

MLOps: When DevOps meets Machine Learning

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

Table of contents

- ▷ Introduction
- ▷ Why MLOps?
- ▷ Three different Setups
- ▷ MLOps vs DevOps vs Agile
- ▷ Issues and Challenges
- ▷ Hands-on Session's Models
- ▷ Hands-on Overview

MLOps

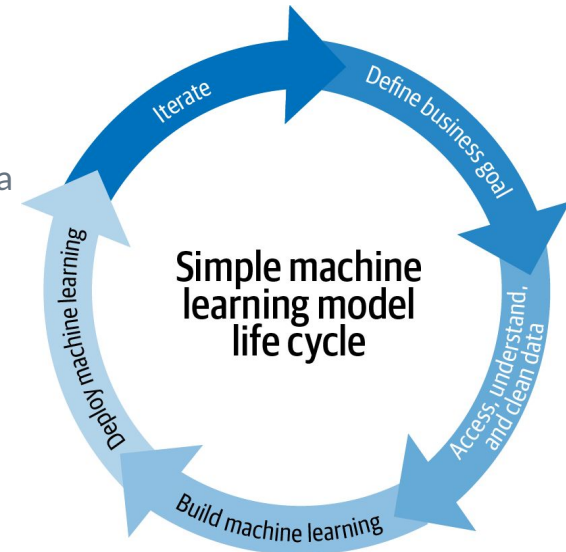
Aims and Needs

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**.

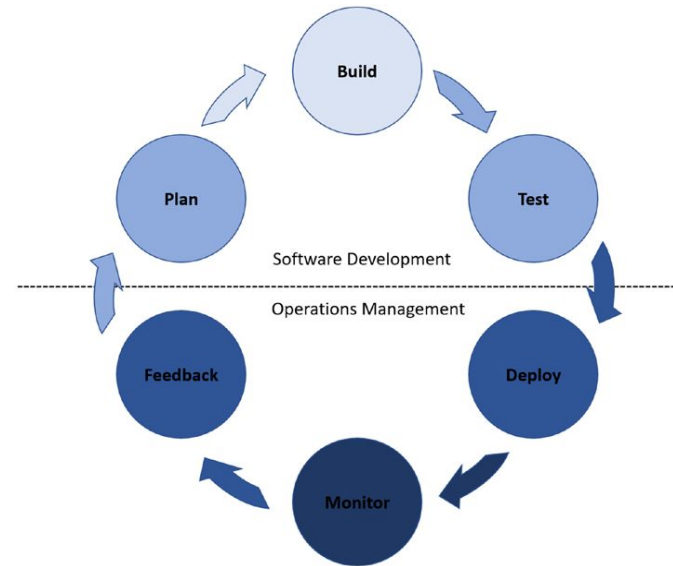
- ▷ **The entire cycle is time consuming:** updating the model on new data patterns and trends, it is a problem which can take up hours of manual labor that can be better spent elsewhere.
- ▷ **Cost Increase:** worsening the overall maintenance costs because the costs for deployed machine learning models 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

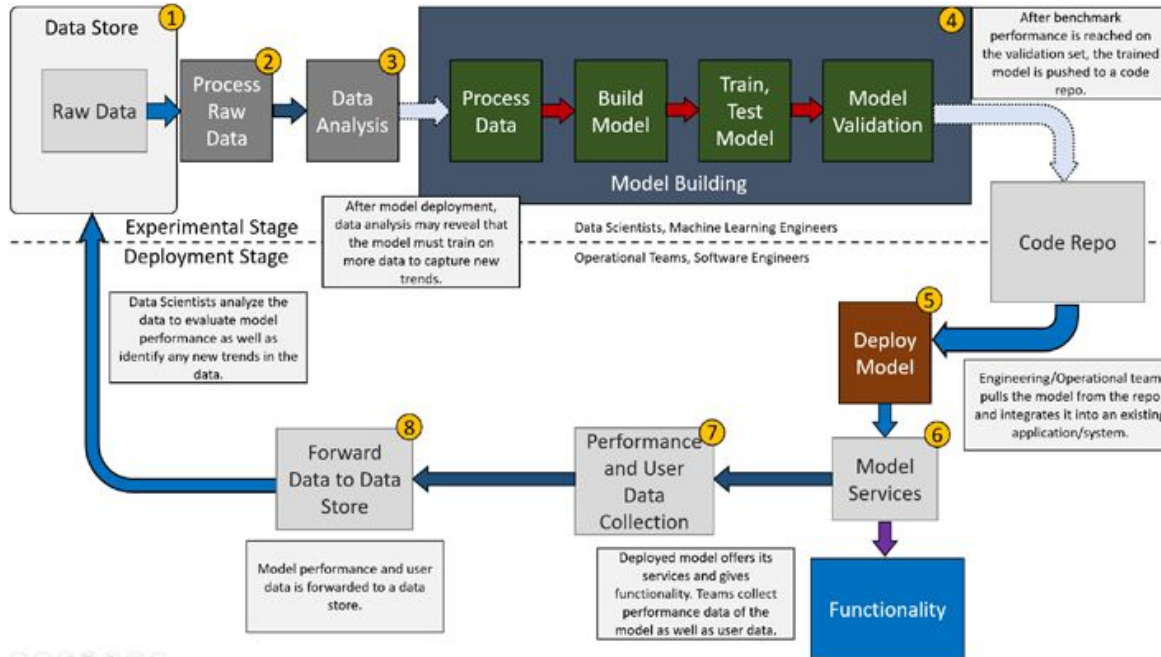
Refers to a setup where **there are no MLOps principles applied** and everything is manually implemented.

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.



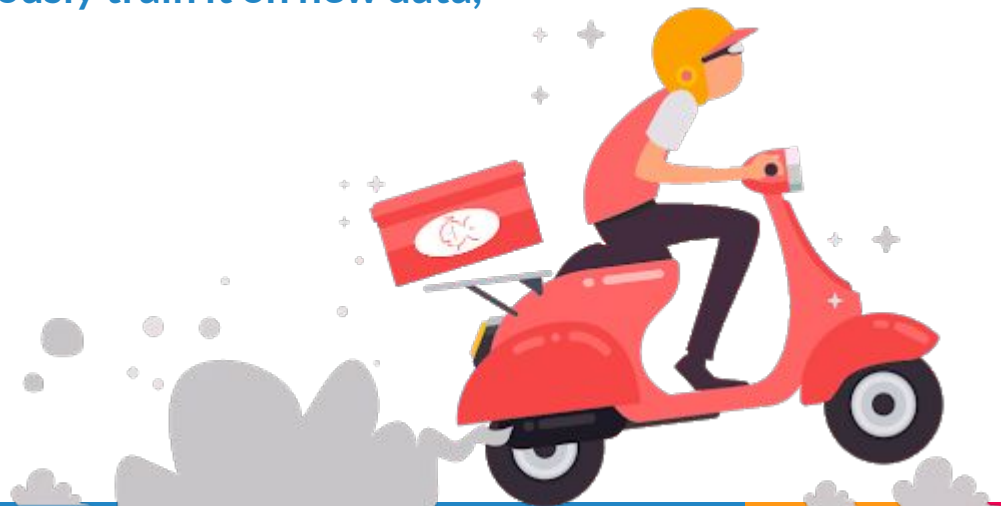
How it works!



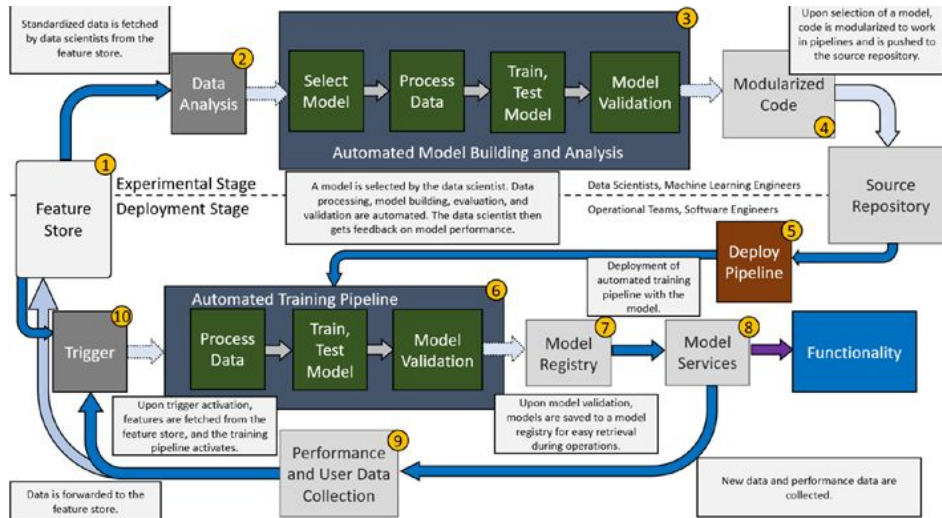
Continuous Model Delivery

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.

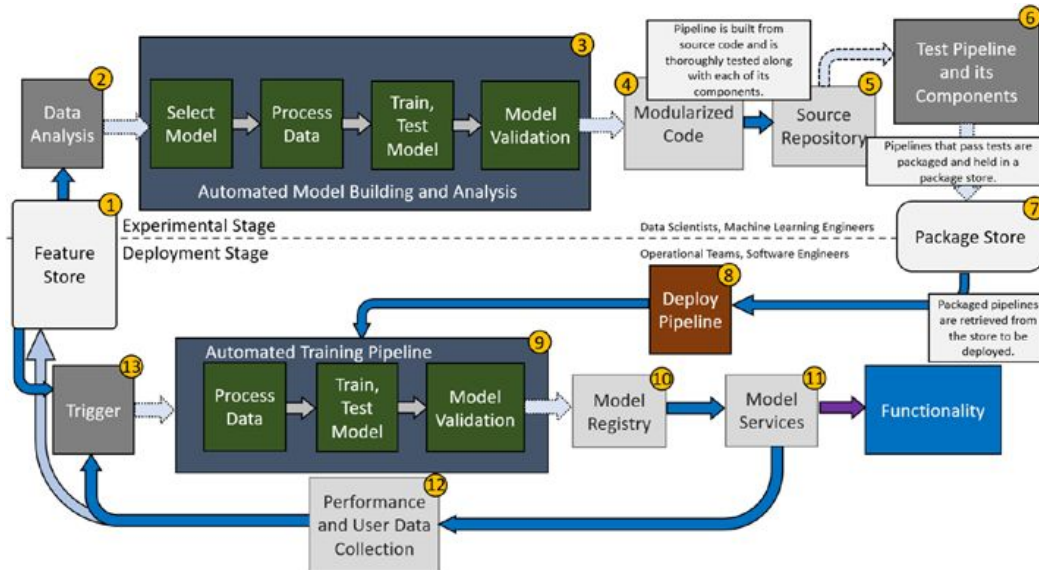
With the barriers to rapid deployment lifted (the tediousness of manual work in the experimental stage) by automation, **models can now be created or updated at a much faster pace**.

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**.

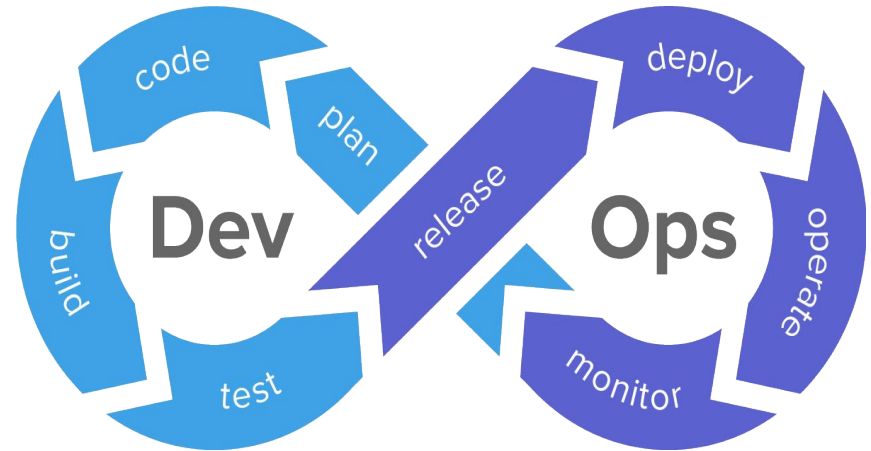
Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for **responding to change quickly**.



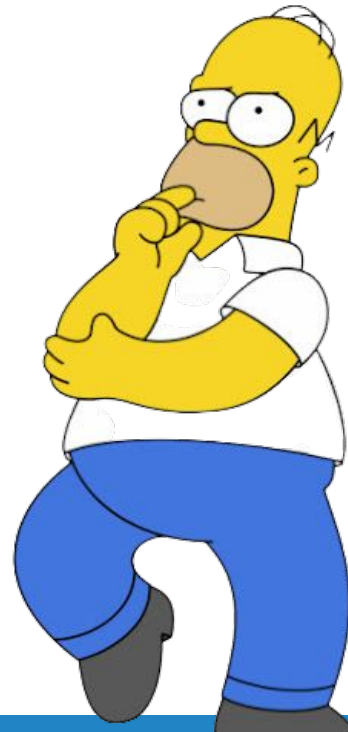
From Agile to DevOps

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?

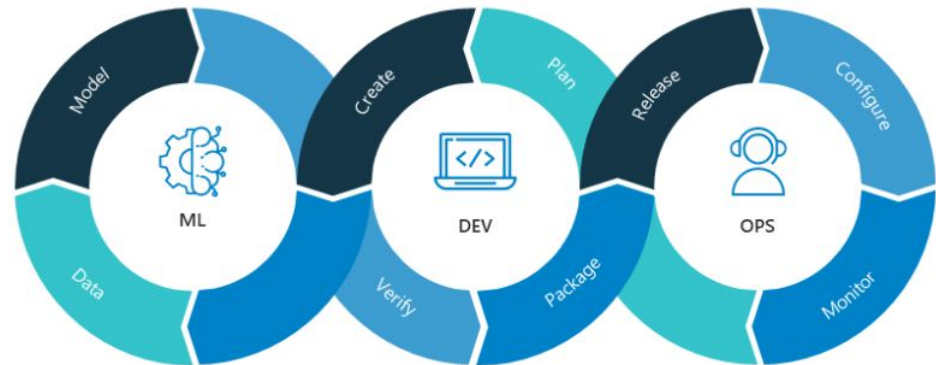


MLOps

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 **addition of new products** ecc. **MLops** can listen to these changes and trigger **automated retraining of the models**

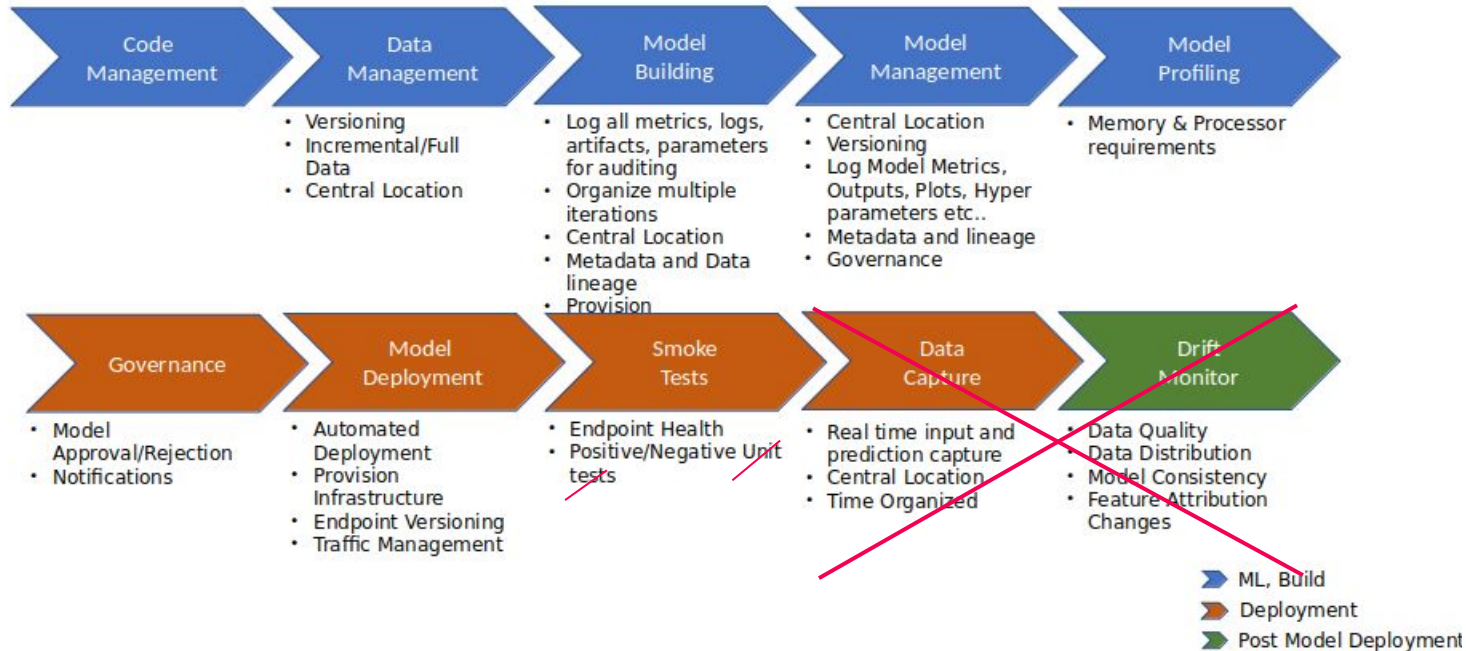
MLOps Models

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

The **models** of our demo

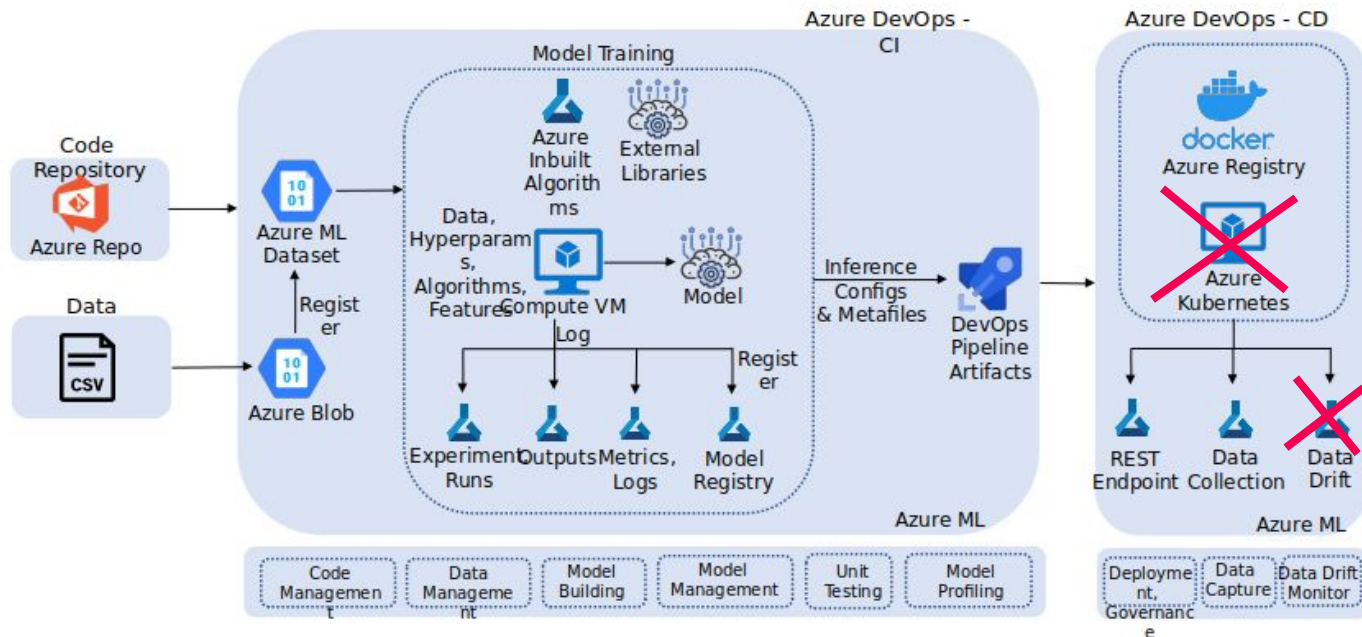
MLOps (ML Operations)

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



Our Hands-On Flow

Azure MLOps + DevOps

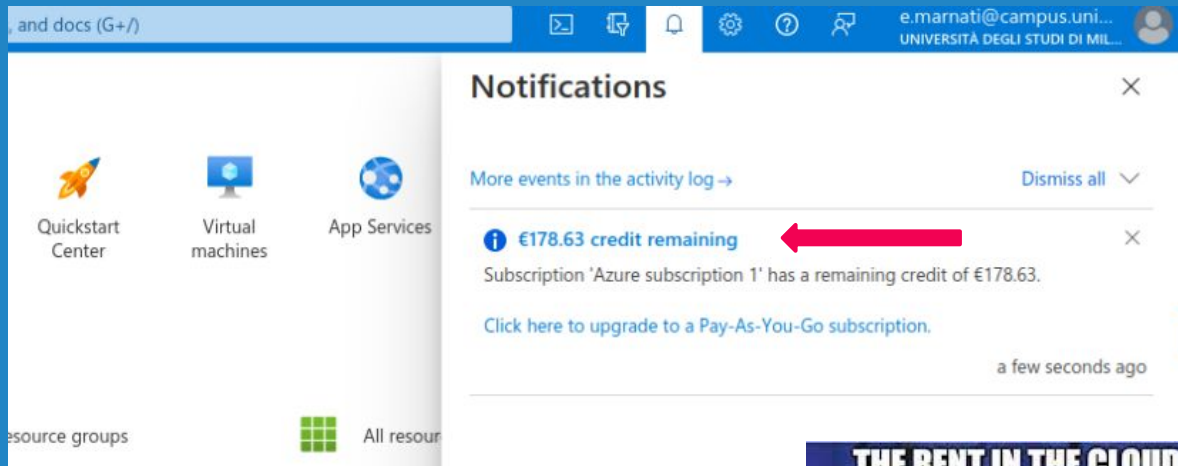


Hands-On

AZURE DevOps + MICROSOFT AZURE MACHINE LEARNING STUDIO

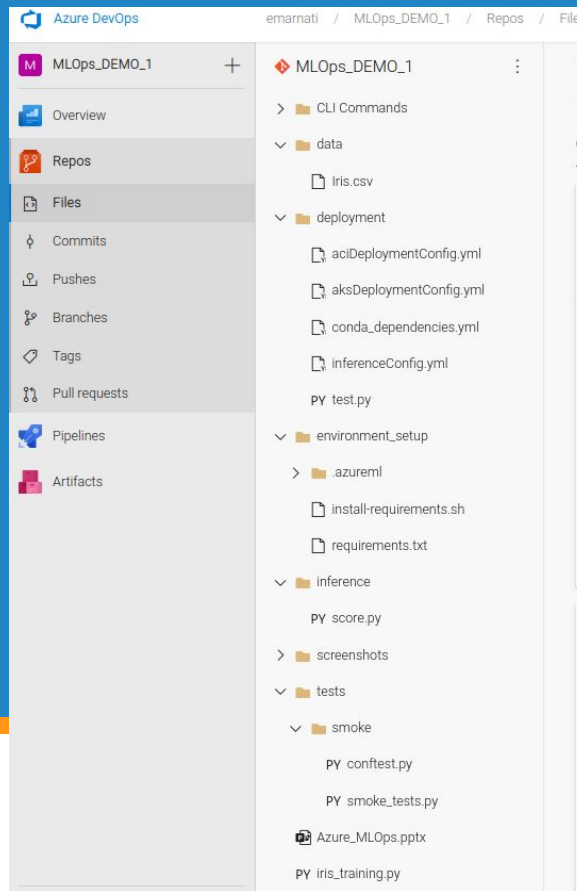
Initial Settings

- 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

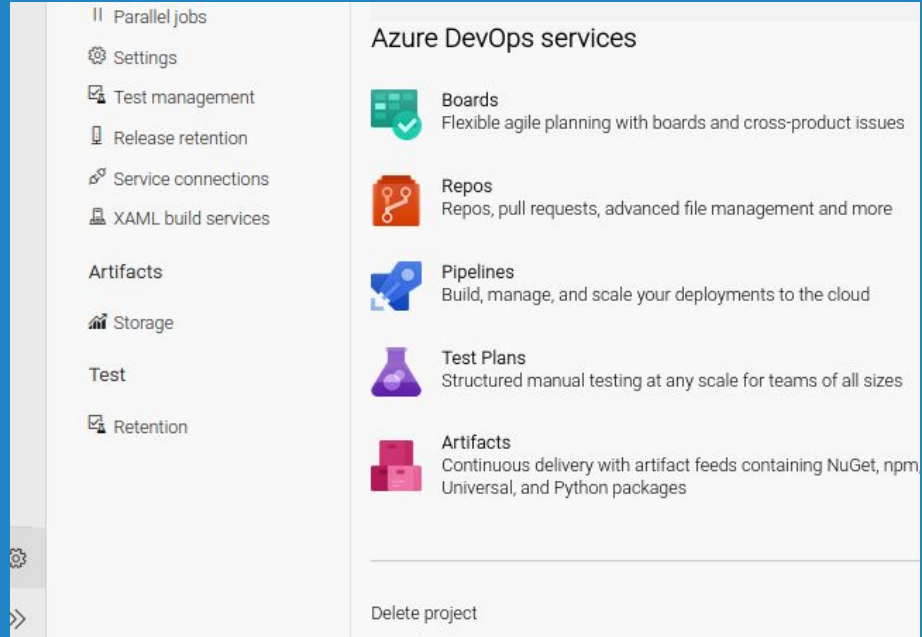


Many thanks Srijith!



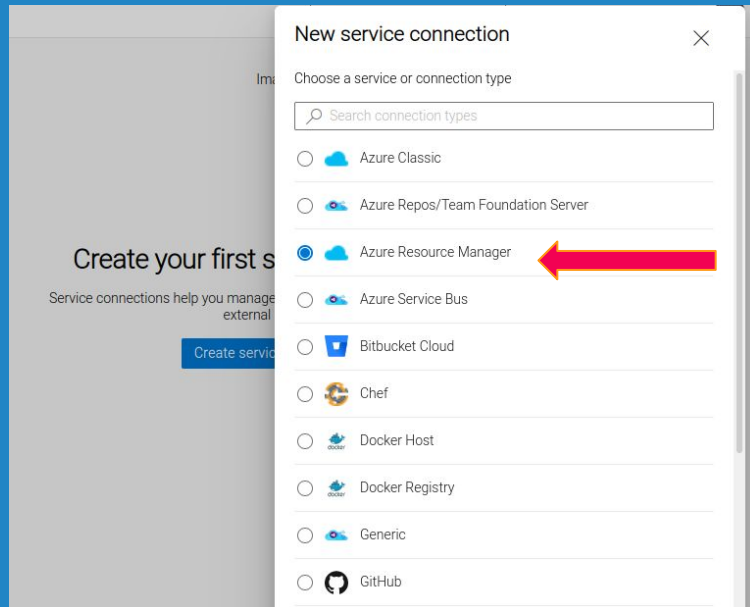
Select Azure DevOps Services

- **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



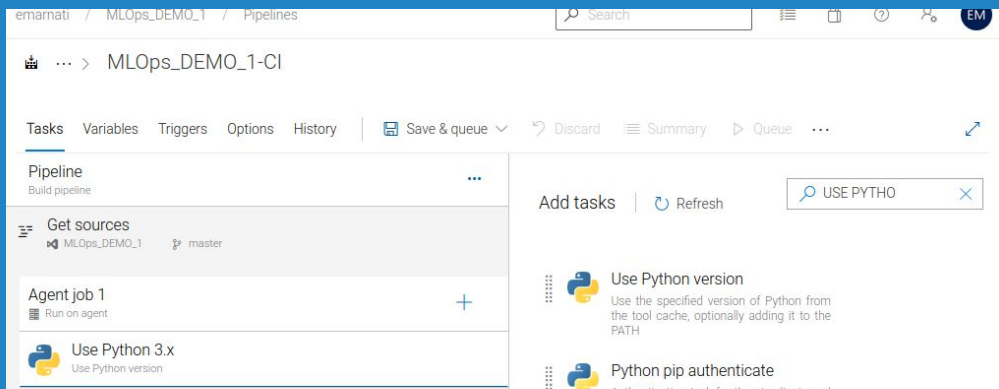
Azure Resource Manager

- 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



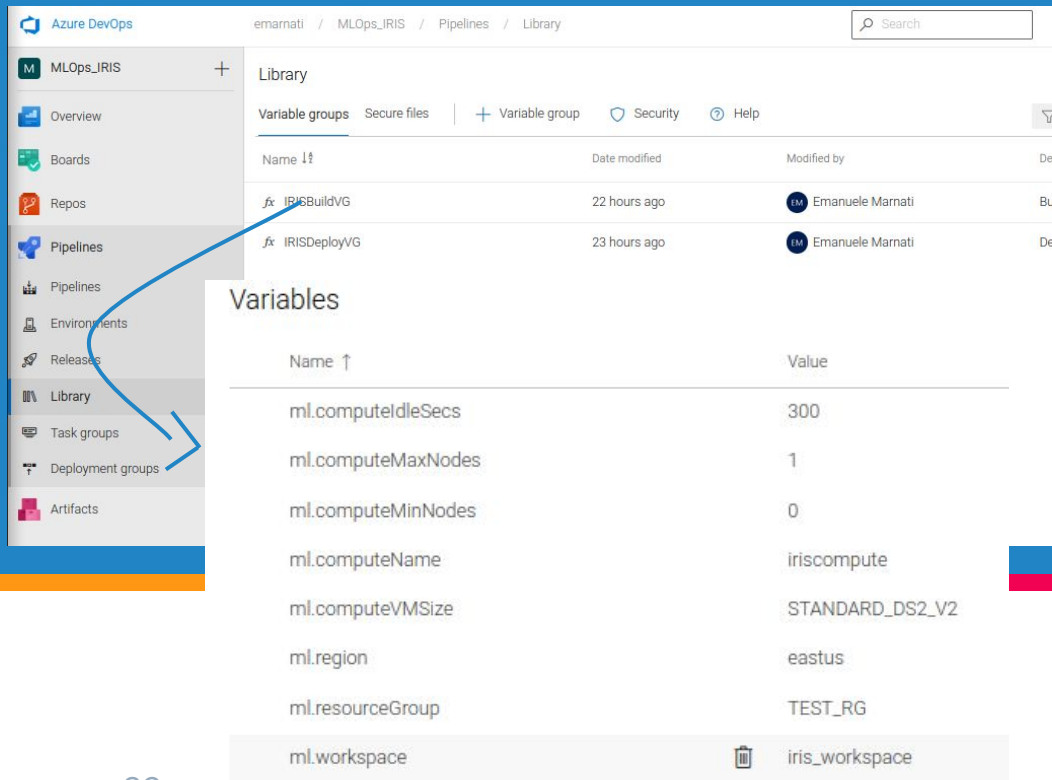
Start Up with Python Settings

- 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



Azure DevOps interface showing the Library and Variables sections.

Library

Name	Date modified	Modified by
fx IRISBuildVG	22 hours ago	Emanuele Marnati
fx IRISDeployVG	23 hours ago	Emanuele Marnati

Variables

Name	Value
ml.computeIdleSecs	300
ml.computeMaxNodes	1
ml.computeMinNodes	0
ml.computeName	iriscompute
ml.computeVMSize	STANDARD_DS2_V2
ml.region	eastus
ml.resourceGroup	TEST_RG
ml.workspace	iris_workspace

GitHub Code Overview

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

```
def 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
```

```
return data_ds

def 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_csv)
    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)

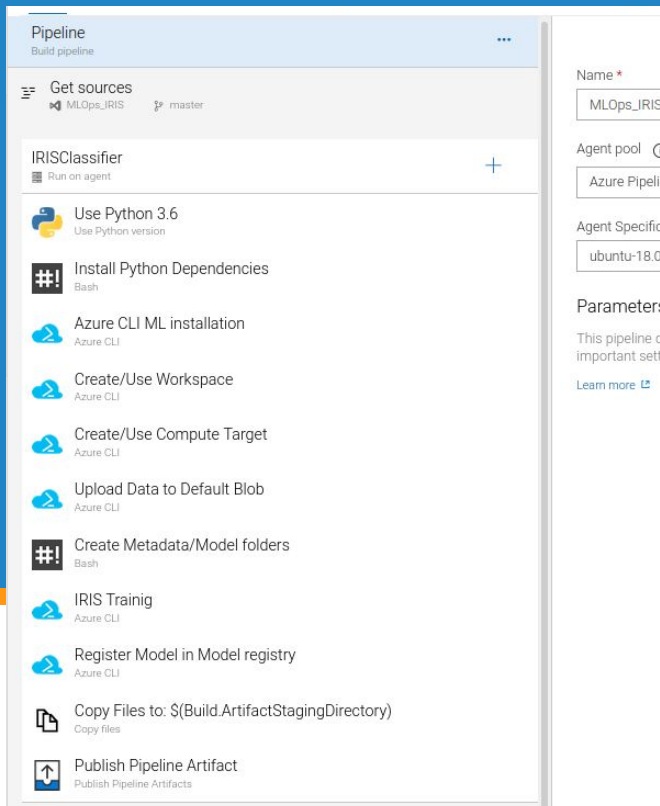
    match = re.search('([^\/]*)$', self.args.model_path)
    # Upload Model to Run artifacts
    self.run.upload_file(name=self.args.artifact_loc + match.group(1),
                        path_or_stream=self.args.model_path)
```

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

The screenshot displays the Azure DevOps web interface. On the left, a sidebar contains navigation links: Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, and Artifacts. The main area is titled 'Jobs in run #6' for the 'MLOps_IRIS-CI' pipeline. A list of jobs is shown, including 'IRISClassifier' (26m 58s), 'Initialize job' (3s), 'Checkout MLOps_IRIS...' (1s), 'Use Python 3.6' (<1s), 'Install Python Dep...' (1m 43s), 'Azure CLI ML installati...' (28s), 'Create/Use Works...' (3m 47s), 'Create/Use Compute ...' (59s), 'Upload Data to Default...' (10s), 'Create Metadata/Mod...' (<1s), 'IRIS Trainig' (19m 24s), 'Register Model in Mod...' (12s), 'Copy Files to: /home/v...' (<1s), 'Publish Pipeline Artifact' (6s), 'Post-job: Checkout ML...' (<1s), 'Finalize Job' (<1s), and 'Report build status' (<1s). The 'IRIS Trainig' job is selected, and its log is displayed on the right. The log shows the execution of the 'IRIS Trainig' task, which runs Azure CLI commands against an Azure subscription. The log includes details about the task, its description, version, author, and help. It also shows the output of the 'az cloud set' command, which sets the active cloud to 'AzureCloud'. The log ends with a JSON object containing the 'cloudName' property set to 'AzureCloud'.

```
1 Starting: IRIS Trainig
2 =====
3 Task : Azure CLI
4 Description : Run Azure CLI commands against an Azure subscription in a PowerShell Core/Shell script wher
5 Version : 2.1.2
6 Author : Microsoft Corporation
7 Help : https://docs.microsoft.com/azure/devops/pipelines/tasks/deploy/azure-cli
8 =====
9 /opt/hostedtoolcache/Python/3.6.15/x64/bin/az --version
10 WARNING: You have 2 updates available. Consider updating your CLI installation with 'az upgrade'
11 azure-cli 2.20.0 *
12
13
14 core 2.20.0 *
15 telemetry 1.0.6
16
17 Extensions:
18 azure-devops 0.22.0
19 azure-cli-ml 1.33.1
20
21 Python location '/opt/hostedtoolcache/Python/3.6.15/x64/bin/python'
22 Extensions directory '/opt/az/azcliextensions'
23
24 Python (Linux) 3.6.15 (default, Sep 6 2021, 07:16:43)
25 [GCC 7.5.0]
26
27 Legal docs and information: aka.ms/AzureCliLegal
28
29
30 Setting AZURE_CONFIG_DIR env variable to: /home/vsts/work/_temp/.azclitask
31 Setting active cloud to: AzureCloud
32 /opt/hostedtoolcache/Python/3.6.15/x64/bin/az cloud set -n AzureCloud
33 Please let us know how we are doing: https://aka.ms/azureclihat
34 and let us know if you're interested in trying out our newest features: https://aka.ms/CLIUXstudy
35 /opt/hostedtoolcache/Python/3.6.15/x64/bin/az login --service-principal -u *** --password*** --tenant **
36 [
37 {
38   "cloudName": "AzureCloud",
```

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)

Microsoft Azure Machine Learning Studio

Home > Models > IRIS:2

IRIS:2

Details Versions Artifacts Endpoints Explanations (preview) Fairness (preview) Datasets

Refresh Deploy Download all

Attributes

Version	2
ID	IRIS:2
Date registered	12/3/2021, 10:22:27 PM
Format	CUSTOM
Experiment name	IRISExperiment
Run ID	IRISExperiment_1638566187_b2999777
Created by	bdd85954-bafa-476f-bf1d-12a838543002

Tags

model : Decision Tree

Properties

No properties

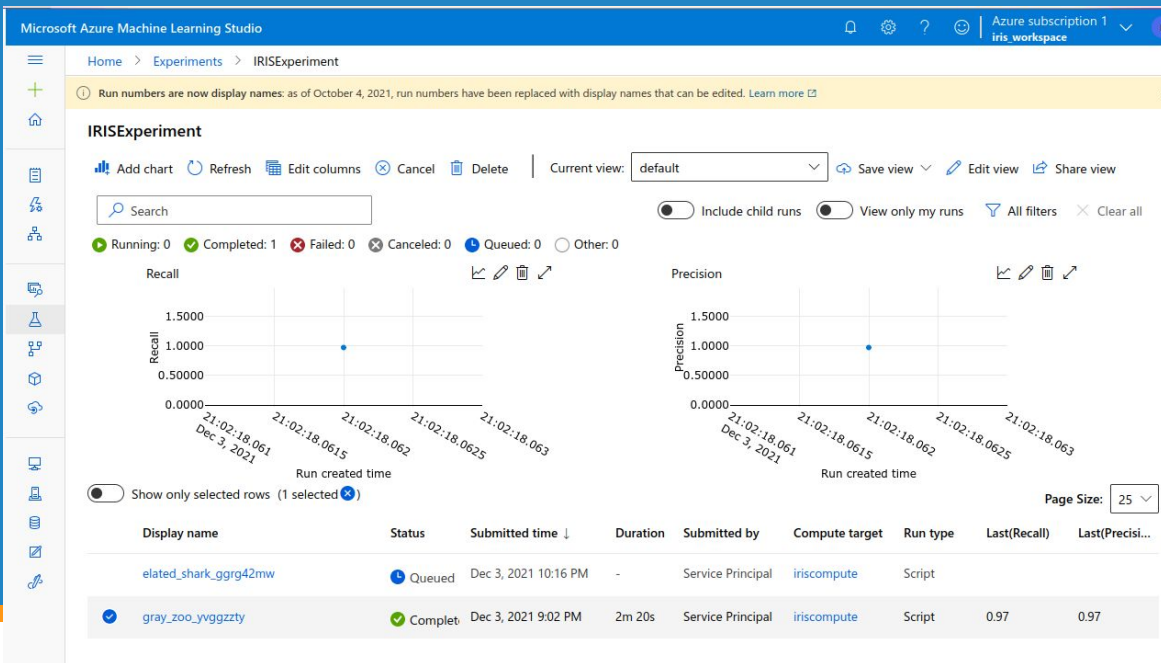
Description

IRIS Decision Tree Classifier



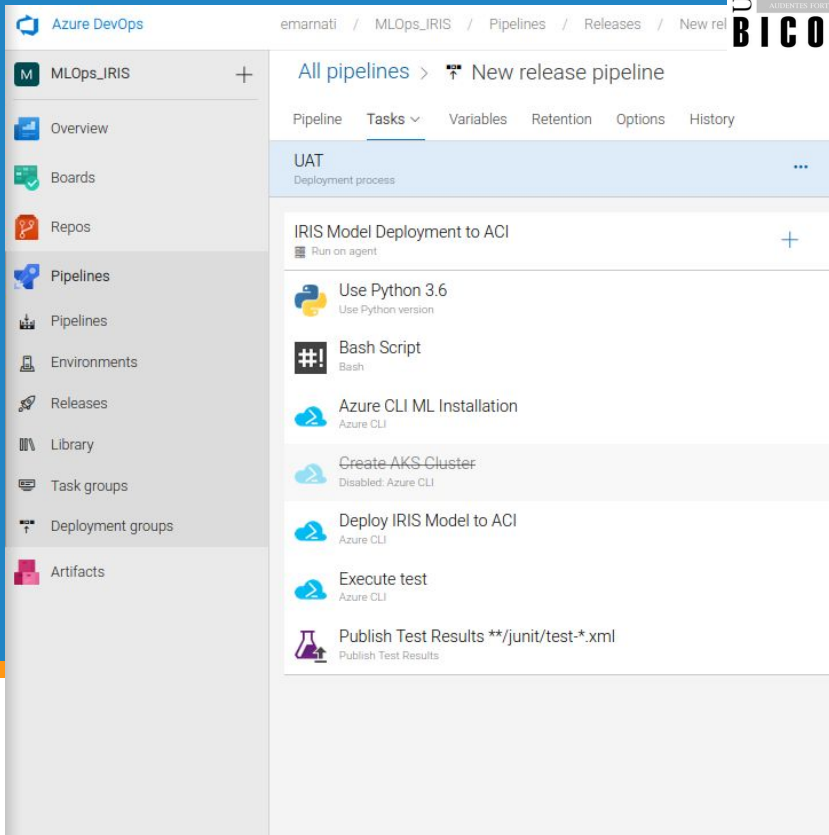
Where are the ML Metrics?

- 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

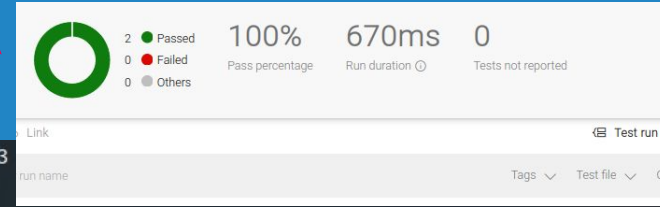
- The process is held by Azure CLI Machine Learning extension
- We trigger the artifact, test and deploy it

The screenshot displays the Azure DevOps interface for a release pipeline named 'Release-1' under the 'MLops_JRIS' project. The pipeline is triggered manually by Emanuele Marnati on 12/3/2021 at 11:03 PM. The 'Release' section shows an artifact named 'MLops_JRIS-CL' with version '7' and a 'master' branch. The 'Stages' section shows two stages: 'UAT' and 'PROD'. The 'UAT' stage is marked as 'Failed' with a message 'Bash Script task failed on 12/3/2021, 11:04 PM'. A red arrow points to the 'UAT' stage. The 'PROD' stage is marked as 'Not deployed'. The bottom of the screenshot shows a terminal window with logs indicating a failure in the 'UAT' stage due to a 'FileNotFoundError'.

```
51 2021-12-03T22:34:33.9138576Z [
52 2021-12-03T22:34:33.9149031Z [command]/opt/hostedtoolcache/Python/3.6.15/x64/bin/az account set --subscription 0726209d-e4d9-4e8a
53 2021-12-03T22:34:34.1859684Z [command]/bin/bash /home/vsts/work/_temp/azureclitaskscript1638570872268.sh
54 2021-12-03T22:34:36.6990334Z ERROR: {'Azure-cli-ml Version': '1.33.1', 'Error': FileNotFoundError(2, 'No such file or directory')}
55 2021-12-03T22:34:36.9651583Z ##[error]Script failed with exit code: 1
56 2021-12-03T22:34:36.9676631Z [command]/opt/hostedtoolcache/Python/3.6.15/x64/bin/az account clear
57 2021-12-03T22:34:37.2400734Z ##[section]Finishing: Deploy JRIS Model to ACI
58
```


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"



```
#!/usr/bin/env python3
import requests

url = "http://71b4b465-d8af-432a-a33c-d6044e154562.eastus.azurecontainer.io/score"

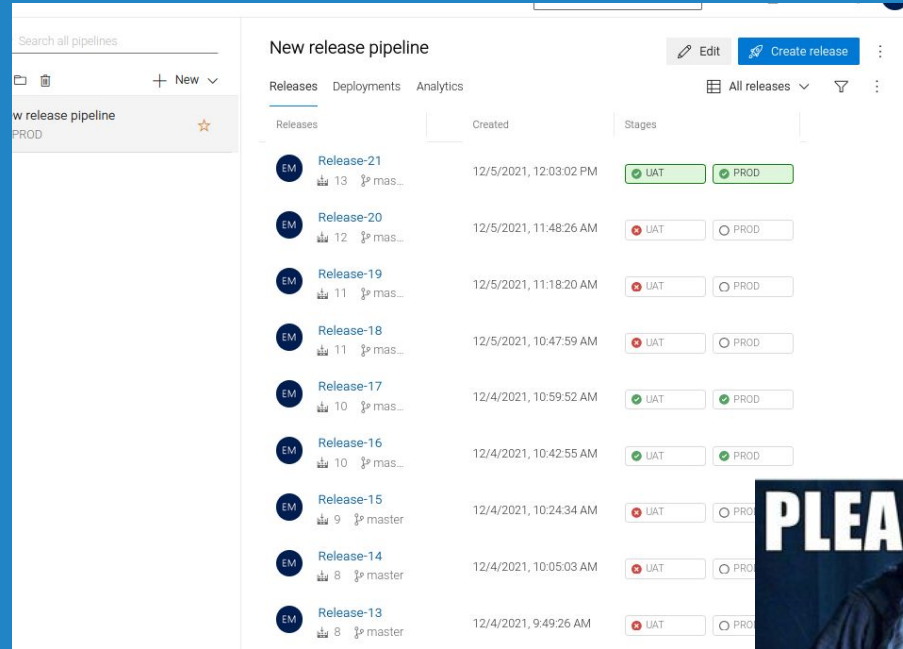
payload="{\"SepalLengthCm\": 6.6, \"SepalWidthCm\": 3, \"PetalLengthCm\": 4.4, \"PetalWidthCm\": 1.3}"
headers = {
    'Content-Type': 'application/json'
}

response = requests.request("POST", url, headers=headers, data=payload)
#print(response.content)
print(response.text)
emix@mx:~/Scaricati/MLOps-IRIS-master/deployment
$
```

```
ImportError: No module named requests
emix@mx:~/Scaricati/MLOps-IRIS-master/deployment
$ python3 test.py
{"output": {"predicted_species": "1"}}
emix@mx:~/Scaricati/MLOps-IRIS-master/deployment
$
```

Release

After some troubleshooting we have our accepted release!



The screenshot shows a 'New release pipeline' interface. On the left, there's a sidebar with 'w release pipeline' and 'PROD'. The main area has tabs for 'Releases', 'Deployments', and 'Analytics'. Below the tabs is a table of releases. The table has columns for 'Releases', 'Created', and 'Stages'. The 'Releases' column shows release names (Release-13 to Release-21) and their status (EM, 13, 12, 11, 11, 10, 10, 9, 8, 8). The 'Created' column shows timestamps. The 'Stages' column shows 'LIAT' and 'PROD' stages with status indicators (green checkmark for success, red dot for failure).

Releases	Created	Stages
Release-21 EM 13 13 mas...	12/5/2021, 12:03:02 PM	LIAT (success) PROD (success)
Release-20 EM 12 12 mas...	12/5/2021, 11:48:26 AM	LIAT (failure) PROD (failure)
Release-19 EM 11 11 mas...	12/5/2021, 11:18:20 AM	LIAT (failure) PROD (failure)
Release-18 EM 11 11 mas...	12/5/2021, 10:47:59 AM	LIAT (failure) PROD (failure)
Release-17 EM 10 10 mas...	12/4/2021, 10:59:52 AM	LIAT (success) PROD (success)
Release-16 EM 10 10 mas...	12/4/2021, 10:42:55 AM	LIAT (success) PROD (success)
Release-15 EM 9 9 master	12/4/2021, 10:24:34 AM	LIAT (failure) PROD (failure)
Release-14 EM 8 8 master	12/4/2021, 10:05:03 AM	LIAT (failure) PROD (failure)
Release-13 EM 8 8 master	12/4/2021, 9:49:26 AM	LIAT (failure) PROD (failure)



Users and management has Approved

- Multiplayer platform
- Immediate Assessment

The screenshot displays the Azure DevOps interface for a release pipeline. The top section shows a summary of the pipeline, indicating it was manually triggered by Emanuele Marnati on 12/4/2021, 10:59 AM. The pipeline consists of two stages: UAT (User Acceptance Testing) and PROD (Production). Both stages are marked as 'Succeeded'. The UAT stage shows a 100% completion rate. The PROD stage shows a '1 warning' on 12/4/2021, 11:09 AM.

The bottom section provides a detailed view of the 'PROD' stage. It shows the 'Pre-deployment conditions' as 'Succeeded'. The 'Approvals' tab is active, displaying a list of approvals. One approval is shown, dated 12/5/2021, 12:11 PM, by Emanuele Marnati. The approval is marked as 'Approved on 12/5/2021, 12:11 PM'.

A blue arrow points from the 'PROD' stage in the top summary to the detailed view of the 'PROD' stage in the bottom section.

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=1BSwYIJUxK0&list=PLZoTAE LR MXVok1pRcOCaG5xtXxgMalple>