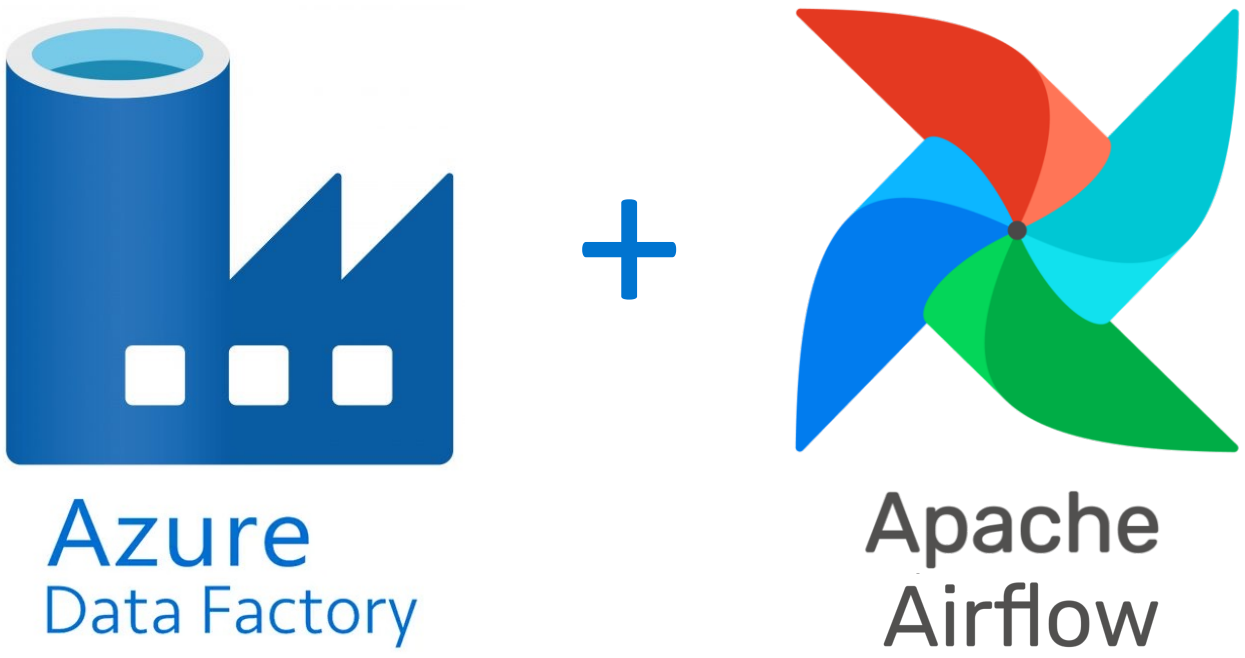
Better ETL with Managed Airflow in ADF



Niall Langley

Data Developer / Consultant

Blog | niall-Langley.me

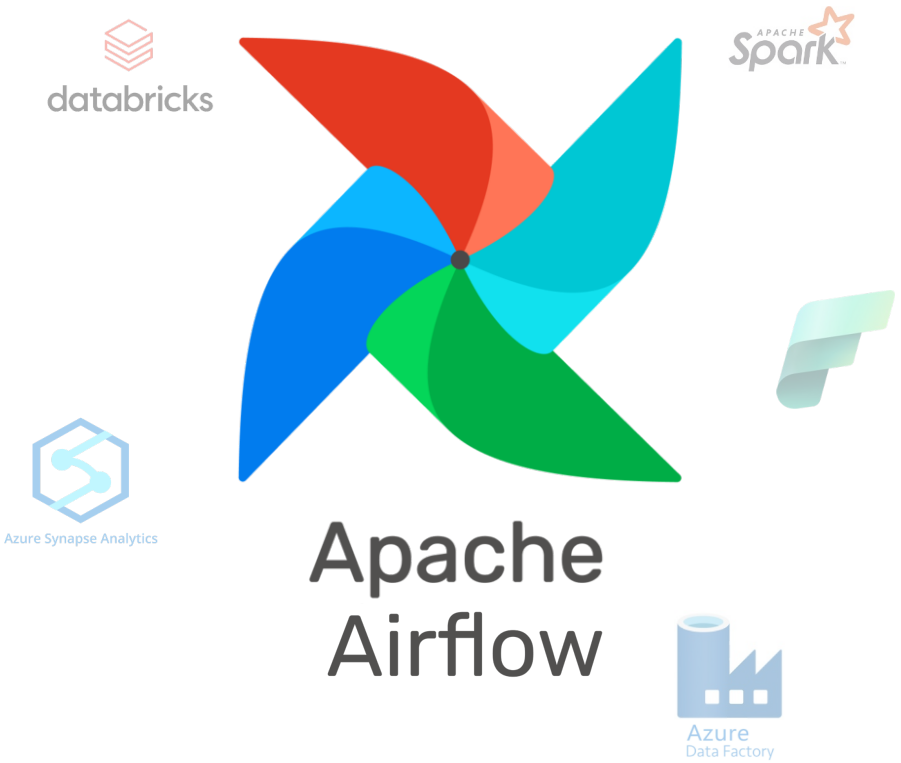
| uk.linkedin.com/in/niall-langley

| @NiallLangley

| https://github.com/NJLangley

Agenda

- What is Airflow
- Why do we need it
- How does it work
- What does it look like
- Conclusions



Apache Airflow™

Airflow™ is a platform created by the community to programmatically author, schedule and monitor workflows.

What & Why

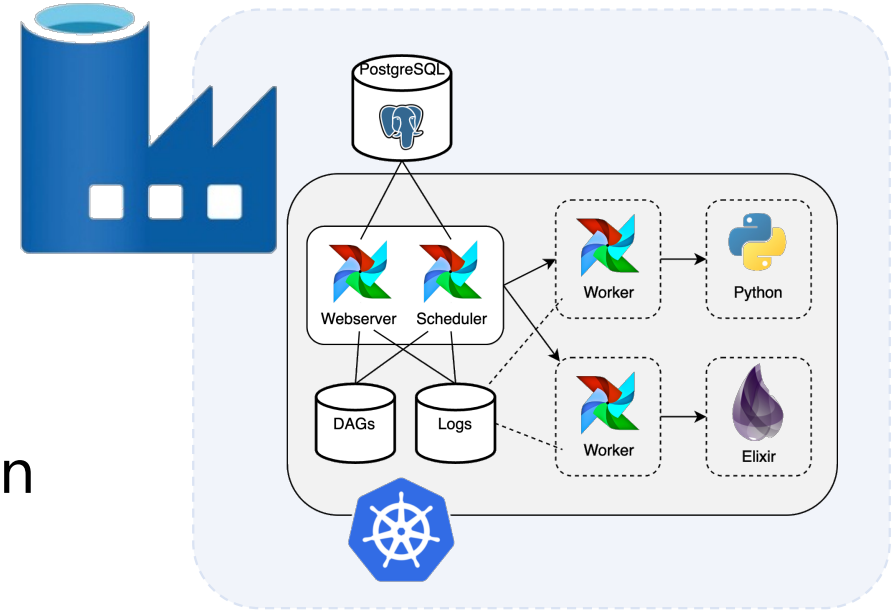
What is Airflow?

- It’s a code first workflow tool that uses Python
- Data Pipelines (Workflows) are built using DAG’s
- Has a powerful scheduling engine
- Use integrations to integrate other services
- Scalable & Extensible
- Great monitoring UI
- Open Source & cross cloud
- Can be run locally
- **It’s a conductor for your ETL**

What's the ADF Managed Bit?

- Previously running Airflow on Azure meant going DIY
- Kubernetes (K8s) Service
  - Scheduler
  - Workers
  - Webserver for UI
  - DAG Database

- ADF Managed Airflow is PaaS
- Supports Azure AAD Authentication

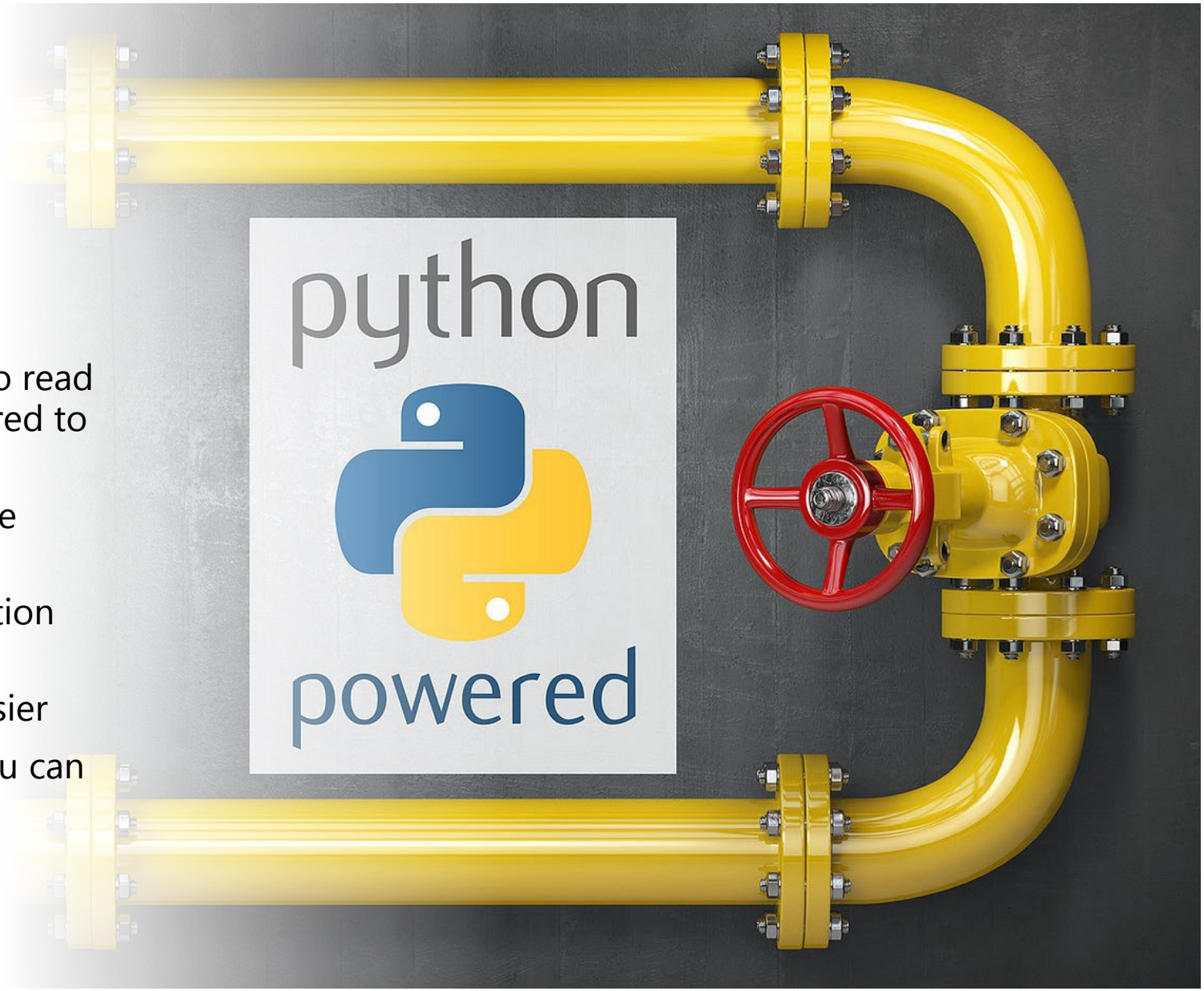


How’s it different to Azure Data Factory?!

- ADF is two tools in one
  - Data Integration
  - Process Orchestration & Scheduling
- Airflow is not a data Integration tool
- ADF is good for simple orchestration
  - Limited loops and conditional branching
  - Limited dynamic execution
  - Error handling and logging awkward
  - No task grouping
  - Not really extensible

ETL as Code

- Python is much easier to read and understand compared to generated JSON
- Coding standards can be enforced
- Encourages modularization and reuse
- Can unit test Python easier
- You can do anything you can do with Python



ADF & Airflow

Use Airflow for Orchestration and Scheduling

Airflow can run bash scripts

Use simple ADF pipelines with Copy Activities to move data

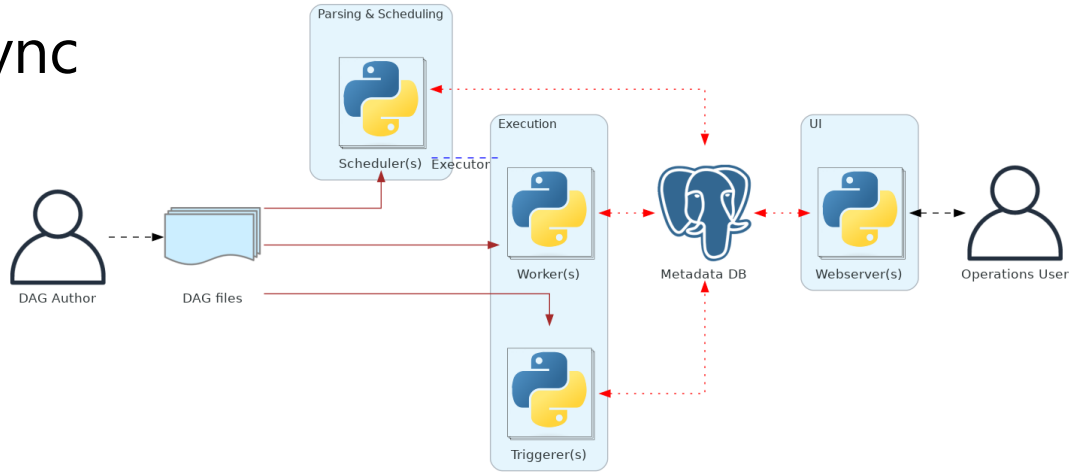
Airflow provides a better monitoring UI

ADF can provide Integration Runtimes to work with Secure Networking

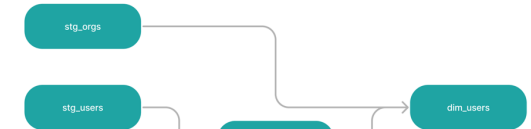
How

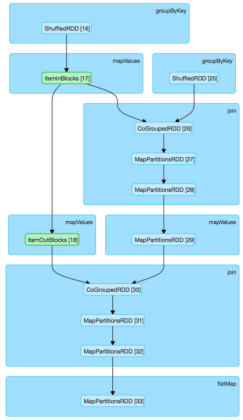
Airflow Core Components

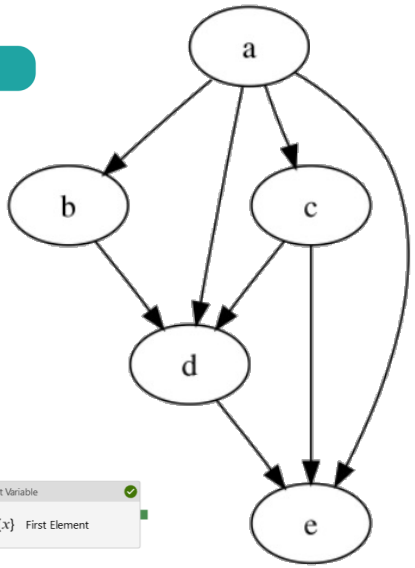
- A Scheduler triggers workflows and submits work to executors
- One or more Executors to do work
- A triggerer which executes async tasks
- A webserver for the UI
- A folder of DAG files
- A Metadata DB to track state
- Connections to other services

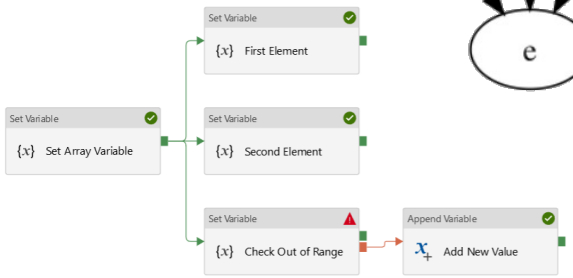


What is a DAG





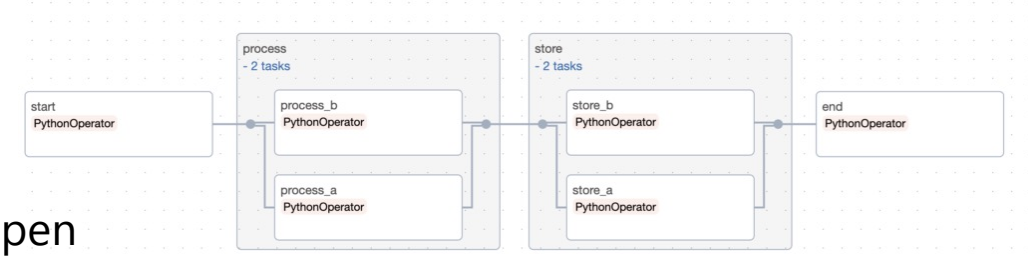




- Directed Acyclic Graph
  - Nodes and Edges
  - Edges have direction
  - No loops
- Can represent a series of steps in a process
- Based on Graph Theory

Airflow DAG Components

- DAG's are composed of **Tasks** and **Relationships**
- Tasks can by sync or async
- Three types of task
  - Operators – predefined task templates
  - Sensors – wait for an external event to happen
  - TaskFlow decorator `@task` on a Python function
- Tasks reference **Connections** to services
- Relationships define the dependencies between Tasks
- Tasks can be grouped using **Task Groups** or **Sub-DAG's**
- Tasks can be deferred, meaning they wait to be triggered



Airflow DAG's are Awesome

- DAG's are defined in Python
- A task can produce a Dataset
- When executed, a Workflow instance executes Task instances
- A multiple instances of a workflow can execute in parallel
- Executors have slots. Each slot can run a task
- If a task is deferred, the slot is released

```python
from airflow.operators.bash import BashOperator
from airflow.operators.python import Python

...

task_a = PythonOperator(
  task_id='task_a',
  python_callable=my_python_function
)

task_b = BashOperator(
  task_id='task_b',
  bash_command='echo "hello"'
)

task_a >> task_b
# Or
task_b << task_a
```

Sounds Good, But What Does it Cost?

- Can choose small or large clusters
- Approx £0.5 - £1 per hour
- But you can’t turn it off

- ADF is charged for CPU time and per activity
- Worked with clients where ADF costs as much as Databricks

Dynamic Metadata Driven ETL

- Metadata driven ETL can be powerful
- Especially for populating initial Lakehouse Layers
  - Landing -> Bronze -> Silver
  - Landing -> Raw -> Enriched
- *Dynamic DAG Generation* uses metadata to build a DAG
- *Dynamic Task Mapping* uses runtime data to dynamically generate tasks
- DAG *tasks* can define *datasets*, which are can trigger other *workflows*

Demos

Conclusions - The Good

- One language to rule them all
  - Python is consistent for data engineers
  - Automated testing can be easier
  - Linting for code consistency
- Can build complex pipelines easily
- Code reuse is better
- Monitoring is really, really good
- Multi-Cloud Native
- Less clicking around the UI

Conclusions - The Bad

- Can’t turn it off!
- There are some security issues – tokens in Git!
- Getting connections to work with Key Vault is tricky
- Default retry timeouts and counts are slow
- Deploying the Integration Runtime is Awkward
- No CLI / PowerShell support – REST API only

- It’s in public preview – so these thing should get fixed

Some Links that Helped Me

Example DAG's

https://github.com/apache/airflow/tree/main/airflow/example_dags

Astronomer DAG Development

https://docs.astronomer.io/learn/category/dags

Airflow Task Groups

https://marclamberti.com/blog/airflow-taskgroup-all-you-need-to-know/#:~:text=Airflow%20taskgroups%20are%20meant%20to,parameters%20correctly%2C%20and%20so%20on.

Databricks notebook return values

https://www.inovex.de/de/blog/fully-managing-databricks-from-airflow-using-custom-operators/