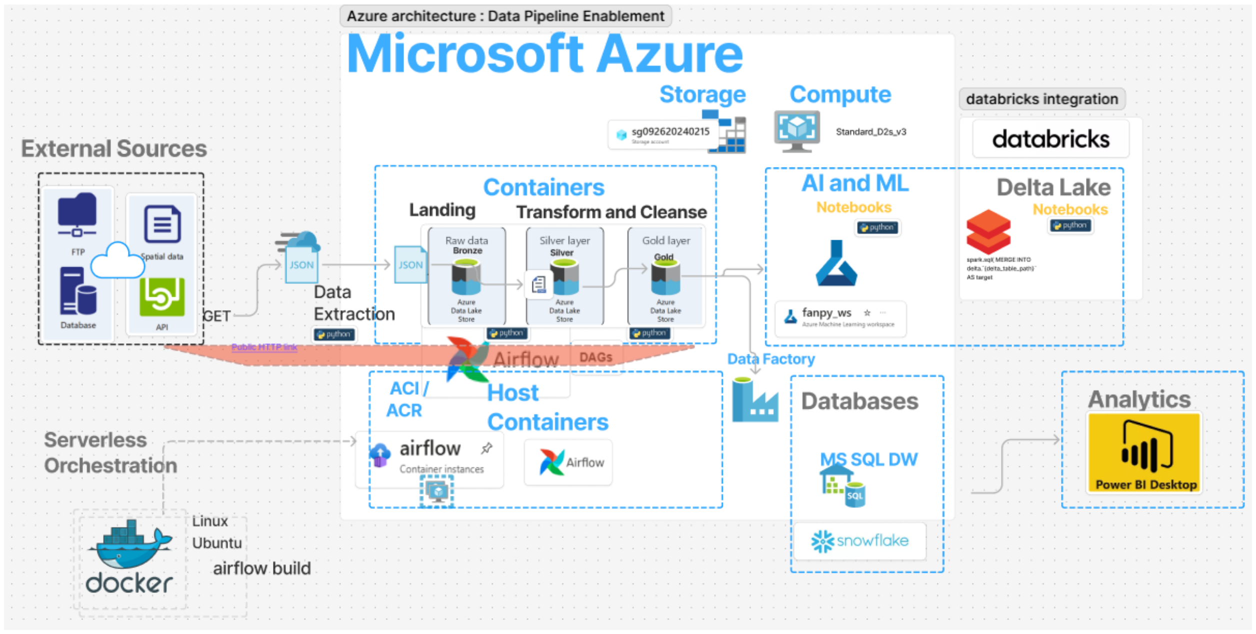


End-to-End Data Pipeline: From API to Analytics with Apache Airflow and Cloud-Native Technologies

Introduction

In order to flex the capabilities of Python, Apache Airflow, and Cloud-Native technologies in a data pipeline solution, I have implemented an environment that enables and supports the end-to-end data ingest to consumption. In most cases, a minimum amount of work is being done in order to prove the concept, prove the purpose, and work through the beginning issues that arise. I leaned heavily on freely available AI's to accelerate and troubleshoot numerous portions of the project. The point is data pipelines achieve value in being flexible, automated, and scalable.

The project showcases an end-to-end data pipeline, orchestrated using Apache Airflow, that seamlessly extracts JSON data from a REST API, processes it, and lands it in a cloud storage environment following the medallion (Bronze,Silver,Gold) architecture, applied to the physical Azure blob containers as the medallion resources. The choice of Azure storage was made primarily for convenience in this project.

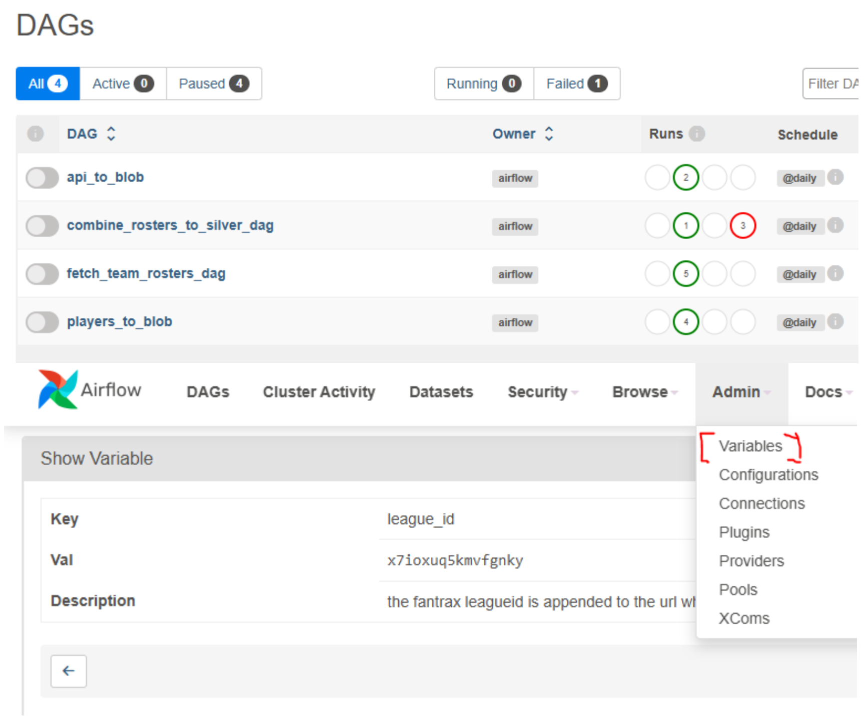
Moving through the container layer, this project features workflow touchpoints from Azure ML hosted python notebook. Experimentation with Azure Auto ML is planned in the near future. Numerous feature engineering python scripts have been authored here, and now be candidates for conversion into DAGs to further refine and shape data upstream in the process. (Closer to the source of the data)

Step 1: Data Extraction – API to JSON Processing

The data pipeline **begins with an HTTP GET request** API that returns structured JSON data. Apache Airflow is used to **schedule and orchestrate the data ingestion process**.

Why Apache Airflow?

Apache Airflow provides a **flexible, scalable, and dependency-aware** orchestration framework for automating workflows. It enables:
✓ **DAG-based scheduling** – Ensures task dependencies and execution order.
✓ **Modular pipeline design** – Each stage (API call, storage, processing) runs independently.
✓ **Retry logic & monitoring** – Reduces failures with auto-retries and logging.


DAGs

All 4 | Active 0 | Paused 4 | Running 0 | Failed 1 | Filter DA

DAG	Owner	Runs	Schedule
api_to_blob	airflow	2	@daily
combine_rosters_to_silver_dag	airflow	1 3	@daily
fetch_team_rosters_dag	airflow	5	@daily
players_to_blob	airflow	4	@daily

Show Variable

Key	league_id
Val	x7ioxuq5kmvfgnky
Description	the fantrax leagueid is appended to the url wh

Apache Airflow DAG Design

A Directed Acyclic Graph (**DAG**) was created in **Python**, implementing:
✅ **Task dependencies** – Ensuring API extraction completes before storage.
✅ **Variable management** – Allowing dynamic configuration (e.g., API endpoint, cloud storage path).
✅ **Logging & monitoring** – Capturing execution status and debugging insights.

Step 2: Storage – Landing Data in the Cloud (Bronze Layer)

Once extracted, the JSON data is **stored in a cloud-based object storage solution** (e.g., **Azure Blob Storage**), following a **Bronze-Silver-Gold data lake pattern**:

- **Bronze Layer (Raw Data Storage)** → Stores raw JSON for lineage and auditing.
- **Silver Layer (Cleaned & Transformed Data)** → Mostly CSVs today. But Standardized and structured for querying.
- **Gold Layer (Aggregated & Enriched Data)** → Optimized for analytics and reporting.

Storage Task in Airflow

A second task in the Airflow DAG pushes the extracted JSON data into cloud storage:

extracted data is persistently stored in the **Bronze layer** for further processing.

Step 3: Processing & Feature Engineering (Silver Layer)

After raw data is stored, an **Azure AI/ML module** processes it:
✓ **Merging multiple JSON files** → Flattens and creates structured CSVs and tables.
✓ **Feature Engineering** → Custom groupings, missing value handling.
✓ **Normalization & Statistical Computations**:

- **Min-Max Scaling** (0 to 1)
- **Z-Score Standardization**

Data is then **loaded into Databricks Community Edition** for:
✅ **Spark-based transformations** (fast, distributed processing).
✅ **Delta Lake implementation** (incremental updates).

Step 4: Analytics & Visualization (Gold Layer)

Once processed, data is made available for **BI and reporting** via **Power BI**, integrating directly with **Azure Blob Storage**.

✓ **Joining CSV files from Blob Storage**.
✓ **Data deduplication & transformation**.
✓ **Modeling for insights**.

Containerization – Why Docker Matters for Airflow

Why Use Docker for Apache Airflow?

Deploying Airflow inside a **Docker container** ensures **consistent execution environments**, eliminating dependency mismatches between development and production. While this project does not include any corporate security components, many use cases do have corporate security needs, and docker is a great way to meet those needs with consistently reliable server builds.

✓ **Reproducibility** – Every deployment runs the same environment.
✓ **Lightweight & Scalable** – Containers can be scaled dynamically.
✓ **Dependency Management** – Includes all Python libraries (e.g., `requests`, `azure-storage-blob`).

Custom Docker Image for Airflow

A **custom Docker image (`airflow-blob:latest`)** was built and deployed to a container registry (e.g., **Azure Container Registry, Docker Hub**), embedding:

- **Airflow DAGs & dependencies**
- **Azure SDK & authentication configuration**
- **Optimized startup scripts**

Key Achievements & Business Impact

🚀 **Fully automated API-to-Analytics pipeline** – No manual intervention needed.
🚀 **Scalable & cost-efficient deployment** – Serverless, containerized architecture reduces costs.
🚀 **Optimized for cloud storage & processing** – Supports **multi-cloud environments**.
🚀 **Robust analytics integration** – Enables **real-time business insights**.

This **modular, containerized pipeline** serves as a **blueprint for scalable data engineering workflows**, adaptable across industries.

Conclusion

This project highlights a **modern, cloud-native data pipeline** leveraging:
✓ **Apache Airflow** for workflow orchestration.
✓ **Docker** for reproducibility.
✓ **Cloud storage** for structured data management.
✓ **ML & Spark processing** for advanced analytics.
✓ **Power BI** for visualization.

This framework **can be expanded** to support **real-time streaming, CI/CD automation, and large-scale AI applications**, making it a **versatile, future-proof solution** for data-driven enterprises. 🚀