# Chapter 2 - The Modern Data Stack

Data tools and practices change all the time. New technologies arise in favour of old ones. Hyped technologies recede again, sometimes even bringing to the fore an old technology. At times, however, certain phases or periods can be identified where a bigger picture emerges.
Here, a key technology such as the cloud changes fundamental ways of handling or even thinking about data. At this point, these elements of the bigger picture turn into **paradigms**.

The current ruling paradigm in the data world is often referred to as **The Modern Data Stack** and it is often contrasted to **The Legacy Data Stack**, which honestly is not a good name because it simply suggests "the new one" has superseded "the old one". To salvage this, some writers differentiate it by some key features such as "cloud stack" vs. "on-premise stack" or call the old stack the "Hadoop stack" (due to its reliance around technologies like Hadoop and MapReduce). Illucidating these concepts is best done through juxtaposition

## The Modern Data Stack vs. the "Legacy" Stack

| Characteristic | Legacy Data Stack | Modern Data Stack |
|---|---|---|
| Architecture | Monolithic | Modular / Micro-services |
| Servers | On-premise | Cloud-based |
| Maintenance | complex - many resources required | Simplified, managed solutions |
| Languages | Java / Scala / Python | SQL-first |
| Data ingestion method | ETL | ELT |

### Democratisation / SQL-first / lowering technical barriers

In the new ELT paradigm, Data Warehouses now perform the bulk of data transformation directly, as their computational power as well as their abilities (think "loading Parquet files directly from a data lake") have increased. Ingested data undergoes little initial transformation and is straightly loaded into some raw data layer in a data warehouse or data lake.

This also means that the bulk of data transformation can now be done **through the Data Warehouse itself**, meaning that raw data can be transformed using SQL, which is a very widespread language that also less technical people know. It truly is a *lingua franca* among Data Analysts, Data Scientists, and Data Engineers alike. This marks a shift away from the classic ETL procedure where the bulk of transformations happened in more complex languages like Java, Scala, or Python. Of course, some use cases still see these languages applied heavily, but more for orchestration or data-cleaning purposes and less for data transformation, or business logic purposes.

Tools like **dbt** are often seen as a good example for this shift: Analytics Engineers can focus on business logic and data modelling because most of their work is done in SQL, which is quite hard to fumble compared to other languages.

## Cloud-native systems for infrastructure efficiency(?)

Many organisations nowadays have been set up in in or have migrated partially or fully to the cloud. That means that modern data tools have to be compatible with several cloud environments, also to keep the flexibility to maybe switch to a different provider in case their pricing model or services suit the company better in the future.

That also applies to cloud data warehouses. In the "legacy" data stack, connecting new applications to the on-premise database was much harder than in the Modern Data Stack. Often, cloud data warehouses like Databricks or Snowflake also integrate quite seamlessly with downstream applications in Python for e.g. machine learning.

## Managed and modular solutions to increase flexibility and abstract away complexity

### Managed solutions because the cloud

Moving to the cloud also often entails using cloud-native solutions like AWS Kinesis instead of self-managed Apache Kafka for event stream handling. This brings us to the aspect of managed solutions. In the past, many Data Engineers have spent a considerable amount of time managing, administrating and operating large Hadoop clusters or other complicated pieces of technology. The modern view of the Data Engineer role has shifted more in the direction of business value creation, communication, and building data tools and platforms. In other words: Spending less time on tweaking and troubleshooting technology and more time solving actual end user problems and creating business value with the data. Too much time was spent getting the tools and their background infrastructure running correctly and too little time was spent being useful for the company. Money was wasted and trust in data was eroded. Managed solutions to the rescue(?)!

> **Manage Solutions**
>
> They are "managed" in the sense that someone owns this service and is responsible for it running smoothly and for you to be able to easily set it up and use it.
> Sometimes, these are newly created services like AWS Kinesis but sometimes these are just proprietary wrappers around open-source technology (e.g. AWS Kafka Managed Solution).

Some tools exist on a continuum, their core functionality is free and open-source if you self-host it (like dbt for example), but you can also pay money to the company that owns the technology to host and manage the deployment for you, often featuring a graphical web interface for it and advanced support.

## Managed solutions because maturity

As the Modern Data Stack has established itself and several standard tasks were identified, paid or open-source managed solutions emerged that tried automating and abstracting these routine tasks.

An example is managed ingestion. Many commonplace data sources like Facebook Business Manager or Google Analytics have a well-known API and you surely will not be the first to try fetching data from it. Naturally, there exist standardised connectors for them so you don't have to re-invent the wheel. These range from transfer protocols like Singer taps to full-blown SaaS tools dedicated to connecting to all your data sources through a web interface and operate their APIs for you, dumping the data in the landing zone/raw layer of your data warehouse, ready for ELT (like Airbyte, Stitch, or more specialised tools like Adverity).

### Modularity as a zookeeper

As your zoo of different tools, solutions, frameworks, and languages grows, you want to make sure they co-exist peacefully without friction and quarrel. Two tools should not interfere with each other and cause problems when they have nothing to do with each other. In legacy data stacks, tools were often tightly coupled to each other and architecture was more monolithic, which meant that changing the system was risky and often was avoided if not enough resources to monitor, revert, or adapt the changes could be brought up.

As a solution, you want small encapsulated tools that abstract away all their complexity behind interfaces, APIs or protocols so they only expose what's truly needed by the outside world and keep their magical secrets for themselves as much as possible. On the other hand, some magical secrets like credentials, API keys, or other shared information still need to be available for all tools from a central point of truth.

### Modularity as code

Increasingly, setting up and maintaining a data stack becomes enshrined in code through tools like Terraform, AWS Cloud Formation or the like. You simply write down which tools, services, or IAM permissions in which configurations you need. Then, managed infrastructure services implement your plan. Your infrastructure turns effectively (or ontologically) into code (Infrastructure as Code).

## Pros and Cons of the Modern Data Stack

- 
    - **Speed of delivery**: Setting up a modern data stack requires less time than before and it thus can create business value faster. This is reinforced by the Infrastructure as Code practice --> changing, starting, or deleting your system is swift and iterating over changes becomes more feasible. You can simply try out what works best.

- - **Cost of maintenance**: With more managed or out-of-the-box tools and tools more geared towards scaling, cost of maintenance are reduced. In some cases, total cost of ownership also is reduced but that depends heavily on the volume of data and the pricing model of the given cloud provider.
- - **Democratisation**: As technical barriers are lowered with more low-code or no-code tools alongside greater reliance on SQL-only tools, less technical and more business-sided roles can be included into data teams. This sometimes makes it easier to hire qualified talent. Also, when a company can truly establish self-serving analytics, business stakeholders can compile their reports by themselves without relying on Data Analysts too much.
- - **Tool proliferation**: As established as the paradigm of the Modern Data Stack might be, many tools in many areas are still not guaranteed to stand the test of time. Thus, becoming aware of all available tools is hard, and an organisation could end up switching tools quite frequently. Additionally, most MDS companies are still startups backed by VC money and the field is far from settled, yet.
- - **Siloed information**: The downside of democratisation can be silos: If everybody does analytics in their own way, different definitions of metrics or other analytics illnesses of siloed organisations can arise.
- - **Quick-win addition**: If your team CAN quickly deliver results, the temptation to do so frequently or even exclusively is big. Iterating fast should not be a substitute for best practices and refactoring your quick-and-dirty code. The MDS is not meant as a substitute for proper software engineering, data modeling, or architecture principles.