SM

DOMAIN METHODS
DATA AND ANALYTICS EXCELLENCE

Building a Modern Data Foundation with dbt: A Practical Guide

Jason B. Hart Data engineering February 12, 2026

▼ Table of Contents

SM

I've helped dozens of companies build data foundations. The ones that fail almost never fail because they picked the wrong tool. They fail because nobody agreed on what the foundation was supposed to do.

This guide covers the decisions that actually matter: how to pick a warehouse, why dbt wins as the transformation layer, which architecture patterns work for mid-size teams, and how to migrate off legacy ETL without losing your mind. If you're a data

SM

Why Most Data Foundations Fail (Hint: It's Not the Technology)

The number one reason data foundations fail is that they're treated as pure engineering projects. Someone decides "we need a modern data stack," the team evaluates tools for two months, picks a warehouse, implements dbt, and three months later you have a technically competent warehouse that nobody outside the data team trusts or uses.

The technology worked. The project still failed.

The Real Problem: Nobody Defined "Trusted Data"

Before you write a single line of SQL, you need answers to questions that have nothing to do with technology:

- Which decisions are people making today with bad data or gut feel?
- Who will use this data, and in what tools?
- What does "correct" mean for your key metrics — and who has the authority to define it?
- When stakeholders say they "don't trust the numbers," what specifically broke their trust?

These aren't technical questions. They're organizational ones. And if you skip them, you'll build pipelines that are technically flawless and operationally useless.

Start With a Business Outcome, Not a Technology Goal

"Implement dbt" is not a business outcome. "Enable the finance team to close the books in 3 days instead of 10" is a business outcome. "Give the marketing team channel-level ROAS they trust enough to reallocate spend" is a business outcome.

Every successful data foundation I've built started with a

SM

the technology decisions. The unsuccessful ones started with the technology and hoped the business value would emerge.

It didn't.

Choosing Your Warehouse: BigQuery vs. Snowflake vs. Databricks

I'm going to be direct here because I think the "it depends" answer that most consultants give is a cop-out. For mid-size SaaS companies, there are three credible choices and they each have a clear sweet spot.

BigQuery: The Simplest Path to Analytics

If your team wants to focus on analytics and reporting — not managing infrastructure — BigQuery is the answer. Serverless compute, no cluster management, deeply integrated with the GCP ecosystem, and the pricing model (on-demand or flat-rate) makes costs predictable.

BigQuery is where I start most engagements for teams that are primarily doing analytics and BI. The learning curve is gentle, the tooling ecosystem is mature, and you're not paying for capacity you're not using.

Snowflake: Multi-Cloud Flexibility

Snowflake's strength is that it runs identically on AWS and GCP, giving you flexibility if your infrastructure spans both. The separation of storage and compute is genuinely useful for teams with bursty workloads — you spin up a warehouse for heavy transforms, shut it down when you're done, and only pay for what you used.

If your company has strong opinions about cloud provider neutrality or you're running multi-cloud infrastructure, Snowflake is the right call.

SM

Databricks: The Data Science Play

If your roadmap includes ML, data science, or advanced analytics beyond BI dashboards, Databricks is the platform to bet on. The lakehouse architecture, Unity Catalog for governance, and native support for Python and Spark make it the strongest choice for teams that need both analytics and data science in one platform.

Databricks has a steeper learning curve than BigQuery or Snowflake for pure analytics use cases. That's the trade-off. But if you know you're heading toward ML workloads, starting on Databricks avoids a painful migration later.

A Word on Azure

I'll say what most consultants won't: Azure is usually not the right ecosystem for mid-size SaaS analytics. Azure Synapse is complex, the tooling ecosystem is thinner, and the data engineering community has less momentum there. If you're already deep in the Microsoft ecosystem for other reasons, I understand the pull. But if you're choosing fresh, GCP or AWS with BigQuery, Snowflake, or Databricks will get you further, faster, with less friction.

This isn't an anti-Microsoft stance. It's a practical one. The open-source tooling, community support, and integration ecosystem for GCP and AWS is significantly stronger for the kind of work we're talking about.

dbt as the Transformation Layer: Why It Wins

dbt isn't just a tool — it's a philosophy about how data transformations should work. And that philosophy is right.

Version-Controlled Transformations

Before dbt, transformation logic lived in stored procedures, SSIS

SM

through menus, and hoping you didn't break something. There was no code review. No pull requests. No history of who changed what and why.

dbt puts all transformation logic in version-controlled SQL files. Every change goes through a pull request. Every decision has a commit history. Your transformation layer gets the same engineering rigor as your application code.

This alone is worth the switch.

Testing as a First-Class Citizen

dbt's testing framework is deceptively simple — **`not_null`**, **`unique`**, **`accepted_values`**, **`relationships`** — but it solves the fundamental problem of data trust. Every model can ship with assertions about what the data should look like. When those assertions fail, you know before your stakeholders do.

I've seen companies go from "the revenue number was wrong for two weeks and nobody caught it" to "we caught a source schema change within an hour because a test failed." That's the difference between a data team that's trusted and one that isn't.

Documentation That Lives With the Code

dbt generates documentation from your **`schema.yml`** files and serves it as a browsable website. Descriptions, column definitions, lineage graphs — all generated from the same codebase that produces your models.

This matters because documentation that lives in a separate wiki gets stale. Documentation that lives with the code gets updated when the code changes, because it's part of the same pull request.

Why It Beats the Alternatives

Stored procedures are impossible to test systematically, difficult to version control, and vendor-locked. SSIS packages are brittle,

SM

are flexible but create maintenance nightmares — every script is a snowflake (lowercase) that only its author understands.

dbt isn't perfect. But for 90% of analytics transformations — staging, cleaning, joining, aggregating, building business logic — it's the best tool available. The 10% where it doesn't fit (streaming, real-time, heavy ML feature engineering) is a topic for another post.

Architecture Patterns That Scale (and Ones That Don't)

The Medallion Architecture: Bronze, Silver, Gold

The most reliable architecture pattern I've seen for mid-size teams is the medallion (or multi-layer) approach:

- **Bronze (staging):** Raw data from sources, minimally transformed. Renamed columns, cast data types, deduplicated. This is your safety net — you can always rebuild everything downstream from here.
- **Silver (intermediate):** Cleaned, joined, business logic applied. This is where you resolve entities, apply business rules, and create the building blocks for analysis.
- **Gold (marts):** Business-ready tables organized by domain — finance, marketing, product, sales. These are what your analysts and BI tools hit directly.

This isn't the only pattern that works, but it's the one that works most consistently for teams between 2 and 15 data people. It's simple enough to understand, structured enough to govern, and flexible enough to evolve.

What Doesn't Scale

Over-normalization. If your silver layer looks like a relational database with 47 tables and a diagram that requires a poster-sized printout, you've over-engineered it. Denormalize for

SM

Premature optimization. Don't build incremental models until you have a performance problem. Don't partition tables until scan costs are actually high. Don't implement complex caching strategies for a warehouse that processes 10GB. Build the simplest thing that works and optimize when the data tells you to.

Building for scale you don't have. If you're processing 50 million rows, don't architect for 50 billion. You'll add complexity, slow down development, and solve problems you don't have yet — while ignoring problems you do have.

The 80/20 rule applies to architecture: get the fundamentals right (clean staging, clear business logic, well-organized marts) and you'll handle 80% of what the business needs. Optimize the remaining 20% when it actually becomes a bottleneck.

Data Governance That People Actually Follow

Most governance programs fail because they're designed by compliance teams who've never built a pipeline. The result is 50-page policy documents that nobody reads, access controls that block legitimate work, and a governance "committee" that meets quarterly to discuss why nobody follows the governance framework.

Here's what actually works.

Clear Ownership

Every model in your dbt project should have an owner. Not a team — a person. Someone who is responsible for its accuracy, its documentation, and its maintenance. Put it in your `schema.yml`. Make it visible.

When something breaks, "the data team owns it" means nobody owns it. "Sarah owns the revenue mart" means Sarah gets paged, Sarah fixes it, and Sarah has the authority to make

SM

Automated Testing Over Manual Review

Manual data quality reviews don't scale and they don't stick. Automated tests that run on every dbt build do both.

At minimum, every model should have: primary key uniqueness, not-null constraints on critical fields, and at least one business logic assertion (e.g., revenue should never be negative, dates should be within a reasonable range). This takes 15 minutes per model to set up and saves hours of debugging later.

Sensible Naming Conventions

Pick a convention and enforce it. I prefer: `stg_` for staging, `int_` for intermediate, `fct_` for fact tables, `dim_` for dimensions. But the specific convention matters less than consistency. If half your models use `stg_` and the other half use `staging_` and a few use `raw_`, you don't have a convention — you have chaos.

Document the convention in your project's README. Enforce it in code review. That's it. No 50-page naming standard required.

Documentation That Earns Trust

Good documentation answers the question a stakeholder is actually asking: "What does this number mean, and can I trust it?"

For every gold-layer model, document: what it measures, what it includes and excludes, where the source data comes from, when it refreshes, and who owns it. That's five things. It takes 10 minutes. And it's the difference between a stakeholder who trusts the dashboard and one who opens a Slack thread asking "is this number right?"

The Migration Playbook: Legacy ETL to Modern Stack

SM

the highest-ROI projects you can do. It's also one of the easiest to botch. Here's the playbook.

Step 1: Audit What You Have

Before you migrate anything, document what exists. Every pipeline, every schedule, every dependency. You'll be surprised — most legacy ETL environments have pipelines that nobody remembers building, data flows that feed reports nobody reads, and critical dependencies that aren't documented anywhere.

Map source systems, transformation logic, output targets, and consumers. If you can't explain what a pipeline does and who uses its output, flag it for investigation before migration.

Step 2: Map Business-Critical Pipelines

Not all pipelines are created equal. Identify the ones that feed revenue reporting, board decks, operational dashboards, and regulatory requirements. These are your priority-one migrations. Everything else can wait.

I typically sort pipelines into three tiers: must-migrate (business-critical, actively used), should-migrate (useful but not urgent), and investigate (unclear value, possibly unused). Start with tier one.

Step 3: Migrate in Phases

Don't try to migrate everything at once. Pick one business-critical pipeline, rebuild it in dbt, validate the output against the legacy system, and put it in production. Then do the next one.

Run legacy and modern pipelines in parallel during the transition. Compare outputs daily. When the modern pipeline has produced consistent, validated results for two to four weeks, you can start routing consumers to the new source.

Step 4: Test Relentlessly

SM

results with documented reasons for any differences.

“The new numbers are different” is the fastest way to kill stakeholder trust in a migration. When the numbers are different, you need to explain why, and ideally show that the new numbers are more correct (e.g., “the legacy pipeline was double-counting refunds”).

Step 5: Sunset Legacy Systems Deliberately

This is where most migrations stall. The new stack is running, but nobody turns off the old one because “what if we need it?” Six months later, you’re maintaining two systems.

Set a sunset date for each legacy pipeline before you start the migration. Communicate it. Stick to it. If consumers haven’t transitioned by the sunset date, escalate — don’t extend.

When to Build vs. When to Call for Help

Most data teams can build a solid dbt-based data foundation themselves. But there are situations where bringing in outside expertise saves time, money, and a lot of frustration.

Signs You Need Help

Your team is underwater with maintenance. If your data engineers are spending more than half their time fixing broken pipelines instead of building new capabilities, they don’t have capacity to design and execute a foundation overhaul. You need someone to come in, build the foundation, and hand it back.

You’re evaluating platforms for the first time. The difference between a good warehouse choice and a bad one is measured in years of productivity. If nobody on your team has built on BigQuery, Snowflake, and Databricks before, you’re making a high-stakes decision with limited information.

SM

You need to move fast. A 6-month learning curve for dbt best practices, warehouse optimization, and governance frameworks is reasonable for a team learning on the job. If you need results in 6 weeks, not 6 months, experience compresses that timeline dramatically.

Your stakeholders have already lost trust. Rebuilding trust is harder than building it the first time. An outside perspective can help identify what broke, fix it credibly, and re-establish confidence in a way that an internal team — the same team associated with the broken numbers — sometimes can't.

What We Do

We build data foundations using dbt, open-source tools, and modern cloud warehouses on GCP and AWS. We scope it, build it, test it, document it, and hand it off so your team owns it completely. No ongoing dependency, no retainer lock-in.

If any of this resonated — if you're staring at a legacy ETL system, evaluating warehouses, or trying to figure out why your team built a modern stack that nobody trusts — we should talk.

Talk To Us About Your Data Foundation

Tags : Dbt | Data foundation | Big query | Snowflake | Databricks | Data governance

Share :

Get posts like this in your inbox

Subscribe for practical analytics insights — no spam, unsubscribe anytime.

Book A Discovery Call

SM

Related Posts

Book A Discovery Call

SM

5 dbt Implementation Mistakes That Kill Data Trust

Jason B. Hart Data engineering

dbt changed the game for analytics engineering. But like any powerful tool, it can create as many problems as it solves — especially when the implementation is rushed or the team doesn't have a clear plan. Here are the five mistakes I see most often when companies implement dbt, and what to do instead. 1. No Testing Strategy This is the most common and most damaging mistake. Teams build dozens of models but write zero tests. Then they wonder why stakeholders don't trust the numbers.

Read More

SM

DOMAIN METHODS
DATA AND ANALYTICS EXCELLENCE

Book A Discovery Call