O'REILLY®

Fundamentals of Data Observability

Implement Trustworthy End-to-End Data Solutions

Early Release
Raw & Unedited
Compliments of
KENSU

Andy Petrella

KENSU

Trust what you deliver

Decentralized data observability solution
for modern data teams.

Kensu helps data teams tackle data issues at scale.

Deliver more value

Understand data usage, improve collaboration, and deliver data products faster.

Reduce costs

At a glance, find the root causes of complex data issues and stop wasting time and resources.

Mitigate Risks

Make sure the quality of your data remains consistent at all times across your projects and applications.

Learn more on

www.kensu.io

Fundamentals of Data Observability

Implement Trustworthy End-to-End Data Solutions

Andy Petrella

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Fundamentals of Data Observability

by Andy Petrella

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com* .

Acquisitions Editor: Jessica Haberman
Development Editor: Gary O'Brien
Production Editor: Ashley Stussy

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

July 2023: First Edition

Revision History for the Early Release

2022-06-14: First Release
2022-08-01: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098133290* for release details.

The views expressed in this work are those of the author(s) and do not represent the publisher's views. While the publisher and the author(s) have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author(s) disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-13323-8

Table of Contents

CHAPTER 1
Introducing Data Observability

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *gobrien@oreilly.com*.

Over centuries and across industries, we have witnessed strategies using data transform until its rapid growth, and associated capabilities (e.g., ML), have led to data becoming the strategy itself.

Although they used to call themselves "data-driven" companies (and many still do), in a sense, all organizations aim to become a *data company*. Data is perceived as an asset, it can be seen as a product that can generate value for the business. Hence, limiting a company strategy to only becoming "data-driven" reduces the competitive advantage the company would have in the market.

For an organization to become a data company, they need to consider several transformations. One of them is their ability to scale their capacity to operationalize the generation of value from data - this has led to identifying a market gap called data operations (DataOps).

Improving data operations means you have to reduce the time to market – the time between discovering a potential data value and its availability for the consumers, a.k.a *time to value*. Then you have to maintain this value at the lowest possible cost – the resources needed to keep the existing consumers satisfied and extend the new ones. This is linked to the *total cost of ownership*, and one of the key factors is the cost of quality assurance – the data quality.

Managing the data quality is the most important barrier for organizations to scale data strategies due to a lack of ability to identify issues and remediate them appropriately.

Data quality is not a new topic. Patterns or solutions that existed to address are not suited to cope with the scalability effect introduced with DataOps. Only event considering tools such as notebook deployment in Databricks or DBT, integrated with automated tests, the time to deliver data projects has shrunk so that data teams can create data much faster. Consequently, the number of data projects continually grew, but also their complexity Therefore when a data issue is detected, finding its cause is so challenging that the only option is to patch the data, which disrupts the data value chain, and generates debts.

Data Observability introduces new approaches, best practices, and capabilities that are needed to create the necessary visibility about data for companies to scale their data ambitions.

Before I dig further into what data observability is and offers at scale, let's first look at how data teams are evolving in and identify the challenges they face.

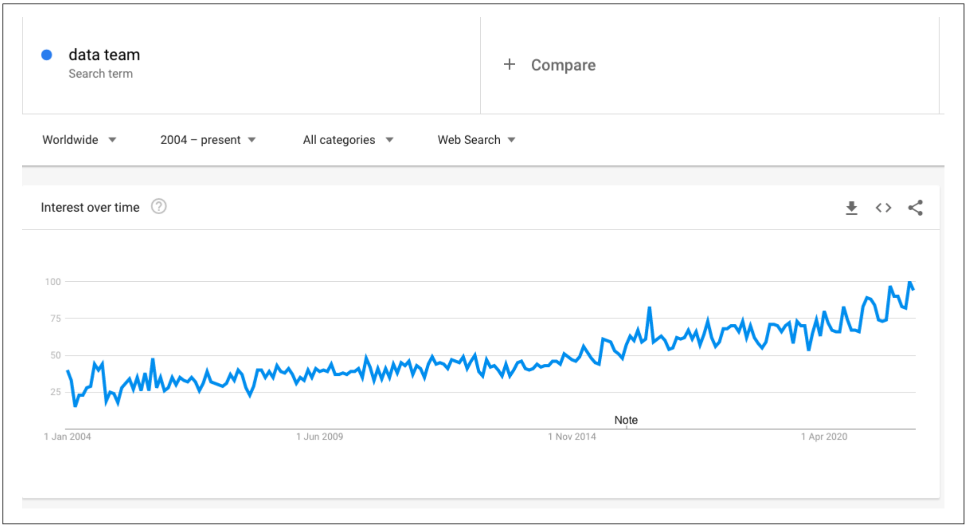
The evolution of data teams

To support the scaling of data value, data teams have started to play a critical role in the implementation of incorporating data into business operations.

This evolution of data teams is similar to how IT teams evolved in the 1950s[1]—when dedicated teams and roles were created to incorporate computing into business operations.

Looking at Google Trends (Figure 1-1) of the term "data team" from 2004 to 2020, we can see that while data teams have always been around, the interest in data teams began increasing in 2014 and accelerated significantly between 2014 to 2020.

[1] Mary K. Pratt, "CIO (Chief Information Officer)," TechTarget, accessed April 11, 2022.



data team
Search term

Worldwide 2004 – present All categories Web Search

Figure 1-1. Google search trend for "data team".

This is despite the fact that interest in "Big Data" began to decrease (as shown in Figure 1-2).

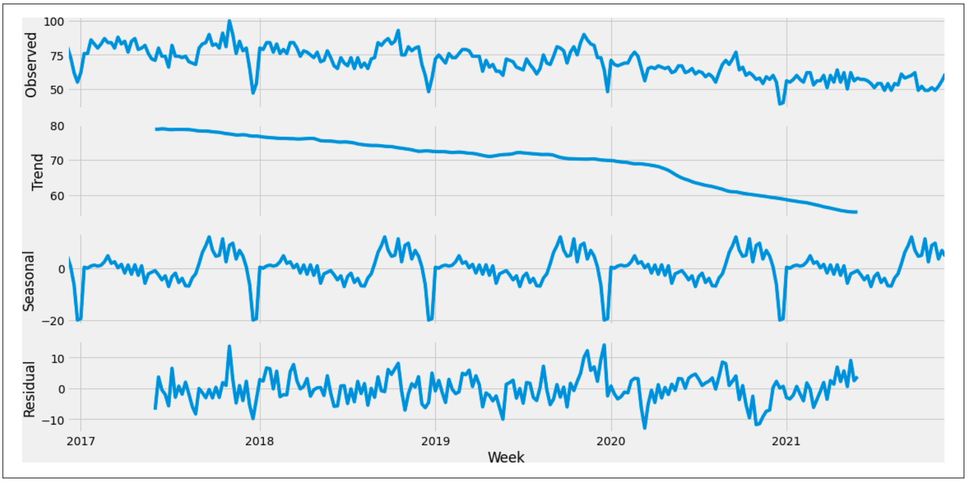
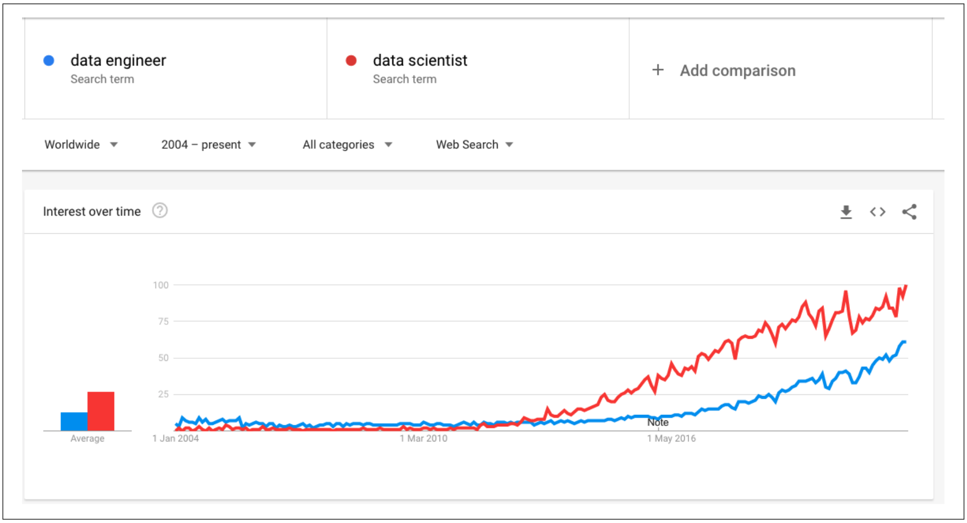


Figure 1-2. Analysis of "Big Data" search term on Google Trends[2]

Obviously, this doesn't mean that Big Data was not needed anymore, but, as Figure 1-3 shows, in 2014 the focus started to shift toward Data Science because of its link with value-generation was more intuitive.

[2] Cf What is Data Observability report



● data engineer
Search term

● data scientist
Search term

+ Add comparison

Worldwide ▾ 2004 – present ▾ All categories ▾ Web Search ▾

Interest over time

Figure 1-3. Google trends for search terms "data engineer" and "data scientist".

However, even as interest began to climb in data science, it was clear that data availability was a roadblock to many data science projects.

The big data teams have not been replaced by data science teams. Instead, data teams have embraced analytics, and thus added this relatively new role of Data Scientist.

As data and analytics became more central to companies' success, providing the right data to the right people remained a constant challenge. Gartner noted that, starting in 2018, data engineers had become crucial in addressing data accessibility challenges and that data and analytic leaders must, therefore "develop a data engineering discipline as part of their data management strategy."

Hence, it became evident that a role dedicated to producing the data for downstream use cases was missing. This is why, since around 2020, as companies began to bring on engineers specifically to help build data pipelines and bring together data from different source systems, the search volume for data engineers has increased significantly. . Today, as also shown by Figure 1-3, data engineering is trending toward catching up to data scientist in search popularity.

Data availability is still not fully solved (at the time of writing), because the underlying reasons are still valid. Data sources (operational) are not entirely known, ingesting highly valuable business data from legacy systems also faces organizational resistance (for security, performance, or compliance reasons). But the separation of roles allows the different skills to build and the resources to be focused on the different phases of data projects.

Search trends are just one example that illustrates the creation and scaling of data teams. Additionally, these search trends highlight how these data teams started to become more organized with members that each has their own specialty and scope of responsibility to implement new products and services based on data.

Challenges with current data management practices

As with any transformation of a company, data transformation and the associated creation of data teams posed the question of the teams' location within the organization. Should there be one central data team? Should it be in IT? Or, maybe there should be one data team per business domain? Should they be under IT, in each domain? But then, how do they collaborate and how do they stay consistent, and efficient? Et cetera, et cetera.

Those questions are being addressed in a large number of ways, however, Data Mesh is one of the few (not to say the only one) that addressed them by rethinking data management at all levels including at the architecture, culture, and organization level.

However, independently of the position of data teams in the structure, let's analyze the impact on data management when they are scaling.

Data management is a vast topic that is widely defined in the literature, for this analysis I will concentrate on how data management is defined by DAMA[3] in their *Data Management Body of Knowledge v2* (DMBOK2)[4] book.

In DMBOK2, data management is composed of many areas, such as Data Architecture, Metadata, and Data Quality. All those areas are participating in leveraging the value of the data, alongside Data Governance which is meant to dictate the policies that must be respected to ensure the data value is generated efficiently and according to the core principles of the organization. This is very well represented in the Data Management Wheel of the framework, see Figure 1-4.

[3] The global data management community

[4] DAMA-DMBOK: Data Management Body of Knowledge v2

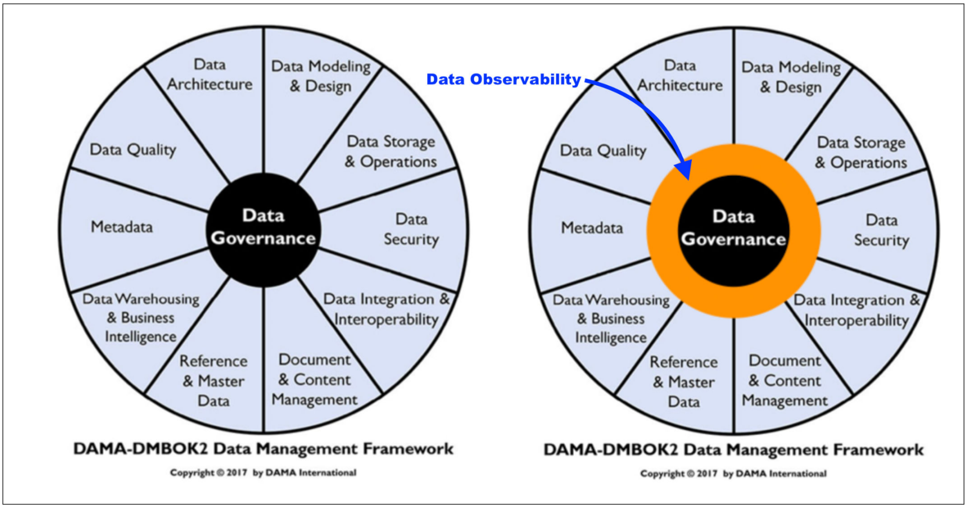

Data Architecture
Data Modeling & Design
Data Storage & Operations
Data Security
Data Integration & Interoperability
Document & Content Management
Reference & Master Data
Data Warehousing & Business Intelligence
Metadata
Data Quality
Data Governance
DAMA-DMBOK2 Data Management Framework
Copyright © 2017 by DAMA International
Data Observability
Data Architecture
Data Modeling & Design
Data Storage & Operations
Data Security
Data Integration & Interoperability
Document & Content Management
Reference & Master Data
Data Warehousing & Business Intelligence
Metadata
Data Quality
Data Governance
DAMA-DMBOK2 Data Management Framework
Copyright © 2017 by DAMA International

Figure 1-4. Data Management Wheel from DMBOK2

As data ambitions and data teams scaled, each area of the wheel has evolved to adapt to the different situations and needs. In fact, in the next section, I will cover how the increase in the importance of Data Governance is impacting the others.

Effects of Data Governance at Scale

Many challenges are resulting from the evolutions of data management practices and technologies, but I'll concentrate on one specific challenge: how to control at scale that the data culture is sustained, by respecting not only the data governance principles but also the definition and the implementation of the resulting policies. Data governance defines the policies and each area is responsible for defining, planning, and executing the implementation.

However, at the implementation, many dependencies across areas are introduced, plus the fact that everything scales, without a harmonized, defined, and global control layer, any area that presents a default can break the machine.

This is why I am proposing to not necessarily change the abstraction presented in this picture but to extend it with an extra area, as centric to data governance, but extrinsic to it.

This Data Observability, which will be formally defined in a later section, "The areas of Observability," makes the responsibility explicit to bridge the culture, its principles, and policies with its implementation and respect across all other areas.

The result is presented in Figure 1-5. below, however, it doesn't mean there is necessarily a need for a data observability role, team, or division. It states that at scale, the control must become explicit as it becomes a challenge that, at a lower scale, it could have been handled in a more ad-hoc fashion.

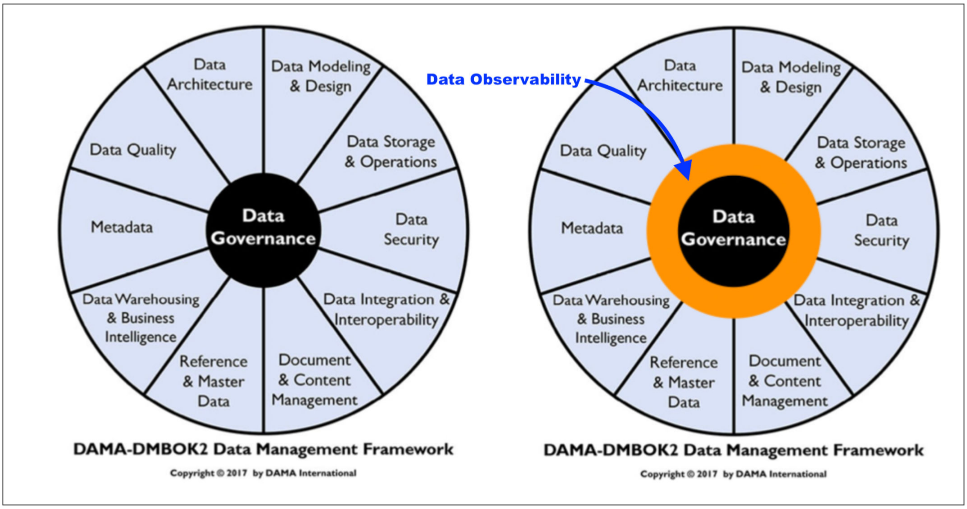

Data Architecture
Data Modeling & Design
Data Storage & Operations
Data Security
Data Integration & Interoperability
Document & Content Management
Reference & Master Data
Data Warehousing & Business Intelligence
Metadata
Data Quality
Data Governance
DAMA-DMBOK2 Data Management Framework
Copyright © 2017 by DAMA International
Data Observability
Data Architecture
Data Modeling & Design
Data Storage & Operations
Data Security
Data Integration & Interoperability
Document & Content Management
Reference & Master Data
Data Warehousing & Business Intelligence
Metadata
Data Quality
Data Governance
DAMA-DMBOK2 Data Management Framework
Copyright © 2017 by DAMA International

Figure 1-5. Data Management Wheel extended with Data Observability

In the remainder of this book, we'll look at how observability extends from the IT DevOps landscape and how to apply it to data and data teams.

To understand why data observability is crucial and must be embraced, we'll consider how data teams are evolving over time, the impacts on the culture, the distributed responsibilities within data teams, and the many challenges that must be addressed and strategies applied to keep data teams effective.

Challenges of Scaling Data Teams

As companies look to scale their data usage, they also must scale their data teams. In this section, I will take you on the journey of growing data teams and highlighting why automation becomes essential.

Consider a data team that starts with a single person tasked to do the data ingestion, data integration, and even the final report. For this person, a first project might consist of producing a view of recent customer acquisitions.

The data engineer would have to query and ingest the source data from the company's *Customer Relationship Management* (CRM) system into the data lake, integrate it into the data warehouse, and create a report that gives the head of sales insight into the progress of the sales made over the past few weeks.

This first data team member is, in essence, a Swiss knife: covering both data engineering and data analysis responsibilities in order to deliver the project outcome (report).

In a larger data team, the skills required for this work are balanced between data engineering, data analysis, and business analyst. As a team of one though, the person must master the technologies required to build and run the pipeline, reporting tools, and understand business KPIs. It is worth noting that each of these areas requires specific expertise.

So far this person is happy, and the scope of work and responsibilities is as shown in Figure 1-6.

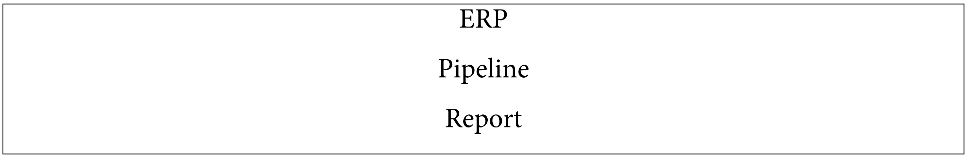


Figure 1-6. A happy single-person data team and its work.

Of course, the process is not quite as simple as depicted above. For example, there should be an entire systems development life cycle (SDLC) to be followed to get the data into production and allow it to be available for analysis by the final user. At a very high level, however, this basic process outlined is what we mostly think of as "producing value" with a data project.

With only one team member and one data project, the whole data ecosystem is relatively under control.

The process is simple enough for one person and the responsibility for the entire project is also attributed to a single person. Because there is just a single person on the team, they also have a close relationship with the business stakeholders—such as the head of sales—and have therefore built some domain knowledge around the use case and the business objective of the project. While this role is typically what a business analyst would perform, in this case, the data engineer takes on this role as well.

Thus, if any issues are raised by the user, it is clear who should work on troubleshooting. After all, he not only executed each step of the process but has knowledge of each area of the process.

However, as stakeholders become aware of the potential and value data can provide, more requests are made to this person for new reports, new information, new projects, and so forth. As a result, there is an increase in the complexity of the demands and ultimately, the individual reaches their limits of production capacity under the current setup. Consequently, it becomes urgent to grow the team and invest in specialized members to optimize productivity.

In the next section, we'll see how this team starts growing as specific roles are created, such as data engineers responsible for providing the data to analysts, data scientists for building analytics and AI/ML models, and data analysts for interpreting the data for the business team's use cases. The team members are working together to respond to business users' requests who will use the insights to make business decisions. See Table 1-1 for the responsibilities and dependencies of each role.

Table 1-1. Data Team (extended) roles, responsibilities, and dependencies

Title/Role	Responsibilities	Dependencies
IT team/ Production and operation	Build and maintain the platform; manage security and compliance; oversee site reliability engineering (SRE)	Receives recommendations from other teams about what to monitor and what to consider as an error.
Data Engineer	Build the data pipelines and manage the data from deployment to production	Relies on IT to build the platform and set up the tools to monitor the pipeline and systems, including data observability tools
Analytic/data science	Build analytics and AI/ML models that can analyze and interpret data for the business team's use cases	Rely on the data team to build the pipeline to generate the data they will be using in their models
Business/ Domain	Sponsor use cases and use data analysis to make business decisions	Rely on the other teams to ensure data and analyses are accurate

There is also a shortage in data engineering, data science, and data analyst skills on the market, hence growing a team is extremely challenging[5].

This is especially true at the beginning when everything needs to be built in parallel, such as the culture, the maturity, the organizational structure, etc. Thus, it becomes even more essential to keep the talent you do have and this means ensuring that their roles and responsibilities align with their skillsets. So, in the next section, we will discover how these roles are added, what they will be expected to do, and what will happen to the team over time.

Segregated Roles and Responsibilities and Organizational Complexity

The single-person team has reached its maximum capacity to build new reports and has added another member to get back on track with the pace of the incoming requests. As stated earlier, it is important to specialize the team members to maximize their output, satisfaction, and, of course, quality of work. So we have now, a data analyst delivering the reports defined with the stakeholders, upon the data sets built by a data engineer.

The impact of segregating the roles is that the data engineer is now farther from the stakeholders and loses direct contact with the business needs. In a way, the data engineer loses part of the visibility about the final purpose of the project as well as part of the responsibility for the final outcome. Instead, the engineer will focus all

[5] https://quanthub.com/data-engineer-demand/

efforts and energy on their own deliverables—the ETL, SQL, or whatever framework or system was used to build the data.

The team and its deliveries are represented in Figure 1-7, where we clearly see the distance and the dependencies taking place.

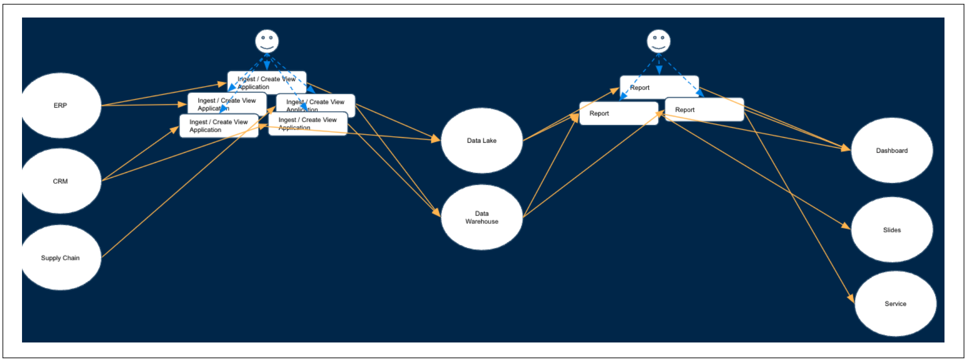

ERP
CRM
Supply Chain
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Data Lake
Data Warehouse
Report
Report
Report
Dashboard
Slides
Service

Figure 1-7. The team is growing, the complexity as well.

Figure 1-7 also shows the scope of produced work under the management by the data team after a few projects have been delivered. The data ecosystem begins to grow, and there is less end-to-end visibility, as it is scattered across both brains, and the responsibility across team members starts siloing.

Next, a data scientist is added to the team, as stakeholders are willing to explore automated decision making without necessarily having a human in the loop, or overseeing the results and to scale the value generated by the data.

As shown in Figure 1-8, this adds more scale, and those projects will require more data at a faster speed, and more complex as the whole process needs to be automated to generate the expected results.

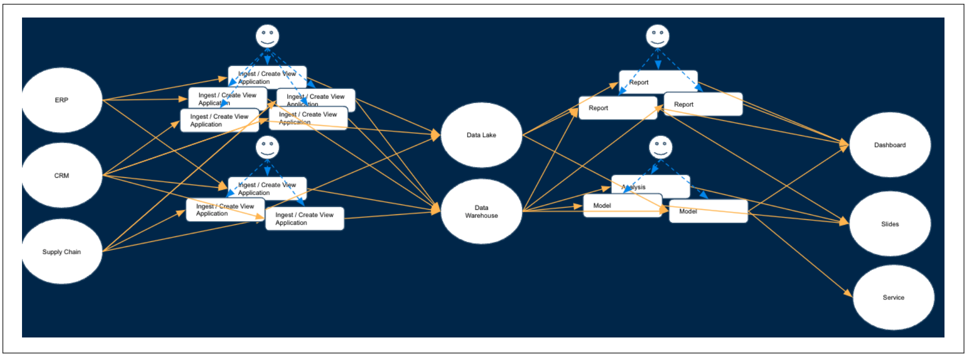

ERP
CRM
Supply Chain
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Data Lake
Data Warehouse
Report
Report
Report
Analysis
Model
Model
Dashboard
Slides
Service

Figure 1-8. Data team evolving toward the use of automated decision making with AI

As time passes and the whole system scales issues inevitably begin to occur As noted in the Google paper, Data Cascades in High-Stakes AI[6] there is a 92% prevalence for at least one data issue to happen in any project.

Because the system has evolved so much and so rapidly, these issues are hard to troubleshoot and resolve. What is happening? No one knows, in fact.

The lack of visibility within the system combined with its complexity makes it like a complete black box, even for those who built it (some time ago).

The organization's data ecosystem, see Figure 1-9, has become like a "legacy system" in IT—only it happened faster.

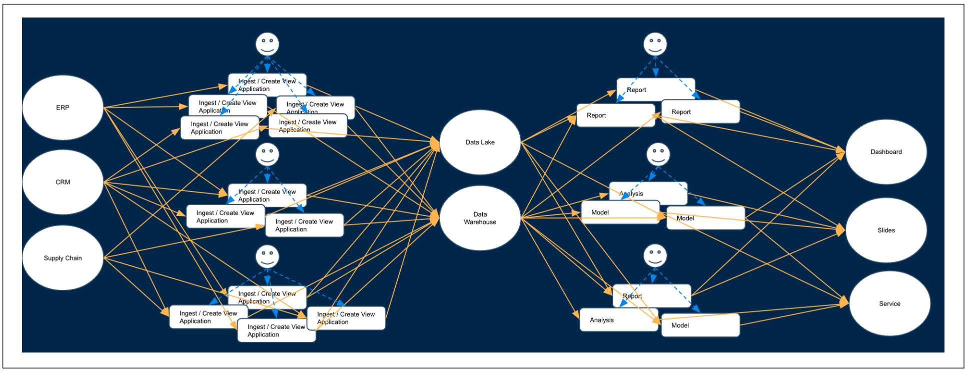

ERP
CRM
Supply Chain
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Data Lake
Data Warehouse
Report
Report
Report
Analysis
Model
Model
Report
Analysis
Model
Dashboard
Slides
Service

Figure 1-9. Efficient data team delivering projects in production after some time

In further sections, I will drive you throughout the journey of handling those issues and their consequences on the data teams' dynamics. Although, before that, let's analyze what is the anatomy of such data issues.

Anatomy of data issues and consequences

Data issues are a constant and common challenge for most organizations. In one survey by Dun and Bradstreet[7], 42% of businesses said they have struggled with inaccurate data.

Data issues can be caused for any number of reasons, but the most common are:

[6] https://storage.googleapis.com/pub-tools-public-publication-data/pdf/0d556e45afc54afeb2eb6b51a9bc1827b9961ff4.pdf

[7] https://www.dnb.co.uk/content/dam/english/dnb-data-insight/DNB_Past_Present_and_Future_of_Data_Report.pdf

Regulatory

Changes in data privacy or other data regulations may require changes in how data is collected, ingested, or stored, which can create unforeseen issues.

Business demands

Different business use cases may require different configurations of the data. One business use case may not require the use of addresses, for example. So, a business user may request that the address column be removed from the data. However, someone else using the same dataset may need addresses for their use case, so their analysis is now incorrect when this information is left off the data.

Human error

Often, data issues are caused by simple human error—someone accidentally deletes a field or column without realizing it. The data then becomes inaccurate, missing, or incomplete.

One of the most challenging aspects of what causes a data issue is that those involved in creating the change to the data or application often don't realize the implications of the changes they've made. And, unfortunately, the issue isn't usually discovered until the end of the data value chain.

Usually, as we discussed in the scaling data team story, it is business users who are running reports and realizing through a gut feeling and their own previous experience using the data that the numbers "don't look right."

But, at that point, it's already too late. Business decisions may have already been made based on faulty data before the inaccuracies were discovered.

With no time to fix data issues, data teams scramble to figure out who has the knowledge and skills to help resolve the issue. Yet, it's often not even clear who is responsible or what knowledge and skills you need to address the issue. Analysts? Engineers? IT?

And the responsibility can change from one moment to the next. Perhaps the analyst made a change in how some information is calculated that is now impacting the sales reports, but perhaps even before that, the data engineering team adjusted one of the connections supporting the sales analytics tool the business users use to run the sales reports.

To try and resolve the issue, everyone is relying on everyone else's memory about what they did or didn't do and are manually trying to find and fix the issue. No one has a clear understanding of what fields or tables affect downstream data consumers, and the only notifications they have set up are basic failure alerts.

The expense and time involved to resolve the issue and its negative impact on businesses' productivity, sales, overall revenue, and even reputation are significant.

The Dun and Bradstreet[8] report also tells us that almost one in five businesses has lost customers due to incomplete or inaccurate data. And, nearly a quarter of companies say poor quality data has led to inaccurate financial forecasts.

Constant data issues can lead to a lack of confidence in making business decisions based on data insights. In a recent study of 1,300 executives, 70 percent of respondents said they aren't confident the data they use for analysis and forecasting is accurate.

Impact of data issues on data team's dynamics

First and foremost, the data issue is detected by the user of the data who starts having doubts because either the data received seems odd compared to what was expected, or the results of the analyses are not what was anticipated.

> This case could happen even if the data has no "issues", because the data changes naturally as it represents the reality, which changes without notice - and so the user might not yet be aware. However, for simplicity, I will assume for the remaining of this study that the issue is real.

Figure 1-10 depicts the current situation, where one of the consumers is starting to have concerns about the data, and who imagines that the issue just discovered may have other, yet to be discovered, consequences in some or all applications that use it (a.k.a. data cascade, issue propagation, …).

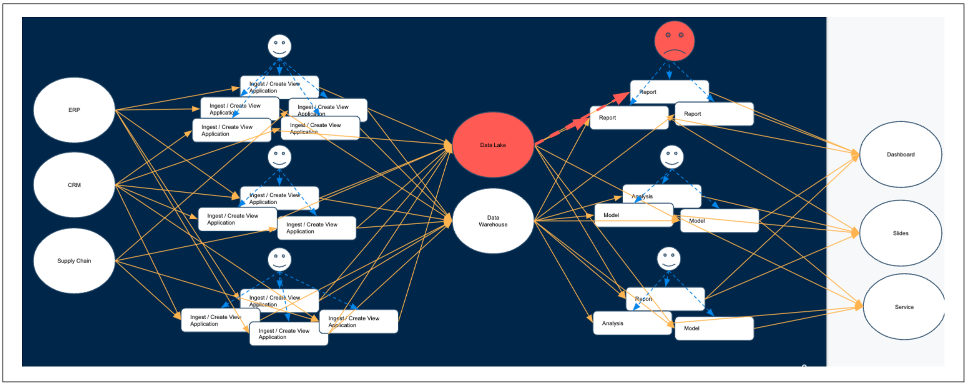

ERP
CRM
Supply Chain
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Data Lake
Data Warehouse
Report
Report
Report
Analysis
Model
Model
Report
Analysis
Model
Dashboard
Slides
Service

Figure 1-10. A user not cool about what has been discovered, a data issue

[8] https://www.dnb.co.uk/content/dam/english/dnb-data-insight/DNB_Past_Present_and_Future_of_Data_Report.pdf

In this scenario, users always detect issues at the wrong moment for them. They need the data at that moment, otherwise, they wouldn't be accessing or checking the data anticipatively.

So the user is stuck and of course, becomes grumpy. Indeed, instead of working on the tasks that were planned, the user has to find out how the data will be "fixed". Consequently, the delivery of the tasks is delayed, which is going to impact the stakeholders.

Moreover, finding how the data can be fixed is not straightforward as the received data might have been corrupted by any of the applications that produce it. At this point, the process to follow depends on how the (data) operations are organized, for example, creating an "incident" (a ticket) with a comment "data is not right please fix ASAP".

Eventually, the data has a data steward attributed to it. The data steward has the responsibility to unblock the user (at the very least) and to analyze the potential other consequences, which may result in more incidents.

To resolve such an incident, the data steward will have to involve the producer (or the producers…) who will need time to diagnose the symptoms and elaborate on a solution.

Well, this sounds easier than it really is. The incident needs to be reassigned to the producer, who is probably already busy with ongoing projects and unlikely to identify (the issue) instantly. That's assuming they are even the right producer!. Instead, all producers will be contacted, probably summoned in a meeting, to understand the situation. They will challenge the "issue" (e.g., "are you sure this is an issue"?), and ask for more details, which loops back to the already angry user (who is most likely not going to be very cooperative).

Consequently, the applications that are touching this data and identified as potential culprits have to be thoroughly analyzed, which involves those tasks:

- Access to the production data to manually verify and understand the data issue by running a series of exploratory analyses (e.g., queries, aggregation computations in spark, …). Given that, the grant to access the data might not be granted yet.
- Repeating this operation by time-traveling the data (when possible) to identify when the issue started to show up (e.g., a delta table can help to do the time-traveling, but still the time needs to be found).
- Access to the production logs of the applications (by SSHing on the machines or from the logging platform), to analyze its behavior and even review its history (version, etc), in case the application business logic has changed so did the

results. Given that, like for data access, the logs might require special grants to be accessible.

- Analyze the business logic to trace back to the data consumed by the applications to identify potential data cascades (issue propagation).
- Repeat until the root cause is found, which data and applications need some repair, and finally execute the back-filling (to reestablish the truth).

Easy peasy, right?

So, let's be honest, because time is critical (remember, the user is upset), it is likely that the root cause won't be tracked but either one of these two temporary-definitive patches will be applied:

- Run an ad-hoc script (or so-called data quality tool) to "fix" the data: which is totally getting the process out of control (e.g., outside the lifecycle of the data, etc.).
- Update one of the applications to "clean" the received data: which is fixing the problem locally and is likely to have awkward side effects; such as removing or imputing null values that would change the distribution of the variables if the number grows up and have side effects downstream (e.g., bad decisions…).

In fact, this process, called "troubleshooting", has puzzled many people, many of who weren't even needed for this incident. It is worth noting that while the troubleshooting was happening the scope of the issue has grown significantly, as shown in Figure 1-11.

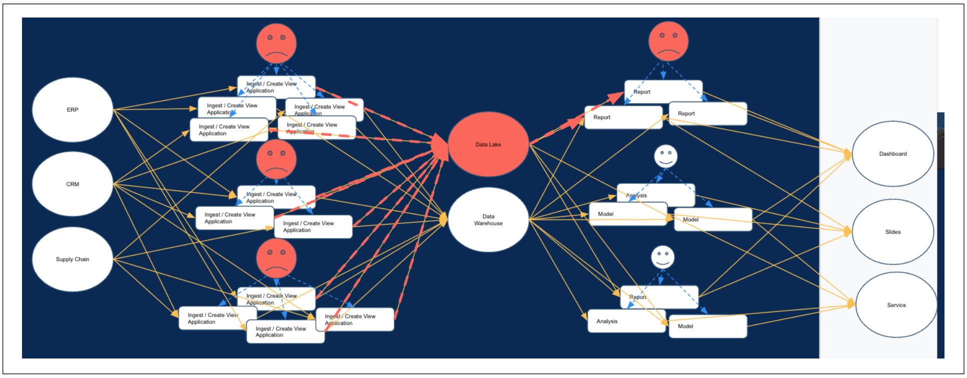

ERP
CRM
Supply Chain
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Data Lake
Data Warehouse
Report
Report
Report
Analysis
Model
Model
Report
Analysis
Model
Dashboard
Slides
Service

Figure 1-11. People involved in the incident analysis (troubleshooting)

Moreover, the issue detected on this data may have other consequences (remember, data cascades!) on other projects, users, and decisions. Which has the effect to expand further the scope of the issue to all usages of the data.

Another process called impact analysis is somewhat close to the troubleshooting we've just covered, however, its goal is slightly different, as it is aiming to prevent issues or communicate them.

In fact, not only is the goal different, but an impact analysis is also much trickier and more sensitive because it requires requesting time from all users to check their data, results, analyses, or decisions.

Even worse, some may discover that decisions were wrong for a longer period of time, potentially as long as the issue has appeared, everything happened silently.

At this point, the scope of the data issue, considering both the troubleshooting and the impact analysis, is as big as Figure 1-12 shows. And the data steward has to resolve this as fast as possible.

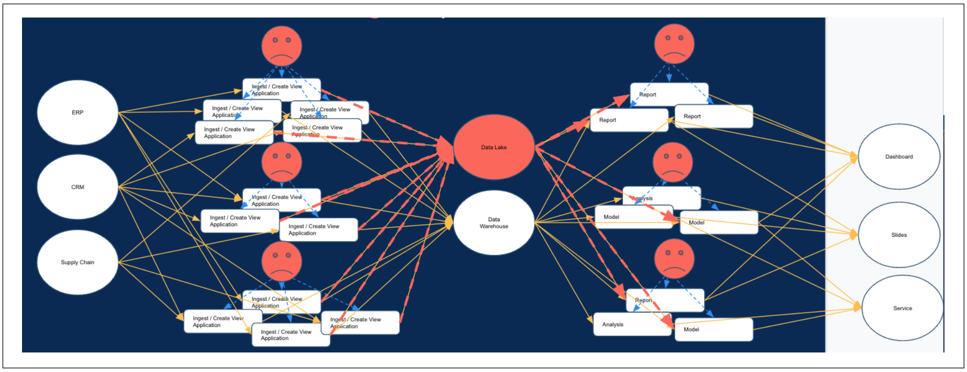

ERP
CRM
Supply Chain
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Ingest / Create View Application
Data Lake
Data Warehouse
Report
Report
Report
Report
Analysis
Analysis
Model
Model
Model
Dashboard
Slides
Service

Figure 1-12. The scope to resolve a data issue.

This is why we call data a "silent killer." It slowed everyone down, destroyed all trust, generated stress, anger, and anxiety without raising any alerts, or "exceptions" (as in software). In fact, data doesn't raise alerts, or exceptions *yet*, because there is a way to enable this capability we will see in the next chapter.

In this analysis, I have essentially highlighted the challenges around the time and the people involved in the process and the significant efforts wasted on ad hoc patching which just creates more technical debt. Despite the importance of those challenges, there is another inestimable resource that we lost instantly at the time when the issue was detected: trust in both the data and the data team.

This trust took months if not years (e.g., for sensitive AI-based projects) to build, but because nothing has been anticipated to maintain it, it fell apart like a house of cards, within seconds. This kind of scenario and the resulting consequences on the general mood lead to discouragement and turnover of highly talented team members.

Hence, we can identify many questions raised throughout the incident management process that are mood-destructors:

- Who is responsible for this issue?
- Why am I the one that:
  - — Discovers the problem?
  - — Is being questioned about it?
- Why is the user telling me the data seems inaccurate?
- Is really the data inaccurate?
- What apps could be causing this?
- What upstream data could be the cause?
- What are the unknown consequences of this issue (i.e., what other projects is this data used in that could also be impacted)?

Understanding why these questions are raised allows us to identify the very source of the problem, and it goes beyond any data issues and their causes. The challenge to address is not data itself, it is the lack of clarity about accountability and responsibility across the data team members and stakeholders, strengthened by the lack of visibility into the data processes in the wild (i.e. production, during its use).

Consequently, when the questions are raised by or to data teams, they create and reinforce barriers at the creation of value using data as they will:

- Take significant time to resolve during which the data is still unusable by the consumer.
- Require significant energy and imply that there is a diminished capability to solve the problem at the source and that a local "patch" will be the likely default solution.
- Provoke anxiety and exacerbate the loss of confidence in deliverables, data, and suppliers.

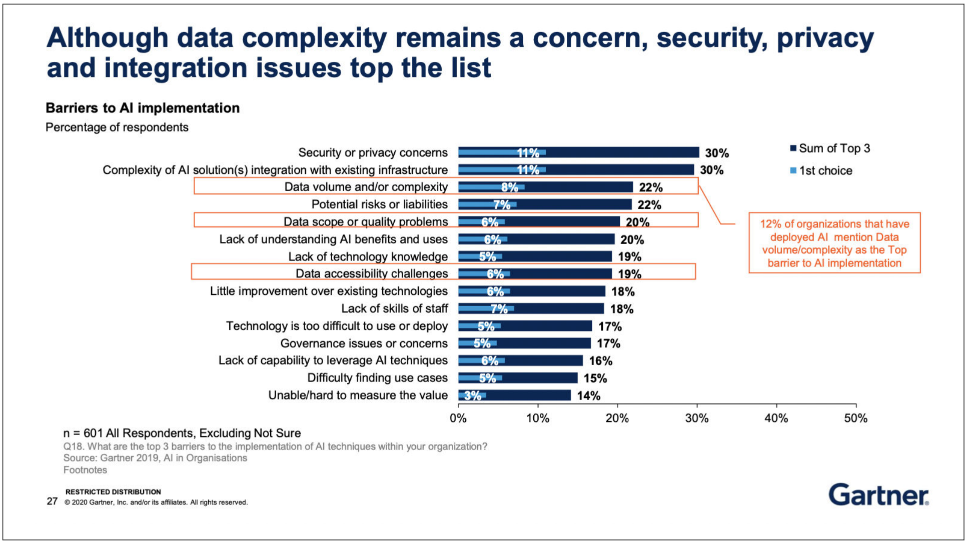
This is where data observability plays a key role—helping to generate greater visibility into the health of the data and the data ecosystem and better assign (distribute, decentralize) both accountability and responsibility.

Scaling AI Roadblocks

In the previous section, we covered the challenges that appear when scaling data teams. In this section I will focus on one of the most critical achievements that any organization is expecting from their data teams, scaling their AI capabilities.

However, these same types of visibility challenges that limit value when issues arise also create significant challenges when it comes to implementing AI.

In an analysis performed by Gartner see Figure 1-13 the most important roadblocks faced by companies in developing their data and AI programs included data volume and/or complexity, data scope or quality problems, and data accessibility challenges.



Although data complexity remains a concern, security, privacy and integration issues top the list

Barriers to AI implementation

Percentage of respondents

12% of organizations that have deployed AI mention Data volume/complexity as the Top barrier to AI implementation

n = 601 All Respondents, Excluding Not Sure

Q18. What are the top 3 barriers to the implementation of AI techniques within your organization?

Source: Gartner 2019, AI in Organisations

Footnotes

RESTRICTED DISTRIBUTION

27 © 2020 Gartner, Inc. and/or its affiliates. All rights reserved.

Gartner

Figure 1-13. Gartner survey's results about AI Roadblocks

The good news is that data observability helps with these roadblocks, but before we get into how, let's look a bit deeper at what the primary barriers are and why.

The survey results indicate that technology-related challenges account for 29% of issues and the complexity of existing infrastructure accounts for 11% of the issues. Interestingly, almost half (48%) of respondents selected challenges that have a relationship with a lack of visibility due to the complexity of the system in place and a lack of clarity on who was responsible for remedying the issues.

Let's take a look:

Security/privacy concerns or potential risks or liability

Nearly one-fifth (18%) of respondents to Gartner's survey picked these issues as their biggest barrier to AI implementation. Security of risk management is all about knowing who, why, and for what purpose the data will be used for. There is also a potential risk or liability if the outcomes from the data are inaccurate because the data itself is wrong and lead to bad business decisions. Finally, there is concern that the outcome may not meet certain ethical constraints.

Data volume and complexity

This issue was the top barrier for 8% of respondents. The complexity and the size of data require a lot of experimentation to understand and derive value from the data. Because of a lack of visibility on experimentations that were performed such as profiling and wrangling, these experimentations are repetitive on the same big and complex datasets–This takes time and effort.

Data scope and quality

For 6% of respondents, data quality issues were the top barriers. If the data quality is unknown, then it's very difficult to have any confidence in the outcome or final results produced from the data.

Governance issues or concerns, lack of understanding of AI benefits and uses, and difficulty finding use cases

A total of 16% of respondents felt that one of the above issues was the biggest challenge in implementing AI. Data governance is a big issue because documentation is manual and time-consuming, which means it's not always done properly and therefore its overall impact and value are not always apparent. But without good data governance, the quality of the data fed into AI algorithms could be impacted, and without visibility on the quality of the data, stakeholders may worry about the security of the data and whether the AI outputs are accurate.

So far, I have taken you on the journey of scaling data teams and AI, and we have identified several challenges such as lack of visibility (data pipelines, data usages, …), of clarity about responsibility and accountability, resulting in distrust and chaos, leading themselves to loss of interest or confidence into data ambitions. In the next section, I will describe what data observability is, and how this approach and capability solve those challenges.

Data Observability to the Rescue

Up to this point, I've discussed the challenges faced by data teams as they grow and the roles a lack of visibility and clear responsibilities play in making it difficult for organizations to scale their data and analytics strategies.

However, this is not the first time we've encountered such challenges, the closest example is in IT when it scaled rapidly, which led to developing the DevOps culture.

DevOps has evolved over the years, as more IT practices (e.g., the service mesh) have required more best practices and associated tooling to support them efficiently.

The most well-known example of such best practices is probably CI/CD but also observability at the infrastructure or application level has become part of any architecture to give more visibility and confidence in their applications and systems while also speeding time to market and reducing downtime.

Associated markets have therefore emerged and grown to support these observability requirements. There are a variety of companies that have developed DevOps-related services and technologies at all levels of maturity of the IT environment (e.g., Datadog, Splunk, New Relic, Dynatrace, etc.).

In IT, “Observability” is the capability of an IT system to generate behavioral information to allow external observers to reconstruct (modelize) its internal state. By extension, continuous observability allows an external observer to continuously modelize the internal state.

Fundamentally, an observer cannot interact with the system while it is functioning (e.g., we can’t log onto the server), it can only observe information that can be perceived, which are therefore called “observations”.

Now, let’s discuss now what those observations are.

The areas of Observability

An IT system is complex by nature as it is composed of several categories that can drastically expand in number, such as infrastructure, cloud, distributed, machine learning, deep learning, etc.

In this book, however, I’ll stop at getting too granular, but will aggregate the categories of an IT system that can be “observed” into several areas, and one of them is related to data, as represented by Figure 1-14.

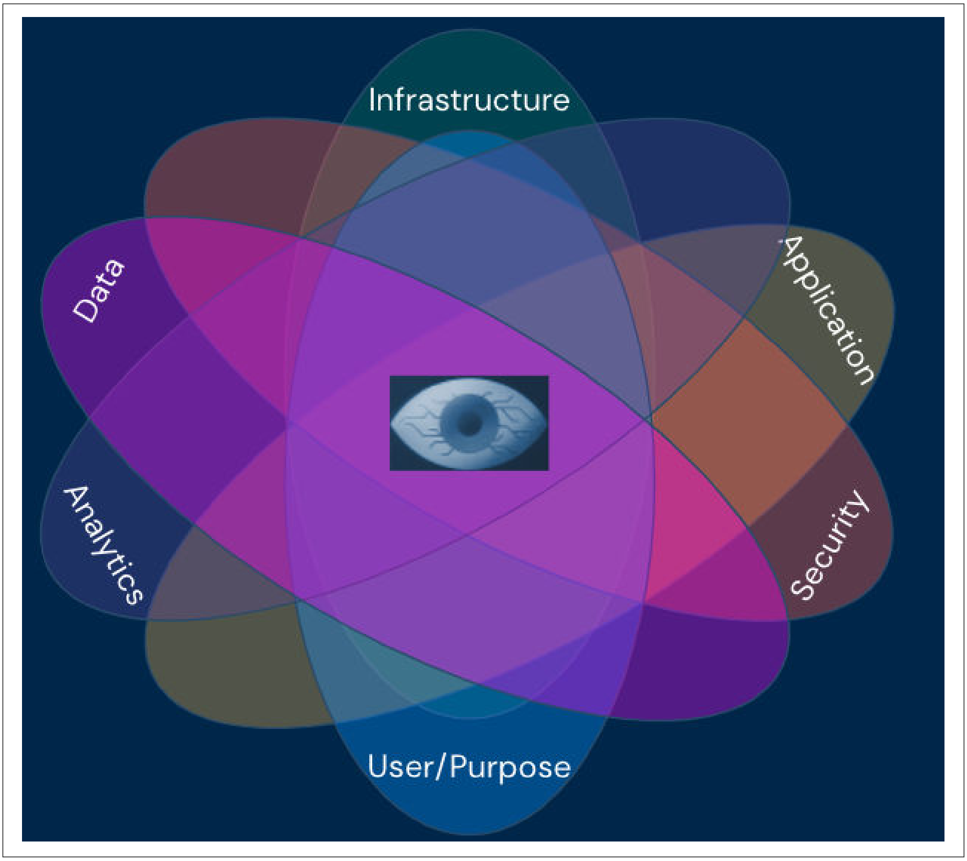


Figure 1-14. Areas of IT Observability

These areas are not totally independent of each other, as they have a lot of interactions to encode the complexities of the system. This is why a Venn diagram seemed to be the best representation.

Before covering in detail the "data" area, the "data observability" then, let's review the others briefly first:

Infrastructure
: Using infrastructure log metrics you can infer the performance characteristics associated with internal infrastructure components. Proactive actionable alerts can be set when there is a failure or certain performance parameters aren't met.

Application
: Observing application endpoints, versions, open threads, number of requests, exceptions, etc., can help determine how well the application is performing and identify if or why there are issues.

User/Purpose
: It is useful to understand and "observe" who is using or implementing applications, what the purpose of a project is, and the goal of the project. This helps to understand the most frequent projects or goals, detect duplicated efforts, or compose centers of expertize.

Analytics
: Observing analytics, from simple transformations to complex AI models, helps identify and learn from the ongoing usages of data and the insights generated from them.

Security
: Observing security-related operations such as modifications of grant accesses or roles, or metrics on which roles are used more often than others, gives visibility on the efficiency of the security and areas of improvement.

Some of the above areas have already been covered largely in the DevOps literature, however, we are focusing specifically on data observability. Therefore, we conclude that data observability can be defined as such:

> Data Observability is the component of an observable system that generates information on how the data influences the behavior of the system and conversely. An observable system is a system having the "Observability" capability.

It is worth noting that Gartner has defined data observability as the following, which aligns well with the above definition:

> Data Observability is the ability of an organization to have a broad visibility of its data landscape and multi-layer data dependencies (like data pipelines, data infrastructure, data applications) at all times with an objective to identify, control, prevent, escalate and remediate data outages rapidly within acceptable data SLAs.
>
> Data observability uses continuous multi-layer signal collection, consolidation, and analysis over it to achieve its goals as well as inform and recommend a better design for superior performance and better governance to match business goals.

The Gartner definition also discusses how data observability can be used (e.g. prevent issues). However, I didn't include this as part of my definition because I want to focus on what it is not what it does (also, I don't define an apple as a fruit that can satisfy my hunger).

That said, it is important to know the benefits of data observability. We'll dive into data observability use cases in more detail in the next section.

Nevertheless, both Gartner and I agree that there is an important "multi-layer" or "multi-area" component to be taken into consideration. I have, however, used the term "areas" because layers imply there is some independence, which, in reality, is not the case.

From this definition, we can look at the dimensions composing data observability as a result of its interactions with the other areas of observability.

I will start with the main dimension formed from observations related to the sole dataset. These are the core, or intrinsic, observations. They are essentially metadata such as the fields (e.g., columns, JSON attribute), the format (e.g., CSV), the encoding (e.g., UTF-8), and to some extent, the definitions of the information available.

They allow an observer to understand (mainly) the structure of the data and how it changes over time.

If we were to stop here, we wouldn't be able to leverage data observability to its maximum capacity. For this, we must connect those core observations to the other areas to give the observer the following benefits.

Infrastructure
: Identify where the data resides physically (e.g., file path on a server, the server hosting a database - ideally its connection string), and it could impact the data itself or the other areas cited below.

Application
: Be aware of what components are storing or using the data (e.g., a transformation script, a web service), it could also include the code repository and the version.

User/Purpose
: Contextualize and ease the knowledge handover with information about how was involved in the data production or consumption, its security settings, and in which projects (which have purposes) the data brings show value.

Security/Privacy
: Control how liable and appropriate are the data collections, accesses, and usages.

Analytics
: Understand how value is created with the data, through transformation (lineage), AI (machine learning training, predictions, …), or even simply migration.

: As for any new concepts, especially those that touch the organizational culture, ways of working, and technologies, it is crucial to understand their use cases. Therefore, in the next section, I will cover the most common and valuable data observability use cases.

How data teams can leverage Data Observability now

The main use cases of Data Observability are currently oriented towards data issue management. However, as data continues to proliferate and its uses expand, there will be many more use cases to come.

Low Latency Data Issues Detection

The more synchronized data observability is with the application (its usage context), the smaller the delay between the issues and their detection will be. In fact, data observability can be leveraged at the exact moment of data usage to avoid any lags between monitoring and usage. This will allow you to detect data issues as quickly as possible, helping you to avoid having data users find issues before you.

Leveraging data observability this way reduces the time to detect (TTD) issues as data engineers are alerted in a timely manner because any data usage issues are observed in real-time (aka synchronized observability).

Efficient Data Issues Troubleshooting

In most organizations, when data issues arise, data engineers spend a lot of time trying to figure out what the problem is, where it originated, and how to fix it. And every step of the process takes a lot of time. With data observability, the time to resolve (TTR) is much faster because there is visibility into the entire system thanks to contextual observability, which provides information on the data itself and the context of its usage. This enables data engineers to fix issues before they impact downstream business users.

Preventing Data Issues

When implemented as part of the entire development lifecycle, including production, data observability provides continuous validation of the health of the data and the data ecosystem. Continuous validation can perceptibly improve the reliability of the applications and prevent data issues, overall lowering the total cost of ownership.

Decentralized Data Quality Management

SLAs can manage and ensure data quality, just as they are used in IT DevOps to ensure reliability and other key metrics. This new managing data quality requires data observability, which on the one hand, can provide synchronized (near-real-time) and continuous validation of the data, which further improves the efficiency of any SLOs in place. But more importantly, on the other hand, data observability will allow SLA and SLOs to be defined at the granularity of the usage, and the context (the application, for example). This capability solves one of the most important roadblocks of data quality management programs, the definition of SLAs by owners, stewards, or SMEs who have a hard time defining a single set of constraints that will supposedly meet all usage expectations. Hence, they cannot come up with a single (central) set of SLAs as each use case is likely to perceive the SLAs differently. The key difference with data SLAs is that they can take a very large number of forms that feels quickly infinite; for example, you could have SLAs for the min representation, number of nulls, number of categories, skewness, quantile 0.99, etc. for a single field from a random CSV file. Hence, leveraging data observability to decentralize the SLAs in a contextualized manner (the usage) is key to managing data quality and defining a culture of accountability and clear roles and responsibilities.

Complementing Existing Data Governance Capabilities

Remember the DAMA-DMBKO2 data governance framework from earlier? Because data observability is part of the architecture and surrounds all of the areas of data governance, it provides visibility into all of the different components that interact at the data level. This enables data teams to automatically create documentation using the same kind of data, data storage, and data modeling and provides greater visibility into the different data models that exist, what data has been published to the data catalog, which analytics have been run on what data, and which master data was used.

The future and beyond

By better understanding the different use cases for data observability, you should now be able to understand how data observability can be used to help optimize your data systems as well as data teams.

In the coming chapters, I'll go even deeper, detailing how to set up these use cases and covering best practices to capture the information necessary for each use case in the most efficient and valuable manner.

CHAPTER 2

Components of Data Observability

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *gobrien@oreilly.com*.

As introduced in Chapter 1, data observability is an area of (IT) observability intersecting its other areas such as applications or analytics. In this chapter, we will cover how data observability, and its intersections, can be added to a system where data must be observed.

As discussed in previous chapters, data observability gives observers a broader spectrum of observations to interpret the internal state of the system by combining all areas. However, this combination can become a challenge itself if some precautions are not respected. This chapter will give you a deeper understanding of what observations are and what they should contain.

We will first review the three channels where observers can access observable information to interpret the internal state of the system in relation with data. Then, we'll see how this observable information can be organized to simplify this interpretation, its model. The chapter will conclude with the third component, that stretches the observability beyond information generated by the system, but mixes it with expectations observers may have about the internal state.

Channels of data observability information

The first component of data observability is the channels that convey observations to the observer. There are three channels: logs, traces, and metrics. These channels are common to all areas of observability and aren't strictly linked to data observability.

Below are definitions of each of the three main channels of observability. You are likely already familiar with these channels, but if not, there are hundreds of books and blogs that delve deeper into defining them. If you do want to read more on the topic, I recommend the book *Distributed Systems Observability*[1], which dedicates all of chapter 4 to defining these channels. I also recommend part two of *Observability Engineering*[2] as well as the REF.

Logs

Logs are the most common channel of observation produced by the IT system. They can take several forms (e.g., line of free-text, JSON) and are intended to encapsulate information about an event. A *line* of a log (typically logs are a stream of lines), is the result of the act of logging.

In IT, logging is a decades-old best practice, especially in infrastructure, application, and security. Logging has been used to not only debug but also optimize IT systems or processes. There are even developed standards for logs, such as Syslog, that specify the log structure and even allow heterogeneous infrastructures to be controlled by a central system.

While logs are crucial to capture information about the behavior of a system, it is difficult to use logs to recreate a multistep process. This is because logs comprise all activities within the system and the logs of a process are likely interwoven with other concurrent processes or scattered across multiple systems (e.g., distributed system, service mesh).

Traces

Traces allow you to do what logs don't—reconnect the dots of a process. Because traces are a representation of the link that exists between all events of the same process, they allow the whole context to be derived from logs efficiently. Each pair of events, an operation, is a span which can be distributed across multiple servers.

Traces, with their spans, are an efficient way to follow operations across services and servers, as information such as the server, the service, the timestamp of the event are

[1] Cindy Sridharan, Distributed Systems Observability, O'Reilly 2018

[2] Majors, Fong-Jones & Miranda: Observability Engineering, O'Reilly 2022

conveyed. So an observer can easily browse the logs of the service, on the server, at the given time to analyze the logs of the specific event for which they need to observe.

In practice, however, spans are also conveying logs that might be relevant for the observer to analyze on the spot, instead of having to reconnect the trace information. Even though observability systems allow integrations between traces and logs to simplify further this process.

> A different form of trace in the data and analytics context is the data lineage (see section REF). Although data lineage is not strictly speaking a trace (because it doesn't connect events), the data lineage connects data sources to encode the provenance or the generational model. However, in certain situations, data lineages can be used to help troubleshoot a process or discover opportunities to optimize.

The last channel of observations is metrics, which is closely connected to logs and traces, as both can contain metrics. However, because of their simplicity, metrics have a significant advantage over logs and traces.

Metrics

Every system state has some component that can be represented with numbers, and these numbers change as the state changes. Metrics provide a source of information that allows an observer not only to understand using factual information but also leverage, relatively easily, mathematical methods to derive insight from even a large number of metrics (e.g., the CPU load, the number of open files, the average amount of rows, the minimum date).

Due to their numerical nature, they need to be recorded. This is why metrics were once part of logs as well as present in traces. Because their usefulness is so straightforward, over the years, we have seen standards to simplify the publishing or collection of metrics independently of regular logs and traces.

Based on what I've described above, it may sound odd to talk about logs, metrics, and traces as being produced by data (a number such as "3" cannot log, measure, or trace anything, it is a number). This is why it is so vital to consider data observability as an area that intersects others, such as applications and analytics, which implies that the information produced by logs, metrics, and traces from several areas must be connected. Let's tackle this in the below section.

Observations model

Because observations can be captured from different channels, in different formats, and are related to several areas of observability, it is important to model them, at least minimally. In the following sections, I will present a model to encode the dimensions covered by data observability that support the solution for the use cases presented in Chapter 1.

I want to emphasize that the model I'm proposing is one I have seen work in many projects and use cases. Though the model may change over time to include more complexities or encode more relations, I intend to show you in detail how the observations from the different areas can work together easily as long as the bridges across them are well enough anticipated to avoid reverse engineering or approximations.

This model will, at the very least, give you a viable starting point to generate information that represents the state of the system you need to observe. However, you can consider it as a core model which you could extend with additional, potentially custom, needs. In chapter 3, I will explain the different strategies you can use to generate automatically most of the information of the model, and, in chapter 4, the associated ready to use recipes for several common data technologies (e.g., python pandas, apache spark, SQL).

> It is true that other areas of observability have not necessarily introduced such a model yet, and this is limiting their ease of use. Having logs, metrics, and traces coming from many infrastructures or applications without a clear structure to recombine them (the intersections) makes the process of generating insights about the internal state cumbersome.
>
> The information are therefore recombined in a best effort manner, using a posteriori analyses which depends also on human intervention.
>
> At the time of writing, Data Observability is raising out of the water, therefore my intent here is to avoid repeating the mistakes made in other areas, due to the youth of Observability, with the introduction of a model to 1) accelerate its adoption 2) ensure its ease-of-use.
>
> Ideally, ISO specifications will be created, endorsed, and widely supported, hopefully based on this model, in the meantime, let's use common sense.

In Figure 2-1 below, the model is designed as a graph to define the entities providing some information about the state and how they are linked to each other. The links are clearly identifying the intersections between data observability and other areas. For example, in the section below, you will learn that a data source (data observability) is linked to a server where it is accessible (infrastructure observability), hence the link itself is part of the intersection of these two areas.

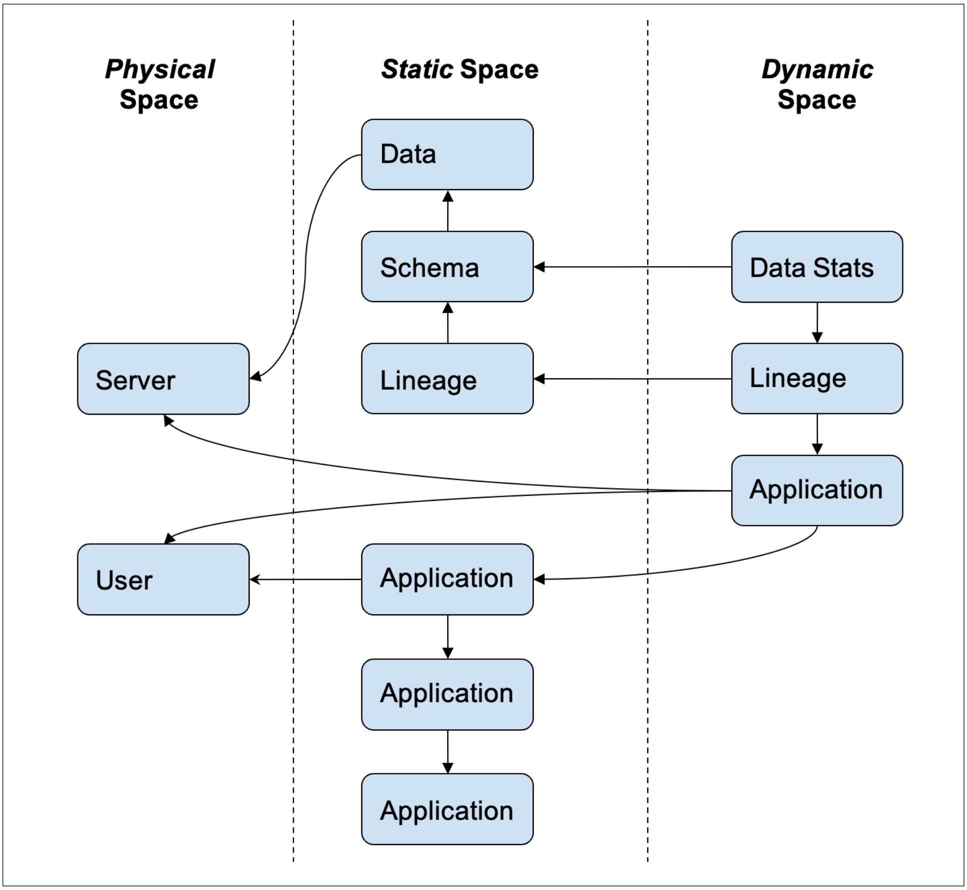


Figure 2-1. Fig 2-1. Data Observability Core Model

The model presented in Figure 2-1 aims at structuring data observations to maximize the overall interpretability for an external observer, and to provide a common language that can be used independently of the people, the time, and technologies.

To improve the readability of the model, it has been split into the three following spaces:

- The *physical space* links observations with tangible entities. Even though a server might be virtual or a user is a system or applicative user, the physical space represents a space that can be physically checked.

- The *static space* represents the entities that are changing relatively slowly in opposition to the dynamic space. It represents a state of the system that could be built manually.
- The *dynamic space* is part of the system that is evolving or changing so fast that it cannot be consolidated, documented, or built manually. This is where automation is not even a question–but a necessity.

Within each space, there are entities introduced to enable some observability. Let's go through each of these entities to understand what their purpose and how their relation to other entities play a role in data observability.

Physical Space

In this section, I will cover the information in the Data Observability area that can be considered tangible: the server and the user (or their virtual alternatives). Let's get started with the server first.

Server

The *server* is related to the machines which are running the IT system being observed. It is meant to provide information about physical appliances. However, in the era of cloud computing, it can also represent a container, a virtual machine, or similar that opens the door to access infrastructure observations.

Because I am defining the core model, I kept the server simple without encoding containers, virtual machines, and clusters for example – which can be extensions of the model. However, the core information about the server must help the observer to tap into other systems (e.g., Datadog) to analyze the infrastructure status.

To ensure this capability to the observer, the type of information the server must convey includes IP, container id, machine name, or anything that can identify the underlying infrastructure appropriately.

In [Link to Come], we'll see an example of how and when the server helps the observer.

User

The *user* is probably the easiest actionable information that can be used by an observer to improve their understanding of a situation. This is because the user mainly has knowledge and use of the system.

Like the server, the user can be a virtual abstraction, such as an identifier that doesn't hold personal information (e.g., GitHub account id) or a so-called system or applicative user such as a *root* or *spark-user*, that gives the observer a hint on the impact of the user on the behavior of the system (e.g., security) or its liability (e.g., privacy).

In other words, the user is a way for the observer to connect the status of the user with the purpose of observability.

In later sections related to the applications within a system I'll discuss the advantages of having such information on hand.

While I have limited the physical space to users and servers, we may think of other elements such as GeoLocation that can also be part of the physical space.

However, I recommend starting with a simple model to get the core visibility needed, then eventually growing it with additional use cases. That said, GeoLocation does fit nicely in the user and infrastructure observability areas.

Now let's address the static space, which includes most of the exclusive data observability entities.

Static Space

Whilst the physical space allows the observer linking events about data to "things" that can be accessed (even virtually). In this section, I introduce the static space that gives the observer the ability to analyze the status of the system at rest, or where the time has less influence than the physical space on its status (e.g., the existence of a server, the code change of a user). The entities are related to data and applications, and their structural evolutions.

Data Source

The *data source* provides information about what data exists and how it is used–whether that's a simple CSV file, Kafka topic, or a database table. The local file path, the topic name, and the table name are the kinds of information that are typically part of the data source.

The data source is present in the static space because data sources do not change rapidly. Of course, files can be copied or moved to other locations, but the model would consider the copied or moved files as new data sources. Renaming a table in a database is also similar to moving it. It's important to note though, that renaming and moving data sources can have big consequences. Applications can fail as they expect the data to be at a specific location, but the location may change in production because the data has been moved (e.g., `mv`). In such cases, there will be several data source entities, one for each data source location, as in fact moving a data source from one location to another is copying then deleting it. This action of copy then delete is actually a transformation which will itself be represented by its own entity, the lineage introduced further in this chapter.

The data source is also an element of observation that resides in the main component of the data observability area and is linked with the server, which is part of the intersection with the infrastructure observability area.

Below are some use cases highlighting where you can gain observability of the data source:

- Data migration to/from another server
- Data duplication or redundancy
- Data availability on a server (e.g., server migration or upgrade)
- Data access (i.e. if a machine is physically connected to access the data)

However, the data is not only a source, it is a repository for several types of information. In the next section, we'll explore what this means.

Schema

While the data source introduced above provides information about how data can be accessed, it doesn't cover what type of information the user may find in the data.

The Schema entity provides visibility into the structure of the data source[3]. The schema is an essential component of *metadata* (others would be definition, tags, etc) and conveys the information about fields (or columns), potentially deep (embedded), available in the data source. Each field of the schema has at least a name and an associated type, which could be *native* (e.g., string, int, address) or *conceptual*[4] (e.g., address, SKU_ID) or both.

Even though the schema is in the static world, it is still a changing entity, as a database table can have columns added, removed, or renamed. However, the schema makes it relatively manageable, although cumbersome, to keep track of the changes.

The schema is linked to the data source because it allows the observer to identify the types of information available in a data source.

Because the schema of a data source is likely to change, the model can encode these changes by keeping all versions of the schema in two different but non-exclusive methods. Each method supports interesting use cases:

1. Each modification of the data source's schema creates a new schema entity, along with the time of change or version.

[3] The name Schema relates to concepts such as XML-Schema, JSON-Schema and alike.

[4] As in ontology's concepts.

2. Each consumer of the data source creates a schema entity related to its own usage of the data source.

Interestingly, the schema, depending on the semantics presented above, is an observation that is either part of the (a.) main component of data observability or (b.) an intersection between data and analytics observability.

At this point in the model, we are capable of recovering decent visibility of the data. However, we still lack a fair amount of information about how the data plays out with the remainder of the system, especially applications and analytics. The remainder of this section will focus on how to achieve data observability within these two areas.

Lineage

The *Lineage* entity, or more precisely the *technical data lineage*, is probably the most difficult, and rarest, information to uncover. There is a lot of literature on how a lineage can be used (REFs?)(I'll list several at the end of this section), but there is little about how to collect it concretely.

Lineage, literally the line+age, refers to the direct connections between data sources. The lineage of a data source is therefore the set of direct upstream data sources and their lineages. The technical data lineage can be either at the data source or field level. Considering that the schema level is more complex to generate than at the data source level as it provides information on how data sources' schemas are connected to each other. That results in a connected graph where each field of a data source has its own lineage that connects it to the fields of the upstream data sources, as shown in Figure 2-2.

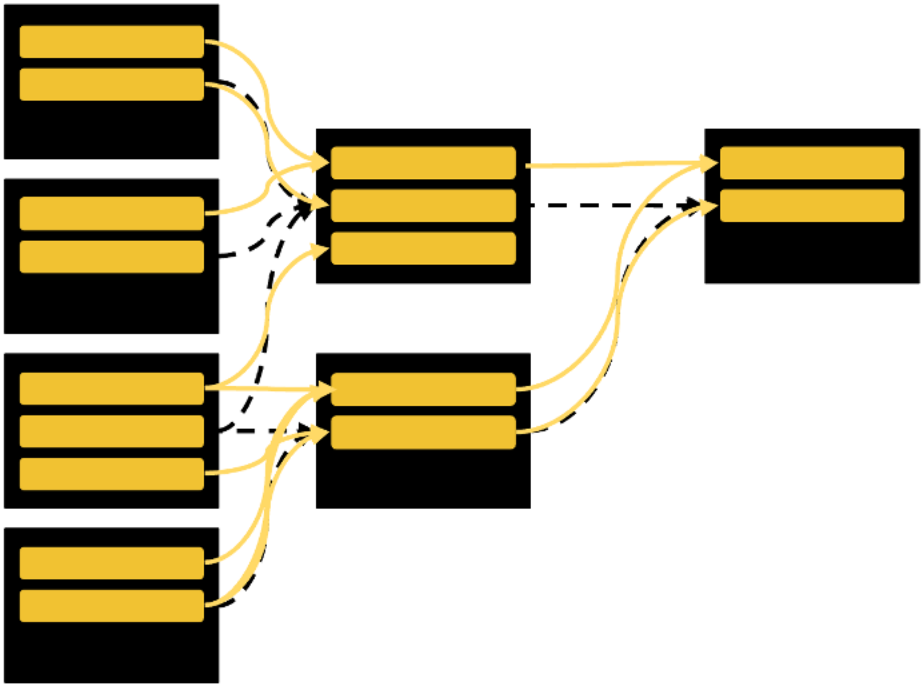


Figure 2-2. Fig 2-2. Example of technical data lineage - in black = data source level, yellow = field level

The connections between data sources are created using applications that execute *transformations* of inputs (sources) into outputs (results). These transformations can include:

- A SQL query issued by a Java web server transforming a table (input) into a JSON HTTP response (output)
- SQL generated by a reporting tool computing and presenting (output), a key performance indicator from a series table (inputs).
- A Spark job that creates a view in a data warehouse (output) from a number of Parquet files (inputs).
- A Kafka application using a streaming API that consumes a topic (source) and produces another topic (output).
- A machine learning script training a model (output) on a feature matrix created from a CSV file (input).

Even a human copying figures from a sheet of paper (input) into an Excel sheet (output) is a lineage.

The information contained in the lineage is essentially the mapping between input and output data sources and/or schemas.

To encode this mapping with the data source and schema, the model presented in Figure 2-1. shows that the lineage is linked to the schema which enables the following use cases:

- [Use Case To Come]
- [Use Case To Come]

Although conceptually simple, the complexity of lineage is because it is:

- Cumbersome to document: All data sources must be documented, and each operation applied to all fields as well.
- Likely to change: even a simple modification of a SQL query will change the data lineage.
- Numerous: There are so many transformations happening continuously on a company's data. Even a simple CRUD application is composed of many transformations.

Despite these complexities, I have left lineage information as part of the static space because a lineage relies on humans who transformed data to create their desired output.

When considering lineage outside of the main component of data observability, we can see that it also fits well with:

- Analytics observability: Lineage contains the structure of the analysis and how it is performed with data.
- User/purpose observability: Lineage encodes what can be achieved with the data sources.
- Application observability: Lineage is executed by an application, and a data application exists to execute lineage.

There's a lot to explore when speaking about the contribution of lineages to user/purpose observability and application observability, which we'll explore further.

Application

The *application* gives access to another part of the data observability because it is key to providing visibility of the data. In fact, the applications are at the center of all information: they are the product of the user, the container of the analytics, the consumer and producer of data, and they run on infrastructure while also configuring or implementing the security.

It is essential that information about the application is recorded so the observer can understand which system is at play and the associated logic.

I tend to consider the application like a door to the other areas of observability, because the application is relatively central to everything. In fact, referring to an application can be as simple as:

- The *groupId: artifactId* of a Spark job
- The path to a workbook in Tableau
- The organization/module_name of a python script
- The URN of a step in an orchestrator descriptor (e.g., GCP Workflow) scheduling an SQL
- etc.

Figure 2-1 shows that the Application does not refer to anything but still has the most central position in the observation model, it must be executed. I will explain how this is done in [Link to Come].

One could argue that lineage should be directly connected to the application. For example SQL joining two tables in Oracle and creating a view in Snowflake is coded in the application that will run it, hence creating a direct connection between both entities.

However, I believe that this view couples Data and Application Observability areas too strongly in the static space. There are several cumbersome instances where this direct connection is completely inappropriate:

- Lineage modification: When the lineage is modified and a new lineage is created, the initial lineage is no longer part of the application. In this case, the lineage should be connected with the Application Version (see below).
- Lineage is generated at runtime: When the lineage is generated at runtime, such as SQL created on the fly, all the SQL is not existing in the application until it runs and is used. In this case, the lineage would be connected to the runtime (execution) of the application (see further Application Execution in the Dynamic Space).

Of course, the application is not disconnected from the lineage, it just won't link to it directly. Instead, it plays the role of a container by embedding most of the observation model via the two components, the Application Repository and Application Version.

Application Repository

The *Application Repository* is often seen as the outlier in the model presented in Figure 1., because it gives the information about where the source of the application resides (e.g., source code, report configuration).

At first glance, there seems to be very little, or no connection to the data. However, the information provided by the Application Repository is clearly intended to encode the connection between Data Observability and Application Observability.

Because an Application can have its source moved around, the *Application Repository* can represent its latest location (hopefully not several at the same time) and gives a hint about where it resided in the past. This notion of temporality is something that I have chosen to encode in a different entity–in the Application Version.

One of the primary usages of the repository is to give the observer the direct location where they should be analyzing the system. For example, if there is an issue with the generation of a data source by an application, knowing where the application repository is tells the observer exactly where to look. However, they will also need to know the version of the application to analyze the situation appropriately at the time the issue happened, which is where Application Version is useful.

Application Version

Application Version is the final entity –and one of the most critical as it is the glue between the other entities– in the static space. The *Application Version* is the entry point in the Observations model that refers to the exact version an application was running when an issue occurred with the data.

The version can be the version number (e.g., v1.0), a git hash (e.g., d670460b4b4aece5915caf5c68d12f560a9fe3e4), or even simply a timestamp (e.g., 1652587507).

The version allows the observer to browse the history in the Application Repository for the running Application under supervision. The version is also useful when conducting root cause analysis. If the version is too old, it may indicate that the application has not been (yet) updated. If the version is unknown/newer, it may indicate that a bug in the logic has been introduced or an unknown transformation has occurred.

You'll see that the version is not directly connected to the lineage or any entity closer to the core Data Observability entities (e.g., Data Source, Schema). Thus, to create a bridge between Application and Data Observability, we must take into account what is executed, read, transformed, and produced. This is what the Dynamic Space is for.

Dynamic Space

The Dynamic Space gives the observer the capability to leverage the behavior of the system using the data. Its main purpose is to create visibility on what is called the *runtime,* which is highly dependent on the status of the environment in which the system is at play (i.e., its runtime context).

Also, the status of the environment is very sensitive to events that occur during runtime, such as the data changing due to real-world events, the application being killed due to a virtual machine reboot, etc. In other words, it is very dynamic and unpredictable (at least deterministically), which is why I call it the *Dynamic Space*.

The composition of the Dynamic Space is described in the structure that follows, starting with the Application Execution.

Application Execution

The *Application Execution* entity is key to providing the observer with information about the application that is running and its use of data.

As previously noted, the application is the artifact that opens the door to greater visibility of other components within the observation model because its source describes the analytics (e.g., lineage) using the data. By capturing the real runtime and its execution, the application execution conveys information to the observer such as:

- An ID: This allows the observer to identify which Application Observability information is relevant to review (e.g., in an application logging system).
- The starting date: This allows the observer to know, for instance, the period during which the aforementioned information is relevant.
- The current configuration and how it's changed across executions: This allows quick identification of incorrect settings.

The application execution directly connects to other entities, such as the application (what), the server (where), and users (who). Because the observer is aware of the application execution, and its connection to the behavior of some data usage within the application (see Lineage Execution and Data Metrics), the observer will be able to:

- Connect to the server to analyze how it may influence the execution, or even better, use the server observability information to gain this visibility without connecting to it (e.g., splunk, elk).
- Know exactly which applications are running or used to run on a server, and analyze its performance or load.

- Identify easily and quickly who has performed the execution and thus is likely to be able to help describe the situation.

But what about the connection with the application? The model presented in Figure 1. doesn't connect application execution to the application, because what is most important for an observer to know about the execution of an application is its version. Additionally, because the version refers to the source code, you can infer that it is also referring to the application.

Having the information about which version is being executed is extremely powerful. This information provides instantaneous insights, such as the version deployed is not the latest, or which changes may have caused an issue (e.g., *git diff*).

Also consider that an application is likely to be executed in different versions across different servers (e.g., environments), making the information described above even more valuable.

Lineage Execution

The *Lineage Execution* entity is the less obvious, but also the most complex, to create because it will likely have a large number of instances.

A Lineage Execution simply gives information about the execution of a lineage. Therefore it can represent many different things, such as the execution of an SQL job, a Python transformation, a copy command, machine learning training, etc. Lineage Execution is meant to provide the observer with explicit information about how a lineage behaves, including how often and what time connections to data sources occur.

I wouldn't say that a Lineage Execution conveys a lot of information intrinsically but it is a keystone of the observation model. Without Lineage Execution, most of the use cases of data observability presented in Chapter 1 are fantasy, or rely on best effort reconciliation, which inevitably, leads to spurious correlations (mainly while reconciling the Data and Application Observability areas).

A lineage execution is rich in information about its connections to other core entities such as Lineage and Application Execution. If the link to Lineage is clear, then the observer can tell which lineage is observable and can analyze the link further.

A lineage can't be executed alone because it only represents the connection between data sources (schemas) not *how* these connections are implemented. This information is available in the code, which can be observed using the Application Version. For example, a data transformation could have been implemented in Spark RDD, then in Spark DataFrame, then in Spark SQL, then a materialized view in Snowflake, etc.

To understand how a lineage is executed, the observer has to use information about the application execution. You can think of lineage execution as an internal program that is started by the application, for example an application creates an entry in a table when certain events are consumed from Kafka, then it executes the related lineage.

Moreover, an application is not limited to running a single lineage or a lineage only once or even always the same lineages. Because of this diversity, giving the observer visibility of this extreme (nondeterministic) situation is essential - as it is unthinkable that this complexity could be held in a person's brain.

Due to the centrality of Lineage Execution in the model, the observer gains a holistic visibility (but not limited to) on:

- Which Lineage is executed by…
- Which Version of…
- Which Application in…
- Which Repository consumes and produces…
- Which Data Sources having…
- Which Schema executed by…
- Which User on…
- Which Server.

In graph lingo, we call this a traversal (collecting information while walking a graph). I call this a *freaking mind-blower*.

But we can extend further this visibility with data metrics. One of the exciting things about Data Metrics is that it has the potential to support data quality in a bottom-up, scalable, and real time fashion.

Data Metrics

Data Metrics is the specification of Metrics in Observability. Data metrics is often the most common suspect when discussing observability, as it is easily leveraged in analysis to infer or predict (future or dependent) events - e.g., using their temporality dependencies with multivariate time series analytics, Markov Chains, and the like.

In this context of Data Observability, I am focusing on the metrics that we can relate directly to data, such as descriptive statistics (see below for more examples). However, the data metrics entity comes with a subtlety that can have a huge impact on its usefulness–the metrics are always attached to their analytical (e.g., transformations) usage.

The role of Data Metrics is to give information to the observer about what the values of the data look like under certain conditions during their consumption or production. There are at least three types of metrics that can be considered:

- **Univariable**: a metric related to one of the fields/columns of the observed data source. For example:
  - Descriptive statistics of a column (e.g., min, max, nulls, levels)
  - Distribution (e.g., density, skewness)
  - Formula (e.g., the sum of squares, product)
- **Multivariable**: the result of the combination of several fields/columns. For example:
  - Formula (e.g, the sum of the products of two fields).
  - Combined (e.g., Kolmogorov-Smirnov)
- **Extrinsic**: the result of the aggregation of fields with fields from other data sources. This metric is tricky, and can even be considered a borderline metric, as it is too close to a KPI or a result.

It is important to note that these metrics are not intended to replace the exploration analysis required during the analytical phase of a data project (e.g., data science, report). Those metrics are meant to allow the data to be understood in the context of their usage, not before. In other words, the primary usage of the data metrics is not to provide potential users with the initial insights they may find.

Nevertheless, data metrics are key for their potential users to gain trust in the likelihood of the data being reliable, because if the data source is reliable for the other usages, there is a good chance that their own usage of the data will also be reliable. Remember that trust (particularly related to data quality) requires time and is stronger when the reliability of the data is tied to real and understandable usages.

> The *likelihood* in the sentence above is important because it is not fully granted that if a data source is reliable for the applications of user X, it will also be reliable for the applications of user Y. Users X and Y are likely to use the data differently. For example, imagine if user X computes the arithmetic mean of a field and user Y its geometric mean. In this case, how would a single unexpected zero impact each model[5]? And what if the data source is suddenly empty, how would that also change the impact on the different use cases[6]?

[5] The user Y ends up with the value 0, whilst the value of X has probably not changed a lot (especially if there are many values).

Hence, the data metrics must be linked to the data source it is exposing the *behavior* (the correct term should be *status*) to the observer. In Figure 1, I show that data metrics are linked to the Schema, this is because the metrics are highly dependent on the fields present in the data source during its usage.

Finally, those metrics have to be linked to the usage of the data source, which is represented by the Lineage Execution, which represents when the data sources combination is executed. This link comes with a never-seen-before power in observation models–giving the observer the ability to understand the status of a data system.

Expectations

So far we have covered the types of information and the entities that give an observer the ability to understand how data behaves in a system.

In this section, I am going to address the third component of data observability, which pertains to creating visibility when the data behaves as expected. As you'll see, the expectations can come from different parts of the data ecosystem (e.g., users, applications, …), and if expectations are visible to an observer, it gives the observer key insights into how the observed status of the data compares to how it is expected to behave.

By setting specific expectations, just as tests do in software development, the observer can better understand the behavior of the data aligns with expectations. Moreover, if some expectations are not met, the observer knows instantly what requires their attention, as, again, tests reports indicate to software developers where they need to focus their attention.

So they give essentially visibility about what the observer expect to happen, and they represent either a positive or a negative event. For example, knowing that an expectation, such as address always starts with the number, is true for 99% of the entries, ensures that the data that can't be used is only 1%, there is no incidents, it just provides the observer two informations: the expectation itself and if it is true or false. On the other hand, if a transformation is known to fail if a secondary is not complete, in this case, when it becomes false, an incident can be clearly created, triggered and an associated decision can be made (e.g., not executing the transformation at all).

When setting expectations, you will want to both set rules for how the data should behave as well as detect anomalies. The next section will look at both of these areas of Expectations as well as how an application can become its own observer and make decisions regarding the status of the data's behavior.

[6] Boom… division by 0. At least, Y would get an error this time :-).

Rules

A Rule is a common tool used in development and monitoring, it is basically a function that evaluates the state of the data. If the current state passes the rule, it will return a “true” response, and if it doesn’t, it will return a “false.” The function’s logic is often quite simple, making it so that interpreting the result is easily relatable to the inputs. For example, rules using a machine learning model to return the result are closer to an anomaly detection than rules.

> The Oxford’s definition of “rule” is One of a set of explicit or understood regulations or principles governing conduct or procedure within a particular area of activity.
>
> I don’t consider a trained machine learning model to be explicit, even if a fully explainable one would be borderline.

Therefore, rules for data observability would be created using information conveyed by Schema, Data Metrics, Lineage, etc. For example, if a Data Metric reports the number of null values for the field *age*, a rule could be the *number of null values for age must be below 10*, or *less than 5%*, or even *can’t increase by more than 0.01% between two executions of the same lineage*. Another example of using the Schema information would be *the age field must be present and of X type of integer*.

In fact, there are two categories of rules:

Single-state based
: These rules only act on the information about the state of the system at a given time, such as in the above example where null ages must be below 10. The state information could also be combined to create intrinsic checks, such as comparing the number of null values to the total count (e.g., no more than 10%).

Multi-state based
: These rules require more than one state, such as the states of one or several periods of time, continuous states, or a sampling. Multi-state based rules could even include a list of states randomly drawn from the available states.

While the entities of the Observation Model are computed or extracted from available information, the rules are less factual and require more insight to create. The insights needed can come from several sources such as:

- Human knowledge and experience
- History of the system
- Similarities with other systems

Let’s look at two ways to make rules.

Explicit Rules

Explicit rules are mainly created by users as a representation of assumptions that they are building up and which they expect to hold true when their applications will run and be used in production.

Those assumptions come mainly from the following channels.

Experience

A person's experience with the data or the addressed use case is a great way to generate rules because it is based on past experience or applied knowledge. I am referring to intuition or gut feelings that are the result of years of experience forming assumptions based on use cases or practical experience (e.g., networking, etc). However, it doesn't mean that those assumptions are always reliable. A person may be biased or misunderstand the full situation. Nevertheless, it is still worth considering these types of "instinctual" assumptions as they are far better than nothing and, more importantly, they express the expectations of the user. Then, if those rules are not met, thanks to the observability of the system, the team has the opportunity to perform an analysis to determine the rules' validity and potentially discard or adjust them.

Exploration

While working with the data to create the pipeline, the report, features, and other types of data transformations, a person will be learning, or at least better understand, the data at hand[7]. Therefore assumptions will be formed and integrated into the application based on these learnings. For example, if the data has no duplicates or a turnover always above zero, if you turn them into assumptions, you are likely not going to make them explicit or may not be specified in the business logic to be implemented–although those assumptions are a great source for rules. One way to help this process for developing rules could be to create *data profiles* (using profilers). Profiling data generates a series of static rules inferred from the data provided. However, these rules need to be used with caution and reviewed thoroughly, leveraging the experience of the team to select the most appropriate ones while avoiding introducing rules too close to the data at hand and therefore likely to fail rapidly in production. The usage of data profilers will be detailed further in Chapter 3.

Discovery

When application and data are in use, new cases can be discovered, such as unknown-unknown (unexpected cases) or corner cases (not considered in the business logic). When discovered, they are generally considered as incidents that need to be analyzed, and most likely fixed. As discussed in Chapter 1,

[7] Data at hand refers to the data available during the development phase.

there are several processes that must be conducted such as root cause analysis, troubleshooting, and impact analysis, to determine and fix the issue. Each of these analyses present opportunities to introduce new rules discovered along the pipeline.

It is also important to note that rules can be outdated as the world is constantly changing (e.g., products are boycotted, laws are adapted, platforms are upgraded). Consequently, the maintenance of these rules must be considered. I often encourage teams to perform a quick review of the rules, especially for experience-based ones, with every new version of the application using or producing the data at the very least. Validating rules justly requires a simple feedback flow; however, there are alternatives such as using assisted rules.

Assisted Rules

In the previous section, I introduced rules created entirely as the result of human analysis. Here, I will introduce an alternative method, which allows rules to be discovered.

Although rules cannot be based on non-explicit behavior, they can still be discovered using simple analysis (e.g., heuristics) or even learned (e.g., association rule mining). Such rules are created with the assistance of the observations. In this way, they are "data-driven."

The power of assisted rules is not only the capability to create new rules, but also to update existing ones, even the explicit ones. Done efficiently, assisted rules lessen the burden required to maintain rules in the long run.

The way to introduce assisted rules in a data observability system is by keeping humans in the loop. Because rules are structurally comprehensive, they can be reviewed by people who have the knowledge to tune, accept or reject them. Validating rules in this manner is important to increase the level of trust in the rules, and ultimately, the accuracy of the issues the rules detect. For example, an assistant system can estimate (e.g., based on self-correlation) that the *amount* seems to take its value around 19,945.62 with a standard deviation around 783.09, and therefore propose a few rules such as following some distributions– which could also either estimated from observations, or be an observation itself like the Kullback–Leibler divergence with a few distributions *a priori*. The possibilities are limitless, as long as we have enough observations at hand. That said, assisted rules should, over time, also leverage knowledge accumulated during the discovery processes, that is the rules created after the fact, which provides information on how to anticipate *unknown-unknowns*.

The opportunity to recommend and update rules is an advantage over automated anomaly detection, which I will discuss in further detail. However, such anomaly detection is, by definition, not involving the human much, the lack of control they provide and their randomness are not generating the level of certainty about what is under control. To increase and strengthen this feeling, which is the goal of Data Observability, I will first tackle the relationship between rules and Service Level Agreements (SLAs).

Connection with SLA/SLO

Service level agreements (SLAs) and service level objectives (SLOs) have been around for quite some time to establish contracts and are a way to avoid subjective disputes about the expected quality of the service. They help build a sustainable relationship between producers or service providers with their consumers and their users.

These relationships also exist in the data world. Data teams are, in a service fashion, producing data for dedicated use cases or to be made available for future consumption.

Thus, both data teams and data consumers/users have expectations, even if they are unspoken, undocumented, and not intuitively shared when it comes to data. This has created the issues I covered in the first chapter. SLAs and SLOs are good solutions to eliminating many of these issues.

Service level agreements for data are contracts between the parties involved comprised of expectations that can be met, and if not, a fine or other penalty is applied. The fine/penalty is potentially as important as the length of time that elapses in which the expectations are not met. Hence, operational metrics such as "time to detection" and "time to resolve" are key for sustainable success.

To establish an SLA in data, it is a prerequisite to have the input of the users to make sure the quality of the service will be high enough for them to actually use the data. The more users taken into account in the SLA, the greater the service, the higher the service levels, and the better the relationship is - as long as the associated cost to ensure them is acceptable.

There are several complex facts that need to be considered when establishing the SLA for data:

How to define the constraints

The user may have a hard time defining the important constraints due to the number of possible constraints. Considering that each field in a data source can have 10+ associated metrics (e.g., for age there can be *min*, *max*, *mean*, *std* but also the quantiles, the estimated distribution, the null values, the negative ones, etc).

Data usage will vary

Each user will have a different use case for the data, resulting in a different set of expectations and thus constraints.

High number of users

With self-serve data and analytics, the number of users is exploding, thus creating more data use cases and more constraints.

This is where data observability is essential. Data observability observations can be used as Service Level Indicators (SLIs) by providing enough information about both the data and user behavior based on which rules can be identified, assisted, and maintained.

Because both producer and consumer generate these observations, it becomes easier to find a consensus as to which KPIs from the SLIs that should be used to determine which SLAs are possible for the data team to meet. This will set the bar for the SLOs across all SLAs while keeping the door open for other SLIs to be used later during discovery of SLAs. In the meantime, both consumers and producers can use them as SLOs (or simply informative, through a notification system for example).

Without data observability the scope is too vast for both parties, which prevents them from committing to a set of expectations without sound evidence of their effectiveness, and a way to improve over time.

Automatic anomaly detection

Rules are a powerful way to ensure data matches expectations in the context of where the data is used. However, rules have a dependency on people to create and validate them (at least, this is what I strongly recommend). This dependency, on people's time and availability, is undoubtedly a bottleneck to scale exponentially with data validation. Although, companies reaching this stage need to have already reached a high maturity level.

Also, assisted rules have a constraint on their structure as they must be explicit and fully explainable, which may be a weakness over other more opaque solutions, such as learning-based rules.

Considering that rules are meant to intercept expectations that are not met, another method we haven't discussed yet is automatic anomaly detection using the data observability information available.

An anomaly is the state of the system which is different from the expected states. This is the connection between expectations explicitly encoded as rules and expectations implicitly discovered when unmet.

In our case, we have the states of the system corresponding to the history of the data observability information based on the core observation model. Hence, those states can also be seen as a stochastic process with many random but connected (non independent) variables (each instance of the entities could be considered as a variable).

Automatic anomaly detection, therefore, is the process of leveraging some of the learning-based methods listed below to infer or predict when a state should be considered an anomaly.

The methods that can be used nowadays are numerous and include machine learning:

Supervised learning
: This requires labeling data observability information (or the state itself) to work properly. Hence, it requires time to collect enough data to train the algorithm as well as time from the team to label (with high accuracy) the anomalies.

(semi-)Unsupervised learning or statistical analysis
: This method can help identify or predict potential abnormal states. These states can then be presented to the team for labeling. However, it is cumbersome to label the predictions as anomalies automatically, even in extreme cases (e.g., outliers), without a second review.

Reinforcement learning and active learning
: This method is quite advanced and still requires some time from the team. Not only would agents need a lot of simulations to be validated, the active learning is also essentially based on the human in the loop.

My intent is not to dig into each method, because it would require at least another book and there are already enough papers and books on each of those topics. However, let me emphasize that these methods, although capable of scaling in theory, still come with downsides.

The cold start problem
: In order for anomaly detection to be efficient, data and people are needed to ensure the learning phases are reliable.

The black box worry
: The above methods generate anomalies automatically but with little information about the reasons. Consequently, the team must reconstruct the context and understand the reasons.

The performance challenge
: Anomaly detection can't be 100% accurate. Therefore, their performance needs to be under control to avoid *alert-fatigue,* where the team starts to ignore the detected anomalies.

Coming back to the black box worry, it is important to note that nowadays a lot of effort is put into making machine learning models explainable (which is, in my humble opinion, a component of analytics observability). This is a challenge that is not fully solved, especially for methods such as deep learning. However, keeping an eye on the evolution of this capability is a must to reduce the effect of the black box worry.

I'll wrap up this section on anomaly detection by also mentioning another method rarely considered to address the black box worry, and moreover, provides an interesting mix of rules and anomaly detection – the learning of finite state machines. This method could be promising under circumstances where data observability information is growing in volume, especially if the rules having triggered are also labelled as true or false positives.

Prevent Garbage In - Garbage Out

If there is one cause for which the data community should be fighting it is the ban of the "Garbage in-Garbage out" excuse. This common excuse is generally used as the last line of *defense* when a data producer is being asked why the data is wrong. Often, with too much confidence, the explanation provided is, "you know, my application that produces the data works as expected, it follows the business logic requested, but it was provided with garbage data, so of course, it produced garbage results."

Although this excuse is used as an explanation or a way to waive the blame, it does little to placate the user who still has no real solution at hand, so it is considered, appropriately, as an excuse for not meeting data quality expectations.

In other words, it is a "dodge" that creates more harm than good. An opposing view is that if an application can be aware garbage comes in, it should be forbidden to generate garbage out. To do this the application needs to be able to

- Assess provided data as garbage
- Qualify the created data as garbage

The good news is that the third component of data observability, the expectations, addresses these exact requirements.

With Data Observability implemented, here are the different scenarios that are possible:

- Incoming data is classified as garbage before the application generates its results. The application can then either:
  - — Not proceed and avoid the garbage out
  - — Make best efforts to clean the data, ensure the cleaning process is observable, and then proceed
- Incoming data is classified as garbage based on the qualification of the results. The application can then either:
  - — Roll back the data and log it

— Log why the data is garbage, due to a posteriori discovery of garbage in

— Like in the first case, try to clean the incoming data or its output, still ensuring those processes are observable

The logic described above could be implemented using the two, non-mutually exclusive procedures. The first is the use of conditions.

Pre/Post-Conditions

Conditional statements are commonly used in programming, such as when implementing an HTTP endpoint. For example, if the endpoint's business logic is expecting a JSON payload encoded in UTF-8 that contains a non-null integer *age* field under the *customer* top-level field and these conditions are not met, the application can decide to return an error to the client or be lenient for certain cases such as the encoding by re-encoding before proceeding. Also, if the HTTP endpoint's side-effect is to update a table and then get the current state of the data, but it gets several rows instead of one, it can take actions such as rollback, delete, or simply return an error letting the client know about the potential garbage data it sent.

What I just described is something very natural for any programmer, it is even more than a best practice, it is intuitively used. Therefore, the idea would be to simply also use this pattern when creating data applications, pipelines, and so forth.

The interesting advantage of implementing pre and post conditions directly within the data application is that it is well guarded without depending on other external components to ensure its viability. Of course, the system used to build data needs to allow such practices, which have not systematically been addressed in some systems, especially the low-code no-code (e.g., ETL, reporting tools).

> In recent years, not only because of the influence of DataOps, which is a set of methods and tools to go to production faster, the data system has reduced the entry-level required by people to use the data. Therefore, the practice of setting of pre/post conditions is not necessarily well-known or seems like an additional unnecessary burden – as unit tests used to be considered in the early 2000s.
>
> On the other side, the whole world is putting a significant focus on the culture of accountability and respect for data.
>
> In my opinion, accelerating time to production, lowering the skills, and increasing the accountability are not playing well together, unless the data systems allow the people to also mature and take some responsibility. Some technologies have partially included this concept, such as *dbt,* which includes data tests but with the current limitation that tests are not executed in production. This is addressed further in this book (REF).

In the next section, I describe the circuit breakers, which conceptually are not so different from the conditions described here, but structurally give an alternative when the system does not (yet) have first-class support for the conditions.

Circuit Breaker

The circuit breaker originates from electrical circuits. In electrical systems, the circuit breaker is a safety device designed to protect an electrical circuit from damage[8]. This device closes the circuit to allow the electric current to flow until certain conditions are met that could break the circuit. In such an instance, the device breaks the circuit (opens it), forbidding the current to circulate until the underlying issues are fixed and an action is taken to close the circuit again.

This idea has been reused in software architecture by Michael Nygard[9] and supported by Martin Fowler[10], where an additional component wraps a function to prevent its execution under certain conditions. The prevention is as simple as returning an error directly when the wrapper is called and the *circuit is open*.

In a data pipeline, this may be more of an alternative to pre/post-conditions, especially in the case where the user has no capabilities to introduce conditions into the system used to create the pipeline, such as in low-code-no-code systems or restricted interfaces (whether graphic or API). Therefore, the system has extra components in the pipeline (circuit) to break it in case conditions are not met. In fact, implemented as such, circuit breakers are similar to pre/post-conditions external to the applications.

The nice thing with a circuit breaker is that they seem to be easier to add after the fact (especially good for legacy systems). However, they do come with downsides. For example, if the conditions depend on the steps before and after (e.g., case specific SLA/Os, custom checks, …), then the circuit breaker is yet another application to maintain and align with the steps' logic[11]. Hence, circuit breakers are likely to be present before and after each step, which makes the pipeline three times as long as it would have been if the conditions had been added in the applications directly.

[8] https://en.wikipedia.org/wiki/Circuit_breaker

[9] https://learning.oreilly.com/library/view/release-it/9781680500264/

[10] https://martinfowler.com/bliki/CircuitBreaker.html

[11] This is likely to result in troubles…

About the Author

Andy Petrella has been in the data industry for almost 20 years, starting his career as a software engineer and data miner in the GIS space. He has evangelized big data for more than a decade, especially Apache Spark for which he created the Spark-Notebook (that has 3100 stars on Github).

During his time evangelizing Spark and helping hundreds of companies in the US and in EU work on their data pipelines and models, he has witnessed the lack of visibility and control of data jobs after they are deployed in production.

Since 2015, he has been talking to tech and data-savvy people to build a sustainable solution for this problem. That is: "how to make data observable" in a way that can be adopted smoothly by any data practitioner.

Today, he is regularly invited to companies to educate their data teams, whilst running Kensu, which has more than 50 years of total development time dedicated to building the set tools to help data engineers and their peers to build trust in what they deliver.

Also he is in ongoing talks with advocates such as Gartner to create a definition of Data Observability that refers to all its important facets. Finally, he has written books, blogs, slides, training materials, etc. since 2013, including many materials with O'Reilly.