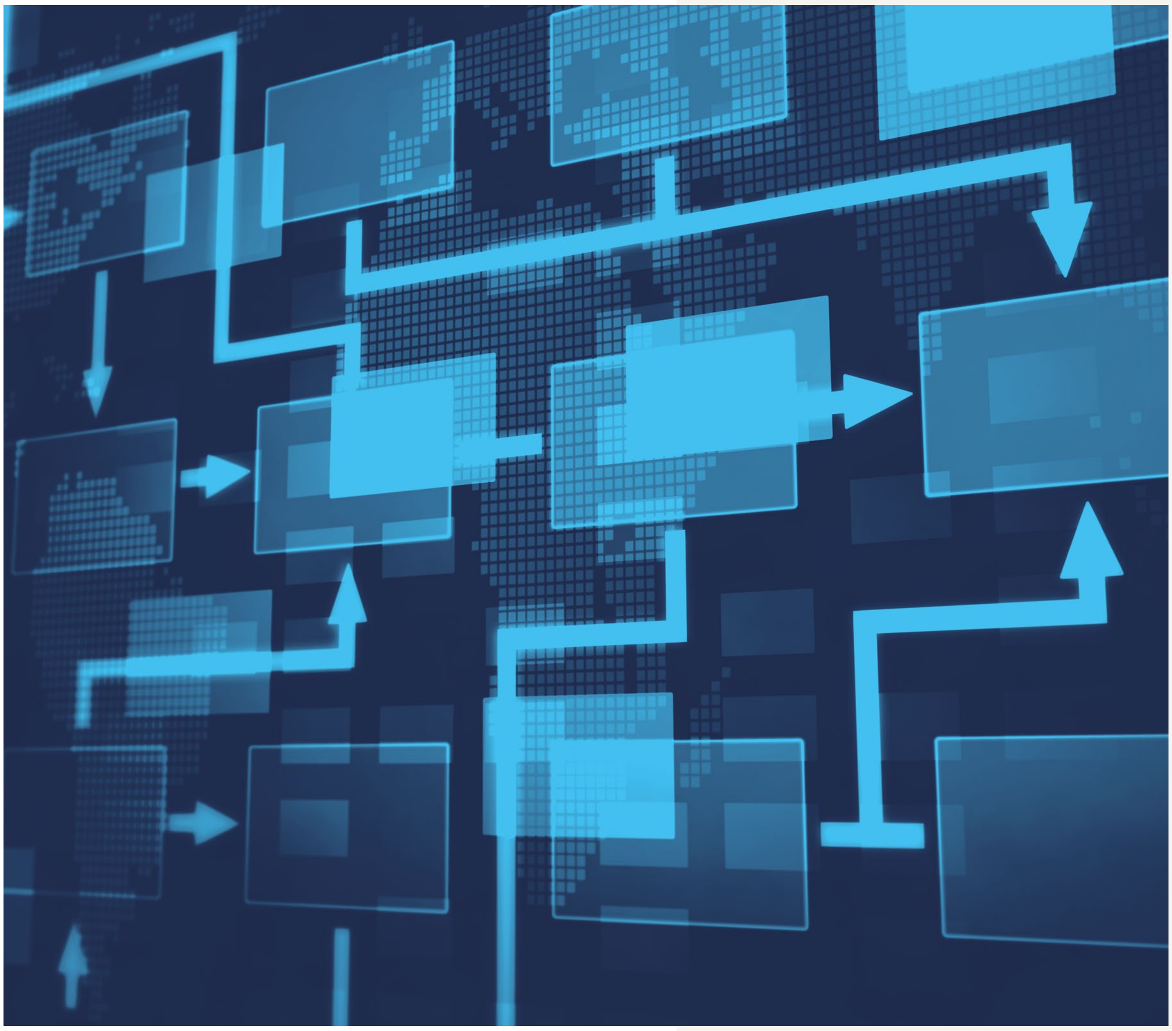
Orchestrating your Databricks workloads

PYTHON PREDICTIONS
A TOBANIA COMPANY

Authored by **Michaël Van Hoyweghen** | Data Foundations Lead



Introduction

The Databricks Lakehouse is a powerful platform that allows organizations to store, process, and analyze massive amounts of data. Its key strength is its ability to handle complex data transformations within analytic or operational workloads. This means it can handle transformations for the models in your data warehouse or lakehouse, as well as for the data processing for your web app.

Data transformations often consist of multiple stages where one table serves as input for another, or with subsequent transformation stages writing to layers or zones in your data lake: e.g. bronze, silver and gold. These stages and dependencies are defined through orchestration.

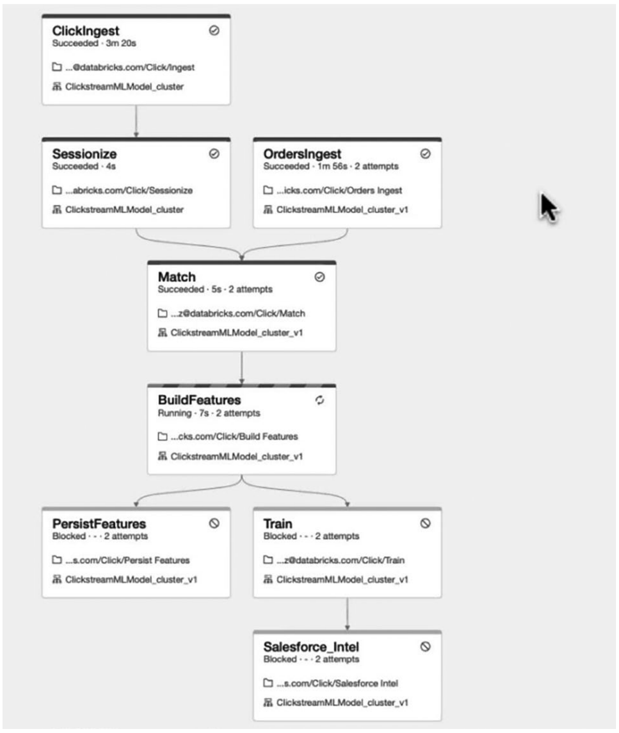
In this article we'll have a look at different ways of orchestrating your Databricks workloads. More specifically, we will discuss the benefits and drawbacks of the following tools:

- Databricks Jobs
- Azure Data Factory
- Airflow
- dbt
- Delta Live Tables

We hope this article will inspire fellow data engineers and architects during their next Databricks projects!

Databricks Jobs

Databricks Jobs is the native way of orchestrating tasks such as notebooks, jars, python applications, dbt jobs, or delta live tables. Connecting tasks in order of execution creates a workflow in the form of a Directed Acyclic Graph (DAG), as shown in the figure below. Databricks also allows you to schedule your job to run at a later time.


ClickIngest
Succeeded · 3m 20s
...@databricks.com/Click/Ingest
ClickstreamMLModel_cluster
Sessionize
Succeeded · 4s
...abricks.com/Click/Sessionize
ClickstreamMLModel_cluster
OrdersIngest
Succeeded · 1m 56s · 2 attempts
...icks.com/Click/Orders Ingest
ClickstreamMLModel_cluster_v1
Match
Succeeded · 5s · 2 attempts
...z@databricks.com/Click/Match
ClickstreamMLModel_cluster_v1
BuildFeatures
Running · 7s · 2 attempts
...cks.com/Click/Build Features
ClickstreamMLModel_cluster_v1
PersistFeatures
Blocked · - · 2 attempts
...s.com/Click/Persist Features
ClickstreamMLModel_cluster_v1
Train
Blocked · - · 2 attempts
...z@databricks.com/Click/Train
ClickstreamMLModel_cluster_v1
Salesforce_Intel
Blocked · - · 2 attempts
...s.com/Click/Salesforce Intel
ClickstreamMLModel_cluster_v1

In DAGs like the one shown on the left, the nodes represent the tasks (transformations) and the edges represent the dependencies between the tasks. Usually, the edges correspond with tables or files in which intermediate results are stored.

It is generally recommended to use job clusters when running non-interactive workloads, because they are cheaper than interactive clusters (same as all-purpose compute). A job cluster can be shared among the tasks of a Databricks job. This means you only need to wait for one cluster to start, reducing run time and cost. However, job clusters can't be shared across jobs. So when you have many parallel jobs with a variable workload, it may be beneficial to use one auto-scaling interactive cluster to avoid the overhead associated with running many job clusters in parallel – such as many driver nodes and the cost of high total cluster startup time.

A great feature of Databricks Jobs is that you can link a task directly to a notebook in a remote repository, instead of using notebooks from Repos or your Workspace. This way you can prevent people from unintentionally making changes to your notebooks or changing branches in the Databricks Repo that your job uses. This also means that you do not need a separate deployment step in your CI/CD flow.

Pros

- Full control over your Databricks job (cluster parameters, monitoring, ...)
- You can use a single job cluster for all tasks
- Allows you to run delta live tables or dbt jobs as a task
- Context sharing between tasks using Task Values
- Direct link to your repository
- Reduces TCO by keeping transformation and its orchestration in one tool

Cons

- You can only orchestrate Databricks workloads. Dedicated orchestration tools can be used to integrate your Databricks job into a bigger workflow.
- No straightforward way to store the workflow in a repository. You have to manually copy the json representation or use the jobs API of Databricks to automate this. This API can also be used for automated deployment.
- Limited flow control, e.g. no way to include/exclude tasks based on conditions

More info: https://docs.Databricks.com/workflows/jobs/jobs.html

Data Factory

Data Factory is the go-to orchestration and ingestion tool on Azure. It allows you to create, schedule, and manage data pipelines that can move and transform data from various sources to different destinations. Here we will focus on the capabilities of Data Factory to orchestrate Databricks workloads.

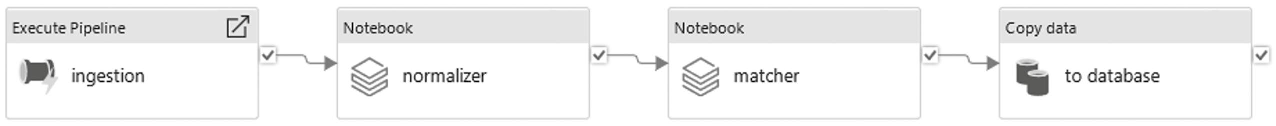
Data Factory is a dedicated orchestration and ingestion tool which allows for the creation of workflows that consist of much more than just Databricks tasks. You can create an end-to-end flow where data is extracted from a source and loaded into its destination. While in Databricks you can also connect to several sources and destinations (e.g. using Spark connectors or Python libraries), they are not visible in your DAG.

Data Factory allows you to run notebooks, jars, or python scripts on Databricks. These activities can be chained together to create a workflow, as shown in the image below.

Data Factory provides more control in your flow than Databricks. You can use metadata in your flows, for example allowing the execution of one activity to depend on the status or output value of another. In Databricks jobs you are limited to Task Values.

There are, however, some limitations regarding the orchestration of Databricks workloads. For example, each activity (notebook, jar or python script) triggers one separate job. As a result, you cannot share a job cluster among them and a new cluster has to be started for each notebook, costing time and money. A solution is to use a shared interactive cluster, but these are more expensive. (As mentioned in the section on Databricks jobs, there are use cases with multiple parallel jobs where an interactive cluster might be cheaper). Another option would be to trigger a multi-task job which can be defined in Databricks. Unfortunately, there is no ADF activity for this purpose: you need to use a web activity, connecting to the Databricks Jobs API.

Data Factory is also limited in its use of Databricks' built-in features. It does not provide single-node job clusters, direct links to your notebook repository, or the ability to share clusters.



Pros

- Advanced orchestration, allowing you to use metadata and conditional flows.
- Integrates Databricks tasks in a bigger workflow.
- Data integration next to orchestration.

Cons

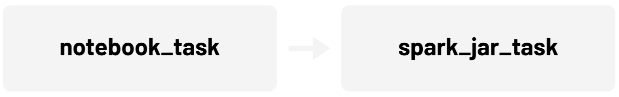
- Each notebook or jar triggers one separate job.
- There is no Databricks activity to run a Databricks job. You need to use a web activity.
- Doesn't provide all the options for Databricks, e.g. single-node clusters.
- Increases overall costs if introduced for the sole purpose of orchestration.

Airflow

Apache Airflow is an open-source platform for programmatically creating, scheduling, and monitoring workflows. Just like the tools discussed above, it allows users to define, execute, and manage workflows as directed acyclic graphs (DAGs) of tasks. Airflow provides a rich user interface for managing and monitoring workflows, and supports integration with various data sources and processing frameworks.

While Data Factory is a proprietary tool that is only available on Azure and focused on integration with other Azure tools, Airflow is open-source and can be used with a wider range of tools. One downside is that Airflow is not a managed tool: you need to provide the infrastructure on which it runs. Managed versions of Airflow do exist, such as Amazon MWAA, Data Factory Managed Airflow, and Cloud Composer.

Another significant difference with ADF is that in Airflow, you need to define your flows in a programmatic way in Python rather than through a UI. This might be more challenging for some, but it provides more flexibility and control. The flow can still be visualized as a DAG, as shown in the image below.



Source: https://www.databricks.com/blog/2017/07/19/integrating-apache-airflow-with-databricks.html

Airflow is more performant than Data Factory when it comes to the orchestration of Databricks workloads. It provides the following two operators (which use the Databricks Jobs API in the background):

- *DatabricksRunNowOperator:* Run an existing Databricks job, consisting of one or multiple tasks.
- *DatabricksSubmitRunOperator:* Create and run a new job in Databricks, consisting of a notebook, jar, python script or delta live tables pipeline (see below).

These operators are simple wrappers around the Databricks API call that allow you to submit the json payload that the API expects, and to make use of the full functionality of the corresponding endpoints of the Databricks API. Moreover, the first operator offers a functionality which is not available out-of-the box in Data Factory, where the use of a web activity is required.

Pros

- Advanced orchestration: flexibility and fine-grained control.
- Integrates Databricks tasks in a bigger workflow.
- You can run existing Databricks jobs, not just notebooks or jars.
- Full control over the Databricks job parameters.

Cons

- Installation and management overhead, for the non-managed option
- Increases complexity (programmatic interface)

Dbt

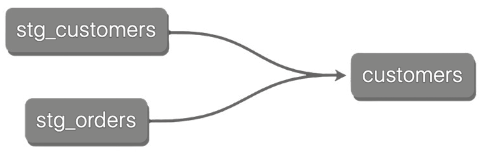
Dbt, or Data Build Tool, is a popular data transformation tool that allows you to write your transformations as SQL select statements. Additionaly, it lets you orchestrate your data flows.

Dbt is mainly used in ELT workflows for data warehouses such as Snowflake, BigQuery, or Databricks where the source data has already been loaded into the warehouse. Dbt takes care of turning that source data into data that is ready for BI consumption. This is the big difference with Data Factory or Airflow: with dbt we write the code for our transformations inside the dbt environment (we're not using Databricks notebooks, jars or python scripts). Keep in mind though, while you write your SQL statements or python commands in dbt, it isn't dbt who executes them. Instead, dbt provides the compiled SQL files to the target data warehouse that will execute them. The big advantage here is that your solution is more easily portable — if you need to migrate to a different data warehouse, the transformation tool (dbt) and the majority of your SQL code can be kept (although there may be differences between the SQL flavors of the two data warehouses).

Dbt comes in two versions: managed (dbt Cloud) and open-source (dbt Core). dbt Core allows you to create data flows and define dependencies, while the managed tool adds several features such as the ability to schedule those flows.

New to dbt? Read more about it here.

Instead of defining your data pipelines using a set of tasks, dbt manages how your data is transformed using a set of tables or models. Using the source and target tables defined in your queries, dbt generates the DAG automatically. In your DAG, the nodes represent tables and the edges represent the dependencies between them. Those dependencies basically correspond with the transformations that turn one table into the other. Note the difference with the DAGs of the tools described above.



Source: https://docs.getdbt.com/docs/build/sql-models

An advantage of this type of DAG is that it provides data lineage: a clear view of how the data flows through the system from table to table. This enables transparency and easier troubleshooting.

Dbt also offers an easy and structured way to integrate automated data quality testing and documentation into your flow.

After running your dbt job, dbt will have all the tables created that are part of your DAG and have them registered in the hive metastore or unity catalog.

With dbt you can either connect to regular Databricks clusters (from the Databricks Data Science & Engineering environment) or to Databricks SQL warehouses (from the Databricks SQL environment). The second option might make the most sense for dbt, since Databricks SQL is the data warehouse part of the Databricks Lakehouse. You do need to create the clusters or warehouses beforehand within your Databricks workspace, so you are not able to make use of cheaper job clusters.

Another drawback in using dbt to define your Databricks transformations is that it does not support streaming transformations, so limiting you to batch pipelines.

Pros

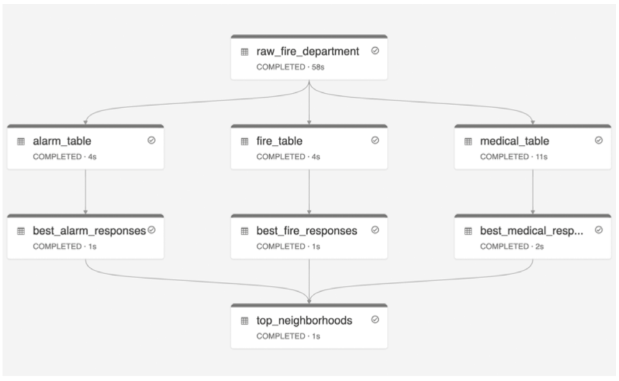
- Provides data lineage
- Automated data quality testing and documentation
- Portability of your transformation code

Cons

- No access to all features of Databricks Spark code (e.g. Structured Streaming)
- Doesn't allow for job clusters
- No scheduling available in dbt Core

Delta Live Tables

Delta Live Tables (or dlt) is another framework within the Databricks offering that allows you to define your data flows. Flows are created based on tables and their dependencies, as in dbt.



raw_fire_department COMPLETED · 58s
alarm_table COMPLETED · 4s
fire_table COMPLETED · 4s
medical_table COMPLETED · 11s
best_alarm_responses COMPLETED · 1s
best_fire_responses COMPLETED · 1s
best_medical_resp... COMPLETED · 2s
top_neighborhoods COMPLETED · 1s

Source: https://learn.microsoft.com/en-us/azure/databricks/workflows/delta-live-tables/delta-live-tables-cookbook

Delta Live Tables can be seen as Databricks' answer to dbt. Next to the table based DAGs, it also offers the automated data quality testing and the choice between SQL and Python to define your flows.

One major difference with dbt is that it supports streaming pipelines using Spark Structured Streaming. This is probably the most important use case of Delta Live Tables: specifying and managing the DAG of a streaming pipeline.

Instead of using regular job clusters, interactive clusters or SQL warehouses, dlt has its own type of compute. It is more expensive than jobs compute, but can be cheaper than interactive compute, depending on the dlt product edition (core, pro or advanced).

Dlt is native to Databricks, meaning that you cannot move your transformation code as easily as with dbt. Migration to another data warehouse will only be relatively easy if your transformations are non-streaming and written in SQL.

Pros

- Supports streaming pipelines (<-> dbt)
- Provides data lineage
- Automated data quality testing

Cons

- Is less portable to other data warehouses than dbt
- More expensive compute than regular Databricks jobs

Conclusion

As you can see: many tools exist to orchestrate your Databricks workloads. I hope this article has given you some insights on which one is best suited for your specific use case.

Are you interested in having one of these solutions implemented at your organization? Don't hesitate to get in touch with us via the contact details below.

Good luck with your Databricks adventures!

Thibault Fabry
Business Development Manager
Tobania.Data

thibault.fabry@pythonpredictions.com
+32 475 80 48 93

PYTHON PREDICTIONS
A TOBANIA COMPANY