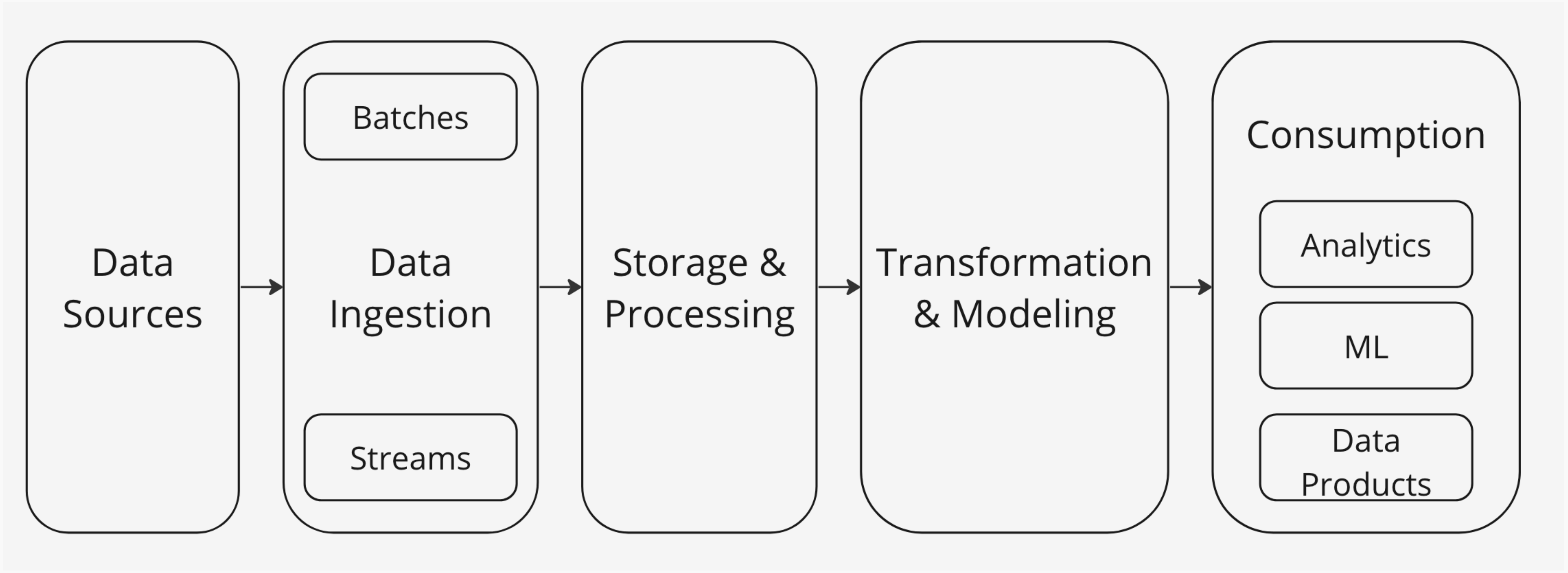
Esoon Ko
IT Högskolan

Data Engineering and Agile methods - Recap

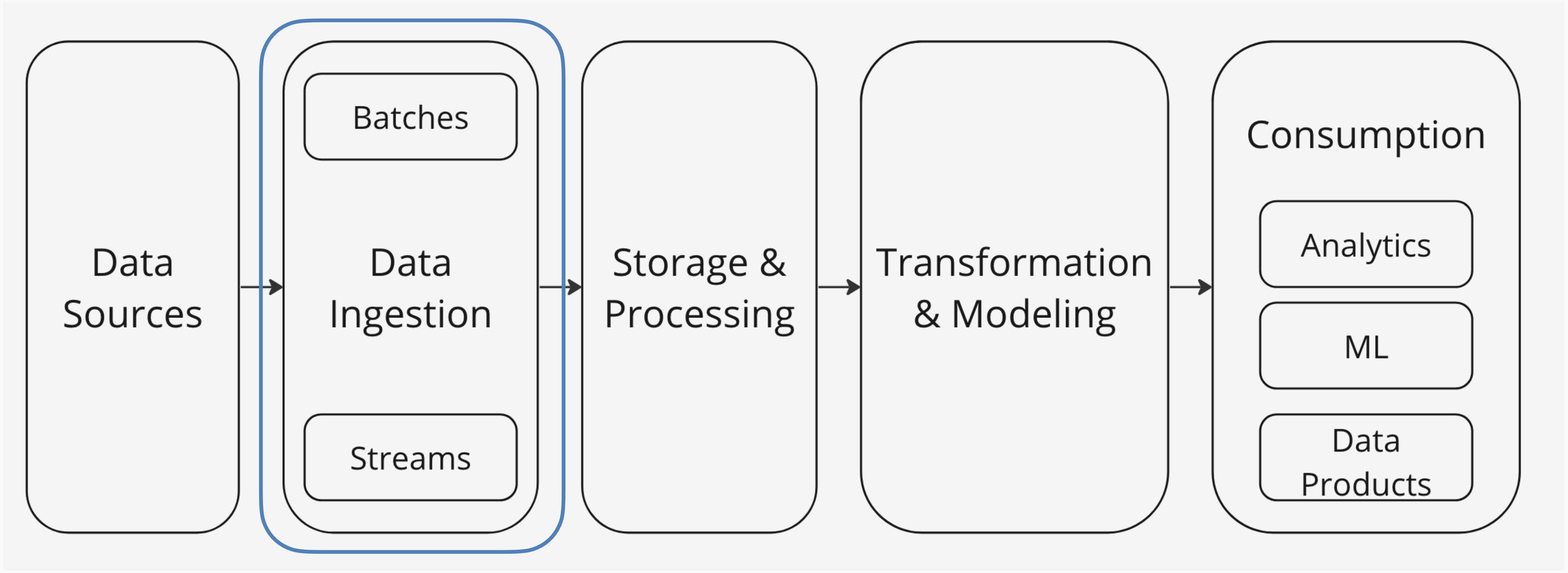
Modern Data Stack

Modern Data Stack/**Data platform**



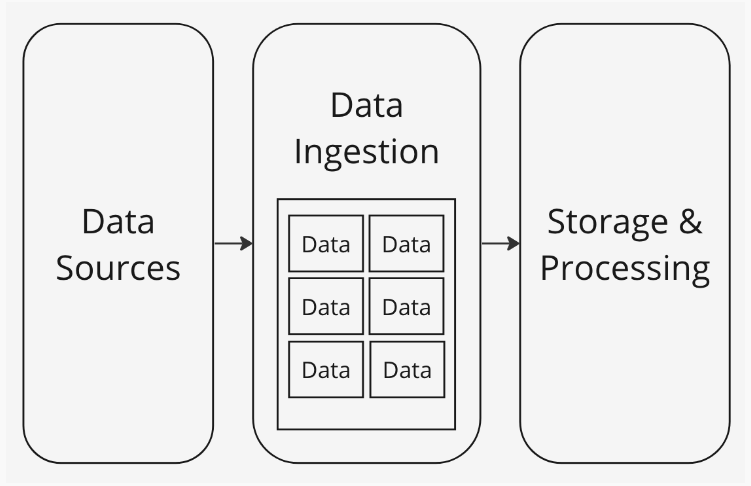
Data Ingestion

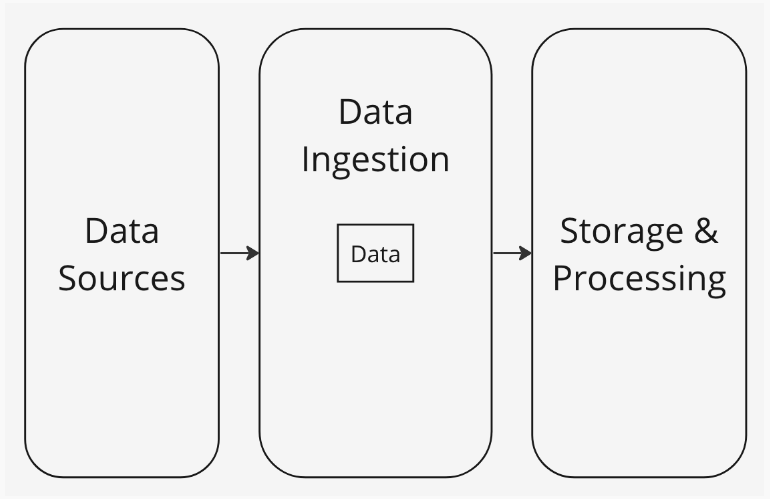
Data is ingested from source into storage.



Data Ingestion

Data is ingested from source into storage.





Batch

Data ingestion occurs from a source at a pre-defined time or in pre-defined groupings. Data is often pulled.

Benefits: Suitable for large amount of data that can be ingested in intervals.
Disadvantage: Does not provide data in real time.

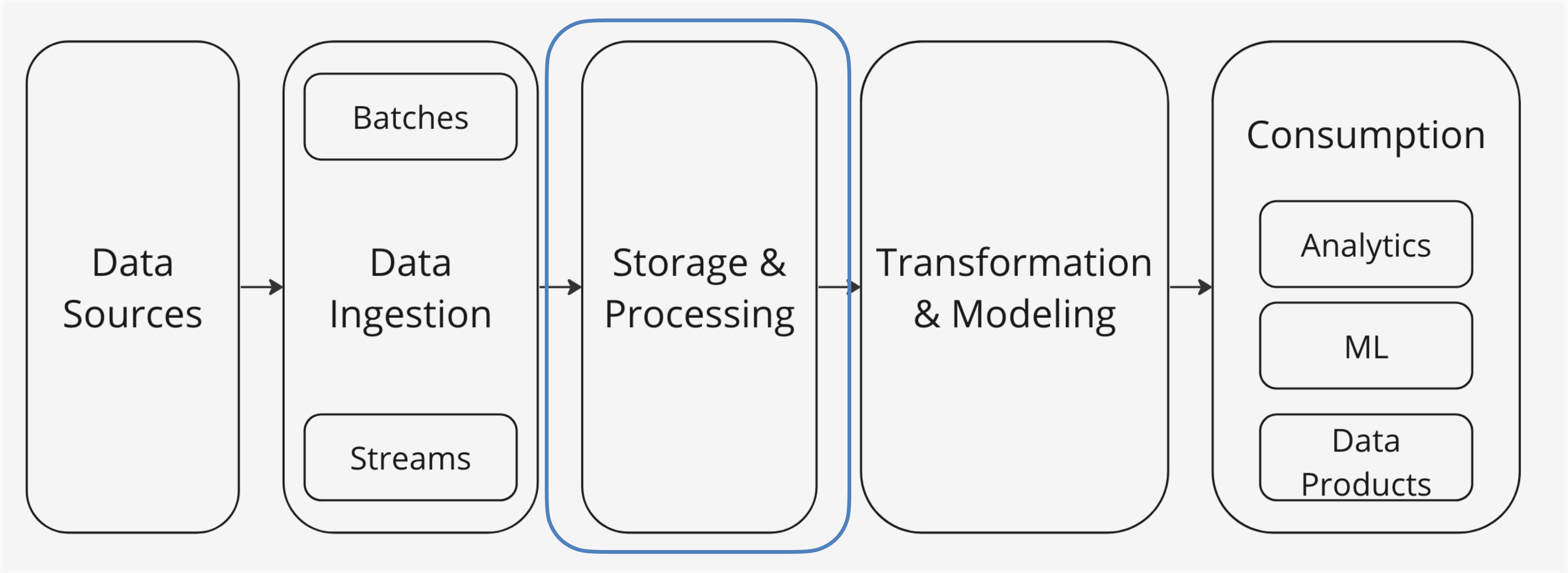
Stream

Data is ingested as soon as it is available in the source. Data is often pushed.

Benefits: Receive data in near real time.
Disadvantage: More difficult to process large amount of data that batch ingestion is capable of.

Data Storage & Processing

Where data is stored and processed.



Data Storage & Processing

Where data is stored and processed. We have many possibilities here…
On premise? Cloud?
Data warehouse? Data lake?

Data Warehouse

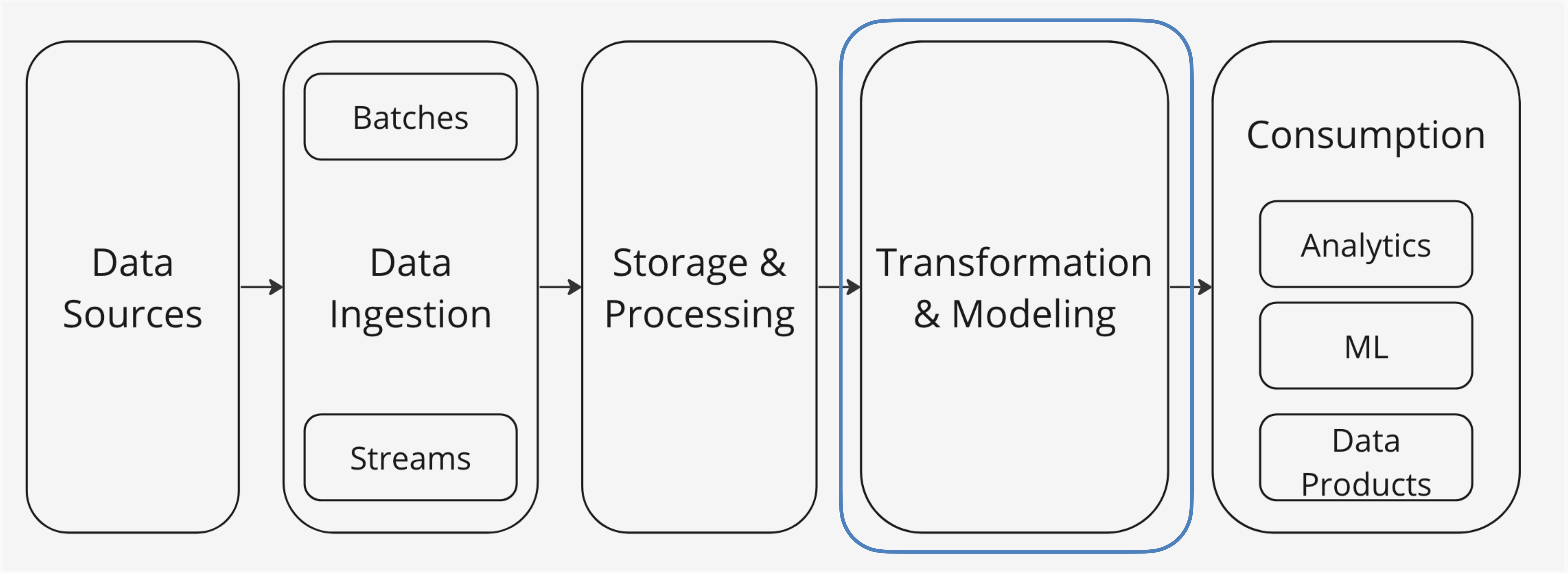
Data repository that provides data storage and compute, usually leveraging SQL queries for data analytics use cases.

Data Lake

…also a data repository that provides storage and compute. But is able to do it for structured and unstructured data. In most implementation data lakes are cloud storage that works similarly to your local file storage.

Data Transformation

Data transformation usually means cleaning raw data and enriching it in order to make it consumable for analysis or reporting.



Data Transformation

Data transformation usually means cleaning raw data and enriching it in order to make it consumable for analysis or reporting.

Common techniques used in data transformation include:

- Cleaning: This involves removing or correcting errors, inconsistencies, and missing values in the data.

- Normalization: Normalizing data involves scaling it to a common range to eliminate variations in scale that can affect analysis.

- Aggregation: Aggregating data involves summarizing detailed data into a more compact form, often by grouping it based on certain criteria and calculating summary statistics.

- Joining and Merging: Combining data from multiple sources by matching records based on common attributes.

- Derivation: Creating new variables or features from existing ones through calculations or transformations.

- Filtering: Selecting a subset of data based on specific criteria.

Data Modeling

Data modeling involves creating a conceptual representation of the structure and relationships within a dataset. It provides a blueprint for organizing and understanding data, enabling efficient storage, retrieval, and analysis. Data modeling helps ensure that data is organized efficiently, supports accurate analysis, and facilitates communication between stakeholders involved in data management and analysis processes.

Key components of data modeling include:

- Entity-Relationship (ER) Modeling: Identifying the entities (such as objects, people, or events) within a dataset and defining the relationships between them.

- Schema Design: Designing the structure of a database or dataset, including tables, fields, and constraints, based on the data model.

- Normalization: Ensuring that the database design follows normalization principles to minimize redundancy and dependency.

- Data Integrity: Enforcing rules and constraints to maintain the accuracy, consistency, and validity of data.

- Data Flow Diagrams: Visual representations of how data moves through a system or process, depicting inputs, outputs, and transformations.

SO - Basically continuation of the database course!

Data Modeling

Data modeling involves creating a conceptual representation of the structure and relationships within a dataset. It provides a blueprint for organizing and understanding data, enabling efficient storage, retrieval, and analysis. Data modeling helps ensure that data is organized efficiently, supports accurate analysis, and facilitates communication between stakeholders involved in data management and analysis processes.

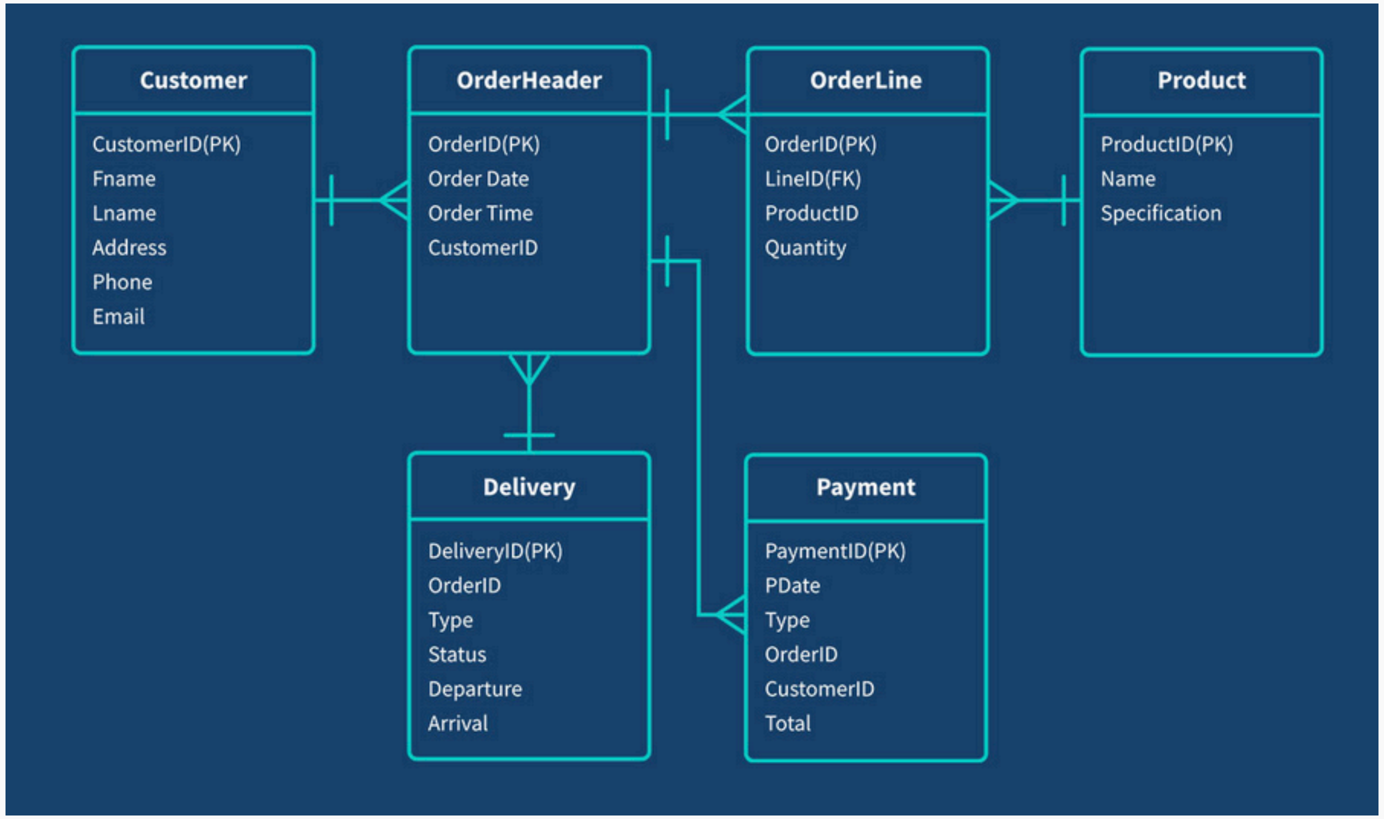


Image taken from “What is data modeling” - Qlik

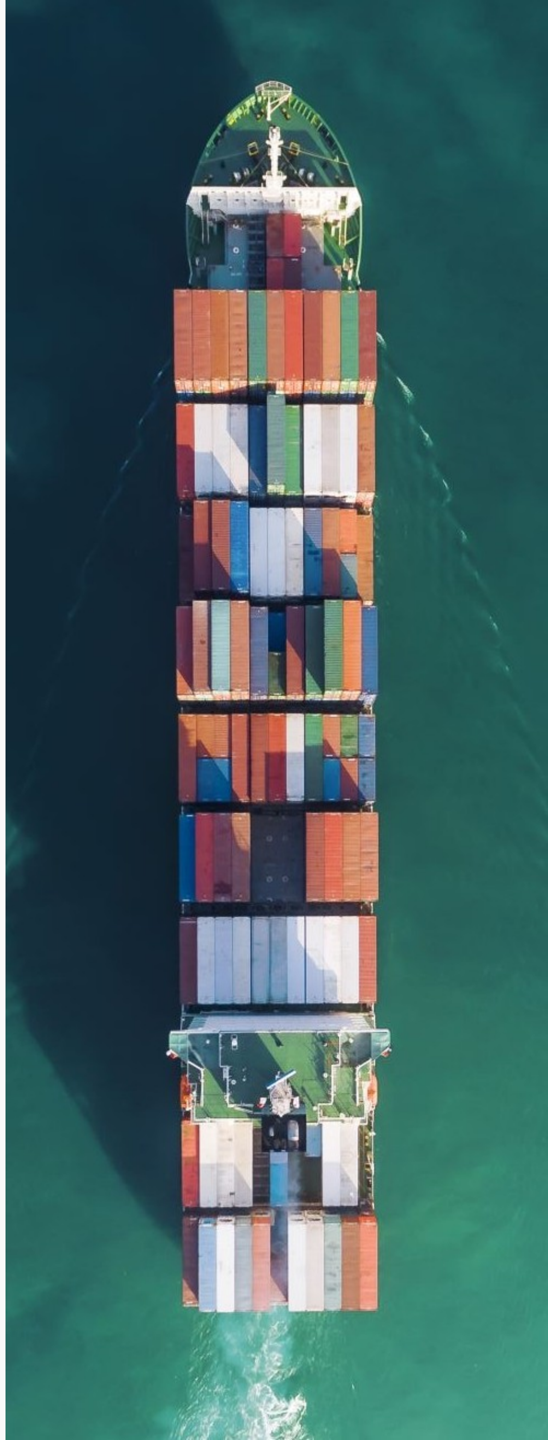
Data Consumption Layer

The data consumption layer, also known as the presentation layer or the access layer, is a crucial component is usually the final layer. Its primary function is to facilitate the access, visualization, and interpretation of data by end-users. Essentially, it serves as the interface between the underlying data sources and the users who need to interact with the data for decision-making, analysis, or reporting purposes.

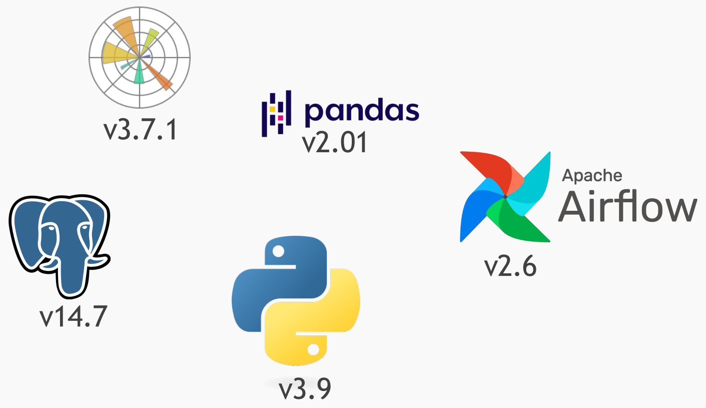
dashboard

ML model

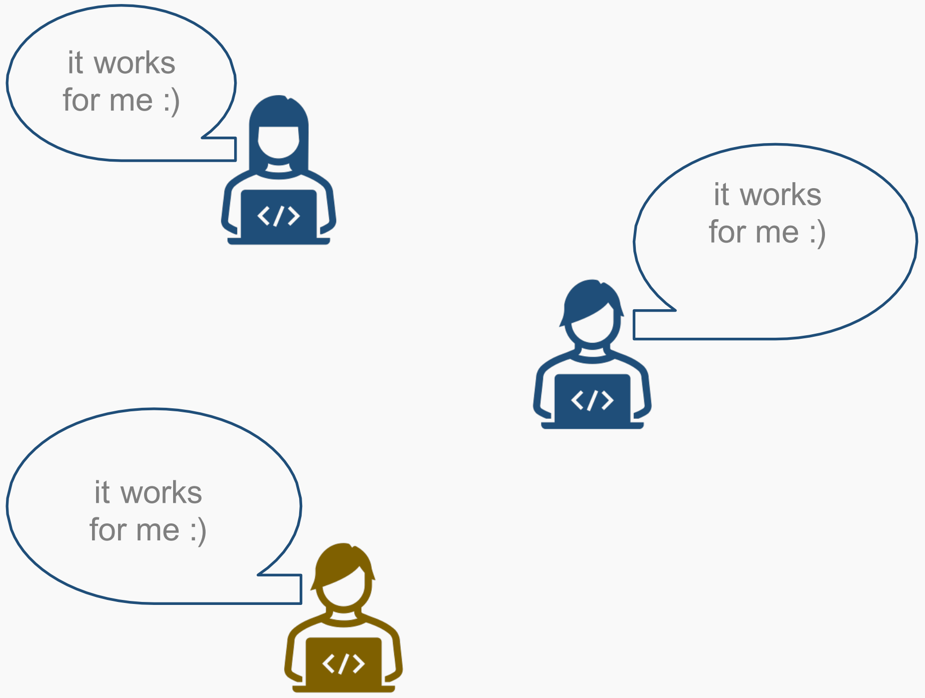
docker containers



An **environment** that works for everyone



install different softwares
and packages

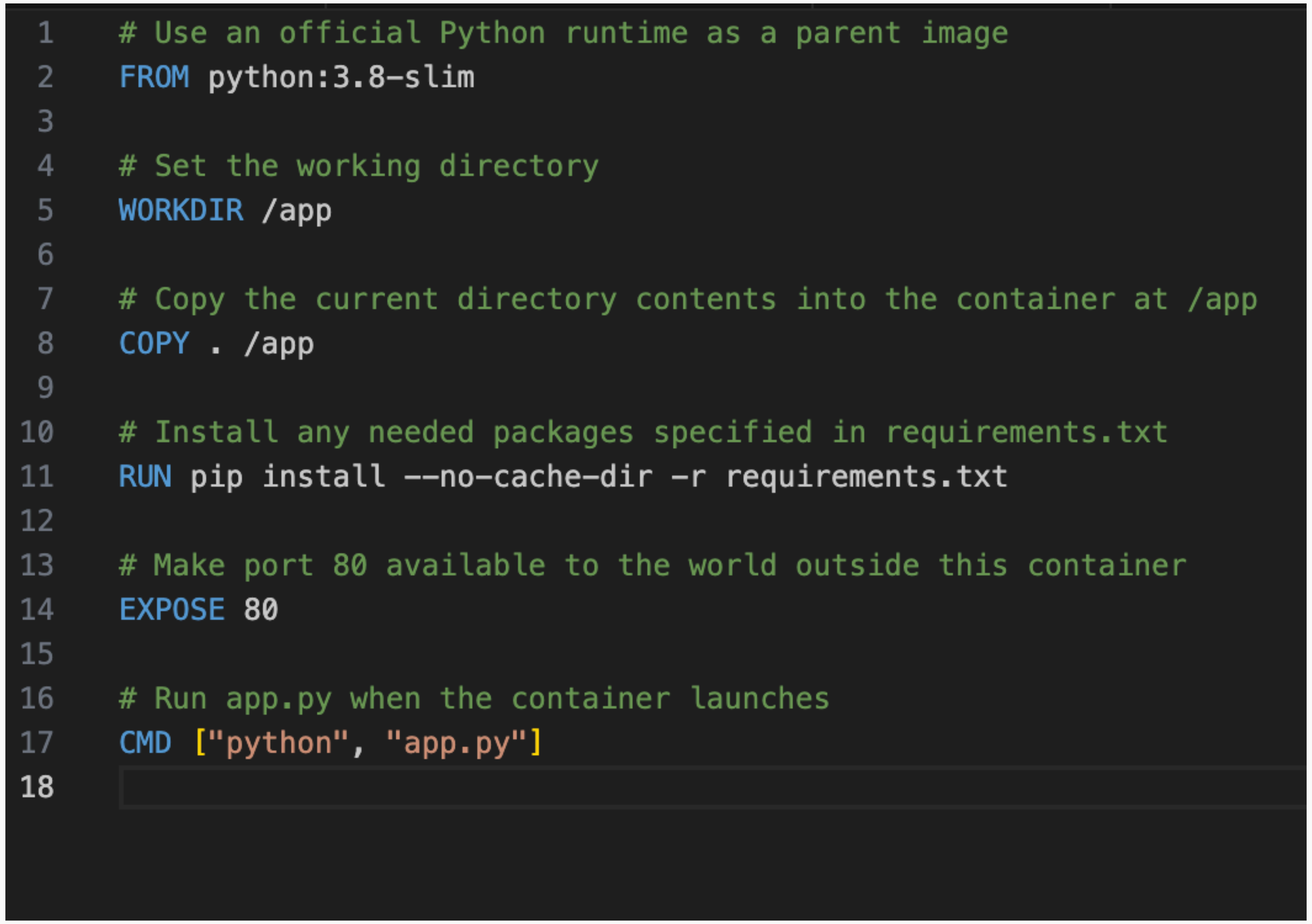


Docker **Images**

Docker image refers to lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files.

Characteristics:
- Immutable
- Layered
- Base image (Ubuntu, alpine, node and so on)

Images are made through **Dockerfile**



```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Run app.py when the container launches
CMD ["python", "app.py"]
```

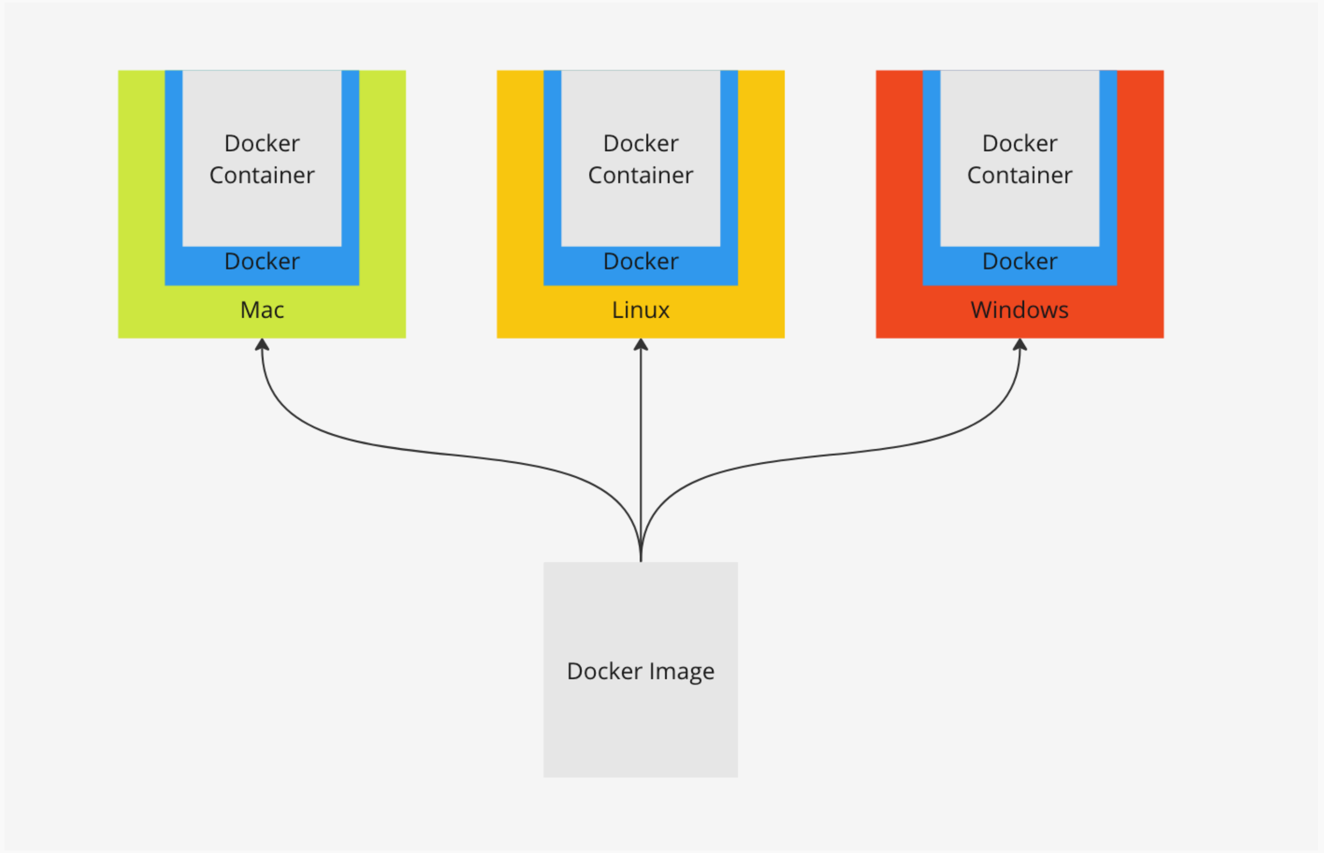
Example of Dockerfile

Docker **Containers**

Docker container refers to a runnable instance of an image. It encapsulates the application and its dependencies but shares the host system's kernel.

Characteristics:
- Easy to start, stop, move and delete
- Isolated (but shares kernel with host)

You can see images as blueprint for creating containers. You instantiate containers from an image and thereby you can create multiple containers that are the same no matter the environment.

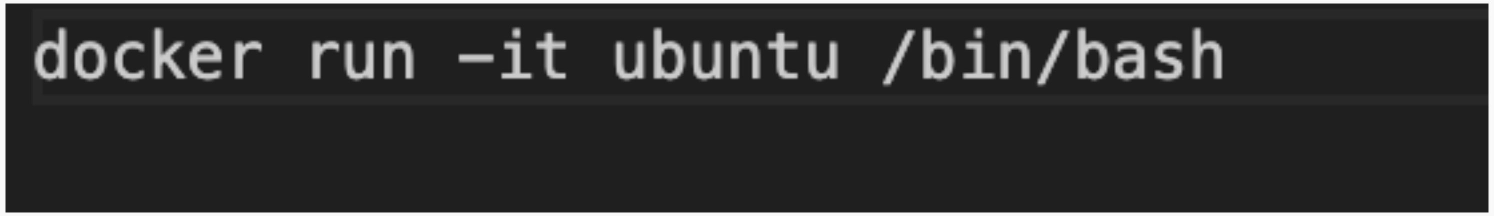


Remember this?

```
docker run hello-world
```

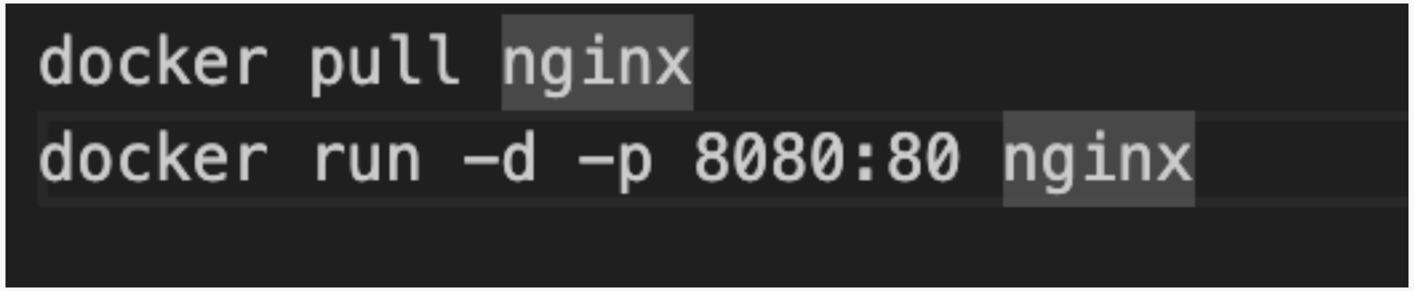
Or this?

Using Docker as Ubuntu Bash



```
docker run -it ubuntu /bin/bash
```

Using Docker as nginx server



```
docker pull nginx
docker run -d -p 8080:80 nginx
```

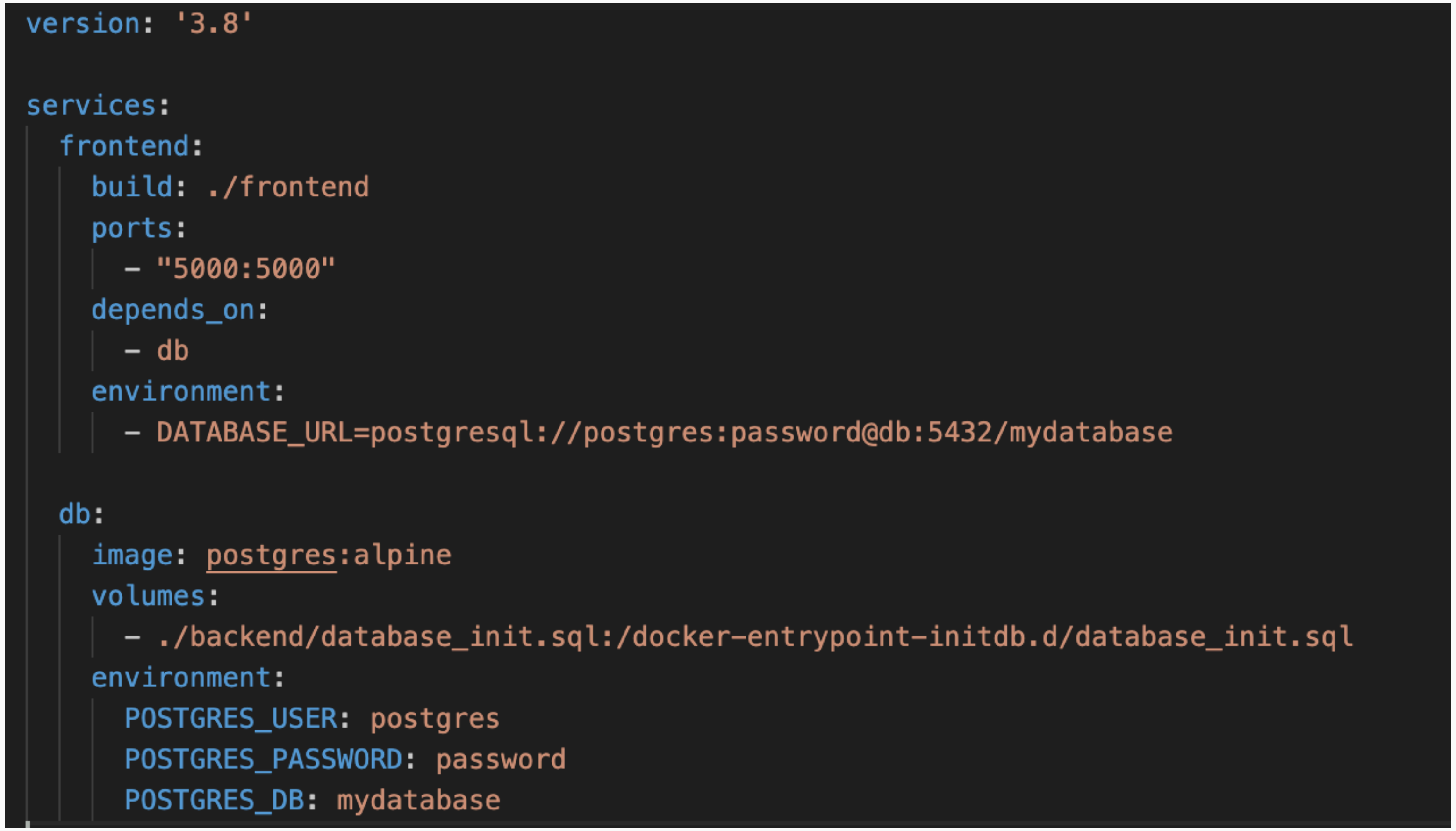
More here: https://docs.docker.com/reference/cli/docker/
Or run —help

Docker Compose

A tool for defining and running multi-container Docker applications. It allows you to use a YAML file to configure your application's services and their dependencies, then with a single command, you can create and start all the services defined in your configuration.

Done through a YAML file!

Allows for:
- Defining Services
- Linking Services
- Simple Configuration
- Lifecycle Management
- Environment Variables
- Scaling



```yaml
version: '3.8'

services:
  frontend:
    build: ./frontend
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://postgres:password@db:5432/mydatabase

  db:
    image: postgres:alpine
    volumes:
      - ./backend/database_init.sql:/docker-entrypoint-initdb.d/database_init.sql
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
```

Volume and Networking

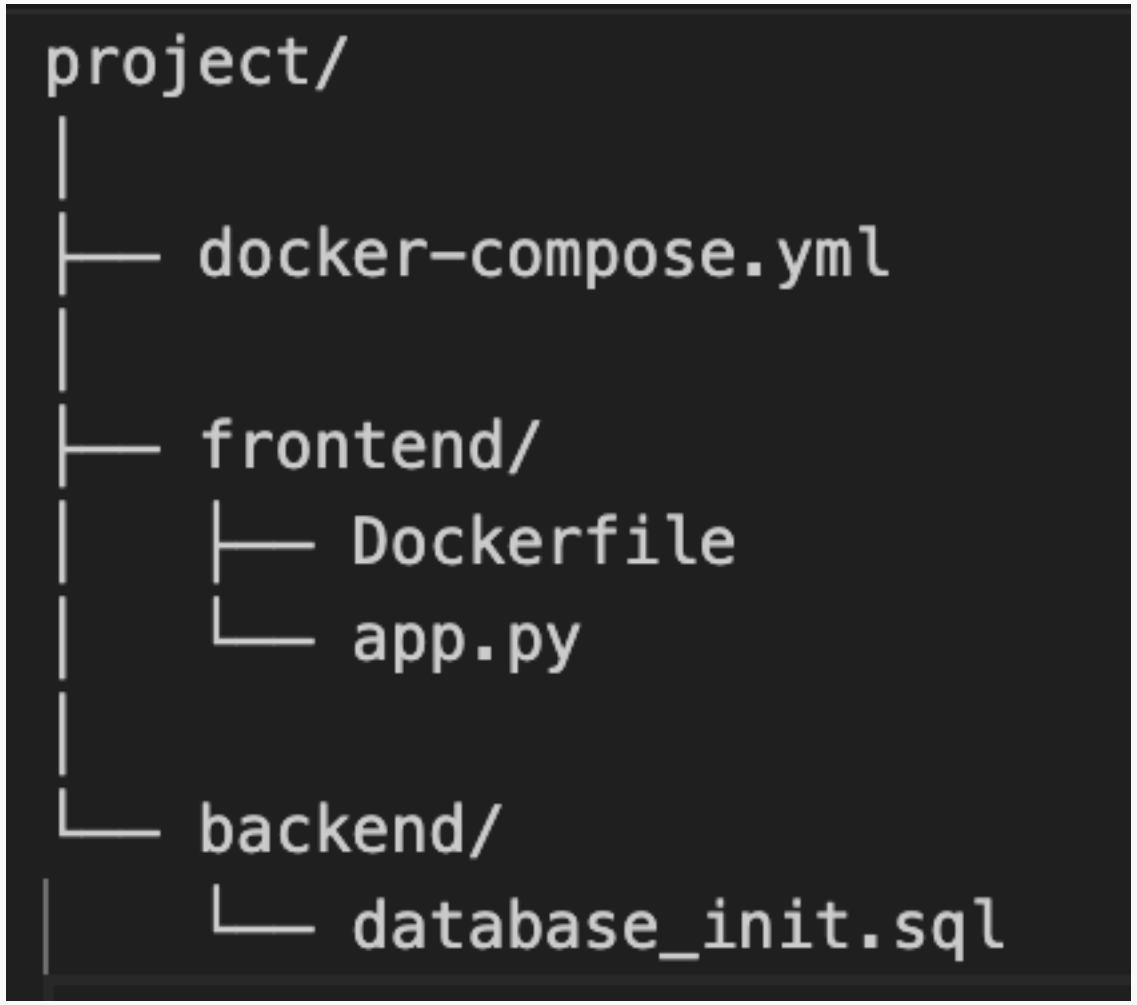
Docker storage that allows for:

- Persistent storage: Volumes are used to persist data generated by and used by Docker containers.
- Decoupling storage: Volumes allow you to decouple the storage lifecycle from the container lifecycle.

Networking allows for containers to communicate with each other

Docker Compose

- Lets revisit the code we had

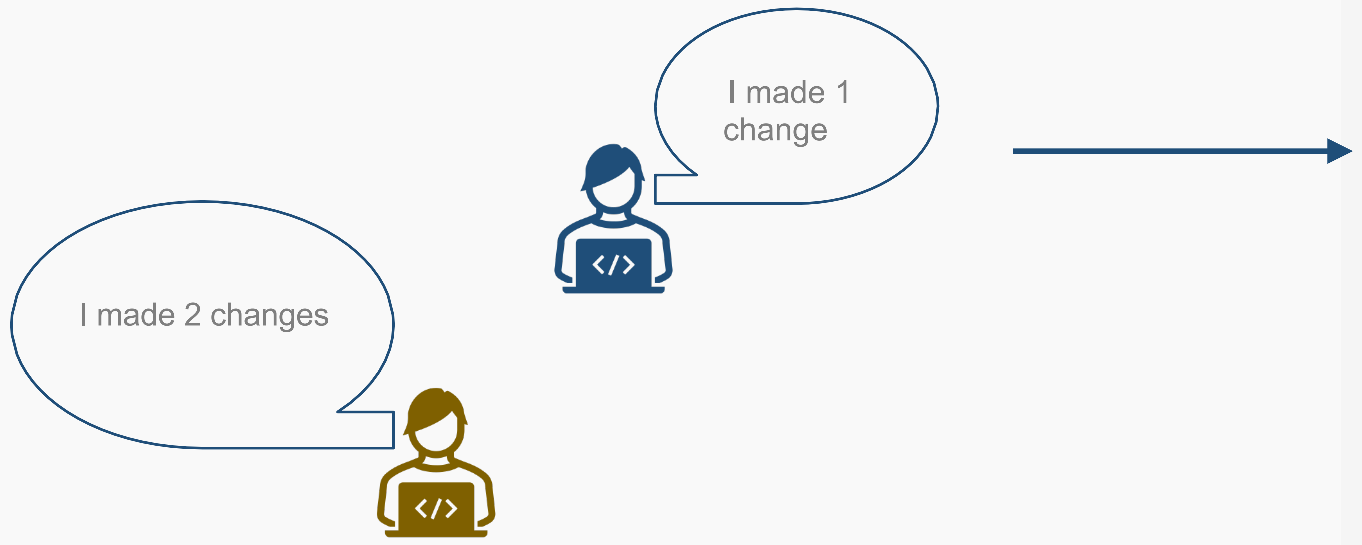


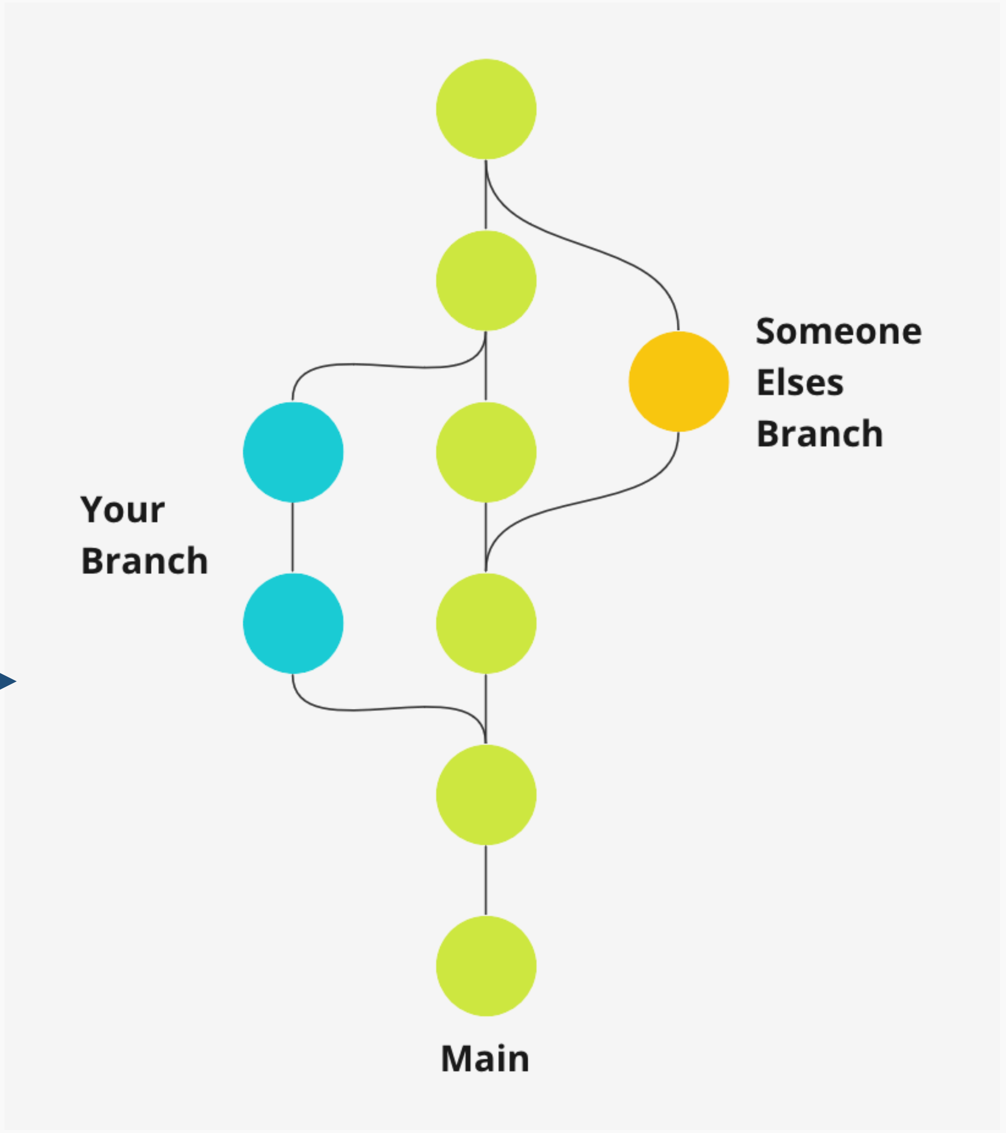
```
project/
│
├── docker-compose.yml
│
├── frontend/
│   ├── Dockerfile
│   └── app.py
│
└── backend/
    └── database_init.sql
```

Git and Version Control

We need a way to **keep track of changes**

We do that through **version control**

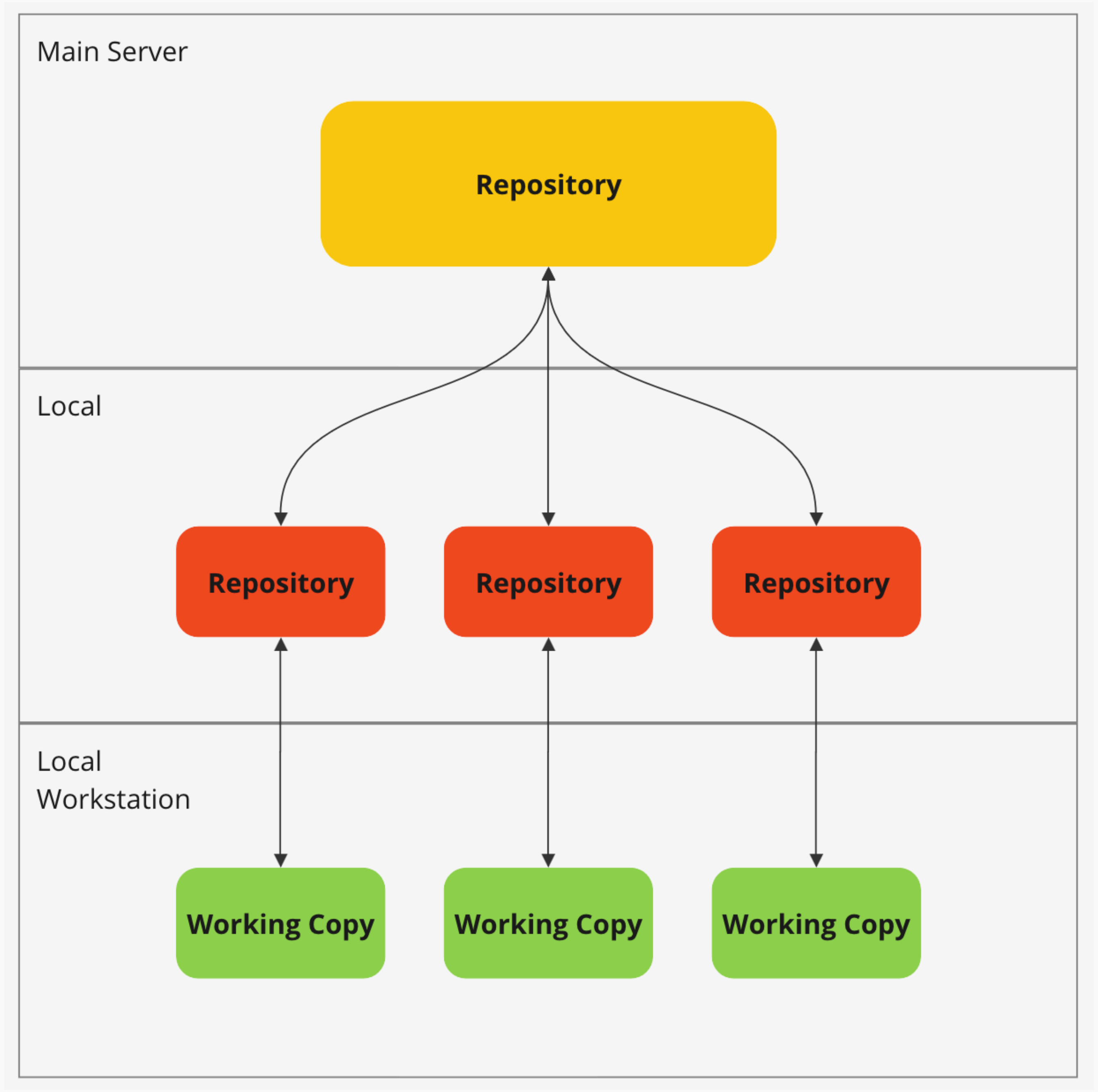




Repo

Storage location where your project's files and the entire history of changes are kept.

Think of it as a project's folder that includes all the files and the history of modifications



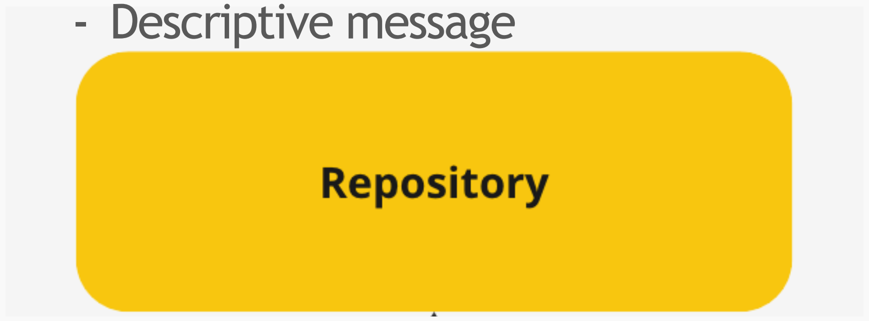
Commit

A commit is a snapshot of your repository at a specific point in time.

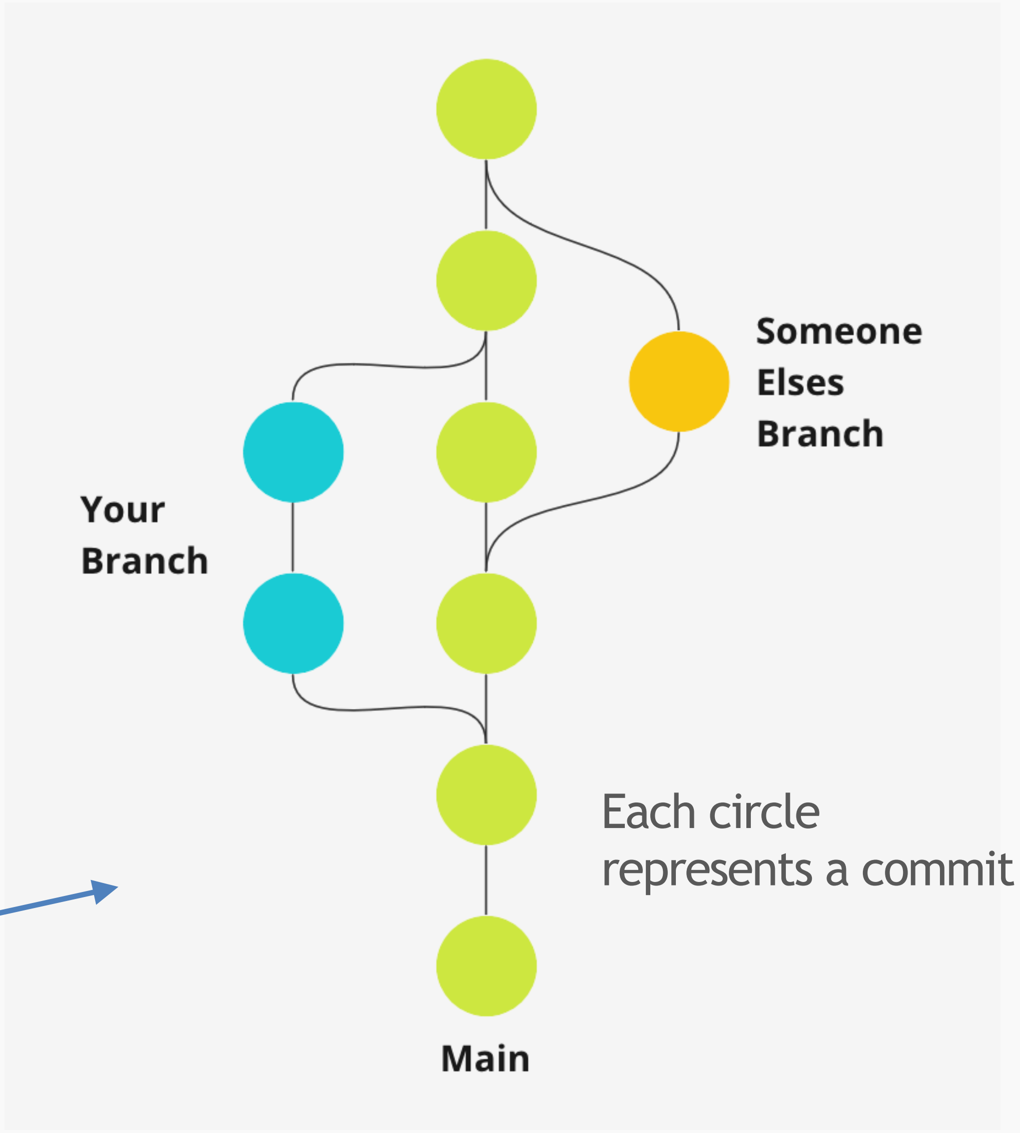
Every change we want to keep we commit.

Best practice - USE MESSAGE!

- Bad Commit:
    - Add all changes in one commit
    - Messages like “Fixed it…”, “It works…”
- Good Commit:
    - Relevant changes in a commit
    - Descriptive message



Inside the repo

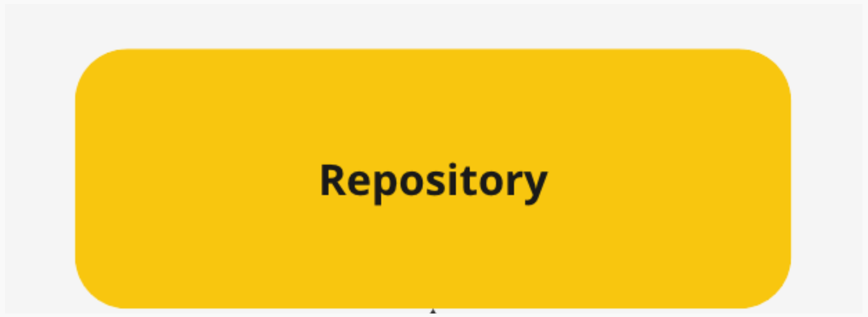


Each circle represents a commit

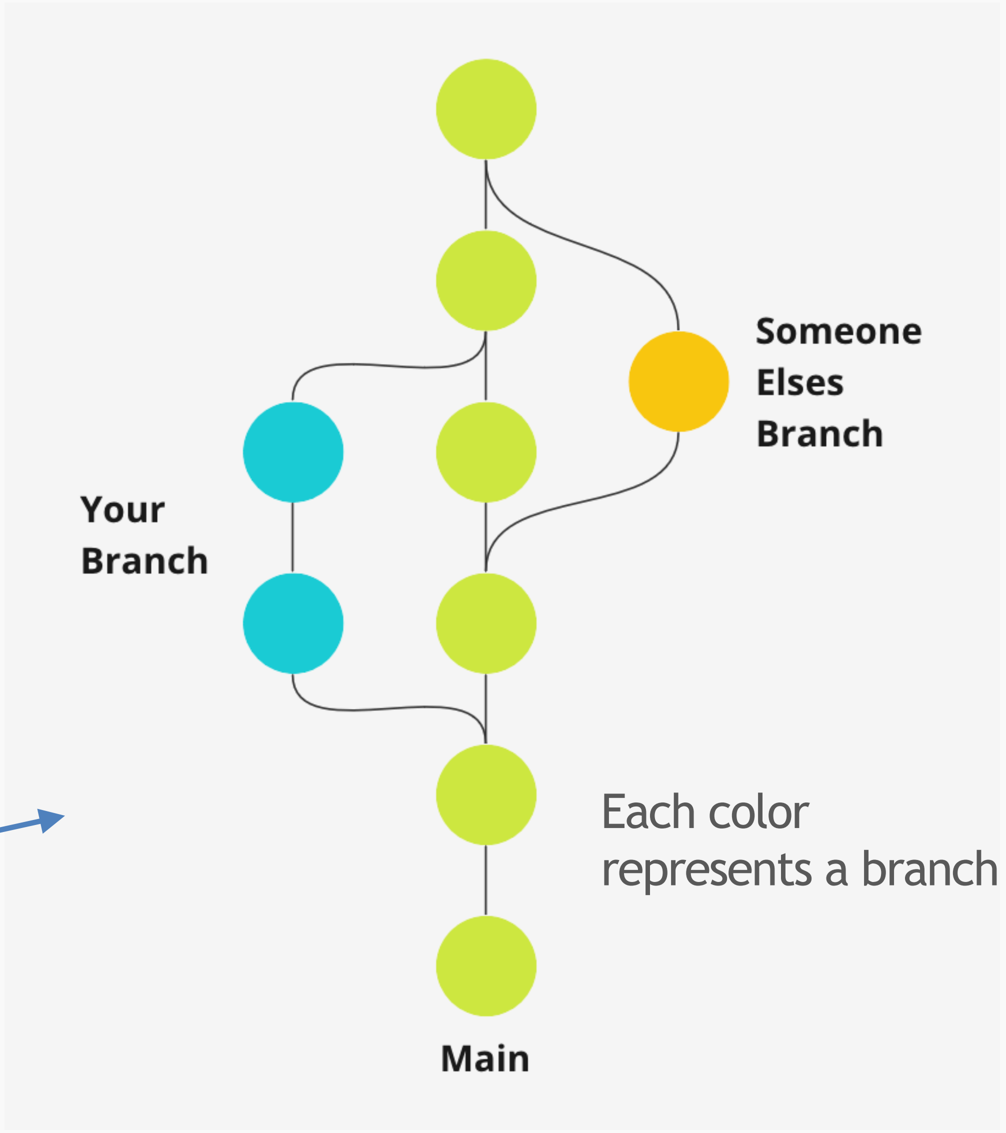
Branch

A parallel version of the repository. It allows you to work on different versions of a project simultaneously.

As the name suggests, a different path that the project takes

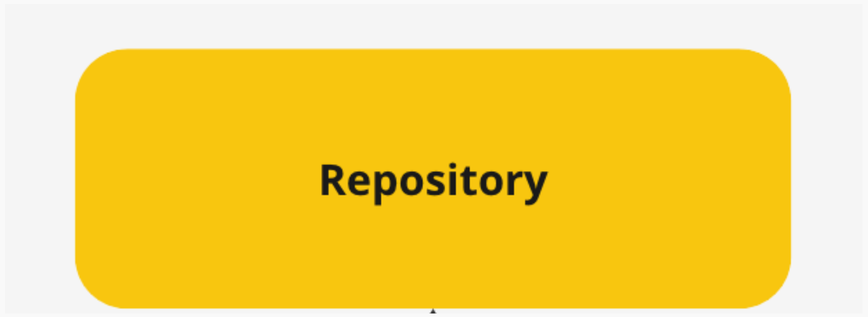


Inside the repo

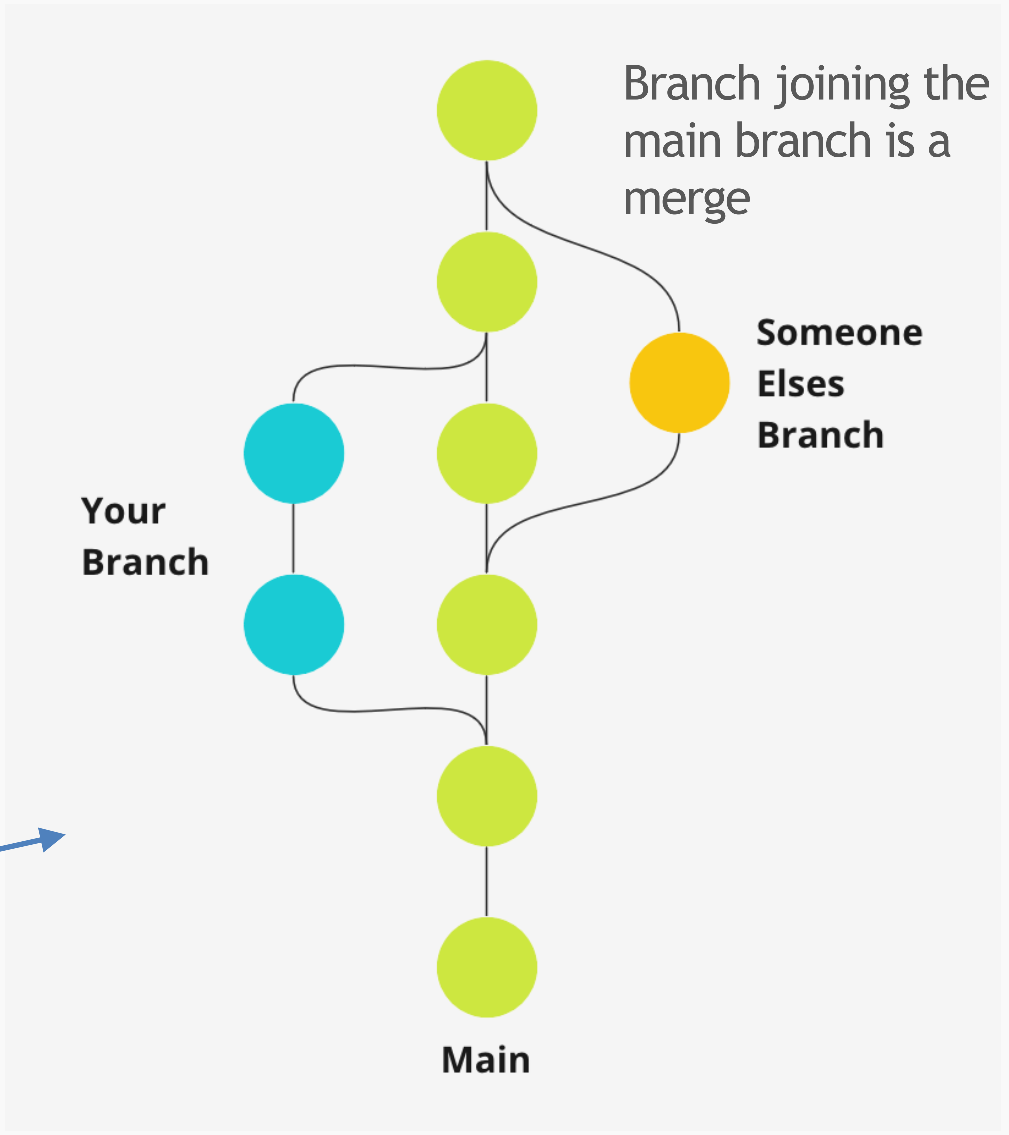


Merge

Merging is the process of integrating changes from one branch into another.



Inside the repo

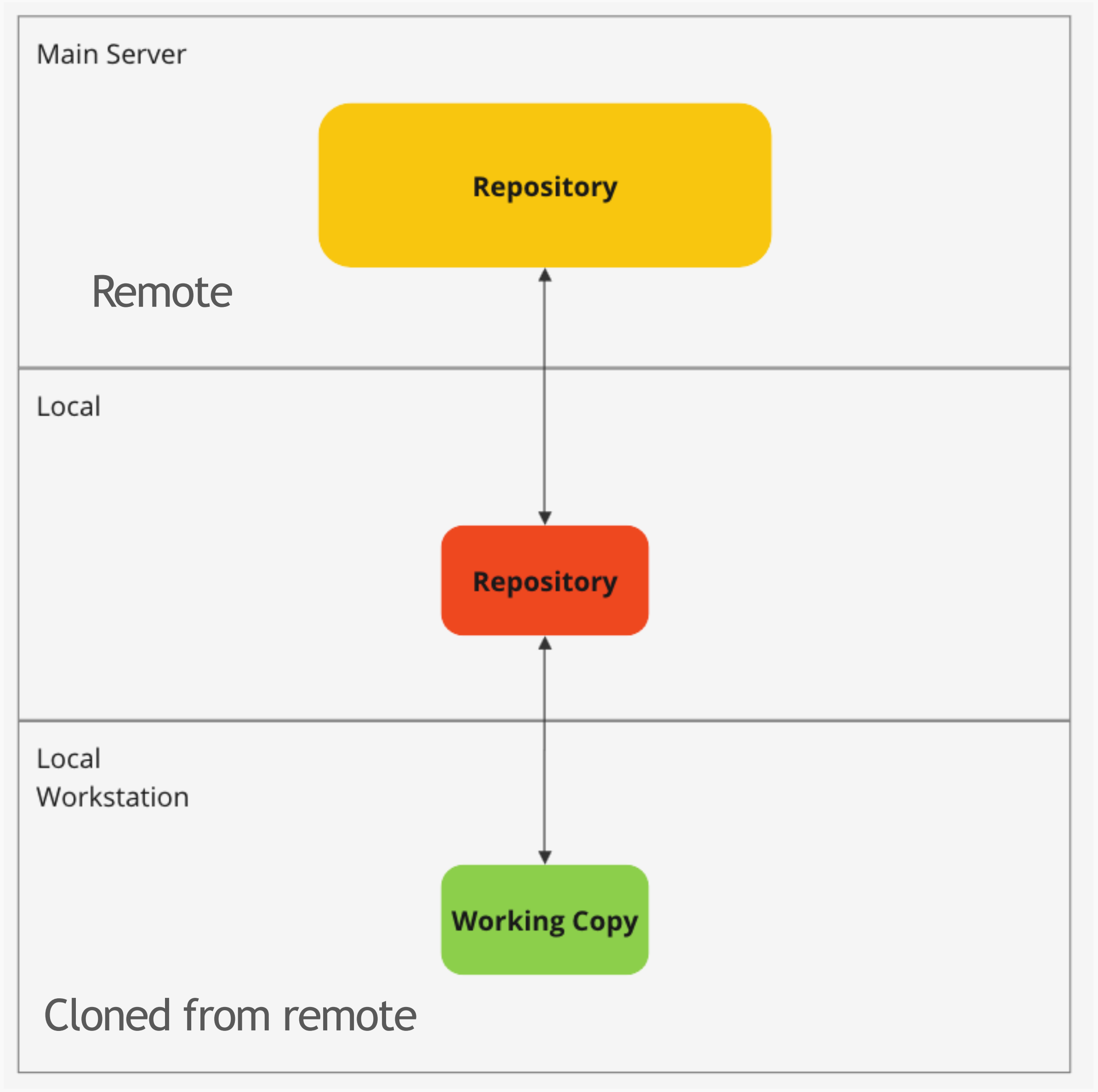


Remote

A remote is a common repository stored on a server, allowing team members to collaborate.

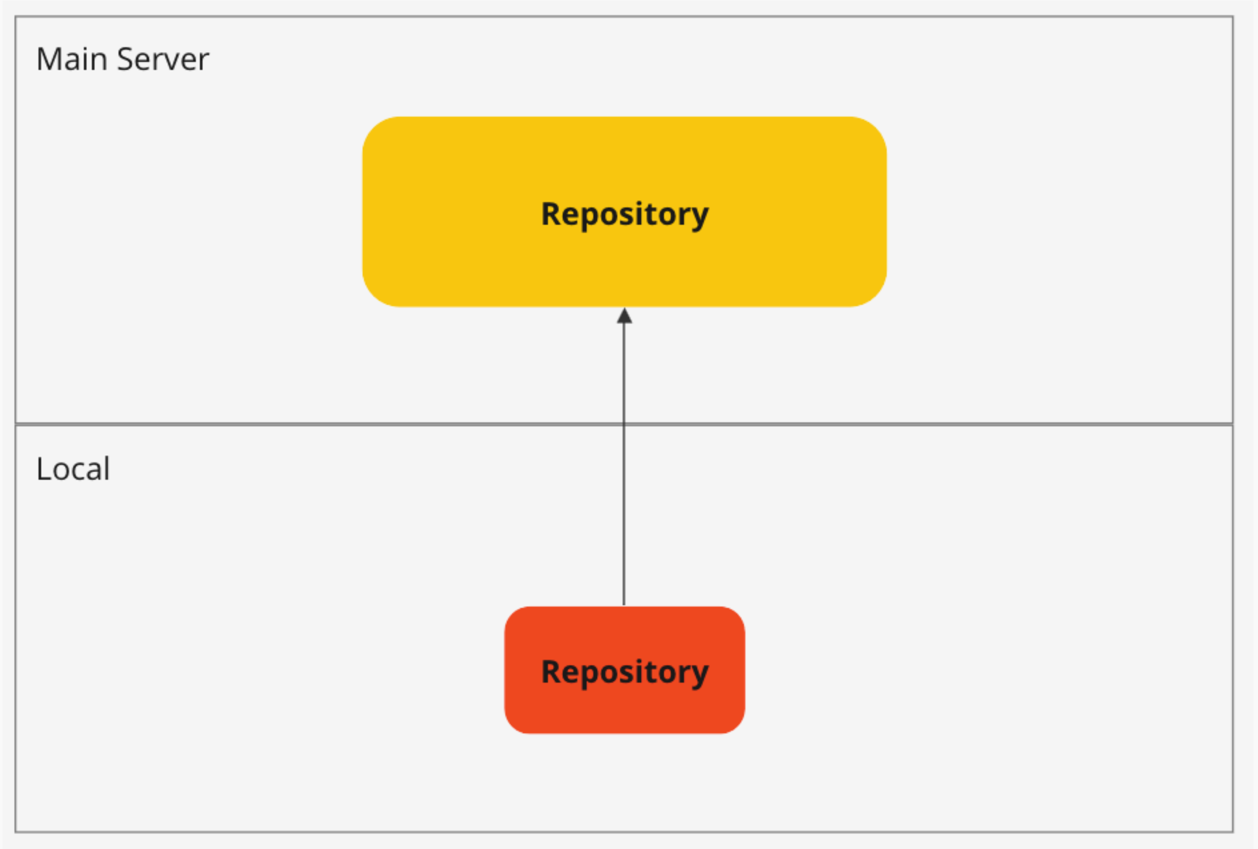
Clone

Cloning is creating a copy of an existing repository.



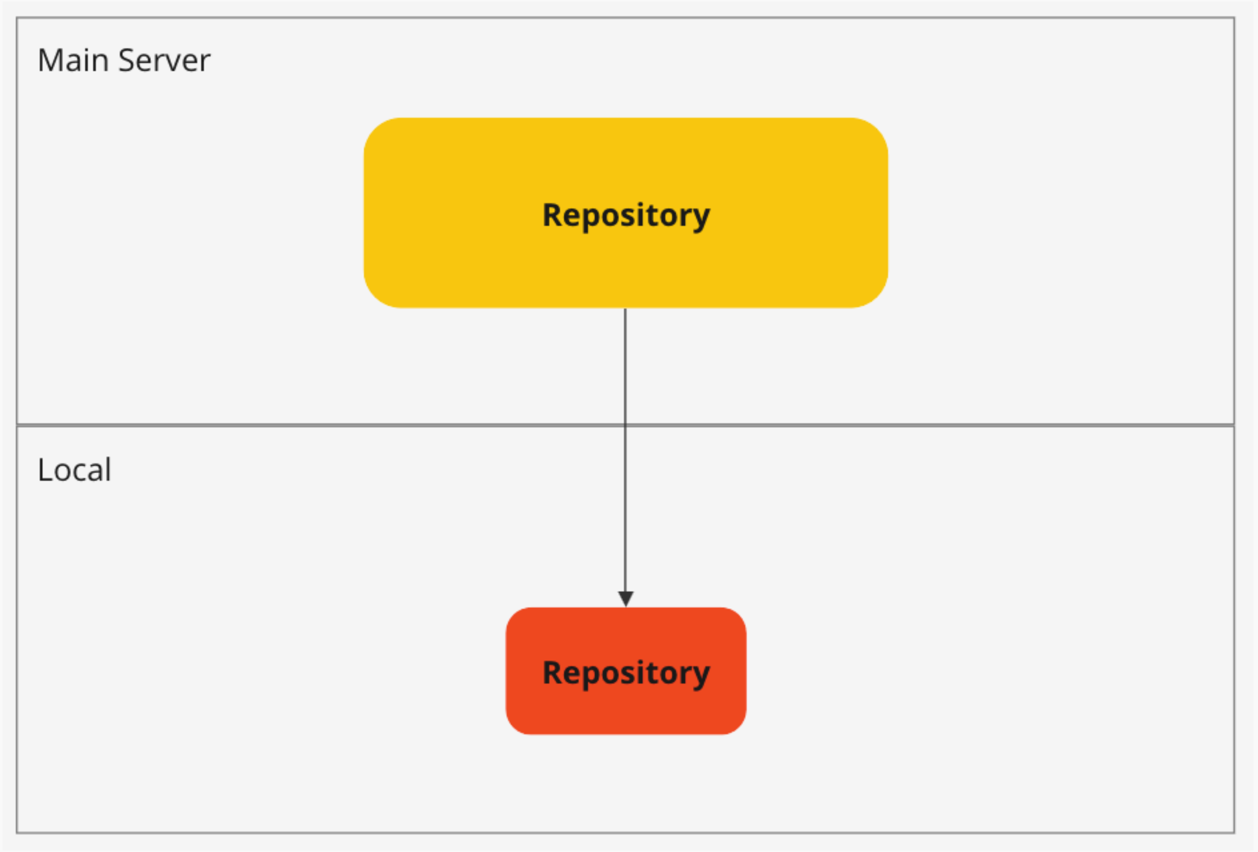
Push

Pushing is sending your committed changes to a remote repository.



Pull

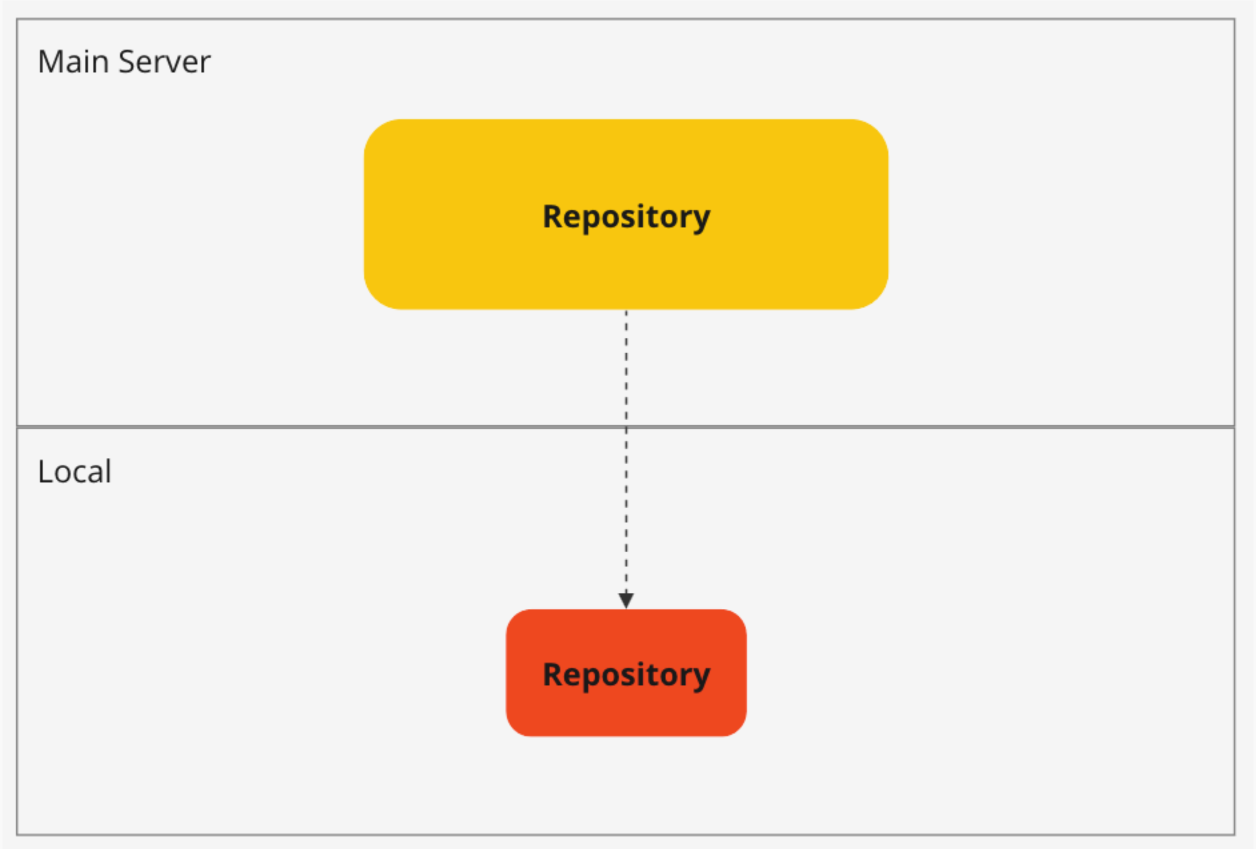
Pulling is fetching and integrating changes from a remote repository into your local repository.



Fetch

Fetching is downloading changes from a remote repository without integrating them into your local repository.

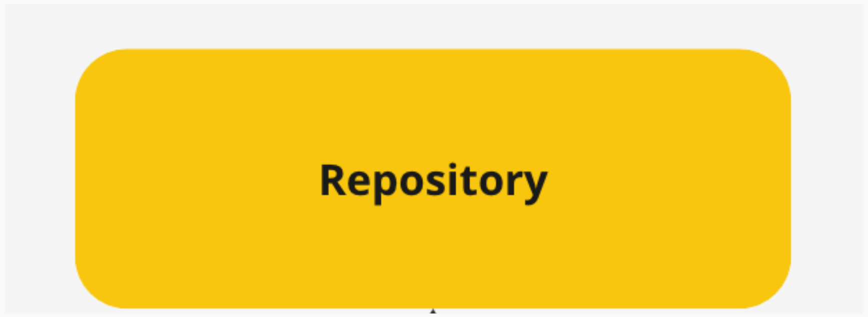
Fetching is like checking for updates on a document but not yet merging them into your current version.



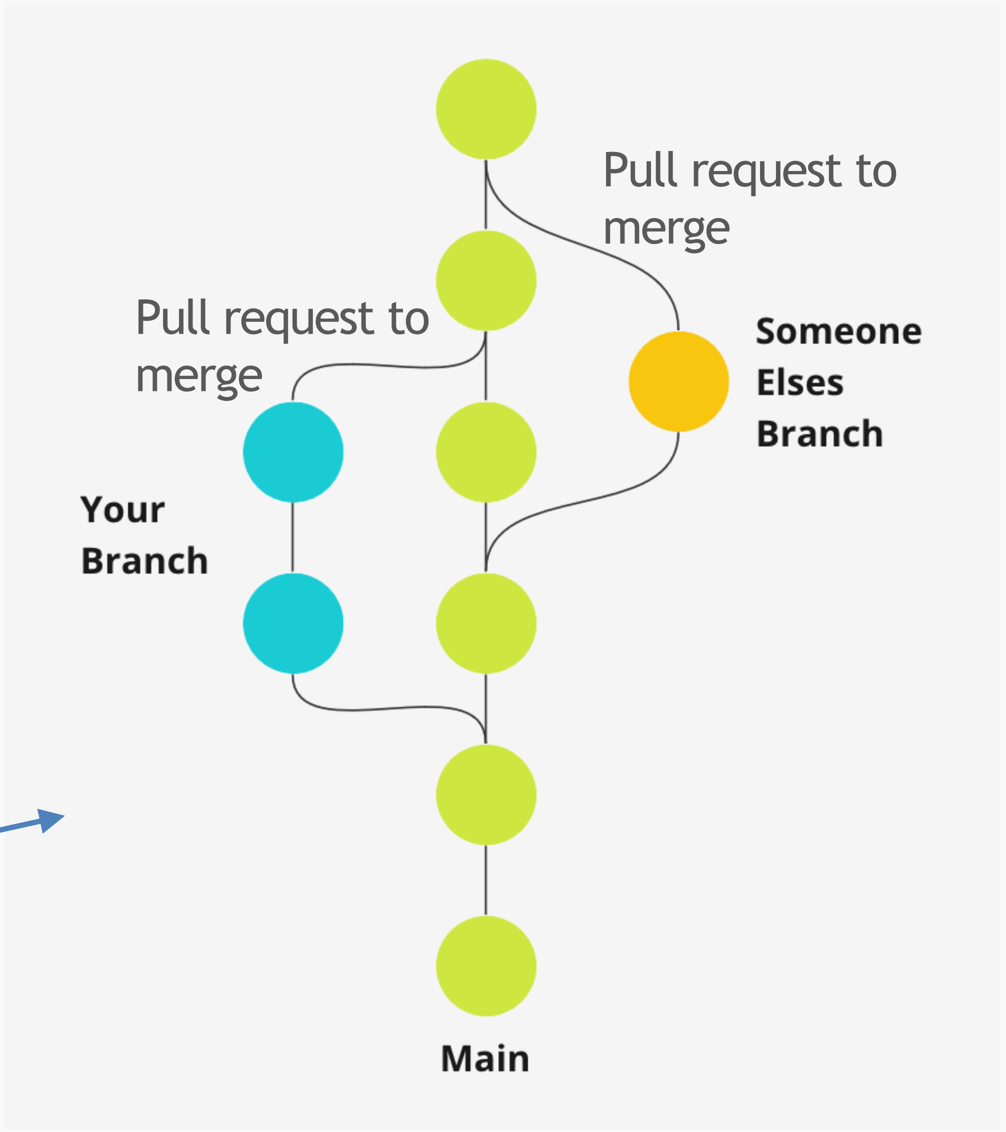
Pull Request

A pull request is a method of submitting contributions to a project. It is a request to merge your changes into the main repository.

Think of a pull request as a proposal to add your changes to the shared project, asking others to review and accept your updates.



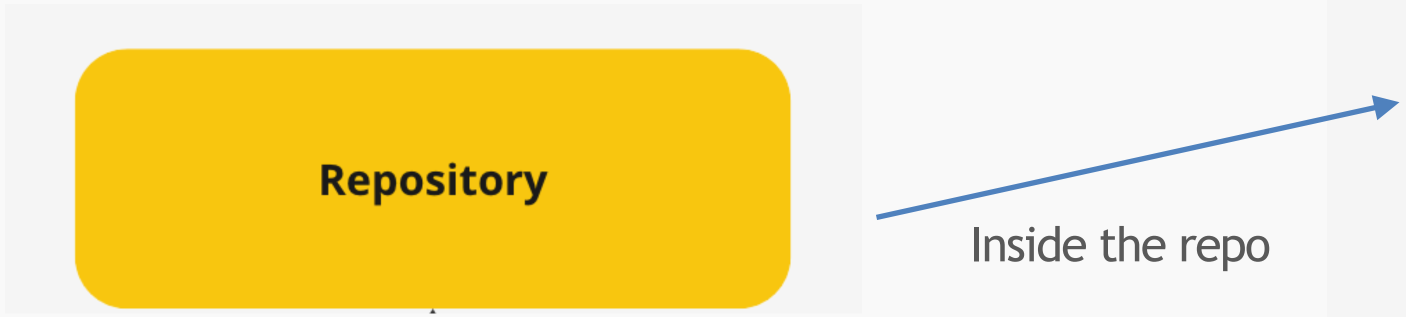
Inside the repo

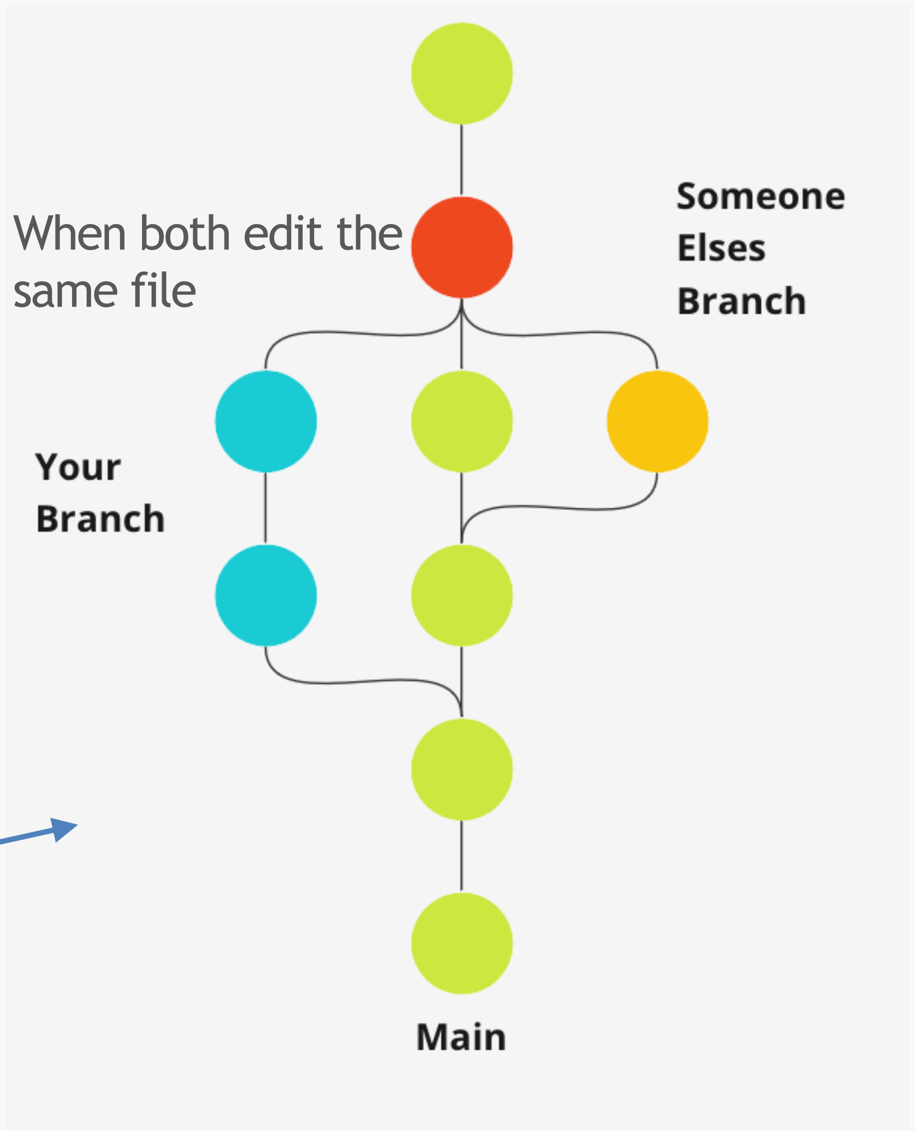


Conflict

A conflict occurs when changes in different branches contradict each other, requiring manual resolution.

Think of a conflict as two different edits on the same paragraph in a document, where you need to decide how to combine them.

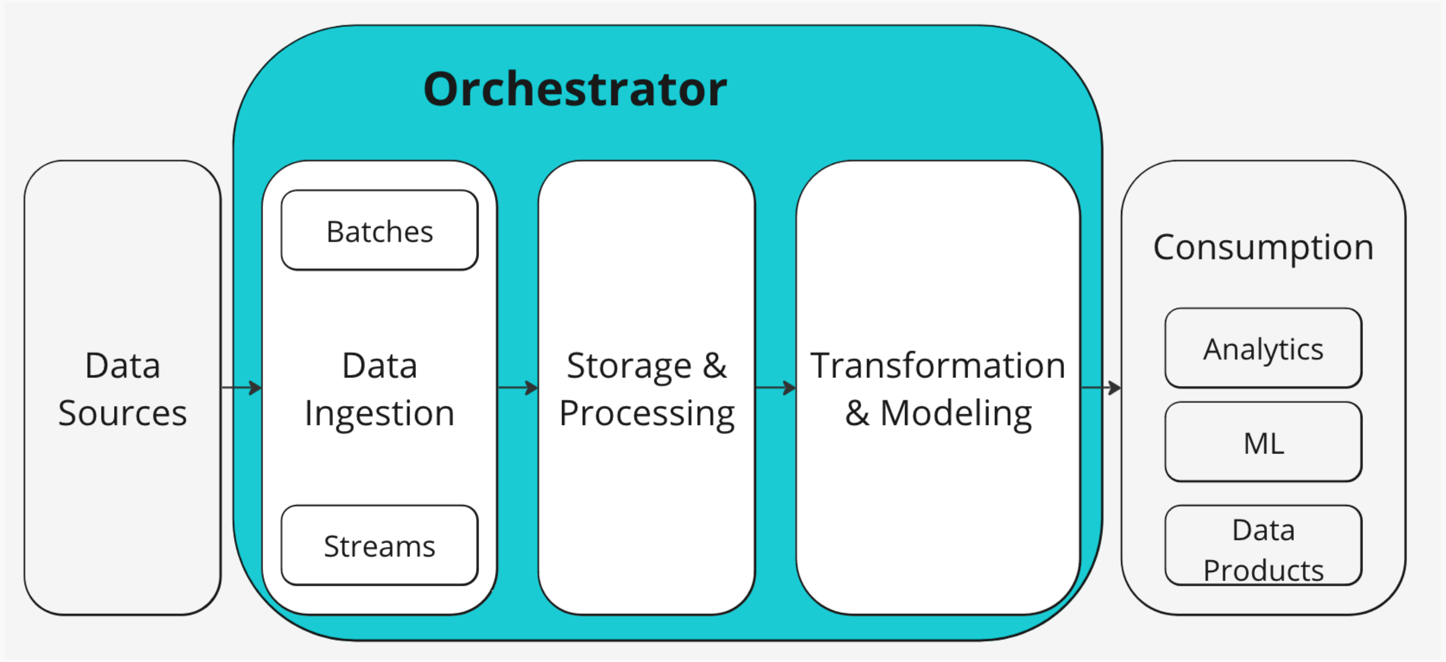




Airflow and Orchestration

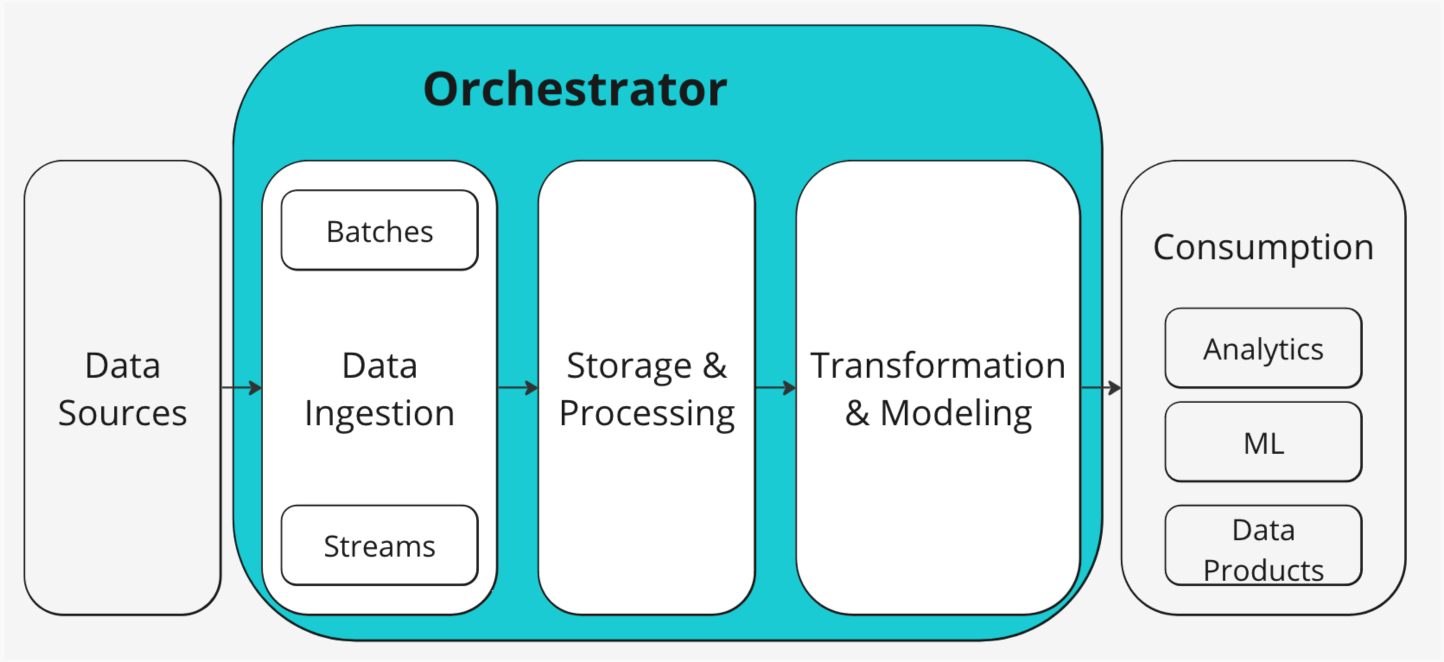
Orchestration

Automated process of managing, coordinating, and optimizing data pipelines and workflows. It ensures that data flows efficiently and accurately from one system to another, maintaining quality and integrity.



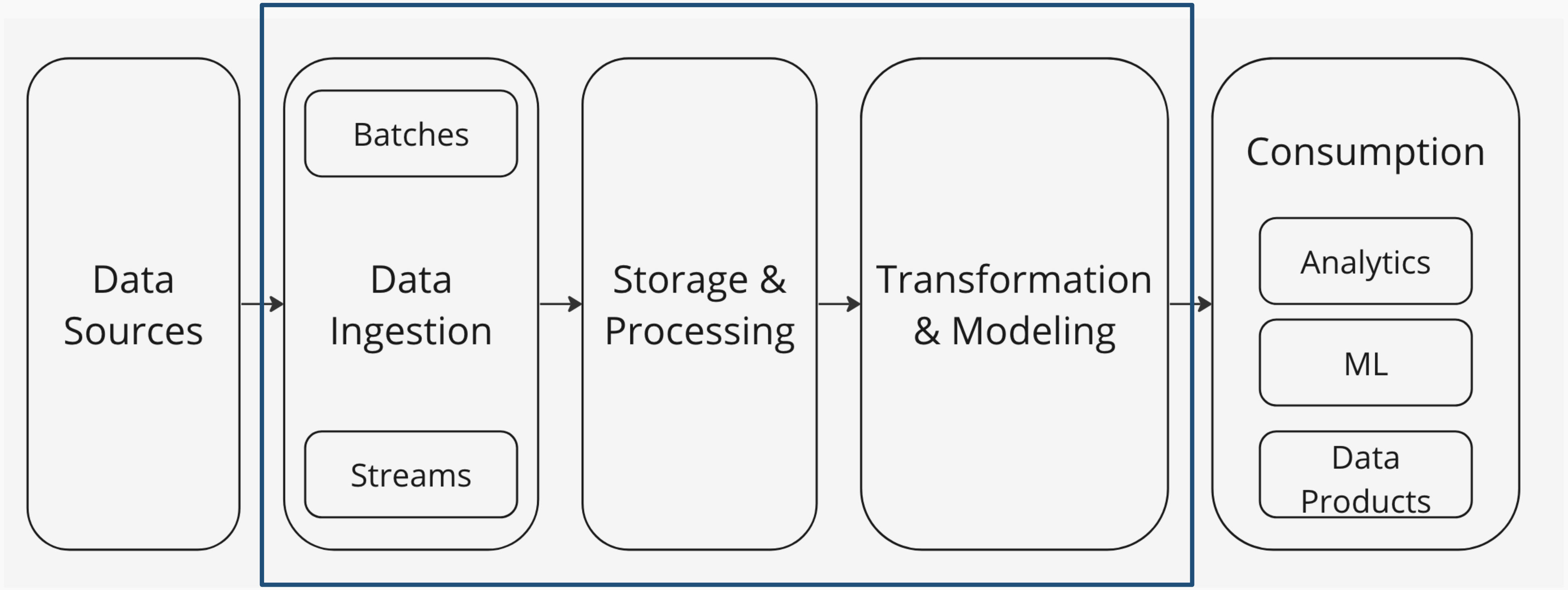
Why **Orchestration?**

- **Efficiency**: Reduces manual intervention and automates repetitive tasks.
- **Scalability**: Handles large volumes of data and complex workflows.
- **Reliability**: Ensures consistent and error-free data processing.
- **Agility**: Quickly adapt to changes in data sources, processing needs, or business requirements.



Workflow

A sequence of tasks.
The data lifecycle we drew up can be seen as containing a workflow



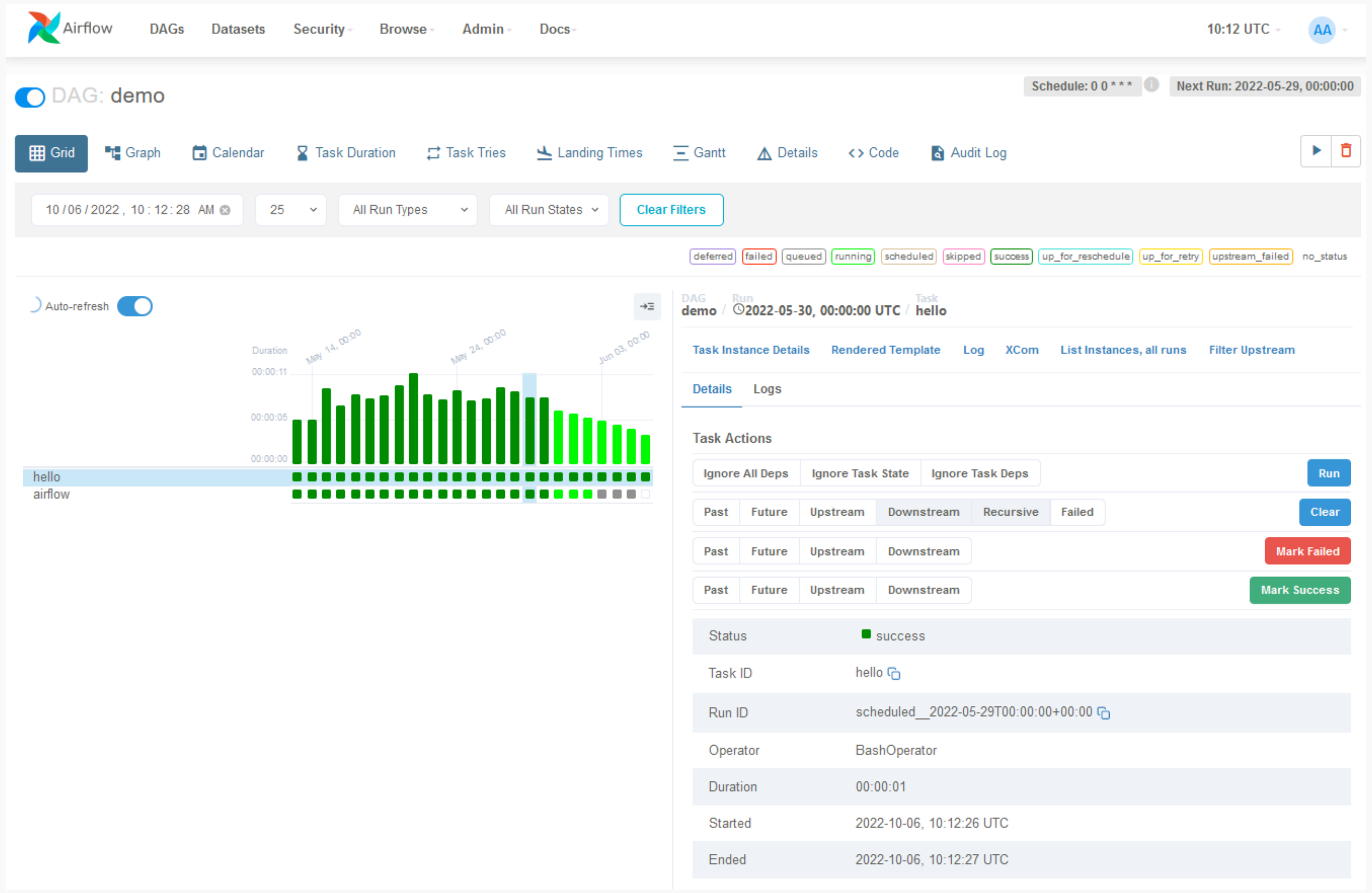
Apache Airflow

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows.

It allows you to define workflows as code, making it easier to maintain and reproduce complex data pipelines.

It is written in Python and uses Directed Acyclic Graphs (DAGs) to represent workflows.

Historical fact: Originally made by Airbnb to manage their workflows and was open source, becoming an Apache incubator project.



Status	🟩 success
Task ID	hello
Run ID	scheduled__2022-05-29T00:00:00+00:00
Operator	BashOperator
Duration	00:00:01
Started	2022-10-06, 10:12:26 UTC
Ended	2022-10-06, 10:12:27 UTC