Accelerators for Cloud Data Platform Transitions

How to prepare for a new analytics workflow

dbt Labs

Table of Contents

About this guide

Moving to a cloud data platform is an exciting decision, but can also be a challenging project for larger organizations. Gaining stakeholder alignment, considering team redesign, and strategizing new data management paradigms aren't flip-of-the-switch events.

dbt Labs has helped hundreds of enterprise organizations accelerate their move to a more modern approach to analytics, without losing momentum. We believe the following field-tested approach will make a meaningful difference in reducing the burden of your own transition.

In this guide, you'll learn:

- How to build a data delivery pipeline that matches your new speed of business
- How to enable data developers and consumers to confidently self-serve insights
- Why governance and management practices change through cloud migration, and how to respond

Of course, we know this guide might not answer all your questions! If you find yourself needing more guidance, the dbt Labs' solution architects are standing by to help.

We also recommend visiting the ~30,000 member dbt Community Slack, where several cloud platform channels have been created to share best practices and advice. Jump into the #advice-dbt-for-beginners and #advice-data-modeling channels when you're ready to get started.

About dbt Labs

Since 2016, dbt Labs has been on a mission to help analysts create and disseminate organizational knowledge. dbt Labs pioneered the practice of analytics engineering, and developed the primary tool in the analytics engineering toolbox, dbt.

dbt enables anyone who knows SQL to transform data safely and efficiently using modular code, version control, testing, and automated lineage and documentation. We're fortunate to have seen more than 27,000 data practitioners join the dbt Community to support one another in pursuit of analytics work that is more accessible and reliable.

Build a data delivery pipeline that matches your new speed of business

Prior to the existence of cloud data platforms, organizations had a finite amount of compute and storage that carried an inverse amount of data management overhead. Provisioning infrastructure, weighing costs — transformation work was a delicate practice, owned only by those with the requisite permissions.

The arrival of cloud platforms made this work cheaper and easier, but no more accessible, coordinated, or reliable. While moving to the cloud is an enormous first step, deciding how your team collaborates on data transformation development is the next most important step to take.

> "The data transformation process is complex enough that you need a framework to manage it. Otherwise, the team spends too much time reactively chasing bugs and figuring out what scripts should be running first, second, third and so on."

Matt Winkler,
Solutions Architect, dbt Labs

Reduce code and streamline troubleshooting with modularity

Traditional on-premise data platforms often rely on monolithic SQL stored procedures, or proprietary transformation solutions that make troubleshooting a week-long affair for a small number of individuals. Cloud storage removes compute limitations that can contribute to additional delays, but doesn't solve for the sheer volume of code, and time spent developing and troubleshooting it.

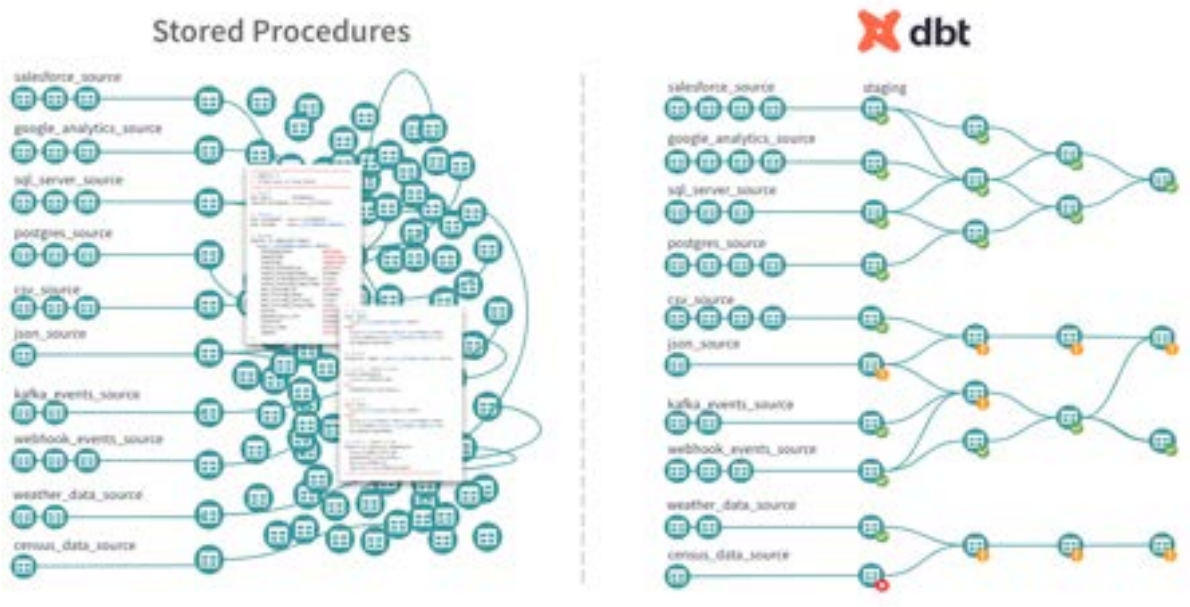
The software engineering best practice of modularity helps expedite code writing and debugging, while also making this work more readily understood by other teammates. Modularity in data transformation makes code referenceable and extensible so that developers don't start from scratch every time. Modular code:

- Reduces total code surface area and thus opportunity for error
- Keeps your code DRY (following Don't Repeat Yourself best practices)
- Improves code consistency across a growing organization
- Helps you take advantage of the scalability of a cloud data platform by decreasing the time it takes to refresh your data when code blocks are run in parallel

> "..the goal here ultimately with the way that you define models is that you want to enhance the modularity…when we enhance that modularity, we're expediting our development cycles because we only run the related snippets of SQL code. And we're maintaining code dryness, where if something is defined in one place I'm not having to redefine that over and over again because I'm utilizing a single model that has that definition."

Anais Vaillant,
Solutions Architect,
dbt Labs

Once modular code becomes the norm, your team can focus on optimizing discrete parts of the data graph—another best practice borrowed from software engineering resulting in more efficient and timely data refresh runs that save your organization compute cycles and help deliver insights faster.



Untangle the mess of stored procedures using modular, referenceable, version controlled code

dbt helps promote modularity by leveraging packages (portable blocks of code for common use cases), reference function for graphing relations amongst models, and Jinja templating for SQL.

If you're not sure where to start on your own journey to modularity, the "Modular Data Modeling Technique" chapter in dbt Labs' Analytics Engineering Guide is a good place to start. Sold on the method but worried that the volume of stored procedures your organization relies on might prevent a move to this way of work? The free dbt Learn course, Refactoring for Modularity provides a step-by-step transition plan. Or, read and hear how JetBlue converted 26 data sources with 1200 models to modular dbt models.

Make changes quickly (and safely) with data lineage

When storage and compute costs pushed transformation outside the warehouse, tracing data from source to report was a challenge. Cloud data platforms' elastic storage and flexible compute bring transformation work inside the data management layer, enabling greater transparency in how data evolves over time.

Unfortunately, even with cloud data platforms, this work is still largely a manual effort. To build a full understanding of relationships amongst models in your data platform,

you need to run your full data load job, which writes metadata about your data model to your data platform. Then you have to push that metadata into another tool to visualize the relationships or develop complex queries to view lineage amongst objects in your model. This is not an insignificant amount of work!

Tools like dbt can reduce this burden by automating the creation of lineage as you code. Leveraging the reference function generates your project's model interdependencies, or DAG (directed acyclic graph), which dbt uses to build tables in order in your cloud data platform.

The dbt lineage graph shows how data is transformed from source to downstream applications

dbt lineage graph helps visualize the movement of your data from source to destination.

Creating a relationship graph that shows how your data evolves over time is a critical step in reducing time to audit, investigate, and troubleshoot your data models.

Test data as it's transformed to improve quality and trust

Data tests help identify data quality and model soundness, but have been overlooked by most legacy on-premise solutions due to the time and resources they require. In legacy environments, tests use scarce compute resources, which means they're often run post-refresh, or not at all. If data issues are identified, models are repaired and re-run in their entirety before results can be delivered to end-users—eroding confidence in data timeliness and quality.

Cloud data platforms provide scalable compute resources which enable data testing to become a first-class part of data transformation work. However, many data teams still struggle to understand where and how testing makes the most sense.

"When I led a machine learning team, by far the number one headache that ate up extra time was the lack of adequate data testing. Input data schemas changed because our web app team dropped fields unexpectedly. We'd started to get character strings that exceeded the limits our models had seen before. Distributions changed with no way to inspect or audit those changes.

We tried to build custom code to capture these situations as they arose, but that was playing whack-a-mole. We needed a testing framework that could evolve along with our data and had a transparent relationship to the data powering our models."

Matt Winkler,
Solutions Architect, dbt Labs

The following tips from the dbt Analytics Engineering Guide might help you rethink your cloud data platform testing strategy:

1. **Run tests during the transformation process.**
   Running in parallel enables data teams to troubleshoot issues faster.

2. **Output to SQL to increase transparency.**
   Tests can be written to output the faulty records so your team knows exactly where to start their investigation.

3. **Start with reactive testing.**
   Once you get into a good cadence it's easy to get carried away. Start with tests for uniqueness, not nullness, and accepted values.

4. **Get proactive.**
   As your testing framework expands, it's important to consider things like data freshness and domain-specific problems that can only be solved proactively.

5. **Build a culture of testing.**
   Is every developer leveraging tests? Is there a team-wide process for resolving issues? Are stakeholders aware of these efforts?

Tools like dbt Cloud make it even easier to build and schedule tests that run during the data refresh process, eliminating downtime and improving data delivery timelines. In addition to custom test creation, out of the box dbt tests include:

- **Uniqueness:** checks to see if there are any duplicate records in a specific column, which is a useful check for unique identifier columns
- **Not null:** checks for any null records in a specific column that should not have any null values
- **Accepted values:** checks to each record in a column to make sure it matches a predetermined set of values
- **Relationships**: compares columns in separate models to see if they contain the same records known as referential integrity

In addition to the out of the box dbt tests, there are dbt packages that can extend the number of pre-built tests.

- dbt-utils: offers 15 additional schema tests
- dbt-expectations: offers 50+ column tests

Once tests are in place, the next step is to automate the running of these tests against development data before committing your changes to production.

Automate data tests with continuous integration

Prior to cloud warehousing, changes made to data transformations were not automatically tested and reviewed before moving to production. Test data sets were expensive to produce and to maintain, and running tests of new or modified code against verified data was slow and manual. This often led to downstream errors only spotted in production.

Now with cloud data platforms, copies of production data can be provisioned, refreshed, and programmatically run each time a change to the code is submitted. However, the process of ensuring new code is safely integrated requires a new paradigm.

In the world of software development, this problem is solved with continuous integration. Continuous integration (CI) is the process of ensuring new code is integrated with the existing code base in a manner that is safe, controlled, and documented to reduce downtime and increase reliability.

> "When I was a consultant, one of the most consistently rewarding experiences was introducing new clients to the world of continous integration for analytics.
>
> Once they saw how easy it was to set up, and how much value it added to their work, they couldn't imagine going back to the 'hope this works...' method."

Afzal Jasani
Solutions Architect,
dbt Labs

Like version control, this traditional software engineering best practice has also made its way to the data ecosystem, enabling CI tools like GitHub Actions, GitLab, and Azure DevOps to be invoked directly within the data development process. Now new analytics code can be safely merged to reduce the opportunity for error prior to production. Here's how it works:

- **Define your continuous integration steps and code them into your CI platform.** Review with business stakeholders to align expectations for code reviews prior to moving changes to production. Building trust in this step is critical.
- **Set up copies or clones of your database to develop or validate changes.** Developers can then isolate work and test changes in a separate staging database before pushing to production.
- **Let CI do the rest.** Continuous integration workflows automatically isolate code changes to identify which data models to run, saving your team time, and compute.

"CI used to be this elusive concept that data teams could appreciate—but couldn't action. We just didn't have the tools to bring CI out of software development and into data transformation.

Now it's one of the first conversations I have with distributed teams looking for a better way to work together."

Sung Won Chung
Solutions Architect,
dbt Labs

Adopting this way of work might feel very familiar for data engineers, or those well-versed in CI best practices. For those new to these workflows and tools, dbt Cloud enforces by-default Git best practices which promotes consistency across teammates with varied skills.

To learn more about setting up your own CI process, check out the following articles from the dbt Labs team. We strongly believe adopting CI will have a major impact on improving the quality of data code changes and lead to more users being able to contribute to your data projects.

- Enabling CI
- Setting up CI to only run modified and downstream models
- Adopting CI/CD with dbt Cloud

Enable data developers and consumers to confidently self-serve insights

Documenting sources, models, metrics, dimensions, and reports improves data literacy, and trust. But it hasn't always been easy. On-premise data platforms focused on defining how and where to move data—not why. As a result, the only documentation tooling available related to column names and data types. Business users were forced to maintain a separate workbook of data definitions that frequently fell out of sync with the warehouse.

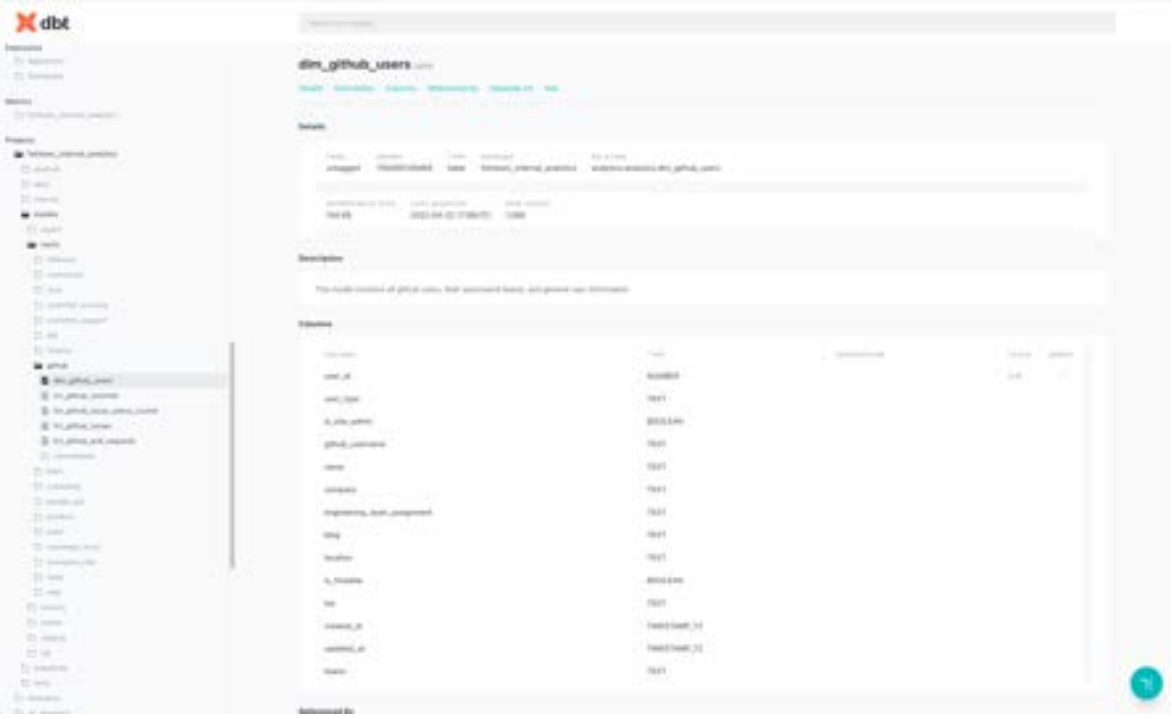
Cloud data platforms help, because their focus on producing reliable, scalable, and extensible platforms has unlocked an API-driven approach to gathering metadata about your data models previously not available in on-premise solutions.

These APIs surface table, column, data profile, and query information from your data platform, but unfortunately lack the business reason for the table or column. Business definitions typically have to be added after your models are coded and moved to production as part of a separate solution. This creates a handoff in the analytics creation process and can lead to outdated definitions or worse, undocumented models.

Use analytics engineering best practices to expedite documentation

Following a modern analytics engineering approach enables contributors and business stakeholders to document data objects as they are created or updated. This documentation can live in your analytics code repository alongside your transformations and then be published to a modern data catalog solution.

dbt Cloud enables your developers to document while they are adding or modifying your analytics. And then it creates a searchable web-based documentation site you can share at your organization to help users answer questions about tables, views, and columns.



This is an example of the document site generated by dbt.

To learn more about dbt Cloud's documentation features go to our documentation page or view our documentation for dbt docs.

Adapting to new governance and security requirements

The tooling once required to transform data was prohibitive to those without the requisite skills, like Python. Coupled with the constant care and feeding of on-premise data platforms required, meant data engineers struggled to keep up with both infrastructure and model management.

The scalability of cloud data platforms enable data teams to focus less on database management, and more on building repeatable contribution frameworks. This opens up opportunities for more members of the organization—especially those with the most business context—to safely contribute to analytics models.

Enabling collaboration in this process:

- Increases the number of users contributing to your analytics code
- Increases visibility into the analytics code within your organization
- Reduce data doubt with a clear rollback strategy for breaking changes

But do more contributors mean more risk? Not necessarily. By codifying the rules that govern a collaborative data development process, you can enforce least-privileged access, increase top-down transparency, and reduce the burden of preparing for audit or security reviews.

Mitigate unknown or unrecoverable data change risks using software engineering best practices

Prior to your transition to the cloud, you may have stored analytics code in GUI-based tools or in an on-premise database as stored procedures. These tools limited access to transformation— for good reason. Those with transformation rights have the ability to modify stored procedures, muddy code with unstructured comments, or even make changes without realizing—risking broken pipelines and inaccurate data. Moving analytics code to cloud data platforms gives you the ability to more easily create copies of data your team is working on so they can safely develop new analytics features.

However, making copies of data doesn't mean the code used to transform it is also copied. It's time to choose a version control platform and train your team to work with this system to provide your organization with the ability to rollback code issues and keep a trail of changes.

Introducing Git to increase collaboration and provide guardrails

Software engineering teams have taken advantage of version control software like Git, Subversion, or Mercurial to manage their code for years. This software helps teams contribute to shared code (with an option for rollback) so that all changes are accounted for. Your data teams can work in the same way, using the same technologies to avoid knowledge bottlenecks that harm data engineer and analyst productivity.

“Git is pervasive in software engineering, but 90% of the data teams I start working with aren’t sure whether it’s necessary for their team—let alone how they would start. In those situations I ask what they would do if their entire data pipeline disappeared in one night. How much work would it take to recover? In a Git-enabled workflow like dbt, a single command brings it back. Without—they could be looking at 12 months or longer to rebuild.

And it’s not hard to get started. A simple set of version control guidelines and tools that default to best practices drastically lowers the barrier to entry for folks safely engaging in transformation work.”

Randy Pitcher,
Solutions Architect, dbt Labs

By moving your analytics code to a central repository like GitHub, your team can make a copy of the current code, and add new fields or tables to help with your analysis using SQL. Once satisfied with your changes you can save (commit) your changes and create a pull request. The pull request will signal to your analytics code repository that a set of checks (CI/CD) need to take place before these changes can be added to your production models. Once all of the checks have passed, the changes can be merged into your production analytics code repository and become a part of the production data analytics.

Git-enabled transformation solutions like dbt Cloud make contributing and documenting code as simple as drafting a new pull request. Just like in software engineering, data pull requests help you communicate changes you intend to publish. To ensure consistency and build accountability, it's useful to adopt a pull request checklist like the one we use at dbt Labs, referenced below:

- **Description and motivation:** This is the intro to your PR and should allow the reviewer to quickly be able to understand the reason for opening this PR. If your actual code is the "how", the description is the "what" and "why."

- **Screenshots:** Add screenshots of lineage that will help the reader understand what's being requested. If using dbt, it's easy to quickly pull the DAG for reference

- **Validation of models:** Confirm that your model is doing what you intended it to do. This could be an ad-hoc query to validate data, or one of the above mentioned tests

- **Changes to existing models:** Delineate post-merge instructions like running a full refresh after updating an incremental model.

- **Prepare for launch:** Follow the same quality checks every time. Have you added tests and documentation to new models? Updated the README file? For more examples of what to include, see the dbt Labs launch checklist.

To learn more about setting up a version control process, check out this article by dbt Labs' Analytics Engineers.

Iterate and learn from your cloud data platform journey

We appreciate the time you invested reviewing this guide. As mentioned at the outset of this guide, you're not alone in this journey. The dbt Community Slack is full of people who have been in your seat and have shared their experiences. If you're thinking more about whether dbt Cloud Enterprise could be a fit, the Solution Architects quoted in this guide are always excited to chat. We look forward to seeing you join the dbt Community and hearing of your successful transition to the cloud in the future.

Helpful resources:

- The Analytics Engineering Guide
- dbt Cloud Quick Start Guide
- Try dbt Cloud by Signing Up for a Trial Account
- Contact dbt Labs Sales to Discuss How dbt Cloud can Help Your Org Adopt a Modern Analytics Approach


dbt Labs
www.getdbt.com