Migrating Our Data Pipelines: From Azure to AWS Cloud

Anurag Sharma 7 min read · Oct 4, 2025

As a data engineer deeply involved in our organization's data ecosystem, I'm excited to share insights into our latest project of October 2025: migrating our data pipelines from Microsoft Azure to Amazon Web Services (AWS). This transition isn't just a technical shift — it's a strategic move to align with evolving data sources and infrastructure needs.

In this blog, I'll walk you through the "why" behind the migration, break down our current Azure setup, outline the innovative approach we're adopting on AWS, and address a unique challenge we're preparing for.

Let's dive in!

Why Migrate to AWS?

The primary driver for this migration is a change in our upstream data providers. Our original source data team has decided to decommission their existing pipelines, and the new data source team will be delivering the same datasets with improved data collection standards directly onto the AWS platform. This shift presents an opportunity to modernize our infrastructure, reduce dependencies on outdated systems, and leverage AWS's robust ecosystem for better scalability and cost efficiency.

Understanding the Current Azure Process

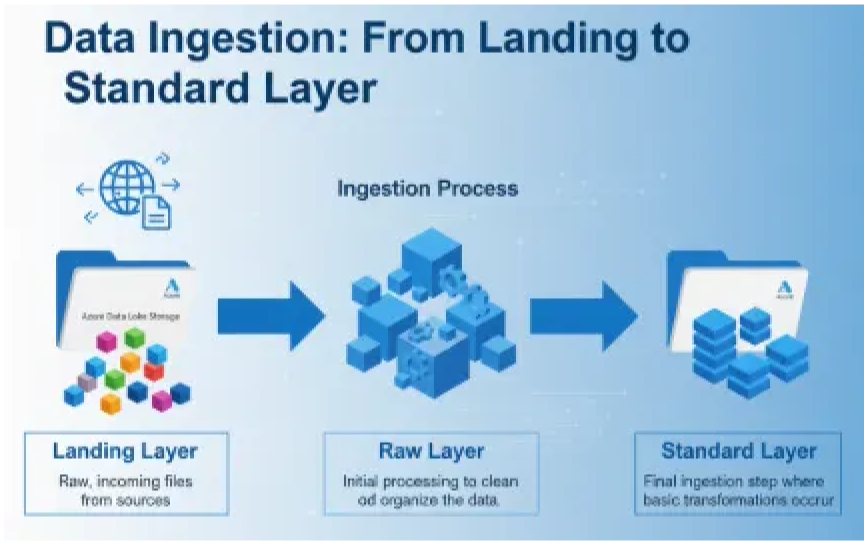
To appreciate the migration, it's helpful to first understand how things work today on Azure. Our data pipeline follows a layered architecture, with distinct teams handling different stages to ensure data quality and reliability.

Data Ingestion: From Landing to Standard Layer

Data arrives in our Azure Data Lake Storage (ADLS) in a structured progression:

- **Landing Layer**: Raw, incoming files from sources.
- **Raw Layer:** Initial processing to clean and organize the data.
- **Standard Layer**: Final ingestion step where basic transformations occur.

This entire ingestion process is managed by a dedicated team responsible for ingestion. Their role is straightforward: perform datatype conversions, remove null values and replace them with defaults (0 for NULL int Column, # for NULL String Column, etc.), and ensure the data is in a standardized format. Importantly, no complex business logic or explicit rules are applied here — it's all about preparing a clean foundation without altering the data's meaning.


Data Ingestion: From Landing to Standard Layer
Ingestion Process
Azure Data Lake Storage
Landing Layer
Raw, incoming files from sources
Raw Layer
Initial processing to clean od organize the data
Standard Layer
Final ingestion step where basic transformations occrur

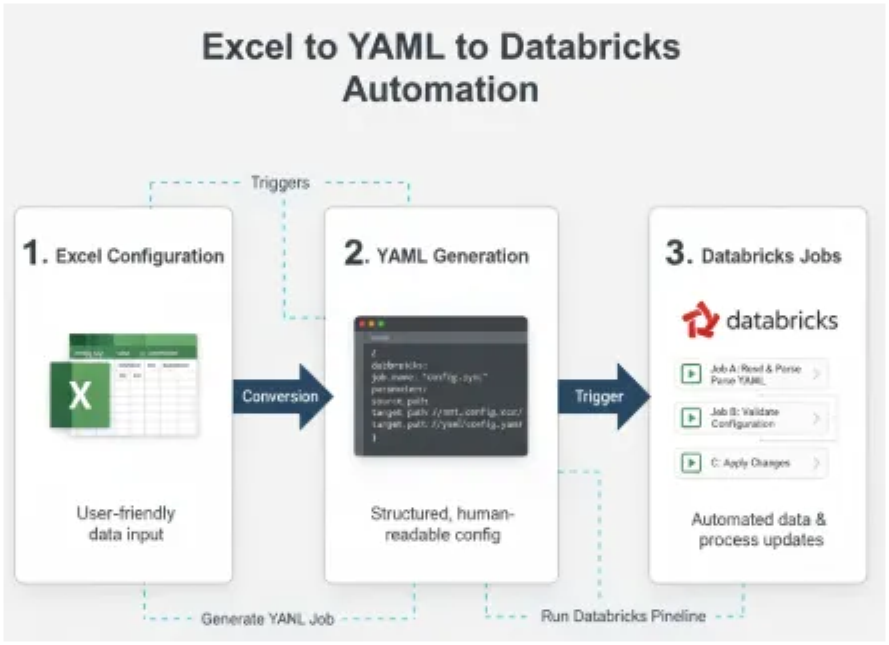
Curation: Applying Transformations and Logic

Once the data reaches the standard layer, our curation team takes over, which is my team. This is where the real magic happens: we apply business-specific transformations, integrations, and validations to curate the data for downstream use.

In Azure, these curation pipelines are powered by Databricks, orchestrated through a custom framework built by the infrastructure team. The framework relies on YAML configuration files to define and automate the ETL (Extract, Transform, Load) flows, but it all starts with Excel files. These Excel sheets serve as the primary interface for developers, capturing everything from source-to-target mappings to detailed transformation logic. From there, the Excels are converted into workflow YAML files that drive the automation.

> *Excel -> Workflow -> Airflow -> [Databricks Job1, Databricks Job2…etc.]*

This Excel-driven approach packs a lot of functionality into a familiar spreadsheet format, allowing developers to define complex workflows without diving straight into code. However, as the system has scaled, it's become quite complex to manage — thankfully, this migration gives us a chance to move away from it entirely.



To give you a clearer picture, let's break down the pros and cons of this framework:

Pros:

- **Comprehensive ETL Capabilities:** It supports full ETL processes, from extraction to loading, all configurable within Excel.
- **Direct SQL Query Writing:** Developers can embed SQL queries directly for custom transformations.
- **Data Filtering Options:** Built-in capabilities for filtering data based on conditions.
- **Integrated Data Quality (DQ) Checks:** Allows defining DQ rules via parameters or custom queries, ensuring validation at each step.
- **Seamless Physicalization:** Handles the push to MS-SQL, including creating external tables and loading into internal ones.

These features made it a powerful tool when it was first implemented, especially for teams comfortable with spreadsheets.

Cons:

- **Steep Learning Curve**: It's tough to understand at first glance — the logic is buried in sheets and formulas, not immediately obvious.
- **Version Control Challenges**: GitHub doesn't handle Excel-to-Excel comparisons natively since it's indirect "code." We rely on tools like Beyond Compare, which adds friction.
- **Cumbersome Code Reviews**: Reviewing changes means scrutinizing spreadsheets, which is error-prone and time-consuming compared to code diffs.
- **Inefficient Workflow**: The process is fragmented — Excel to YAML to Databricks, with each step often triggering separate Databricks jobs. Many of these could be consolidated into fewer, more efficient runs, but the framework doesn't lend itself to that easily.

Overall, while it served us well initially, the growing complexity has highlighted the need for a more streamlined, code-first approach — which is exactly what we're gaining with our AWS migration.

Data Quality (DQ) Checks: Ensuring Integrity

Quality is non-negotiable in our pipelines. After each ETL step in Excel, we run comprehensive DQ checks. These validate key aspects like:

- No duplication of business keys during transformations.
- Consistency across stages to quickly identify failures or anomalies.

This layered DQ approach helps pinpoint issues early — whether in integration, temp, or prepared layers — preventing cascading errors that could impact reporting.

Final Step of Azure Process: Physicalizing Data to MS-SQL for Reporting

The curated data doesn't stay in Hive forever. The last phase pushes it to Microsoft SQL Server (MS-SQL), which feeds into Tableau for visualizations and analytics used by our business teams and stakeholders.

Physicalization Process:

- Create an **external table** in MS-SQL pointing to the Hive prepared layer's path in ADLS.
- Load data from this external table into an **internal table** in MS-SQL. This "physicalizes" the data, making it fully managed within the database for faster queries and better integration with reporting tools.

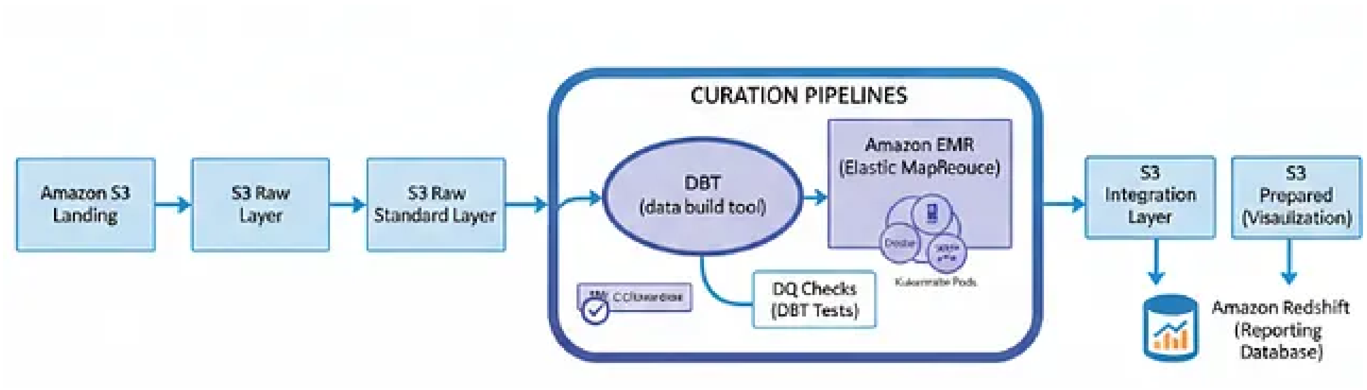
This completes the end-to-end Azure pipeline: from raw sources to insightful reports.

The AWS Migration: A Fresh, Modern Approach

Now, onto the exciting part — the migration to AWS! We're replicating the core data lake structure but introducing cutting-edge tools for efficiency and scalability. Files will now land directly in Amazon S3, mirroring the ADLS layers: landing → raw → standard.

Ingestion on AWS: Continuity with the Ingestion Team

The ingestion team retains responsibility for getting data to the standard layer. They'll handle the initial landing in S3 and perform the same basic transformations (datatype fixes, null handling) to maintain consistency. This ensures a smooth handoff without reinventing the wheel.



Curation Redefined: DBT + EMR for Transformations

Here's where things get innovative. Starting from the standard layer, we're building entirely new curation pipelines using **dbt (data build tool)** running on **Amazon EMR (Elastic MapReduce)**.

- **Why DBT?** DBT excels at managing data transformations as code, making it version-controlled, testable, and collaborative — perfect for our curation needs.
- **Execution Environment**: DBT models will run on EMR clusters, containerized with Docker and orchestrated via Kubernetes pods. This

setup provides scalability, fault tolerance, and easy deployment.

- **Layered Framework**: We've developed multiple layers within the DBT framework, each building on the last. Docker images are created for these layers and deployed onto EMR Kubernetes, allowing pods to spin up dynamically for jobs.

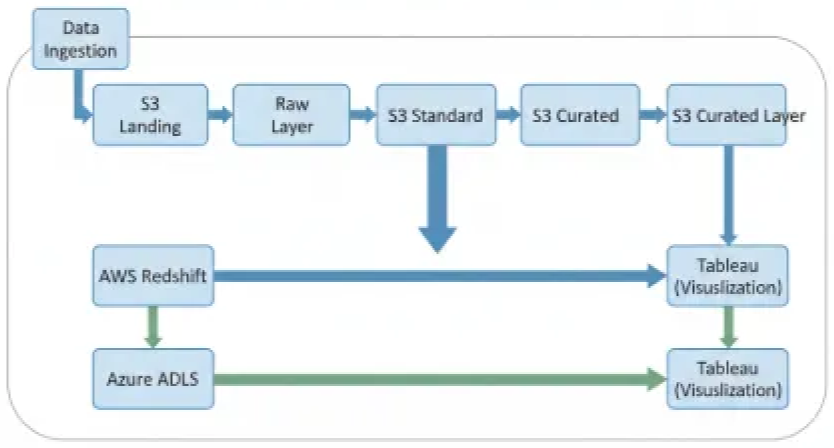
In this new flow:

- Data from the standard layer undergoes transformations and business logic in DBT models.
- Outputs are written back to S3 layers, creating integration and prepared equivalents.
- DQ checks will be integrated into DBT tests, ensuring the same rigor as before.
- Finally, we'll physicalize the prepared data into a relational data warehouse — Amazon Redshift for reporting, maintaining compatibility with Tableau.

This AWS setup not only matches Azure's functionality but enhances it with serverless-like scaling, better cost controls, and modern DevOps practices.

A Unique Challenge: Dual-Platform Reporting

As we transition to AWS, we've encountered an additional requirement that adds complexity to our migration. Our stakeholders, understandably cautious about such a significant change, want to validate the new AWS pipelines by comparing their outputs against the existing Azure-based reports. To facilitate this, we're tasked with replicating the final curated datasets from AWS S3 back to Azure ADLS. From there, we'll rebuild complete Tableau reports to allow side-by-side comparisons of "old data" (Azure) versus "new data" (AWS).



This dual-platform reporting introduces several challenges:

- **Data Replication:** We need to set up a robust process to sync the prepared datasets from S3 to ADLS without introducing errors or latency.
- **Tableau Integration:** The replicated data in ADLS must feed into Tableau seamlessly, mirroring the existing MS-SQL-based reporting structure.
- **Long-Term Maintenance**: Since the migration is expected to take over a year, maintaining this dual pipeline will become increasingly cumbersome. Eventually, we'll need to decide whether to decommission the Azure pipeline entirely or keep it as a fallback, both of which pose logistical and cost challenges.

This replication process is a temporary but critical step to build trust in the AWS pipeline. We're working closely with stakeholders to ensure transparency and accuracy during this validation phase, but we anticipate that decommissioning the Azure pipeline will be a significant undertaking once the migration is complete.

Learnings and Key takeaways:

The migration of data pipelines from Azure to AWS, driven by the source data team's shift to AWS.

Learnings include the complexity of Azure's Excel-YAML-Databricks framework, which, despite offering robust ETL and DQ capabilities, is hard to manage and review.

AWS introduces a streamlined approach with DBT and EMR, enhancing scalability and maintainability.

A significant challenge is dual-platform reporting, requiring data replication to Azure for stakeholder validation, which may complicate decommissioning.

Modern tools like DBT simplify workflows, stakeholder trust is critical, and phased migrations with robust testing are essential for success.

Data Engineering Data Engineer Databricks AWS Azure

Written by Anurag Sharma

83 followers · 3 following

Edit profile

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!