Open in app ↗

Medium Search Write

Get unlimited access to the best of Medium for less than $1/week. Become a member


# How We Slashed Incidents and Saved Time with Simple Data Engineering Hacks

Anurag Sharma 5 min read · Jun 5, 2025

Imagine this: you're a data engineer in a fast-paced analytics company, juggling 400+ daily data jobs like a circus performer spinning plates. The stakes are high — reporting and analytics are very important to the business, and every failed job feels like burning money and resources. Worse, when a job crashes in production, the dev team is not always around to swoop in and save the day. Sound familiar? That was our reality — until we devised a few simple, game-changing solutions that reduced incidents, saved time, and allowed us to focus on building cool stuff instead of firefighting.


**The Problem: Chaos in the Data Pipeline**

In an analytics company, time is money, and data is the fuel that keeps the engine running. Our team managed a sprawling pipeline of over 400 daily jobs, loading data into our systems to power critical reports and analytics.

But when a job failed in production, it was chaos. Upstream issues, environment glitches, or sneaky data problems could bring everything to a halt. And if the dev team was not around for any reason, we were stuck. Hours, sometimes days, were wasted just figuring out *why* a job failed. Was it a Spark issue? A data quality issue from the source? Or a business logic bug? The lack of clarity was killing us.

**The Lightbulb Moment: Simplify and Automate**

One day, after yet another late-night debugging session, we had enough. Our goal was clear: reduce incidents, minimize dependency on developers for initial triage, and make debugging so easy even an intern could handle it. The solution? A mix of straightforward tools and automation that tackled the most common failure scenarios head-on.


Here is how we did it.

**Step 1: The Confluence Lifesaver**

First, we tackled environment-related issues, like memory overflows or task failures in Apache Spark. These were the simple problems that often had predictable fixes. We created a **Confluence page** that became our step-by-step troubleshooting guide for common issues.

For example:

• **Memory issue?** Restart the Airflow job. Still failing? Check the logs for specific error codes.

• **Task failure?** Retrigger the job. If it flops again, escalate to an **incident**.

This wasn't rocket science — it was a living, breathing document that any DevOps engineer could follow without waking up a developer at 2 a.m. The result? Fewer incident tickets and faster resolution for straightforward issues. Plus, the Confluence page became a knowledge hub, updated with new fixes as we learned from each failure. It was like giving our team a cheat sheet for production chaos.

**Step 2: The Redshift Table That Changed Everything**

Next, we tackled the trickier beast: data issues. When a job failed, it was often because of bad or missing data from an upstream source. But pinpointing *which* source was the culprit could take hours of detective work. Enter our secret weapon: a **Redshift table** designed to make data issue debugging a breeze.

We built a table with three key columns:

• **job_name**: The name of the job that failed.

• **upstream_dependencies**: The source systems or tables the job relied on.

• **fails_query_check**: A pre-written Spark SQL query to check if the failure was due to a data issue (e.g., missing rows, null values, or schema mismatches).

We automated the process by building an **Airflow job** that would:

1. Pull the relevant fails_query_check queries from the Redshift table.
2. Run them against the source data for the failed job.
3. Spit out a clear verdict: “Upstream data issue detected” or “All clear, check the job logic.”

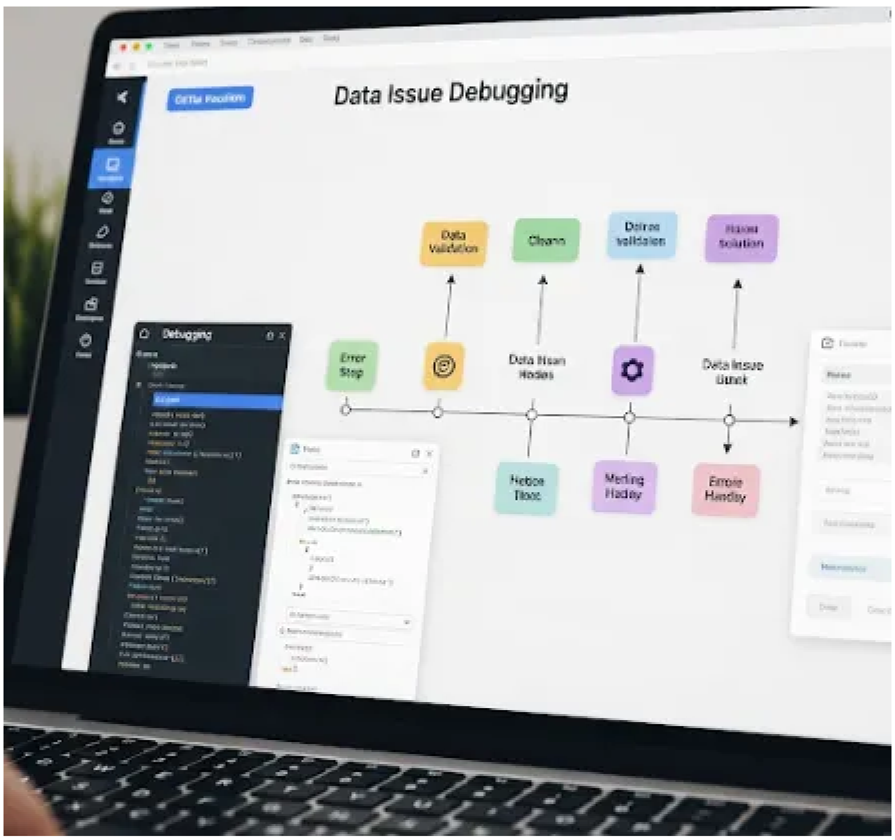
This meant a DevOps engineer could quickly determine if the failure was due to bad data from a source team (which they would escalate) or something else, like a code bug. No more developers spending half their morning scanning through logs to realize the issue was out of their control.

**The Impact: Less Chaos, More Clarity**

The results were nothing short of magical. By implementing these simple tools — a Confluence page for environment issues and a Redshift table with automated query checks — we slashed our incident tickets by a whopping 40%. Developers no longer had to play Sherlock Holmes every time a job failed. Instead, they were only pulled in for true business logic issues, which we tracked as stories in our project management tool.

Here is what we gained:

- **Time savings:** Debugging time dropped from hours to minutes for most failures.
- **Reduced incidents:** Clear triage steps meant fewer unnecessary escalations.
- **Empowered DevOps:** Our ops team could handle initial investigations without developer hand-holding.
- **Happier teams:** Source teams were notified faster about data issues, and developers could focus on building features instead of firefighting.




## Why It Worked (and Why You Should Try It)

We just used tools we already had — Confluence, Redshift, and Airflow — in clever ways. The Confluence page gave us a centralized knowledge base, while the Redshift table and Airflow automation turned data issue debugging into a self-service process.

Here is the playbook:

**1. Document common fixes:** Create a Confluence page (or any wiki) with step-by-step guides for frequent issues like memory errors or task failures.

**2. Build a failure diagnostic table:** Store job metadata and diagnostic queries in a database like Redshift or Snowflake. Make it easy to query and maintain.

**3. Automate triage:** Use a tool like Airflow to run diagnostic queries automatically and report results.

**4. Iterate and improve:** Keep your documentation and table updated as you learn from new failures.

Data Engineer Data Engineering Data Analysis Data Analytics Automation

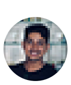
## Written by Anurag Sharma

Edit profile

82 followers · 3 following

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

## No responses yet



Anurag Sharma him/he

What are your thoughts?