Safeguarding Your Data Pipelines: How to Prevent Failures from Data Problems

Anurag Sharma 7 min read · Aug 17, 2025

Our IoT analytics platform processes massive volumes of data generated by devices daily. These pipelines, built on Databricks and Spark, transform raw IoT data into actionable insights for customers. However, pipeline failures occasionally disrupted report delivery. While some failures stemmed from infrastructure issues (e.g., Databricks environment or Spark runtime errors), our focus was on data-related issues, such as missing files, duplicate records, or unavailable dependent tables. These issues often originated upstream, where data was generated, and impacted downstream teams responsible for report generation.

To tackle this, we needed a system to:

1. Proactively detect data quality issues before they cause pipeline failures.
2. Isolate data issues from infrastructure problems for faster resolution.
3. Notify the responsible upstream teams with actionable insights to kickstart remediation.
4. Minimize downtime and ensure daily reports are delivered on time.

Our solution leveraged Apache Airflow to create a dedicated data validation pipeline, separate from our main production pipeline, to check for data issues in a development environment. This approach allowed us to catch problems early and streamline communication between upstream and downstream teams.

Our Solution: A Data Validation Pipeline in Airflow

We designed a series of Airflow tasks to validate the quality of incoming IoT data for each day's partition. These tasks focused exclusively on data issues, leaving infrastructure-related checks (e.g., Databricks or Spark errors) to a

separate pipeline. Below is an overview of the tasks and their roles in ensuring pipeline reliability.

Task 1: Check if the Data Source File/Data is Available

The first task verifies whether the source data (e.g., raw IoT data files or database records) for the day's partition is available in the expected location (e.g., S3 bucket, database, or data lake). This task runs a query or file check to confirm the presence of data. If the data is missing, the task triggers a SQL query to log the issue and sends an email notification to the upstream team responsible for data ingestion.

Example Query (Pseudocode):

```
SELECT COUNT(*) FROM source_location WHERE partition_date = 'YYYY-MM-DD';
```

If the count is zero, the task flags the issue and includes details like the missing partition and timestamp in the email.

Task 2: Check for Duplicates on Primary or Business Key

Duplicate records in IoT data can skew analytics and lead to incorrect reports. This task checks for duplicates on the primary key (e.g., device_id) or business key (e.g., device_id + timestamp) for the day's partition. We run an SQL query to identify any duplicate entries and log the results.

Example Query (Pseudocode):

```
SELECT device_id, COUNT(*) AS count
FROM iot_data
WHERE partition_date = 'YYYY-MM-DD'
GROUP BY device_id
HAVING COUNT(*) > 1;
```

If duplicates are found, the task emails the upstream team with a sample of problematic records, enabling them to investigate and clean the data.

Task 3: Check for Dependent Tables/Data Availability From Other Teams

IoT analytics often relies on joining data from multiple tables (e.g., device metadata, user profiles, or historical data). This task ensures all dependent tables or datasets are available and populated for the day's partition. It runs checks against each dependency and flags any missing or incomplete datasets.

Example Query (Pseudocode):

```
SELECT COUNT(*) FROM dependent_table WHERE partition_date = 'YYYY-MM-DD';
```

If any dependency is unavailable, the task notifies the team responsible for that dataset, including details about the missing data.

Task 4: Validate Data Schema and Completeness

Beyond the initial checks, we added a task to validate the schema and completeness of the data. This ensures that the data conforms to the expected structure (e.g., required columns like device_id, timestamp, and sensor_value are present) and that critical fields are not null or malformed.

Example Query (Pseudocode):

```
SELECT COUNT(*) AS null_count
FROM iot_data
WHERE partition_date = 'YYYY-MM-DD'
AND (device_id IS NULL OR sensor_value IS NULL);
```

If schema issues or missing values are detected, the task logs the issue and notifies the upstream team with a breakdown of problematic records.

Task 5: Check Data Volume and Anomalies

To catch unexpected data patterns, we introduced a task to monitor data volume and detect anomalies. For example, if the number of records for a given day deviates significantly from the historical average (e.g., a 50% drop or spike), it could indicate a data ingestion issue. We use a simple statistical check to flag such anomalies.

Example Query (Pseudocode):

```
SELECT COUNT(*) AS record_count
FROM iot_data
WHERE partition_date = 'YYYY-MM-DD';
```

```
-- Compare with historical average
SELECT AVG(record_count) AS avg_count
FROM historical_counts
WHERE partition_date >= DATE_SUB('YYYY-MM-DD', 7);
```

If the record count is outside a predefined threshold (e.g., ±2 standard deviations), the task alerts the upstream team with details about the anomaly.

Task 6: Automated Email Notifications

For every task failure, we configured Airflow to run predefined SQL queries to gather detailed error information (e.g., missing files, duplicate records, or anomaly details). The results are formatted into a concise report and emailed to the respective upstream team using Airflow's EmailOperator. This ensured that the team responsible for generating the data received actionable insights to begin remediation immediately.

Additional Enhancements: Scaling and Monitoring

To make the solution scalable and maintainable, we implemented the following enhancements:

- **Dynamic Task Generation:** We used Airflow's DAG templating to dynamically generate validation tasks for multiple IoT data sources. This

allowed us to reuse the same pipeline structure for different devices and datasets without duplicating code.

- **SLAs and Alerts:** We set Service Level Agreements (SLAs) in Airflow to ensure that validation tasks are completed within a specific time window. If a task took too long (e.g., due to a large dataset), Airflow sent an alert to the operations team.
- **Monitoring Dashboard**: We integrated the validation results with a monitoring dashboard (built using tools like Grafana) to provide real-time visibility into data quality metrics, such as the number of missing files, duplicates, or anomalies detected daily.
- **Retry Logic**: For transient issues (e.g., temporary unavailability of a file), we configured tasks to retry a set number of times before failing, reducing unnecessary alerts.
- **Audit Logging:** All task results, including successful validations, were logged in a centralized audit table for historical analysis and compliance purposes.

These enhancements ensured that our solution was not only reactive but also proactive in maintaining data quality and pipeline reliability.

Impact: Before and After

The implementation of the Airflow-based data validation pipeline significantly reduced pipeline failures and improved report delivery reliability. Below are the key metrics demonstrating the impact:

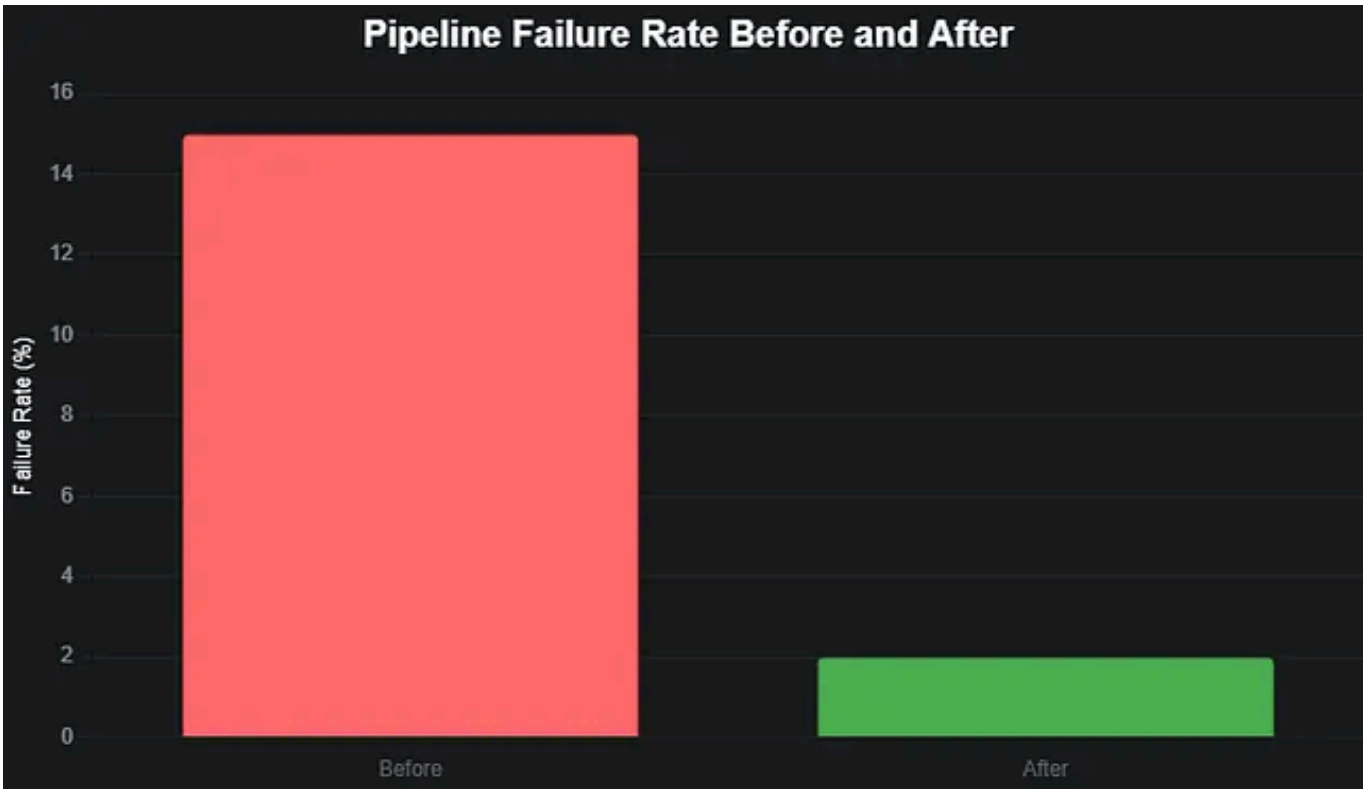
Before Implementation

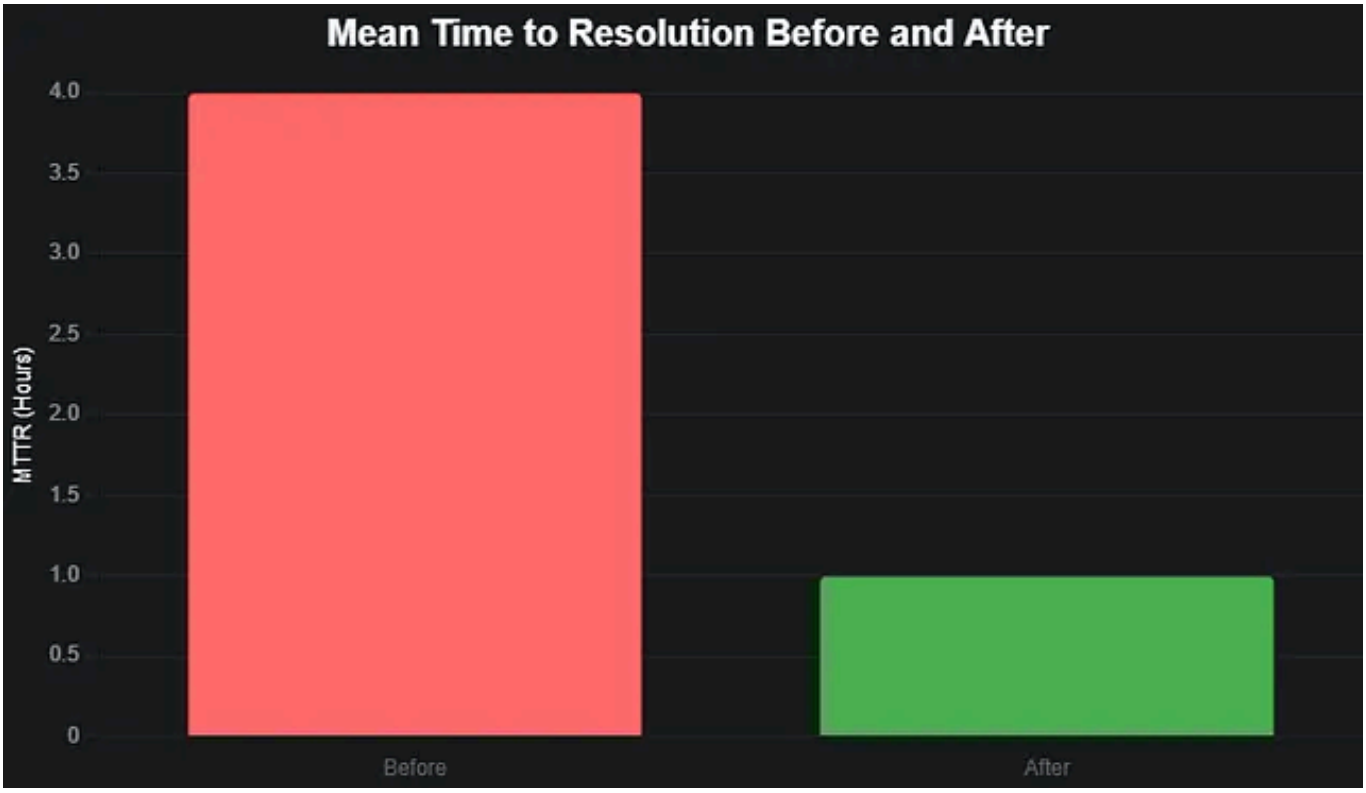
- **Pipeline Failure Rate:** Approximately 15% of daily pipeline runs failed due to data issues, leading to delayed reports.
- **Mean Time to Resolution (MTTR):** Data-related issues took an average of 4 hours to identify and resolve, as teams manually investigated failures.
- **Customer Report Delays:** 10% of customers experienced delayed reports, impacting satisfaction.
- **Team Collaboration:** Limited visibility into upstream data issues caused delays in communication and resolution.

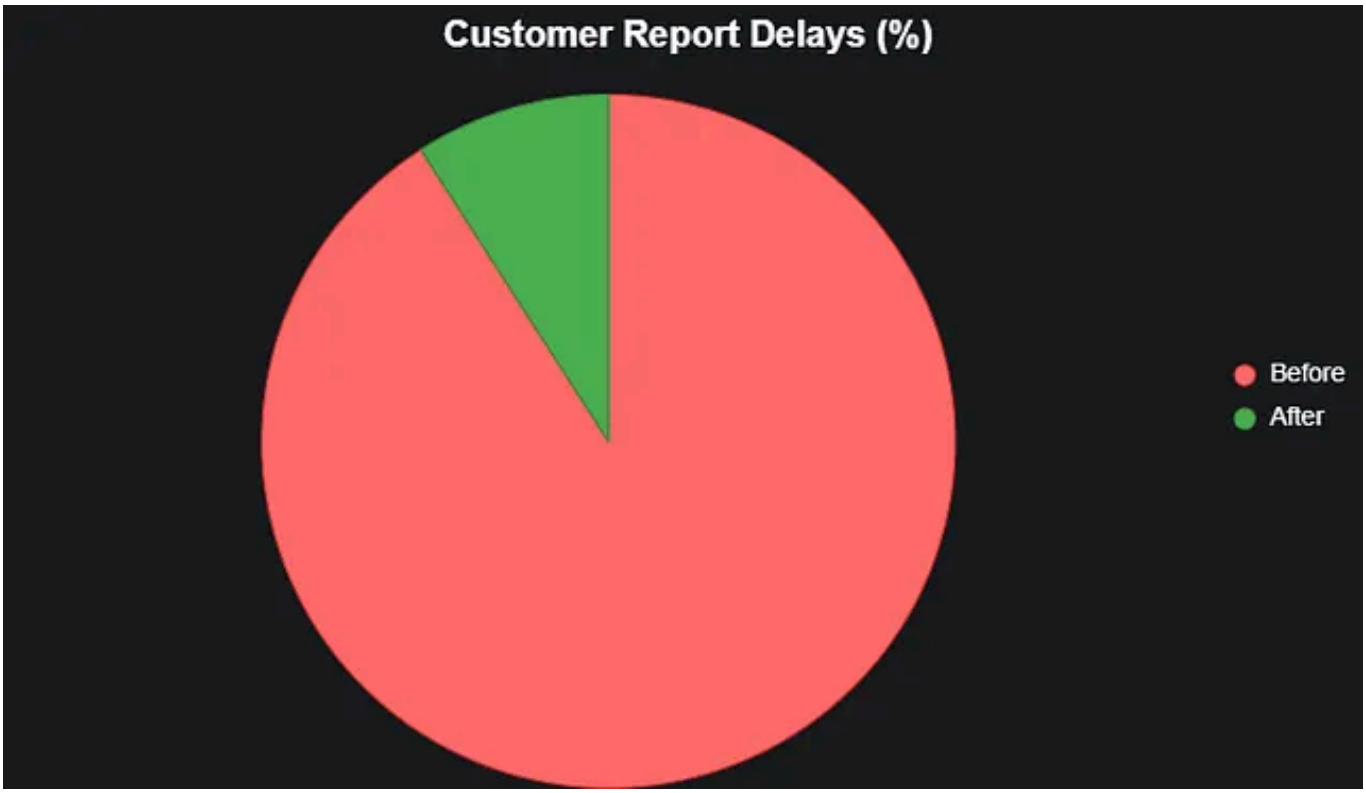
After Implementation

- **Pipeline Failure Rate:** Reduced to 2% (only infrastructure-related issues remained, as data issues were caught early).
- **MTTR:** Dropped to 1 hour, as automated notifications provided actionable insights to upstream teams immediately.
- **Customer Report Delays:** Reduced to less than 1%, ensuring near-100% on-time report delivery.
- **Team Collaboration:** Improved handoff between upstream and downstream teams, with clear error reports reducing back-and-forth.

The charts below visualize the before-and-after impact of our solution.


Pipeline Failure Rate Before and After
Failure Rate (%)
16
14
12
10
8
6
4
2
0
Before
After


Mean Time to Resolution Before and After
MTTR (Hours)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
Before
After



Key Benefits and Lessons Learned

1. **Proactive Issue Detection:** By running validation tasks in a development environment, we caught data issues before they impacted production pipelines.
2. **Faster Resolution:** Automated notifications with detailed error reports enabled upstream teams to start fixing issues immediately, reducing MTTR.
3. **Improved Customer Satisfaction:** Near-100% on-time report delivery boosted customer trust and engagement.
4. **Enhanced Collaboration:** Clear handoff of issues with actionable insights improved communication between upstream and downstream teams.
5. **Scalability:** The Airflow pipeline's modular design allowed us to scale validations across multiple IoT data sources efficiently.

Lessons Learned:

- **Early Validation is Key:** Validating data quality in a separate pipeline before production processing prevents costly failures.
- **Automation Saves Time:** Automated checks and notifications significantly reduce manual effort and human error.
- **Clear Communication:** Detailed error reports are critical for enabling upstream teams to take swift action.
- **Continuous Improvement:** Regular reviews of validation results helped us refine checks and thresholds over time.

Conclusion

Our Airflow-based data validation pipeline transformed the reliability of our IoT analytics platform. By proactively detecting and addressing data quality issues, we reduced pipeline failures, minimized resolution times, and ensured consistent report delivery to customers. The before-and-after metrics highlight the tangible impact of our solution, with failure rates dropping from 15% to 2%, MTTR shrinking from 4 hours to 1 hour, and customer report delays nearly eliminated. This approach not only strengthened our analytics operations but also fostered better collaboration across teams, proving that a well-designed data validation strategy is essential for mission-critical pipelines.

Spark Databricks Big Data Data Analytics Automation