Nebula: The Journey of Scaling Instacart's Ads Data Pipelines with Spark and Lakehouse

Devlina Das, Engineer @ Instacart
Arthur Li, Engineer @ Instacart

instacart

June, 2023

Agenda

- Introduction
- Growth Challenges
- Motivation to build Data Lakehouse architecture
- Improvements: Lakehouse & Spark Applications
- Transition from batch only to streaming/Incremental processing

Introduction

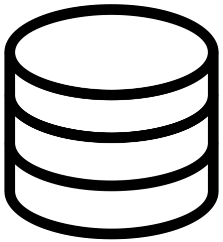
Instacart: A leading online grocery platform
Creating a world where everyone has access to the food they love and more time to enjoy it together.

Presenter:

- Arthur Li, Software Engineer, Data Platform
- Devlina Das, Software Engineer, Ads Measurement Platform

instacart

Frequent internal data users: 1,000 +



Data size: 40 PB +

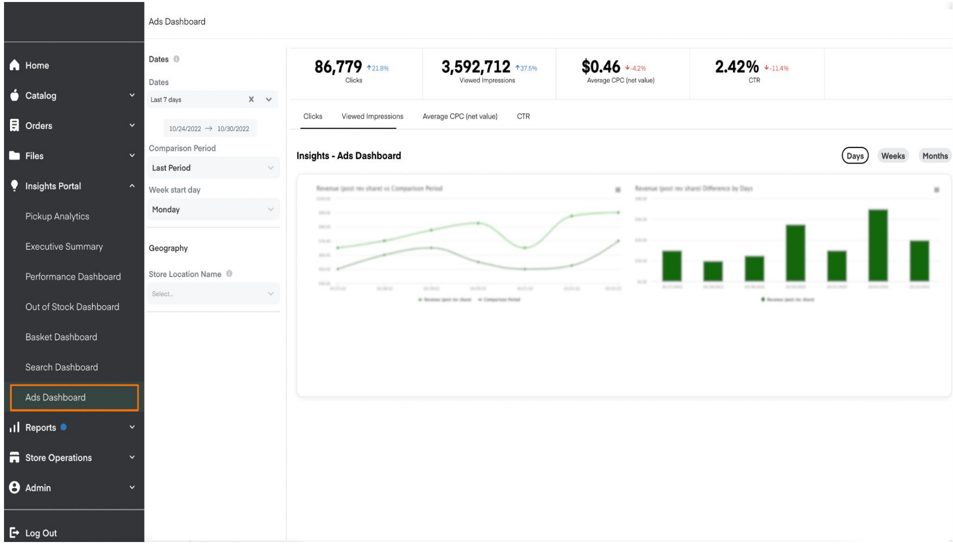
Data sources: mobile clients, retailers, internal services, vendors...

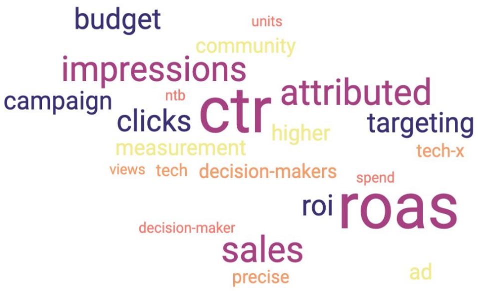
Data Teams at Instacart: Our goal is to build and reduce the friction of accessing timely and reliable data across Instacart and for our partners.

instacart

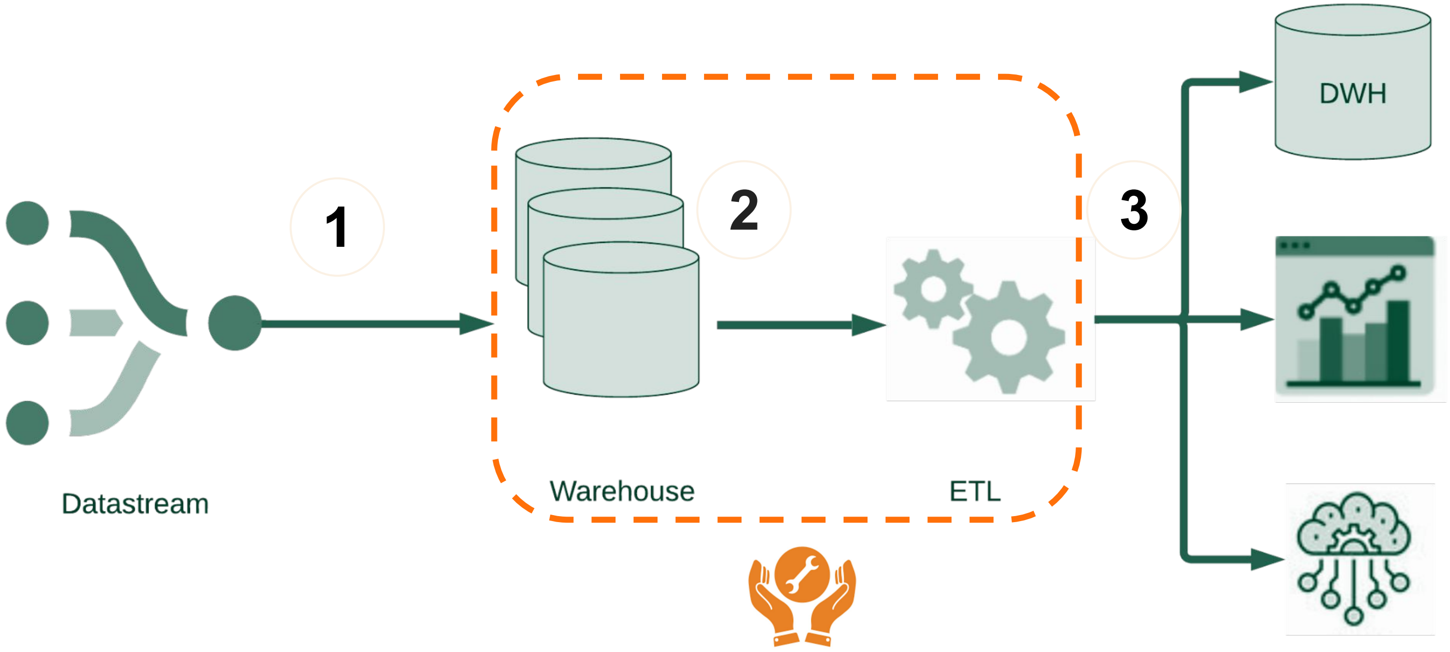
Use Case: Ads Measurement

Measurement pipelines powers business critical metrics used directly by customers for ads performance tracking and billing





Pipelines



Motivation To Change

Cost

Scale with some tools. Not always the right tool, with the right characteristics

Readability

SQL is powerful and convenient for simple tasks. Becomes hard as complexity grows

Collaboration

Challenging to manage how logic is shared and maintained

Testing

Can you test locally? Unit testing non-composable SQL is difficult

Requirements on Instacart Data system

Multi-language support

Centralize all our data on low cost object storage

Interoperability with other systems

Multi-language support

Core data infra/engineering team:

ML engineers/ Engineering team:

Data analysts/Data Engineering :

Flexibility comes with cost such as code reusability and difficulties for different teams to understand

Use s3 as storage

Scale about Instacart's data:

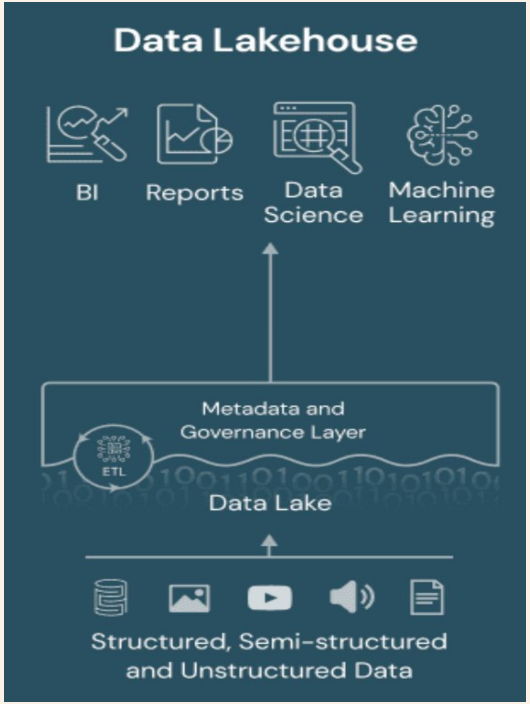
- 40+ PetaBytes (most of the historical data are not being frequently accessed)
- 30 + Billion new events being ingested to power ads, customers, shoppers systems
- Millions of prod/dev tables

- Intelligent Tiering
- Life cycle policies

In light of the feature/cost requirements, our decision was to embrace the concept of data lakehouse.

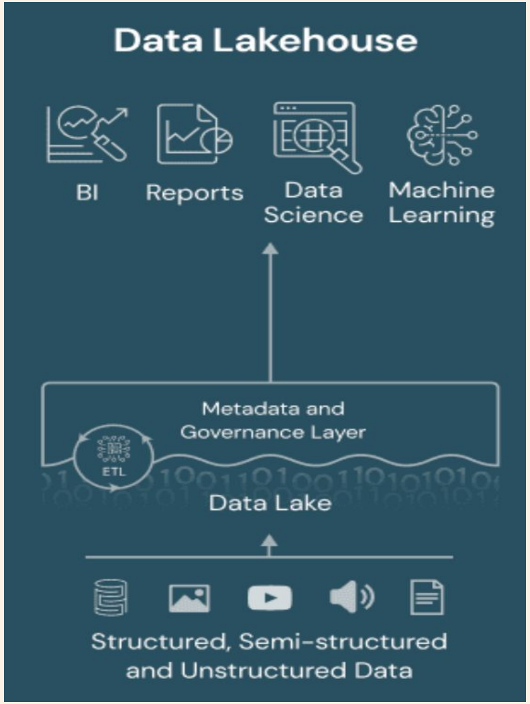
- Unified Data Platform for processing

  structured/unstructured data
- Efficient Data Processing: Batch/Incremental
- Cost-Effective Storage
- Advanced Analytics Capabilities(Beyond just sql)



Challenges:

- Permission management on s3
- Data applications development support: CI/CD, monitoring

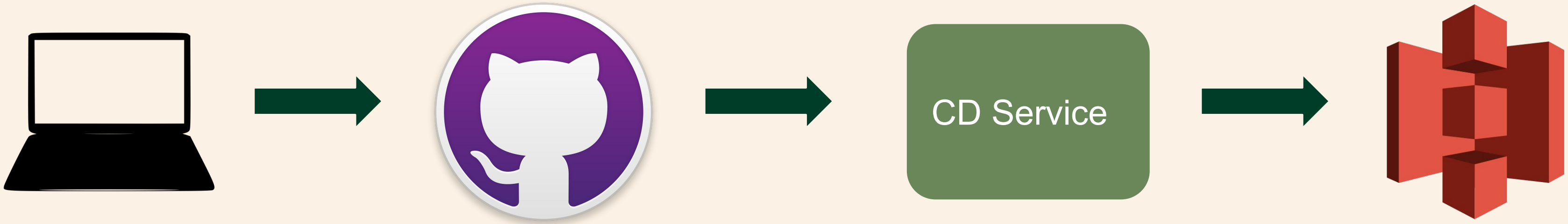


Self-served permission management module

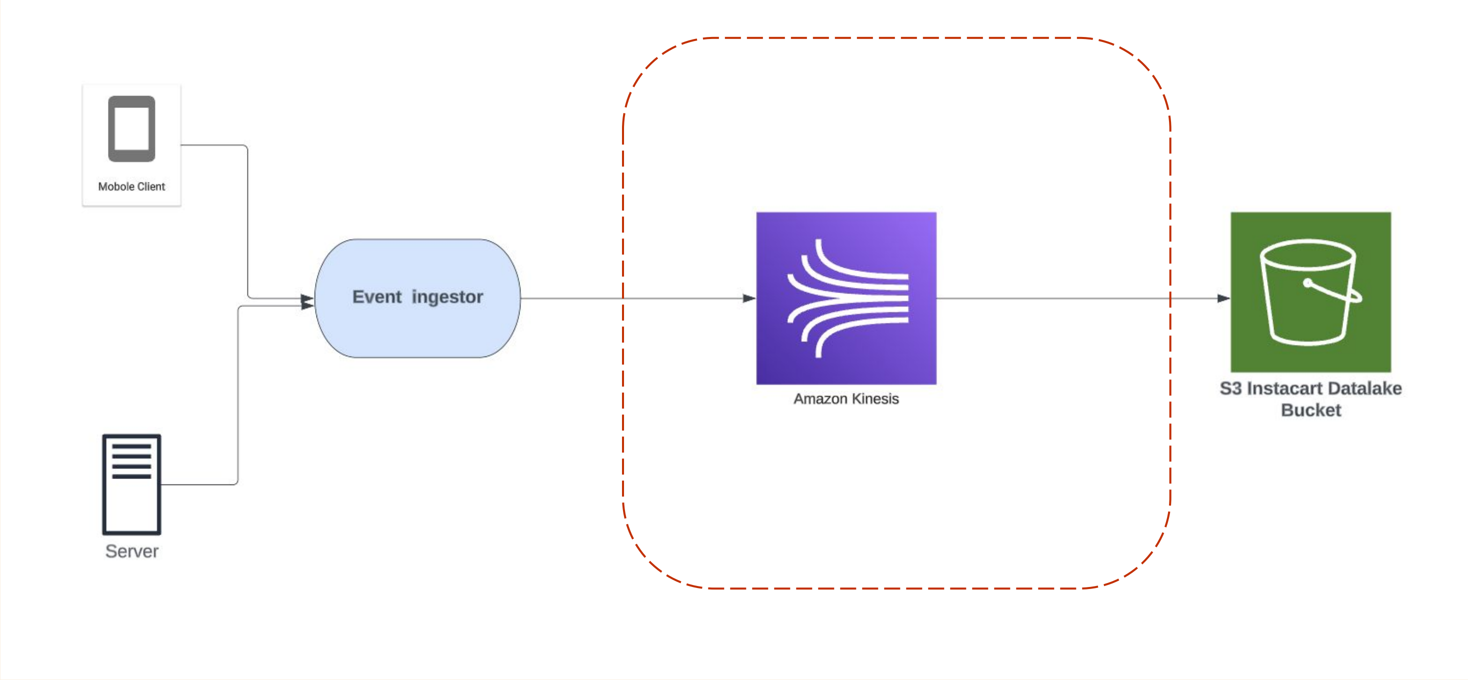
- Based on Terraform
- Provided abstraction for engineering team to manage the permission for their own data sets at s3 prefix level
- Engineering Team will be responsible for make changes on permission changes and approve PR
- CI will trigger the policy change

```
module "ads-data-db" {
  source = "./datalake-db"
  db_name        = "ads_data"
  db_description = "xxx"

  owning_team_role_ids = xxx

  read_grants = [
    reader_team1.role_ids,
    reader_team2.role_ids
  ]

  write_grants = [
    writer_team1.role_ids
  ]
}
```

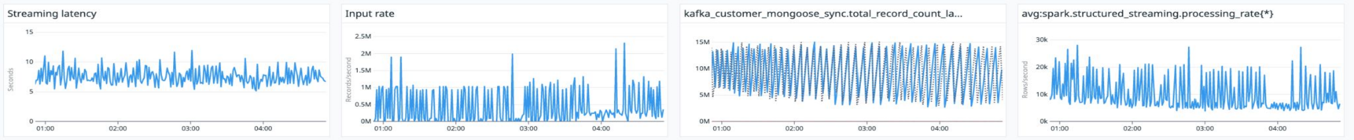
CI/CD for pipeline development



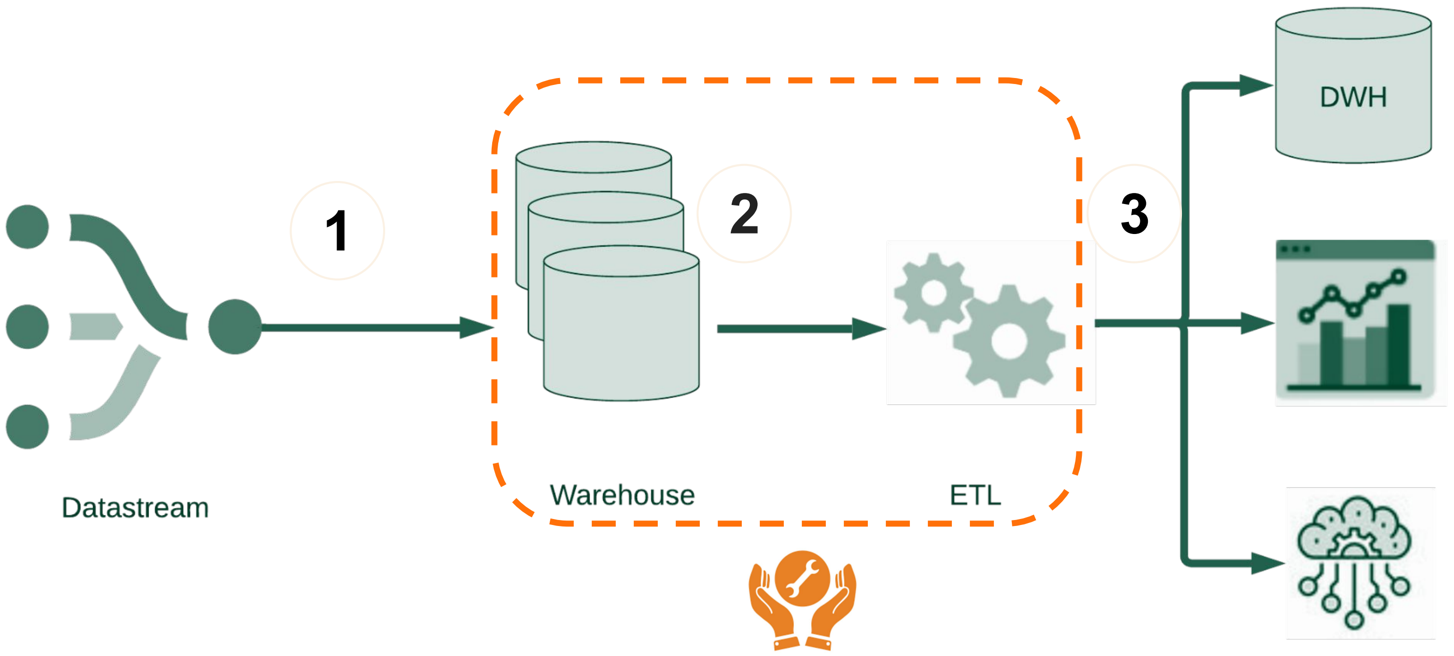
Deep dive on Events ingestion



Deep dive on Events ingestion –2



Pipelines



Incremental Pipelines (Before)

```
{% if 'event_time_begins_at' in (dag_run.conf or {}) and 'event_time_ends_at' in
(dag_run.conf or {}) %}
 AND EVENT_DATE_TIME_UTC >= $LOOKBACK_WINDOW_START
 AND EVENT_DATE_TIME_UTC <= $LOOKBACK_WINDOW_END
 AND EVENT_ID NOT IN (
   SELECT EVENT_ID FROM RAW_CONVERSIONS_NEBULA_ROLLBACK WHERE EVENT_DATE_TIME_UTC
>= $LOOKBACK_WINDOW_START and EVENT_DATE_TIME_UTC <= $LOOKBACK_WINDOW_END
 )
{% else %}
 AND loaded_at >= $LOOKBACK_WINDOW_START
 AND loaded_at <= $LOOKBACK_WINDOW_END
 AND EVENT_ID NOT IN (
   SELECT EVENT_ID FROM RAW_CONVERSIONS_NEBULA_ROLLBACK WHERE ETLED_AT_UTC >=
$DEDUPE_CHECK_LOOKBACK_WINDOW_STARTS_AT and ETLED_AT_UTC <=
$DEDUPE_CHECK_LOOKBACK_WINDOW_ENDS_AT
 )
{% endif %}
```

Custom window manipulation for frontline and backfill

Incremental Pipelines (Before)

```
{% if 'etl_begins_at' in (dag_run.conf or {}) and 'etl_ends_at' in (dag_run.conf or {}) %}
-- RUNNING IN MANUAL TRIGGER MODE

SET LOOKBACK_WINDOW_START = '{{ dag_run.conf.get("etl_begins_at") }}'::TIMESTAMP_NTZ;
SET LOOKBACK_WINDOW_END = '{{ dag_run.conf.get("etl_ends_at") }}'::TIMESTAMP_NTZ;
SET RUN_MODE = 'ETLED_AT_OVERRIDE';

{% elif 'event_time_begins_at' in (dag_run.conf or {}) and 'event_time_ends_at' in (dag_run.conf or {}) %}
-- RUNNING IN EVENT_DATE_TIME_UTC REPAIR MODE

SET LOOKBACK_WINDOW_START = '{{ dag_run.conf.get("event_time_begins_at") }}'::TIMESTAMP_NTZ;
SET LOOKBACK_WINDOW_END = '{{ dag_run.conf.get("event_time_ends_at") }}'::TIMESTAMP_NTZ;
SET RUN_MODE = 'EVENT_DATE_TIME_OVERRIDE';

{% else %}
-- RUNNING IN SCHEDULE MODE

SET ETL_LOOKBACK_BEGIN = '{{ next_execution_date.isoformat() }}'::TIMESTAMP_NTZ  - INTERVAL '3 hours';

-SET MIN_ETL_BEGINS = (SELECT DATEADD('HOUR', -1, MAX(ETLED_AT_UTC)) FROM RAW_CONVERSIONS_NEBULA_ROLLBACK);
```

Several modes to handle edge cases

Spark Streaming

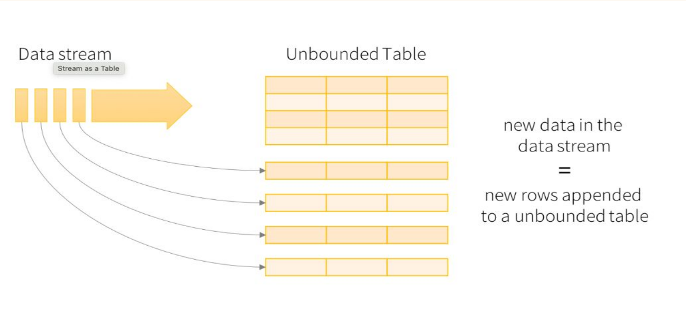
Divides the live stream into batches

Continuous stream of data is abstracted DStream(Discretized Streams)

Internally, a DStream is represented as a sequence of RDDs.

https://www.databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

Spark Structured Streaming

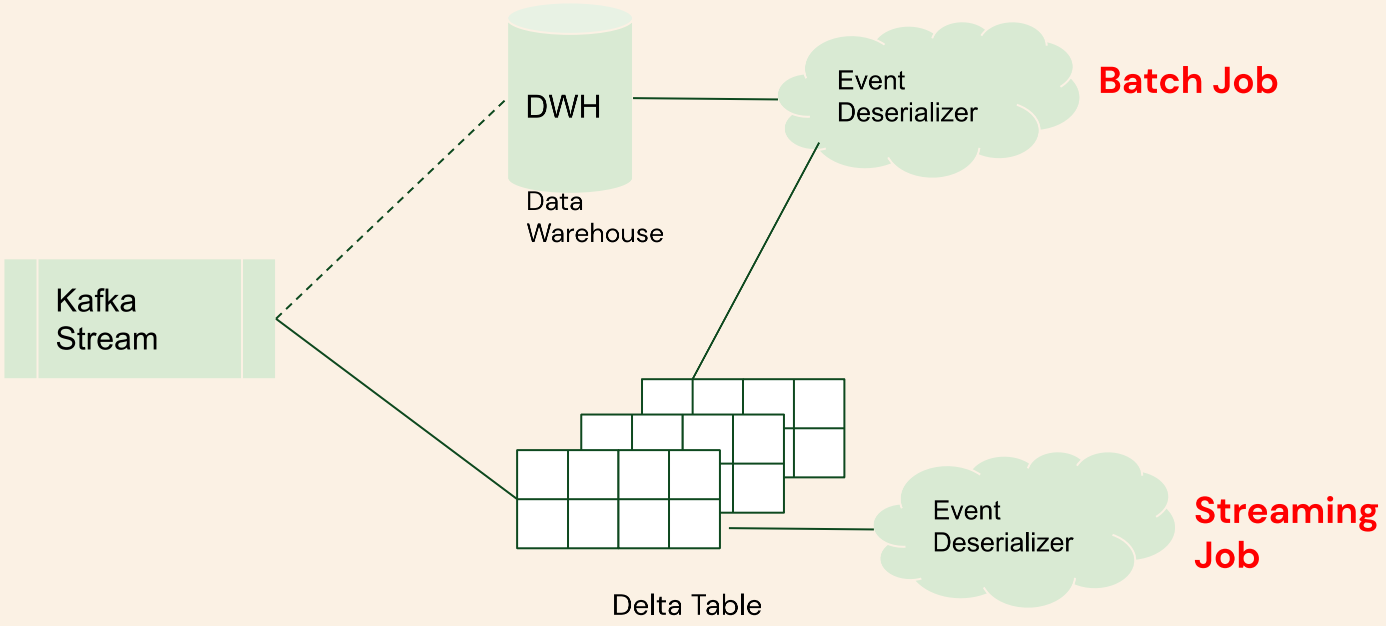


Data stream as an unbounded table

Treat a live data stream as a table that is being continuously appended.

Express streaming computation as standard batch-like query as on a static table

Event Processing Improvements



Event Processing with Structured Streaming

```
if (cliArgs.backfill())
    ExtractBackfill.apply(cliArgs, pipelineName)
else
     Extract.apply(cliArgs, pipelineName)
```

Stream Read

```
sparkSession.readStream
      .format("delta")
      .option("startingTimestamp",
      startingTimestamp)
      .load(deltaPath)
```

Batch Read

```
sparkSession.read
       .format("delta")
       .option("startingTimestamp",
              startingTimestamp)
       .option("endingTimestamp",
              endingTimestamp)
       .table(deltaTable)
```

Event Processing with Structured Streaming

Stream write

```
streamingDf.writeStream
  .foreachBatch { (batchDf: DataFrame, batchId: Long)
=>
    batchDf.persist()
    totalRowCount += batchDf.count()
    batchDf.unpersist()
    ()
  }
  .outputMode("append")
  .queryName(table)
  .trigger(Trigger.Once)
  .option("checkpointLocation", checkpointPath)
  .start()
  .awaitTermination()
```

Batch Write

```
df.write
    .format("delta")
    .partitionBy(partitionByColumn)
    .mode("overwrite")
    .option("replaceWhere",
deltaReplaceSql)
    .save(path)
```

Raw SQL was getting convoluted

```sql
-- We are using a modified last touch attribution model. For each order item, the last ad click that
-- occurred in the past 14 days will be attributed. If there are no ad clicks in the past 14 days
-- corresponding to this order, the last view will be used.

ALTER SESSION SET
  TIMEZONE = "UTC"
;

SET IS_MANUAL_TRIGGERED_RUN = {{ run_id.startswith('manual__') }};
SET IS_MONTHLY_LOOKBACK_RUN = FALSE;

{% if 'attribution_etl_window_end_date_pst' in (dag_run.conf or {}) %}
SET ETL_WINDOW_END_DATE_TIME = '{{ dag_run.conf.get("attribution_etl_window_end_date_pst") }}'::datetime;

{% else %}
SET ETL_WINDOW_END_DATE_TIME = CONVERT_TIMEZONE('UTC', 'US/Pacific', current_timestamp());

{% endif %}

-- Using a 3 day or CH lookback window buffer to take into account orders changes that may occur upstream.
-- If it's the first dag run of the day, we use the CH lookback window. For all other runs, we use 3 days look back period.
-- Note: We are using the assumption that the ads_attribution dag runs every 2 hours. Changing the frequency might break the lookback period logic.
{% if 'order_window_start_date_pst' in (dag_run.conf or {}) %}
SET ORDER_LOOKBACK_WINDOW_START_DATE = '{{ dag_run.conf.get("order_window_start_date_pst") }}'::date;

{% else %}

SET IS_MONTHLY_LOOKBACK_RUN = iff((extract('minute', current_timestamp()) + extract('hour', current_timestamp()) * 60) < 180, TRUE, FALSE);

SET ORDER_LOOKBACK_WINDOW_START_DATE = iff($IS_MONTHLY_LOOKBACK_RUN,
  CONVERT_TIMEZONE('UTC', 'US/Pacific', FP_CONSOLIDATION_START_TIME(current_timestamp()))::date,
  DATEADD('DAY', -3, $ETL_WINDOW_END_DATE_TIME)::date);

{% endif %}
```

MADE WITH GIFOX

Wins so far

- Quick and easy to set up
- Logic mostly fits in a single query
- Code runs as is in test mode

Cons

- Repeated logic copy/pasted
- No Modularity
- Poor composability
- Not unit test friendly

Modular Code in Spark

- Generalized ETL entrypoints
- Each stage is pluggable

```
val runPipeline =
 for {
   inputDfMap <- Extract.apply(cliArgs)
   outputDf   <- Transform.apply(inputDfMap, cliArgs)
   result     <- Load.apply(outputDf, cliArgs)
 } yield result
```

Modular Code in Spark

Structured in logical blocks

```
val stgOrderItems: DataFrame = getStgOrderItems(inputDataDfMap)

val attributableEventOrders: DataFrame = getAttributableEventOrders(inputDataDfMap,
stgOrderItems, cliArgs)

val attributableEventOrdersWithPartitionCol: DataFrame = attributableEventOrders
 .withColumn(AEOSchema.getPartitionColumn(),
to_date(attributableEventOrders(AEOSchema.ORDER_ITEM_CREATED_DATE_TIME_PT),
"yyyy-MM-dd"))

attributableEventOrdersWithPartitionCol.cache()
val attributableEventOrdersCount = attributableEventOrdersWithPartitionCol.count()
```

Dev + Productivity Improvements

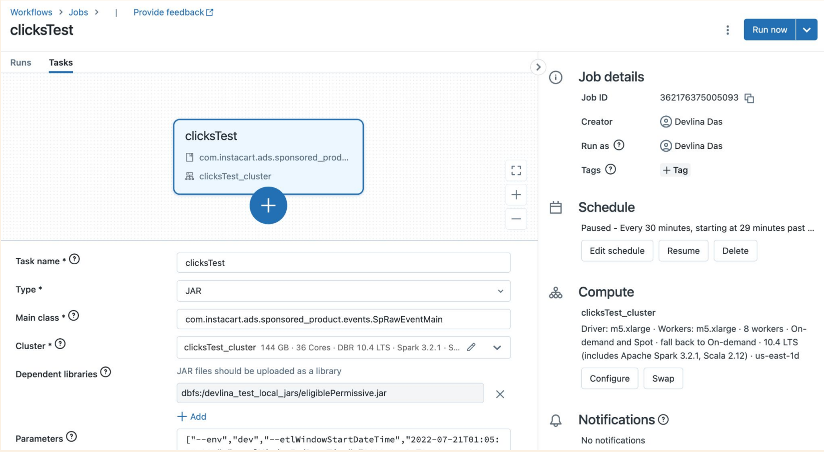
- DRY: Shared modules and libraries

- Faster code iteration with in-memory local testing
  - sbt test
  - quick dev feedback loop

- Delta schema management
  - Schema evolution with Schema on write paradigm
  - https://delta.io/blog/2023-02-08-delta-lake-schema-evolution/
  - https://www.databricks.com/blog/2019/09/24/diving-into-delta-lake-schema-enforcement-evolution.html

Operational Improvements

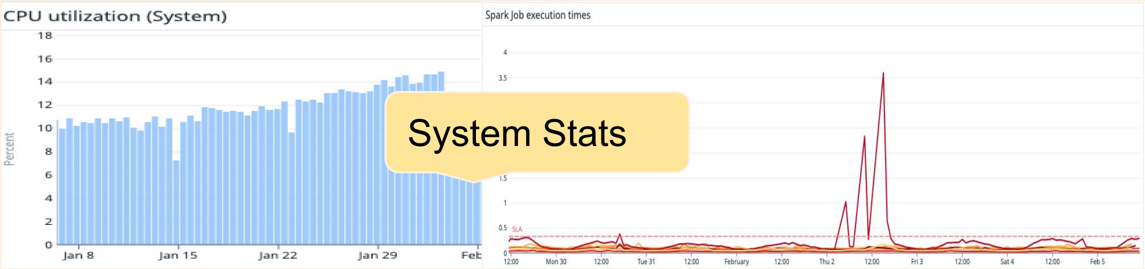
Shadow Testing with on-demand job clusters

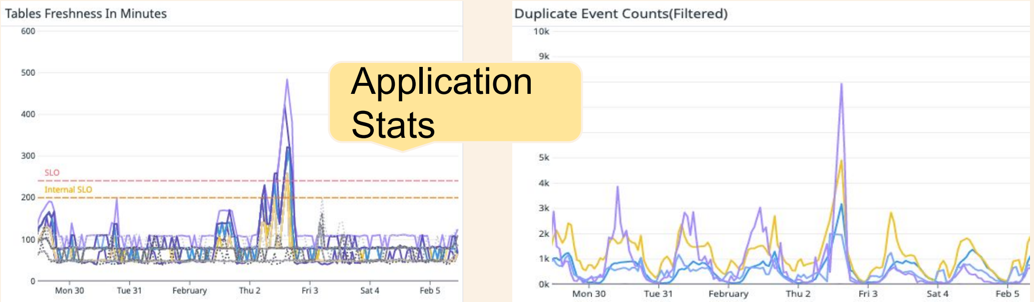


Cluster resource control for speed and cost

```
"driver_node_type_id": "i3.4xlarge",
"enable_elastic_disk": "true",
"node_type_id": "r4.2xlarge",
"num_workers": "20",
"policy_id": "9C60384B6B001F8C",
"spark_conf": {

"spark.databricks.adaptive.autoOptimizeShuffle.
enabled": "true",
"spark.databricks.io.cache.enabled": "true",

...

}
```

Observability Improvements





Test Coverage

Missed Lines	Total Lines	Cov.
	96	90%
	65	87%
		0%
		0%
	18	100%
	5	100%
	1	100%
17 of 185	185	90%

Great Possibilities Ahead!

- Unit Test framework
- Advanced monitoring
- Generalized Pipeline Framework
- Unified Schema Management

Ana Lemus, Arthur Li, Brandon Williams, Bruno Caminada, Craig Flockhart, Devlina Das, Ji Wu, Kieran Taylor, Luke Snyder, Mark Lee, Mohit Gupta, Nate Kupp, Navin Sridhar, Peter Lambe, Prateek Jaipuria, Rohith Macherla, Roy Moranz, Sai Karthik Varanasi, Sanchit Gupta, Shelby Ferson, Shihuan Liu, Srikanth Reddy, Trey Zhong, Yingshi Zhang


Questions
instacart