## Introduction

Data lineage is a critical aspect of data management that enables organizations to track what data sources they have and how they are used. When a company is small and young, understanding your data lineage is relatively simple. As a company grows, though, and the number of teams both consuming and producing data increases, the challenge of fully understanding data lineage also grows. This leads to scenarios like the one recently described by the analytics team of a large bank: “We consume data. We know what tables we use. Where they’re coming from? We don’t know. If there’s a problem, don’t know. If we find something weird, we kick it upstream, and tell them, something’s going on.”

Despite the increased recognition and emphasis on data lineage in recent years, many people only have a rough sense that it’s something their organization ought to have. In this blog post, we cover what you need to know when deciding how to actually implement data lineage in your organization.

## Data lineage is a feature

One of our beliefs here at Bigeye is that lineage is a feature, not a product. As such, it’s usually bundled with either data quality tools or data catalogs.

“Lineage is a feature that helps you do other things,” Egor Gryaznov, Bigeye’s CTO says. “For example, Bigeye needs lineage so we can help our users get to the root cause of the problem and understand what's being impacted. It's a tool that is helping the user get to their end goal, which is to figure out where the problem is coming from.”

For a data catalog, presenting data lineage as a graph that you can click around helps users discover relevant tables for their work. As Gryaznov puts it: “I want to navigate to things that are upstream, read about them, understand who owns them and what they are used for, and how they are used in catalogs.”

## OpenLineage

[OpenLineage](https://openlineage.io/) is the industry-standard open-source framework for data lineage. It provides a standard interface for you to publish lineage in a structured way to a standardized repository. It allows for a more consistent experience when integrating lineage with many different tools.

Note that frameworks like OpenLineage don’t actually automatically collect the lineage. Rather, it’s like an API that you have to declare your lineage to.

## Buy or Build

Generally speaking, we recommend purchasing rather than building a custom lineage solution.

It’s unlikely that lineage is a core competency or problem for your team, and while there may be situations where you have highly customized ETL processes that necessitate building lineage into the ETL framework - like Uber did with its SQL templating ETL framework - this is uncommon. In most cases, if you are using industry-standard tools, we recommend that you first check whether those tools already have lineage bundled in as a feature.

## What’s available out-of-the-box from data warehouses?

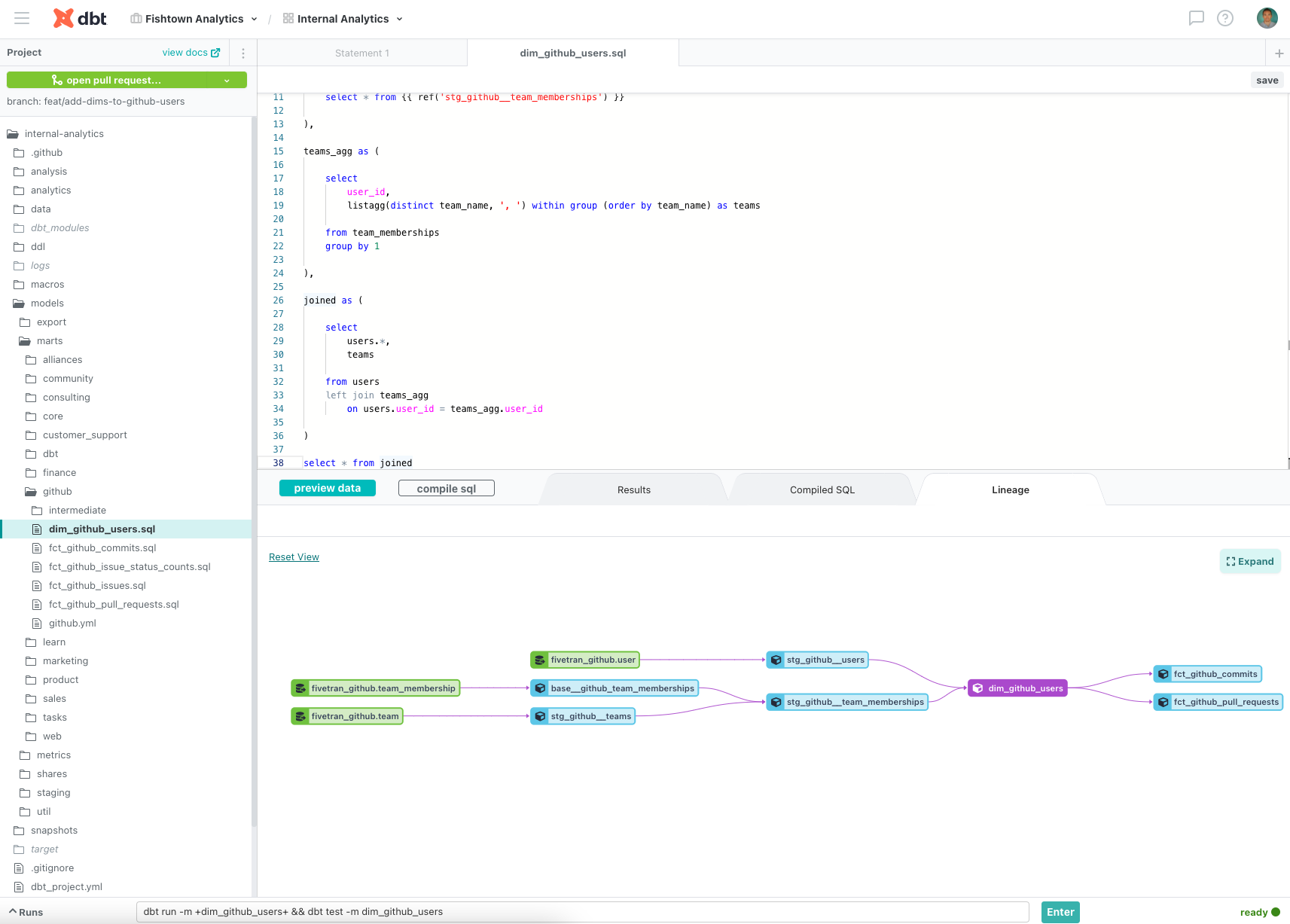
BigQuery, Snowflake and Redshift make available [some amount of data lineage information out-of-the-box](https://docs.snowflake.com/en/sql-reference/account-usage/access_history.html). They do this by breaking down queries run in the data warehouse. For example, if you have a query that reads from one table and writes into another, or reads from many and writes into one, then the data warehouse will log that these objects were accessed by this query and this object was written into by this query.

Subsequently, users can go into the data warehouse and ask, for each query, which tables were read and which tables were written into.

While this gives you basic lineage information, you run into limitations fairly quickly. For example, [DBT models are expressed as views](https://docs.getdbt.com/docs/build/materializations#configuring-materializations) rather than materialized tables. This means it’s just a SQL statement base, and there’s no table that is ever written “into”.

Note:: More traditional “transactional” systems like MySQL, Postgres, SQL Server, etc, do not typically provide lineage information, because transactional databases see many more queries every day than analytical databases, and they are unable to store the logs for more than a few days.

## What’s available out-of-the-box from DBT?

If you are using DBT for your data transformations in-warehouse, relationships between data sources and models can be automatically inferred; if you use DBT cloud, a visualization of this DAG that represents the data lineage can be viewed and explored. 

The DAG provides a visual representation of the upstream dependencies, the models that must be processed before the current one, and the downstream relationships, the models impacted by the current one. It also shows the flow of data transformations in a defined and non-cyclical manner.

You can additionally add documentation and meta fields to nodes in the DAG.

### What’s available for Airflow lineage?

For many organizations, a large percentage of their data pipelines are scheduled with Airflow. If this is the case for your organization, you can leverage this by embedding your data lineage ingestion *into* Airflow.

If you are using a hosted Airflow like Astronomer, it’s likely that the [Airflow already comes with an OpenLineage integration](https://www.astronomer.io/blog/3-ways-to-extract-data-lineage-from-airflow/), and certain common Airflow operators, when used in a DAG, will emit OpenLineage events. You will also be able to write custom lineage extractors for your own Airflow operators.

Even if you are using open-source Airflow, off-the-shelf lineage services will usually provide hooks/callbacks that can be used to send metadata about Airflow tasks to the lineage service.

## What’s available from off-the-shelf solutions like Atlan, Callibra?

Once you’ve collected your lineage metadata information from DBT or Airflow or your dashboards, it still needs to be additionally processed and assembled into a graph.

To do this, you might turn to an off-the-shelf data lineage solution like Atlan or Collibra. These solutions will usually ask you to connect your data warehouse and assemble the data lineage automatically via some combination of:

* Piggy-backing off the parsing that warehouses have already done
* Extracting information from DAG/task/job name metadata sent from callbacks in Airflow or dashboards
* Writing their own custom SQL parsers to parse and extract lineage from SQL
  + You may be wondering why SQL-parsing is necessary if the metadata has already been declared from something like Airflow? The biggest advantage to extracting lineage from production SQL is scalability. While metadata/job documentation may get stale, the SQL actually populating the tables is always ground truth.

## Building your own lineage graph

Despite the proliferation of off-the-shelf lineage services, there are still reasons why you might want to build your own - for example the need for column-level lineage and language support.

One of Bigeye’s clients, for example, has an ETL pipeline that is essentially data in Snowflake, with Python and R scripts that read the data, do some computation, and write it back out. In order to get lineage from that, you have to have access to that Python code and be able to parse it and understand it.

Given that it’s obviously difficult from an engineering perspective to write all the parsers and all the automation around this, it’s highly recommended that as much as possible, you declare lineage explicitly in your data pipeline job configuration.

[Here](https://slack.engineering/data-lineage-at-slack/) are [some](https://www.uber.com/blog/databook/) [references](https://netflixtechblog.com/building-and-scaling-data-lineage-at-netflix-to-improve-data-infrastructure-reliability-and-1a52526a7977) on how some large organizations built their own data catalog/data lineage services.