matillion

10 Best Practices for Maintaining Data Pipelines



Summary

In today's data-driven world, data teams build more and more data pipelines to meet increasing demand. They are the lifeblood that enable data to flow seamlessly from various sources to your data warehouse, allowing businesses to gain valuable insights and make informed decisions.

As data teams create an increasing number of data pipelines, they often face a challenge: technical debt accumulates, akin to structural cracks, and this necessitates specialized resources for repair. While building data pipelines has become easier than ever; many data teams spend most of their time maintaining pipelines, and data quality often takes a hit when pipelines are not regularly maintained. Building data pipelines is only half the battle; maintaining them is equally critical for long-term success.

With this perspective in mind, we introduce this eBook, a comprehensive guide to the ten best practices for maintaining data pipelines.



Table of Contents

1 **Best Practice 1:** Good Documentation 04

2 **Best Practice 2:** Test as you Build 05

3 **Best Practice 3:** Organizing Your Codes 06

4 **Best Practice 4:** Data Loading and Transformation on One Platform 07

5 **Best Practice 5:** Adding Naming Conventions to Database Objects 08

6 **Best Practice 6:** Use Views and Temporary Tables for Intermediate Tables 09

7 **Best Practice 7:** If you cannot quality check your pipeline, then you shouldn't build it 10

8 **Best Practice 8:** Breaking Down Complex Data Logic 11

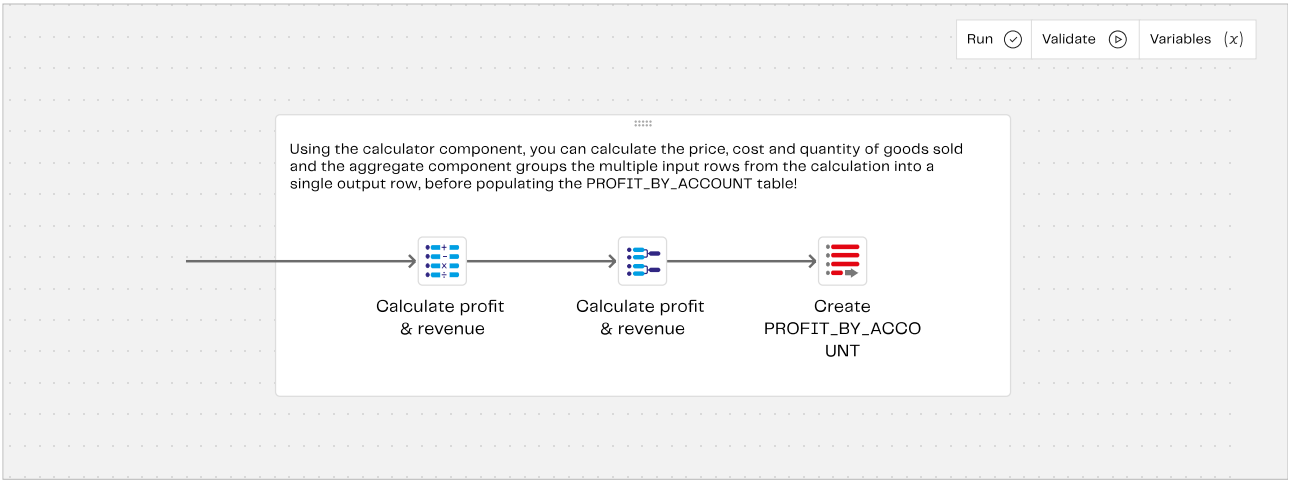
9 **Best Practice 9:** Build in those kill-switch and retry mechanism 12

10 **Best Practice 10:** Define your change management before building your first pipeline 13

1 Best Practice 1: Good Documentation

Pipeline maintenance live and die by how good the accompanying documentation of the pipeline is. Proper documentation is a fine science that lies between — not too little but not too much — documentation. Extensive documentation on a shared drive is essential for audits, particularly in regulatory settings. However, it's less likely that Data Engineers will maintain such documentation, especially in a fast-paced CI/CD environment with frequent sprint changes. In traditional high-code environments, teams often used comments within the code. However, when code branches into multiple jobs, its high-level purpose can become obscured, leading to abstraction and reduced clarity. Low-code platforms with graphical interfaces offer a clear advantage by making branching transparent, enhancing the readability of the entire pipeline, not just the code itself.

In Matillion ETL, documentation is automatically generated by scanning the canvas, creating concise to moderately detailed documentation, significantly saving Data Teams time. We also encourage Matillion's data teams to include brief notes on canvas components. These notes serve a dual purpose, aiding auto-documentation and providing valuable insights to new team members about the pipeline's purpose and functionality. These notes are often just a few lines but collectively weave a comprehensible data narrative, even for non-technical business users. They are generated as HTML documents, which can be conveniently added to document repositories.

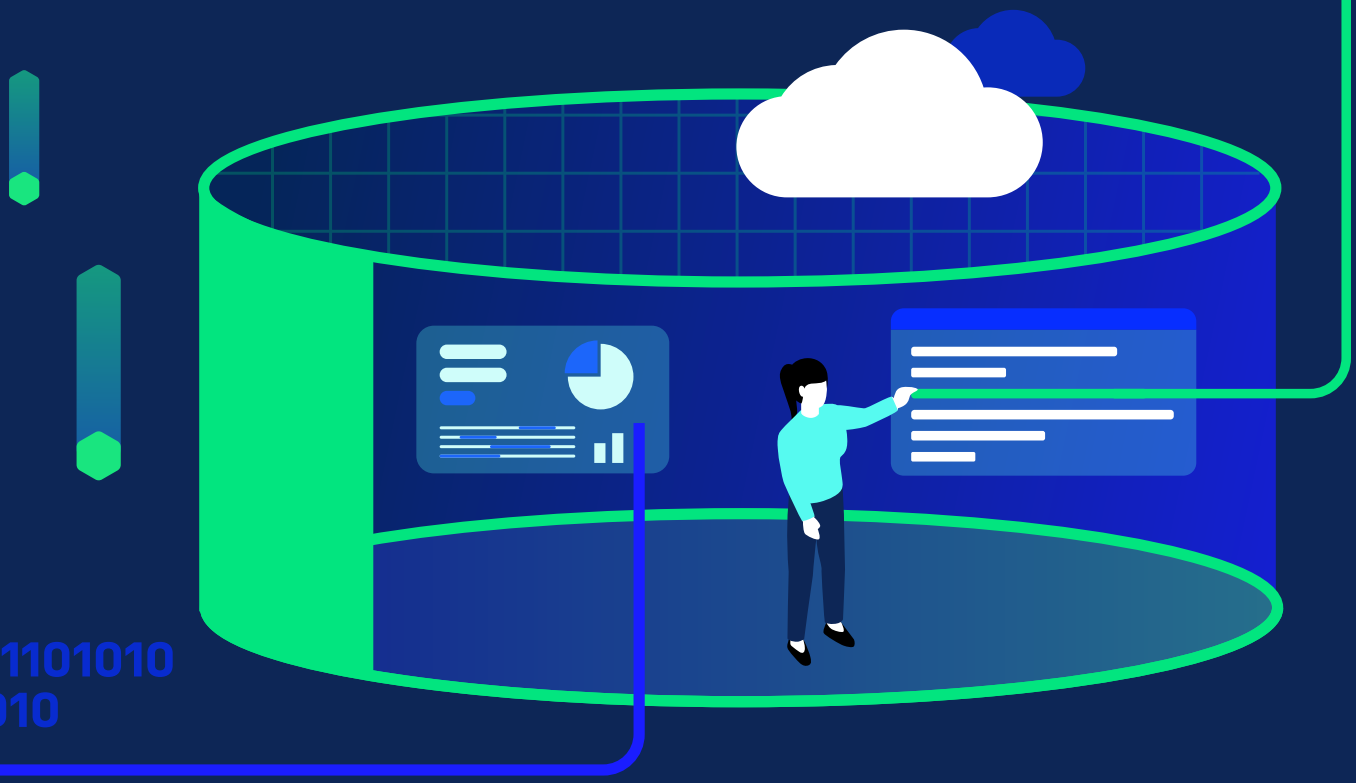


2 Best Practice 2: Test as you build

Efficient pipeline management hinges on robust initial construction. This necessitates ongoing unit testing during the development process. Unit testing for data pipelines is a well-established practice, akin to the fundamental principles of software and data engineering.

The critical factor is the ease with which a data platform facilitates these unit tests. Productivity soars when data teams can swiftly sample their work while building pipelines. If a data professional must execute an entire pipeline or switch tools merely to assess a single calculation, their productivity diminishes. In contrast, those who can test specific transformations in-situ enjoy heightened efficiency.

The choice of tools plays a pivotal role; data teams lacking the right tool may perceive testing operations as 'Large,' whereas the same tests in a more efficient tool could be deemed 'Small.'



3 Best Practice 3: Organizing Your Codes

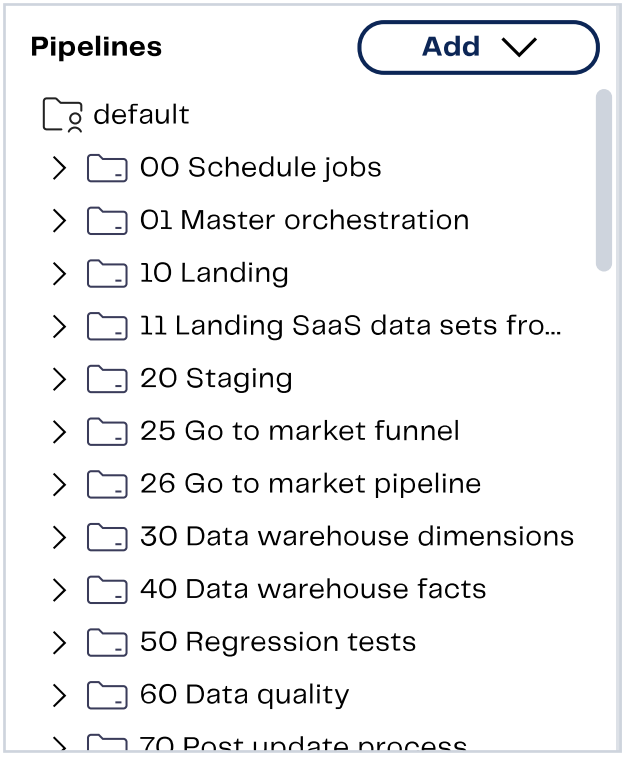
Modern data tools offer diverse solutions for seamless navigation within code-stacks, a crucial requirement as data teams frequently shift between pipelines. Consolidating all your code into a single large folder structure can lead to chaos.

To enhance organization, it's advisable to group codes based on their function. For instance, data extraction codes belong in the "landing" folder, data preparation in the "staging" folder, and so on. This principle extends to codes responsible for building fact and dimension tables, as well as those for data regression testing. Yet, even this organization can benefit from a standardized naming convention, particularly in complex organizations. Often, teams find it beneficial to add a numbering system to clarify the execution order.

For instance:

Loading folder
1_load_aws_data.sql
2_load_salesforce_data.sql
3_load_azure_data.sql
Transform folder
1_transform_aws_data.sql
2_transform_salesforce_data.sql
3_transform_azure_data.sql

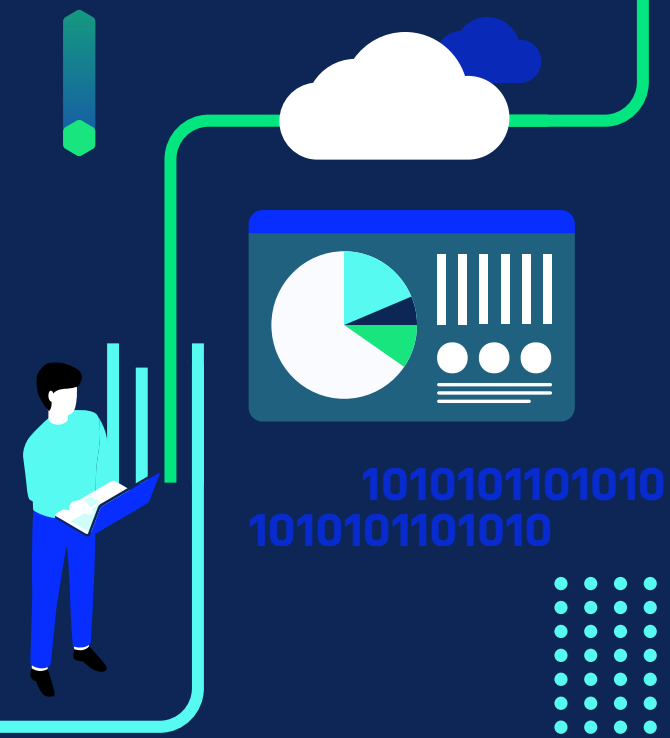
However, in some cases, the execution order may not be linear, as one code in one folder may depend on another in a different folder. This introduces the concept of branching, as mentioned earlier. To maintain productivity, data tools should empower teams to delve into sub-codes, recognizing that pipeline codes exist in a three-dimensional view. Many tools fall short, offering navigation but lacking the ability to drill down into sub-codes. A robust data tool should facilitate users in drilling down, drilling up, and moving both forward and backward, ensuring efficient code management.



4 Best Practice 4: Data Loading and Transformation on One Platform

While discussing data pipelines, we've primarily focused on the movement of data from external sources into cloud data warehouses. However, we must broaden our perspective to include local data movements within the data warehouse itself. For instance, the transfer of data from one schema to another can be viewed as a data pipeline. It's not uncommon for some data teams to employ separate tools for data loading and data transformation. When this occurs, it introduces complexities, as these become independent platforms, resulting in dependencies and hindering the establishment of consistent workflows. Ultimately, this impacts the data team's efficiency in terms of maintenance.

A more efficient approach is to use a single tool for both data loading and transformation. This not only benefits the data team but also extends to data analysis and data science teams. Employing one platform across various teams, encompassing low-coders and high-coders, fosters a vibrant data community with shared skills and knowledge. Such organizations facilitate transitions across the data spectrum, from data engineering to data science, creating a dynamic data culture that promotes streamlined data pipelines and, as a result, easier maintenance.



5 Best Practice 5: Adding Naming Conventions to Database Objects

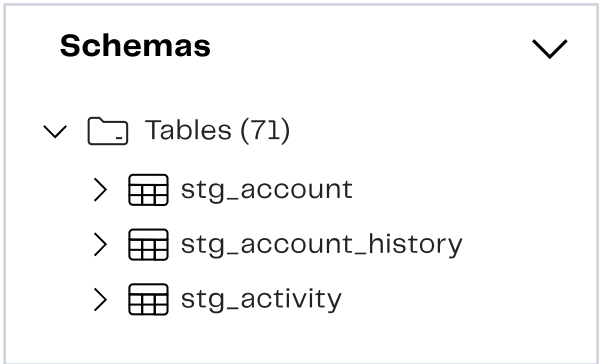
As organizations expand, so does the volume of data within their pipelines, resulting in a multitude of objects, including databases, schemas, and tables. Consider a typical scenario where data teams extract, transform, and create fact and dimension tables from a data source. Multiple tables would be created or updated for each stage. Similar to the code-level navigation we discussed earlier, it's crucial to logically organize these objects.

For example, here at Matillion this is how our data team structures our own database objects:

The scheme db_source will contain all raw data from the sources and will have the prefix 's_' e.g. s_salesforce_opportunities and s_marketo_leads.

Similarly, the schema

1. db_stage will contain transformed tables with the prefix 'stg_'
2. db_dwh will contain transformed tables with the prefix 'f_' for facts tables and 'd_' for dimension tables.
3. db_report will contain specific tables use for analysis & data science.



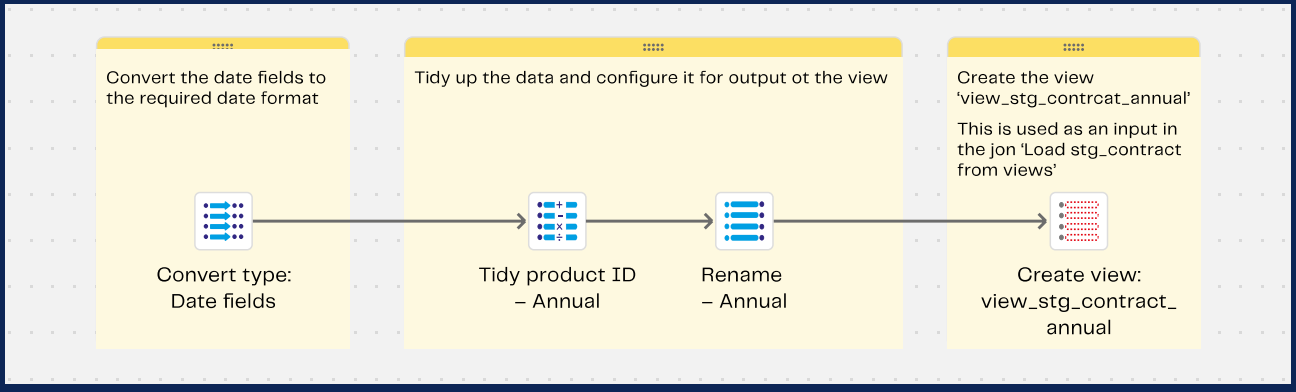
Implementing a robust naming convention for your database objects expedites troubleshooting data-related issues. Furthermore, it enables a decoupled architecture where data within each schema can theoretically be processed separately, leveraging a shared-nothing database architecture like Snowflake. These benefits contribute to enhanced performance optimization, monitoring, and alerting.

6 Best Practice 6: Use Views and Temporary Tables for Intermediate Tables

Cost efficiency is a critical aspect of pipeline maintenance that is often overlooked. While storage costs have significantly decreased over the past few decades, they still represent a tangible expense. In this context, we recommend breaking down complex code into smaller, more manageable code modules. This approach naturally results in the creation of multiple staging tables. These staging tables are typically temporary and become obsolete once their respective operations conclude. Consequently, there's no justification for retaining these tables in your database, incurring unnecessary storage costs for your organization.

Modern data platforms, including **Amazon Redshift**, **Databricks**, and **Snowflake**, provide solutions such as temporary tables, transient tables (in the case of Snowflake), and even views to tackle this issue. Another often-overlooked advantage of using temporary tables or views in your data pipelines is their role in mitigating against data swamp. A data swamp arises when your data environment becomes cluttered with outdated, irrelevant, undocumented, and inaccurate data. This often occurs when the pipelines responsible for creating permanent tables either no longer exist or have undergone significant changes.





7 Best Practice 7: If you cannot quality check your pipeline, then you shouldn't build it

Data quality stands as a primary challenge arising from our contemporary data proliferation paradigm. While we've highlighted in this blog that building pipelines has become relatively straightforward, the issue of data quality is often the first casualty when these pipelines aren't consistently maintained.

Most modern pipeline tools, including Matillion, come equipped with built-in features like auto-fixes for schema drift and basic observability through logs and alerts. However, when operating at a large scale, these tools may fall short of comprehensive data quality assurance. As a result, additional pipelines are required to monitor the production pipelines themselves.

Data teams frequently develop regression test pipelines to scrutinize aspects such as data completeness, freshness, and uniqueness. These pipelines tend to be more intricate and may even involve data science modeling for specific operations like data profiling. In environments with high code complexity, the need for a dedicated data quality tool becomes evident to simplify the intricacies of data quality assurance.

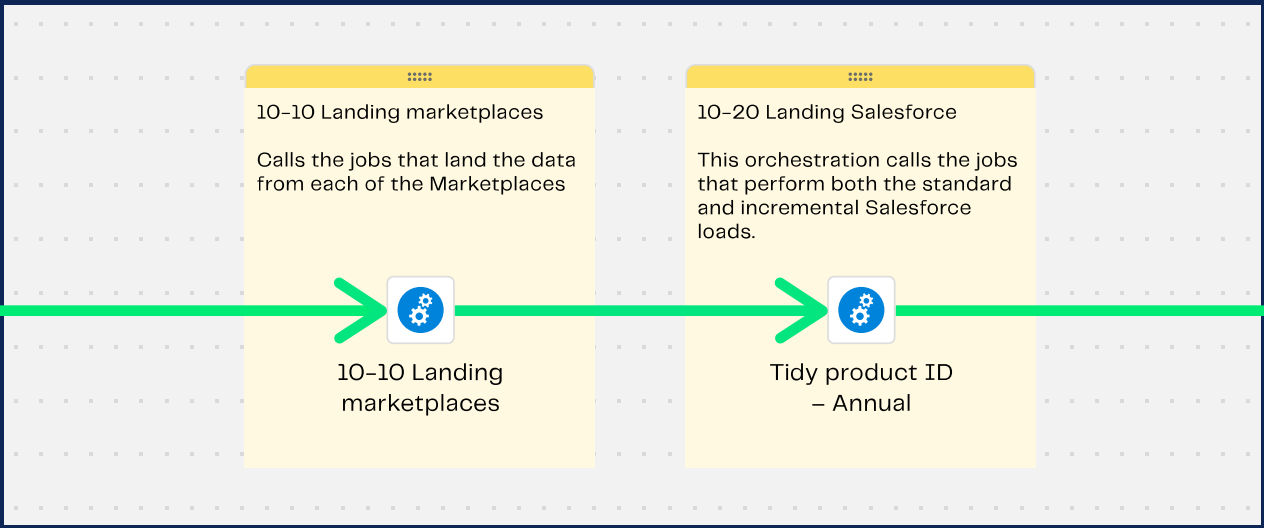
However, the advantage of low-code tools, as opposed to high-code environments, lies in their simplicity. Low-code tools streamline the integration of new pipelines into the observability framework, making both construction and management notably more accessible.

8 Best Practice 8: Breaking Down Complex Data Logic

Breaking large code into smaller parts has been a well-established industry practice for decades, offering various advantages such as improved comprehension, enhanced data quality checks, and streamlined maintenance. However, in practice, even with the best intentions, smaller, decoupled code can become complex when data teams compromise on best practices to adapt to rapidly changing business rules.

In environments with a substantial amount of code, the task of decoupling code is typically undertaken by the most experienced team member, given their expertise in doing so efficiently. Consequently, in high-code environments, the process of breaking down large code into smaller segments can be more costly due to the need for specialized resources.

Conversely, low-code platforms with intuitive graphical interfaces facilitate a more cost-effective approach to this task. At Matillion, the data team regularly delegates code decoupling to less experienced members, thanks to the user-friendly canvas interface. This simplifies the process, making it more straightforward without requiring an in-depth understanding of complex coding intricacies.



9 Best Practice 9: Build in those kill-switch and retry mechanism

Data failures are an inevitable part of the data landscape, and modern data teams are actively engaged in proactive detection and automated rectification. As previously discussed in this blog, we've explored the concept of data quality observability. However, it's essential to recognize that simply having these observability tools in place isn't enough; they must serve a more proactive purpose than being mere ornamental dashboards in a visualization tool.

Our objective is to leverage the insights gained from observability to trigger data-driven actions that can autonomously rectify issues within data pipelines. This is where orchestration logic comes into play, offering control mechanisms like If/Else statements, retry protocols, and conditional branching.

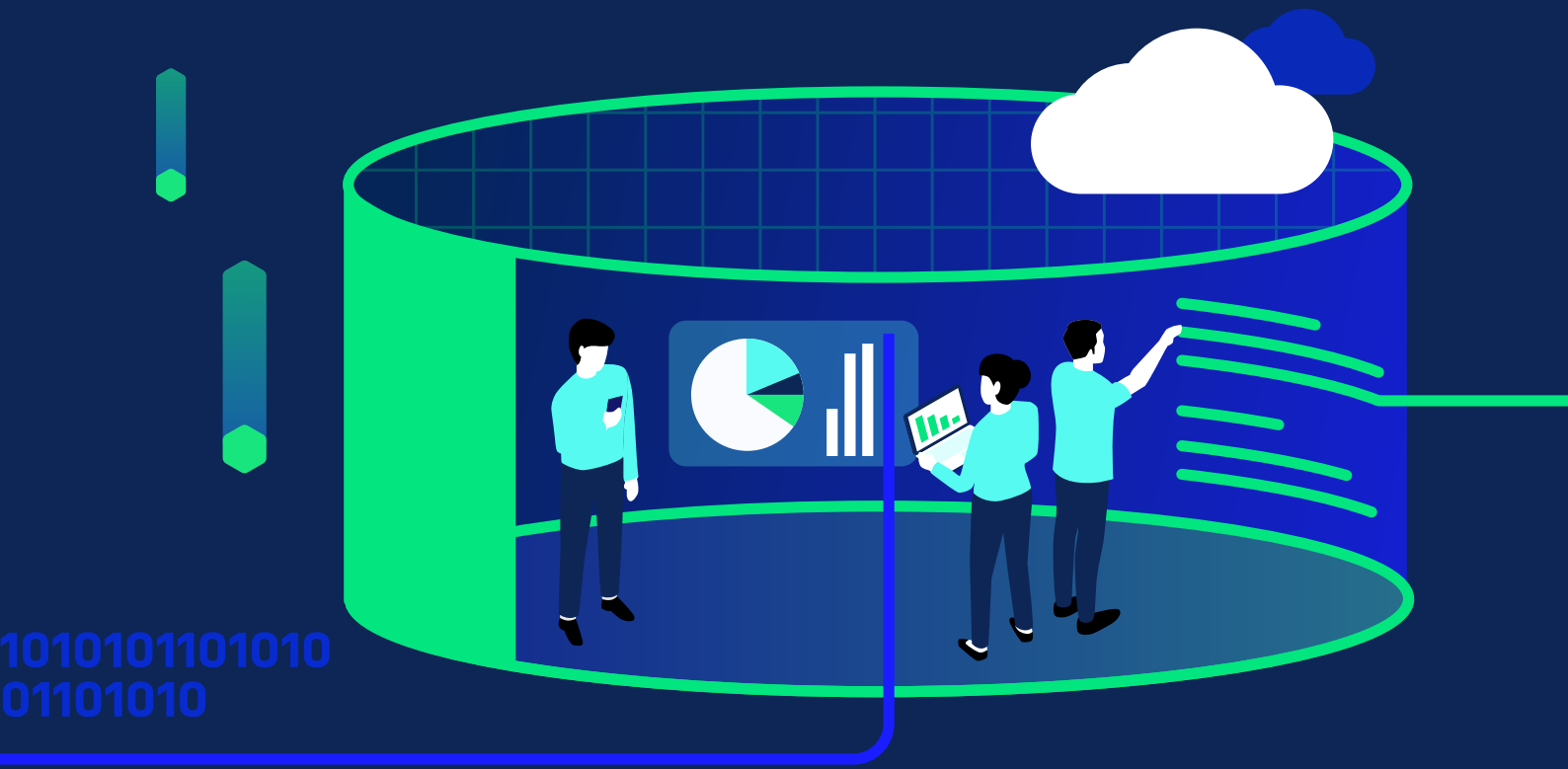
In most scenarios, when a specific operation encounters an issue, a straightforward retry is the desired course of action. These retry mechanisms play a vital role in ensuring the reliability and graceful recovery of data processing tasks, significantly reducing the troubleshooting workload on the data team. Nonetheless, these mechanisms can become intricate, especially in complex business domains. The core principle remains consistent: automating the resolution of transient errors.

Conversely, if an error persists beyond the retry attempts, the flow should halt to prevent further issues. When constructing new pipelines, data teams must carefully consider how to implement these flow management operations and develop contingency plans for addressing major data failures.

Best Practice 10: Define your change management before building your first pipeline

DataOps has matured into a well-defined and comprehensive domain that encompasses various sub-disciplines. These include the integration of Agile methodologies, CI/CD practices, and robust security measures. The advantages of DataOps are undeniably poised to enhance pipeline maintenance significantly.

Consequently, it is imperative that, prior to constructing pipelines, data teams give thoughtful consideration to how these initiatives align with the overarching DataOps framework. This involves addressing fundamental questions, ranging from managing version control and traceability to fostering cross-functional collaboration.



Leveraging Matillion's Data Productivity Cloud for Effective Data Pipeline Maintenance

Matillion's Data Productivity Cloud simplifies and automates data movement, transforming data pipelines into more manageable and maintainable structures. With its low-code approach, Matillion enables data teams to document, break down logic, organize codes, and perform unit testing efficiently. It offers an all-in-one platform for data loading and transformation, fostering a dynamic data culture. Additionally, Matillion supports data quality checks, orchestration logic, and change management, all within a unified platform.

Ready to Get Started?

With Matillion, you're not just building data pipelines; you're empowering your entire data team to achieve more. **You can begin in minutes with a free trial of our platform**. Unlock the power of data productivity today!

matillion