

# Introduction to Data Engineering

with a hands-on  
demonstration of  
**Apache Airflow**

How to Orchestrate an ETL Data  
pipeline with Apache Airflow

---

# About me



**Sebastian Daberdaku**

Senior Data Engineer @  
CardoAI

B.S. and M.S. in Computer Engineering @ UNIPD  
Ph.D. in Information Engineering @ UNIPD

Background in research (bioinformatics, clinical/health informatics), Machine Learning and Data Science.

Expertise in building and managing data ingestion pipelines with technologies such as Spark, Airflow, Python, DataBricks, TrinoDB and the related cloud architecture with Terraform.

# About Cardo AI

**REVOLUTIONIZING**

**CAPITAL MANAGEMENT**

in the

**DIGITAL ECONOMY**



# Data Engineering

## Data, Data, Data!

- Data is growing at an unprecedented rate (volume):
  - Social Networks
  - Transaction Logs
- Fast streams of data (velocity):
  - Sensor data
  - Machine-to-machine data
- Different kinds of data (variety):
  - Text
  - Media
- How reliable is your data (veracity)?
- By 2025 we will be producing 463 exabytes of data per day (1EB=1e18B).

# More Data, more insights!

Data is abundant & diverse



As is how we store, process and analyze it

Streaming



Machine Learning



ETL



Modeling



BI



# Importance of Data Engineering in today's world

- 85% of big data projects fail (Gartner, 2017)
- 87% of data science projects never make it to production (VentureBeat, 2019)
- The underlying data architecture is generally not reliable enough to make business decisions.
- This is where data engineers come into the picture!

# What does a Data Engineer do?

Data engineers work in a variety of settings to build systems that collect, manage, and convert raw data into usable information for data scientists and business analysts to interpret.

They design and build systems for collecting, storing, and analyzing data at scale.

Their ultimate goal is to make data accessible so that organizations can use it to optimize their performance and make business decisions.

# Life in the Big Data Universe

## Data Scientist

- Building models
- Validation/testing
- Algorithms
- Continuous improvement

Knowledge of:

- Statistics
- Linear Algebra
- Machine Learning
- Deep Learning
- Python, R, Matlab

## Data Engineer

- Data pipelines
- Manage platforms
- Productionalize Algorithms
- Agile Development

Knowledge of:

- Platforms
- Algorithms
- Software Dev.
- Python, Java, Scala, Spark

## Data Analyst

- Deep domain knowledge
- Report Generation
- Data Exploration
- Hypotheses Testing
- Pattern Discovery
- Correlations



# What does a Data Engineer do?

## ETL and ELT Pipelines!

- **Extract** data from external systems;
- **Transform** the data to improve data quality and establish consistency;
- **Load** the data into a target database.

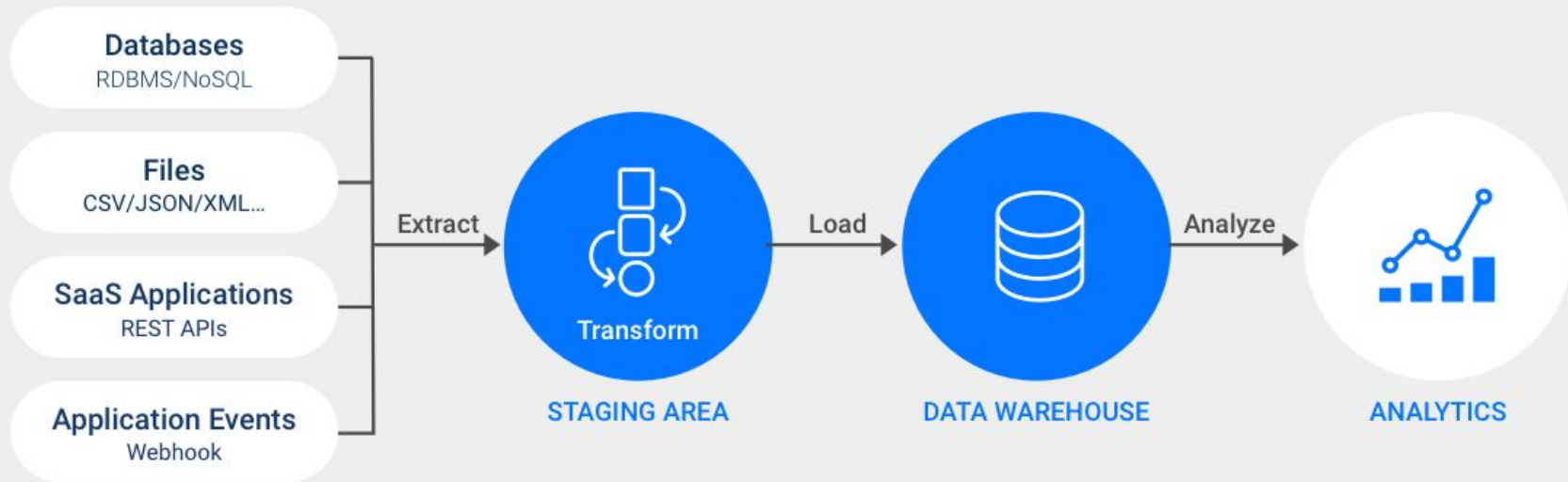
## **Extract, transform and load (ETL)**

Extract, transform, and load (ETL) is a data integration methodology that extracts raw data from sources, transforms the data on a secondary processing server, and then loads the data into a target database.

ETL is used when data must be transformed to conform to the data regime of a target database.

The extracted data only moves from the processing server to the data warehouse once it has been successfully transformed.

# ETL PROCESS



## **Extract, load and transform (ELT)**

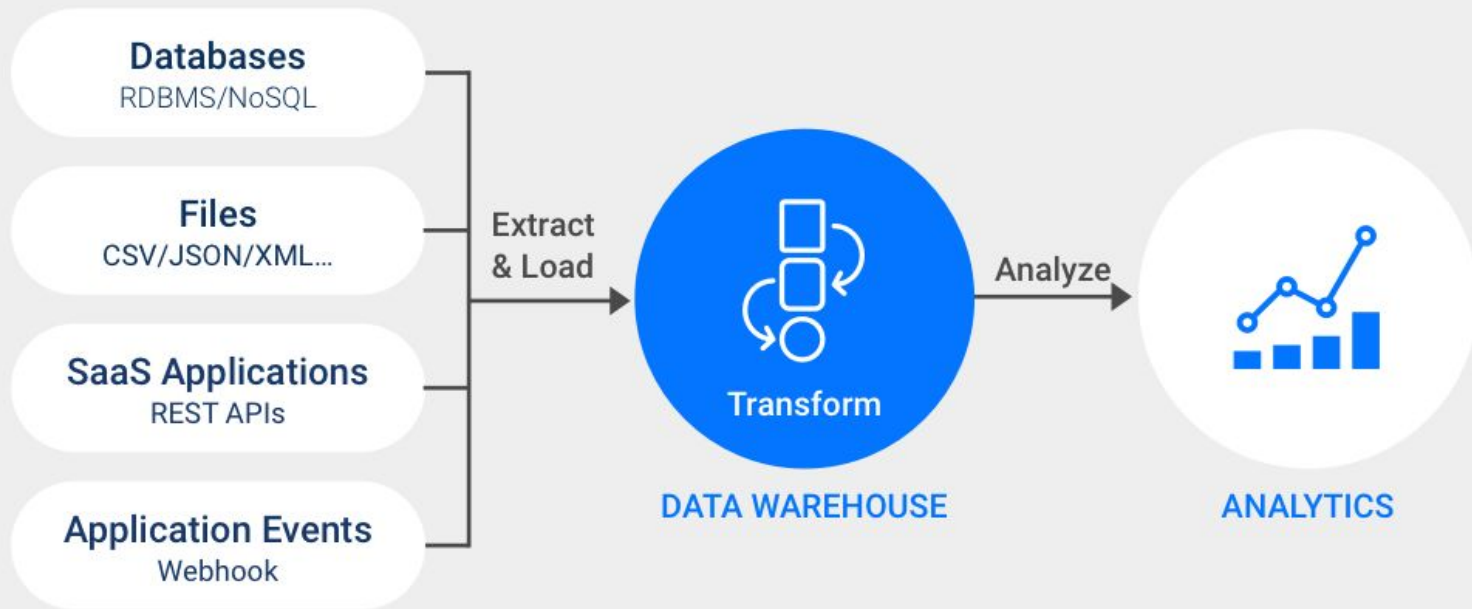
Unlike ETL, extract, load, and transform (ELT) does not require data transformations to take place before the loading process.

ELT loads raw data directly into a target data warehouse, instead of moving it to a processing server for transformation.

With ELT, data cleansing, enrichment, and data transformation all occur inside the data warehouse itself. Raw data is stored indefinitely in the data warehouse, allowing for multiple transformations.

ELT is a relatively new development, made possible by the invention of scalable cloud-based data warehouses.

# ELT PROCESS





Apache  
**Airflow**

## **APACHE AIRFLOW**

Apache Airflow is a platform  
for programmatically  
authoring, scheduling, and  
monitoring workflows.

# Introduction to



Apache  
**Airflow**

When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative.

Workflows are modelled as directed acyclic graphs (DAGs) of tasks.

Airflow is an orchestrator allowing you to execute your tasks at the right time, in the right way, in the right order.

# AIRFLOW PRINCIPLES

## **Scalable**

Airflow has a modular architecture and uses a message queue to orchestrate an arbitrary number of workers. Airflow is ready to scale to infinity.

## **Dynamic**

Airflow pipelines are defined in Python, allowing for dynamic pipeline generation. This allows for writing code that instantiates pipelines dynamically.

## **Extensible**

Easily define your own operators and extend libraries to fit the level of abstraction that suits your environment.

## **Elegant**

Airflow pipelines are lean and explicit. Parametrization is built into its core using the powerful Jinja templating engine.



# AIRFLOW FEATURES

## Pure Python

Use standard Python features to create workflows, including date time formats for scheduling and loops to dynamically generate tasks.

## Easy to Use

Anyone with Python knowledge can deploy a workflow. Apache Airflow does not limit the scope of the pipelines; can be used to build ML models, transfer data, manage infrastructure, and more.

## Robust Integrations

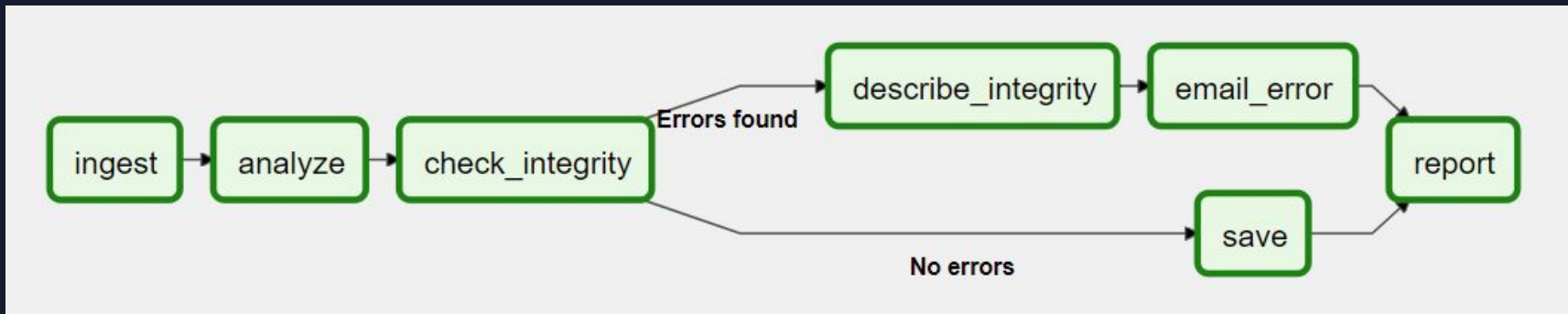
Airflow provides many plug-and-play operators that are ready to execute your tasks on Google Cloud Platform, Amazon Web Services, Microsoft Azure and many other third-party services.

## Useful UI

Monitor, schedule and manage workflows via a robust and modern web application. No need to learn old, cron-like interfaces. Always have full insight into the status and logs of completed and ongoing tasks.

# AIRFLOW OVERVIEW

**Airflow** is a platform that lets you build and run workflows. A workflow is represented as a **DAG** (a Directed Acyclic Graph), and contains individual pieces of work called **Tasks**, arranged with dependencies and data flows taken into account.



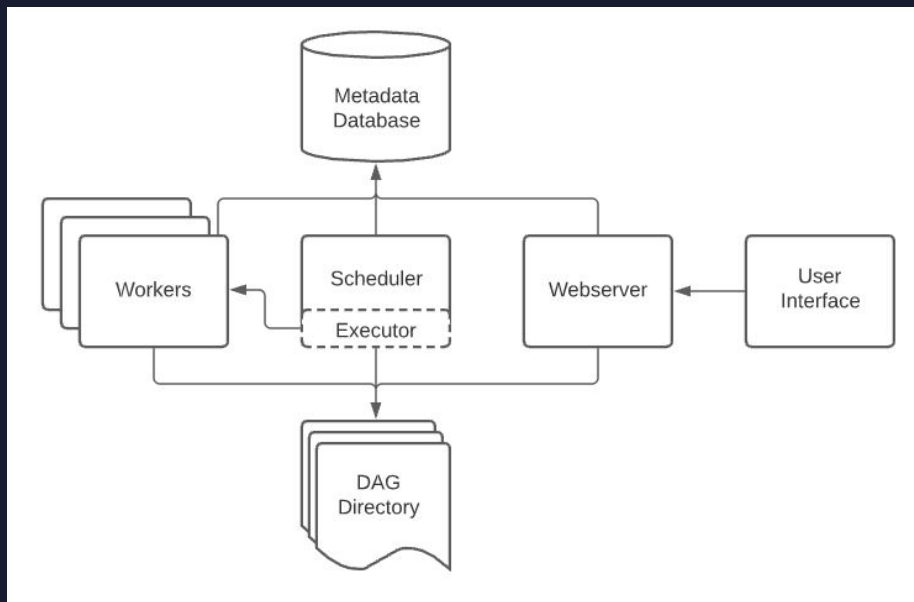
A DAG specifies the dependencies between Tasks, and the order in which to execute them and run retries; the Tasks themselves describe what to do, be it fetching data, running analysis, triggering other systems, or more.

# AIRFLOW TASKS

There are three basic kinds of Task:

- **Operator:** predefined task templates that can be stringed together quickly to build most parts of DAGs.
- **Sensors:** a special subclass of Operators which are entirely about waiting for an external event to happen.
- **TaskFlow-decorated tasks:** custom Python functions packaged up as Tasks.

# AIRFLOW ARCHITECTURE



**Scheduler:** daemon in charge of scheduling workflows;

**Executor:** Class defining how tasks should be executed;

**Worker(s):** Process/sub-process/POD executing tasks;

**Web Server:** Flask server with Gunicorn serving the UI;

**Metastore:** DB where the state is stored.

# HANDS-ON EXERCISE

In this guide, we will be writing an ETL data pipeline. It will extract cryptocurrency data from [Sentiment API](#), transform the data into a CSV file, and load the data into a Postgres database, which will serve as a data warehouse.

External users or applications will be able to connect to the database to build visualizations and make policy decisions.

## What you will learn:

1. How to set-up a development Apache Airflow deployment;
2. How to scrape data from SanAPI;
3. How to write a DAG script;
4. How to load data into a SQL database;
5. How to use Airflow Task-Flow API.

# EXERCISE GOAL

We want to build a data warehouse where we collect various metrics for different cryptocurrencies.

These metrics should be updated regularly with a fixed schedule.

SanAPI will provide unregistered users the following metrics for free:

- marketcap: the total market capitalization for the given coin;
- priceBtc: the coin to BTC exchange rate;
- priceUsd: the coin to USD exchange rate;
- volume: trading volume.

Users can request the metrics for a given coin in a given time-range and with a resolution of up to one second.

For the sake of this exercise, our objective will be to compute the average value of these metrics in the last hour, with a hourly schedule.

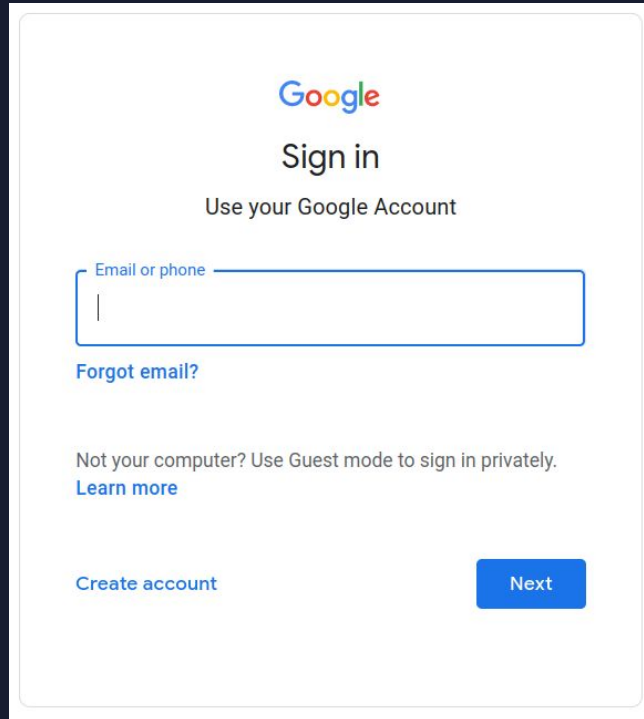
# Requirements

## What you need:

1. Basic knowledge of Python;
2. Basic knowledge of SQL;
3. A Google account if you want to use Google Cloud Shell, otherwise you can run the development environment on your local machine if you have Linux/Unix.

# Environment set-up

Log in to <https://shell.cloud.google.com/> with your Google account;

A screenshot of the Google sign-in page. At the top is the Google logo. Below it is the text "Sign in" and "Use your Google Account". There is a text input field with the placeholder "Email or phone" and a vertical cursor. Below the input field is a link "Forgot email?". Further down is the text "Not your computer? Use Guest mode to sign in privately." followed by a link "Learn more". At the bottom left is the text "Create account" and at the bottom right is a blue button labeled "Next".

Google

Sign in

Use your Google Account

Email or phone

[Forgot email?](#)

Not your computer? Use Guest mode to sign in privately.  
[Learn more](#)

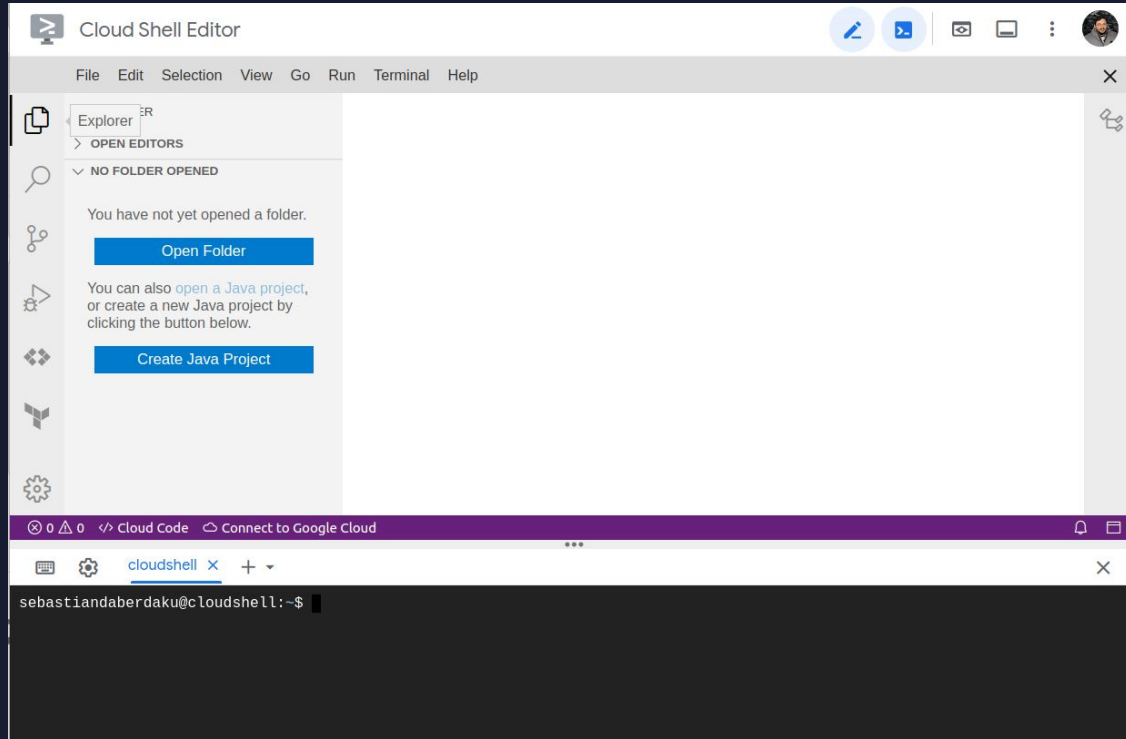
Create account

Next



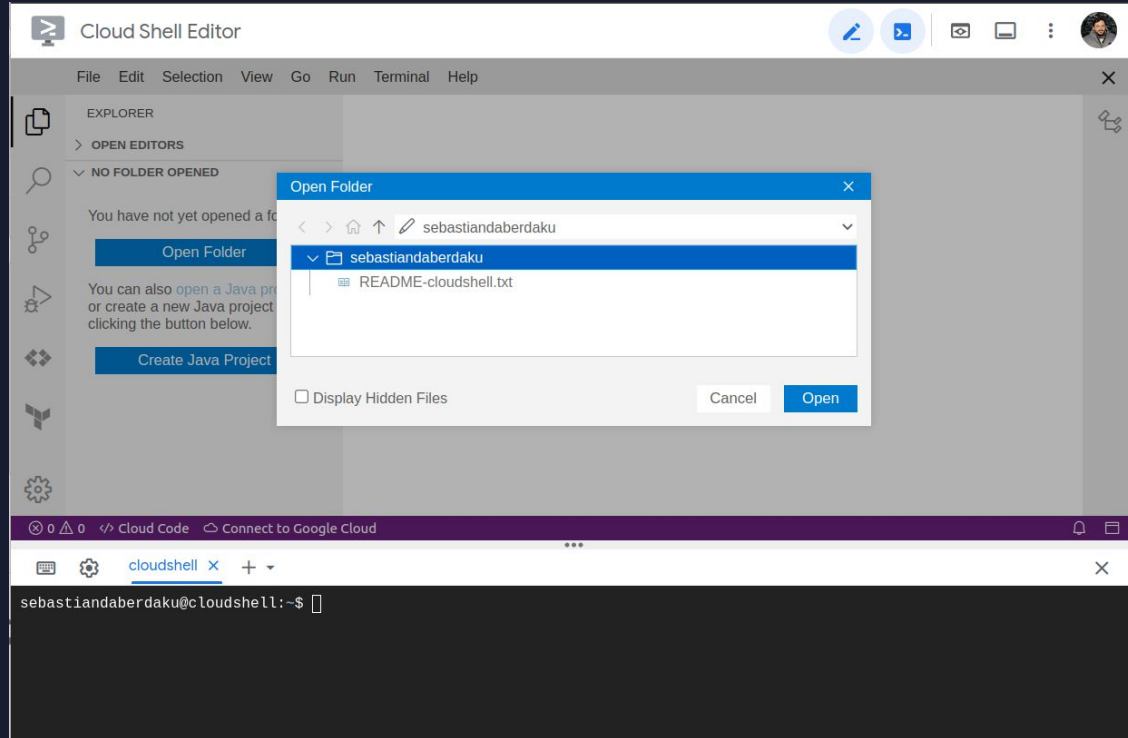
# Environment set-up

On the left panel, navigate to the “Explorer” tab.



# Environment set-up

Click on the “Open Folder” button, elect the working directory to use, and click “open”.

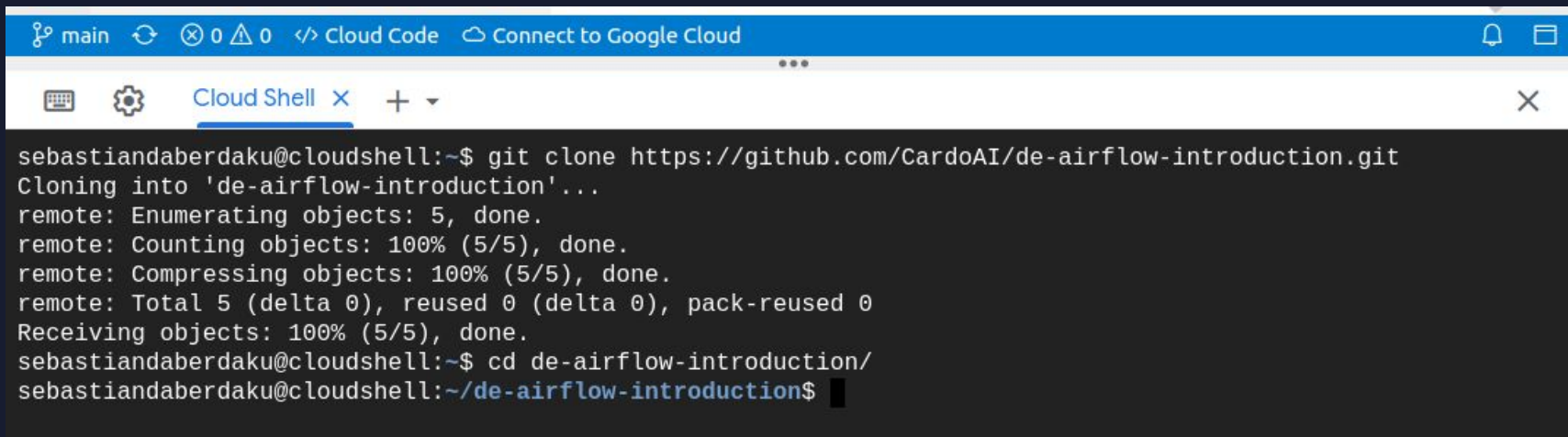


# Environment set-up

Clone the following GIT repository with the exercise code:

```
git clone https://github.com/CardoAI/de-airflow-introduction.git
```

Navigate inside the de-airflow-introduction folder to view the exercise-related files.



```
main 0 0 Cloud Code Connect to Google Cloud
Cloud Shell x +
sebastiandaberdaku@cloudshell:~$ git clone https://github.com/CardoAI/de-airflow-introduction.git
Cloning into 'de-airflow-introduction'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
sebastiandaberdaku@cloudshell:~$ cd de-airflow-introduction/
sebastiandaberdaku@cloudshell:~/de-airflow-introduction$
```

# Getting data from SanAPI

Our data source: Santiment API.

Santiment is an all-in-one market behavior and network intelligence platform for cryptocurrencies.

To retrieve data from SanAPI we need to connect to its API (we will use the [sanpy](#) Python library).

In order to access real-time data for some of the metrics, you'll need to set the API key, generated from an account with a paid API plan.

We will use the **pandas** Python library for data exploration and transformation, in order to compute the desired metrics.

# Running the pipeline

Airflow comes with a built-in SQLite3 database (not to be confused with the destination database).

We will use PostgreSQL as a destination database.

The `setup.sh` bash script will install all the required dependencies, including PostgreSQL, and then start the Airflow scheduler and web server.

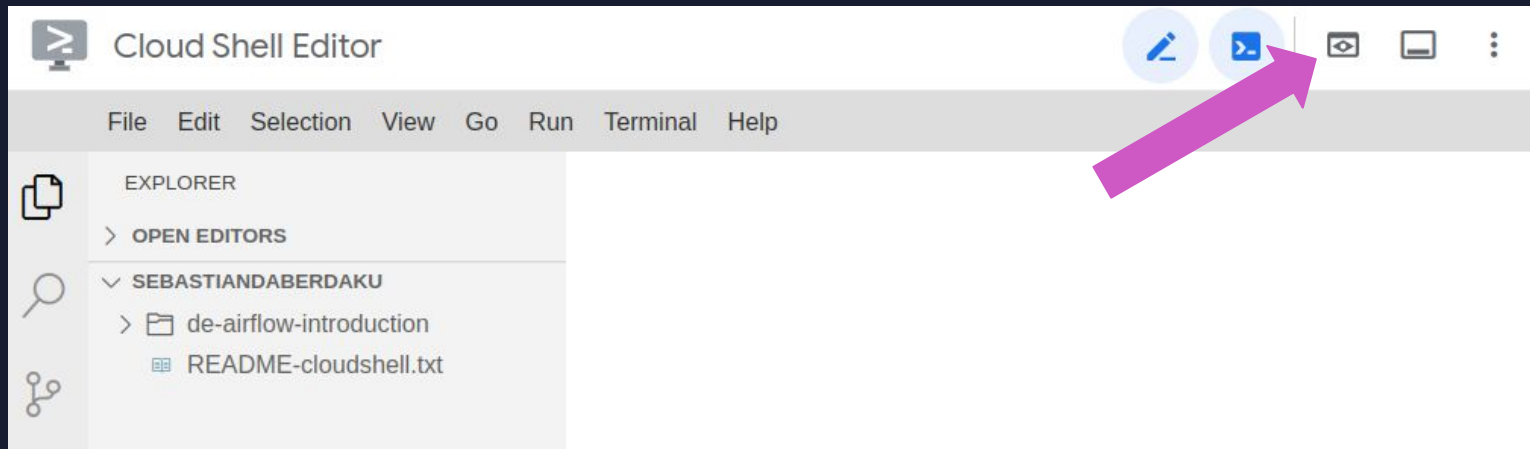
Once the PostgreSQL database and Airflow are up and running, you should see their output logs appended to the following files in the working directory:

- `postgresql.log`
- `airflow.log`

The admin airflow credentials will be visible in the **`airflow.log`** file.

# Running the pipeline

The Airflow Web UI can be viewed by clicking on the “Web Preview” button, and then selecting “Preview on port 8080”:



The detailed explanation of all UI functionalities can be found here:  
<https://airflow.apache.org/docs/apache-airflow/stable/ui.html>

**Live hands-on**



# Thank you!

---

## Contacts

**Sebastian Daberdaku**

Senior Data Engineer

[sebastian.daberdaku@cardoai.com](mailto:sebastian.daberdaku@cardoai.com)

Cardo AI



*Disclaimer: the logos represented here are referred to both clients and users of our technology*