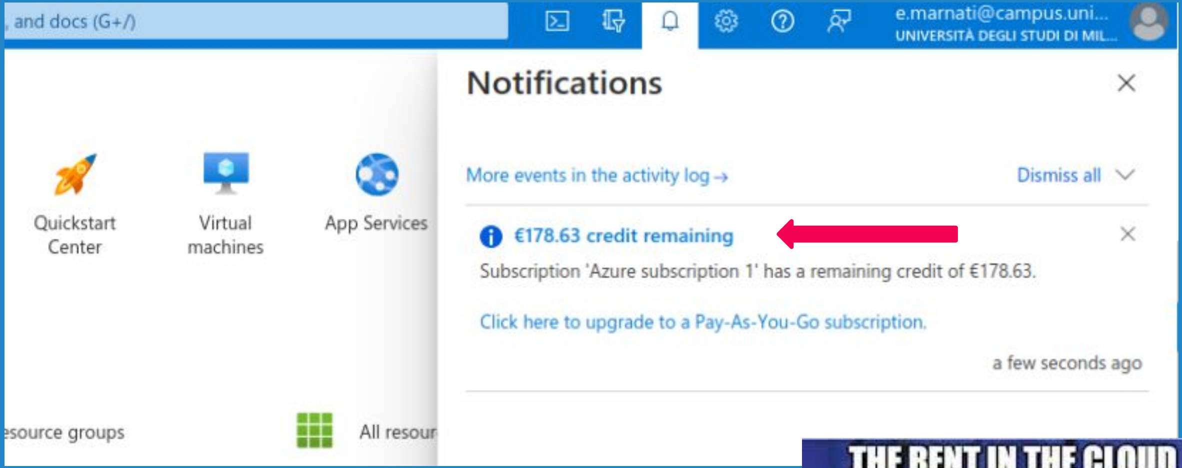


Hands-On

AZURE DevOps + MICROSOFT AZURE MACHINE LEARNING STUDIO

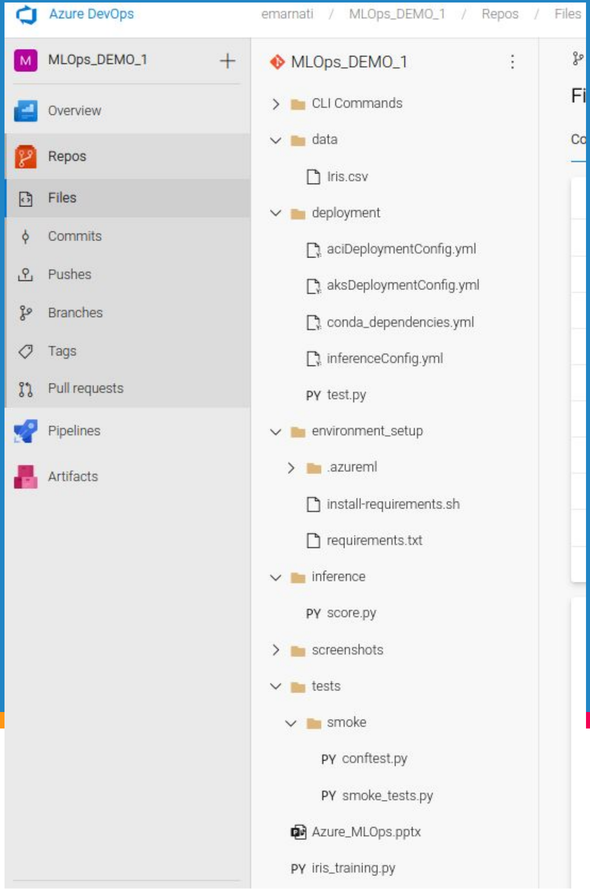
Initial Settings

- Microsoft Corp kindly gives you 200 $ credit
- Monthly free access for basic function on Azure (no Kubernetes or MLFlow + integration)
- The credit quickly reduces



Get some Classification Task: IRIS Database

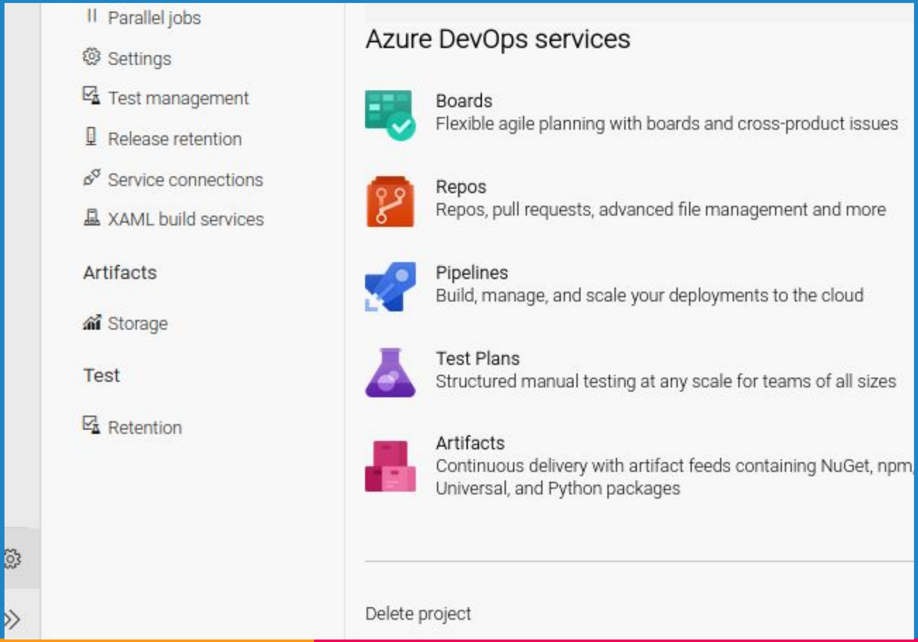
- Import some **python code** from outside Azure
- Choose a basic binary classification task (IRIS Dataset)
- Simply select a **GitHub repo** to interact with
- This project is static in time and therefore **no data drift can be assessed**



Many thanks Srijith!

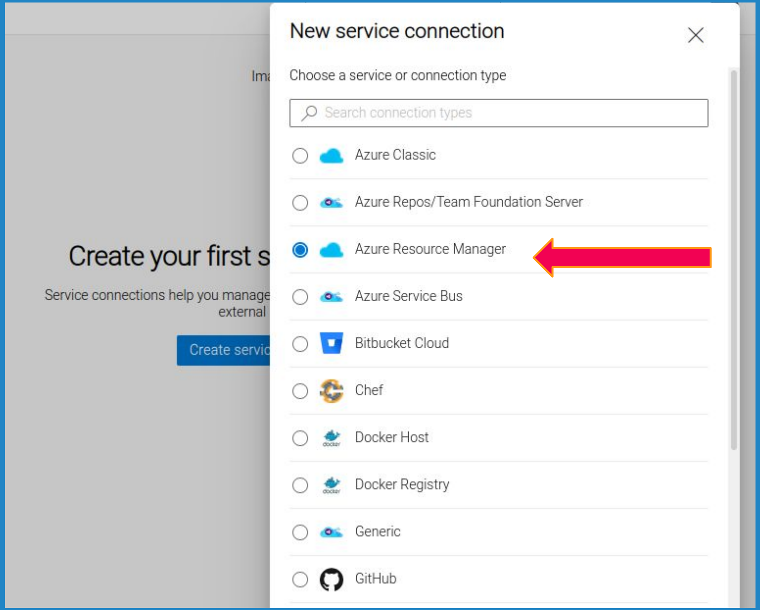
Select Azure DevOps Services

- **Repos:** enable connection between some code outside (you can connect directly with Visual Studio / VSCode)
- **Pipelines:** is the process of connected different phases mainly with Azure CLI (Azure Command Line Interface)
- **Artifact:** will be our ML product, ready to deploy



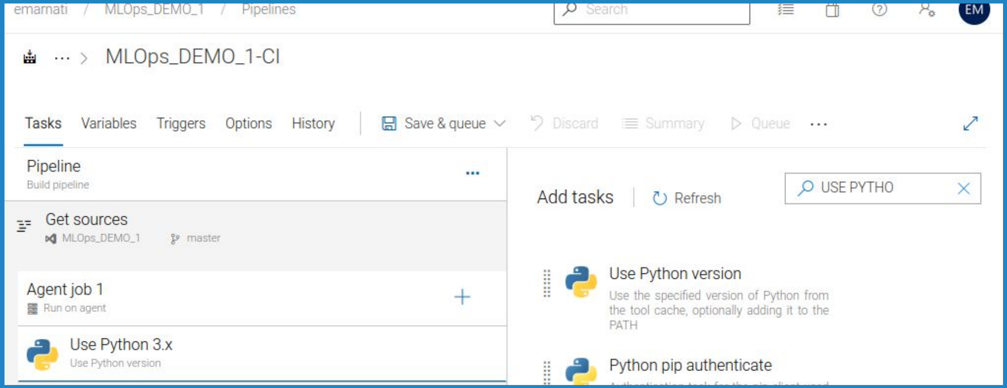
Azure Resource Manager

- It is important to create a service principle (selecting a resource group)
- It will be extensively used into CI pipeline and CD too
- Without service principles we will not be able to authenticate our pipeline



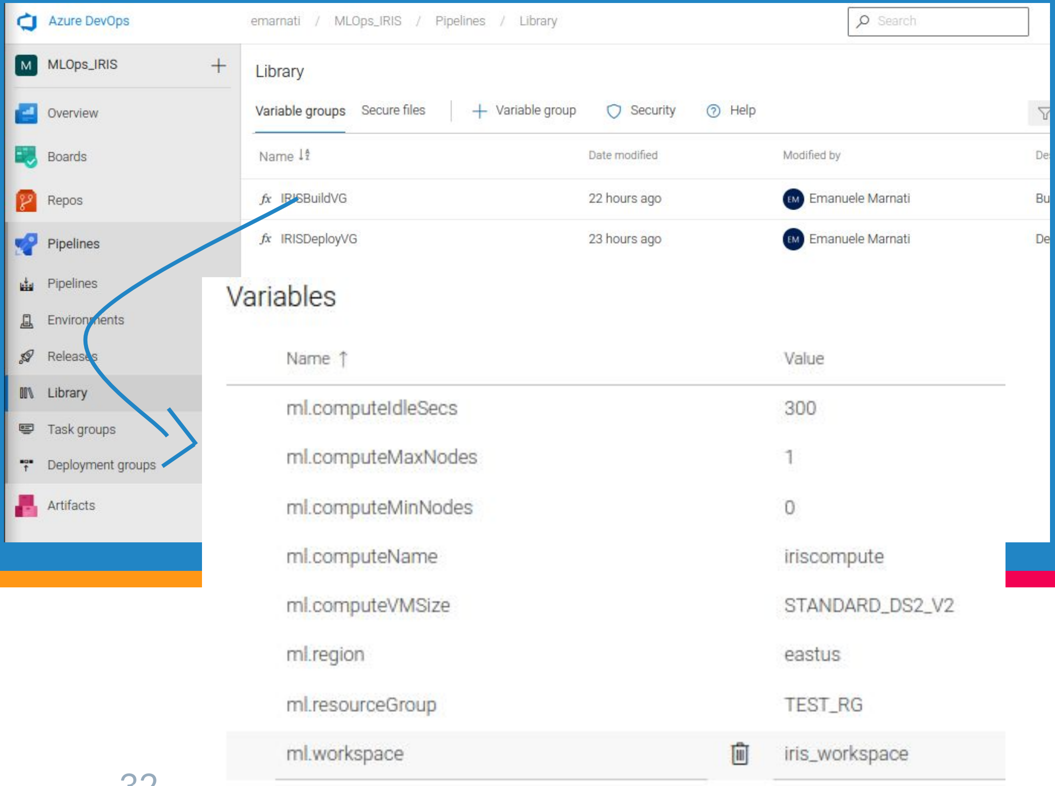
Start Up with Python Settings

- You have different AGENT JOB
- The ration is: “The simplest way” so we won’t integrate python anymore just bash and shell
- A DevOps person without knowing python will still be able to operate



The Library

- **Variables are arguments which are getting passed to the CI pipeline**
- **You can avoid duplication of variables**
- **You can here specify: the compute machine, the vm, the region, the resource group, the workspace name**



GitHub Code Overview

- create_pipeline function

- get_files_from_datastore function

- We use the parameters from Azure (it's all harmoniously integrated)

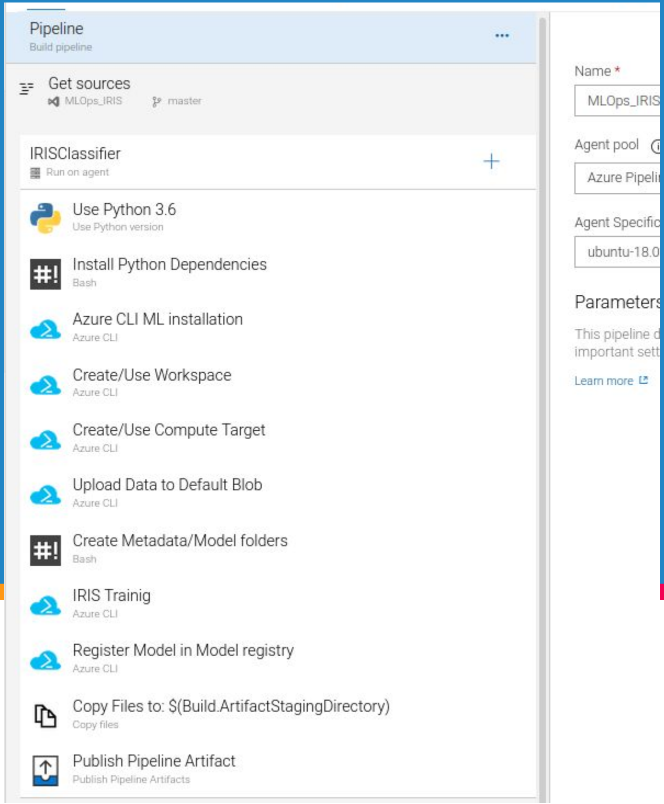
```python
ef get_files_from_datastore(self, container_name, file_name):
    '''
    Get the input CSV file from workspace's default data store
    Args :
        container_name : name of the container to look for input CSV
        file_name : input CSV file name inside the container
    Returns :
        data_ds : Azure ML Dataset object
    '''
    datastore_paths = [(self.datastore, os.path.join(container_name,file_name))]
    data_ds = Dataset.Tabular.from_delimited_files(path=datastore_paths)
    dataset_name = self.args.dataset_name
    if dataset_name not in self.workspace.datasets:
        data_ds = data_ds.register(workspace=self.workspace,
                        name=dataset_name,
                        description=self.args.dataset_desc,
                        tags={'format': 'CSV'},
                        create_new_version=True)
    else:
        print('Dataset {} already in workspace '.format(dataset_name))
    return data_ds
```

```python
    return data_ds

ef create_pipeline(self):
    '''
    IRIS Data training and Validation
    '''
    self.datastore = Datastore.get(self.workspace, self.workspace.get_default_datastore(
    print("Received datastore")
    input_ds = self.get_files_from_datastore(self.args.container_name,self.args.input_cs
    final_df = input_ds.to_pandas_dataframe()
    print("Input DF Info",final_df.info())
    print("Input DF Head",final_df.head())

    X = final_df[["SepalLengthCm","SepalWidthCm","PetalLengthCm","PetalWidthCm"]]
    y = final_df[["Species"]]

    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=1984)

    model = DecisionTreeClassifier()
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print("Model Score : ", model.score(X_test,y_test))

    joblib.dump(model, self.args.model_path)

    self.validate(y_test, y_pred, X_test)

    match = re.search('([^\/]*)$', self.args.model_path)
    # Upload Model to Run artifacts
    self.run.upload_file(name=self.args.artifact_loc + match.group(1),
                         path_or_stream=self.args.model_path)
```

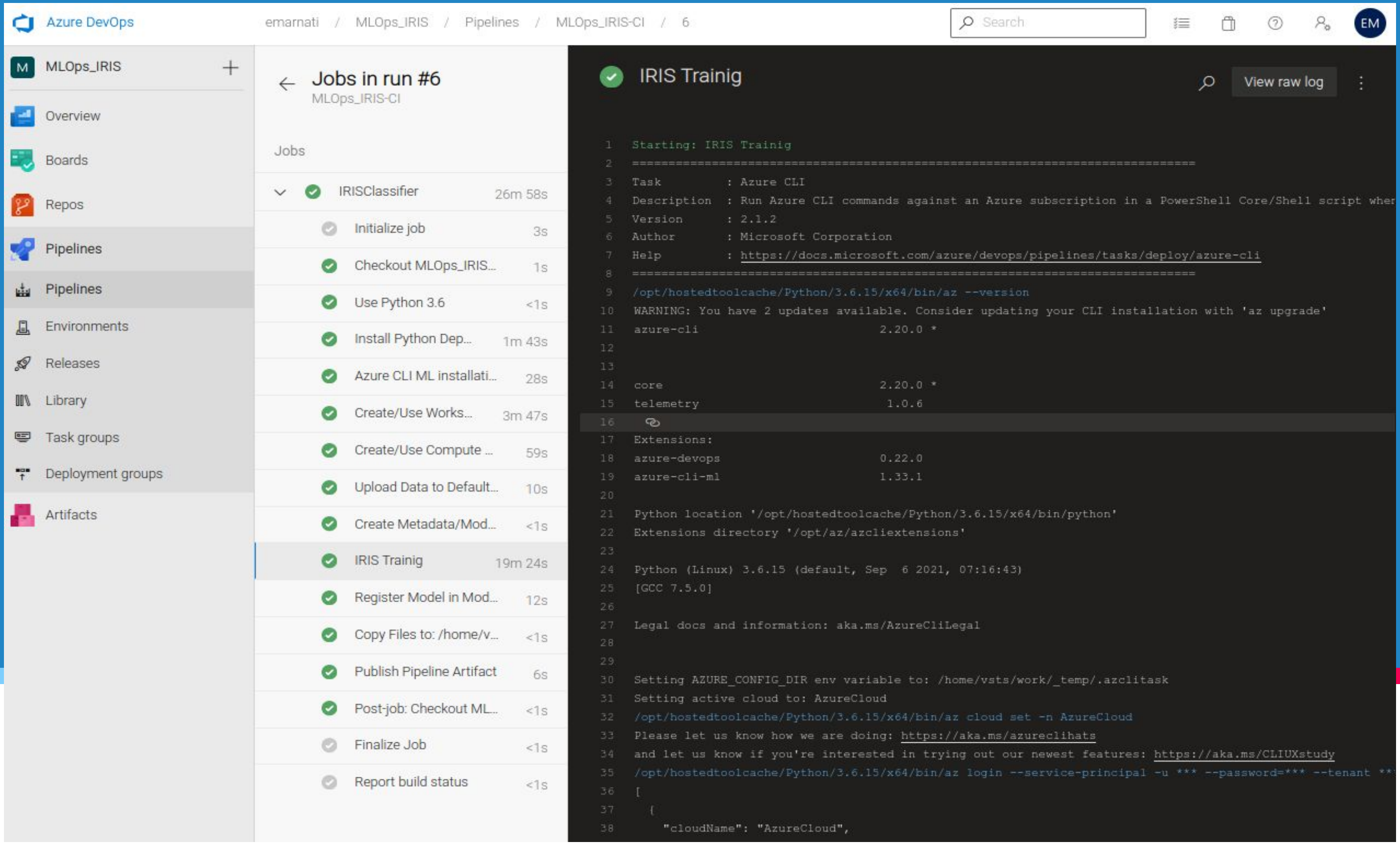
THE FINAL BUILD PIPELINE

Components

- Install the **dependencies**
- **Azure CLI** ML installation
- Create the **workspace**
- Set the compute **target machine** for the training script
- Upload the .csv file into an **Azure Blob Storage**
- Create metadata folder
- Execute the script and **register the model**
- Pick the **artifact** and put it inside the CD pipeline (next phase)
- Create metadata folder



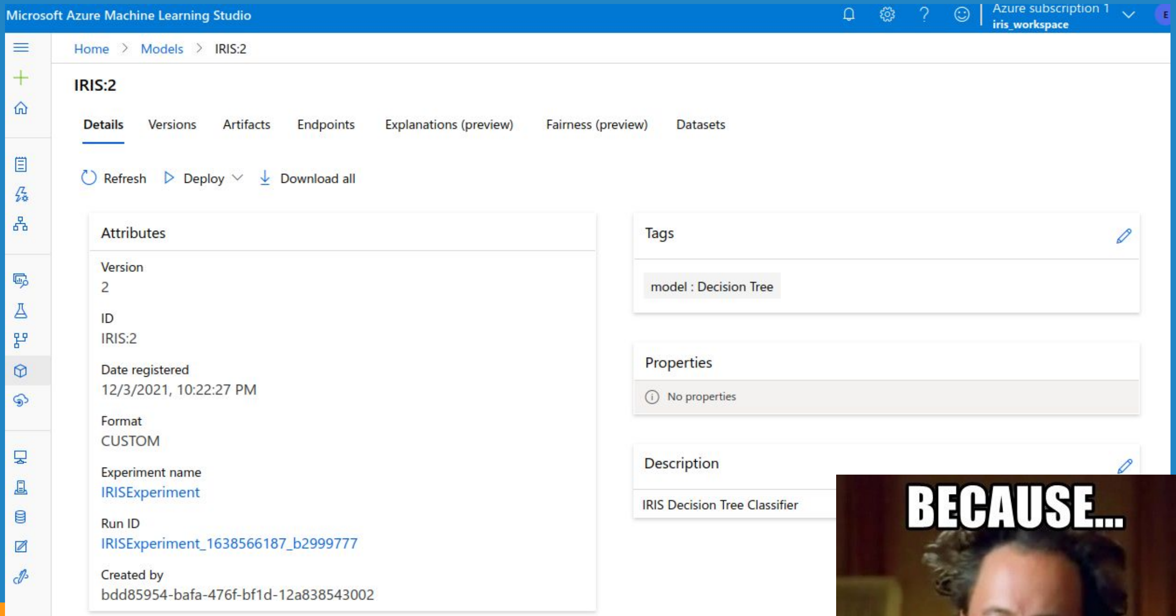
Dig deeper into container Log – **LIVE RUNNING**



Where is the ML Experiment?

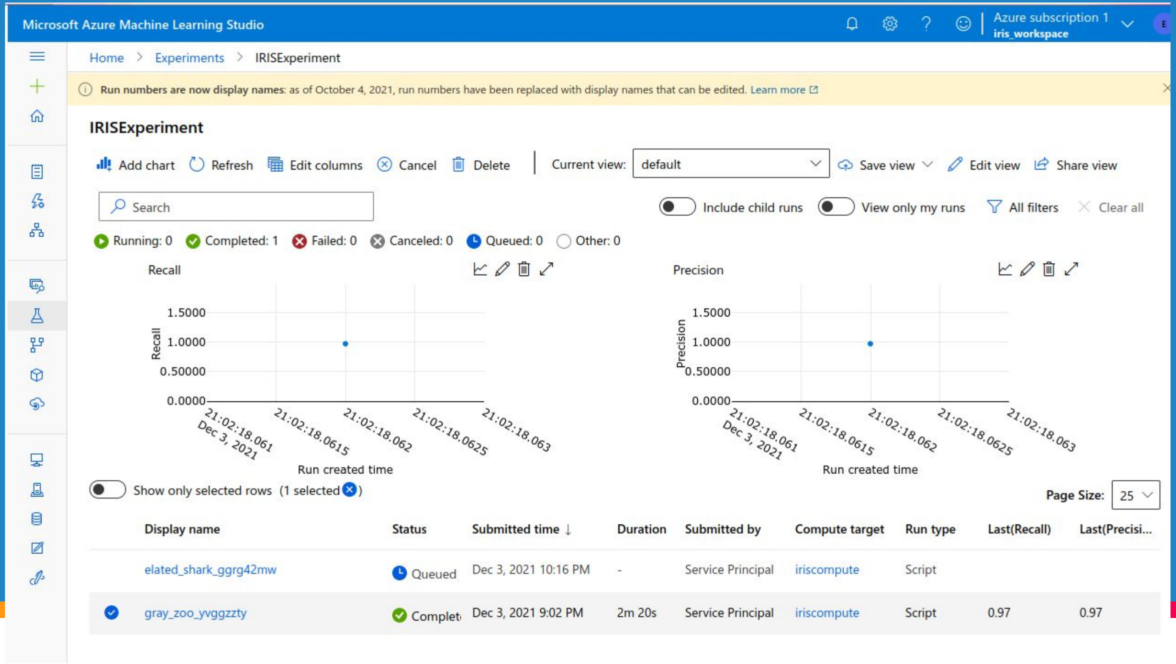
So far we have seen a **DevOps environment**, but with one click we can switch to the **Microsoft Azure Machine Learning Studio** environment.

Experiment: is the main component, logical grouping of all your runs (different iterations of your model training)



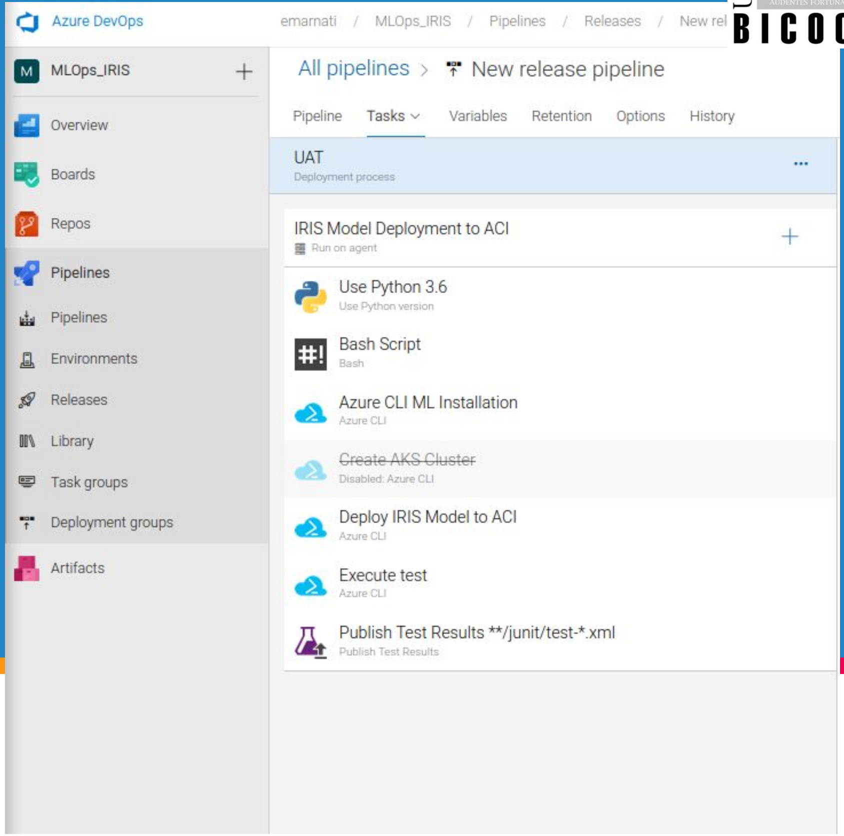
Where are the **ML Metrics?**

- **All the metrics are here logged and monitored**
- **You can plot a graph with recall, precision, accuracy**
- **You can confront all these metrics between different runs**



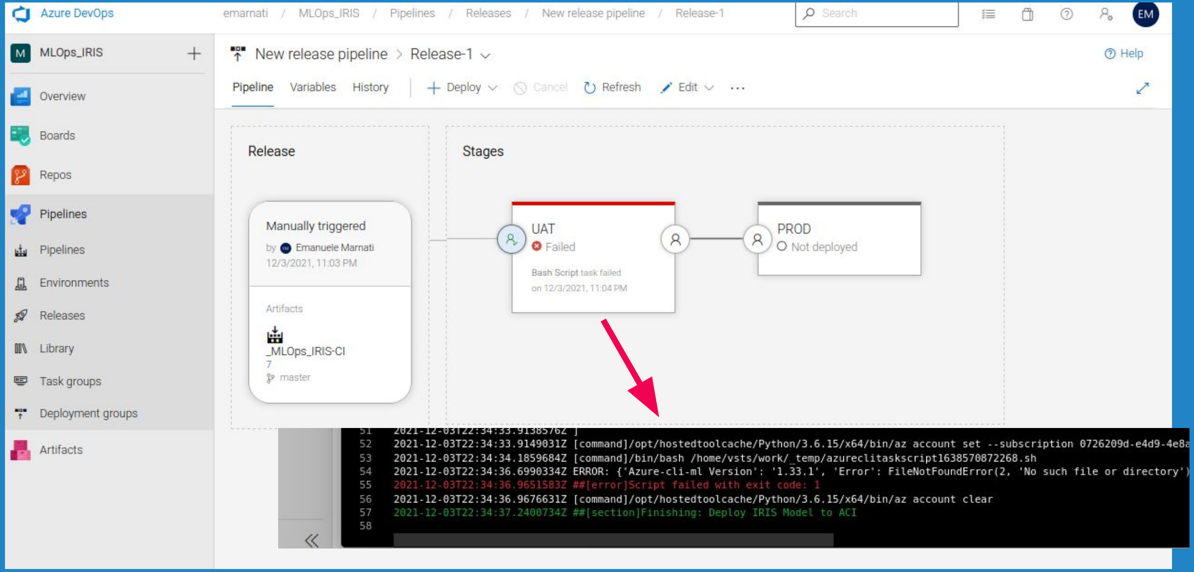
Deploy Pipeline - Release

- Similar process as with the build part
- Some dependencies installation first
- We cannot use Azure Kubernetes Service for deploy, so we use Azure Container Registry
- Create metadata folder
- In production you shouldn't use Container Registry, is not reliable!
- We perform the smoke test and publish its result



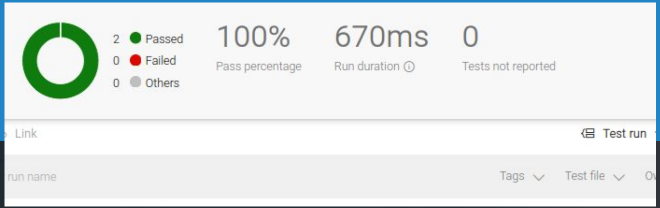
Release Workflow - Testing - Production

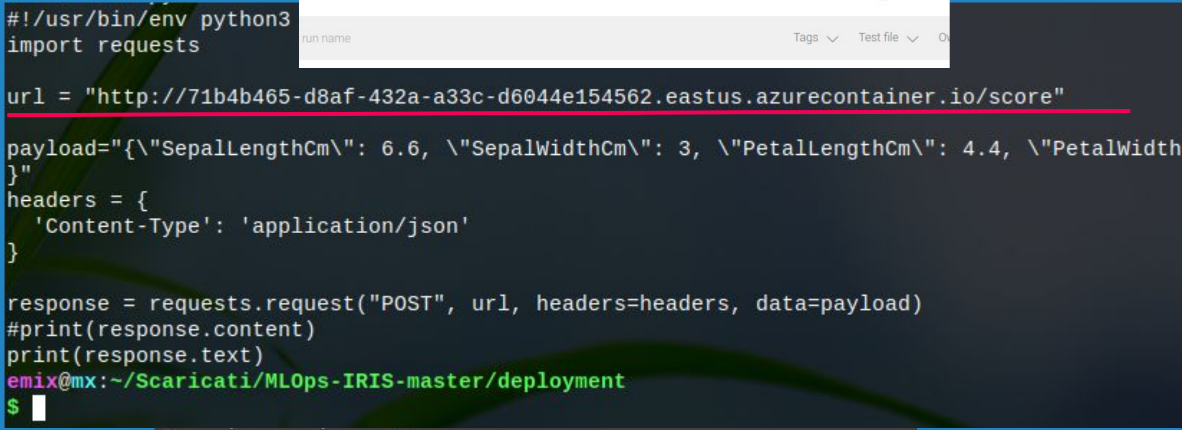
- The process is held by **Azure CLI Machine Learning extension**
- We trigger the artifact, **test and deploy it**

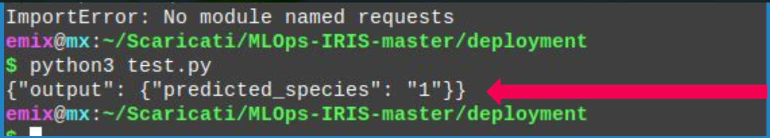


How hands my test?

- After some **tries and effort**, the test is okay
- With our script connected to the **Endpoint** we run **test.py**
- The predicted species is **"1"**

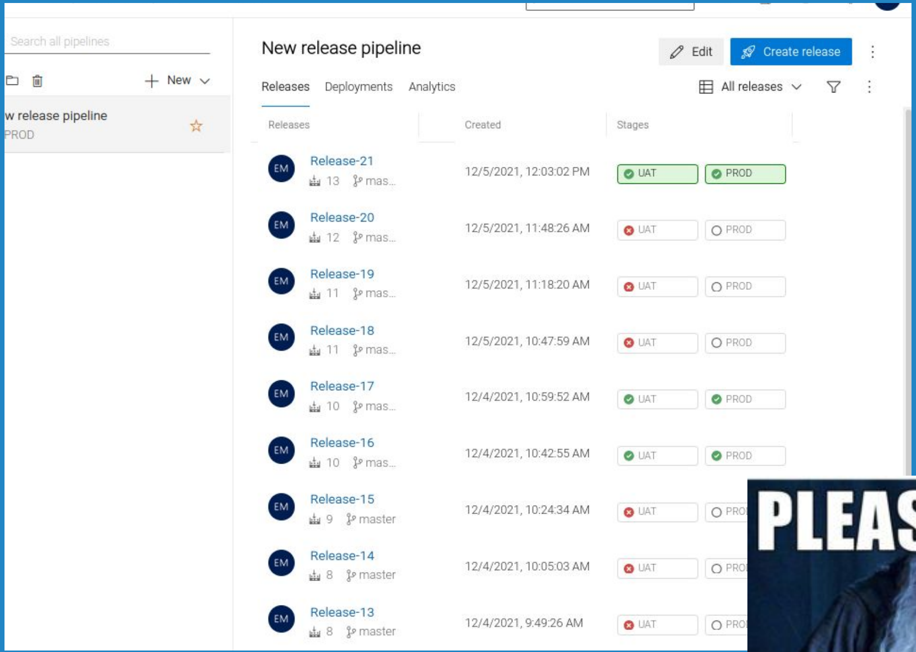






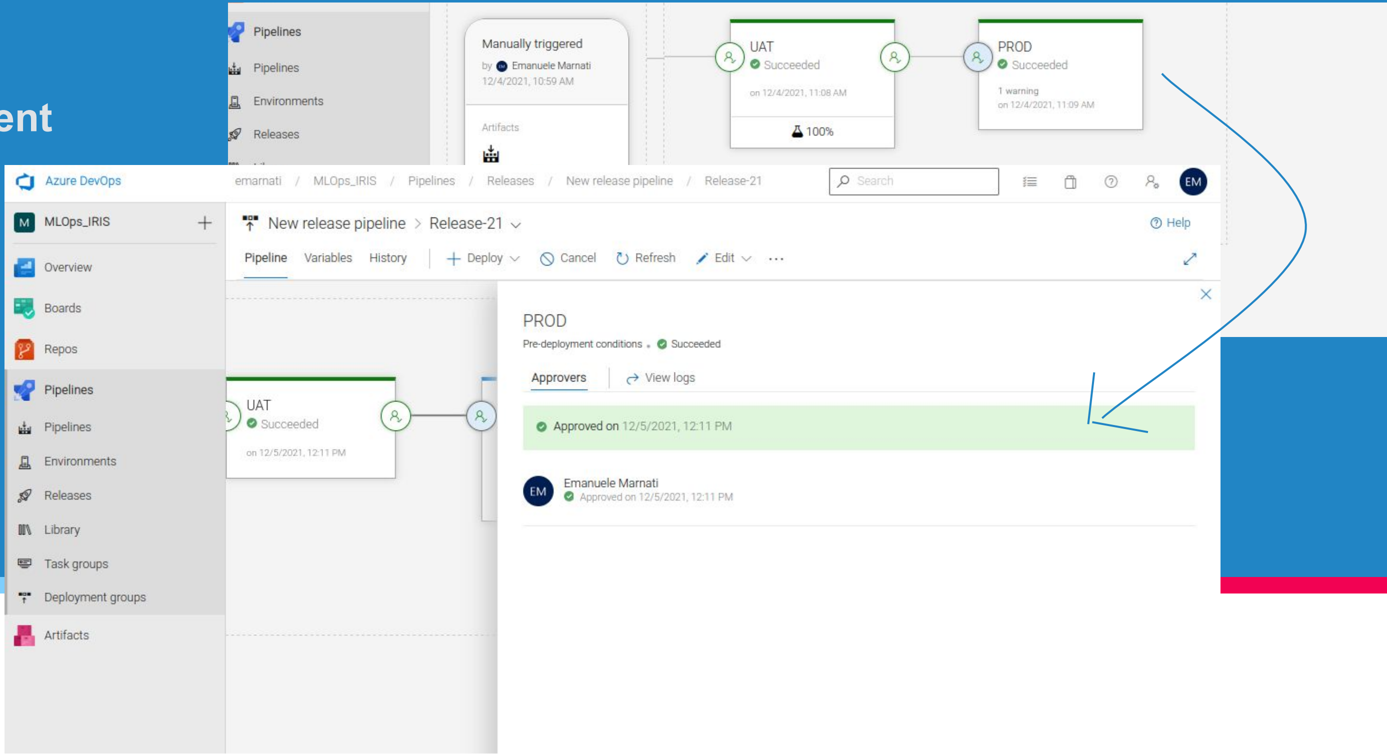
Release

After some troubleshooting we have our accepted release!



Users and management has Approved

- **Multiplayer platform**
- **Immediate Assessment**

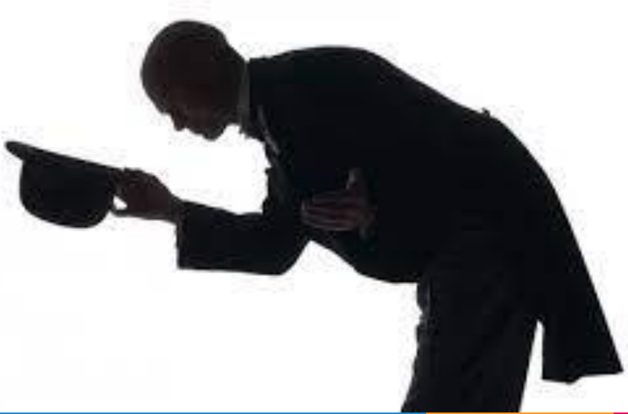


Conclusion and Further Improvements

- Real Time Continuous Data Ingestion
- Azure Kubernetes Service
- Data Drift integration (not with IRIS Dataset)
- Data Quality Assessment
- Integrated Learning
- Continuous Learning

Thanks for your **Attention!**

- **Check out the code and presentation on GitHub: https://github.com/emixstream/MLOps-On-Azure-**



References & Bibliography

- https://doi.org/10.1007/978-1-4842-6549-9
- https://deepchecks.com/how-is-mlops-different-from-devops-a-detailed-comparison/
- https://www.atlassian.com/devops/what-is-devops/agile-vs-devops
- https://www.youtube.com/watch?v=ZVWg18AXXuE
- https://www.youtube.com/watch?v=pLd7xF0z5Zs
- https://github.com/srijiths/MLOps-IRIS
- https://www.youtube.com/watch?v=1BSwYlJUxK0&list=PLZoTAELRMXVOk1pRcOCaG5xtXxgMalpIe