# Automate Model Lifecycle

## Lecture plan

1️⃣Reminders
2️⃣Objective
3️⃣Experiment tracking with MLflow
4️⃣Automating the Model Lifecycle with Prefect

# 1️⃣Reminders

🧰 **Google Cloud Platform**

- **Console** vs **CLI** vs **code**
- **Authentication**: one method for each interface

🧰 **Cloud Storage**

- **Immutable** data
- Files, images, sound, video

🧰 **Big Query**

- **Relational** data
- Columnar storage & partitions

🧰 **Compute Engine**

- **Virtual machine**
- Setup operating system + code environment

🧰 **direnv**

- Generate **environment variables**
- Declared in `.env`

```
BUCKET_NAME=le-wagon-data
```

- Used in **CLI** or `Makefile`

```
gsutil ls gs://$BUCKET_NAME
```

- Used in **code**

```
bucket_name = os.environ["BUCKET_NAME"]
```

# 2️⃣ Objective

**What is our progress?**

**Task:** Create the **WagonCab** app 🚘 and connect it to a model that is **always up-to-date** 🌱 with the latest fare prices

✅ Train the model from data in a **data warehouse** 🔍

✅ Save the trained model in a **storage solution in the cloud** 📦

✅ Run the model training on a **virtual machine** 🤖

**What's next?**

❌ Issues:

- The model needs **constant care** to adapt to the market
- We cannot continue to **manually train** each model

🎯 Create a robust model **lifecycle** ♻️:

- Ensure the **reproducibility** of the training in the future
- **Track** the performance of the model over time
- **Serve** multiple *versions* of the model
- **Automate** the model lifecycle

**Robust Lifecycle**

❓ How to **track** our model experiments over time 📝

1. Store the experiment parameters (data, code, environment)
2. Store the model performance

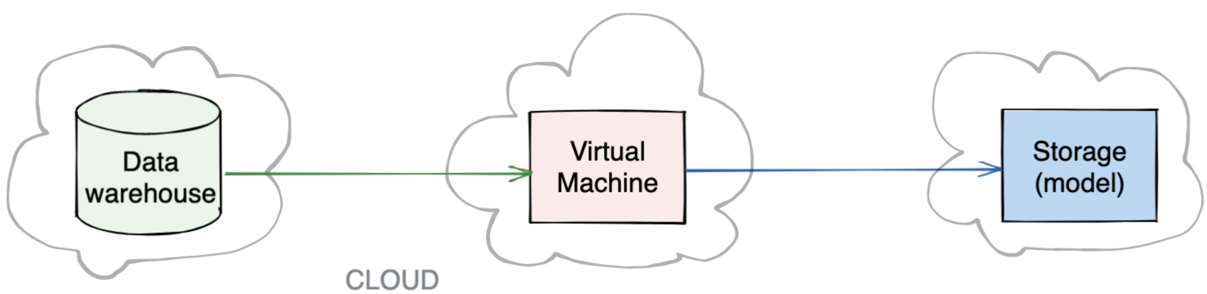
3. Store the trained model

❓ How to **automate** the model lifecycle ♻️

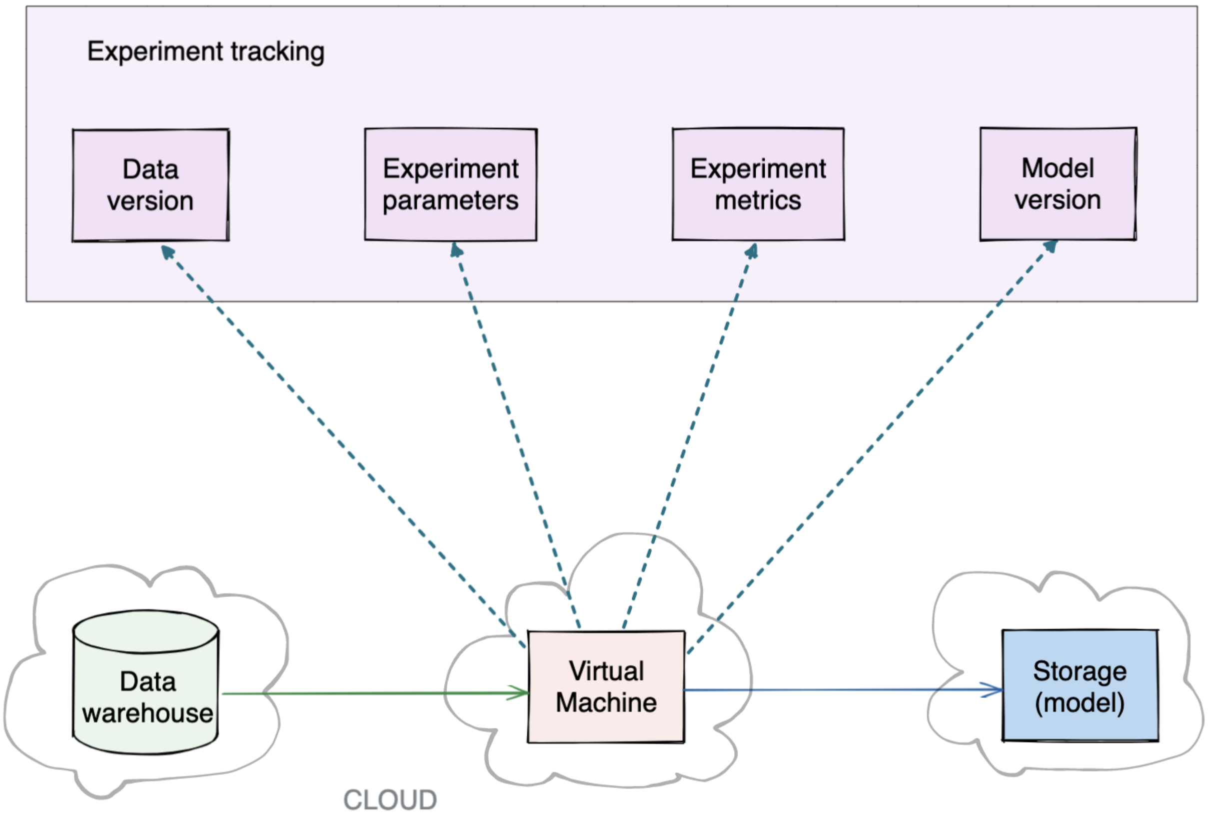
1. Formalize the model workflow
2. Have it run periodically

# 3️⃣ Experiment tracking with MLflow

`Cloud training`



## Experiment tracking




**Tracking Requirements**

⚙️ Experiment **params** & **metrics**:

- Code version (git commit id/hash/sha)
- Code parameters
- Training environment (Python + package versions)
- Preprocessing type
- Model hyperparameters
- Training metrics

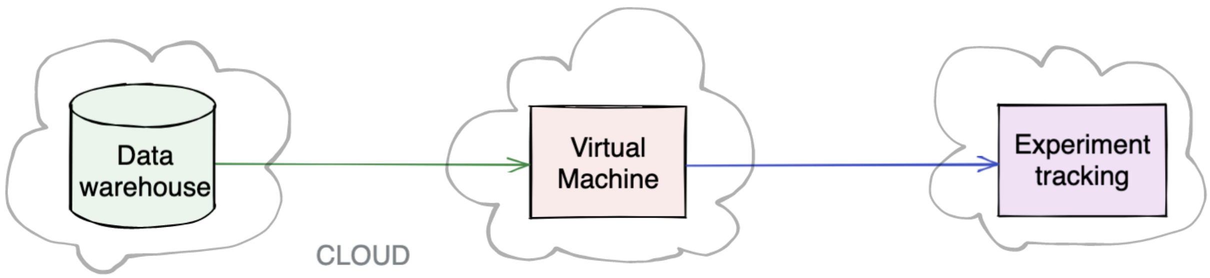
🧬 **Model** version:

- Persisted trained model
- Version number

📦 **Data** version:

- Good practice to also keep track the data used for the training
- We used `start_date`, `end_date` and a size `1k` vs `200k` vs `all`
- more professional approach: [DVC](DVC) (Data Versioning Control - git diffs for datasets)

Experiment tracking




❓ **How to track our experiments?**

👉 *MLflow* **server**

- Stores experiment **tracking data** in a *database*
- Stores experiment **trained models** in a *file storage system*
- Can be hosted on a **local** machine or in the **cloud**

👉 *MLflow* **UI**

- Web interface to visualize **tracking data** and annotate the **trained models**

👉 *MLflow* **CLI**

- We will not focus on it, just be aware that it exists

👉 *MLflow* **code**

- `mlflow` *Python package*
- Pushes tracking data and trained models to the *MLflow* server through an API

**Track Experiment + Save Model**

**import mlflow**

```python
mlflow.set_tracking_uri("https://mlflow.lewagon.ai")
mlflow.set_experiment(experiment_name="wagoncab taxifare")
```

```python
with mlflow.start_run():

    params = dict(batch_size=256, row_count=100_000)
    metrics = dict(rmse=0.456)

    mlflow.log_params(params)
    mlflow.log_metrics(metrics)

    mlflow.tensorflow.log_model(model=model,
                            artifact_path="model",
                            registered_model_name="taxifare_model"
    )
```

**Load model**

```python
import mlflow

mlflow.set_tracking_uri("https://mlflow.lewagon.ai")

model_uri = "models:/taxifare_model/Production"

model = mlflow.tensorflow.load_model(model_uri=model_uri)
```

More **info** in the MLflow tracking, registry, and model stages docs

# Livecode 🚧

🎯 Track experiment **parameters**, **metrics** and store the **trained model**

- Let's use the MLflow server provided by Le Wagon

👉 You will learn to deploy your own MLflow server later on

**Track Experiment**

🕵️ Explore the `mlflow_run` decorator in `registry.py`

💻 Add model tracking

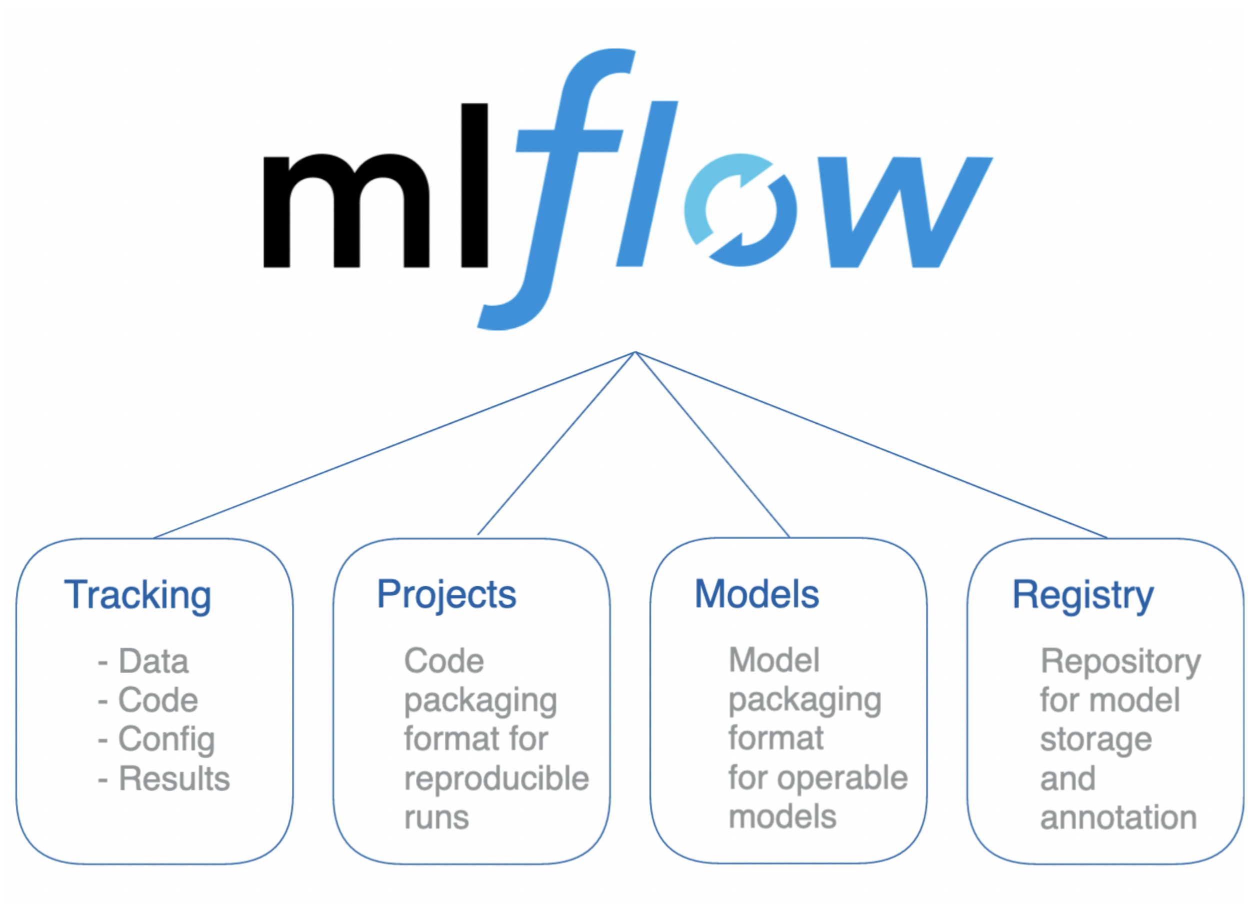
- Add parameters
- Add metrics
- Store model
- Make a prediction with the stored model (with a model version)

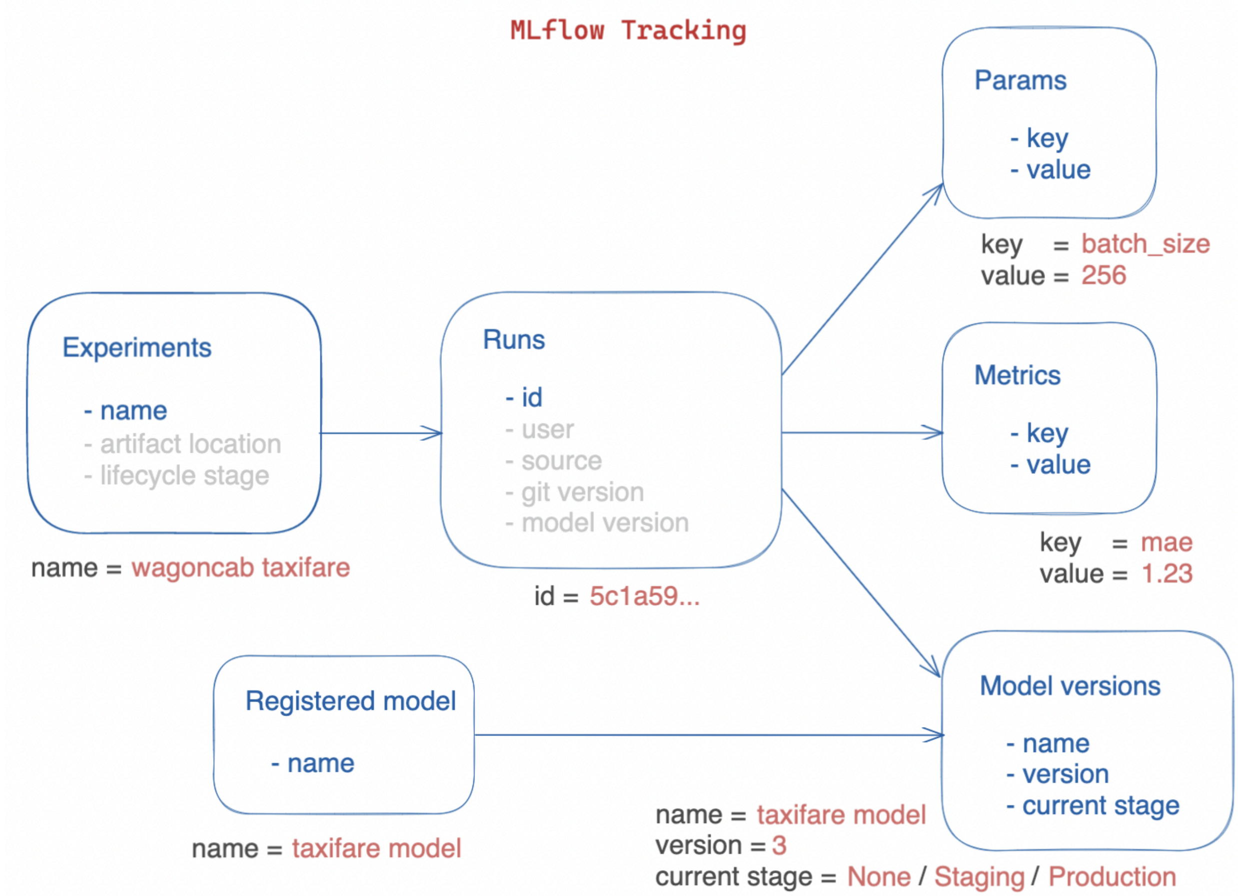
## Theory 🔭

📖 What we saw:

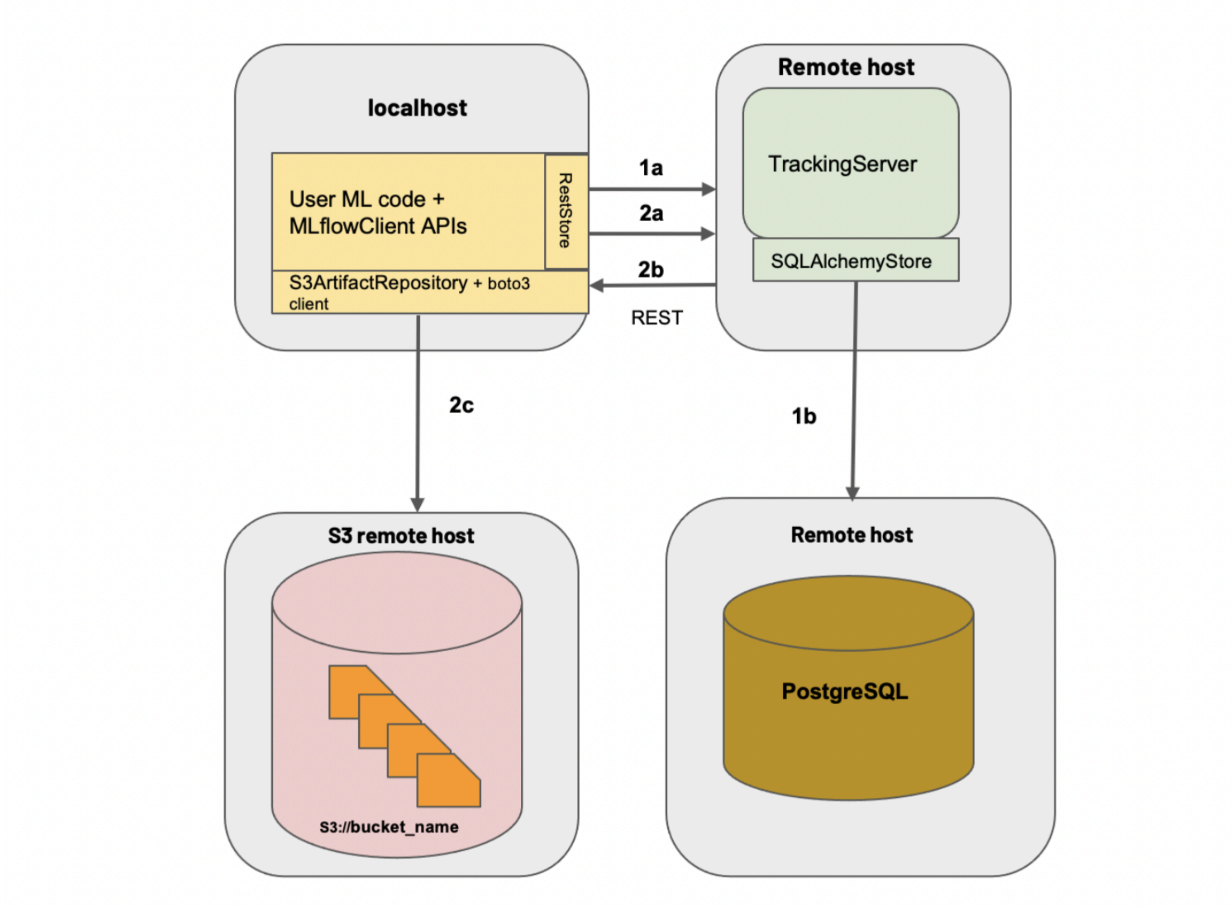
- `parameters` and `metrics` tracking
- `model` storage

## MLflow



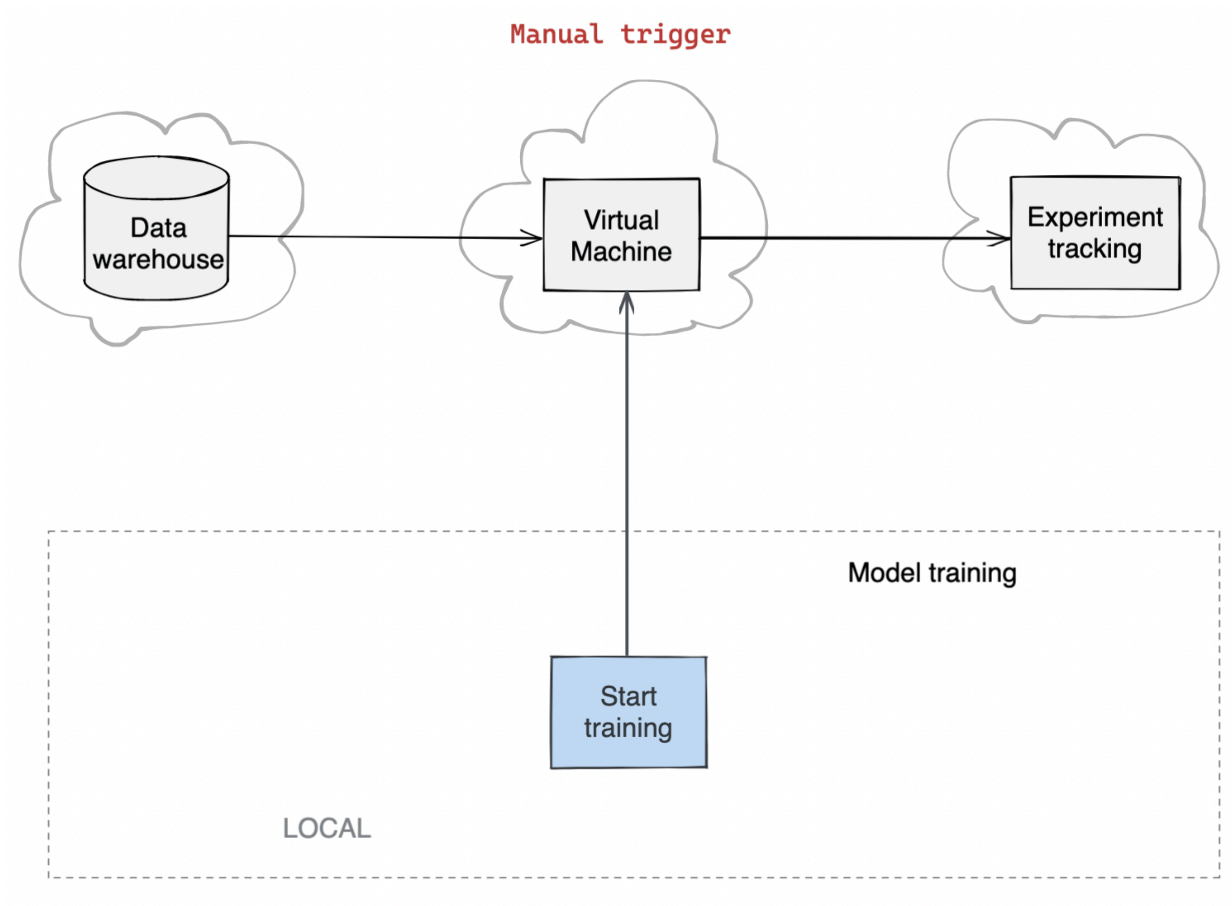


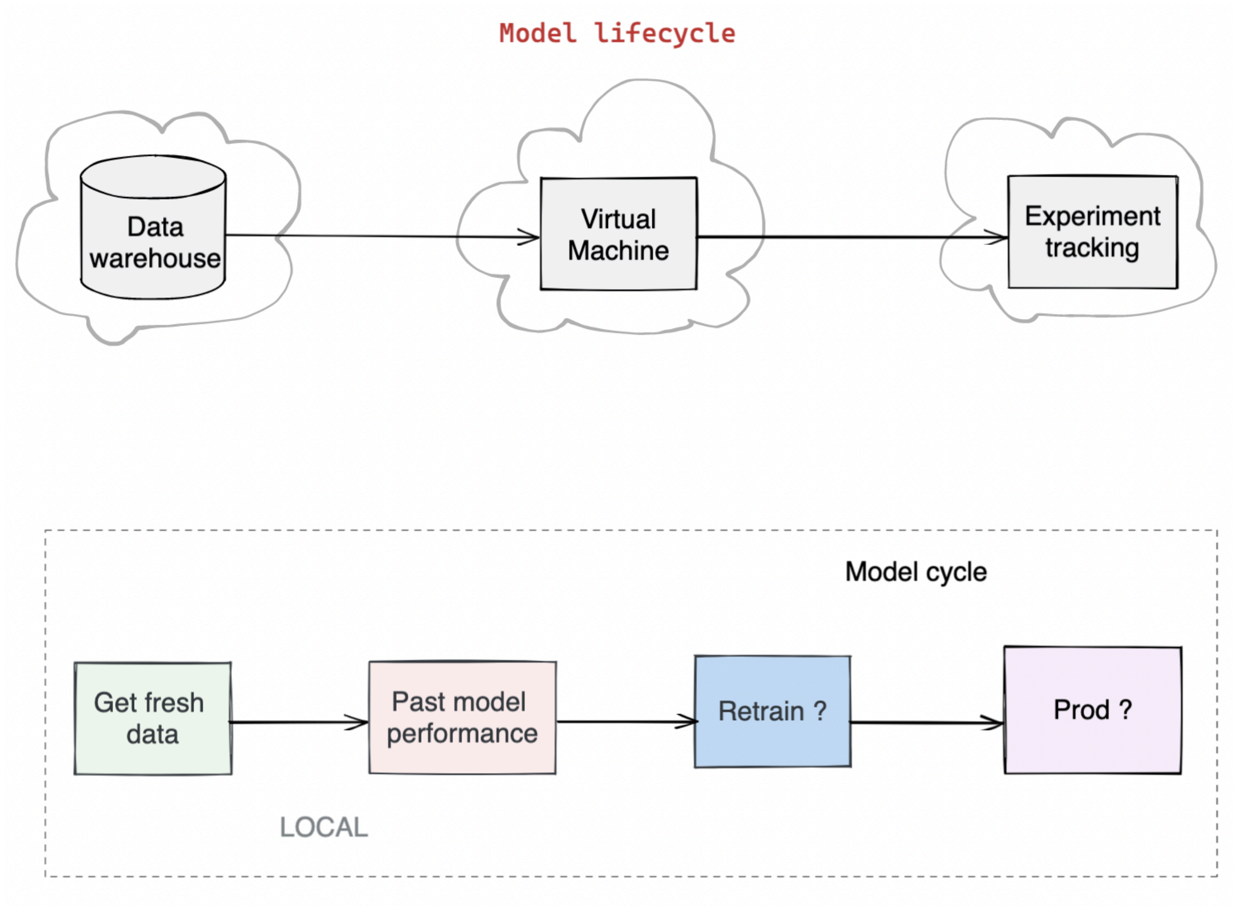

**MLflow Server Architecture**



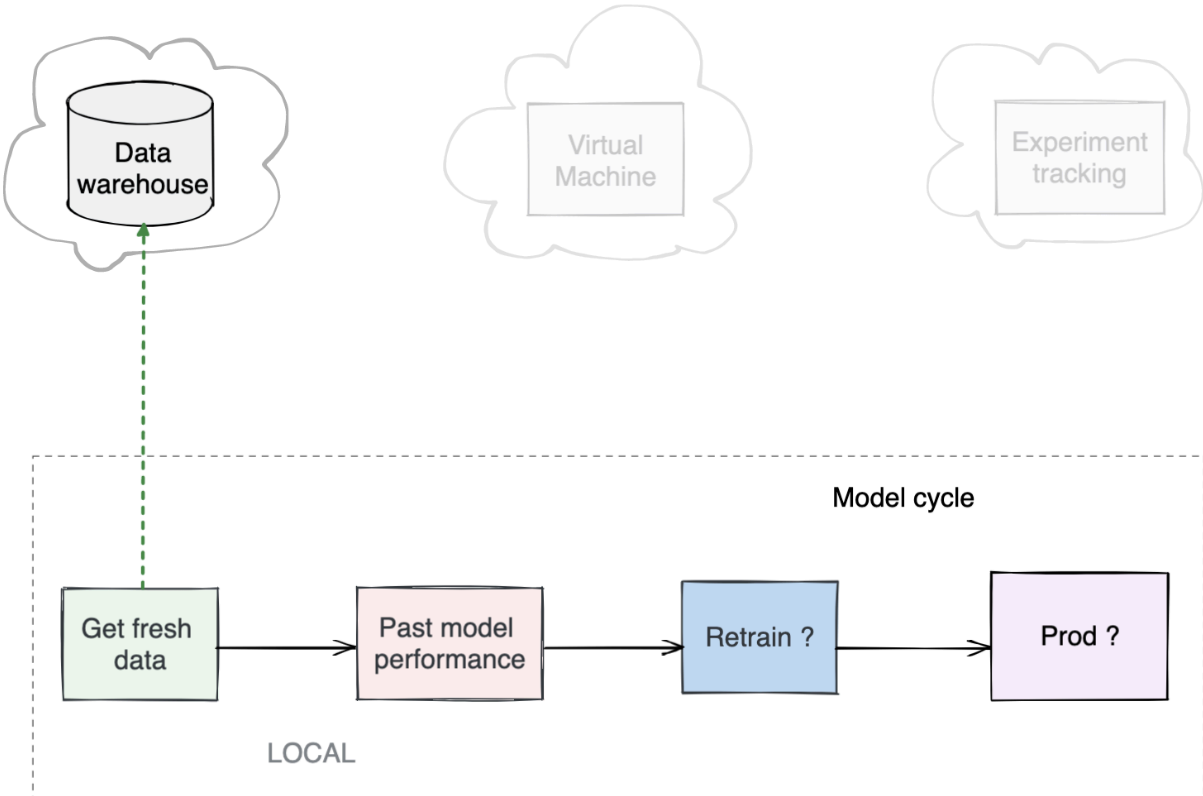

[MLflow tracking server + artifact store]

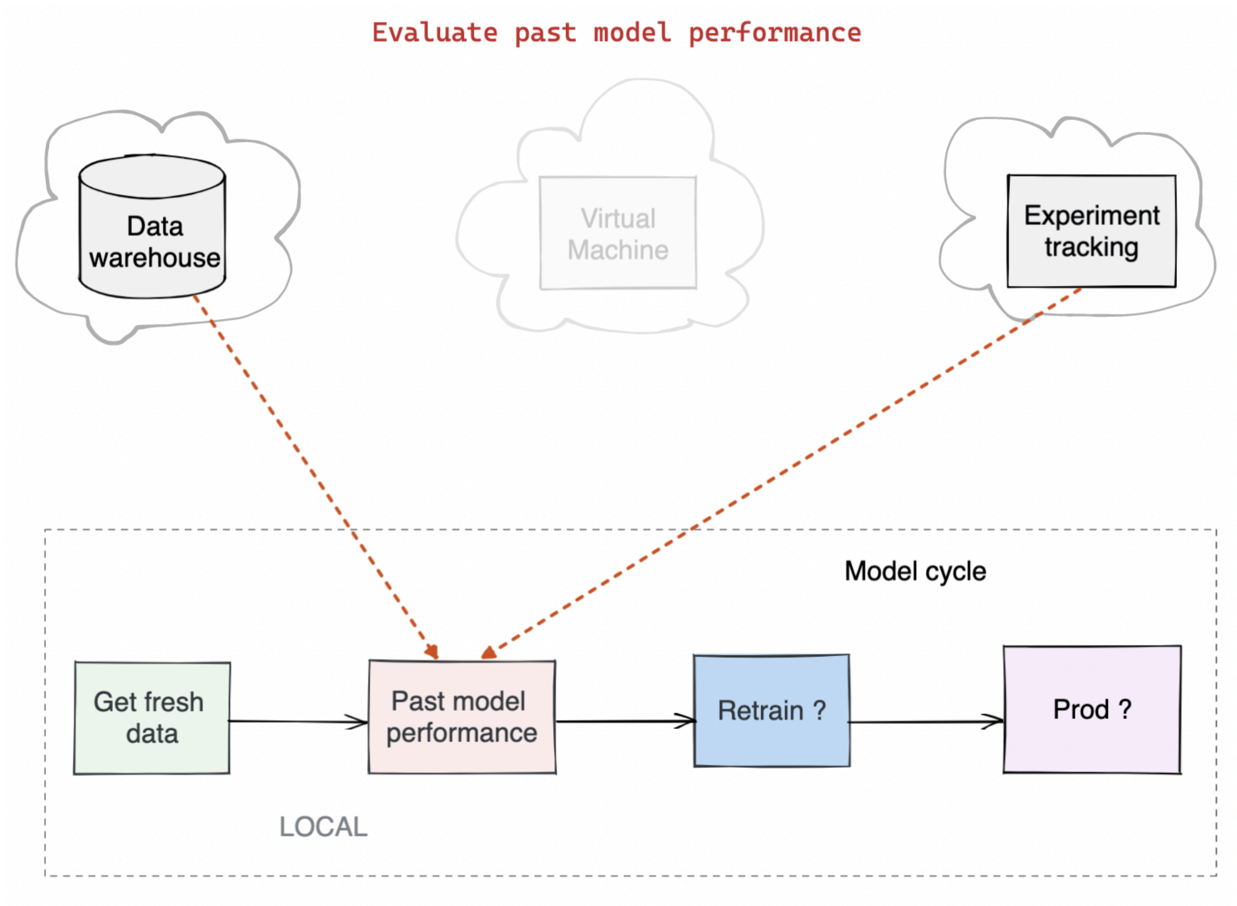
# 4️⃣ Automating the Model Lifecycle with Prefect


Manual trigger
Data warehouse
Virtual Machine
Experiment tracking
Model training
Start training
LOCAL

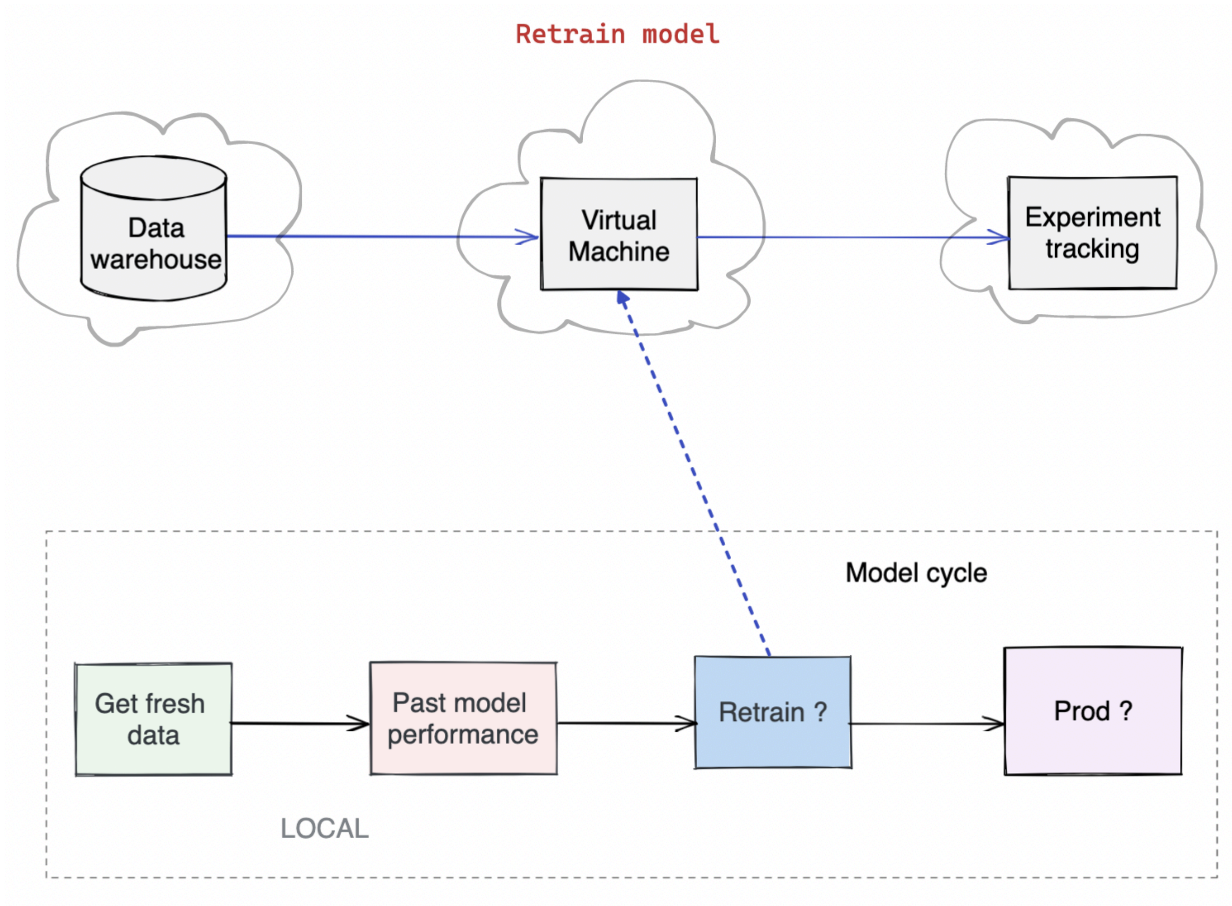

Model lifecycle
Data warehouse
Virtual Machine
Experiment tracking
Model cycle
Get fresh data
Past model performance
Retrain ?
Prod ?
LOCAL

Get fresh data

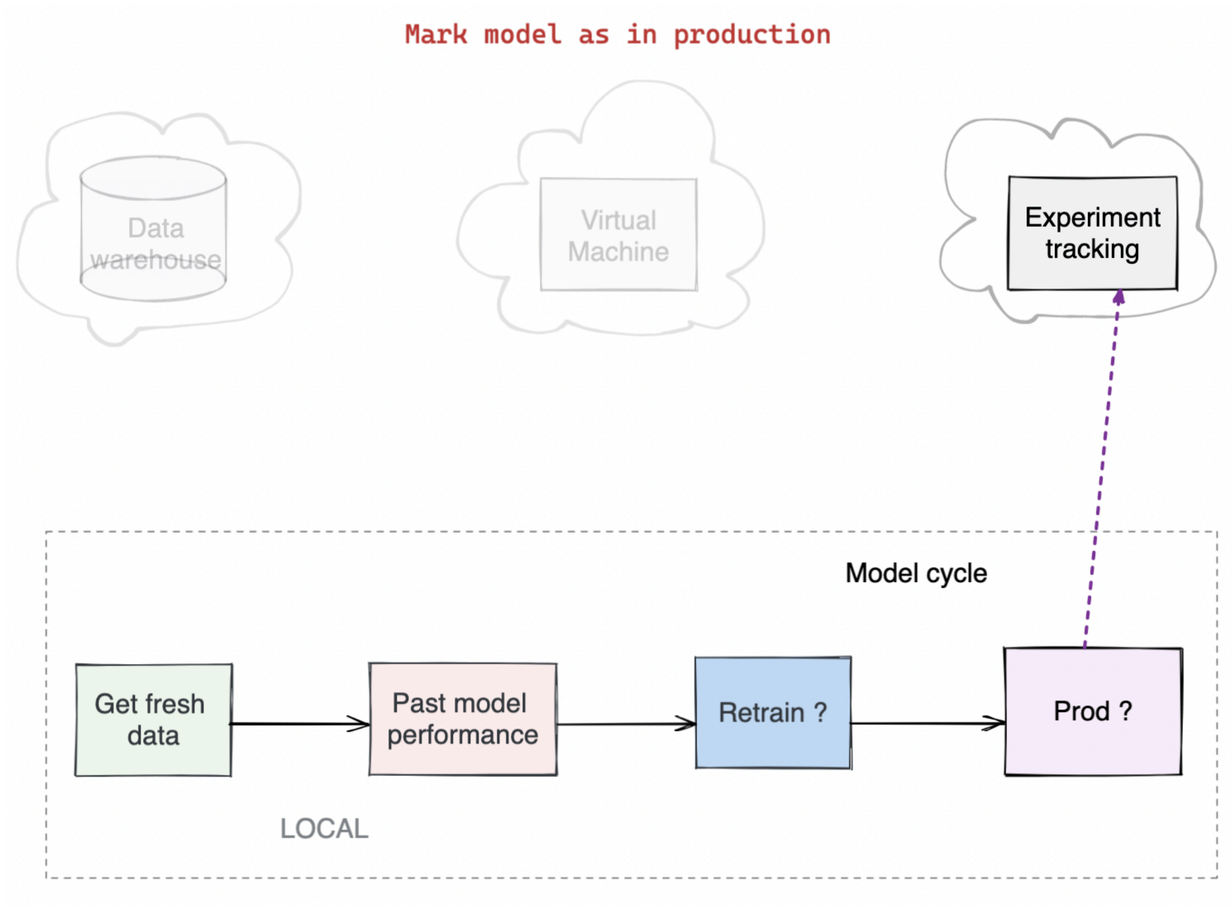




Evaluate past model performance
Data warehouse
Virtual Machine
Experiment tracking
Model cycle
Get fresh data
Past model performance
Retrain ?
Prod ?
LOCAL

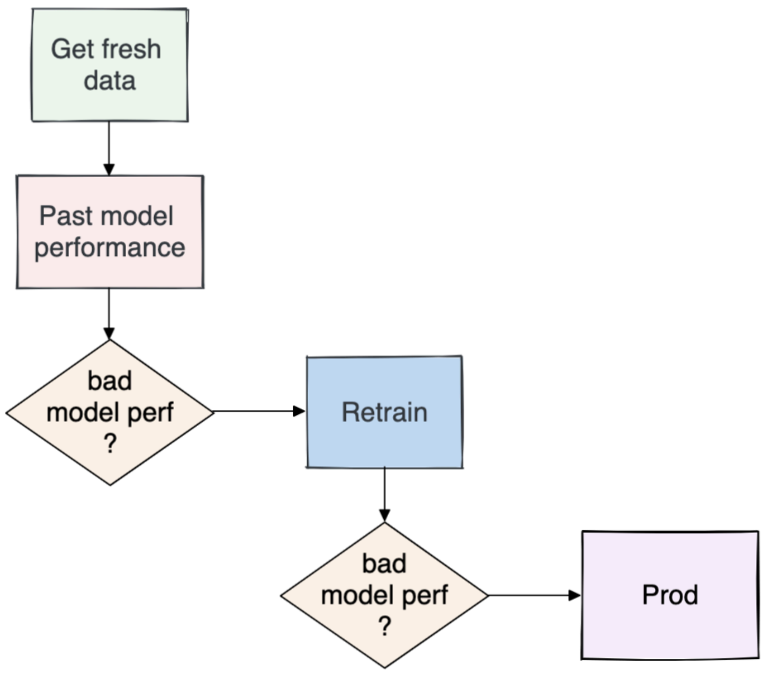


Retrain model
Data warehouse
Virtual Machine
Experiment tracking
Model cycle
Get fresh data
Past model performance
Retrain ?
Prod ?
LOCAL

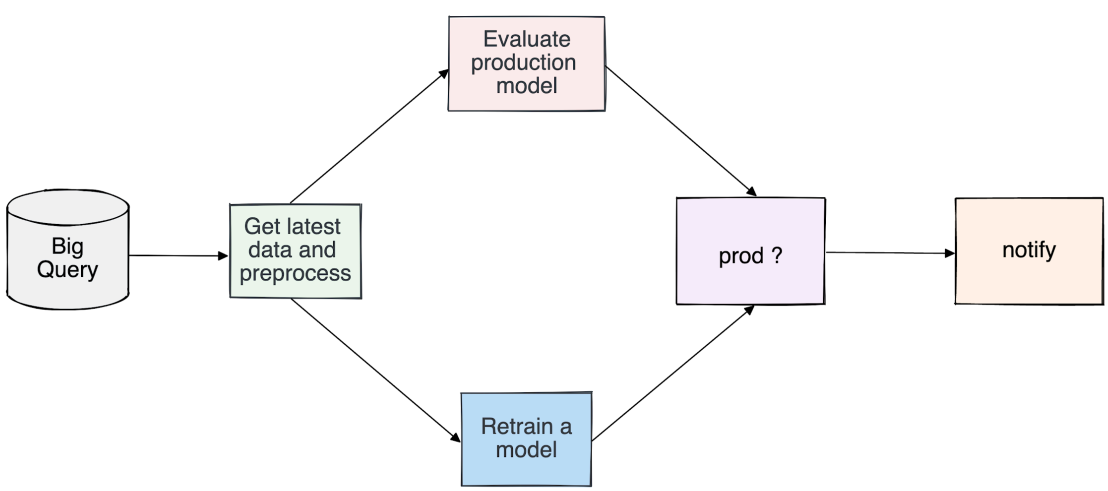


Mark model as in production
Data warehouse
Virtual Machine
Experiment tracking
Model cycle
Get fresh data
Past model performance
Retrain ?
Prod ?
LOCAL

## Model lifecycle

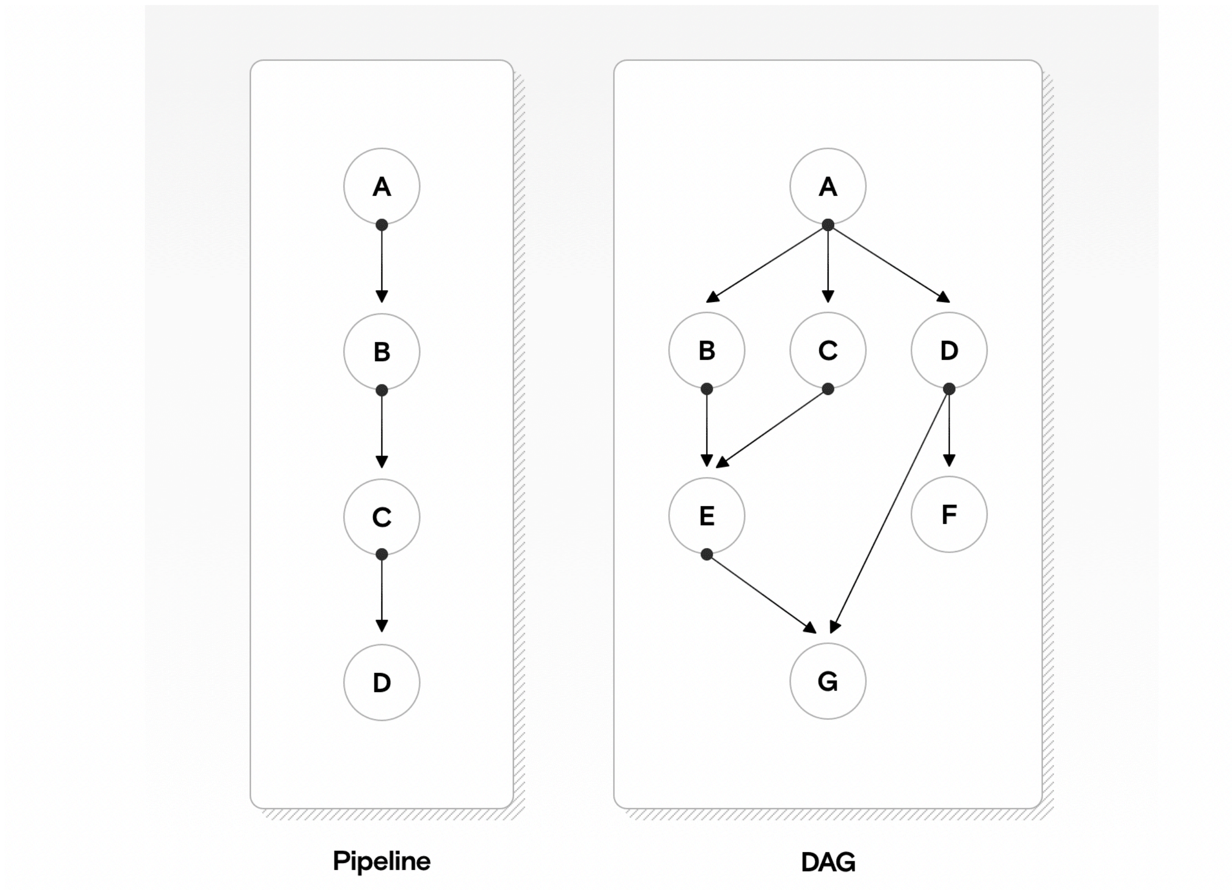



## Goal of the challenges



# Direct Acyclic Graph

👉 Workflow of tasks




# Livecode 🚧

🎯 Decompose model lifecycle
**Workflow**

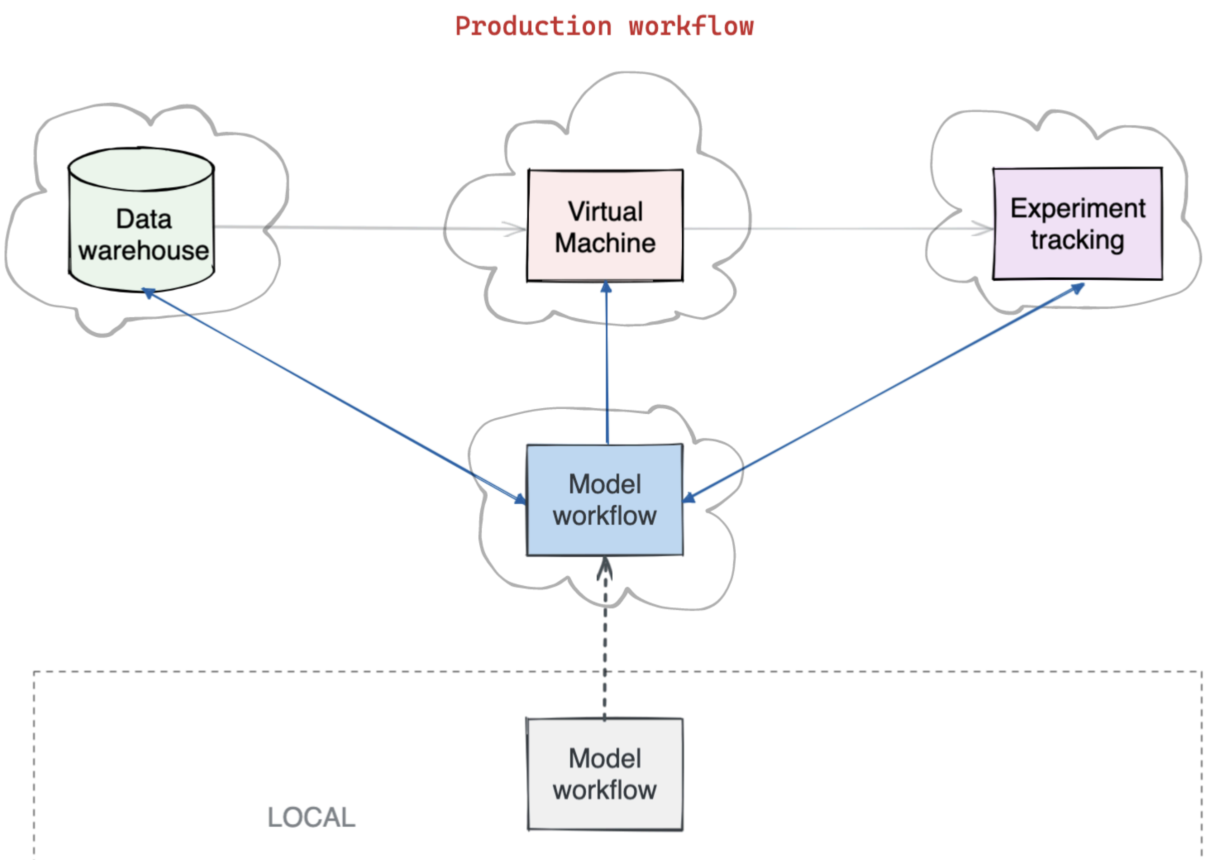
💻 Formalize the manual tasks to handle one model cycle (we will extend this further in the challenges!)

- Create the task functions + plug into existing package entry points
- Preprocess new data
- Evaluate past model performance
- Train new model

## Theory 🔭

📖 What we saw:

- Each model lifecycle is **unique**
- Some decisions cannot be automated
- Question your lifecycle vs **business needs**
- Decision to retrain the model depends on
  - Past model **performance**
  - Training **time**
  - Training **cost**




❓ **How to automate our workflow?**

👉 *Prefect* **server**

- Stores **workflow execution** parameters and results in a *database*
- Can be hosted on a **local** machine or in the **cloud**

👉 *Prefect* **UI**/*Prefect* **CLI**

- Web interface to **parametrize** and **visualize** the workflow execution

👉 *Prefect* **code**

- `prefect` *Python package*
- Create the **workflow** and **tasks**

## Livecode 🚧

🎯 Automate the model workflow
**Model Workflow**

💻 Formalize model workflow

- Create workflow by adding annotations
- Run workflow locally
- Deploy workflow to production

## Theory 🔭

📖 What we saw:

- Orchestrated tasks, scheduled or event-driven
- Production `workflow`

## Prefect

👉 **Manual** trigger vs **event-driven** vs **scheduled**

👉 Parallel vs distributed executor

👉 Handles task failure, notifications, retry patterns

👉 Fast learning curve ([Prefect vs Airflow](#)), [task library](#)

**Tasks**

```python
from prefect import task, flow

# Define your tasks
@task
def task1(*args, **kwargs):
  pass

@task
def task2(*args, **kwargs):
  pass

# Define your workflow
@flow
```

```
def myworkflow():
    # Define the orchestration graph ("DAG")
    task1_future = task1.submit(*args, **kwargs)
    task2_future = task2.submit(*args, **kwargs, wait_for = [task1_future]) # <-- task2 starts only after task1

    # Compute your results as actual python object
    task1_result = task1_future.result()
    task2_result = task2_future.result()

    # Do something with the results (e.g. compare them)
    assert task1_result < task2_result

# Actually launch your workflow
if __name__ == 'main':
    myworkflow()
```

**CLI for Self-Hosted Prefect Server**

**Self-hosted Prefect server**

```
prefect server start            # start prefect server
```

**Workflow**

```
prefect config set PREFECT_API_URL=http://127.0.0.1:4200/ap       # switch backend to self hosted

prefect agent start -q 'default'          # start local agent and pull tasks from the default queue
```

**CLI for Prefect Cloud**

**Prefect Cloud**

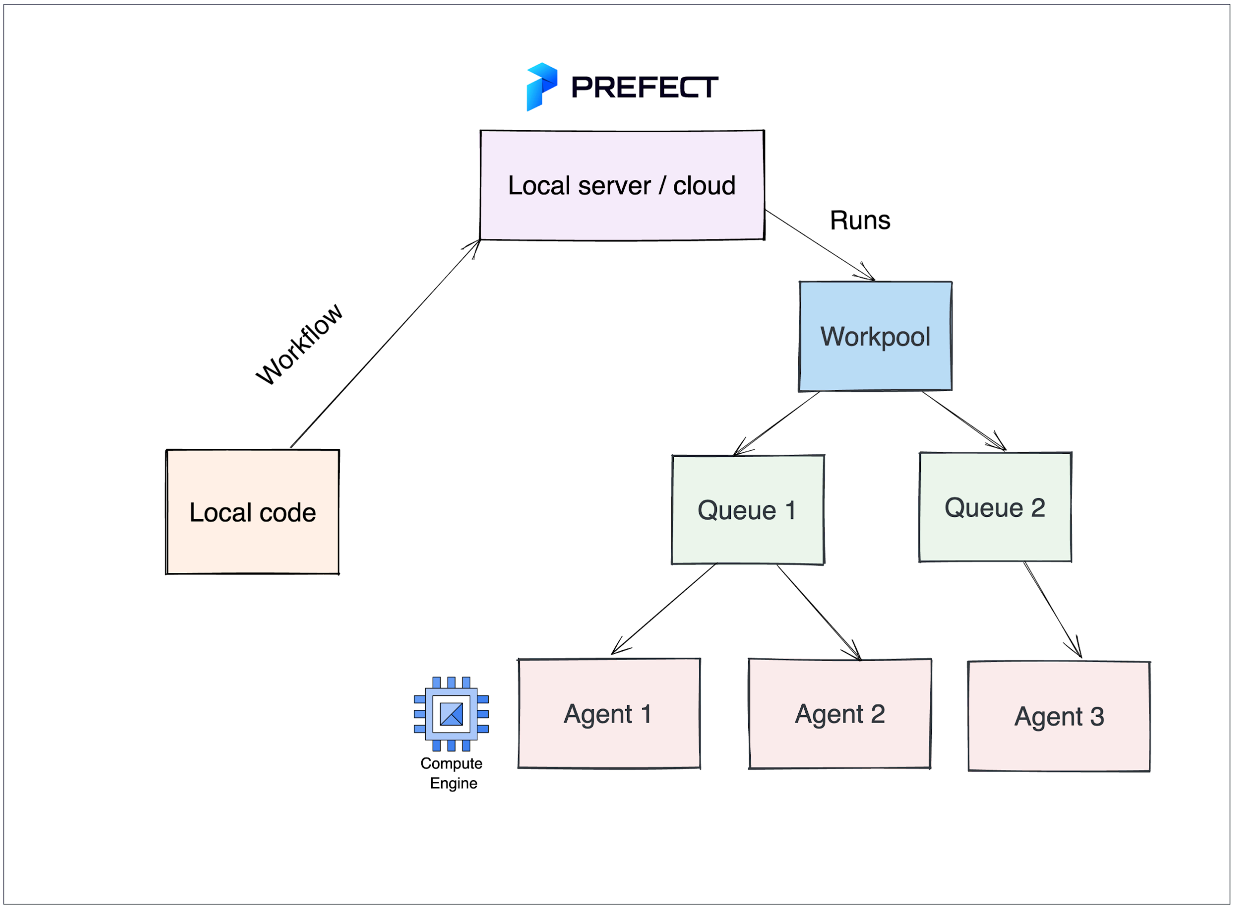
- Generous free tier (main limitation is workflow history being 7 days)
- Create an account on [Prefect Cloud]

**Workflow**

```
prefect cloud login            # switch backend to prefect cloud

prefect agent start -q 'default'          # start local agent and pull tasks from the default queue
```

## Structure with agents




## Why do we need agents ❓

- Same as **cloud training** we do not want to monopolize your machine
- We want to be able to **centralize tasks** but then **distribute to the right machine** (i.e. send runs to a gpu v non-gpu queue)
- Prioritize runs that need to happen immediately vs throughout the day!

## Why would we self-host a server vs the cloud ❓

## Cloud ☁️

- Generous free tier but lack of history
- Super easy to use!
- Pretty pricy with non personal tier

## Server 🖥️

- Can be cheaper
- More direct control over the server
- Can host on k8s
- Security/gdpr reasons

# Your turn! 🚀