Data Saturday Holland

dbt with Databricks

Transformations at Scale!

Thank you sponsors!















RateData Saturday Holland

SUPPORT CASPER

1 review = 1 €

Towards beating pancreatic cancer

About the Presenters



Shasidhar Eranti
Sr. Specialist Solutions Architect @ Databricks, Amsterdam

Data Engineering and Streaming



Kamalendu Biswas (Kamal/Biswas)
Specialist Solutions Architect @ Databricks, Amsterdam

Data Warehousing and Migrations

Agenda

- Databricks Introduction
  - Data Intelligence Platform
  - Databricks SQL
- dbt Introduction
- dbt Integration with Databricks
- Demo Scenario
  - Classic ingestion demo
- Challenges
- Latest dbt <> Databricks Features
  - Ingestion with Streaming Tables
  - Materialized Views
- What's next & Questions
  - dbt Roadmap - 5min

10,000+
global customers

$1.5B+
in revenue

$4B
in investment

Inventor of the **lakehouse**
&
Pioneer of **generative AI**

databricks

The data and AI company

Gartner-recognized Leader
Database Management Systems
Data Science and Machine Learning Platforms

Creator of

DELTA LAKE

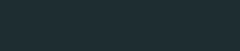
mlflow™

APACHE Spark™

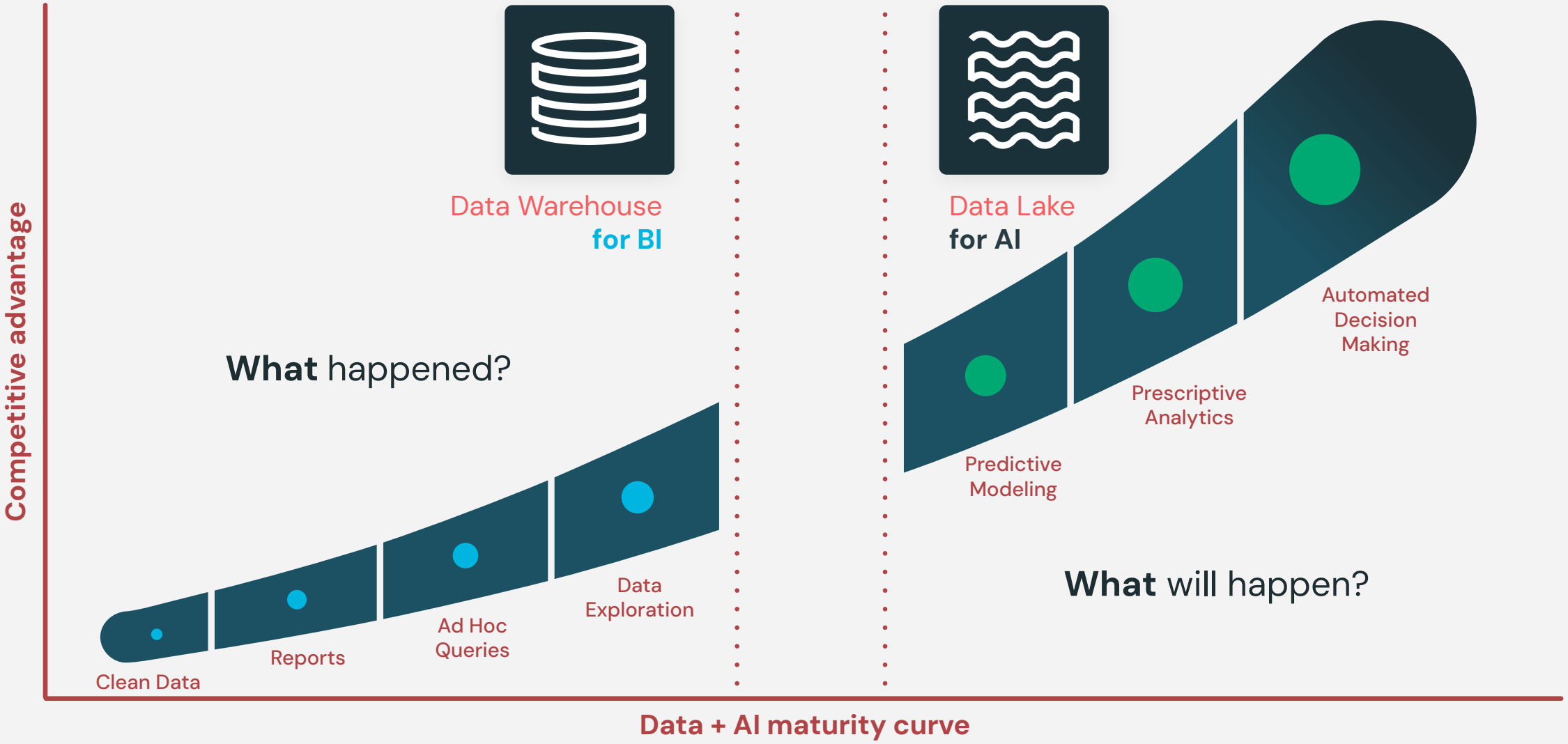
Mosaic MPT

©2024 Databricks Inc. — All rights reserved

Every company wants to become a Data+AI company



Two incompatible architectures get in the way



Databricks invented and championed the Lakehouse category

What Is a Lakehouse?

by Ben Lorica, Michael Armbrust, Reynold Xin, Matei Zaharia and Ali Ghodsi

January 30, 2020 in Engineering Blog

What is a lakehouse in Microsoft Fabric?

Article • 08/30/2023 • 5 contributors

Feedback

BigLake

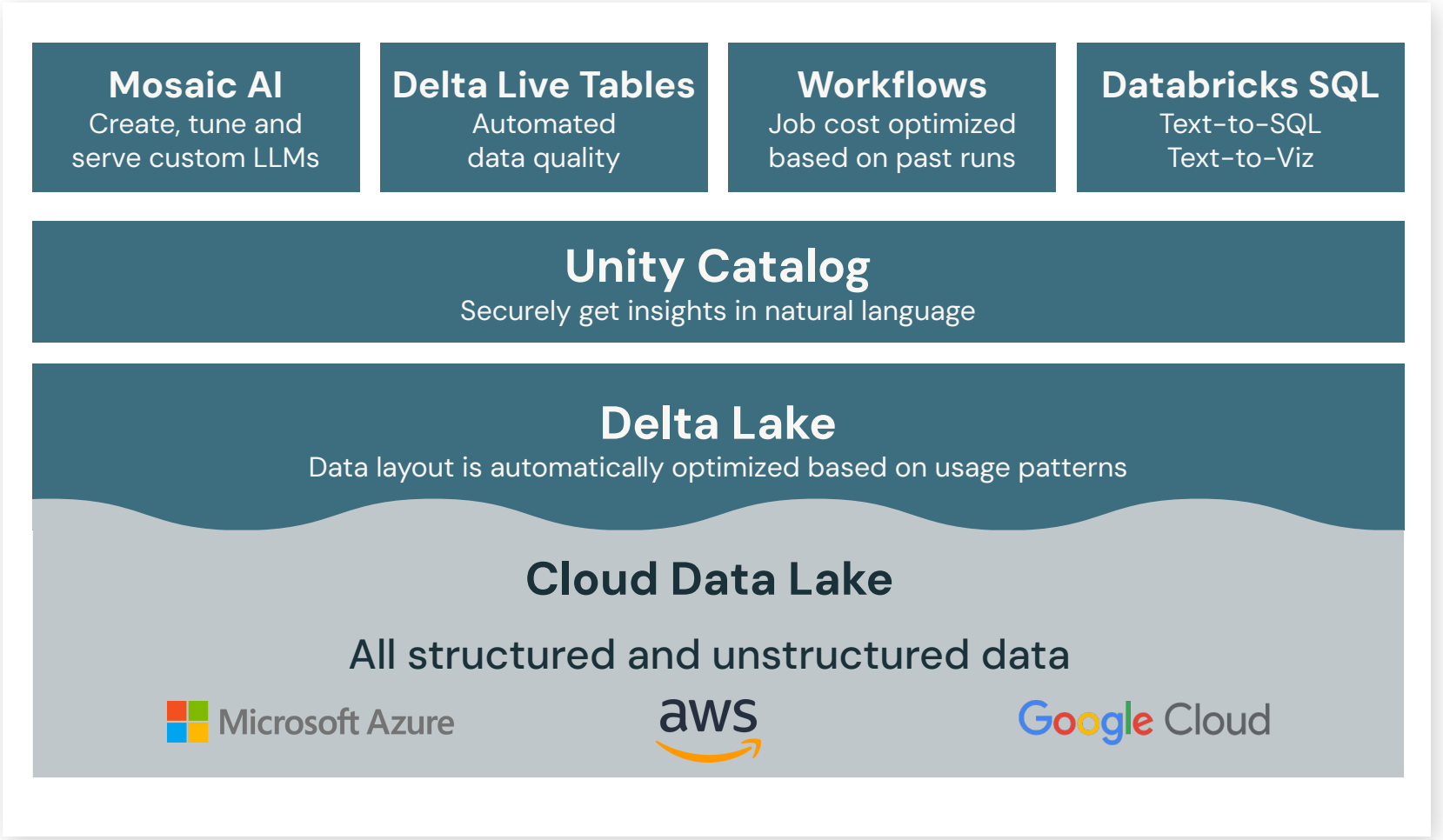
BigLake is a storage engine that provides a unified interface for analytics and AI engines to query multiformat, multicloud, and multimodal data in a secure, governed, and performant manner. Build a single-copy AI lakehouse designed to reduce management of and need for custom data

The lake house approach to data warehousing with Amazon Redshift

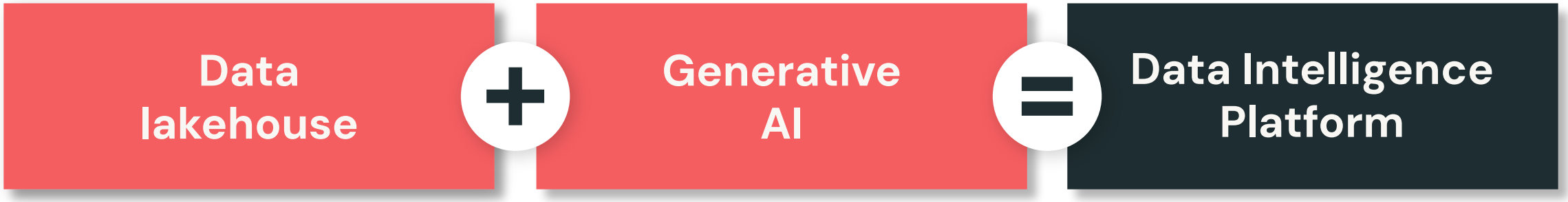
SNOWFLAKE FOR DATA LAKEHOUSE

Transactional data lake architecture pattern for unified analytics, AI/ML, and other collaborative workloads.

Databricks Lakehouse



Data Intelligence Platform

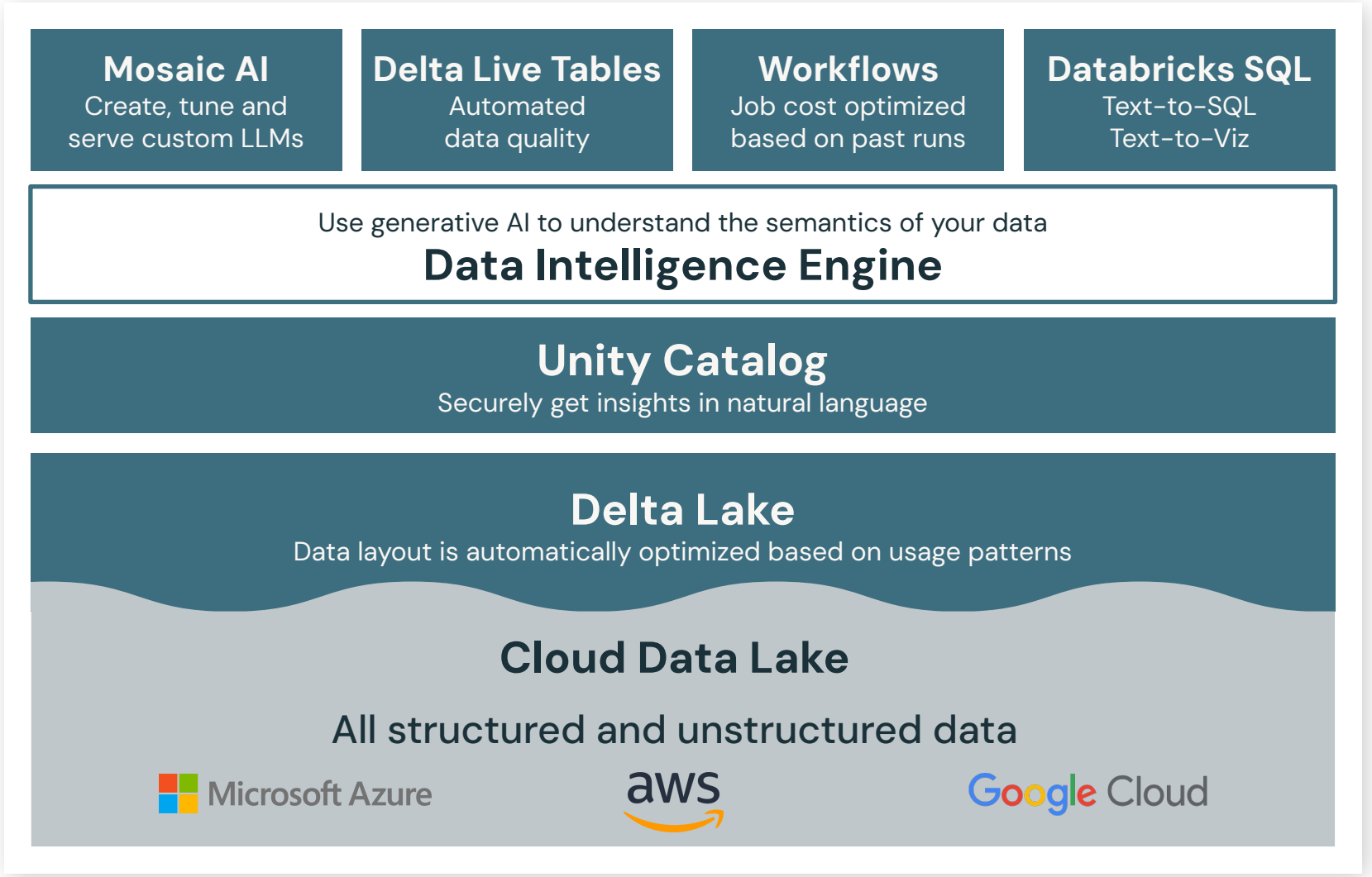


An open, unified foundation for all your data

Easily scale and use data and AI

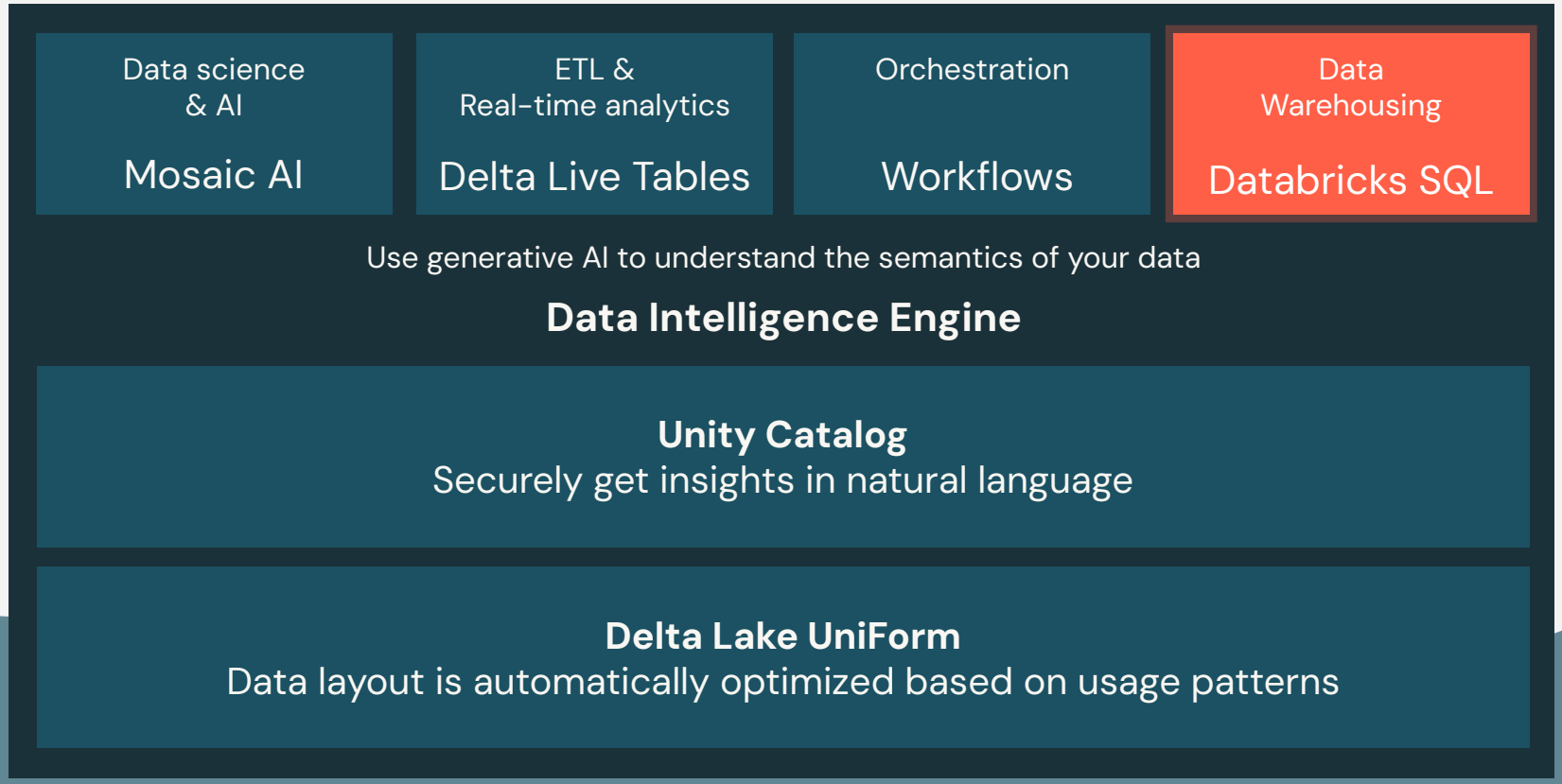
Democratize data + AI across your entire organization

Databricks Data Intelligence Platform



Databricks SQL

Databricks Data Intelligence Platform



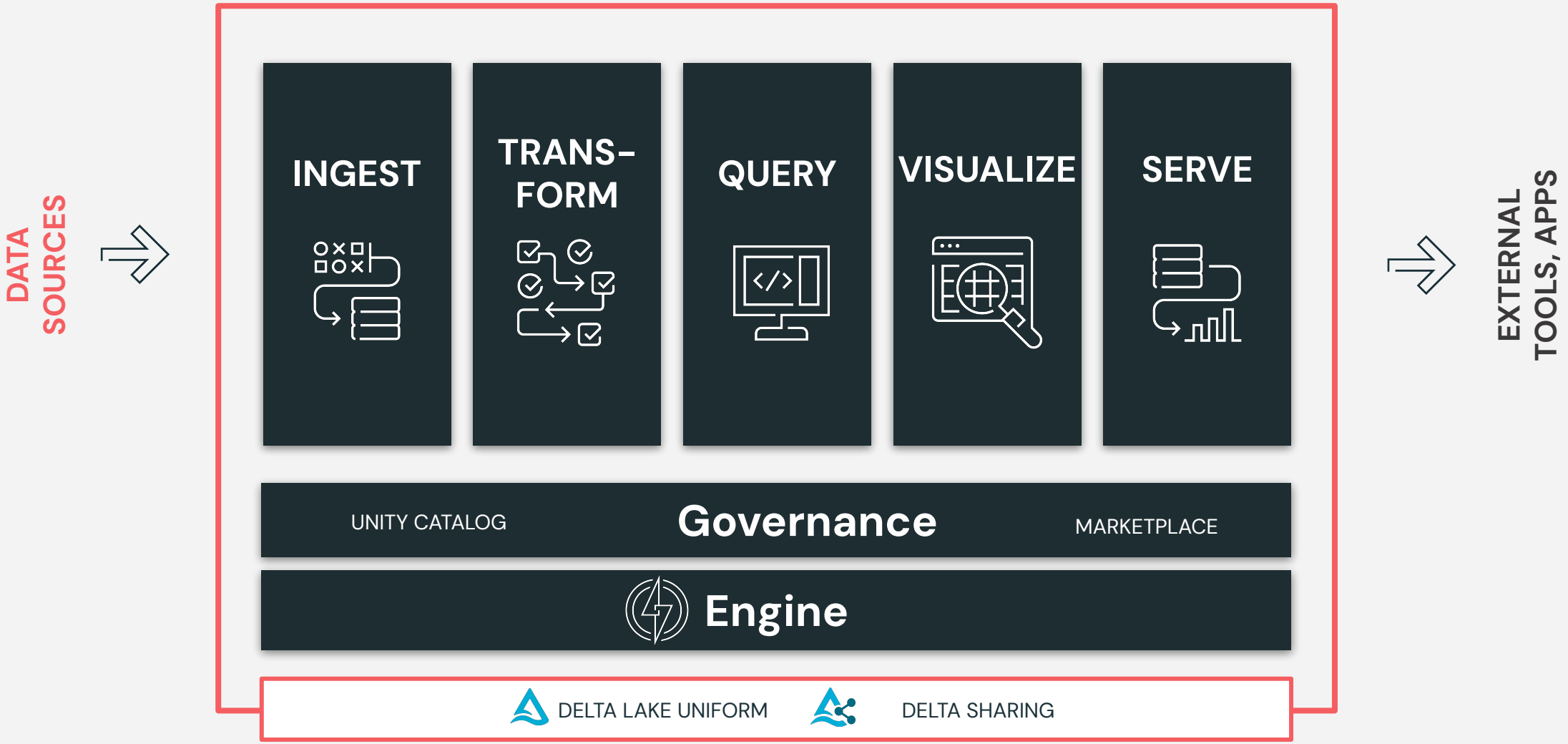
Open Data Lake

All raw data
(Logs, Texts, Audio, Video, Images)

Databricks SQL

- Text to SQL
- AI-driven queries
- AI-driven serverless computing scales for cost efficiency and peak performance
- AI-driven debugging and remediation

Databricks SQL



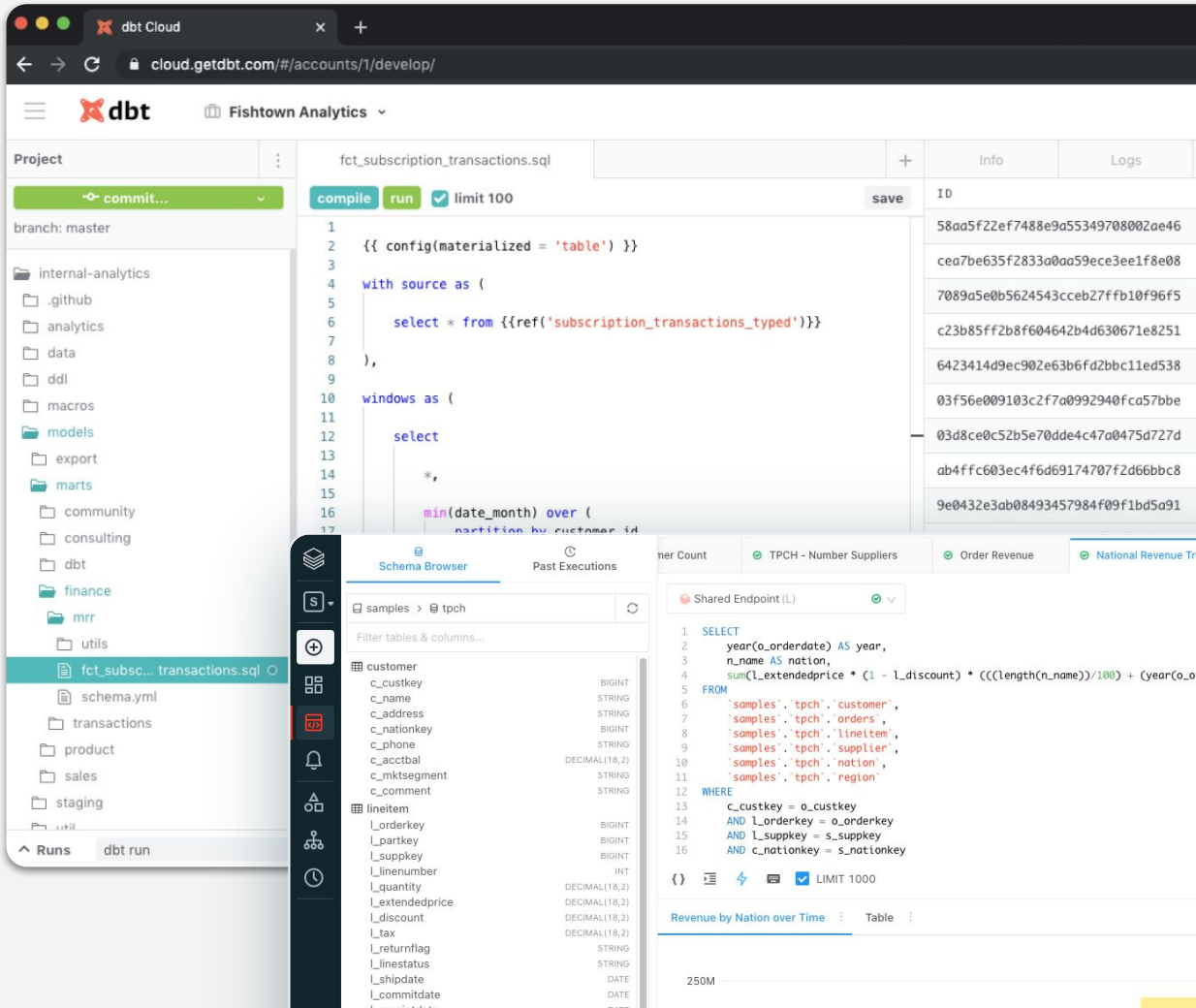
Data Transformation: dbt Labs

Analytics engineering on the Lakehouse made simple

Databricks and dbt Labs **simplify analytics engineering on the lakehouse.**

[NEW] Users can ingest and transform streaming data in their dbt pipelines **using Streaming Tables and Materialized Views**.

Run your dbt projects as a task in a Databricks job to automate tasks and schedule workflows



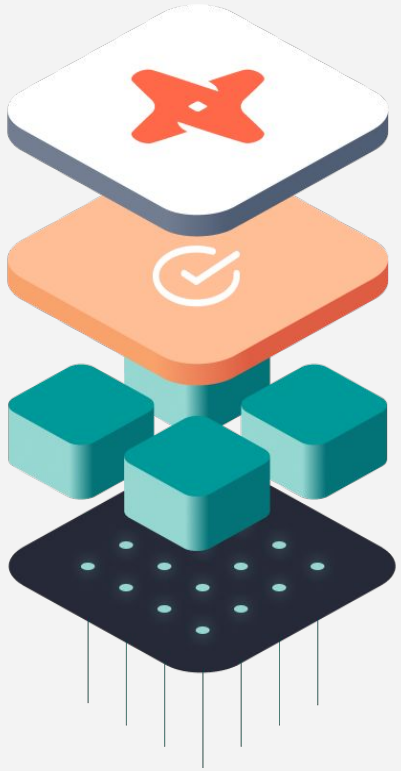
databricks + dbt

©2024 Databricks Inc. — All rights reserved

dbt

What is dbt?

dbt™ is a SQL-first transformation workflow that lets teams quickly and collaboratively deploy analytics code following software engineering best practices like modularity, portability, CI/CD, and documentation.



How dbt works

Version Control and CI/CD
Deploy safely using dev environments. Git-enabled version control enables collaboration and a return to previous states.

Test and Document
Test every model prior to production, and share dynamically generated documentation with all data stakeholders.

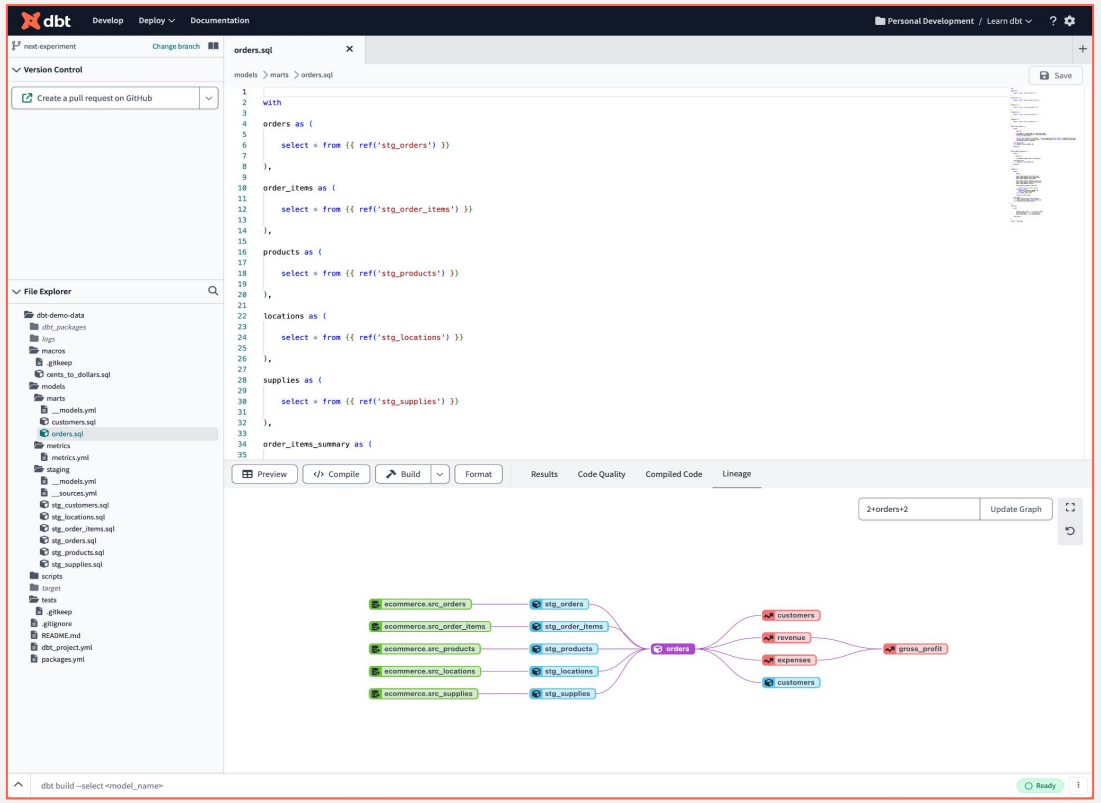
Develop
Write modular data transformations in **.sql** or **.py** files – dbt handles the chore of dependency management.

Analytics Engineering with dbt

A SQL-based development environment for end-to-end data transformation

1 **SQL-based** lowers barrier to entry

2 **Version Control** for CI/CD-ready



3 **Job Scheduling** from Cloud IDE

4 **Integrated testing** for quality

5 **Macros** for reusability

6 **DAG inferred** as you code

7 **Documentation** auto-generated

dbt
Core Concepts

dbt Core Concepts

Sources	Name and describe the data loaded into your warehouse by EL tools `{{ source() }}`
Targets	Information about your connection to the warehouse `target.name/type/schema/env`
Seeds	CSV files in your dbt project, to be loaded onto your data warehouse using the `dbt seed` command
Models	Transformations primarily written as a `select` statement and saved as a `.sql` file (Python models are also supported)
Macros	Macros in Jinja are pieces of code that can be reused multiple times – they are analogous to `functions`
Snapshots	Implement type-2 Slowly Changing Dimensions over mutable source tables

dbt Materialization

Materializations	Strategies for persisting dbt models in a warehouse. There are four types of materializations built into dbt: • ***view*** - model is rebuilt as a view on each run, via a `create view as` statement • ***table*** - model is rebuilt as a table on each run, via a `create table as` statement • ***incremental*** - allows dbt to `insert or merge records` into a table since the last time that dbt was run • ***ephemeral*** - dbt refers this model as a `Common Table Expressions`

Databricks supports:

- ***materialized_view*** - MVs store `pre-computed results` based on the latest data from source tables
- ***streaming_table*** - STs enable `continuous, scalable ingestion` from any data source including cloud storage, message buses (EventHub, Apache Kafka) and more

dbt Commands

dbt <command>

deps	Downloads dependencies for a project
seed	Loads CSV files into the database
test	Executes tests defined in a project
run	Runs the models in a project
build	**Build and test all selected resources (models, seeds, snapshots, tests)**
retry	Retry the last run dbt command from the point of failure
docs	Generates documentation for a project

Dependency Management

Airflow Tasks Dependencies

```
start_task >> [
                extract_orders_task,
                extract_customers_task,
                extract_products_task
                ]

extract_orders_task    >> merge_data_task
extract_customers_task >> merge_data_task
extract_products_task  >> merge_data_task

merge_data_task >> [clean_data_task,
                    validate_data_task
                   ]

clean_data_task >> combine_data_task
transform_data_task >> combine_data_task
validate_data_task >> run_data_quality_checks_task
combine_data_task >> load_data_task
run_data_quality_checks_task >> load_data_task

load_data_task >> end_task
```

dbt Auto-Generated Dependencies

```
SELECT
  ord.order_key       AS  order_key,
  ord.total_price     AS  total_price,
  cust.customer_key   AS  customer_key,
  con.country_key     AS  country_key,
  dt.date_key         AS  date_key
FROM
  {{ ref('dim_order') }} ord
  LEFT JOIN   {{ ref('dim_customer') }}  cust
      ON cust.customer_id = ord.customer_id
  LEFT JOIN   {{ ref('dim_country') }}  con
      ON cust.country_id  = con.country_id
  INNER JOIN  {{ ref('dim_date') }}  dt
      ON dt.date_full     = ord.order_date
```

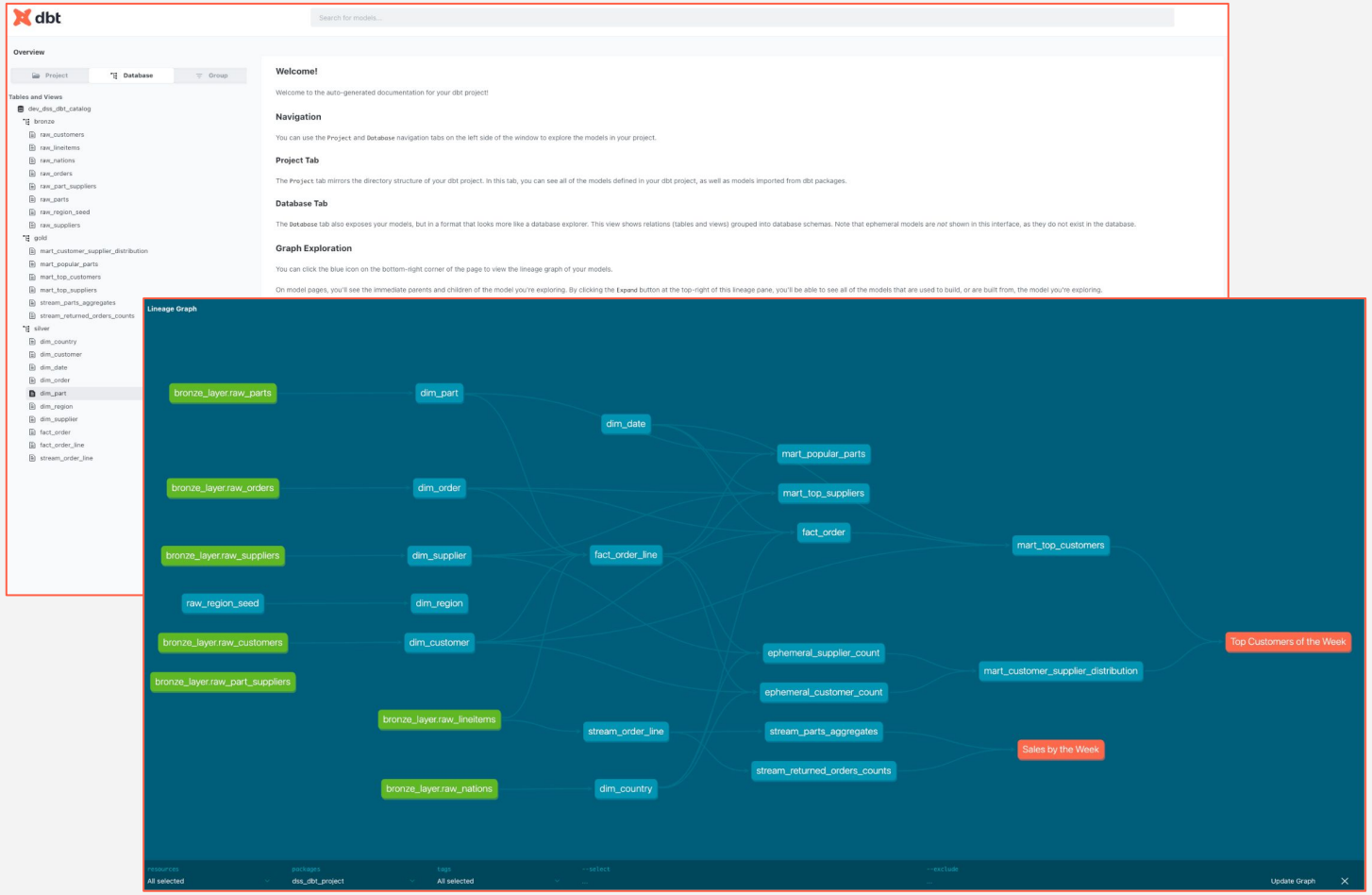
Documentation & Lineage

Unity Catalog Users:

- Can upload dbt docs to UC
- Leverage UC Lineage
  - Column Level
  - Cross Platform (Query Federation)
  - Includes all data artefacts
  - Available via API or System Tables
  - Supports ERD with PK and FK

Non Unity Catalog Users:

- Generated out of your dbt project
  - Information about your project
  - Information about your data warehouse
- Hosting options
  - Use dbt Cloud
  - Host on S3/ADLS/GCS
- Show and track lineage between models



dbt + Integration with Databricks

Getting Started

- dbt-databricks package
  - Open source, forked from dbt-spark
  - Lots of Databricks-specific niceties:
    - Delta by default,
    - no ODBC dependency
    - Unity Catalog

- Running in Workflows
  - Support for running dbt projects as a job task
  - All you need is access to the git repo with your project

- Running in dbt Cloud - via Partner Connect
  - Simple **dbt Cloud** integration uses dbt-databricks

dbt Targets

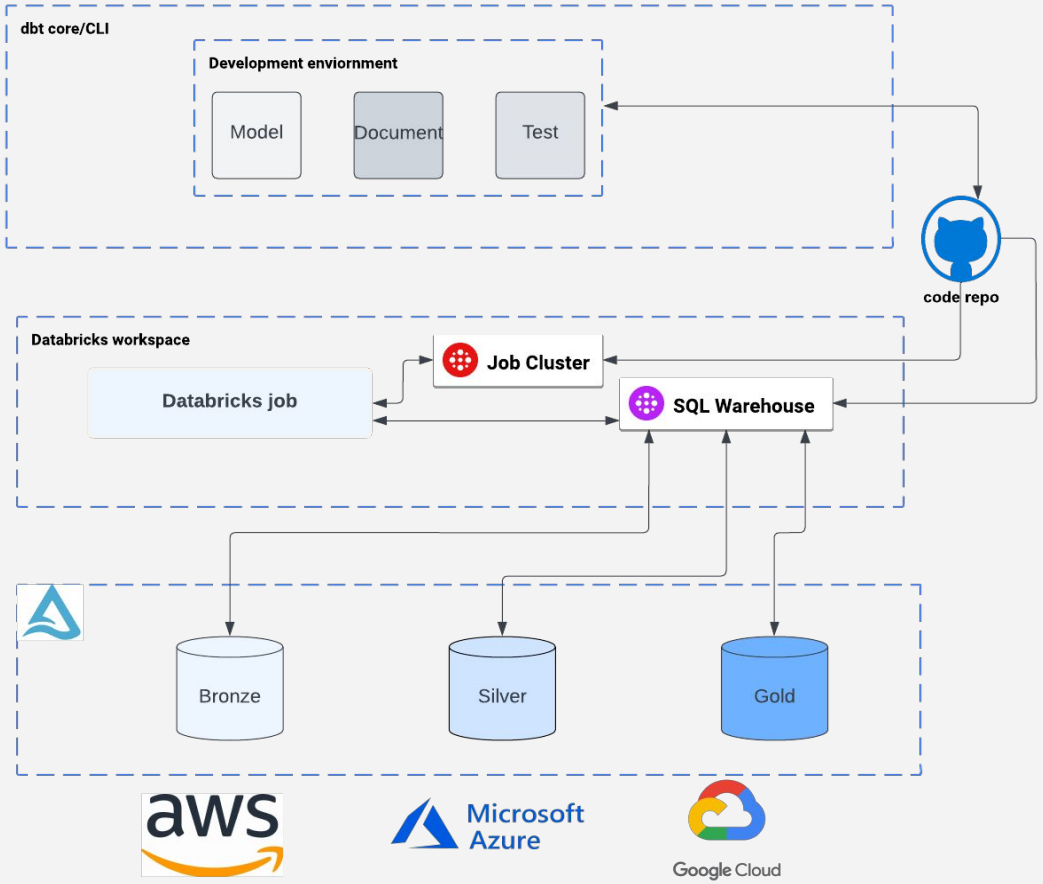
target	This is the default target your dbt project will use. It must be one of the targets you define in your profile. Commonly it is set to **dev**.
type	Databricks
catalog	Databricks catalog, default is hive_metastore
schema	Databricks schema
host	All Purpose Cluster or SQL Warehouse
http_path	Databricks compute resources URL
threads	The number of threads the dbt project will run on
token	Databricks access token `{{ env_var('DBT_ACCESS_TOKEN') }}`



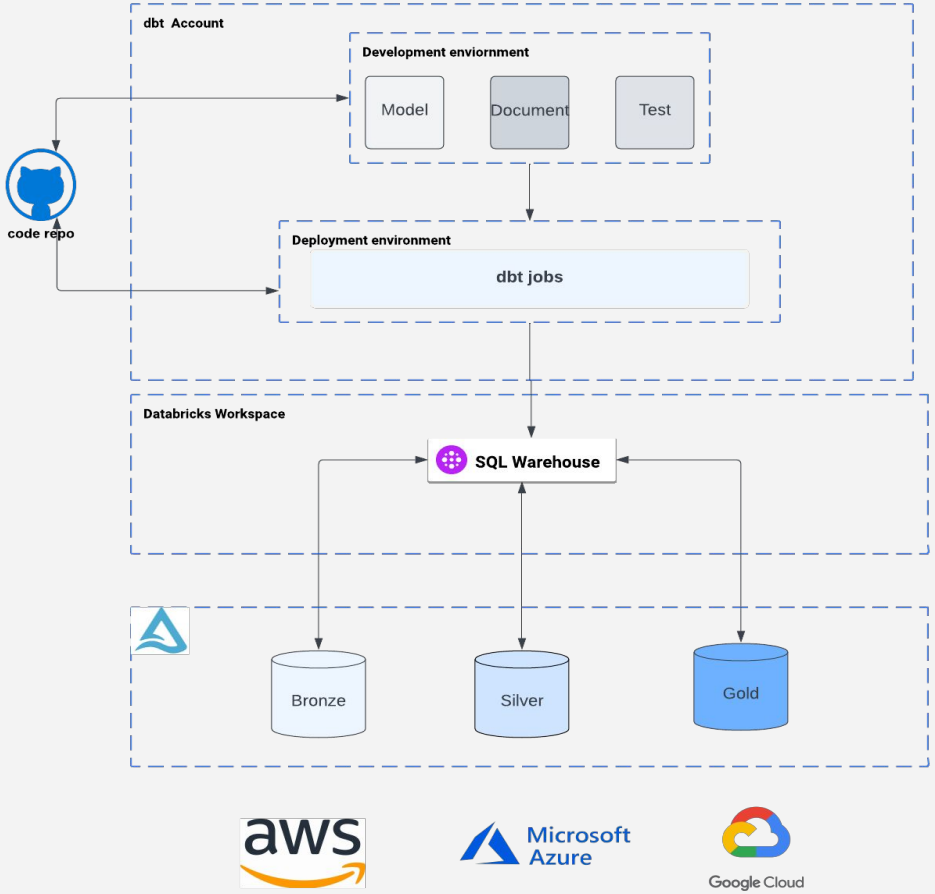
Demo

Orchestration

Databricks (dbt core)

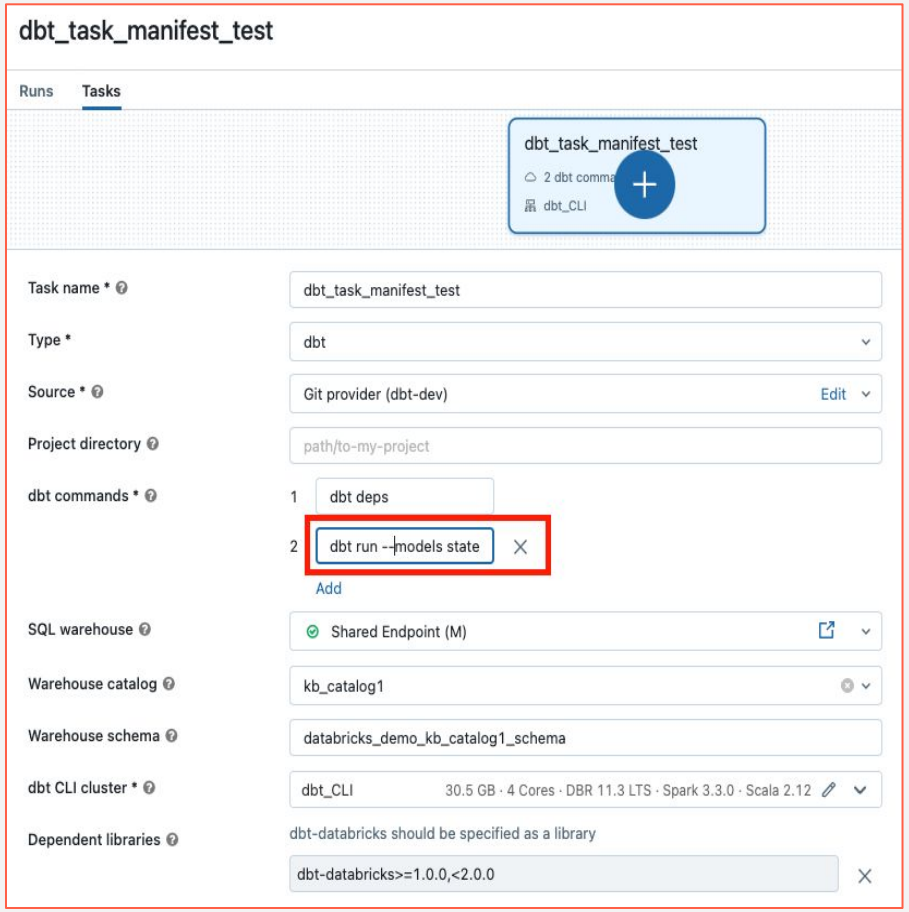


dbt cloud

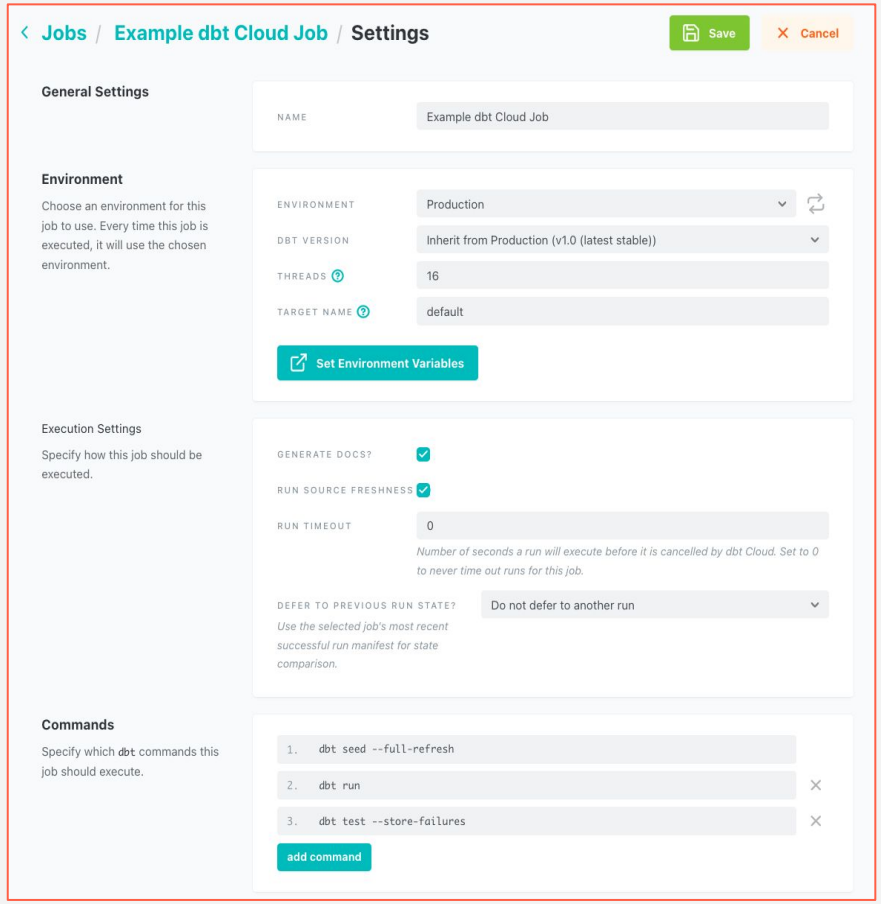


dbt Job

Databricks Workflow – dbt task



dbt Cloud Job



Classic Demo

Demo: Datasets

Streaming:

- airline_trips

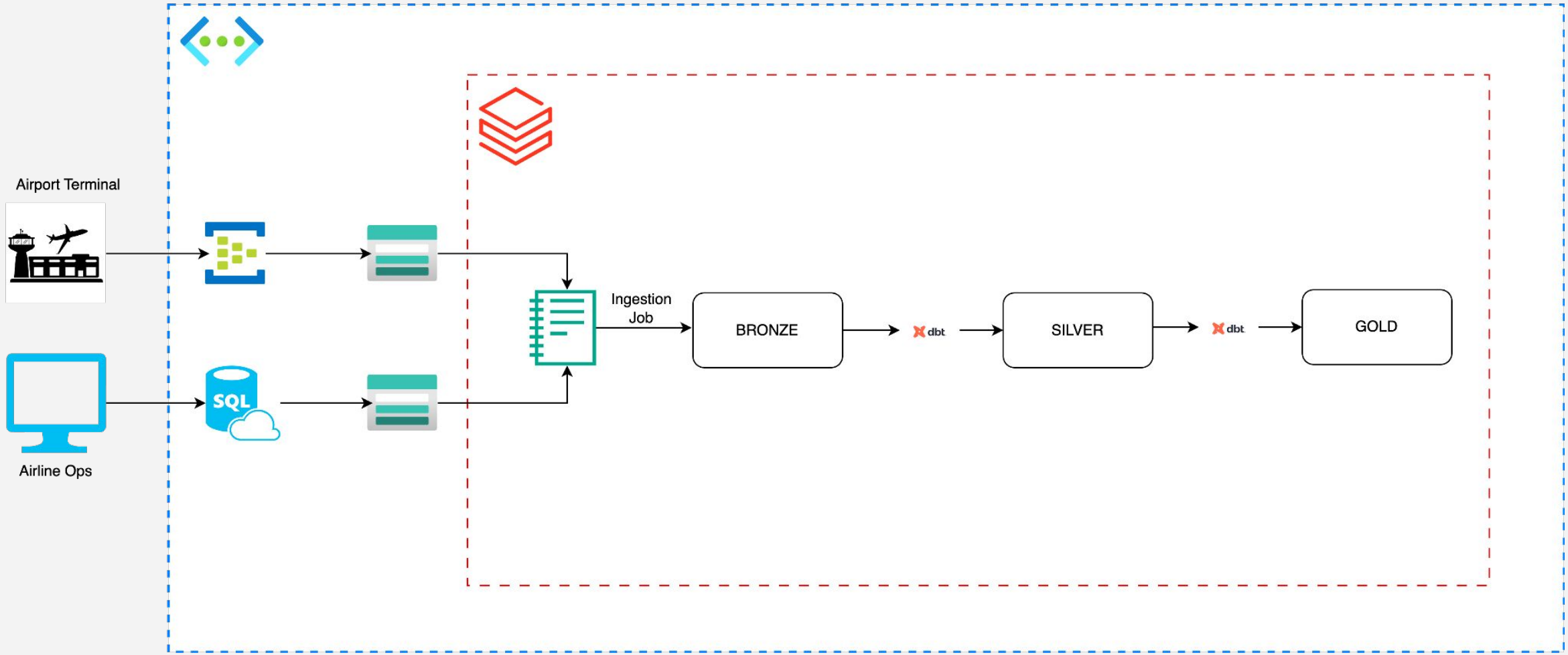
Static:

- airline_codes
- airport_codes

Demo: Concept



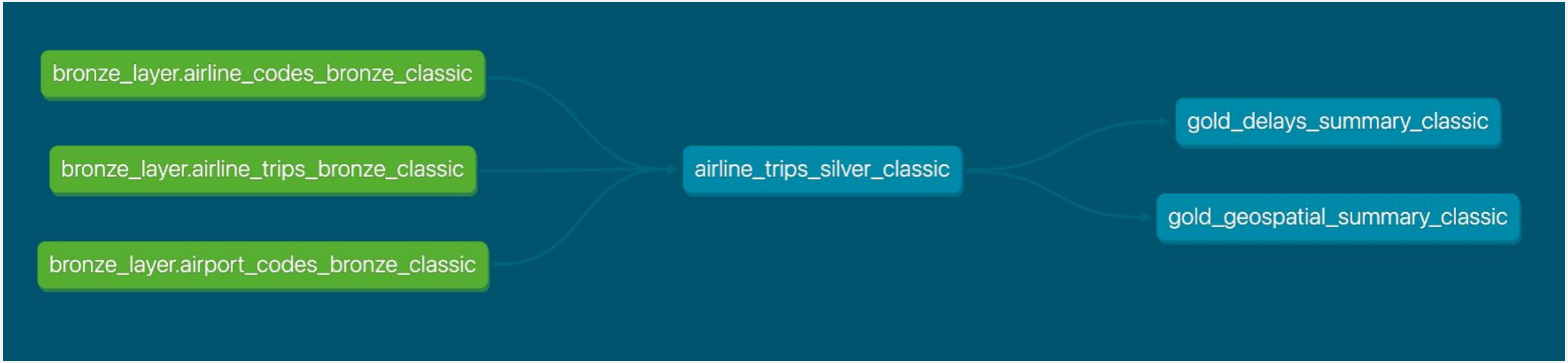
Demo: Architecture



Demo: Pipeline

Source Delta Tables:

- airline_trips [ingested using a spark streaming job]
- airline_codes [ingested using a spark batch job]
- airport_codes [ingested using a spark batch job]



Challenges

- Custom ingestion framework or job
  - Extra pre-step in the end to end process
  - External execution engine required to perform this step
  - More administrative and operational overhead
  - Additional orchestration and dependency management
- Hard to handle Reprocessing or reloading of data if required
- Non-unified medallion architecture - Bronze not part of dbt project
- Scaling can be challenging in handling higher data volumes

Integration

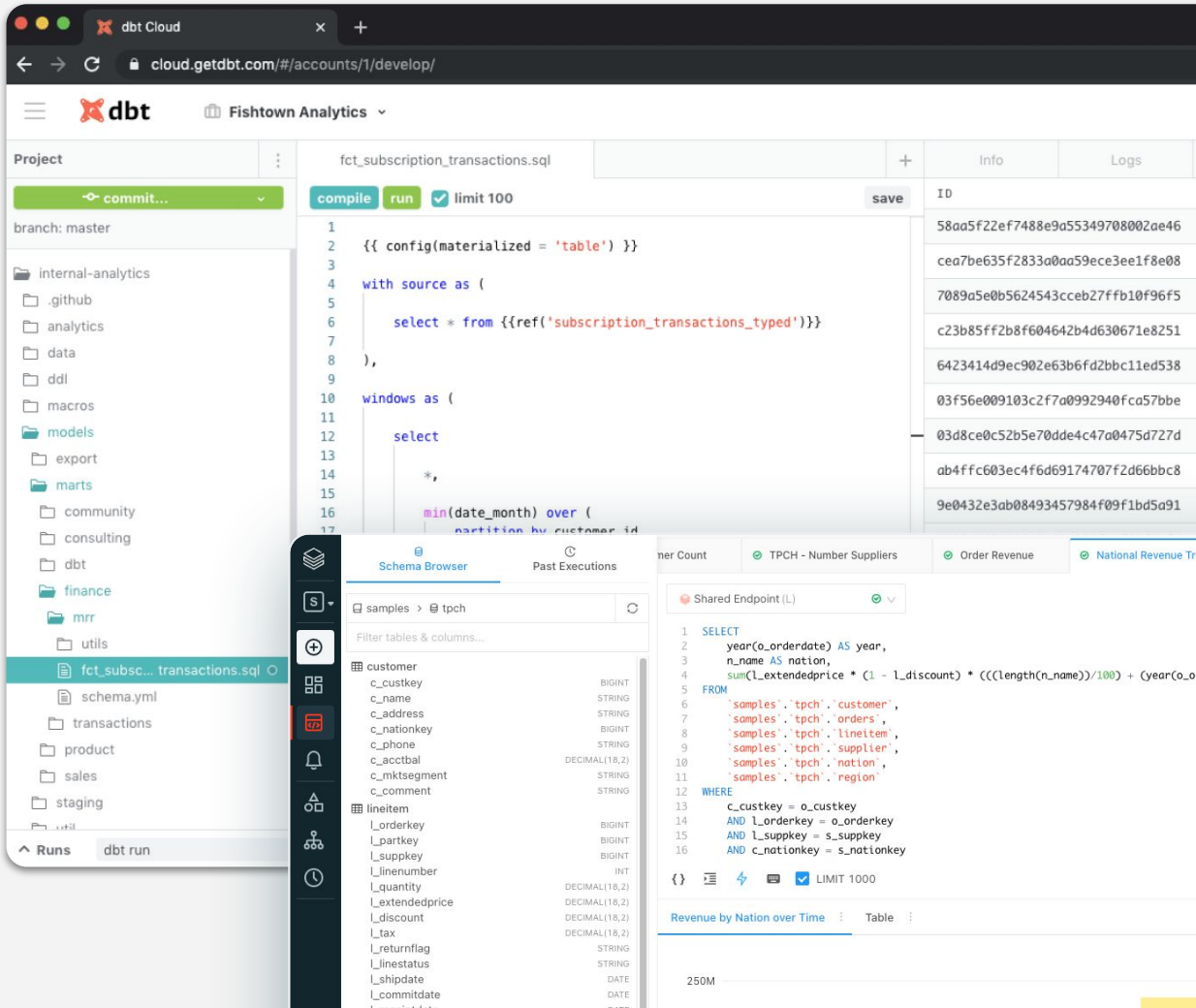
Data Transformation: dbt Labs

Analytics engineering on the Lakehouse made simple

Databricks and dbt Labs **simplify analytics engineering on the lakehouse.**

[NEW] Users can ingest and transform streaming data in their dbt pipelines **using Streaming Tables and Materialized Views**.

Run your dbt projects as a task in a Databricks job to automate tasks and schedule workflows

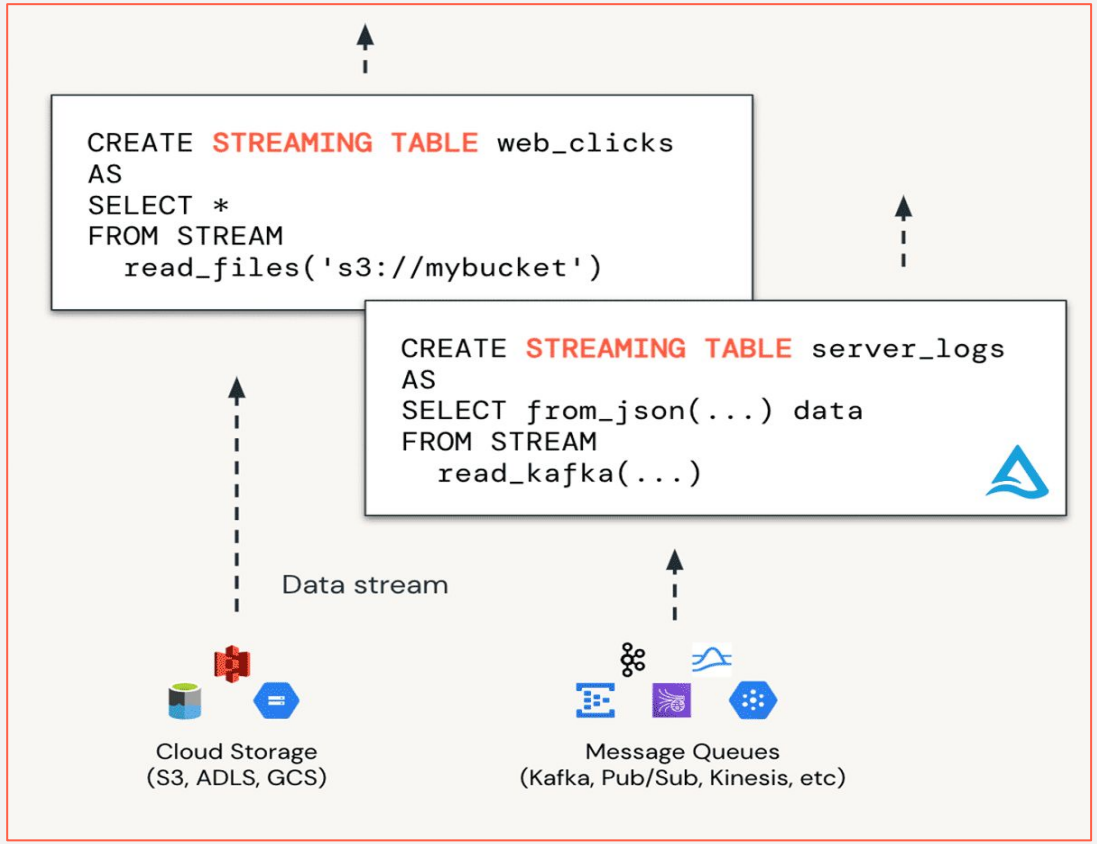


databricks + dbt

©2024 Databricks Inc. — All rights reserved

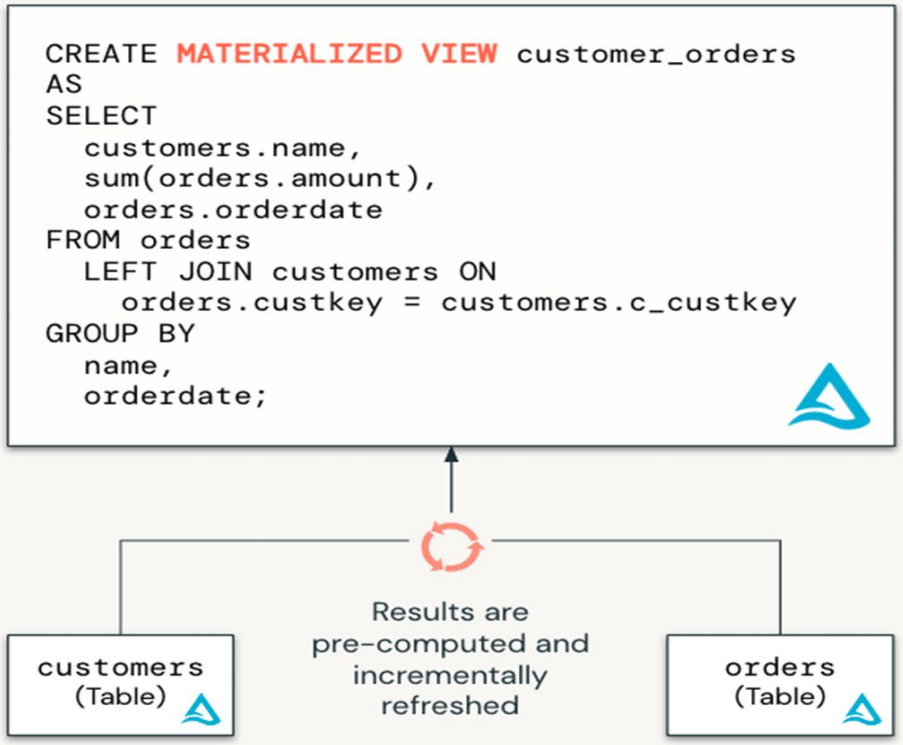
Streaming Tables

- Ideal for bringing data into “bronze” tables or even “silver” tables.
- Continuous, scalable ingestion from any data source including cloud storage, message buses (EventHub, Apache Kafka) and more.
- Ability to support real-time analytics.
- More efficiently handle high volumes of data via incremental processing vs large batches.
- Possible to schedule incremental refreshes

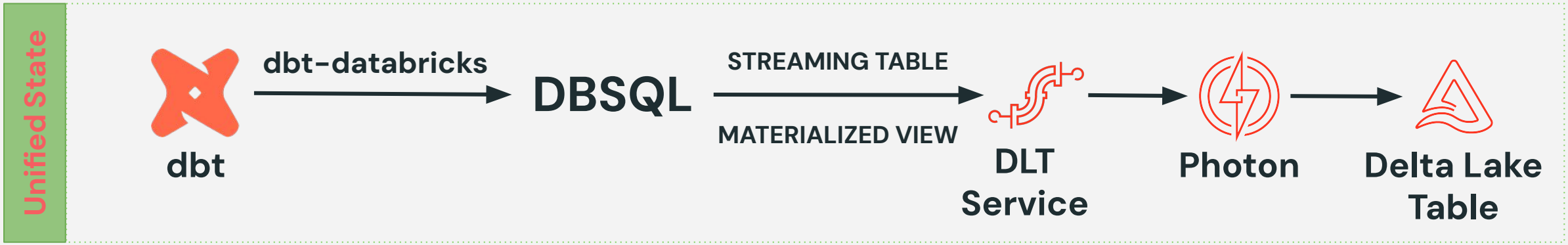


Materialized Views

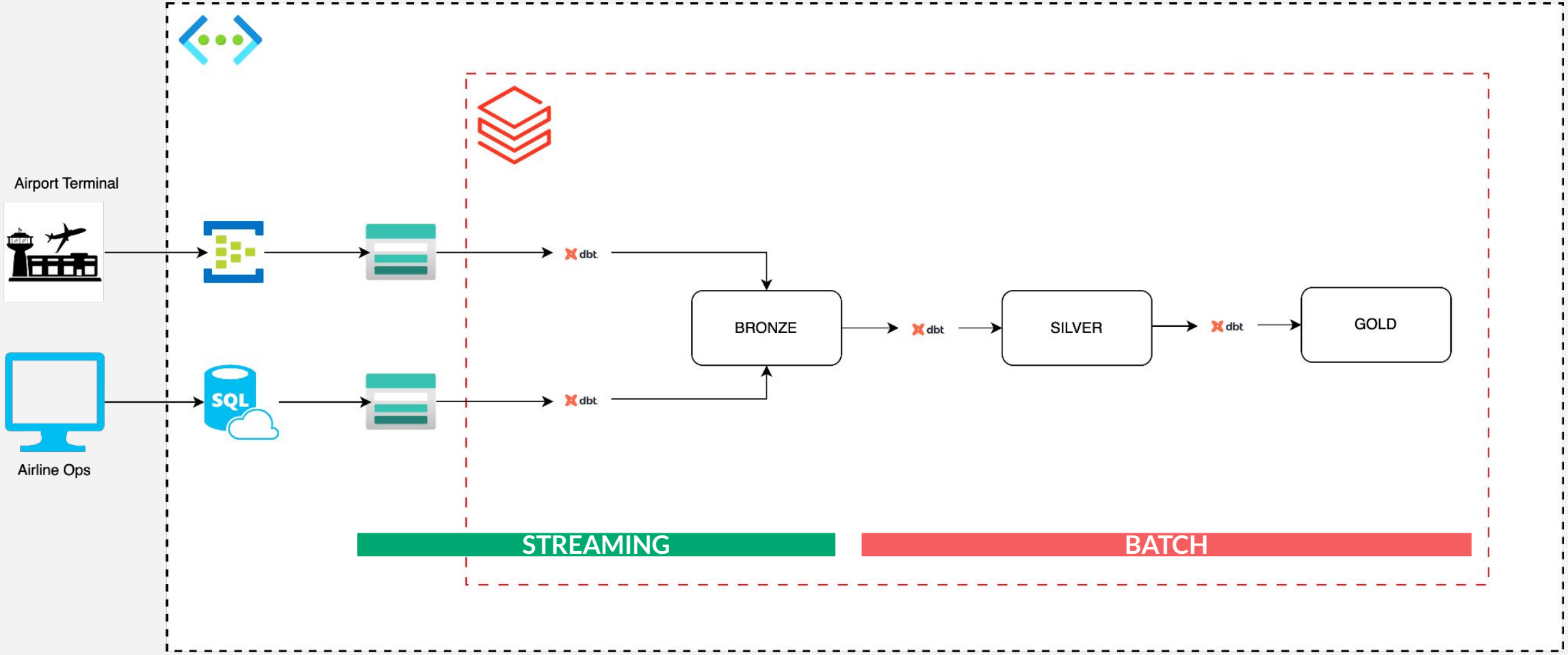
- Accelerate BI dashboards. Because MVs precompute data, end users' queries are much faster.
- Reduce data processing costs. MVs results are refreshed incrementally.
- Improve data access control for secure sharing. More tightly govern what data can be seen by consumers by controlling access to base tables.
- Possible to schedule incremental refreshes



Integration: Concept



Integration: Architecture



Integration: Pipeline



Integration Demo

What Next?

dbt Roadmap v1.9

https://github.com/dbt-labs/dbt-core/issues/10624

[EPIC] Incremental Model Improvements - Microbatch #10624

Open | 22 of 31 tasks | QMalcolm opened this issue on Aug 28 · 3 comments · Fixed by #10751



QMalcolm commented on Aug 28 • edited — Contributor

Incremental models in dbt is a materialization strategy designed to efficiently update your data warehouse tables by only transforming and loading new or changed data since the last run. Instead of processing your entire dataset every time, incremental models append or update only the new rows, significantly reducing the time and resources required for your data transformations.

Even with all the benefits of incremental models as they exist today, there are limitations with this approach, such as:

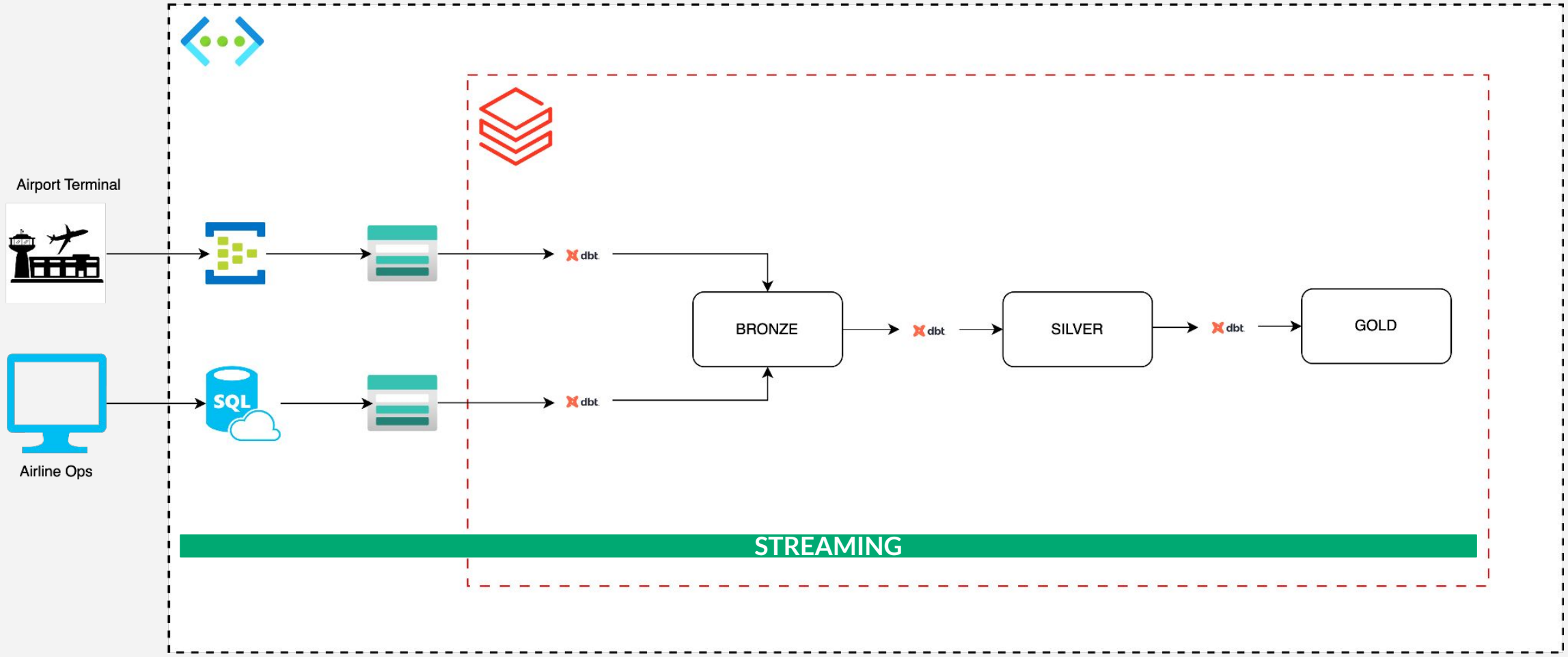
- burden is on YOU to calculate what's "new" - what has already been loaded, what needs to be loaded, etc.
- can be slow if you have many partitions to process (like when running in `full-refresh` mode) as it's done in "one big" SQL statement - can time out, if it fails you end up needing to retry already successful partitions, etc.
- if you want to specifically name a partition for your incremental model to process, you have to add additional "hack"y logic, likely using `vars`
- data tests run on your entire model, rather than just the "new" data

In this project we're aiming to make incremental models easier to implement and more efficient to run.

```
-- proposed spec
{{
    config(
        materialized='incremental',
        incremental_strategy='microbatch',
        event_time='date_day',
     batch_size='day',
     lookback_period='3'
    )
}}

select * from {{ ref('stg_events') }} # this
ref will be auto-filtered
```

dbt <> DBSQL: Future Architecture



Rate This session

Q&A

Thank you!

