ATLASSIAN

Data Mesh and Compliance in a Multi-Regional Data Lake at Atlassian

Our journey building a customer facing Data Lake

Mike Dias
Principal Engineer

Paul Ashley
Principal Engineer

Agenda

Opportunities and Challenges

The motivation behind the Customer Data Lake

Data Replication

Our Data Mesh approach for CDC replication

Streaming Architecture

Realtime analytics with Delta Tables

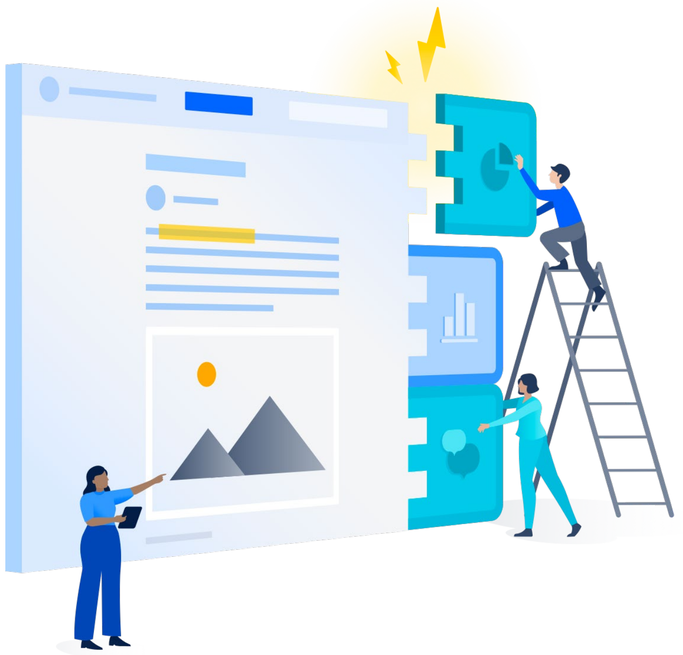
Multi-Region Compliance

Data Residency and GDPR at scale

Putting all together

Did we actually achieve our goals?

Opportunities and Challenges

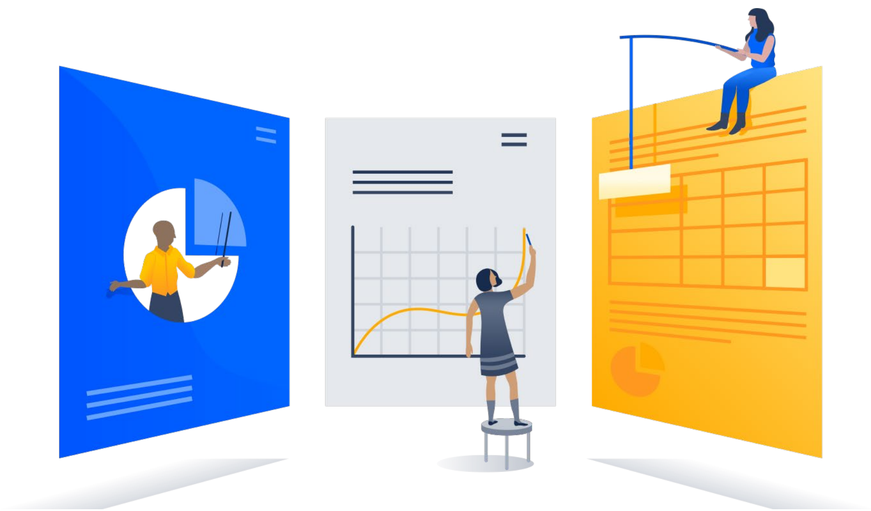


Atlassian Analytics

With the advent of the Chartio acquisition, we wanted to create a seamless analytics experience for customers on top of their own data

Data Exports

Customers migrating from our on-prem solutions to our cloud struggled with the missing direct access to their databases to export their data



In-Product Analytics

We want to get the data out of product databases in order to provide efficient analytical experiences in-product without overloading the operational databases

ML Training

We want to create Machine Learning models that better fits customer data to produce more acurate results, leading to a better experience

Let’s build a customer facing Data Lake!

Data Replication

Replication Challenges

Many Technologies

From monoliths to microservices and serverless. We have a wide rage of technologies and architectures in multiple languages.

Many Databases

From sharded fleets of RDSs to DynamoDB and MongoDB. Data is stored in many different databases now and it will continue to evolve over time.

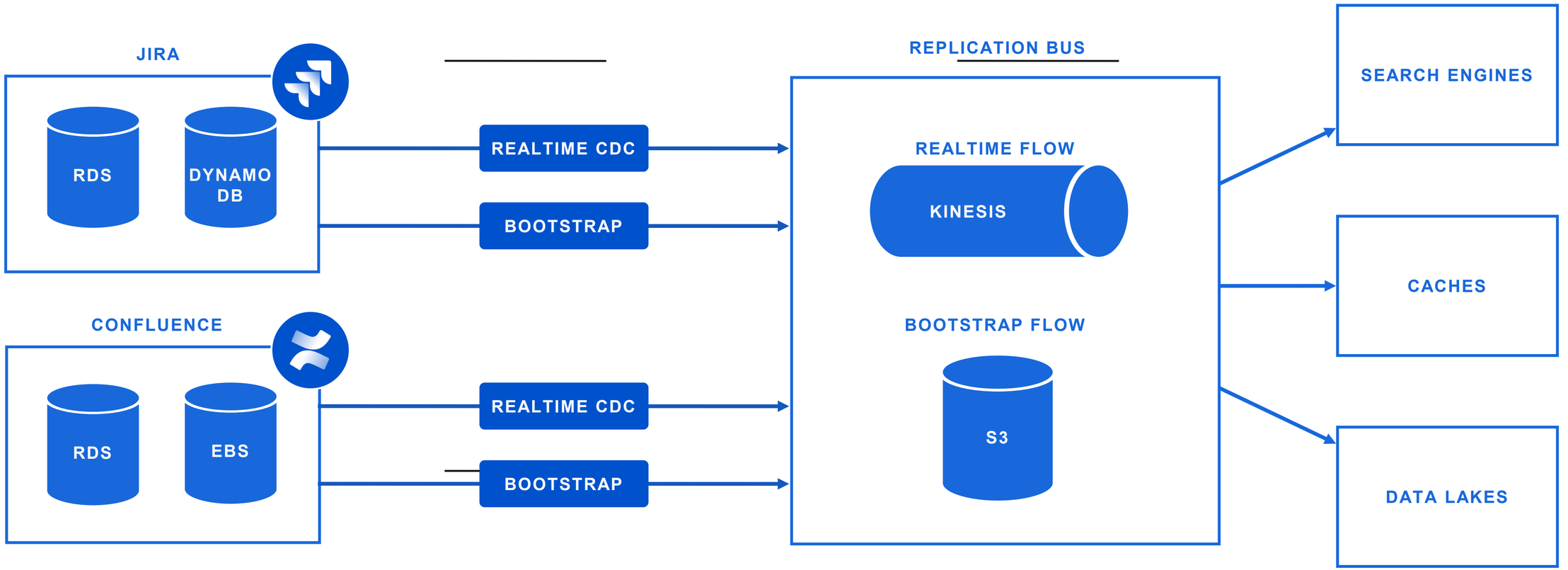
Many Teams

Dynamic teams constantly changing due to reorgs, find the right people to talk about specific parts of the products can be challening.

Logical vs Physical

	Logical Replication (Application instrumentation)	Physical Replication (Database instrumentation)
Technology availability (How much it depends on specific technology capabilties?)	Technology independent ✓	Technology dependent ✗
Stability (How often it can break?)	Slowly evolving contracts ✓	Rapidly evolving contracts ✗
Flexibility (How easy is for producer to change their systems?)	Most flexible ✓	Less flexible ✗
Effort (How much we need to invest to replicate data?)	High investment upfront ✗	Low investment upfront ✓

Replication Bus



Replication Protocol

Protobufs

Compact wire format with good backwards compatibility.

LWW CRDT

Garantees converging to the latest version even on unordered transports.

Generic Payload

Any protobuf with support for large messages offloaded to S3.

```
message FacsimileRecord {
  ARI resource_ari = 1;
  int32 entity_type = 2;
  ARI workspace_id = 3;

  int64 version = 4;
  int64 generation_watermark = 5;
  bool is_tombstone = 6;

  oneof payload {
    google.protobuf.Any entity = 7;
    OffloadPointer offload_pointer = 8;
  }
}
```

Schema Management

Centralized repo

All conceptual are defined in the same place in Typescript as the agnostic language.

Schema evolution

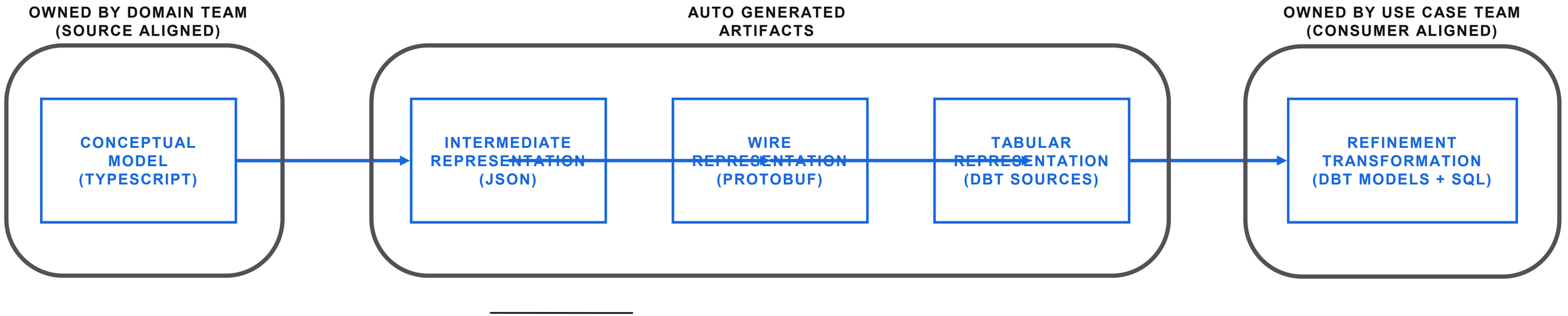
Tools to enforce expand-contract rules to prevent breaking changes.

Domain ownership

Teams are responsible for evolving their models as they evolve their applications.

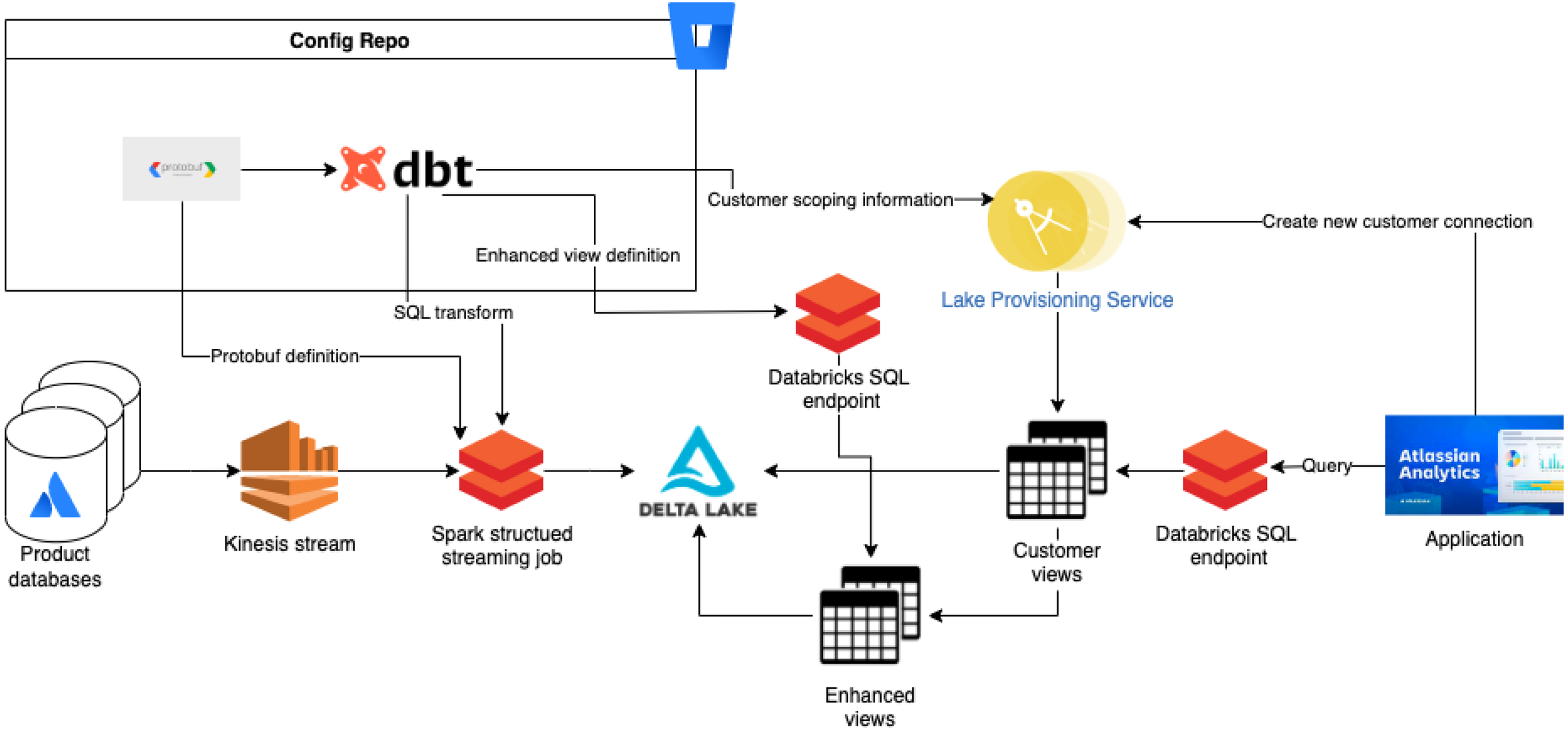
```
export interface JiraIssueManifest extends Manifest {
  owner: Team.JIRA_ISSUE_EXTRACTION;
  status: EntityStatus.ACTIVE;
}

export interface JiraIssue extends Entity {
  issueId: String;
  project: ComposedBy<JiraProject>;
  fields: JiraIssueField[];
}
```

Data Mesh Principles over Artifacts

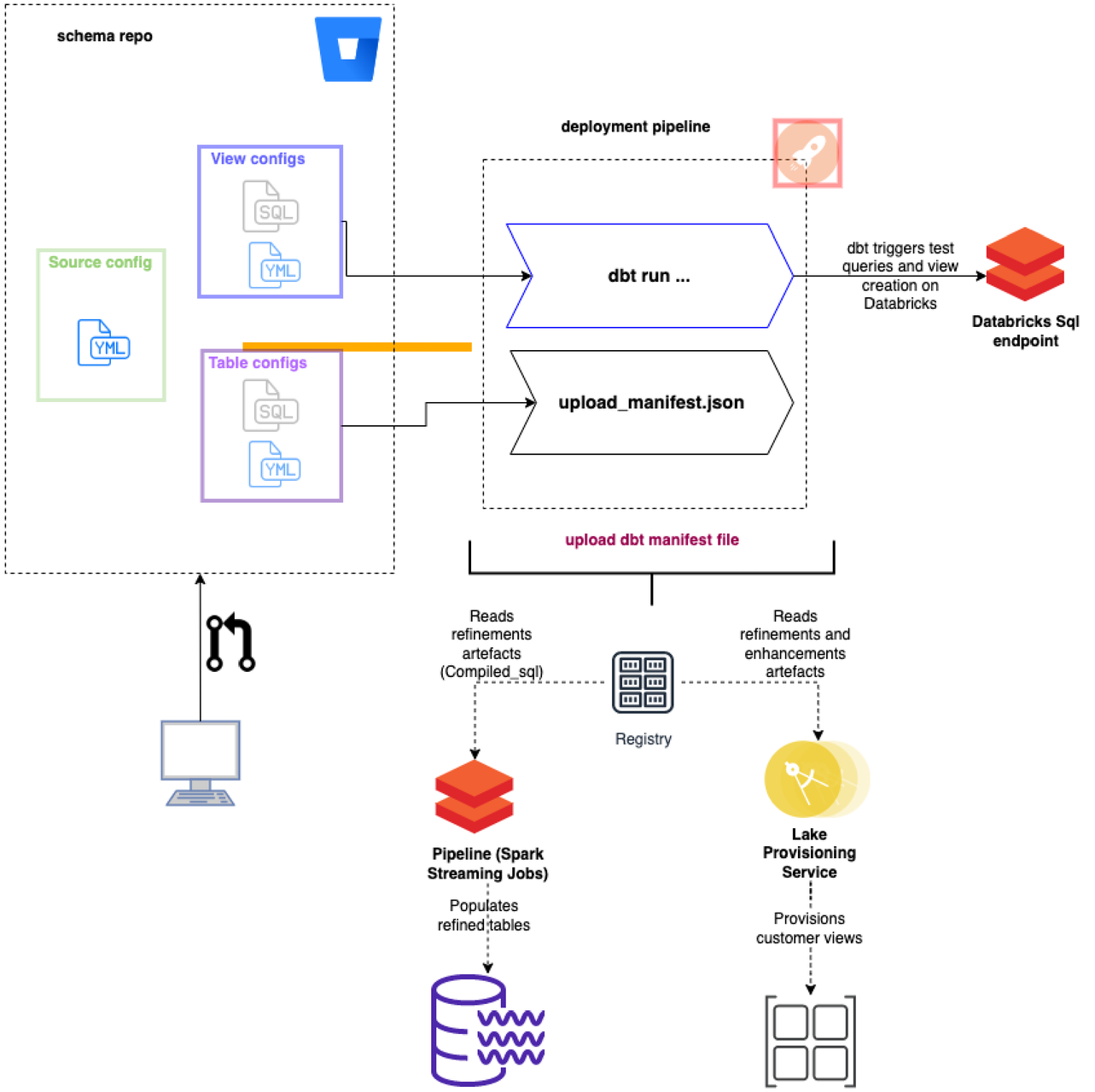


Streaming Architecture

OUR LAKEHOUSE



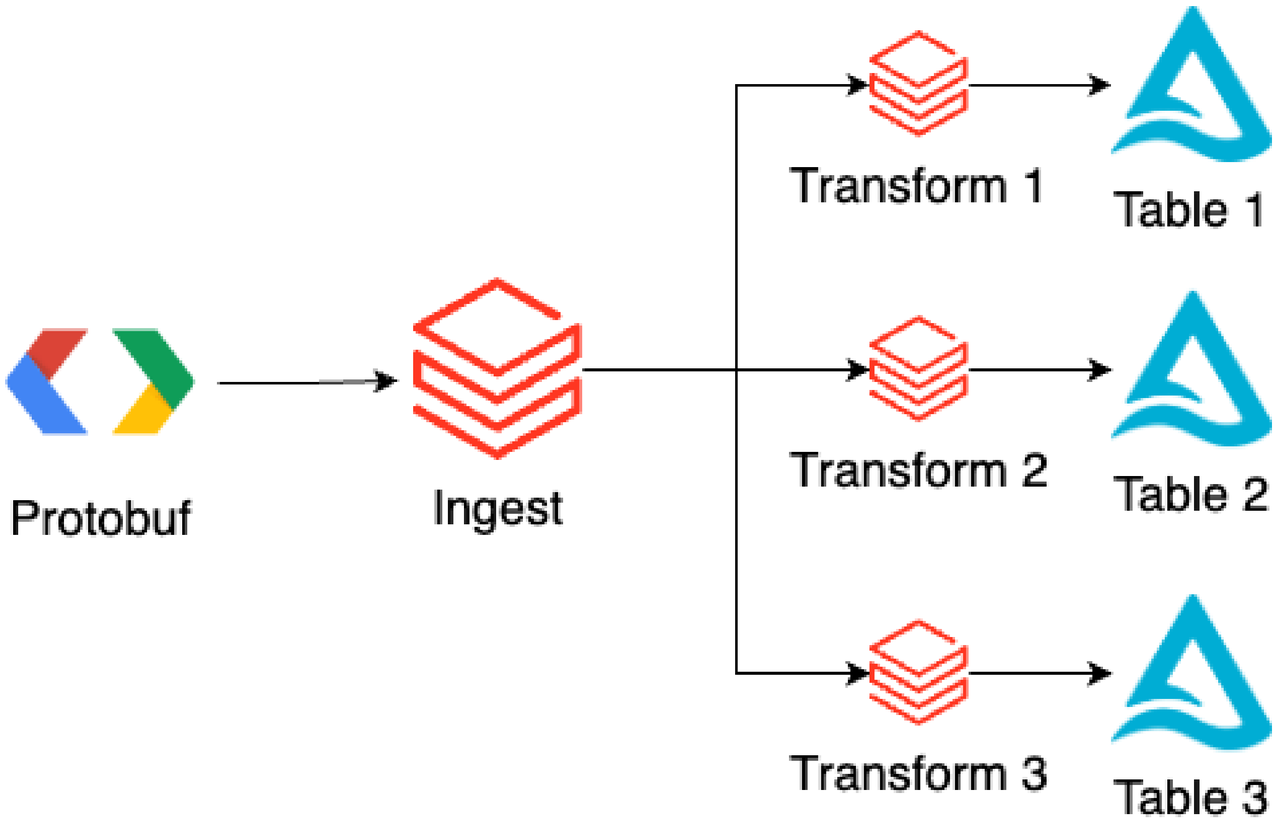
dbt Workflow



Transform Process

1:n proto to table mappings

In the ingestion pipeline each protobuf from a source product maps to many different tables for the customer



SPARK STREAMING JOB - TRANSFORM PROCESS

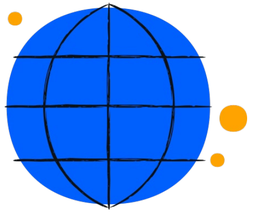
```sql
WITH cte_0 AS (
SELECT
    {{ select_envelope_fields() }}
    CAST(id_value AS STRING) AS id_value,
    CAST(created_at_seconds + CAST(created_at_nanos / 1e9 AS DOUBLE) AS TIMESTAMP) AS created_at,
    CAST(question AS STRING) AS question,
    CAST(title AS STRING) AS title,
    CAST(updated_at_seconds + CAST(updated_at_nanos / 1e9 AS DOUBLE) AS TIMESTAMP) AS updated_at,
    CAST(author_value AS STRING) AS author_value,
    CAST(workspace_value AS STRING) AS workspace_value,
    CAST(question_adf AS STRING) AS question_adf,
    CAST(vote_count AS INTEGER) AS vote_count,
    CAST(chosen_answer AS BOOLEAN) AS chosen_answer,
    CAST(is_resolved AS BOOLEAN) AS is_resolved,
    CAST(classification AS STRING) AS classification,
    {{ select_required_fields() }}
FROM {{ source('manifold_entities', 'AvocadoQuestion') }}
)
SELECT
    {{ select_envelope_fields() }}
    id_value,
    created_at,
    question,
    title,
    updated_at,
    author_value,
    workspace_value,
    question_adf,
    vote_count,
    chosen_answer,
    is_resolved,
    classification,
    {{ select_required_fields() }}
 FROM cte_0
```

dbt

```scala
override def transform(dataframe: DataFrame)(implicit entityCompanion: EntityMapper.EntityCompanion): DataFrame = {
  val dataframeWithEnumsResolved = resolveEnumsInMicroBatch(dataframe, entityCompanion)

  val viewName = getViewNameFromManifold

  dataframeWithEnumsResolved.createOrReplaceTempView(viewName)

  val parsedSql = sql.replace( target = "manifold_entities." + viewName, viewName)

  dataframeWithEnumsResolved.sparkSession.sql(parsedSql)
}
```

```
{
  "compiled_sql": "WITH cte_0 AS (
SELECT
    CAST(envelope_fields_resource_ari_value AS STRING) AS envelope_fields_resource_ari_value,
    CAST(envelope_fields_version AS LONG) AS envelope_fields_version,
    CAST(envelope_fields_is_tombstone AS BOOLEAN) AS envelope_fields_is_tombstone,
    CAST(envelope_fields_workspace_id_value AS STRING) AS envelope_fields_workspace_id_value,
    CAST(envelope_fields_generation_counter AS LONG) AS envelope_fields_generation_counter,
CAST(id_value AS STRING) AS id_value,
CAST(created_at_seconds + CAST(created_at_nanos / 1e9 AS DOUBLE) AS TIMESTAMP) AS created_at,
CAST(question AS STRING) AS question,
CAST(title AS STRING) AS title,
CAST(updated_at_seconds + CAST(updated_at_nanos / 1e9 AS DOUBLE) AS TIMESTAMP) AS updated_at,
CAST(author_value AS STRING) AS author_value,
CAST(workspace_value AS STRING) AS workspace_value,
CAST(question_adf AS STRING) AS question_adf,
CAST(vote_count AS INTEGER) AS vote_count,
CAST(chosen_answer AS BOOLEAN) AS chosen_answer,
CAST(is_resolved AS BOOLEAN) AS is_resolved,
CAST(classification AS STRING) AS classification,
    CAST(row_refreshed_at AS TIMESTAMP) AS row_refreshed_at,
CAST(row_refreshed_at_day AS DATE) AS row_refreshed_at_day,
CAST(shard_id AS STRING) AS shard_id,
CAST(workspace_id AS STRING) AS workspace_id
FROM manifold_entities.AvocadoQuestion
)
SELECT
    CAST(envelope_fields_resource_ari_value AS STRING) AS envelope_fields_resource_ari_value,
    CAST(envelope_fields_version AS LONG) AS envelope_fields_version,
    CAST(envelope_fields_is_tombstone AS BOOLEAN) AS envelope_fields_is_tombstone,
    CAST(envelope_fields_workspace_id_value AS STRING) AS envelope_fields_workspace_id_value,
    CAST(envelope_fields_generation_counter AS LONG) AS envelope_fields_generation_counter,
id_value,
created_at,
question,
title,
updated_at,
author_value,
workspace_value,
question_adf,
vote_count,
chosen_answer,
is_resolved,
classification,
    CAST(row_refreshed_at AS TIMESTAMP) AS row_refreshed_at,
CAST(row_refreshed_at_day AS DATE) AS row_refreshed_at_day,
CAST(shard_id AS STRING) AS shard_id,
CAST(workspace_id AS STRING) AS workspace_id
 FROM cte_0"
}
```

Three Fixed Writer Types



Append

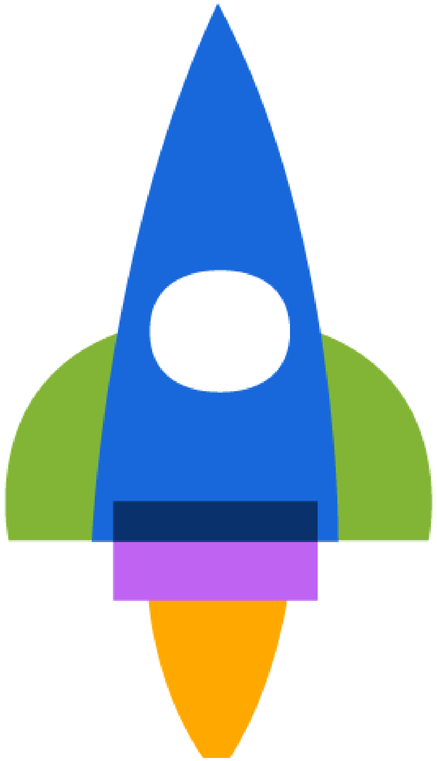
Does not upsert data, appends each row to the table verbatim

Merge

Matches on primary key, taking the highest versioned entity

MergeExploded

For inserting exploded arrays, similar to merge but replaces multiple instances of the same PK



Challenges

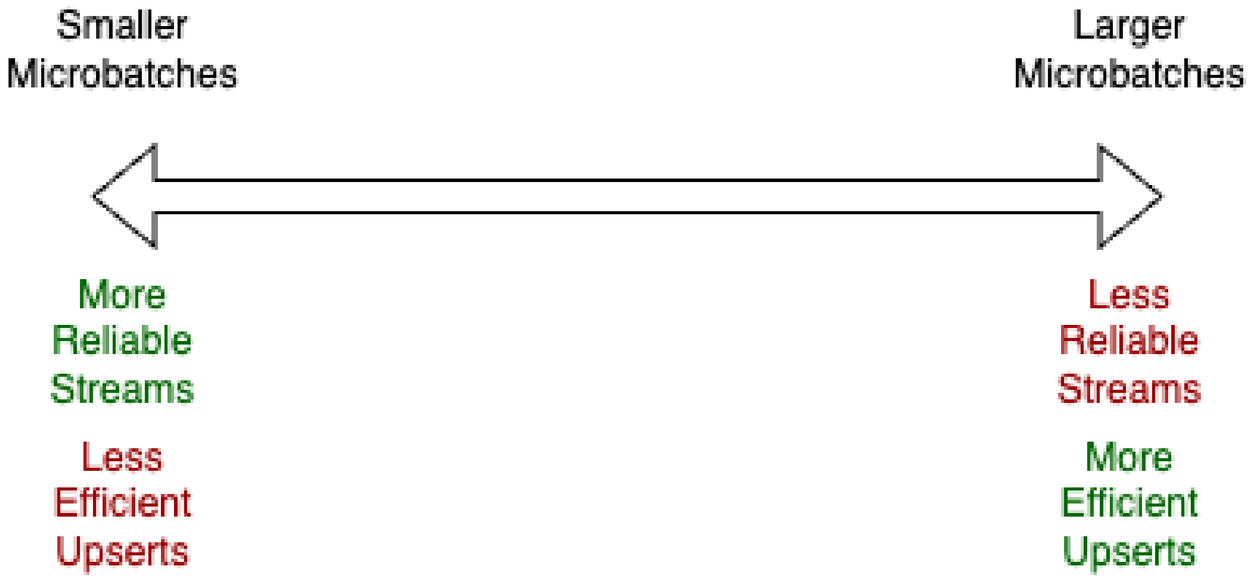
Large Table Stability

Upserting to large tables is slow

We need to upsert in large micro batches for efficiency to scale

BUT

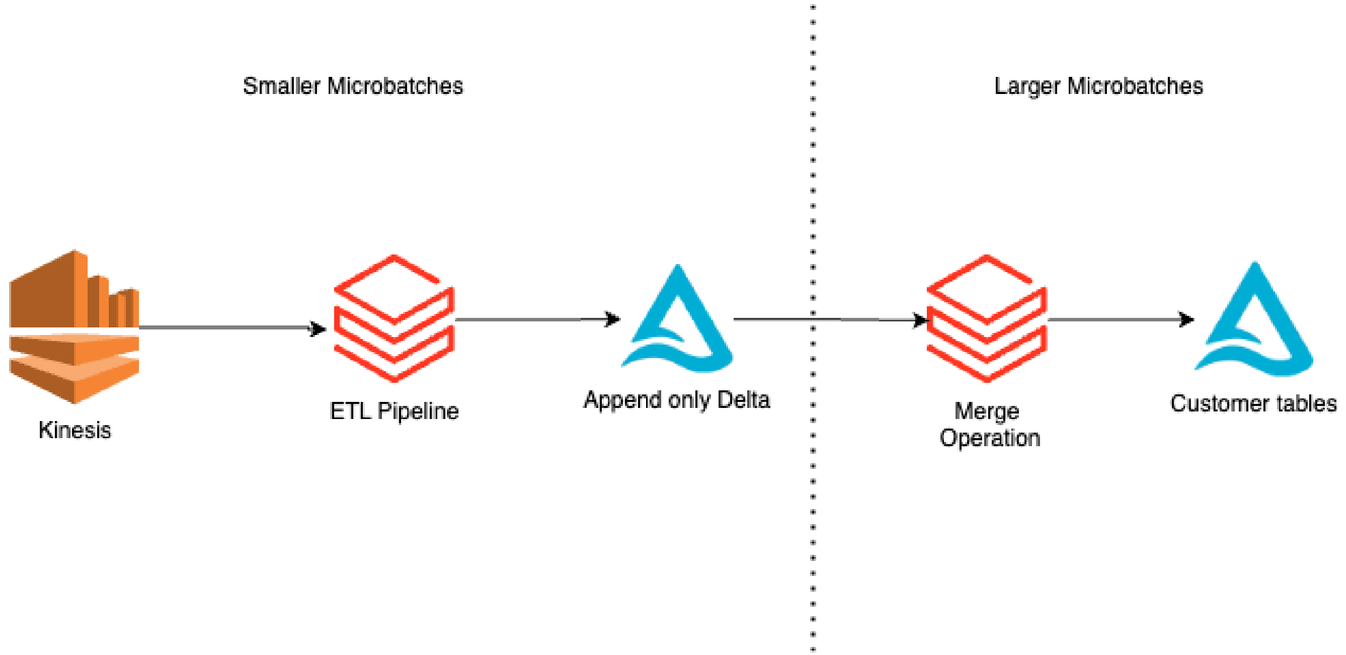
Doing so makes fault tolerance in spark more problematic



Large Table Stability

Mitigation

Break the pipeline into two steps, doing the merge as a final step, reading from an append-only sink



Scaling lots of streaming jobs - Autoscaling

Managing and scaling streaming jobs with dynamic load is hard

We added lag metrics which get sent to a PID controller on our control plane, which then adjusts the workers to try to reach a set point.

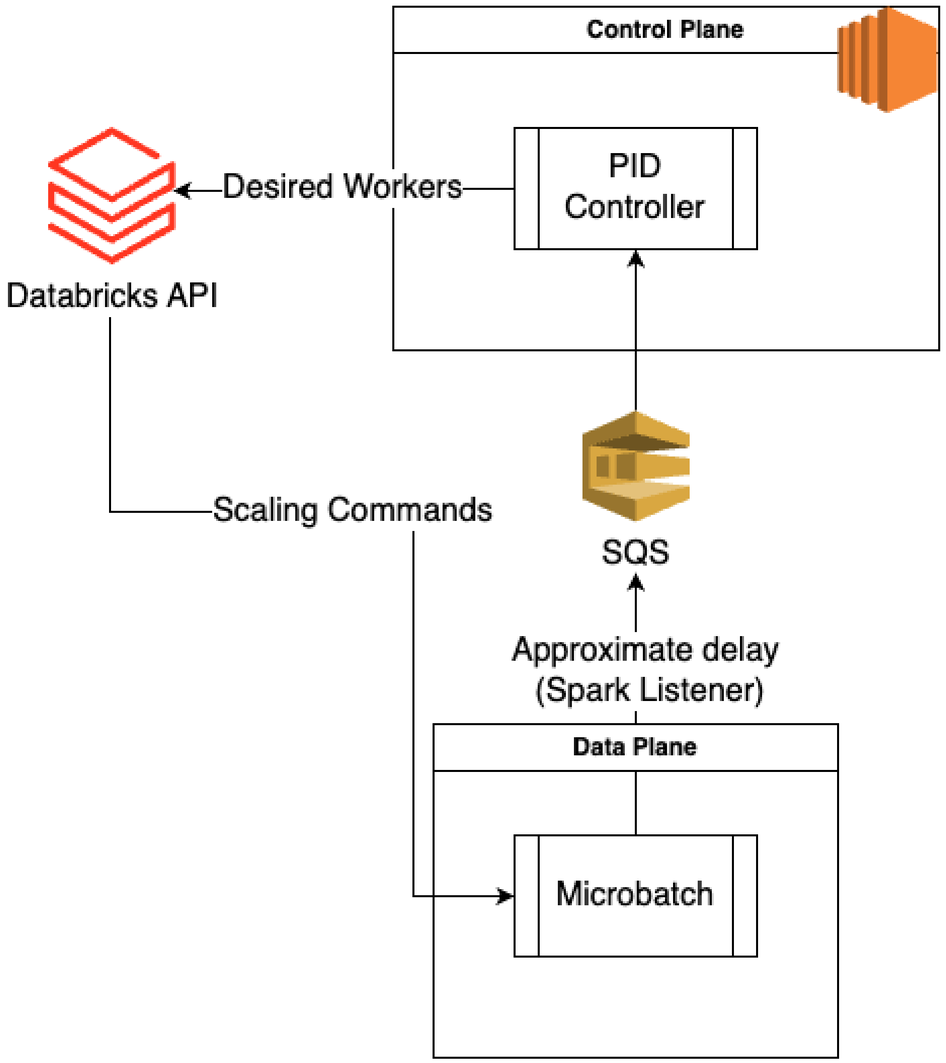
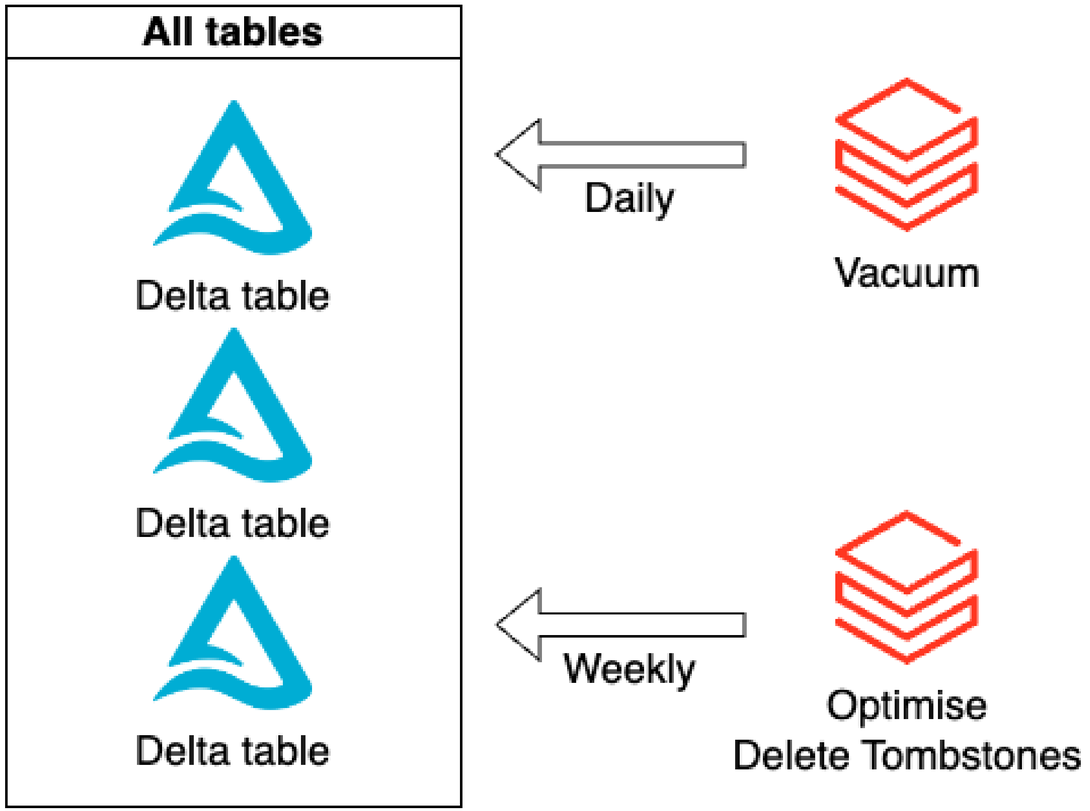


Table Maintenance

Two maintenance schedules

Conflictless jobs (e.g. Vacuum) run daily

Conflicting jobs (e.g. optimise) runs weekly during scheduled downtime



Multi-Region Complia

Why do Multi Region?

Give customers choice

Customers want the option to choose where their data is ultimately housed to comply with regulations.

Improve Ingestion performance

Distributing data across several regions reduces the tables size. . . which improves merge performance

Improve query latency

Having the data closer to the customer reduces query times

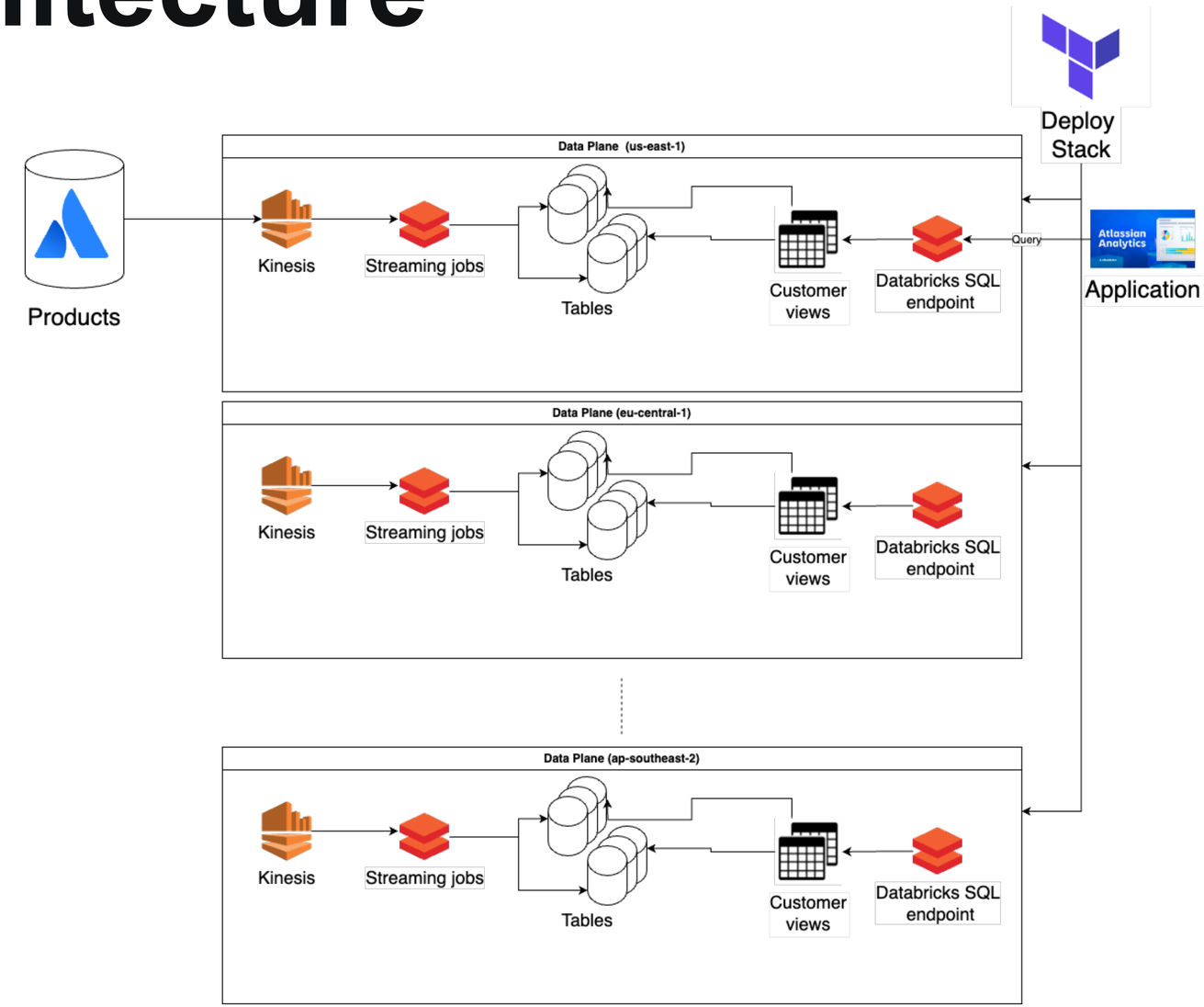
Blast radius reduction

One region or pipeline failure does not affect all of our customers

Multi-region Architecture

Every customer workspace is configured into one of 12 different AWS regions. All Storage and compute for that customer is performed in that region

We deploy our entire data plane architecture across several regions by utilising terraform modules, allowing us to easily replicate the entire stack in a new environment/region.



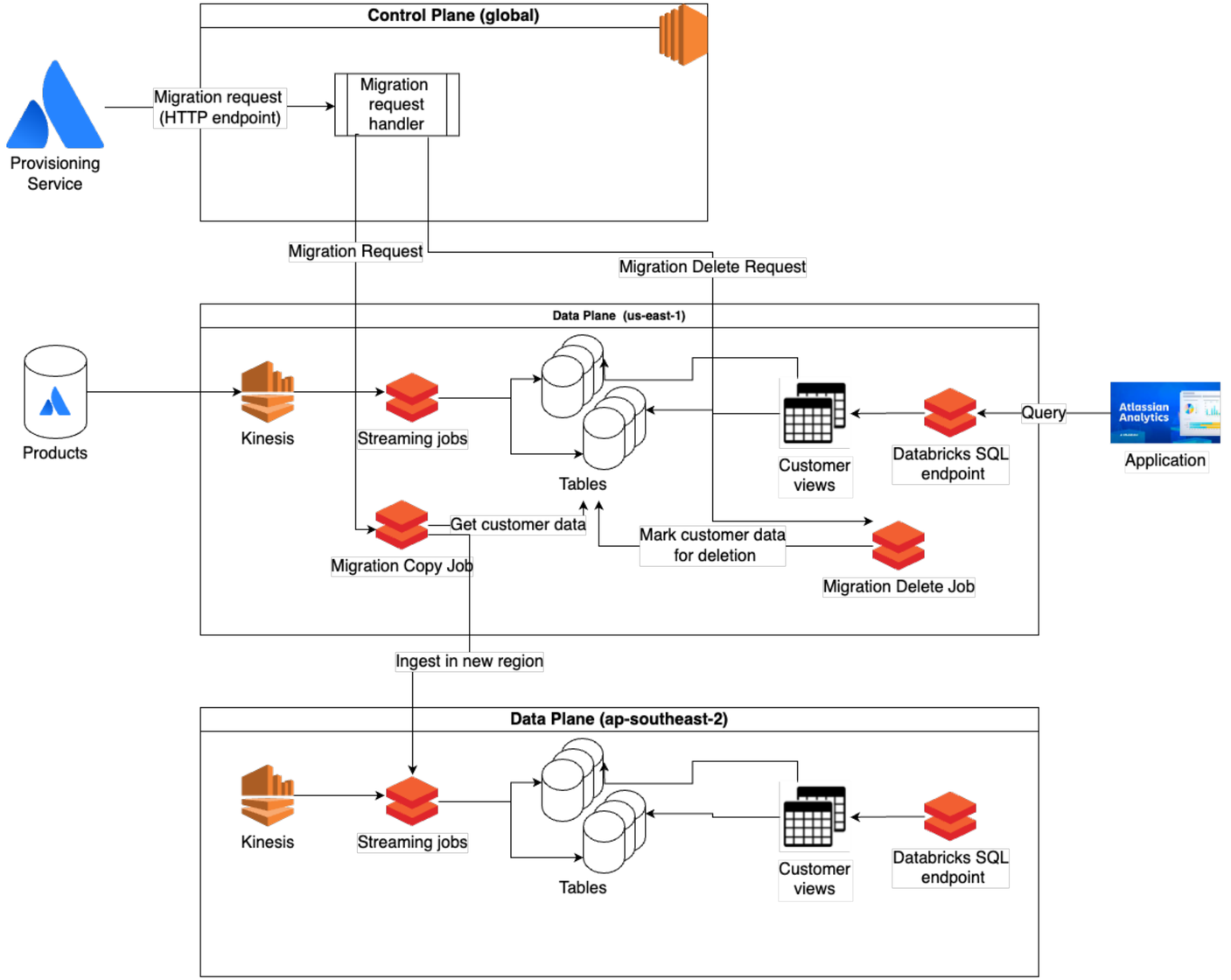
Changing a customers data residency

A customers data region can change by:

1. Customers electing to have their data bound to a specific region
2. Unbounded customers usage patterns indicate they are better suited to another region

Migration requests

Each migration request is handled by our global control plane, which starts a job to re-ingest the data in the new region, followed by a job to delete the old data once the copy finishes



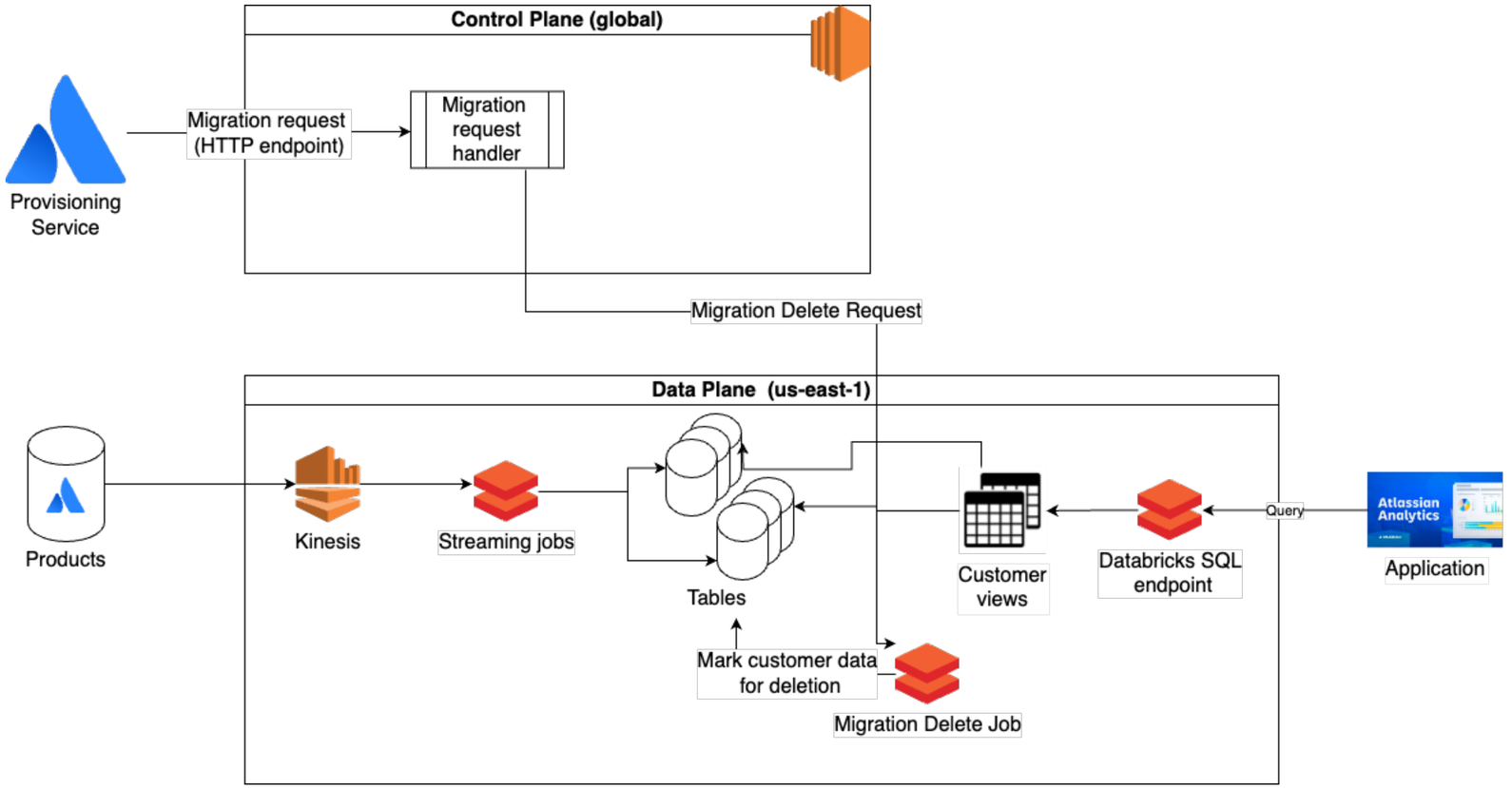
GDPR- Deletions and resurrections

Handled in the same way as migration requests . . . without the copy

Our control plane receives the delete request and executes a job on the Data Plane to mark the data for deletion

Deletion requests

Each Deletion request is handled by our global control plane, which fires off a job to delete the old data



What is “Bring Your Own Key” (BYOK)?

- Additional security feature offered by Atlassian allowing customers to provide and manage their own encryption keys for their data

BYOK - Challenges

Cannot natively encrypt beyond the table level

Natively, encryption only occurs at the table level. Whilst this is possible for our lake, it would involve creating and managing hundreds of thousands of tables

Decryption from SQL warehouses

We needed a way for our SQL warehouses to seamlessly identify and apply the customers keys to be able to read data from the lake

Simplicity

We did not want to have to manage a completely new stack and suite of business logic specifically for this use case.

BYOK - S3 FS strategy

Use AWS Encryption at the HIVE partition level

It’s a common misconception that SSE-KMS only works at bucket-level because most examples focus on bucket configurations but in fact, key ids can be specified at the object level

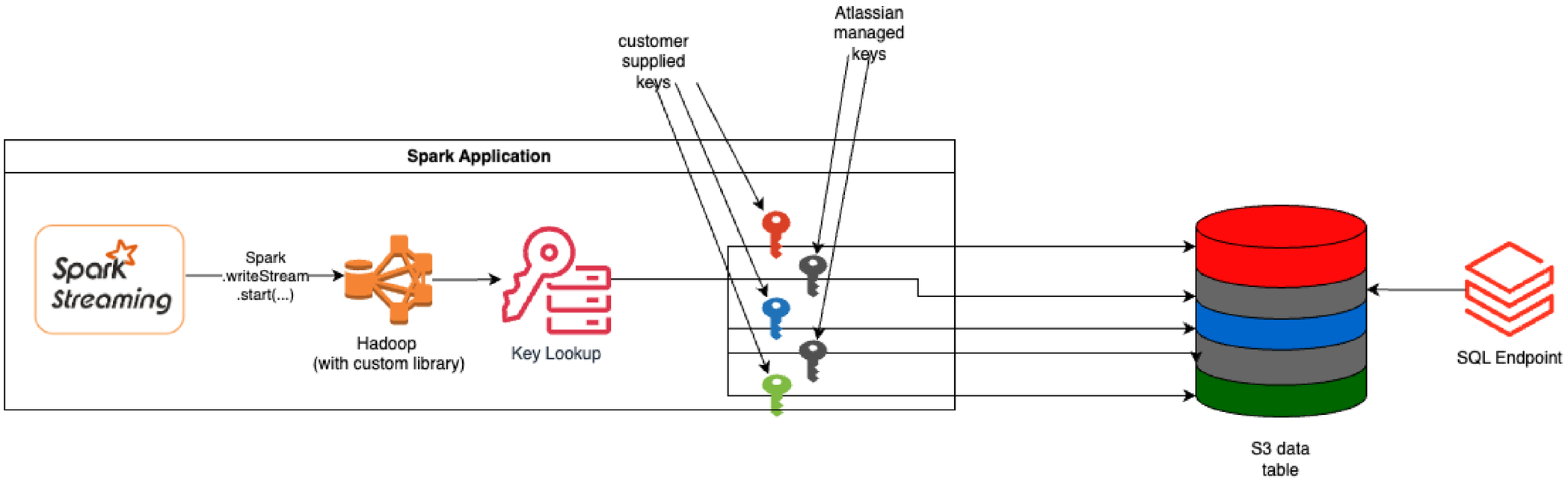
Extend Hadoop S3AFileSystem

create a path-to-CMK mapping configuration and use it to upload objects with the desired keys.

SQL Endpoints honor S3 encryption

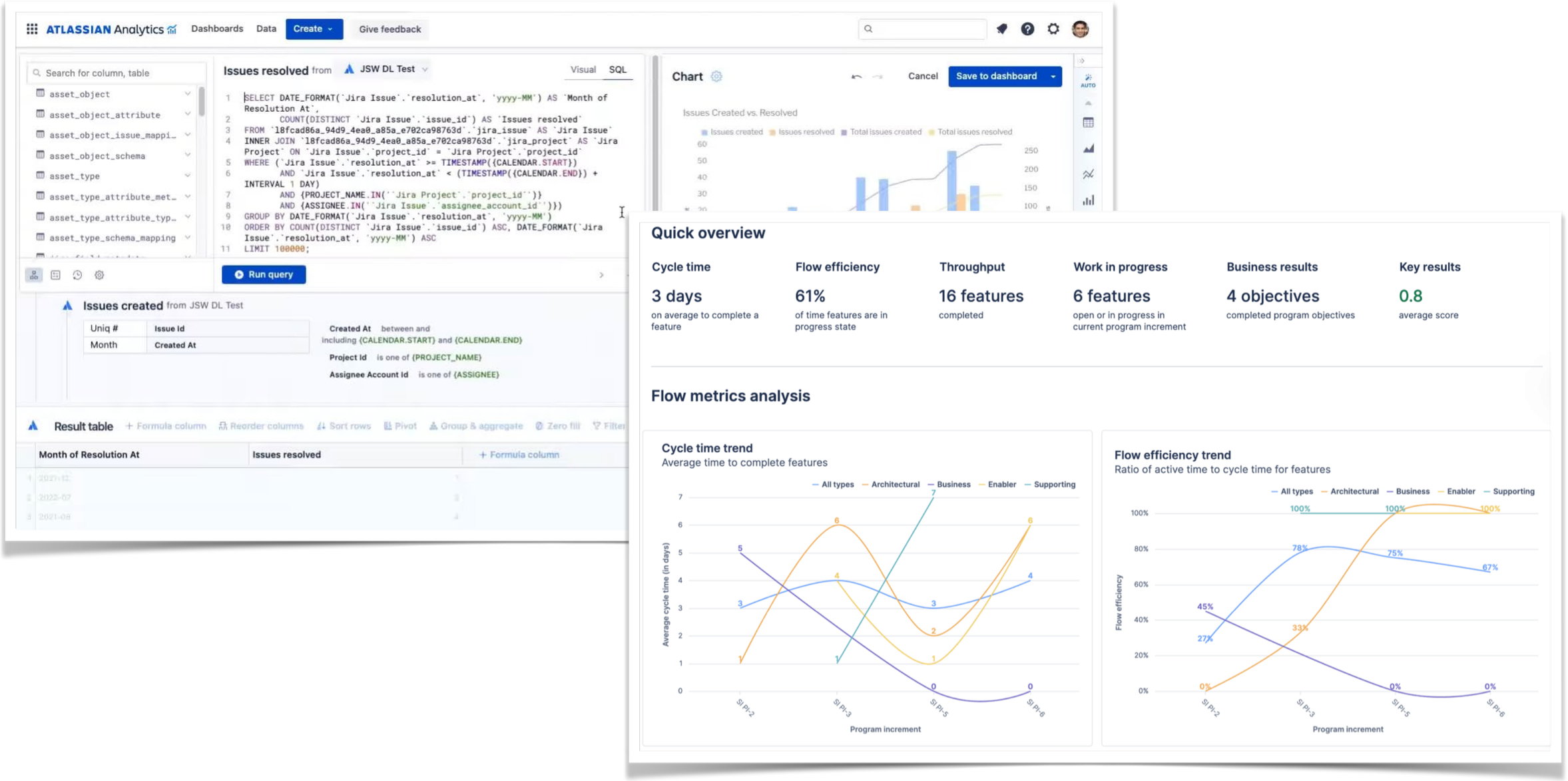
Because this encryption happens at the s3 level, the consumption of data via our SQL endpoints is transparent to users.

BYOK - S3 FS Architecture

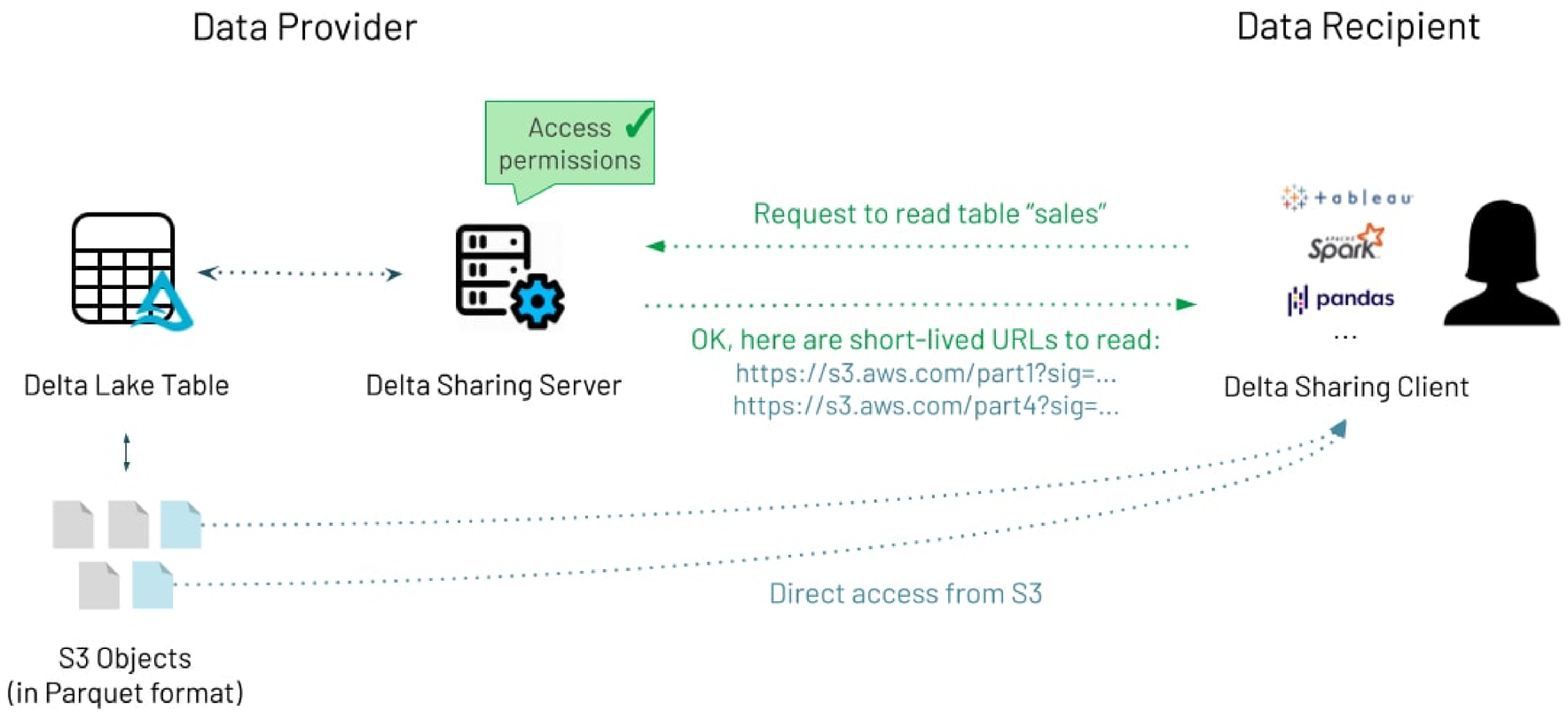


Putting all together

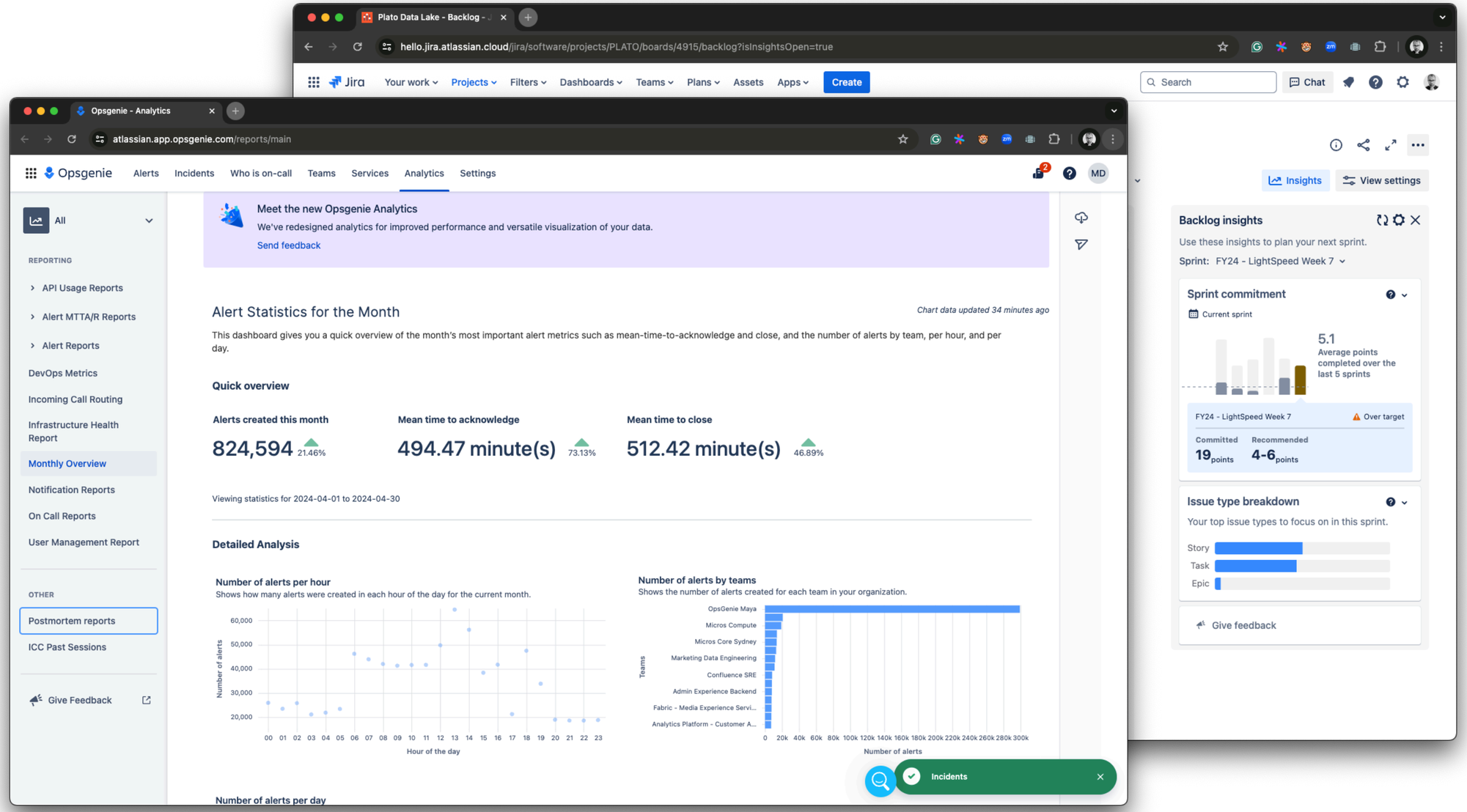
Atlassian Analytics



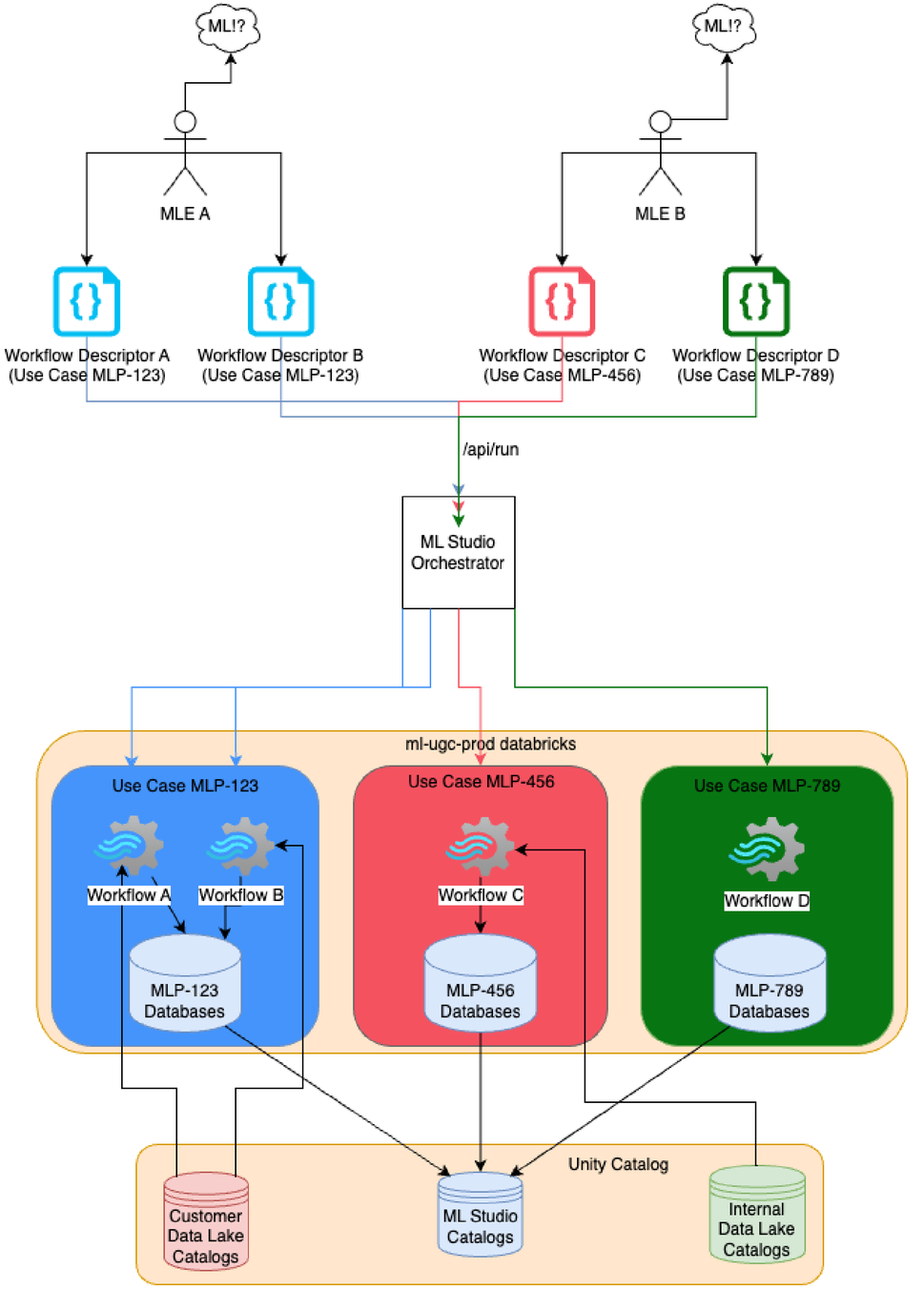
Data Exports



In-product Dashboards



ML Training


ML!?
ML!?
MLE A
MLE B
Workflow Descriptor A
(Use Case MLP-123)
Workflow Descriptor B
(Use Case MLP-123)
Workflow Descriptor C
(Use Case MLP-456)
Workflow Descriptor D
(Use Case MLP-789)
/api/run
ML Studio
Orchestrator
ml-ugc-prod databricks
Use Case MLP-123
Use Case MLP-456
Use Case MLP-789
Workflow A
Workflow B
Workflow C
Workflow D
MLP-123
Databases
MLP-456
Databases
MLP-789
Databases
Unity Catalog
Customer
Data Lake
Catalogs
ML Studio
Catalogs
Internal
Data Lake
Catalogs

Recap

01

The need for a customer data lake

Unleash many data opportunities

02

Data Replication Protocol

Logical replication approach for Data Mesh

03

Streaming Processing

Supporting realtime transformations

04

Compliance Requirements

To meet our most sofisticated customer needs

05

Delivering Value

By shipping new products and experiences

Thank you!