MLOps Engineering
Machine Learning Operations V2.0.0
Sessions 10 - 11

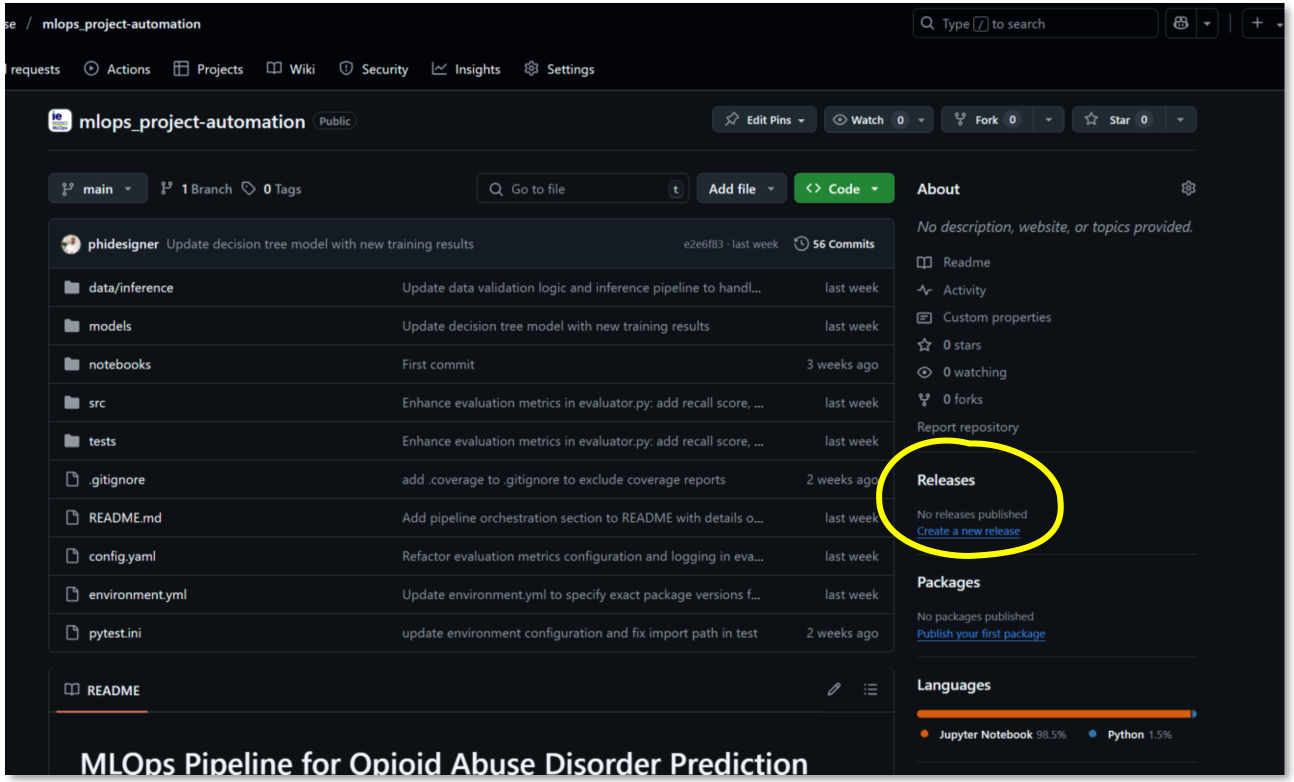
MsC in Business Analytics and Data Science

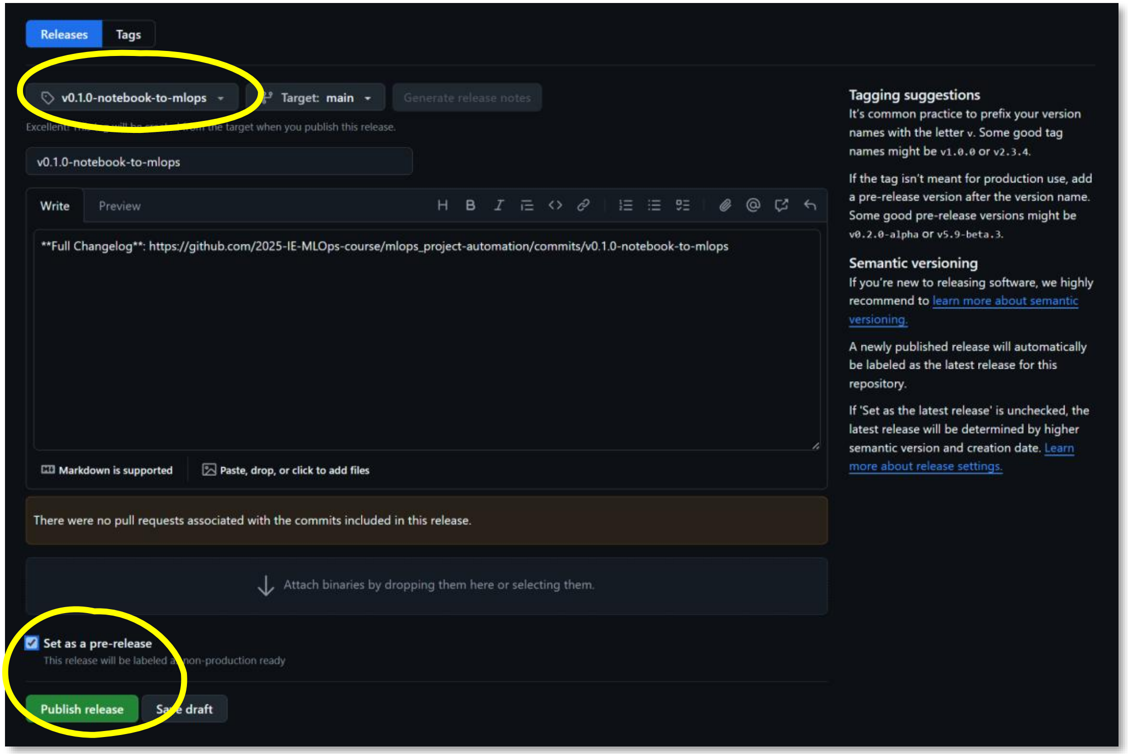
Madrid, Jun 2025

Agenda

- Final group assignment (Q&A)
- Freezing and tagging
- Hydra (F&F) demo
- DVC for data lineage
- Mlflow + WandB

① Freezing your release as a baseline





Updated README.md with Changelog

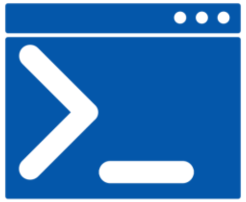
```
# MLOps Project Automation

[... existing content ...]

## Changelog

### v0.1-baseline
- Functional pipeline before integrating MLflow Projects, Hydra, and W&B
- For reference or rollback, check [this release/tag](https://github.com/2025-IE-MLOps-course/mlops_project-automation/releases/tag/v0.1-baseline)
```

2

Use Hydra to simplify, scale, and standardize parameter management across all pipeline steps

Even simple projects need many parameters for pipeline and model management

Hardcoding parameters limits code flexibility, reuse, and scalability



Managing many parameters via CLI becomes messy and error-prone quickly

- Hydra is a Python tool for managing configurations
- Uses YAML files to control pipeline parameters easily
- Solves path, reproducibility, and parameter issues in modular pipelines
- Ensures correct config is used across local and pipeline runs

Hydra keeps projects tidy, lets you change settings instantly, and safely runs many experiments.

Feature	Manual YAML + Argparse	Hydra
Config organization	One big file, hard to manage as projects grow	Split configs, easy to organize
Changing settings	Must edit file or code for each change	Change any setting from command line
Multiple experiments	Manual loops, lots of copy-paste	Run many experiments in one command
Avoiding mix-ups	Risk overwriting old results	Each run gets its own safe folder
Error prevention	Easy to miss typos or wrong types	Catches errors and typos for you
Project growth	Becomes hard to maintain over time	Stays tidy and manageable as you scale
Team collaboration	Difficult to share and reuse configs	Easy to share and reuse pieces

Canonical directory (illustrative)

```
mlops_example/
│
├── config.yaml
├── main.py
├── MLproject
├── conda.yml
├── data/
│   └── train.csv
└── src/
    └── train/
        ├── run.py
        └── MLproject
```

Use MLflow and Hydra to flexibly run, test, and scale modular pipelines with robust, dynamic configuration control

```
# Run the whole pipeline from the repo root

> mlflow run .

•   Triggers orchestrated, multi-step pipeline
•   Handles Conda env setup
•   Hydra config management in production-style
    orchestration

# Run a single step as a subproject

> mlflow run src/train

•   Each step is independently runnable
•   Reproducibility and modularity
•   Each step can be unit tested and experimented
    with separately

# Run orchestrator manually

> python main.py
> python main.py main.steps=train

•   Useful for rapid debugging
•   Hydra config loading and override (CLI)

# Run pipeline manually

> python src/train/run.py
> python src/train/run.py
train.input_path="data/<other>.csv"

•   Immediate feedback for code/test changes
•   Hydra's config composition, CLI overrides,
    experiment folders
```

```
# Override config on the fly

> python main.py train.max_iter=50

•   Dynamically change model, data, hyperparams
    without editing YAML ("overrides")
```

- **MLflow run** for end-to-end, modular, production-ready orchestration
- **Direct Python/Hydra** run for development, testing, and rapid iteration
- **Hydra CLI overrides** to change any config without touching YAML i.e. experimentation and safe parameter sweeps

Tracing the Digital Footprints - Understanding Data Lineage

- **Data provenance** documents a dataset's **entire history**, including its source, transfers between systems, and all changes made over its lifecycle
- **Data origin** specifies the original source, collection method, and context of data creation, such as census records collected through a specific API in a defined year
- **Data movement** traces each transfer step, for example, from API ingestion to cloud storage, and subsequent movement to analytical platforms like HDFS
- **Data manipulation** details any changes or processing applied, from initial cleaning or transformation to later modifications, with careful documentation crucial for transparency and reproducibility

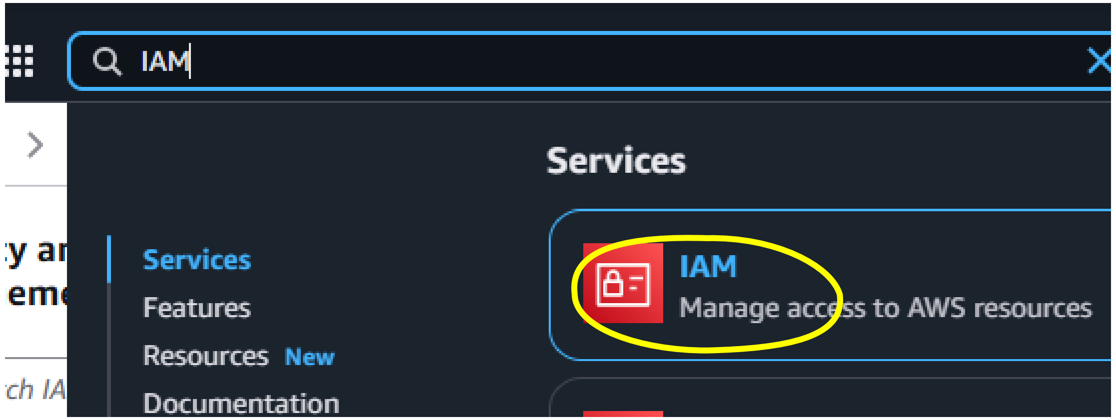
Adding Data Version Control (DVC)

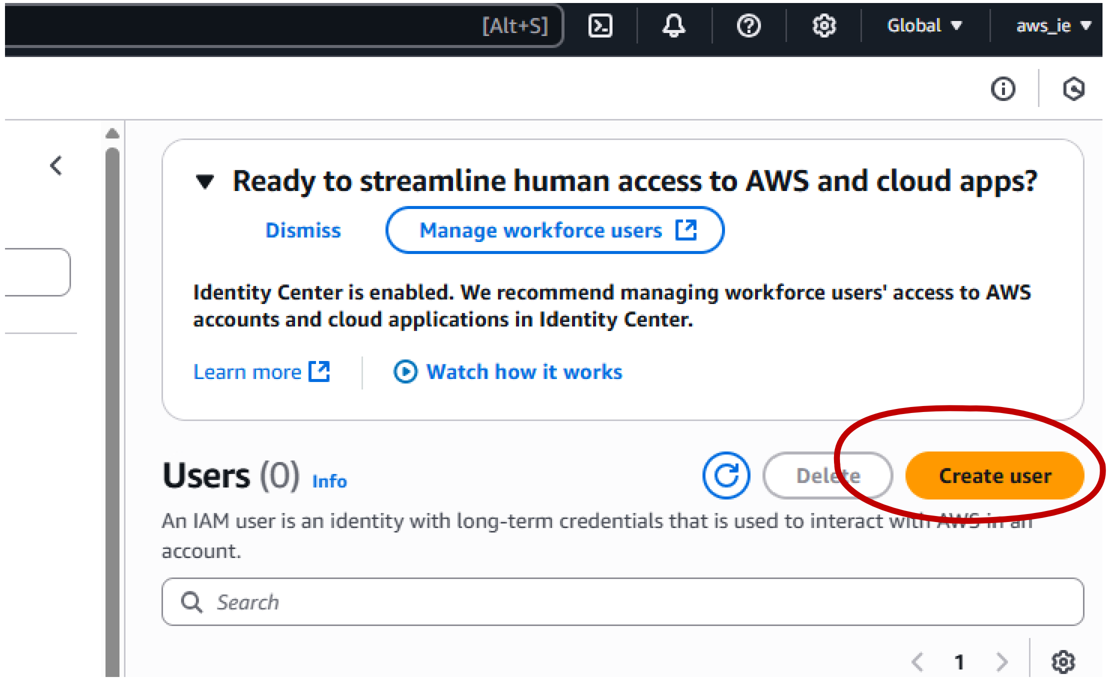

```
# Include DVC in your environment.yml
- dvc
- dvc-s3

# Set up DVC
> dvc init
> dvc add data/

# If any data is being tracked by Git
> git rm -r --cached 'data'

# Enabling auto staging
> dvc config core.autostage true

# Config remote storage
> dvc remote add -d s3remote s3://2025-mlops-bucket/data/

# Push and pull data to/ from S3
> dvc push
> dvc pull
```

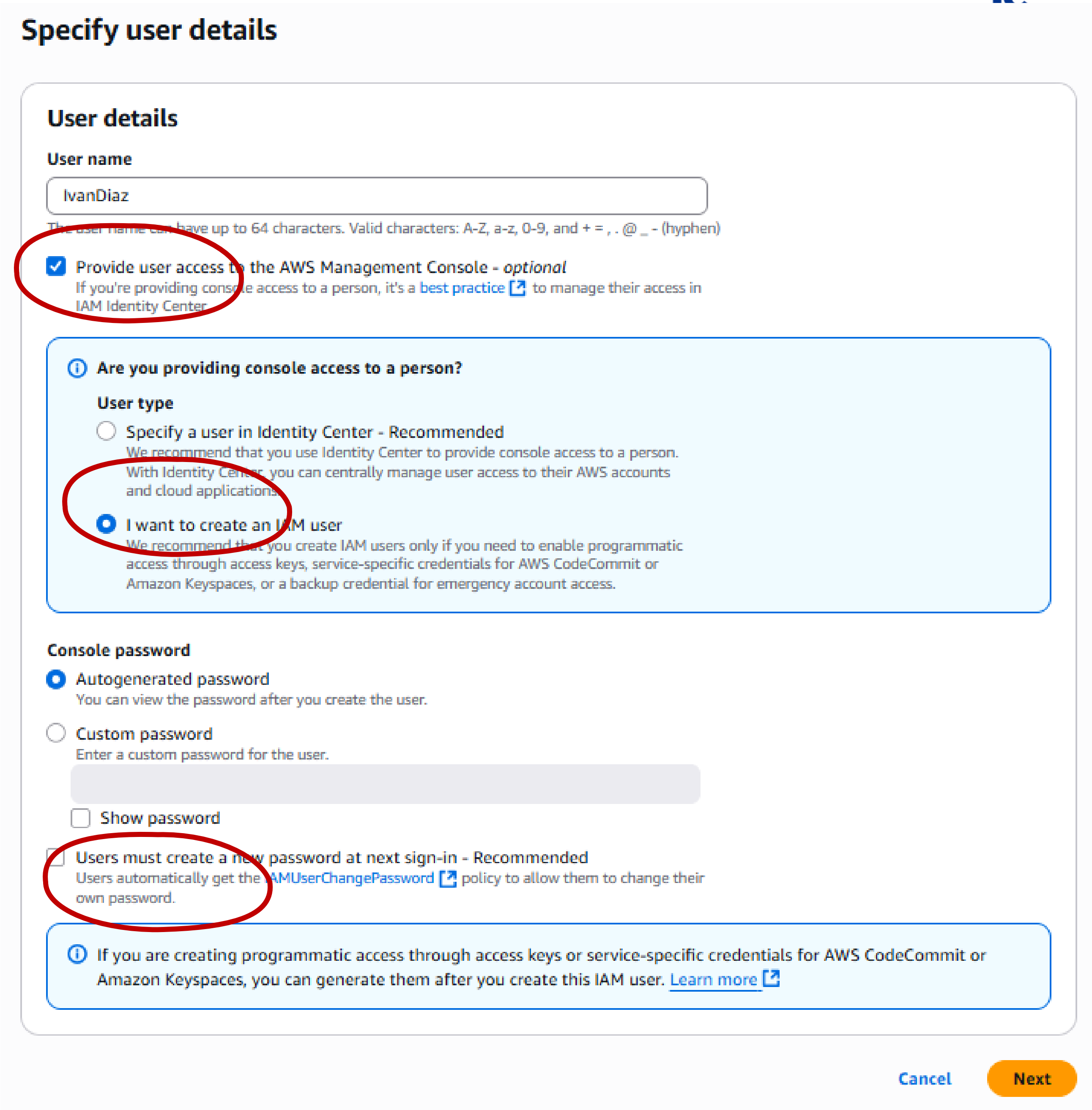
Create a free tier S3 bucket (or any other cloud storage e.g. GCS, Azure)

```
# Include aws's CLI in your environment.yml

- awscli

# Resources to be created at AWS webpage

-  Create AIM user
-  Create S3 bucket
-  Create S3 permissions (AmazonS3Fullaccess)
-  Create Access Key


# Setting up your aws CLI

> aws configure


# Provide credentials in the console

AWS Access Key ID [None]: AKIAxxxxx
AWS Secret Access Key [None]: xxxxx
Default region name [None]: eu-central-1
Default output format [None]: json


# Save in .env

AWS_ACCESS_KEY_ID=AKIAxxxx
AWS_SECRET_ACCESS_KEY=xxxx
AWS_DEFAULT_REGION=eu-central-1
```

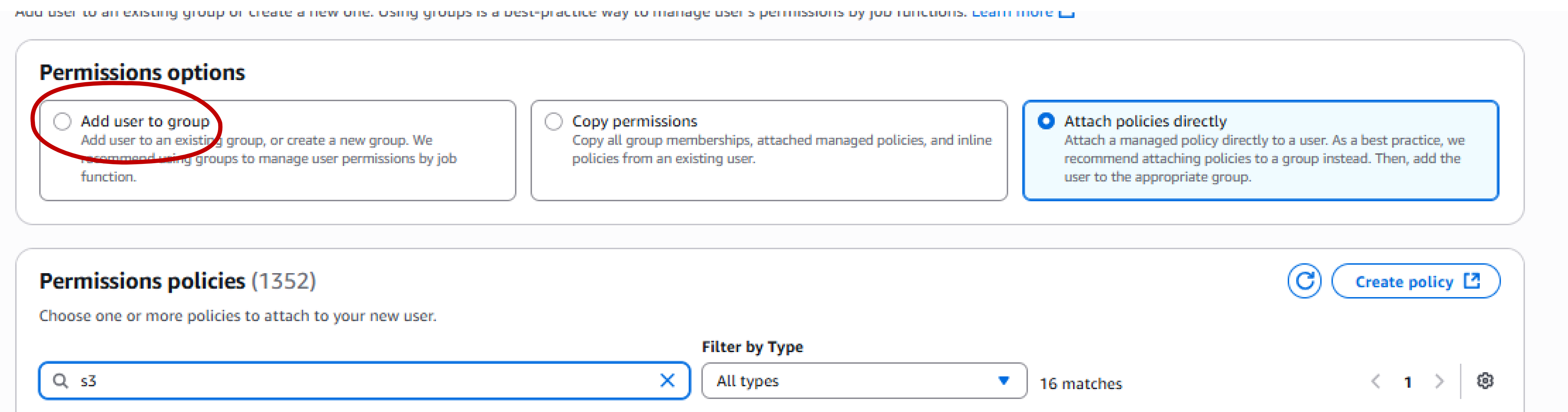
Creating a user in AWS

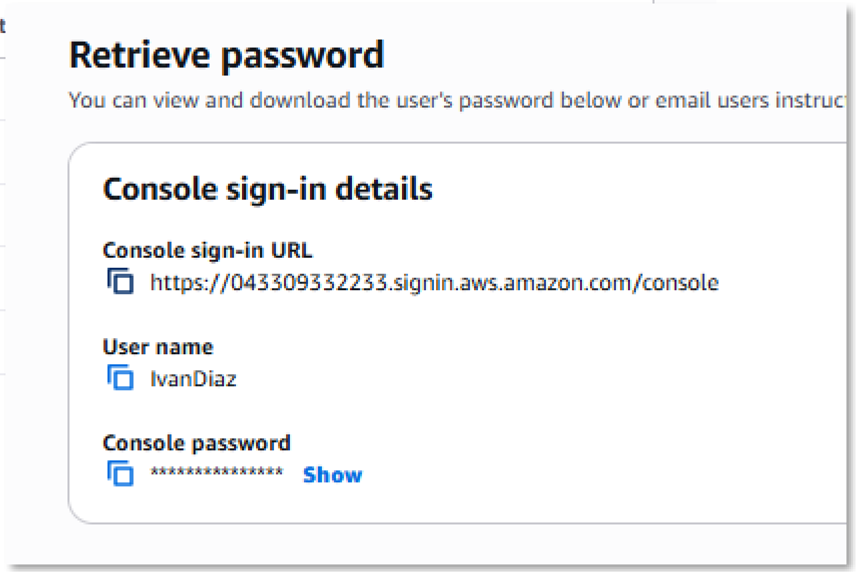




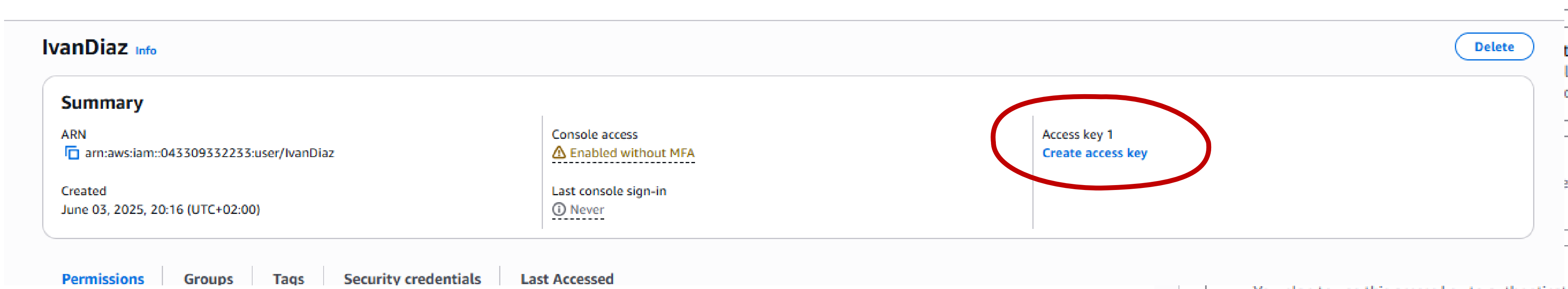


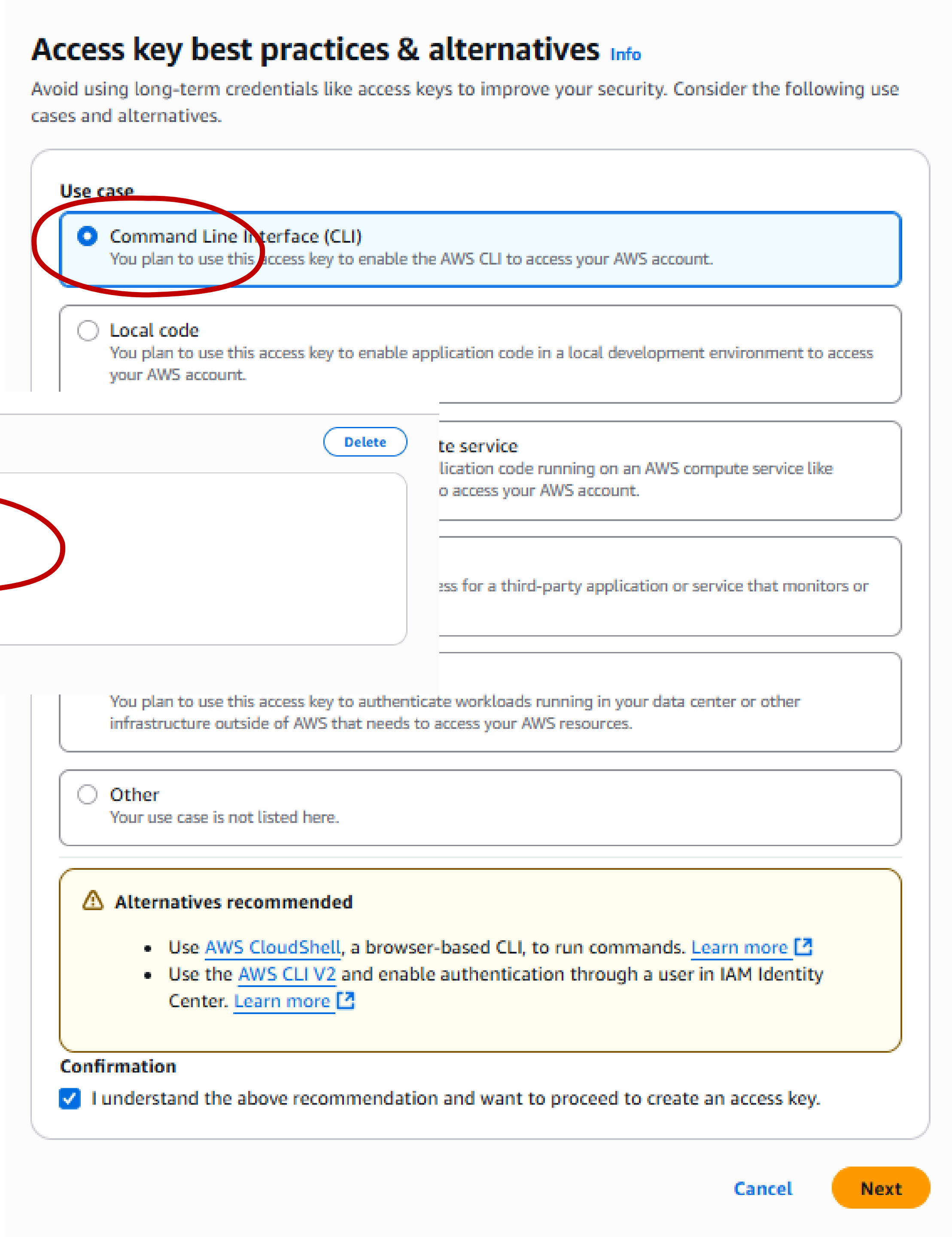
Giving permissions



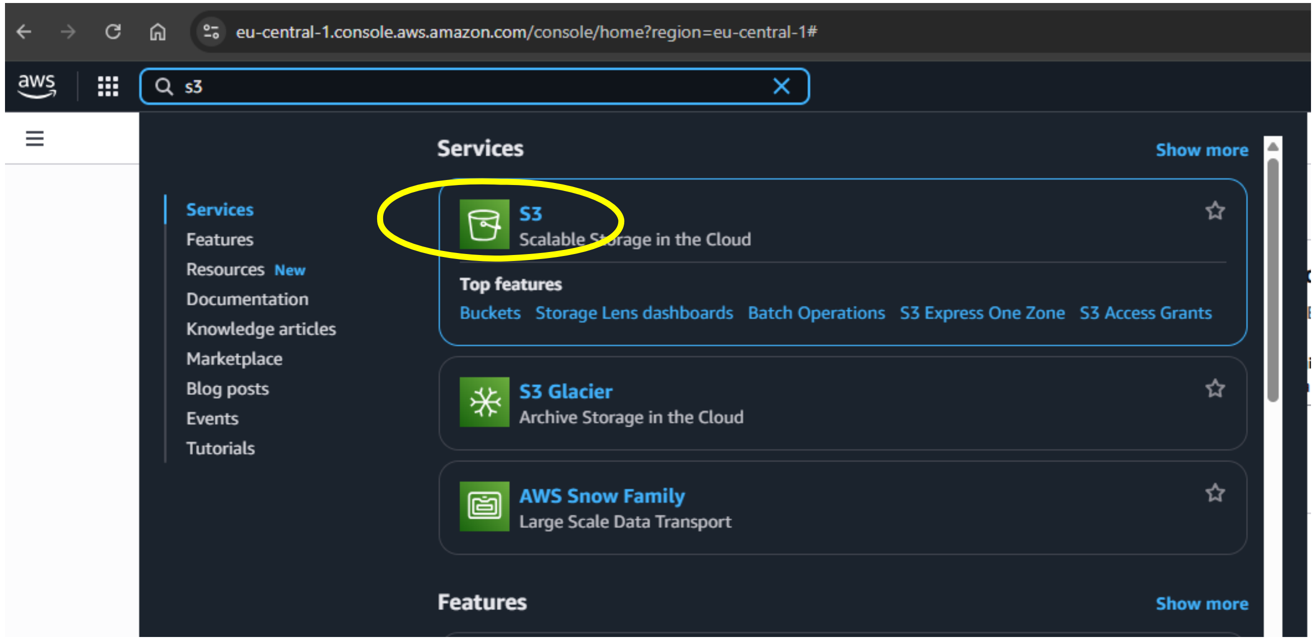


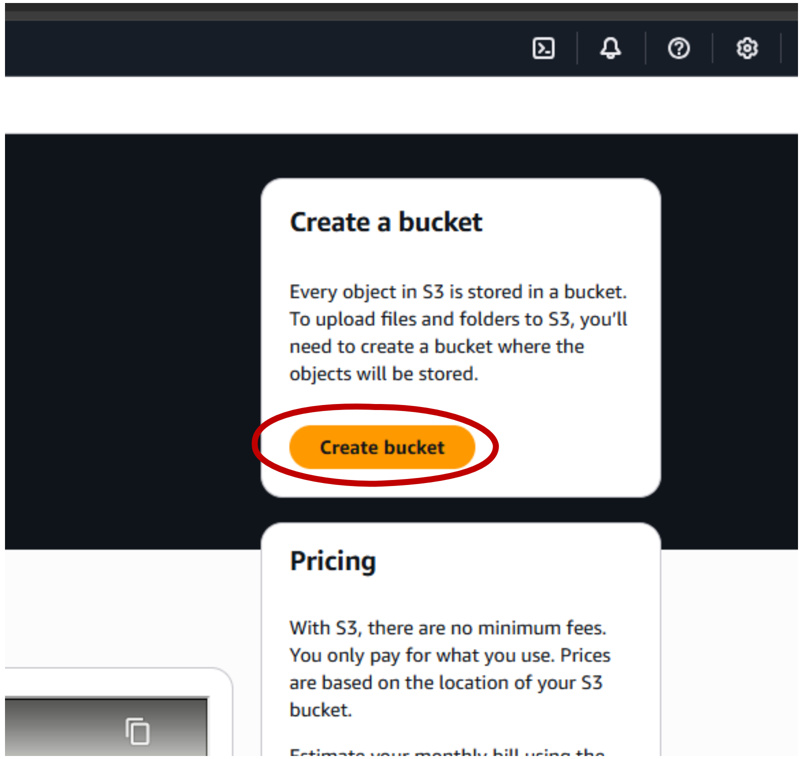
Creating a user Access Key

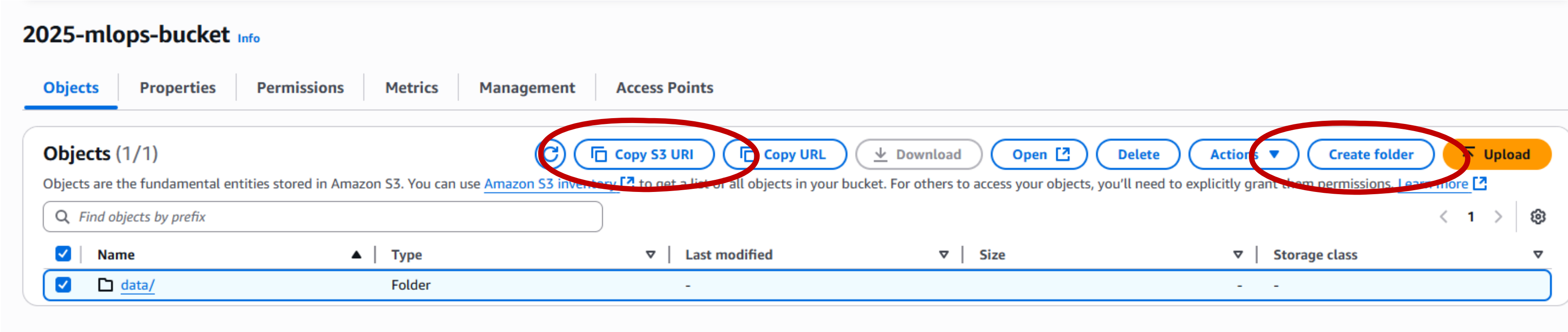




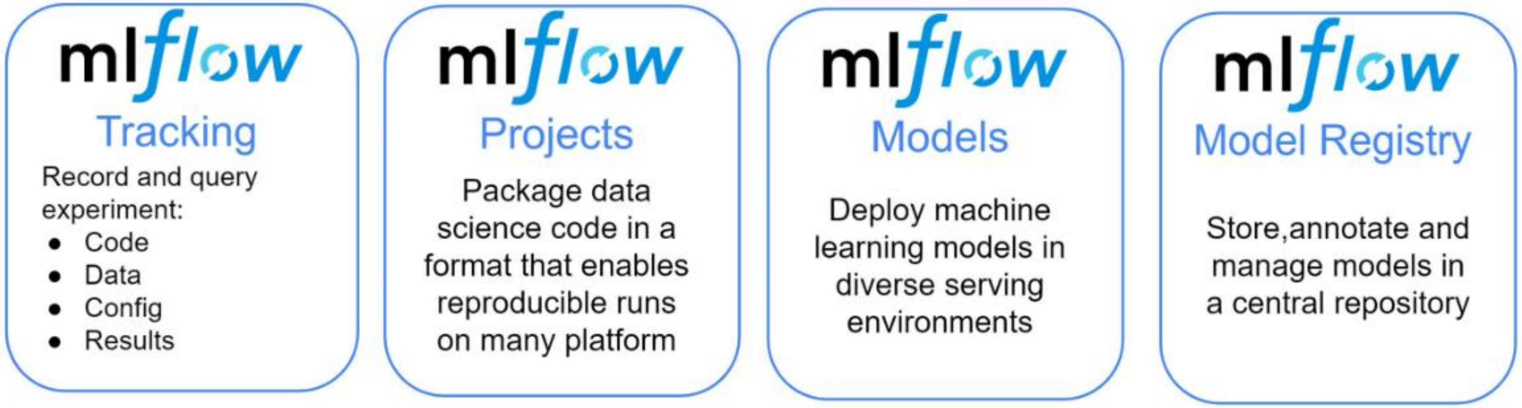
Creating an S3 bucket via AWS web interface (Free tier)







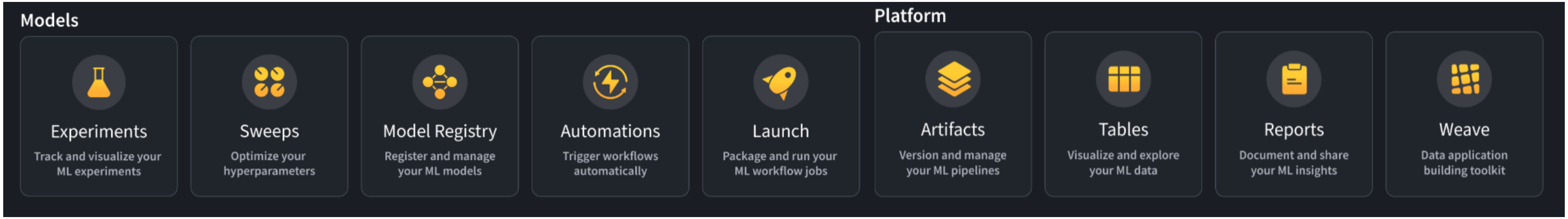
MLflow components



- **Tracking**: Allows you to track experiments to **record** and **compare** parameters and results (runs)
- **Projects**: Allow you to **package** ML code in a **reusable**, reproducible form to share with other data scientists or transfer to production
- **Models**: Allow you to manage and **deploy** models from a variety of ML libraries to a variety of model serving and inference platforms as an endpoint (Azure, AWS, Spark, etc.)
- **Model Registry**: Allows you to **centralize** a model store for managing models' full lifecycle stage transitions: from staging to production, with capabilities for versioning and annotating
- **Model Serving**: Allows you to **host** MLflow Models as REST endpoints
- **Experiments and Runs**: All runs belong to and experiment. For each experiment, one can analyze and compare the results of different runs

Source: mlFlow

WandB components



- **W&B Models** is a set of lightweight, interoperable tools for ML practitioners training and fine-tuning models
  - **Experiments**: Machine learning experiment tracking
  - **Sweeps**: Hyperparameter tuning and model optimization
  - **Model Registry**: quickly track experiments
  - **Automations**: Trigger workflow steps e.g. automated model testing and deployment
  - **Launch**: Scale and automate workloads
- **W&B Platform** is a core set of building blocks for tracking and visualizing data and models, and communicating results
  - **Artifacts:** Version assets and track lineage
  - **Tables:** Visualize and query tabular data
  - **Reports:** Document and collaborate on your discoveries
  - **Weave**: visual development environment designed for building AI-powered software

Source:Weight & Biases

Running your Mlflow pipeline

```
# Running your step-wise MLflow
> mlflow run . -P steps=“data_load”

# Running the entire pipeline
> mlflow run .

# Running for quick local debugging
> python main.py
> python main.py main.steps=“data_load”

# Running data_load in isolation
# from (src/data_load/)
> mlflow run .
> python run.py
```

An MLflow Project is a format for packaging data science code in a reusable and reproducible way, based primarily on conventions. In addition, the Projects component includes an API and command-line tools for running projects, making it possible to chain together projects into workflows

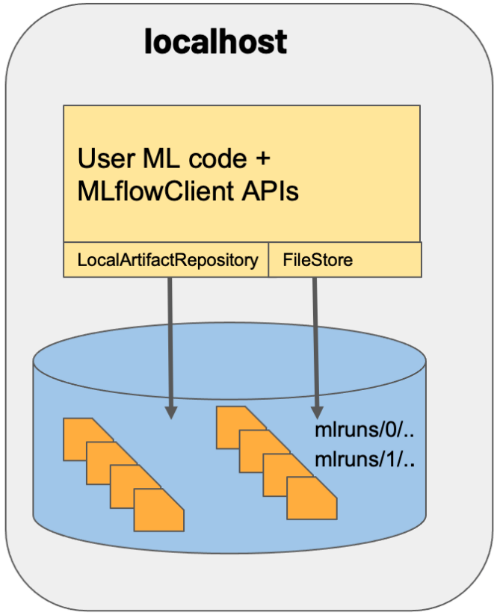
Source: mlflow documentation

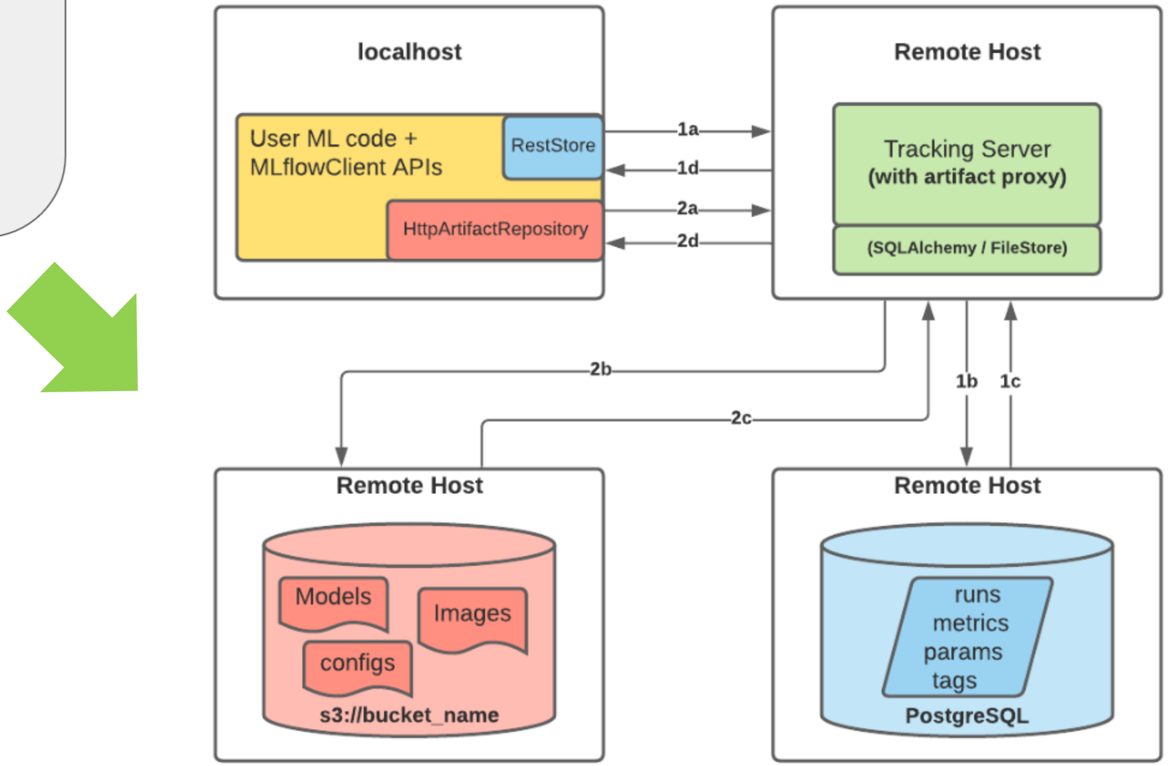
MLflow Projects

```
name: My Project
conda_env: conda.yaml
entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization} {data_file}"
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

MLflow Model Registry

The MLflow Model Registry component is a **centralized model store**, set of APIs, and UI, to **collaboratively manage the full lifecycle of an MLflow Model**. It provides model lineage, model versioning, stage transitions, and annotations





Source: mlflow documentation