Introductory Level

DATA+AI SUMMIT 2022

Dive deeper into

Data engineering on Databricks

Paul Lappas

Product Manager, Databricks

Jason Shivereck

Principal Data Scientist, Rivian

Frank Munz

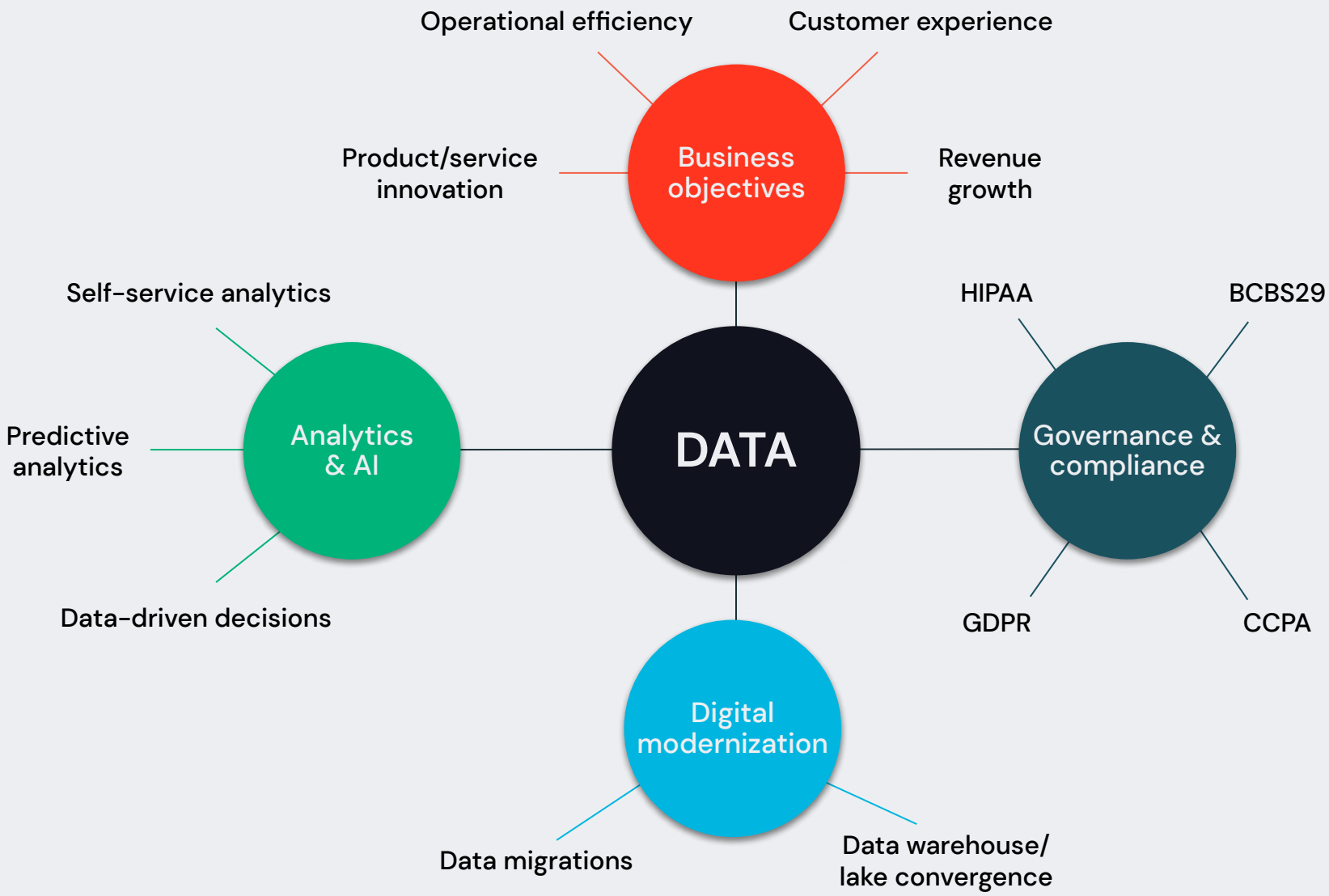
Senior Engineer, Databricks

ORGANIZED BY databricks

Product safe harbor statement

This information is provided to outline Databricks' general product direction and is for informational purposes only. Customers who purchase Databricks services should make their purchase decisions relying solely upon services, features, and functions that are currently available. Unreleased features or functionality described in forward-looking statements are subject to change at Databricks discretion and may not be delivered as planned or at all.

Data is critical to business outcomes



But data engineers have difficulty delivering data

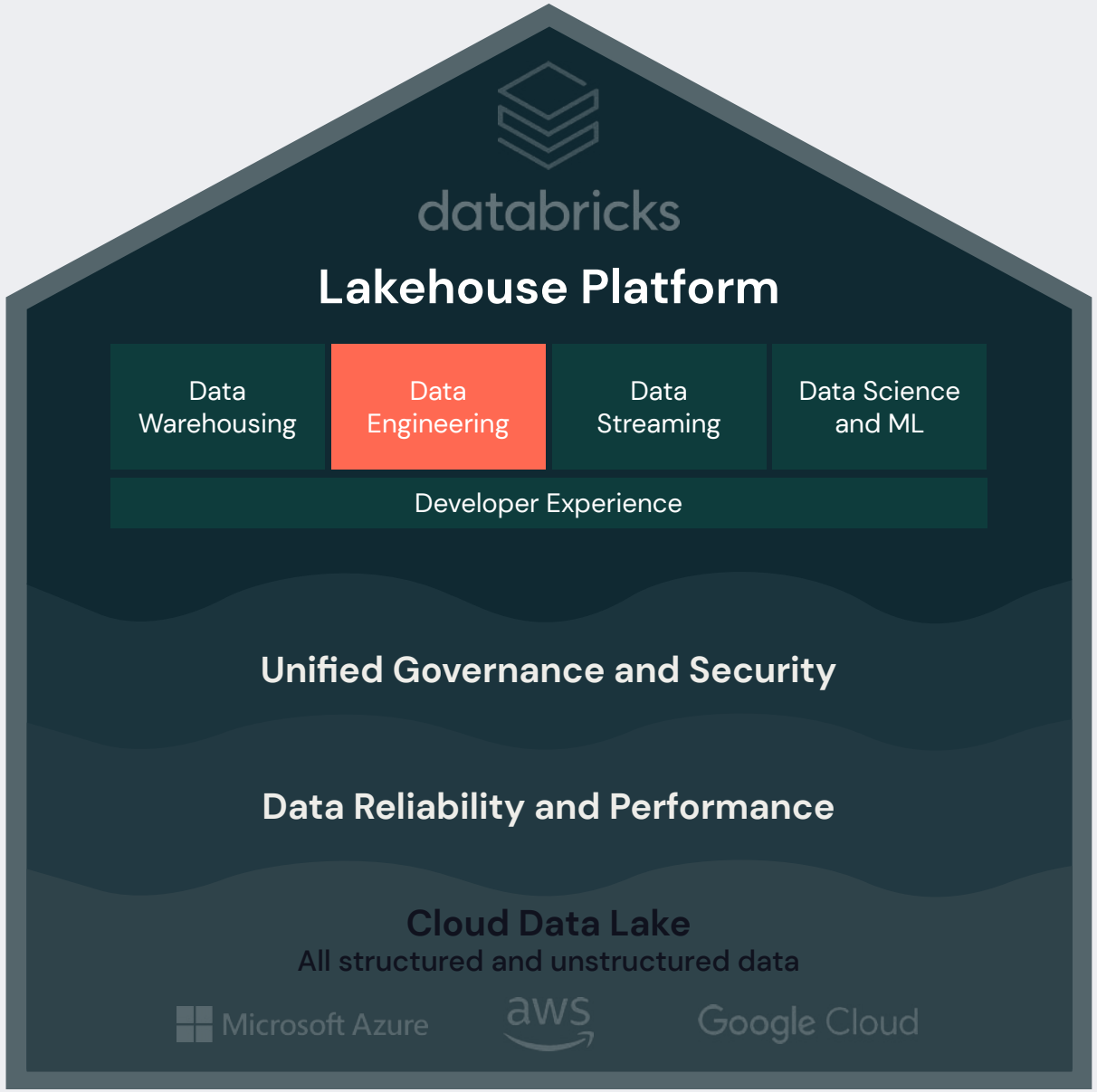
67%

of enterprise data goes unused for analytics and decision making

- Data engineers must spend immense time hand-coding data ingestion & transformations into the data lake
- Building and maintaining scalable infrastructure & performance tuning as the data platform keeps changing
- Increasing volumes and the need for up to date data requires low latency pipelines which are difficult to build and maintain

https://www.frontier-enterprise.com/two-thirds-of-data-available-to-firms-goes-unused/

How can Databricks help?



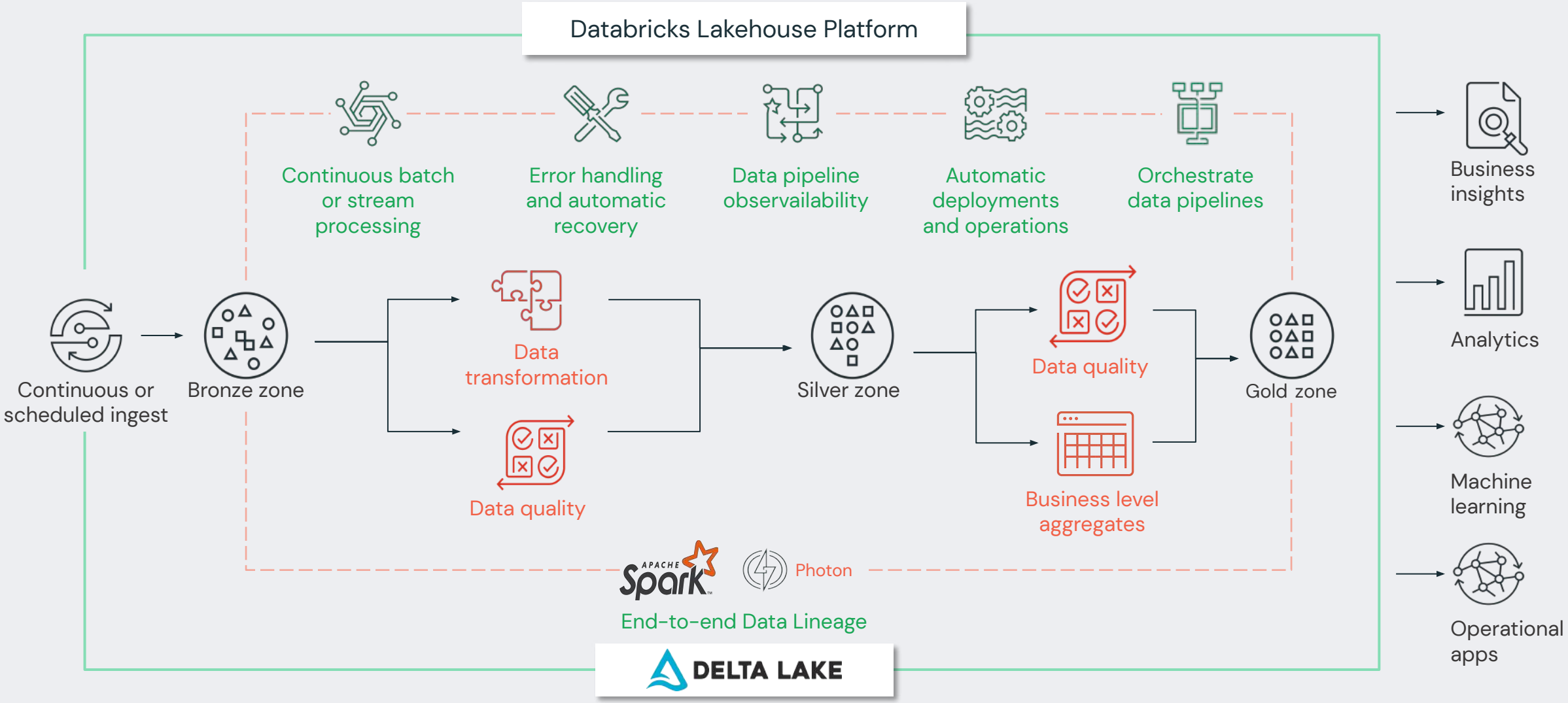
Data engineering

Ingest

Transform

Orchestrate

Modern data engineering in the lakehouse

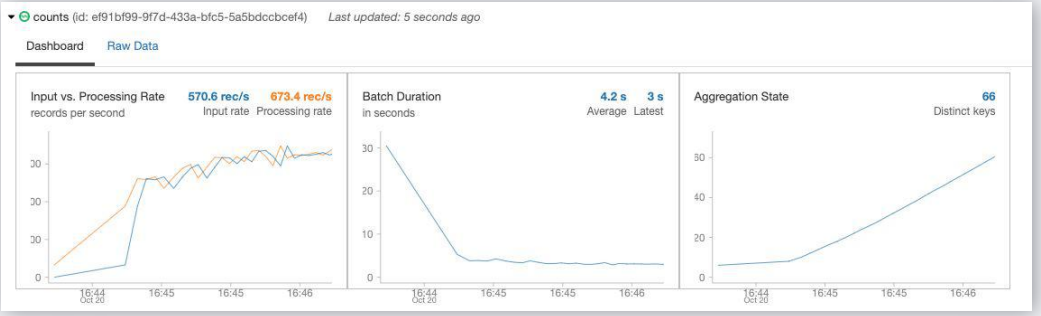


Continuous or scheduled data ingestion

Simple SQL syntax for streaming ingestion

Cmd 2

```
CREATE STREAMING LIVE TABLE sales_orders_raw
COMMENT "The raw sales orders, ingested from /databricks-datasets."
TBLPROPERTIES ("myCompanyPipeline.quality" = "bronze")
AS
SELECT * FROM cloud_files
("/databricks-datasets/retail-org/sales_orders/",
"json", map("cloudFiles.inferColumnTypes", "true"))
```



- Incrementally and efficiently process new data files as they arrive in cloud storage using AutoLoader
- Automatically infer schema of incoming files or superimpose what you know with Schema Hints
- Automatic schema evolution
- Rescue data column—never lose data again

Schema Evolution

- ✓ JSON
- ✓ CSV
- ✓ AVRO
- COMING SOON: PARQUET

Delta Live Tables

Reliable data pipelines made dead simple

```
CREATE STREAMING LIVE TABLE raw_data
AS SELECT *
FROM cloud_files ("/raw_data", "json")
```

```
CREATE LIVE TABLE clean_data
AS SELECT ...
FROM LIVE .raw_data
```

Accelerate ETL development

Declare **SQL or Python** and DLT automatically orchestrates the DAG, handles retries, changing data

Automatically manage your infrastructure

Automates complex tedious activities like **recovery, auto-scaling, and performance optimization**

Ensure high data quality

Deliver reliable data with built-in **quality controls, testing, monitoring, and enforcement**

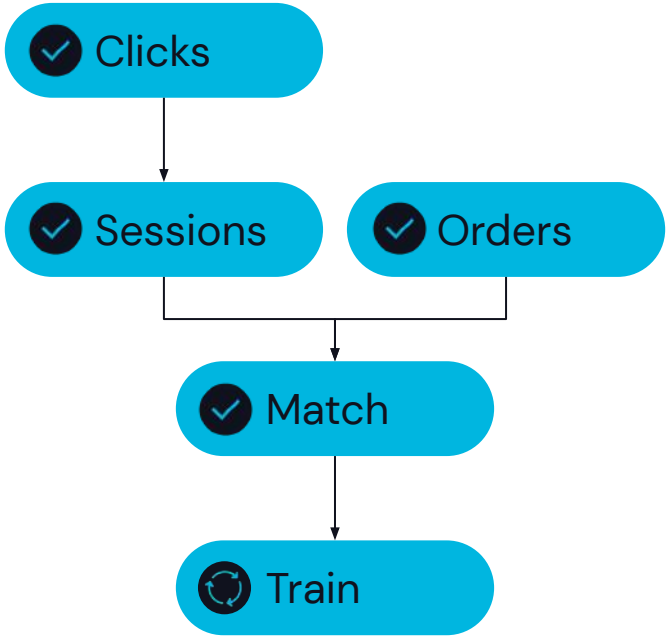
Unify batch and streaming

Get the simplicity of SQL with freshness of streaming with one **unified API**

Databricks Workflows

Reliable orchestration for data, analytics and AI

Orchestrate...



...any task...

Delta Live Tables

Koalas

Auto Loader

Java

Apache Spark

mlflow

SQL

non-Spark

dbt

+more

...across any platform

aws

Google Cloud

talend

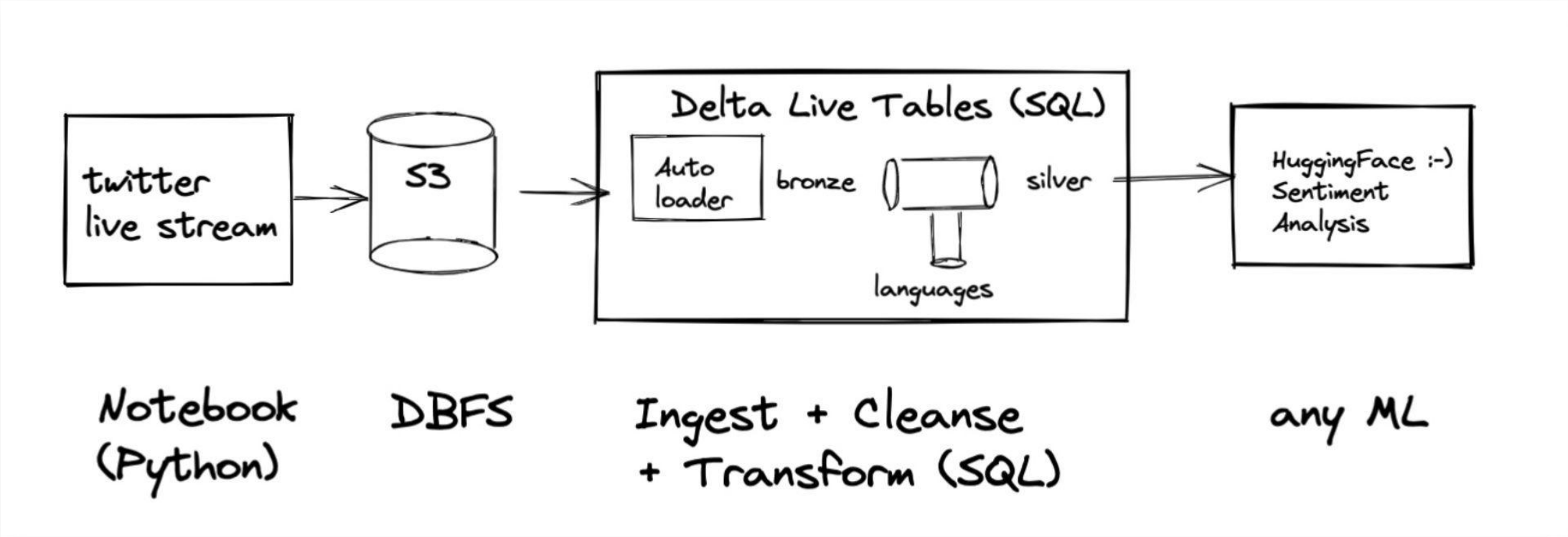
Fivetran

+more

Demo:
Data Engineering with Databricks

Delta Live Tables

Cleanse and Transform Tweets



https://github.com/databricks/delta-live-tables-notebooks

README.md

The Twitter-Stream-S3.py notebook uses Tweepy

I use Tweepy to ingest a live Twitter stream based on search criteria that can be defined, such as "DLT" and "data engineering". The ingested Twitter data is streamed to an S3 bucket. Just imagine this S3 bucket as your data lake. With Databricks, I can use DBFS to abstract the cloud object store as a folder (DBFS is multicloud, it will work the same on ADFS2 and GCS too)

```
# Filter realtime Tweets by keyword and language
try:
    tweet_stream.filter(languages=["en","de","es"], track=["DLT","Delta Live Tables"])
```

The Twitter-Dataflow.sql notebook uses Delta Live Tables in SQL with Autoloader

What matters most in "DLT" is the "P" :-). "P" as in "pipeline" or dataflow. In this example DLT is used together with Databricks Autoloader. Autoloader ingests streaming data into the lakehouse and detects the schema. My DLT pipeline follows the Medallion Architecture and creates a Bronze table for the raw data, then filters the 40 columns per tweet and cleanses the data to ensure only tweets in English are contained in the Silver table. Ensuring data quality is done with SQL constraints (we like to call them Expectations in DLT lingo).

The Twitter-SentimentAnalysis.py Notebook uses Hugging Face Sentiment Analysis Pipelines

For sentiment analysis, I picked Hugging Face (the Databricks platform is open and flexible, so any ML will work). It doesn't get much easier than using a pretrained Hugging Face language model that is even optimized for tweets. The model detects emojis like :-), 😀, 😬 and so on. My goal was to show how almost any kind of ML can be used within the Lakehouse - with emphasis on simplicity.

SCAN ME

RIVIAN

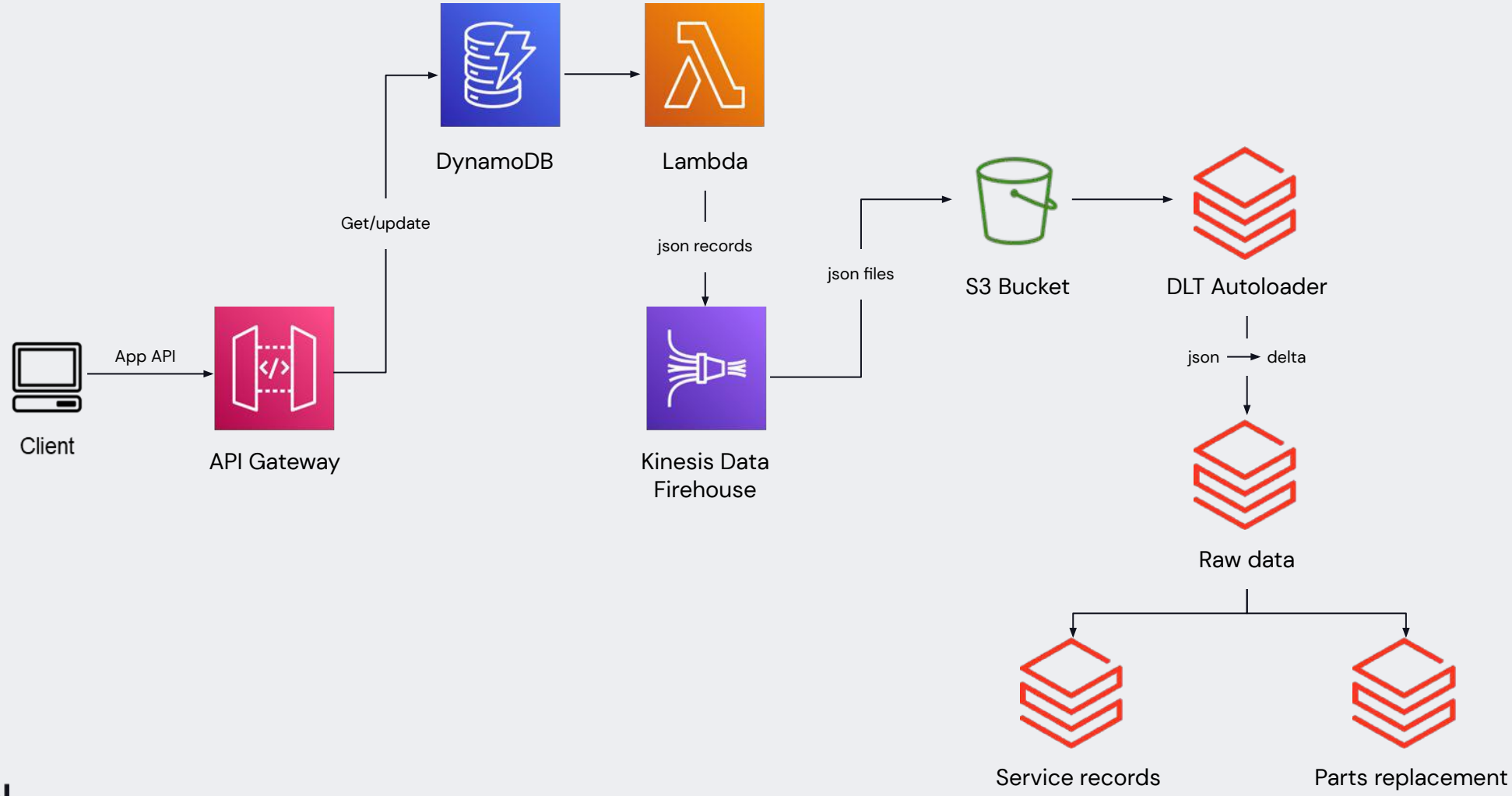
Customer Story

RIVIAN

Service health and vehicle reliability → **Delta Live Tables** to ingest and analyze data from car service stations. Use this data to get Insights into issue types, what parts are being replaced, regulatory reporting, and part replacement forecasting.

Service Data Architecture

Making real-time data available to query in delta lake



Building our ETL pipeline before DLT

```
def service_catalog_json_raw(): #the name of the function becomes the target table's name.
  json_path = 's3a://service-records-bucket/ServiceCatalog/json/'
  schema =StructType.fromJson(jsonRawSchemaJsonValue)
  return (
    # Since this is a streaming source, this table is incremental.
    spark.readStream.format("cloudFiles")
      .option("multiLine", "true")
      .option("cloudFiles.format", "json")
      .option("cloudFiles.useIncrementalListing", "true")
      .schema(schema)
      .load(json_path)
    )

(
  service_catalog_json_raw()
  .writeStream
  .format("delta")
  .option("checkpointLocation", checkpoint_location)
  .trigger(availableNow=True)
  .start(delta_path)
)

spark.sql("CREATE DATABASE IF NOT EXISTS vehicle_services")
spark.catalog.createTable("vehicle_services.service_catalog_json_raw", delta_path)
spark.sql(f"OPTIMIZE '{delta_path}'")

delta_table = DeltaTable.forPath(spark, delta_path)
delta_table.vacuum(30)
```

Specify path to source data

Autoloader data source: json

Functional pattern compatible with DLT

Checkpoints and trigger

Create and optimize table

Building our ETL pipeline with DLT

DLT simplifies the code for ELT best practices

```
@dlt.table(
  comment="raw, list-based json data from ITBI",
  table_properties={"pipelines.reset.allowed" : "false"}
)
def service_catalog_json_raw(): #the name of the function becomes the target table's name.
  json_path = 's3a://service-records-bucket/ServiceCatalog/json/'
  schema =StructType.fromJson(jsonRawSchemaJsonValue)
  return (
    # Since this is a streaming source, this table is incremental.
    spark.readStream.format("cloudFiles")
      .option("multiLine", "true")
      .option("cloudFiles.format", "json")
      .option("cloudFiles.useIncrementalListing", "true")
      .schema(schema)
      .load(json_path)
    )
```

DLT: Define table, properties, and ETL dependencies

Comments enable catalog searchability

Functional pattern defines the materialized table

Standard Apache Spark™ code

Code → graph

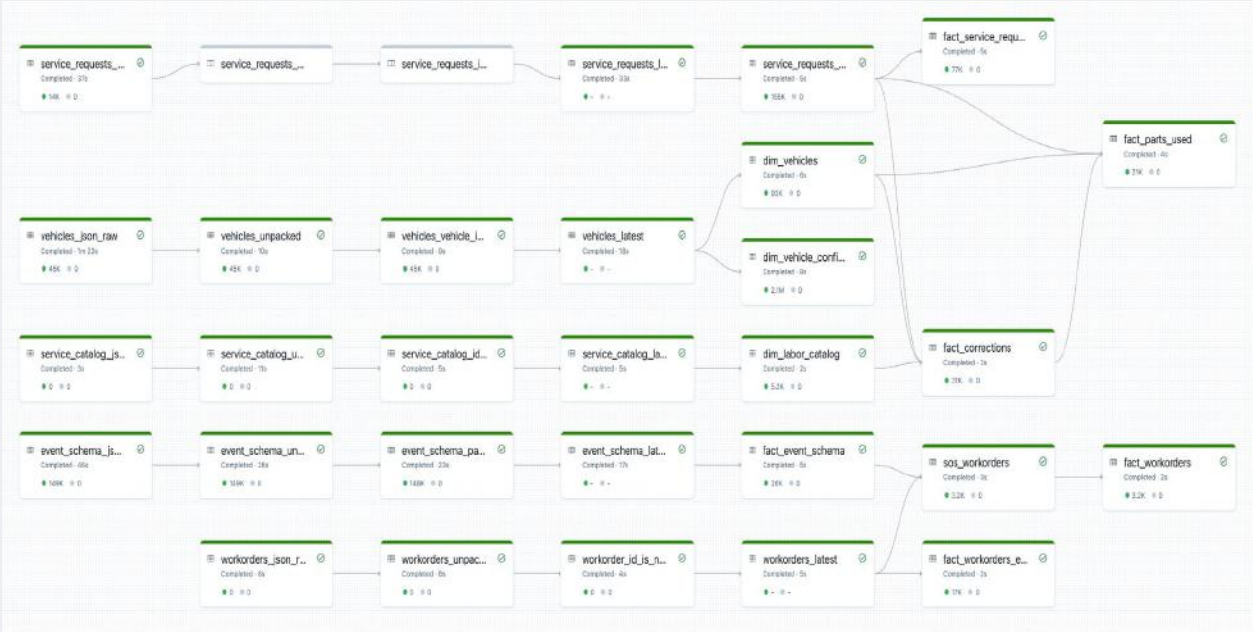
Code first integrates with a modern software stack

- Graphical DAG is a property of explicit relationships defined in the code

```
dlt.read_stream("source_table_or_view")
```

- Code is checked into vcs[git]
- Pipeline is provisioned with CICD/Terraform

```
resource "databricks_pipeline" "this" {}
```



Summary

Data scientist and analysts are empowered to create high quality data pipelines with DLT

- Reduced architectural overhead with automatic best practices
  - Table storage path/format
  - Checkpoint path
  - Create database/create table
  - Optimize/vacuum
  - Continuous and `Trigger.AvailableNow`
- Data validation
- Full refresh
- ETL dependencies

“It's so intuitive that even somebody with only moderate Python skills can create efficient, powerful data pipelines with relative ease”

Tom Renish
Principal Data Architect

“Before DLT, we had an ETL job that would do a full table scan of dynamoDB each day. With DLT, we were able to move to a continuous pipeline that reads a dynamoDB feed from S3 and ingest into a streaming live table, allowing us to go from 24 hours to near real-time data freshness for a fraction of the cost.”

Jason Shiverick
Principal Data Scientist

Delta Live Tables

Modern software engineering for ETL processing

Accelerate ETL development
Apply modern software engineering best practices to deploy pipelines at scale

Automatically manage your infrastructure
Remove overhead by automating complex and time-consuming activities like task orchestration, error handling and recovery, auto-scaling, and performance optimization.

Have confidence in your data
Deliver reliable data with built-in quality controls, testing, monitoring and enforcement to ensure accurate and useful BI, Data Science, and ML

Simplify batch and streaming
Provide the freshest/up-to-date data for your apps with data self-optimized and **auto-scaling** data pipelines for batch or streaming processing and choose optimal cost-performance

Customers Save Time with Delta Live Tables

1.3 trillion rows of sensor data processed efficiently

Manage ingest of highly dynamic data from across LoBs

Saved immense data management time and effort

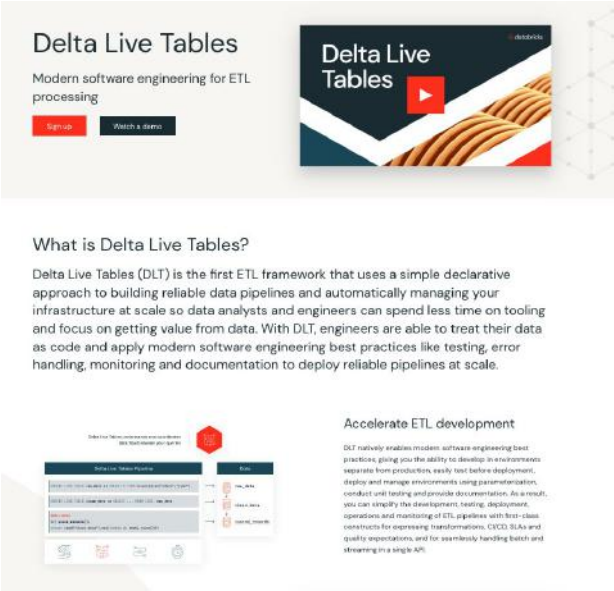
Enabled data analysts to build their data pipelines with SQL

86% reduction in time to production

Enabled the NextGen self-service data quality platform

Databricks Data Engineering

Get started with Data Engineering



https://databricks.com/product/delta-live-tables

Watch our upcoming session!

Get started with Data Engineering

DATA+AI SUMMIT 2022

Breakout Session:

DLT Deep Dive: June 29 @11:30 AM | Room 205

Orchestration Made Easy with Databricks Workflows @2:05 PM | Room 306

Streaming with Databricks @2:50 PM | Room 306

DATA+AI
SUMMIT 2022

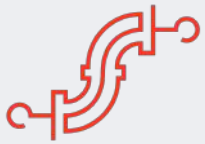
Thank you

Paul Lappas
Product Manager, Databricks

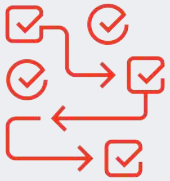
What is Delta Live Tables?

Modern software engineering for ETL processing

Delta Live Tables (DLT) is the first ETL framework that uses a simple, declarative approach to building reliable data pipelines. DLT automatically manages your infrastructure at scale so data analysts and engineers can spend less time on tooling and focus on getting value from data.



Accelerate ETL Development



Automatically manage your infrastructure

Have confidence in your data

Simplify batch and streaming

What is Auto Loader?

Optimized data ingestion for ETL

Auto Loader incrementally and efficiently ingests new data files as they arrive in cloud storage on any cloud without any additional setup.

Autoloader can be used in a DLT pipeline.

Automatic Schema Inference

Steaming or Batch mode

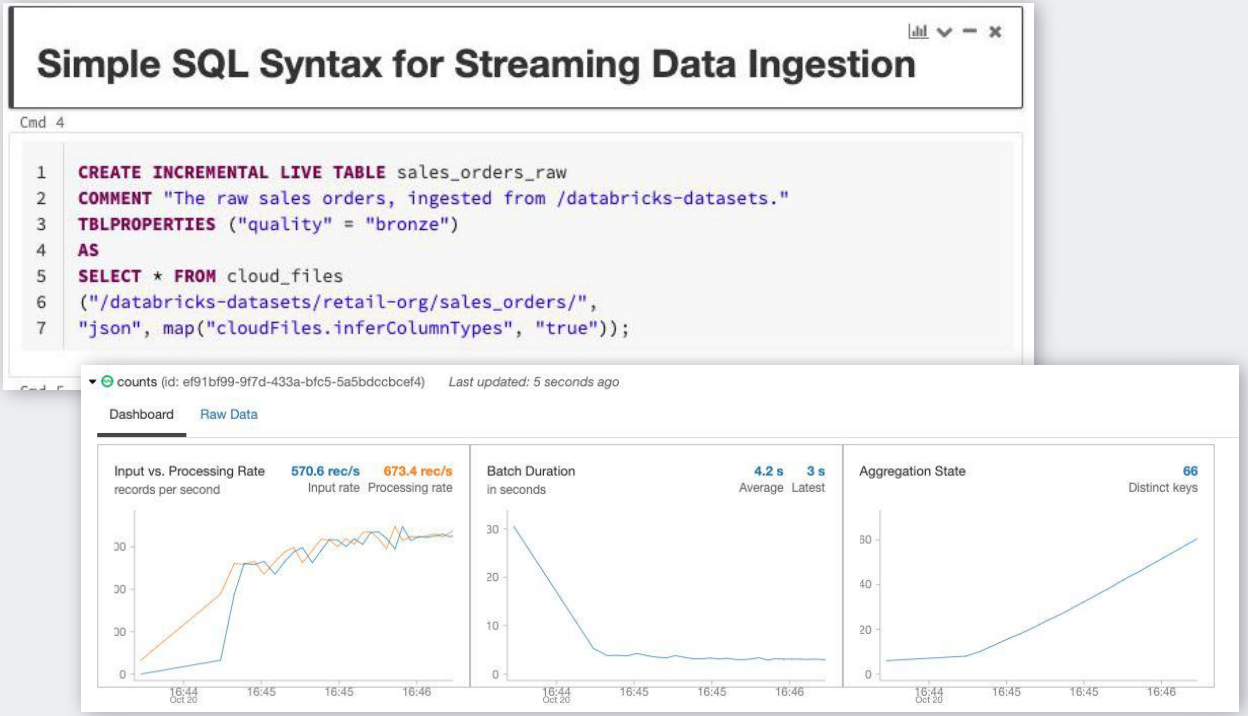
Incremental Ingest

Multicloud, optimized File List

Delta Live Tables

Key Differentiators

Continuous or scheduled data ingestion



- Incrementally and efficiently process new data files as they arrive in cloud storage using Auto Loader
- Automatically infer schema of incoming files or superimpose what you know with Schema Hints
- Automatic schema evolution
- Rescue data column – never lose data again

Schema Evolution

✅ JSON ✅ CSV ✅ AVRO

Coming Soon

PARQUET

Declarative SQL & Python APIs

Source

```
/* Create a temp view on the accounts table */
CREATE STREAMING LIVE VIEW account_raw AS
SELECT * FROM cloud_files(“/data”, “csv”);
```

Bronze

```
/* Stage 1: Bronze Table drop invalid rows */
CREATE STREAMING LIVE TABLE account_bronze AS
COMMENT "Bronze table with valid account ids"
SELECT * FROM fire_account_raw ...
```

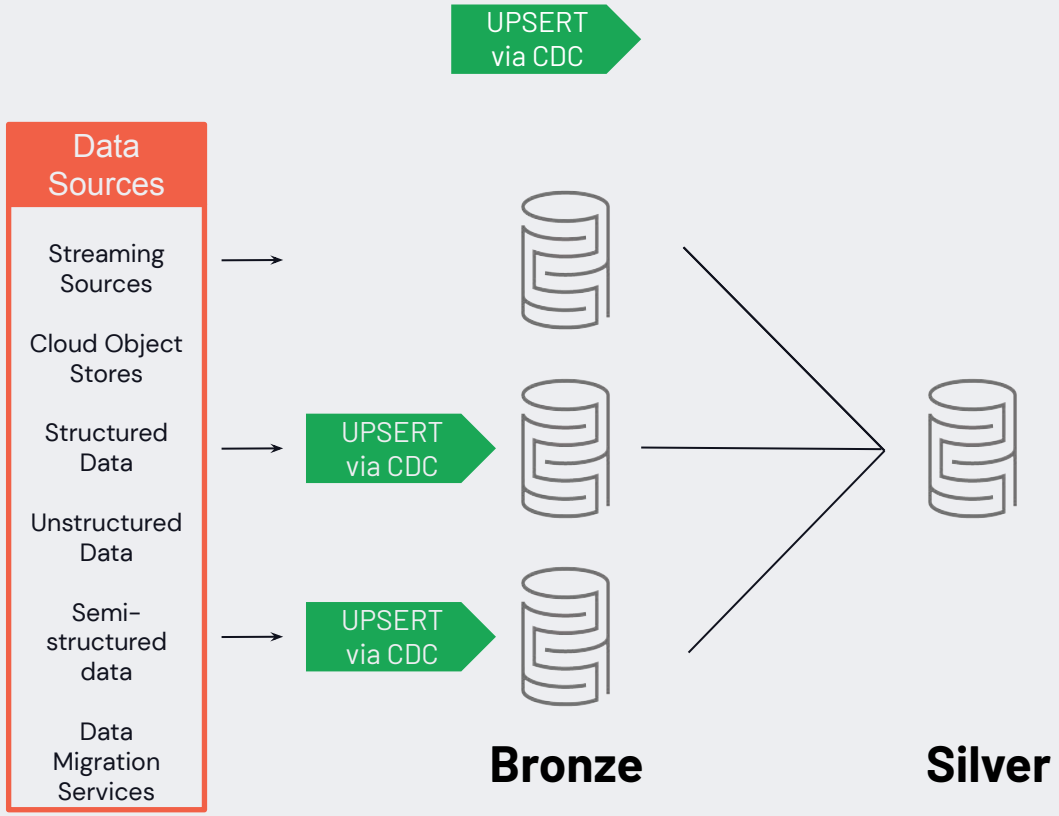
Silver

```
/* Stage 2:Send rows to Silver, run validation rules */
CREATE STREAMING LIVE TABLE account_silver AS
COMMENT "Silver Accounts table with validation checks"
SELECT * FROM fire_account_bronze ...
```

Gold

- Use intent-driven declarative development to abstract away the **“how”** and define **“what”** to solve
- Automatically generate **lineage** based on table dependencies across the data pipeline
- Automatically checks for errors, missing dependencies and syntax errors

Change data capture (CDC)

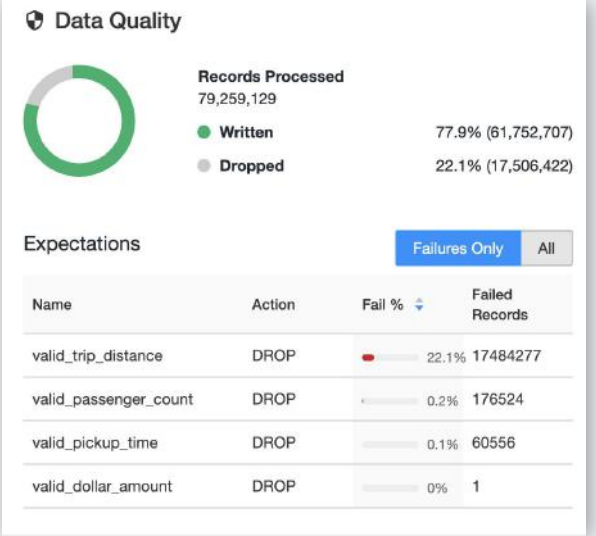


- Stream change records (inserts, updates, deletes) from any data source supported by DBR, cloud storage, or DBFS
- Simple, declarative "APPLY CHANGES INTO" API for SQL or Python
- Handles out-of-order events
- Schema evolution
- SCD2 support

Data quality validation and monitoring

- Define data quality and integrity controls within the pipeline with data expectations
- Address data quality errors with flexible policies: fail, drop, alert, quarantine(future)
- All data pipeline runs and quality metrics are captured, tracked and reported

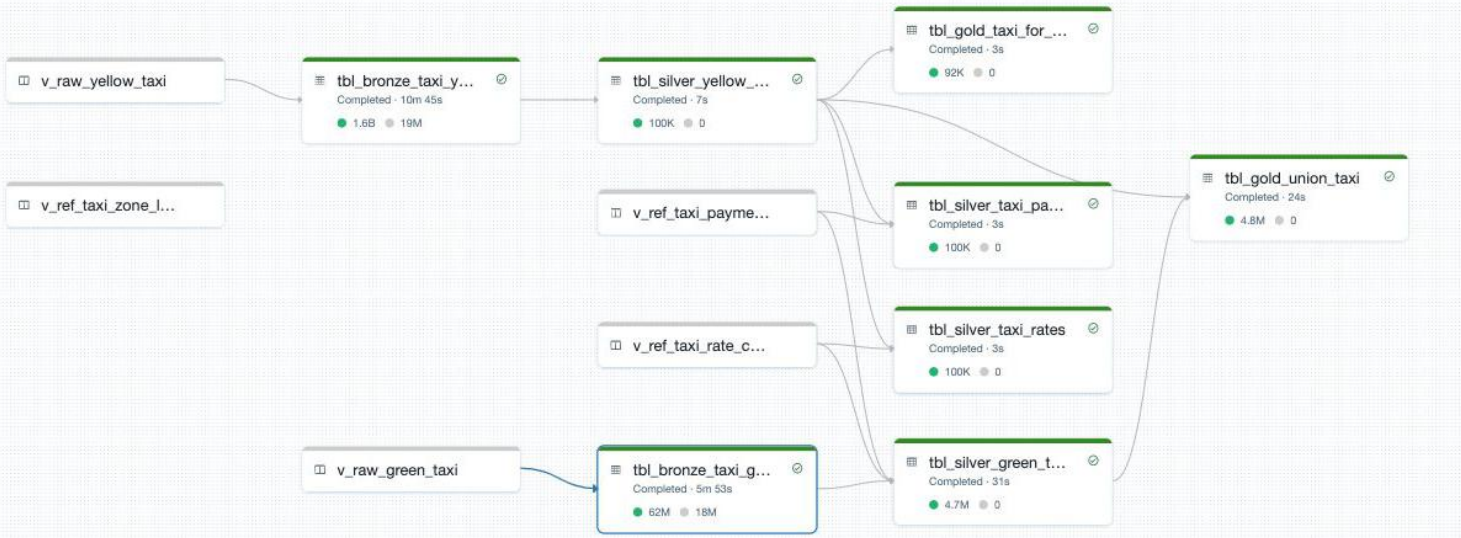
```
/* Stage 1: Bronze Table drop invalid rows */
CREATE STREAMING LIVE TABLE fire_account_bronze AS
( CONSTRAINT valid_account_open_dt EXPECT (acconut_dt is not
null and (account_close_dt > account_open_dt)) ON VIOLATION DROP
ROW
COMMENT "Bronze table with valid account ids"
SELECT * FROM fire_account_raw ...
```



Delta Live Tables SQL Pipeline

Development | Production | Delete | Permissions | Settings | Schedule | Start

2/25/2022, 9:25:02 AM · Completed



total_amount: double
payment_type: integer
pickup_hour: integer
pickup_day_of_week: integer
pickup_month: integer
pickup_year: integer
pickup_minute: integer
pickup_seconds: decimal(8,6)
dropoff_hour: integer
dropoff_day_of_week: integer
dropoff_month: integer
dropoff_year: integer
dropoff_minute: integer
dropoff_seconds: decimal(8,6)

Data Quality

Written	77.9% (61,752,707)
Dropped	22.1% (17,506,422)

Expectations

All | Failures Only

Name	Action	Fail %	Failed Records
valid_trip_distance	DROP	22.1%	17484277
valid_passenger_count	DROP	0.2%	176524

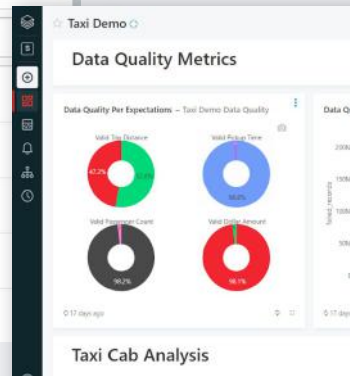
All | Info | Warn | Error | Filter...

4 minutes ago	flow_progress	Flow 'tbl_gold_taxi_for_analysis' has COMPLETED.
4 minutes ago	flow_progress	Flow 'tbl_silver_taxi_payments' has COMPLETED.
4 minutes ago	flow_progress	Flow 'tbl_silver_taxi_rates' has COMPLETED.
4 minutes ago	flow_progress	Flow 'tbl_gold_union_taxi' has COMPLETED.
4 minutes ago	update_progress	Update 1d1ad5 is COMPLETED.

iagram
a flows

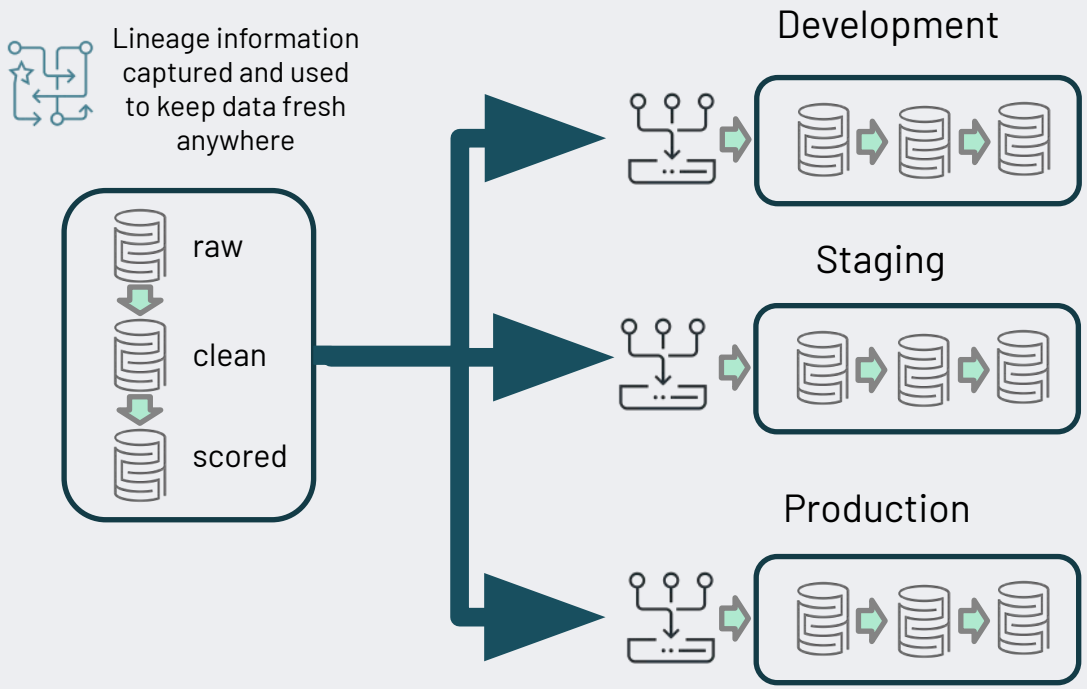
vernance,
line at a

jobs to

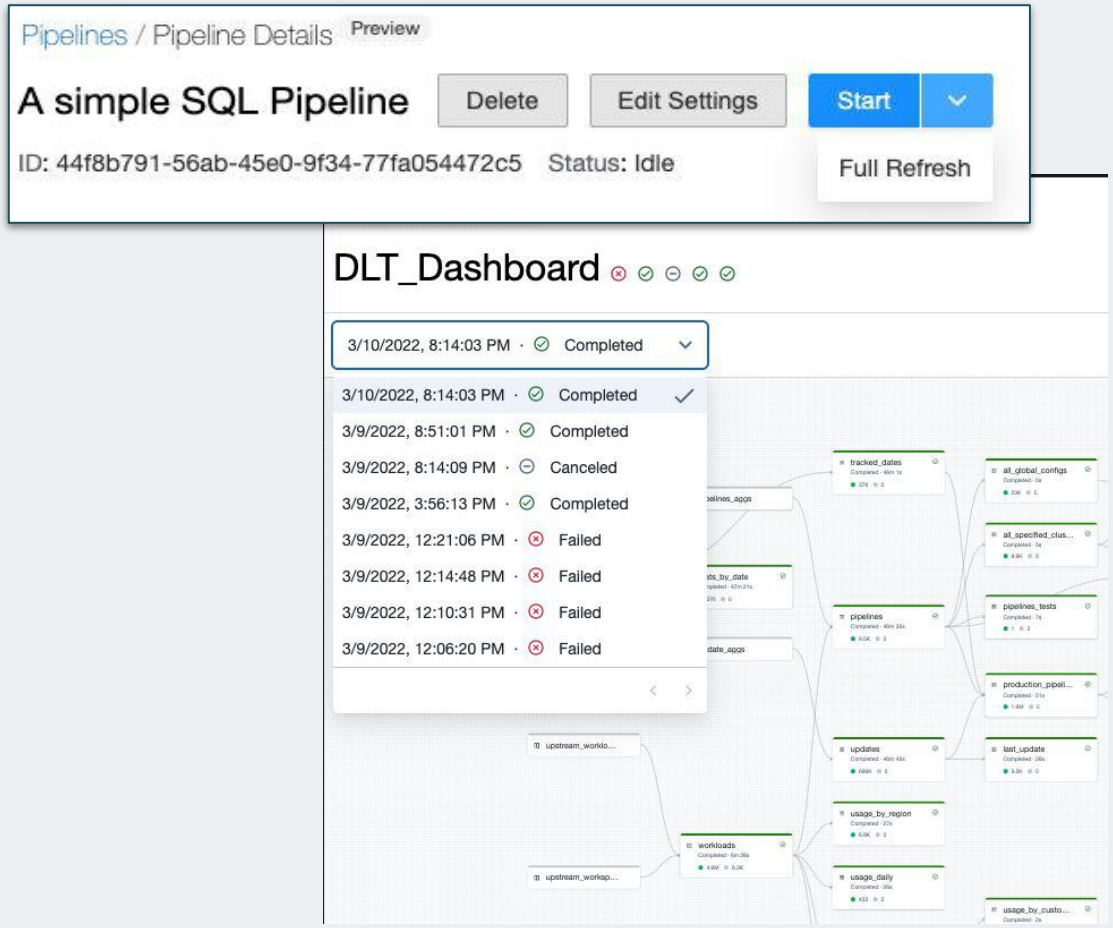


Automated ETL development lifecycle

- Develop in **environment**(s) separate from **production** with the ability to easily test it before deploying - entirely in SQL
- Deploy and manage environments using **parameterization**
- **Unit testing** and **documentation**
- Enables **metadata-driven** ability to programatically scale to 100s of tables/pipelines dynamically



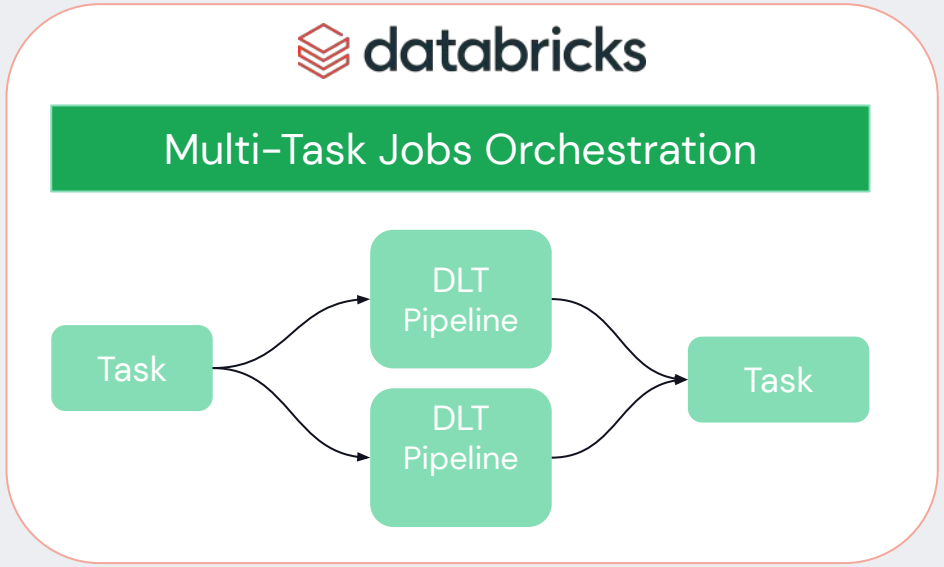
Automated ETL operations



- Reduce down time with automatic error handling and easy replay
- Eliminate maintenance with automatic optimizations of all Delta Live Tables
- Auto-scaling adds more resources automatically when needed.

Workflow Orchestration

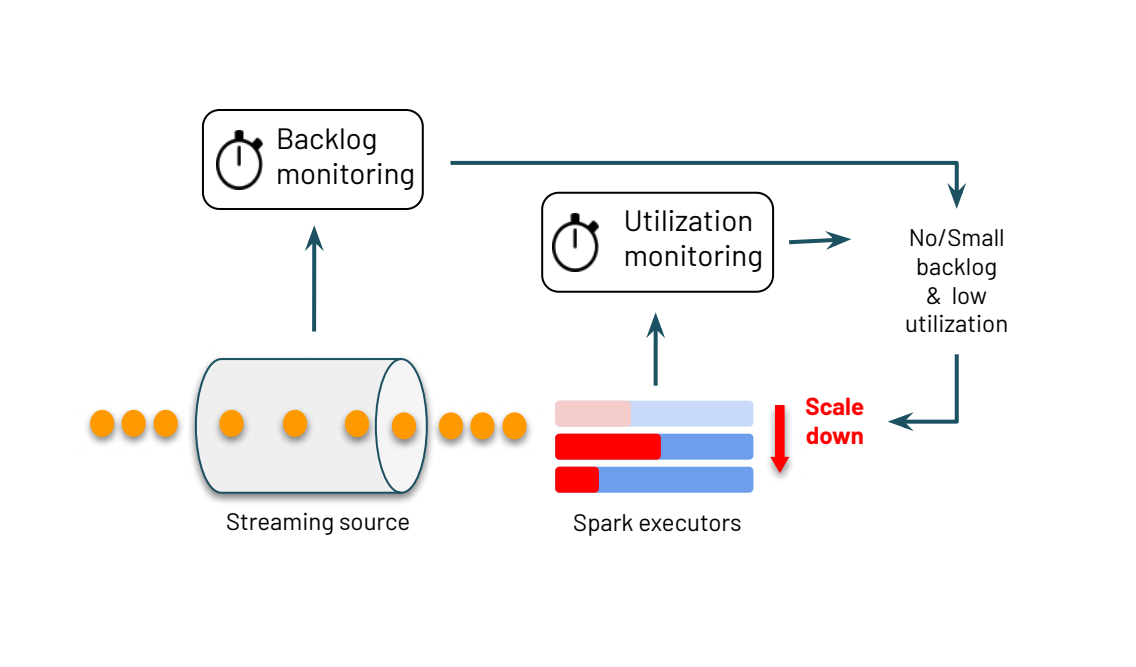
Simplify orchestration and management of data pipelines



- Easily **orchestrate** DLT Pipelines and tasks in the same DAG
- **Fully integrated** in Databricks platform, making inspecting results, debugging faster
- Orchestrate and manage workloads in **multi-cloud environments**
- You can run a Delta Live Tables pipeline as part of a data processing workflow with **Databricks jobs, Apache Airflow, or Azure Data Factory.**

In Preview

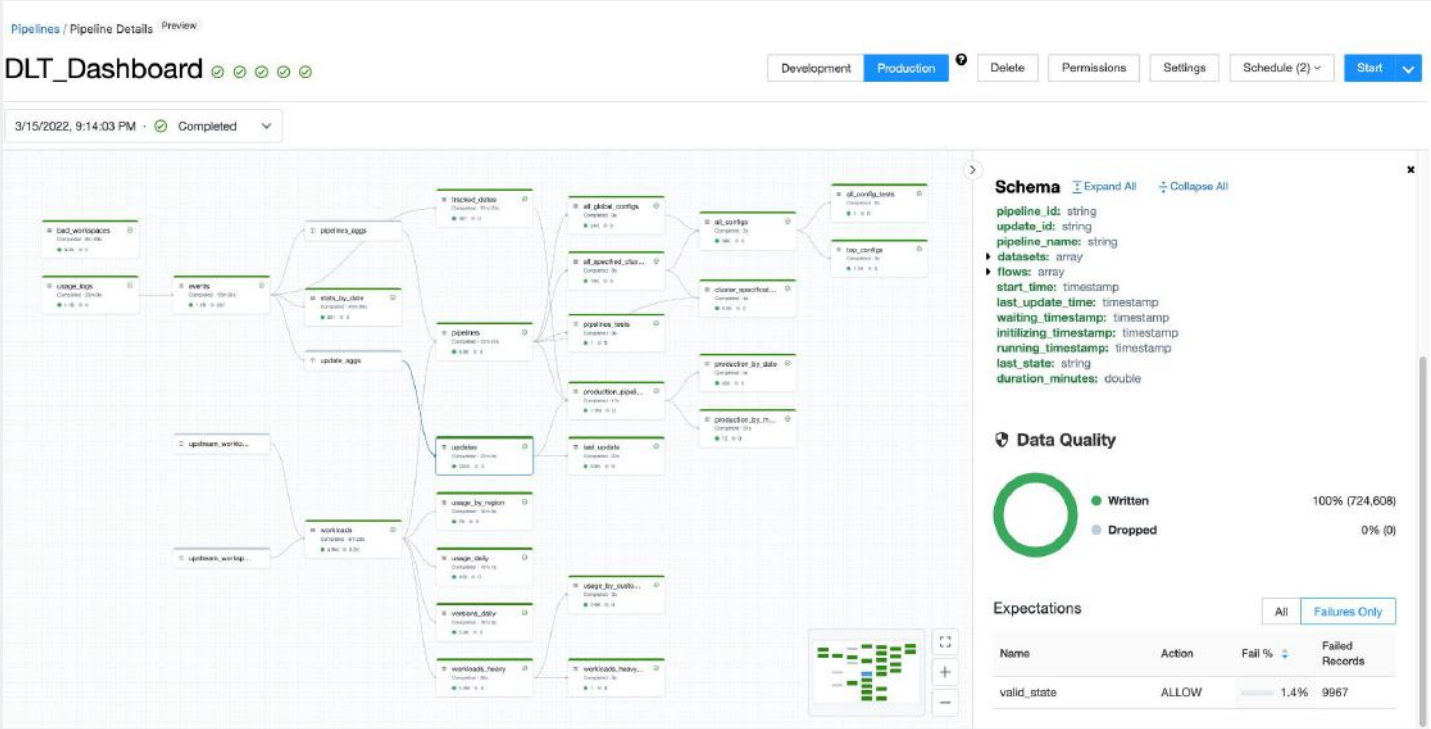
Auto Scale Streaming Workloads (preview)



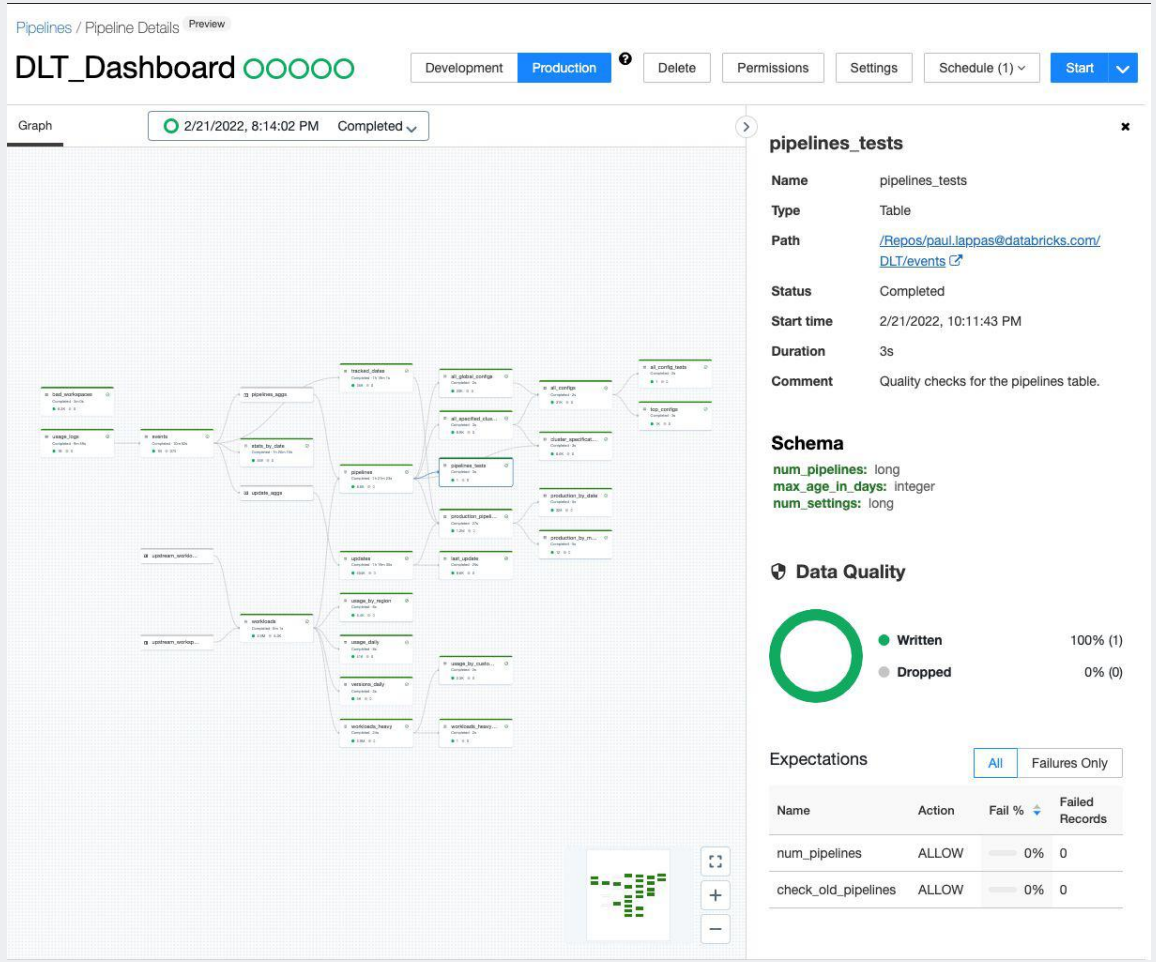
- Built to handle streaming workloads which are spiky and unpredictable
- More efficient cluster utilization. Shuts down nodes when utilization is low while guaranteeing task execution.
- Only scales up to needed # of nodes (even if it's less than the max)

DLT Recent UI Improvements

- Scheduling button
- UI for settings management
- Event history
- Updated DAG visual
- DLT cluster policy
- ACLs for Run perms



Upcoming Features



Data Sample Preview

Preview tables from within the DLT UI, and link to Databricks SQL Query Editor

Selective Refresh of Tables

Re-run individual table(s) for easier development and error recovery

Unity Catalog Integration

Read and Publish from/to UC Managed Storage. 3-level notation support

Coming in 2022

Materialized Views for Databricks SQL

Powered by DLT – Speed up queries with pre-computed results

Main Benefits

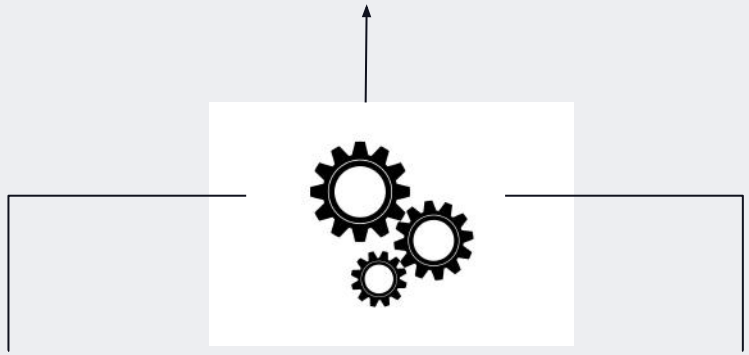
- Accelerate end-user queries
- Reduce infrastructure costs with efficient, incremental computation

Use Cases

- Accelerate BI **dashboards** and ETL queries
- **Streaming**: build MVs on top of streaming live tables
- Easy **ELT**: Simplify reporting by cleaning, enriching, denormalizing the base tables
- **Data Sharing & Access Control:** control what info can be seen by internal and external users and organizations.

sales_report

city	total_rev
SF	24
NY	7



sales

prod	loc	txn	price
i1	l1	tx1	11
i2	l2	tx2	24
i3	l3	tx3	7

store_info

loc	mgr	city
l1	Alice	WA
l2	Bob	SF
l3	Anne	NY

"Delta Live Tables powers the data quality rules engine and metrics repository for our global self-service technology platform. With DLT, we can enable data engineers to easily add data quality checks, and enables end-users with trust and confidence in the information they are using to make decisions. ***- Abhay Prajapati, Principal Data Solutions Architect - JLLT***

Use Case & Challenge

- Self-Service Data Quality Platform
- Wants to enable end-users to believe in and trust the information they are using to power their tools, applications and decision-making.

Why Databricks + DLT?

Impact of DLT