Delta Live Tables:

Delta Live Tables (DLT) is an innovative declarative ETL framework designed to streamline ETL processes across both batch and streaming data flows in a cost-effective and efficient manner. By leveraging DLT, organizations can significantly simplify the orchestration of tasks, management of clusters, monitoring of data quality (DQ), and error handling, thanks to its automated pipeline management capabilities. This powerful framework is engineered to facilitate seamless, high-performance data transformation and integration tasks, ensuring that data processing workflows are not only reliable but also optimized for operational efficiency and cost savings.

With latest improvements DLT is also supported in DBSQL [addition to python]

Below are the DLT Use Cases:

1. Ingestion
2. Stream processing/ micro batches
3. Error handling & auto recovery
4. Data quality monitoring
5. Data pipeline orchestration

A diagram of data processing

Description automatically generated

Key benefits of implementing DLT,

* Accelerate ETL development with many common use cases handled as features, like handling retries on failures, change data feed, orchestration of sequential jobs
* Simplified auto loader for ingestion with multiple inbuilt features [infer schema, evolve schema, watermark tracking for change data]
* Built in Data quality checks and controls on how to handle the failed records. This also enforces constraints on table data
* Simplify the stream processing
* Auto manage the infrastructure for recovery, auto-scaling etc.,

DLT data assets:

Three data assets of DLT,

* **Streaming Table**: record by record processing pipeline (earlier referred as Streaming live table). Suitable use cases of Data ingestion from sources to raw layer where data is append only. Can track new incoming data to keep raw layer up to date.
* **Materialized Views**: Views that are refreshed on schedule to run the transformation of data (that can be aggregation, updates, or simple CDC process like SCD2). Does all that without having complexity of scheduled pipelines, handle different scenarios without writing extra code.
* **Views**: More like temporary iew which are not visible for catalog but for the pipelines only.

DLT Pipeline:

A workflow that contains materialized views and streaming tables. DLT automatically infers the correct order of data load/ process. There are 2 steps of creating DLT pipeline,

1. Create DLT code in notebooks either in SQL or python to declare the DLT datasets
2. Configuration of pipeline resources, dependencies, how updates are processed & how tables are stored.

Pipeline Update:

DLT pipeline run/ pipeline update deploys infrastructure & recompute the data state when a pipeline update started.

Process Data

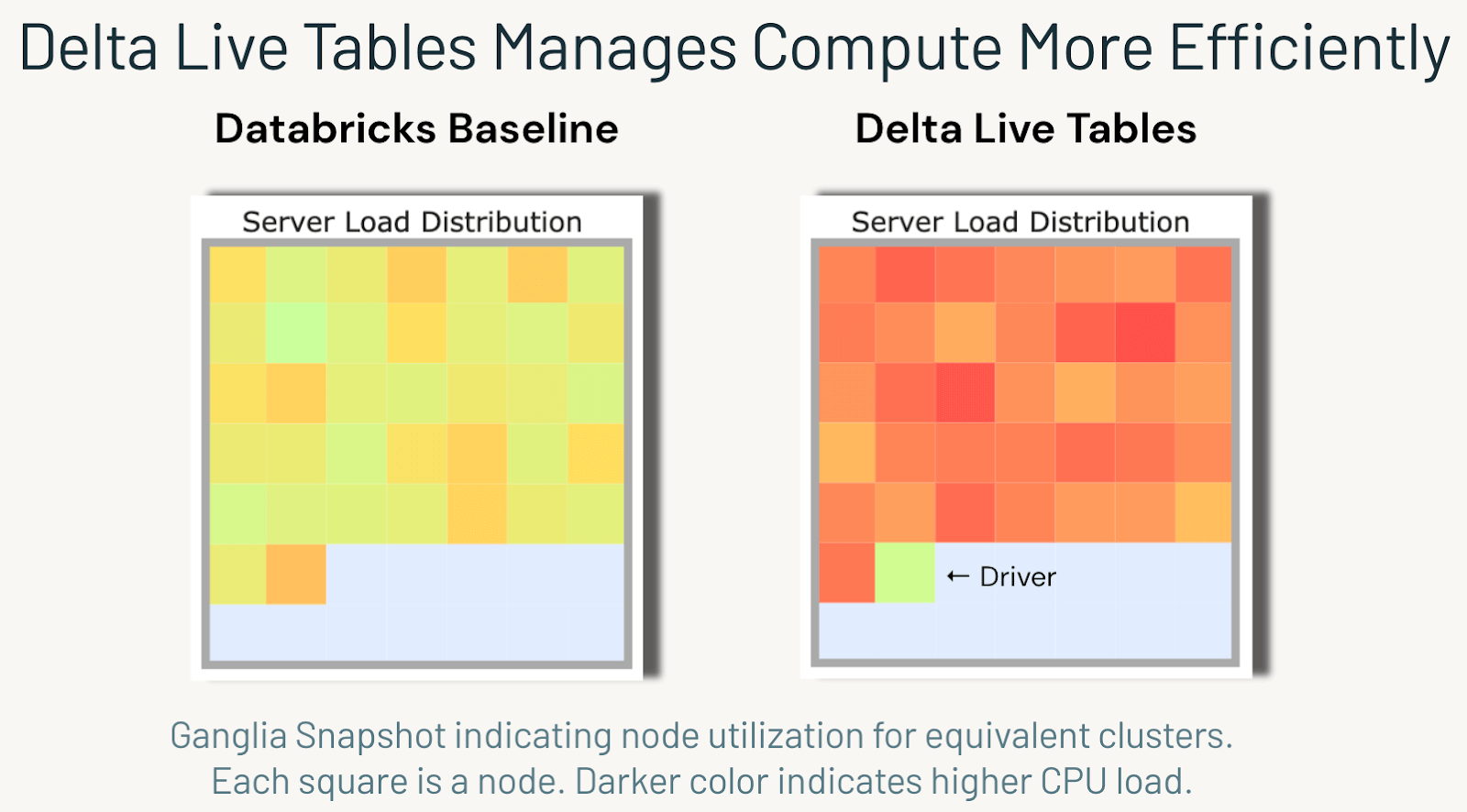
Run checks

[syntax, errors dependencies]

Provision cluster

DLT pipelines proven to be more efficient way of handling ETL for its performance & optimal compute usage. Verified as 2x faster on DLT compared to the non-DLT Databricks baseline, because DLT is more optimal in orchestrating tasks than manual setup [TPC-DI benchmark test].

DLT automatically determines all table dependencies and manages them on its own. When implemented without DLT, we had to create the complex DAG from scratch in our orchestrator to ensure each ETL step commits in the proper order. Automatic orchestration also significantly improves resource management, ensuring work is parallelized flawlessly across the cluster. This efficiency is primarily responsible for the 2x speedup.



Data Ingestion:

Auto loader still make good option to ingest data from cloud storage which can run in both batch & stream. But with auto loader in DLT can ingest data from cloud storage and also from Kafka stream and process data down the line as seamless pipeline. With multiple options to customize on refresh mechanism DLT can read & update data continuously. Also best suited for incremental loads with additional features supported

Data processing:

For data processing in ETL framework using DLT read\_stream make ideal option to read form Delta table and process it as soon as the data refresh in source.

So in a medallion architecture,

Use DLT auto loader for SOURCE -> Bronze

And use read\_stream for Bronze -> Silver & Silver -> Gold

**Data Quality:**

DLT automatically provides real-time data quality metrics to accelerate debugging and improve the downstream confidence in the data.

**Sample DQ SQL:**

CREATE OR REFRESH LIVE TABLE FactWatches (

${Factwatchesschemal}

CONSTRAINT valid\_symbol EXPECT (sk\_securityid IS NOT NULL),

CONSTRAINT valid\_customer\_id EXPECT (sk\_customerid IS NOT NULL))

AS SELECT

c.sk\_customerid sk\_customerid,

s.sk\_securityid sk\_securityid,

sk\_dateid\_dateplaced,

sk\_dateid\_dateremoved,

fw.batchid

FROM LIVE.FactWatchesTemp fw

**DQ Stats checked:**

