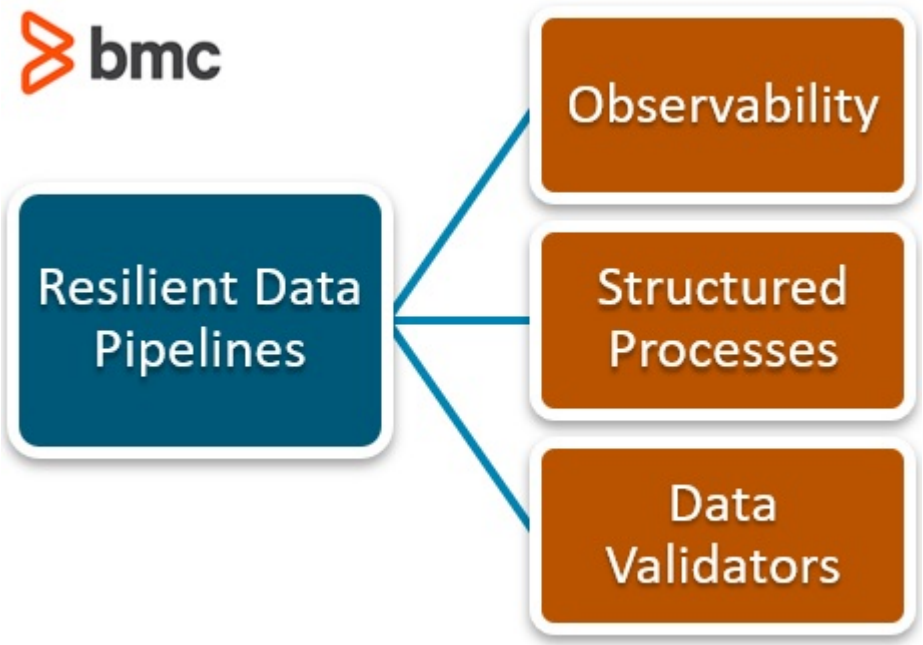
3 KEYS TO BUILDING RESILIENT DATA PIPELINES

A resilient data pipeline detects failures, recovers from them, and returns accurate data to the consumer. Resilience is built on three components: observability, structured processes (including idempotence and immutability), and data validators. Together, these ensure a data pipeline adapts rather than breaks when something goes wrong.

What makes a data pipeline resilient?

Resiliency means having the ability to adapt. A resilient data pipeline adapts in the event of failure. Because data pipelines are designed to transport and transform data from one point to another, a resilient pipeline must:

- Detect failures
- Recover from failures
- Return accurate data to the consumer



Achieving this requires three components: observability, structured processes (including idempotence and immutability), and data validators.

How does observability support resilient data pipelines?

Observability ensures that errors are seen quickly enough to be corrected. If an error goes unseen, it cannot be fixed. Good logging practices help surface errors, and well-written logs allow the error to be identified and located quickly. Developers—usually the people reading these logs—need to understand what happened and what to do about it.

A good log message contains three things:

- An appropriate, consistent category for the type of error that occurred
- A specific description of what caused the error
- Clear guidance on what action needs to be taken to correct the error

If a log gives the developer a path to fix the issue without requiring them to probe the code, it is a good log.

Logs can also be written at both the pipeline and task levels. They should answer:

- Why did the pipeline task fail?
- When was the pipeline scheduled?

Whether you use your own processors or rent from a third party, each processor should produce logs—and you should access third-party logs or write your own to widen your visibility window.

Finally, after data has traveled through the pipeline, logs should confirm that the data has successfully reached its destination.

Observability also extends to pipeline resource metrics. Monitor system latency, batch queue time, and resource consumption per run to catch performance degradation before it becomes a failure.

How do structured processes improve pipeline resilience?

Resilience improves when the infrastructure of a data pipeline is designed with predictability in mind. The goal is to use processes that deliver the same expected result consistently—including in the event of failure. Idempotence and immutability are two such processes, and they help ensure data reaches the end user correctly even when a processor is unavailable, stops mid-shipment, or is triggered multiple times.

Structured processes accomplish two things:

- They ensure good data reaches the end user
- They ensure data arrives even during a systems failure

Idempotence

Idempotence is a function that returns the same result on every repeated execution. Absolute functions and rounding functions are idempotent by nature. Not all functions are idempotent, but functions can be written to behave idempotently.

Consider a deletion function operating on a list of data. If the goal is to remove the number 2 at index 2, a function that simply deletes whatever is at the second index is not idempotent—run multiple times, it removes 2, then 3, then 4 in sequence. An idempotent version would instead specify: if the value at index 2 is 2, delete it. This version can run repeatedly and always produces the correct output. (Wikipedia lists further examples of idempotent functions.)

Immutability

An immutable infrastructure decreases complexity and allows for straightforward reasoning about system behavior.

Traditionally, mutable servers could be updated and modified while in use. The problem is that each update creates slight differences between servers—one running version 1.34, another at 1.74, another at 1.22. Identifying errors across these divergent servers becomes complicated, and versioning the pipeline is difficult when the underlying infrastructure is a moving mix of versions.

Immutability means that data cannot be deleted or altered once written to the server. Immutable infrastructure simplifies pipeline versioning and separates infrastructure build errors from actual data processing tasks.

In an immutable infrastructure, Step 1 might be to create a VM instance and Step 2 to write user data to it. If the VM creation fails, the system reruns Step 1 until a stable instance is ready before any data is written. In a mutable infrastructure, that same failure path is more complex, more error-prone, and harder to version.

What do data validators contribute to pipeline resilience?

Data validators act as the final quality gate in a resilient data pipeline. Validation occurs when data arrives at the consumer, quickly processing incoming data to verify it matches what the consumer expects.

Data validators check for:

- Data types—integer, string, boolean, and other type constraints
- Structural constraints—for example, a valid university email format, a phone number with 10 digits, or a value greater than zero
- Quantitative validation—domain-specific plausibility checks, such as flagging an NBA player height listed as four feet or a dinosaur fossil dated three weeks ago

Validators help ensure that appropriate data is returned to the consumer, but they carry their own risks: depending on implementation, validators can increase pipeline latency, and poorly written validators may not perform their intended function completely.

How does a resilient data pipeline adapt when failure occurs?

Resilience is adapting in the event of failure. By designing observability into the pipeline, failures are announced and made visible, enabling corrective action. Structured processes like idempotence and immutability build an infrastructure that holds up under failure conditions. Data validators confirm that appropriate data reaches the consumer at the end of the pipeline. With proper logs throughout, failures are exposed and—whether through automated or human correction—the pipeline is free to adapt.

Building resilient data pipelines at enterprise scale requires a data pipeline orchestration solution that provides SLA monitoring, automatic retry logic, dependency management, and cross-platform visibility in a single control plane.

Frequently asked questions: resilient data pipelines

What is a resilient data pipeline?

A resilient data pipeline is one that detects failures, recovers from them, and continues to return accurate data to the consumer. Resilience is built through observability, structured processes such as idempotence and immutability, and data validators that verify incoming data meets expected criteria.

What is idempotency in a data pipeline?

Idempotency in a data pipeline means a function returns the same result regardless of how many times it is executed. Idempotent functions are critical for failure recovery scenarios where a processor may restart, be triggered multiple times, or stop mid-execution.

What is the difference between idempotence and immutability in data pipelines?

Idempotence refers to functions that produce consistent results on repeated execution. Immutability refers to data or infrastructure that cannot be altered once written. Both properties improve pipeline reliability: idempotence ensures repeatable outcomes during retries, while immutability simplifies versioning and isolates build errors from data processing.

What should a good data pipeline log include?

A good data pipeline log should include the error category, a specific description of what caused the error, and clear guidance on how to correct it. Logs should be written at both the pipeline and task levels, and should include a confirmation entry when data has successfully traveled through the pipeline end-to-end.

How do data validators protect data integrity?

Data validators check that data arriving at the consumer matches expected types, structural constraints, and domain-specific plausibility criteria. They act as a final integrity check in the pipeline, catching corrupt, malformed, or implausible data before it reaches downstream systems or end users.

Additional resources

For more on this topic, browse the BMC DevOps Blog or see these articles:

- Resilience Engineering: An Introduction
- How to Create a Machine Learning Pipeline
- Machine Learning with TensorFlow & Keras, a multi-part tutorial

The views and opinions expressed in this post are those of the author and do not necessarily reflect the official position of BMC.