# A data platform - *“future proof”*

 microsoft

## Introduction

The IT world is undergoing significant changes in these times (2024) mainly because of the “ChatGPT’s” that are offering a lot of new features around text, sound, images and even videos under the domain of Generative AI - or simply GenAI.

Therefore, it is more crucial than ever to ensure that the right data with the right quality is fed into such services to maximize the benefits of these features. But how could you predict a year ago what we can do today and what big change it is and hence plan for being able to support this. The short answer is - you couldn’t.

And what new “possibilities/challenges” will tomorrow bring. Well, we really don’t know, the only thing we know is that there will be changes and they are likely to happen sooner rather than later.

This document explains how you could create a data platform that will be able to handle “whatever might come” and hence give you the ability to at least from a data perspective navigate these opportunities/challenges.

The content of this document is not a “this is best practice”, but a “you ought to consider this” when establishing your data platform.

[!NOTE] This document will represent an ever lasting *work in progress*, as we learn new stuff all the time. The next topics will most likely be Governance/Compliance and GenAI, as we learn more about this in connection with the different implementations we have of dataplatforms in use.

## Background

This document is based on the experiences of various Microsoft customers who wanted to streamline the creation of a data platform utilizing cloud services for analytical tasks.

This concept of a data platform could most likely be applied on any data platform setup, but that is (currently) not in scope.

[!NOTE] The document is mainly based on Microsoft technology components.

## The Vision

The vision of the data platform described in this document is to a platform where Governance and Compliance are main drivers. Hence the concept of a self consistent dataset is a key element of this approach.

Having said that, the described data platform can be used in any scenario and be adopted to the actual use case.

## Overall paradigm

To ensure that the data platform can handle “whatever might come” it is important to establish a set of guidelines. Hence the overall paradigm is to have a data platform fulfilling the following characteristics.

1. **Consistency** - The solution design is consistent to make it easy to operate and develop solutions further. The consistency level of any data object is always clear. This is to make sure that the value of the data can always be conveyed.
2. **Data Encapsulation** - The data in the data infrastructure can only be accessed through an interface that controls who can access what and when. The interface must let you change the data infrastructure without affecting external systems.
3. **Modularity** - The solutions in the data platform must be built with modularity in mind and with a clear interface, so it is easy to replace, add or remove resources and services.
4. **Technology Independent** - Architecture does not depend on the technology being used. This means that the processes, functionalities, and layers remain the same no matter what technology is used.
5. **Scalability** - Scalability (horizontal/vertical) is part of the solution design from the start, so that the implementation and operations are not impacted by bottlenecks, downtime, or unexpected license purchases.
6. **Restartability** - All services used in the solution should be able to be paused/stopped and even deleted. And be able to be started/recreated. And this without any data loss or change in functionality.
7. **Accountability** - All services used in the solution must be individually trackable for usage - both for security and cost purposes. Agility - The focus is on minimum viable product (MVP) and ongoing feedback to previous steps in the data flow.
8. **Security** - Security must be integrated into the general architecture and the specific solution designs, both for information security and privacy. A change in components must not affect any security aspects. Compliance and governance must be maintained across the different layers over time.
9. **Reuse** - Solutions should be designed for reuse. The architecture should include templates for solution designs that speed up time-to-market and ensure standardization.
10. **Feedback** - The architecture should be continuously adjusted and improved based on feedback from the usage of the data platform.

## *Data* and *Datasets*

The concepts of *data* and *datasets* are **core components** of the data platform.

A very important aspect of especially *data* but also to some degree *datasets* is that a given object must be able to be handled, maintained and secured by itself, that is not through any technology component like a database, hence **file** handling is the overall aspect of this.

When the term *data* is used, it refers to a single data object like a file or table that contains only the *data* for that given object. For example, a source table like customers or invoices.

On the other hand, when the term *dataset* is used, it implies a collection of tables or files that are connected to each other. This could be a data mart setup with customer, product and time dimensions and a sales fact, also known as a star schema.

The *dataset* is the key component of the data platform, and it has some specific characteristics:

* The *dataset* is self-sufficient, meaning it does not depend on any other *data* sources. It contains all the information needed.
* Only the columns/rows that are relevant for the task being supported are included.
* A *dataset* belongs to a group and not to specific individuals - in an Azure setup, the owner would be a group with an Entra ID.
* A *dataset* should have two more groups associated with it, one for creating the content and another for read-only access.
* Access to a *dataset* is granted by adding individuals to the group(s) depending on the task they need to perform.
* A *dataset* is not tied to a specific technology like a relational database. It is stored as files, often csv or parquet based. It can then be provided with the appropriate technology for the task in question – like a relational database.

## Cloud aspects

This document explains how to create a data platform using cloud technology. And this approach to a data platform has some features that are only achievable with a cloud approach.

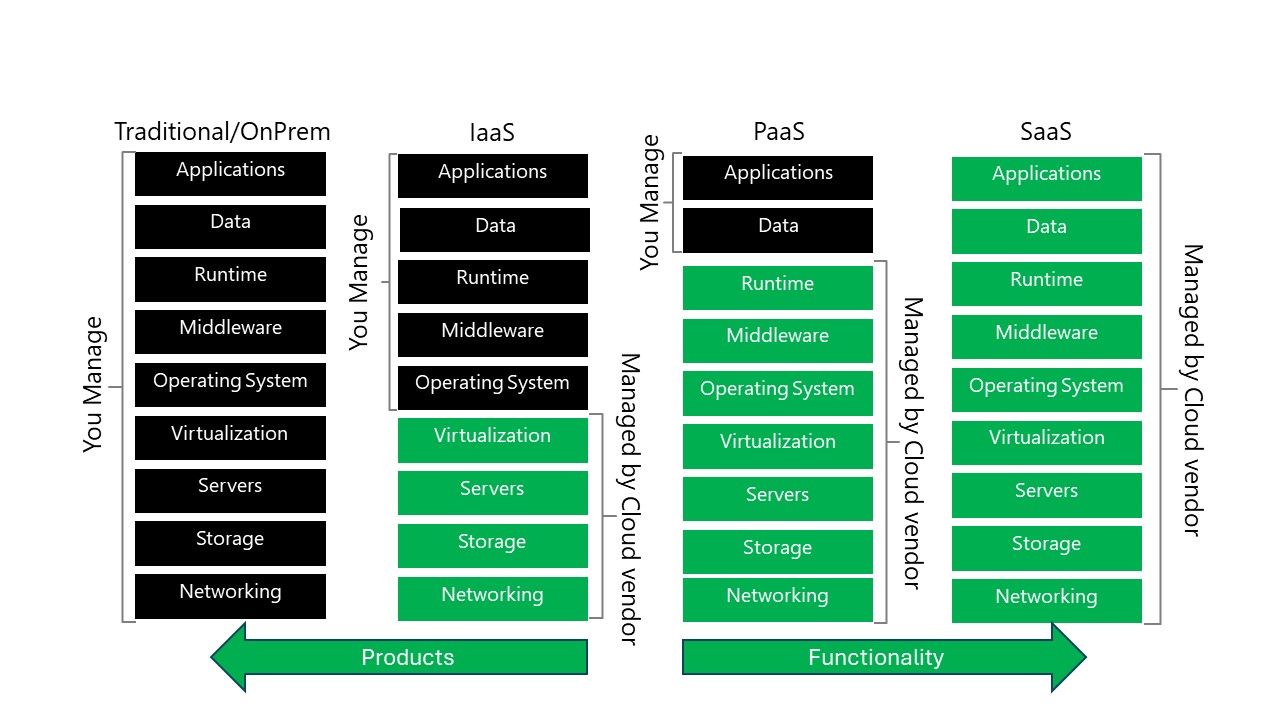


Figure 1

*Figure 1*

As shown on Figure 1 using cloud technologies, gives access to different types of services.

**Infrastructure as a Service (IaaS)** – this provides the ability to create various types of virtual machines and install any software needed on them. In this, the cloud vendor guarantees the service up to the level of the operating system. In the IaaS setup the focus is on which products should be used and hence providing the right “machines” for this.

**Platform as a Service (PaaS)** – this is for services like databases. We don’t have to care about the infrastructure behind these services. The cloud vendor ensures all the required components behind the service itself. This includes updates, new versions and availability. In the PaaS setup the focus is only on what functionality is needed and not on which “product” could do this.

**Software as a Service (SaaS)** – is a “complete” solution like an ERP or a HR system. Here the cloud vendor assures access to a full package of applications, databases, and infrastructure needed for the SaaS service to work.

The data platform discussed is based on PaaS and/or SaaS services. In the chapter “Implementations”, different ways of doing this is discussed with the usage of different PaaS or SaaS services.

A key aspect of the cloud is that “everything is software”. This means that when the need is - let’s say - for a new Linux server, the various components that the server uses - like disks, network cards etc. - these components are generated by sending commands to the cloud infrastructure. So, we rely on software to “produce” these components.

This is known as infrastructure as code (IaC). In the data platform, this is used to create a relational database and load a *dataset* into this database using code.

## Logical architecture

The data platform arranges *data* and *datasets* in different areas according to the logical architecture shown on Figure 2.

This is to ensure that the approach can comply with the “rules” of the paradigm as mentioned previously.

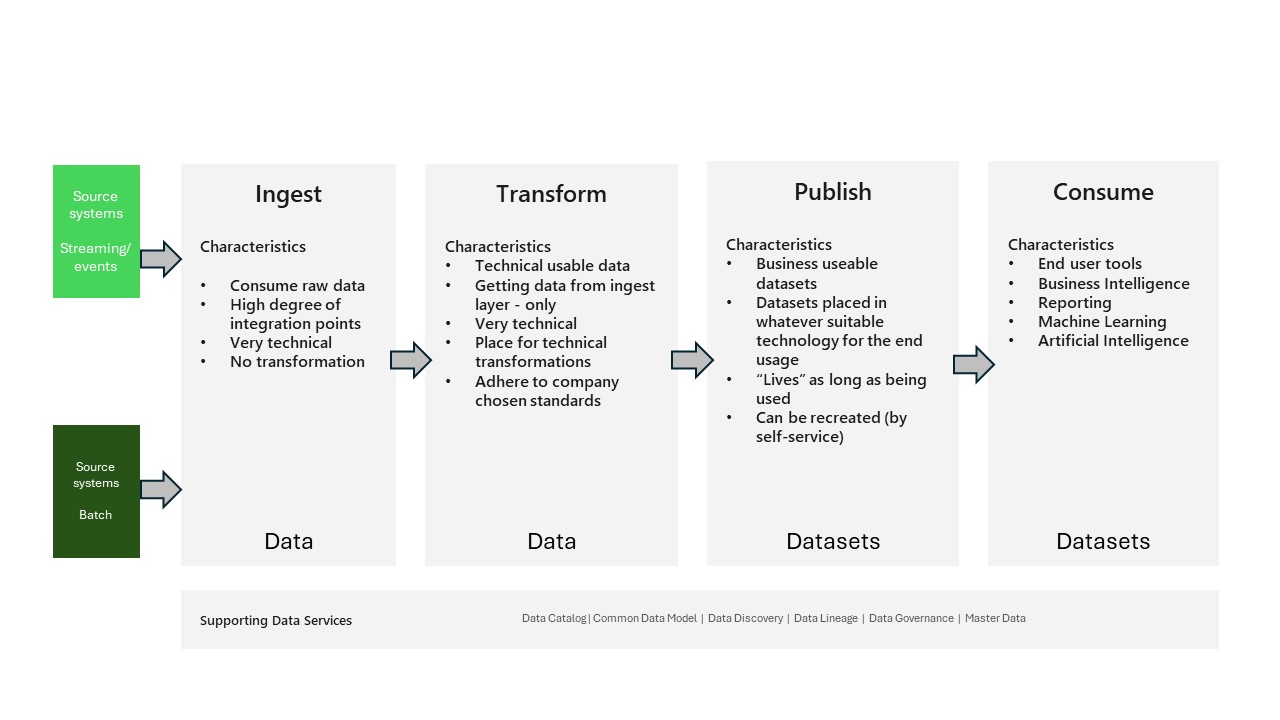


Figure 2

*Figure 2*

The areas represent different states of the journey from *data* to *datasets* and thereby to reporting and analytics.

Referring to Figure 2 the characteristics of the different areas are as follows:

**Source systems** are any system where *data* must be extracted from (batch) or where *data* is sent from (streamed).

The **ingest area** is where *data* from the source systems arrives. *Data* is kept with its original content. If the *data* is “table” data, no changes are made to any rows or columns, not even the format of the data. *Data* is stored in files - usually csv or parquet or in their binary format - like for videos, pictures, or sound. The **ingest area** should have a directory structure that makes it easy to identify the source of the data.

Also, there is no updating or overwriting of existing files - meaning that a new load creates new files. Over time, files in the **ingest area** should be archived or deleted if legal requirements demand this (such as GDPR).

*Data* is technically modified to comply with usable standards in the **transform area**. The *data* objects in this area are independent. This means that they do not have a mix of *data* from the source system, are not changed to be a golden record and do not restrict the number of rows or columns. They are “clean” objects that are convenient to work with when used for creating *datasets*.

From a technical perspective, the main task for transforming *data* is to make sure they have the same format for some of the “difficult” *data* types, such as dates (e.g. splitting time from the date into a separate column) and decimal number (“.,” or “,.” as separators). The second main task is to ensure that the different *data* we want to have available in this area are easily joinable, meaning that the individual *data* object has the “reference keys” needed to be able to join with other *data* objects.

In the **publish area**, we create the *datasets* that are needed to meet the various business needs that require *data* from the data platform. We will apply techniques like star schemas and deliver these *datasets* in what are commonly called *data* marts.

The **consume area** is where the end users of the data platform will access the *datasets* from the **publish area** using the tools they find most suitable.

[!NOTE] The paradigm described above is today also known as the medallion data architecture, where bronze is the **ingest area**, silver is **transform** and gold is **publish**. The medallion architecture does not reflect on the **consume area**. This document will use the terms **ingest**, **transform**, **publish**, and **consume** as this reflects what has been used at the customers that is the inspiration for this document. And it also reflects better the purpose of the different areas in the context of this document.

### Interfaces

Figure 2 indicates that the interface between the different areas is as important as the content of the areas.

These interfaces must ensure the technology independency we want in the platform – it must be easy to change/add new services – as well as ensuring that we know the data pathways.

Most of the customers we work with prefer these three interface options.

**REST API** – the “basic” interface that most (all) new services use to offer their functionalities. This level is highly technical and is not suitable for low/no-code.

**SQL** – the standard data query language that is widely used and supported by different data storage systems. Many people are familiar with SQL from its use in relational databases, but it is also available in other database systems.

If you “only” use the SQL standard language functionalities, this will give you a high degree of flexibility. This means that you will not use any specific capability that the vendor of the underlying technology has implemented – especially the use of the procedural languages provided in i.e. MS SQL Server or Oracle DB.

**Python** – the “new” data processing language. Python is widely used and reflects the current way of handling data. Python is a high-level programming language that can be interpreted. It has built-in data structures that are high-level, and it allows dynamic typing and binding, which make it ideal for Rapid Application Development, as well as for using it as a scripting or glue language to link existing components together.

Python’s syntax is simple, easy to learn and emphasizes readability, which lowers the cost of program maintenance. Python supports modules and packages, which promotes program modularity and code reuse. The Python interpreter and the extensive standard library are free to obtain and distribute in source or binary form for all major platforms. Many programmers love Python because it makes them more productive.

### Supporting Data Services

One more component of the implementation that Figure 2 displays are some Supporting data Services.

These include:

[Data Catalog](Supporting_Data_Services/Data_Catalog/DataCatalog.md) – a “directory database” where we record and explain which *data* and *datasets* are available in the data platform. This would also cover elements like ownership, usage guidelines etc.

[Common Data Model](Supporting_Data_Services/Common_Data_Model/CommonDataModel.md) – Especially for the **publish area**, existing industry *data* models could be useful/relevant.

[Data Discovery](Supporting_Data_Services/Data_Discovery/DataDiscovery.md) – A data science process oriented to business users that allows them to visually explore data and apply advanced analytics to find patterns, gain insight and answer specific business questions.

[Data Lineage](Supporting_Data_Services/Data_Lineage/DataLineage.md) – Documentation on how data is being “transferred” and transformed between the different areas. It can be both a technical lineage – “what actions were performed on the data” - as well as a business lineage - “where do these data originate from?”.

[Data Governance](Supporting_Data_Services/Data_Governance/DataGovernance.md) – Processes, policies, rules and reports to ensure that the data is governed properly. Often based on frameworks like ISO27x or NIST.

[Master Data](Supporting_Data_Services/Master_Data/MasterData.md) – Some *data*/*datasets* might benefit from being treated as master data, ensuring that there is only “one record” of i.e. a customer.

[!NOTE] The document will be enhanced over time to include further description of these services.

## A special case – Real Time

We will approach Real Time data processing as a particular case and examine it from that perspective.

Real Time comes in different flavours, in this document we will use the following terms

* Real Time – *data* that is delivered immediately after collection.
* Near Real Time – *data* that is “delayed” due to communication or processing.
* Dynamic data – *data* that is updated and needs attention.

The data platform method in this document is **not** suitable for the Real Time situation, but it works well for Near Real Time and Dynamic Data. Dynamic data – which is based on events – is processed in the same manner as Near Real Time data in the data platform.

The general approach is that any or all real time *data* is also kept in the **ingest area** for more processing. This enables the data platform to have features that can support a Real Time process where appropriate, but also to manage all knowledge that can be generated over time from the Real Time situations. This can then help to avoid an undesired event from occurring - such as a train delay.

Or give more detailed information about a goal event in a football match and then being able to change odds rapidly based on the historical data available in the data platform.

To illustrate this, imagine a train that is delayed. The system that displays the information to the passengers at the station will receive the Real Time data about the delay right away and update the signs accordingly. In this process, the data itself is not much processed.

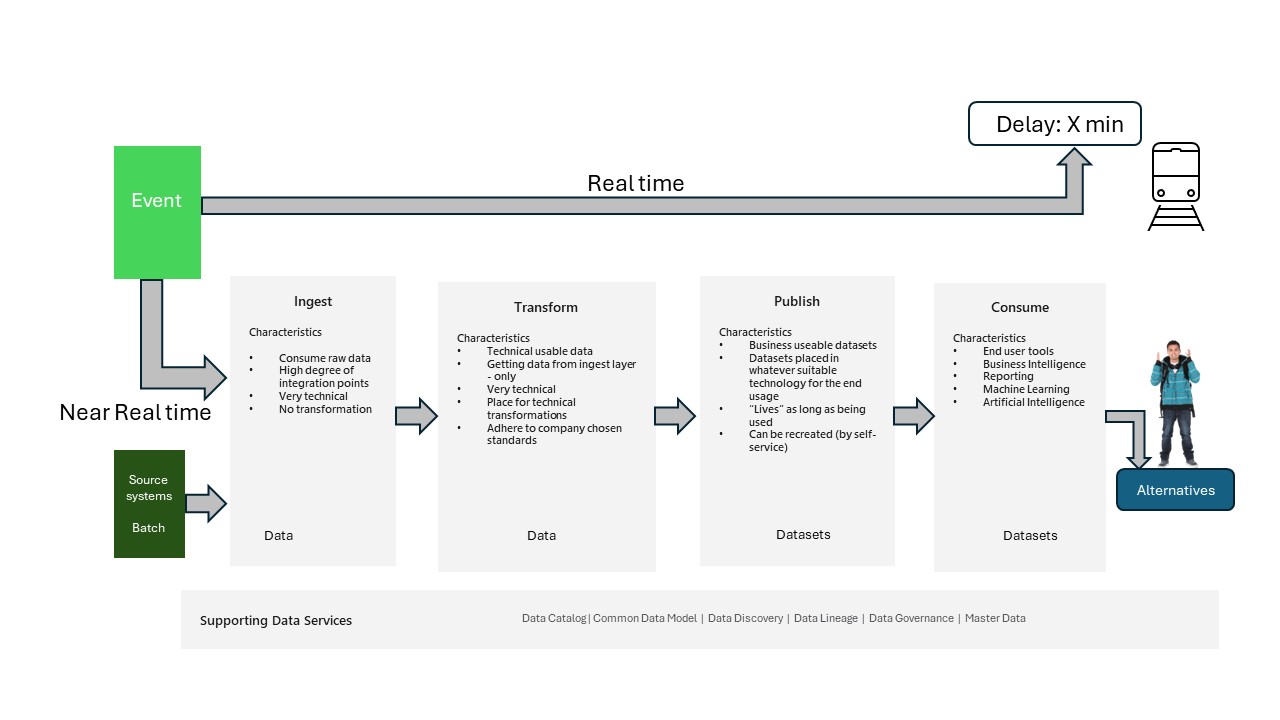


figure 3

*Figure 3*

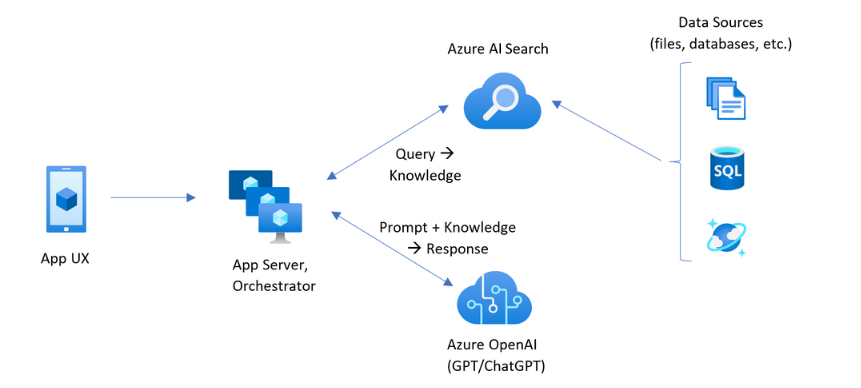
However, this *data* is **also** stored in the data platform, where there is some time, but not much, to generate a suggestion for alternative routes for the individual passengers who are waiting for the delayed train. This information can then be sent to an app on their smartphones.

## Another special case - ChatGPT/CoPilot

As AI/ML/GenAI is becoming more and more accessible the requirements and hence challenges on the data elements being used in such solutions gets even more important.

A typical approach for bringing your own data in scope of especially a GenAI solution is using a method known as RAG, standing for Retrieval Augmented Generation.

RAG is an architecture that augments the capabilities of a Large Language Model (LLM) like ChatGPT by adding an information retrieval system that provides **grounding** data. Adding an information retrieval system gives you control over grounding data used by an LLM when it formulates a response. For an enterprise solution, RAG architecture means that you can constrain generative AI to your enterprise content sourced from **vectorized** documents and images, and other data formats if you have embedding models for that content.

 *Microsoft RAG Architecture*

The decision about which information retrieval system to use is critical because it determines the inputs to the LLM. The information retrieval system should provide:

1. Indexing strategies that load and refresh at scale, for all of your content, at the frequency you require.
2. Query capabilities and relevance tuning. The system should return relevant results, in the short-form formats necessary for meeting the token length requirements of LLM inputs.
3. Security, global reach, and reliability for both data and operations.
4. Integration with embedding models for indexing, and chat models or language understanding models for retrieval.

Azure AI Search is an example of such a “database” which provides indexing and query capabilities, with the infrastructure and security of the Azure cloud.

Through code and other components, you can design a comprehensive RAG solution that includes all of the elements for generative AI over your proprietary content.

Based on the above we must ensure that the data we provide to the solutions created fits exactly what the user has access to, so that the grounding and hence the vector database contains only this. This can be done through the publish layer as this layer represents datasets with the exact content. So the task is to “load” the vector databases being used - and make sure these are stopped/removed after usage.

## Roles

We will use roles as a mechanism of controlling access to the *data* and *datasets* in the data platform as well as assigning the right responsibilities to persons having these roles.

Of course, the roles must be delegated to people. How the roles are connected to people, and if certain people have more than one role, doesn’t really matter. The most important thing is that the processes comply with the “rules” of the data platform paradigm.

This is a list of common roles seen with our clients:

1. Project owner
   * Interface to a steering group/management.
   * Most like likely coming from “Business”
2. Project Leader
   * Leads the specific project
   * Most like likely coming from “IT”
3. Data Engineer
   * Creates the integration between a source system and the **ingest area**.
   * Most likely coming from “IT”
4. Designer
   * Designs (and creates) the different *dataset* to be used by the business
   * Most likely coming from “Business”
5. Transform engineer
   * Creates the transformation processes used in the **transform** and **publish area**
   * Most likely coming from “IT”
6. Data Governance
   * Ensures that the understanding between IT and Business are aligned and that governance principles are followed
   * Most likely coming from “IT/Business”
7. Superuser/Ambassador
   * Defines (and creates) reports/dashboard and other usage patterns
   * Most likely coming from “Business”
8. System owner
   * Provides understanding about how a source system is being used
   * Most likely coming from “Business”
9. Architect
   * Ensures that the architectural principles are being followed
   * Most likely coming from “IT”

Depending on the size of project, size of company and/or framework being used you could also meet roles like Scrum Master, Product Owner, Program Owner, Steering Committee member etc.

The needs of the different roles will naturally differ through a project. The following is an example of what the “workload” could look like:

| Role | Ideation | Iteration 1 | Iteration 2 | Iteration 3 | Iteration 4 | Iteration …. | Vision state |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Project Owner | 100 % | 25% | 25% | 25% | 25% | 25% | 100% |
| Project Leader | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| Data Engineer | 100% | 50% | 25% | 5% | 5% | 5% | 0% |
| Designer | 100% | 10% | 10% | 25% | 25% | 40% | 0% |
| Transform Engineer | 100% | 30% | 30% | 30% | 30% | 30% | 0% |
| Data Governance | 100% | 25% | 25% | 10% | 10% | 10% | 10% |
| Superuser/Ambassador | 100% | 5% | 5% | 20% | 20% | 50% | 100% |
| System owner | 100% | 25% | 10% | 0% | 0% | 0% | 10% |
| Architect | 100% | 50% | 25% | 10% | 10% | 10% | 10% |

1. Ideation – Project start including ideas to be listed (backlogged)
2. Iteration X – the individual “versions” – most likely in 3-month timespans
3. Vision State – overall what is the projects going to accomplish

## Environments

In this section, we will explore how to use the six environments for implementing a data platform project. Depending on your preferences, you can of course decide how many of these environments you want to have and how you want to distribute the tasks that we will describe among fewer or more environments.

1. Sandbox – this environment is used for MVP testing.
2. Project room - development environments.
3. Development - code repository.
4. Testing - functional testing.
5. Quality Assurance - code review.
6. Pre-production - test on production data.
7. Production - production data.

## Data Policies

To control the *data* policies, we need to understand which environment we are working in. This document uses 5 environment types in the discussing of the processes.

1. Sandbox – environment used to test functionality in each service. These environments do NOT contain any business/company data.
2. Project rooms - These environments are used for establishing environments containing tools and data used to fulfill a development task.
3. Non-Production – environments that contain development, quality assurance and test scenarios.
4. Production – environments supporting pre-production and production scenarios.
5. Confidential – environments supporting data processing of highly confidential data.

As mentioned you can find projects handling the establishment of such environments in a secure setup - see “here” for more information.

## Data and security

Data security is a crucial aspect of any organization’s operations. It involves protecting sensitive information from unauthorized access, corruption, or theft throughout its entire lifecycle. By implementing strong *data* security measures, organizations can help protect their valuable assets, meet relevant compliance requirements, and maintain customer trust in the usage of data.

Data security is important because it helps organizations guard against cyber attacks, insider threats, and human error, all of which can lead to data breaches. The four key issues in data security are confidentiality, integrity, availability, and compliance. As data is increasingly the target of attackers, organizations need security at the point of *data* to keep data safe and recover it faster. The goal of data security is to make your data resilient against any kind of misuse that being attacks, errors etc.

In summary, data security is essential for maintaining the confidentiality, integrity, and availability of an organization’s data. It helps protect valuable assets, meet compliance requirements, and maintain customer trust.

In the section [Security](Security/Security.md) you will find a much more detailed discussion on the security aspect. This section will continue with some more technical possibilities.

In Figure 4 you will find several ways to protect data which are available in Azure. This does not address the more general things like network security, Multi Factor Authentication etc. which is assumed to be in place.

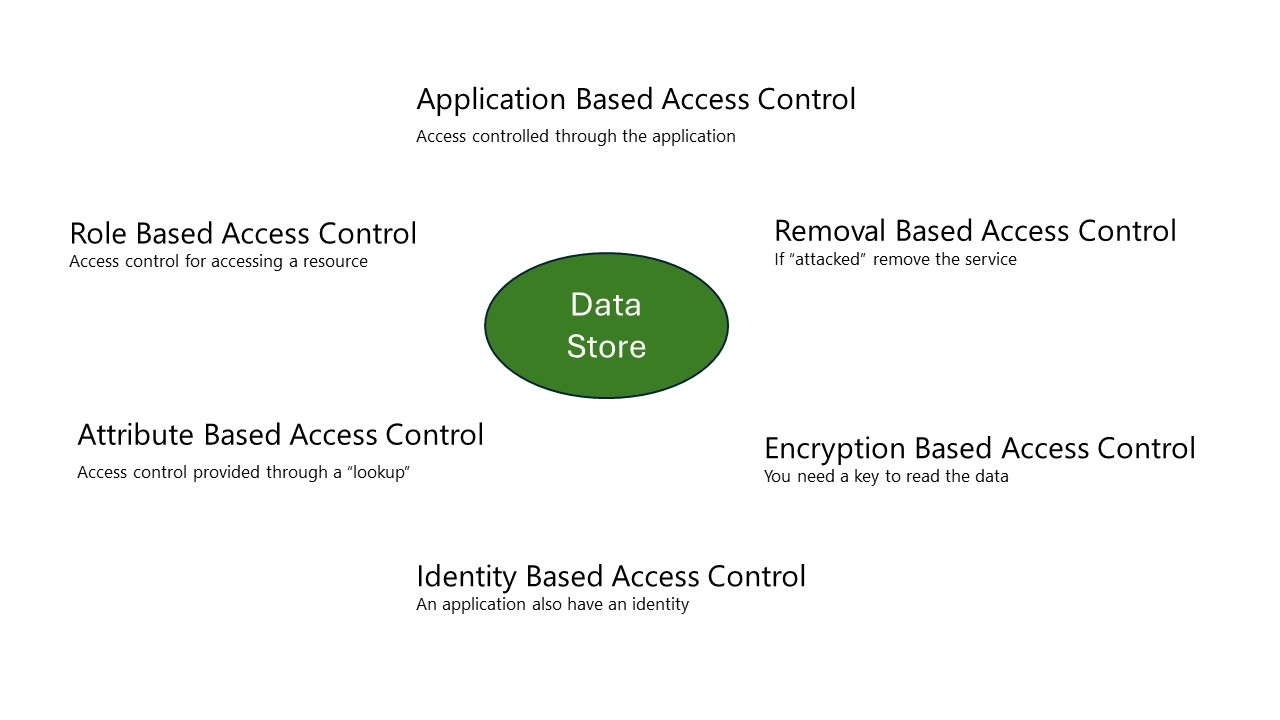


figure 4

*Figure 4*

**Application Based Access Control** – covers that an Application like SAP, Snowflake, Fabric, Dynamics etc. requires a login and hence grants the correct access to the underlying data used in the application. Often the underlying data store is a (relational) database, which is accessed from the application using a service account.

**Role Based Access Control** – also known as RBAC. This controls the access to a given resource and how it can be used. So popular said – can one get to the storage account?

**Attribute Based Access Control** – also known as ABAC - provides an extra mechanism for granting access making a “lookup” in another system. For example, you might be able to get to a storage account but their might a directory that requires that you are part of a given project. In this case you can make an ABAC “lookup” that checks this before providing access – depending on this result.

**Identity Based Access Control** – covers the capability that a given resource can be assigned an identity (becoming “a human”). And then you make sure that access to a given storage account is only provided to this “human” and hence you need to use this application to get to the data.

**Encryption Based Access Control** – this not really Access Control, because the *data* storage will be accessible, but I can only read/use the *data* if I have the Key for decryption. So, it can(should) be part of your defense.

**Removal Based Access Control** – this approach is only feasible in the **publish area**. This approach uses the aspect of the **publish area** that a dataset store only “lives as long as being used”, in this case “…being used correctly”. Because this approach requires the ability to be able to recreate a given *dataset* store, this can also be used as a defense mechanism. So, if an attack is realized the easiest way to stop this is to simply remove the resource under attack if there is a risk of *dataset* loss.

## Data Operations

According to Wikipedia, DataOps is a collection of practices, processes and technologies that combines a holistic and process-oriented view of data with automation and methods from agile software engineering to enhance quality, speed, and collaboration and foster a culture of continuous improvement around data analytics.

While DataOps started as a set of best practices, it has now evolved to become a new and distinct approach to data analytics. DataOps covers the whole data lifecycle from data preparation to reporting and acknowledges the interdependent nature of the data analytics team and information technology operations.

In software development, DevOps emphasizes continuous delivery by using on-demand IT resources and by automating test and deployment of software. This way of software development and IT operations has improved velocity, quality, predictability and scale of software engineering and deployment.

Taking methods from DevOps, DataOps aims to bring these same improvements to data analytics. DataOps should not be linked to a specific technology, architecture, tool, language, or framework.

### Environments and encryption policies

In different environments certain politics regarding encryption could be enforced. The following table is an example of what this could look like.

| Environment/Policy | Sandbox | Project room | Non-Production | Production | Confidential |
| --- | --- | --- | --- | --- | --- |
| Encryption at Rest | Audit | Audit | Audit | Required | Required |
| Encryption in Transit | Audit | Audit | Audit | Required | Required |
| Encryption in Processing | N/A | N/A | N/A | N/A | Required |

* Audit – it is required that a policy audit if encryption is in place, but does not require it. This can be used for risk-assessment.
* Required – the policy will prevent creating data storage of any kind without encryption.
* N/A – Not applicable.

### Environments and Tags

In the different environments different tags should be used to identify the nature of environment. The following table are examples of tags that could be attached to the different environments.

| Environment/Tag | Sandbox | Project room | Non-Production | Production | Confidential | Values |
| --- | --- | --- | --- | --- | --- | --- |
| DataOwner | Audit | Required | Required | Required | Required | name of owner |
| Environment | Required | Required | Required | Required | Required | The type of environment like “Sandbox” |
| CostCenter | Required | Required | Required | Required | Required | Cost Center |

* Audit – tag should be in place.
* Required – tag must be present, if not the deployment will be denied.
* N/A – Not applicable.

### Development environment - Project room

A way of establishing a secure development environment could be to use what is called a Project room in this document. These project rooms represent an isolated environment normally owned by a unique AD-Group (one or more).

In the project room data, tools and code are established/maintained completely isolated. Access to a project room is done by adding or revoking people from the corresponding AD-Groups.

The following figure shows an example of a project room in the data platform environment.

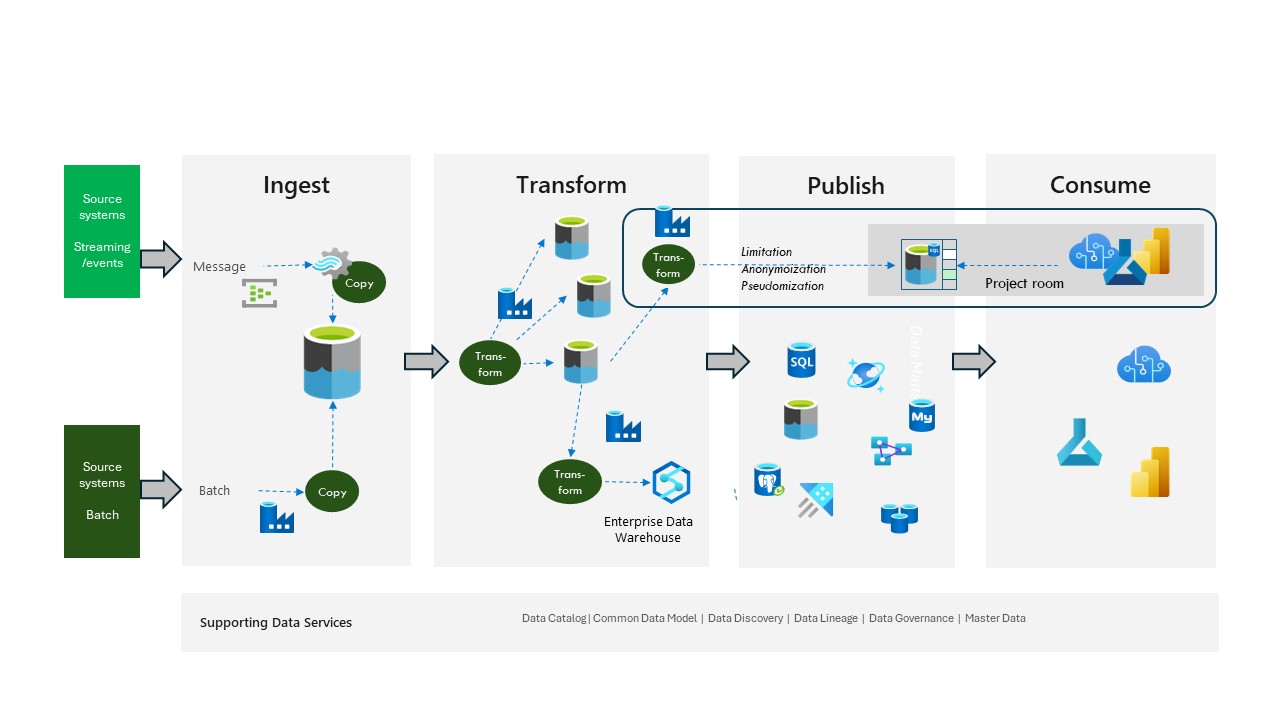


figure 5

*Figure 5*

Development being done in a project room can then be “checked in” to the overall data platform using i.e. a CI/CD process. An example of this is shown later in this document. Any data needed for doing the development could/should undergo a process that makes it a “non-production” *data*/*dataset*.

In case *data*/*datasets* in these project rooms need to be read-only, the ownership should be assigned to a second yet still unique AD-group.

In the rare situations where an integration connection between project rooms is needed, the ownership should be set to a third AD-group, still being unique to the project-room.

## Data Establishment

Guiding principles

1. Legislation like the European GDPR or the CCPA from California must be adhered to.
2. Data can only be used in the context of which they are collected.
3. Development cannot be done on production data.
4. Distinguish between the storage of data - “security” - and the usage of data - “privacy”.

The main technical mechanisms we use in the Azure platform to ensure right handling in the environments will be the following:

1. Data classification - makes it possible to understand the nature of the data we are dealing with.
2. Environment separation - by separating the environments mixture of data can be avoided. This enables a much simpler governance process.
3. Azure politics - using politics ensures that general paradigms like ISO27XXX, NIST or the like are being followed in all environments.
4. Tagging - helps identify important elements in each environment, like owner and environment type.
5. Encryption - using encryption, maybe using own keys, can be a mechanism security wise.

### Process for creating non-production *data*

The process of creating *data* that can be used in the non-production environments can be one (or more) of the following:

* Copy of production data.
* Data generation of fake data.
* Anonymization.
* Pseudonymization.
* Encryption + Dictionary.
* “No relations” *data* .

The method or methods being used will normally depend on the classification of the different data elements. For the purpose of establishing the classification of different *data*/*datasets* a “database” should be maintained carrying this information. This should then be the “lookup” place to establish the right data processing.

### Data processing in detail

In this section we discuss the different possibilities that can be used to establish one or more *datasets* in a non-production environment. You will most likely use a combination of these.

Important to EU-notice: pseudonymous data and anonymous data are treated differently under GDPR, the European Union data protection act. Reference EU-document

#### Copy of production data

Even though this is not a recommended approach sometimes it is possible and allowed to use (a subset of) production data in a non-production environment. These data will then most likely be marked as read-only.

* An Azure service suitable for this task would be Azure Data Factory copy pipelines.

#### Data generation of fake data

In this task you will create fake *data* based on the information of what datatype, length, pattern, content etc your *data* need to adhere to. It is important that this is real fake data and not data that is made fake from a production dataset.

Creating a fake set of data is most often a difficult task, so it is worth making sure that the process being established has some of the characteristics of being re-useable, automatic, scalable and parameterized.

Tools to create fake data within a Python environment, could be: - Faker - SDV - Gretel

#### Anonymization

Anonymization is used when you can scramble a production dataset and bring it to a non-production environment. Anonymous data is data that has been changed so that re-identification of the individual is impossible.

The scrambling can be done in many ways, using different technics like noise addition, substitution, and aggregation.

It is important to notice that anonymization is a “one-way process”, where you will - and must - lose the ability to trace back to the original data. You must also not be able to use this data to connect and use *data* from other production *datasets*.

* Azure Data Factory data flows/Azure Databricks can be used for this, with the extension of Microsoft Presidio.

#### Pseudonymization

Pseudonymization is in essence the same process as Anonymization, with one big difference. Pseudonymous data is data that has been de-identified from the data’s origin but can be re-identified if needed. Tokenization and hash functions can be used to pseudonymize data.

* Azure Data Factory data flows/Azure Databricks can be used for this.

#### Encryption + Dictionary

In this case you protect your *data* with an encryption key and only the people within the Dictionary of this data storage will be able to use the key. This key could be held in an Azure Key Vault.

[!NOTE] This is not a valid GDPR “protection” mechanism because it “only” protects the access to the data, not the usage of the data.

#### “No relations” *datasets*

In this approach you create *datasets* that contain “real” data at a column level, so i.e., if you have a postal-code column then the postal-codes would be real, as well if you in the same record have a street name, that would also be real street names.

The “no relations” approach comes into play when data is being looked at from a row level perspective. In the above case the street name, house number and the postal code would logically make sense, but it will not exist physically.

So, an approach could be to build a set of different *datasets* representing items that is person related from open-source datasets - could be a database with street names in one *dataset*, all postal codes in another *dataset*, the 20 most common first names in a third *dataset*, the 20 most used last names in a fourth *datasets* and so on. And when creating a new row, random values are taken from these *datasets* and use these to create items like “persons”.

* Azure Data Factory *data* flows/Azure Databricks can be used for this.

## CI/CD example

As mentioned above, a way of making sure coding in the data platform is done the “right away” you should consider using Continuous Integration/Continuous Deployment (CI/CD) principles. Such processes have pipeline workflow that describes what processes code goes through when being deployed to production.

The following Figure 6 show a simplified workflow.

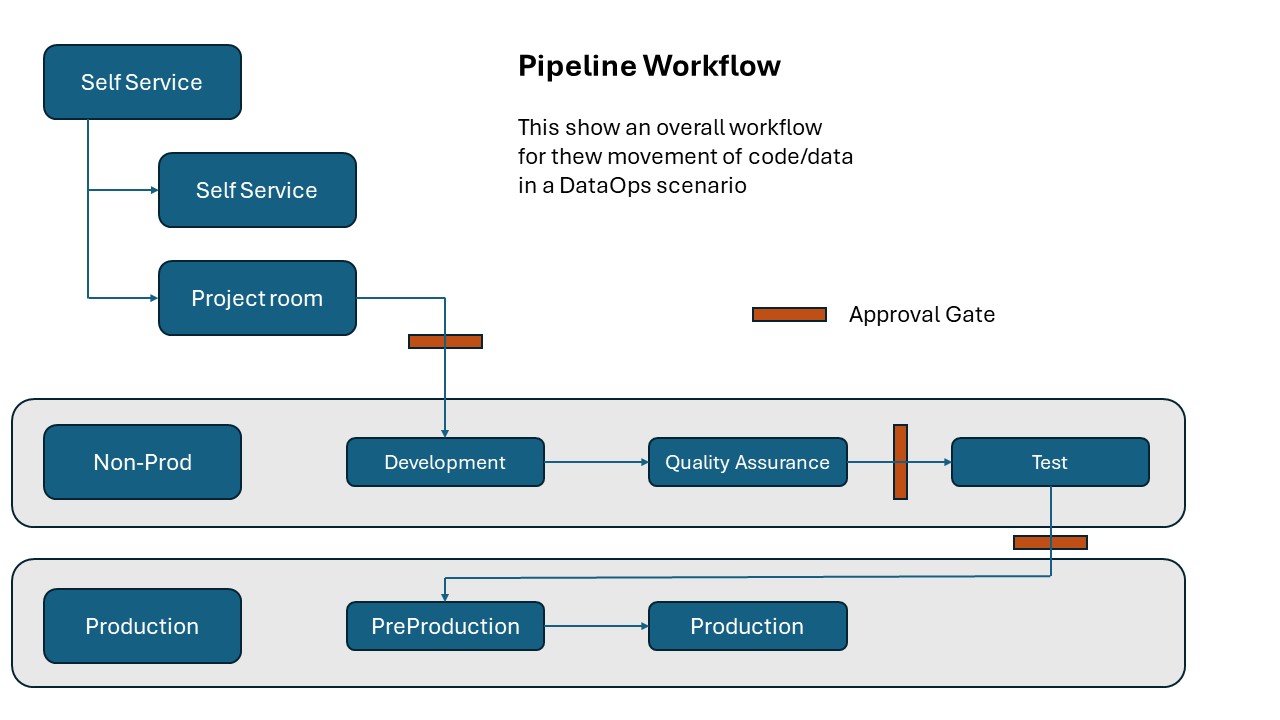


figure 6

A deeper discussion on CI/CD and the data platform can be found in this section [DataOps](DataOps/DataOps.md)

*Figure 6*

In connection with the continuous development and testing you often need to be able to handle data in non-production environments. Most likely you are not allowed to or do not want to use production data in these environments. Also, for testing purposes you may want to introduce faulty data in your datasets to be able to handle any exception handling scenarios.

## A practical approach

Based on the discussions in this document Figure 7 shows what this could look like in “real life”.

On the left in this figure, you see the source system which is owned by “someone”, usually known as the system owners. These system owners are responsible for assuring that the data platform has access to the right systems. So, on the figure we have 3 systems called App 1, App 2 and App 3 and they are each owned by a system owner here named System Owner 1 to 3.

In the middle we find the data platform with the **Ingest**, **Transform** and **publish area**. In the **ingest area** you see that data is taken one-to-one from the different App 1 to 3. Then we have a transform process that refines these raw data into their usable state.

Then on the right side of the figure we see what is required by the end-users in the **consume area**. The first user called data User 1 needs *data* that only comes from App 1, so the dataset needed called *data* Product A is a straightforward process.

The Data User 2 needs data that comes from both App 1 and 2, but data present in App 3 most be excluded from that dataset, so in this case the process is a little more complicated, but because the **transform area** represents an area where easily can combine (and also exclude) data the fundamentals for doing this is in place, hence it be done rather smoothly.

The same goes for the Data Product C which represents data from App 2 excluding data present in App 3.

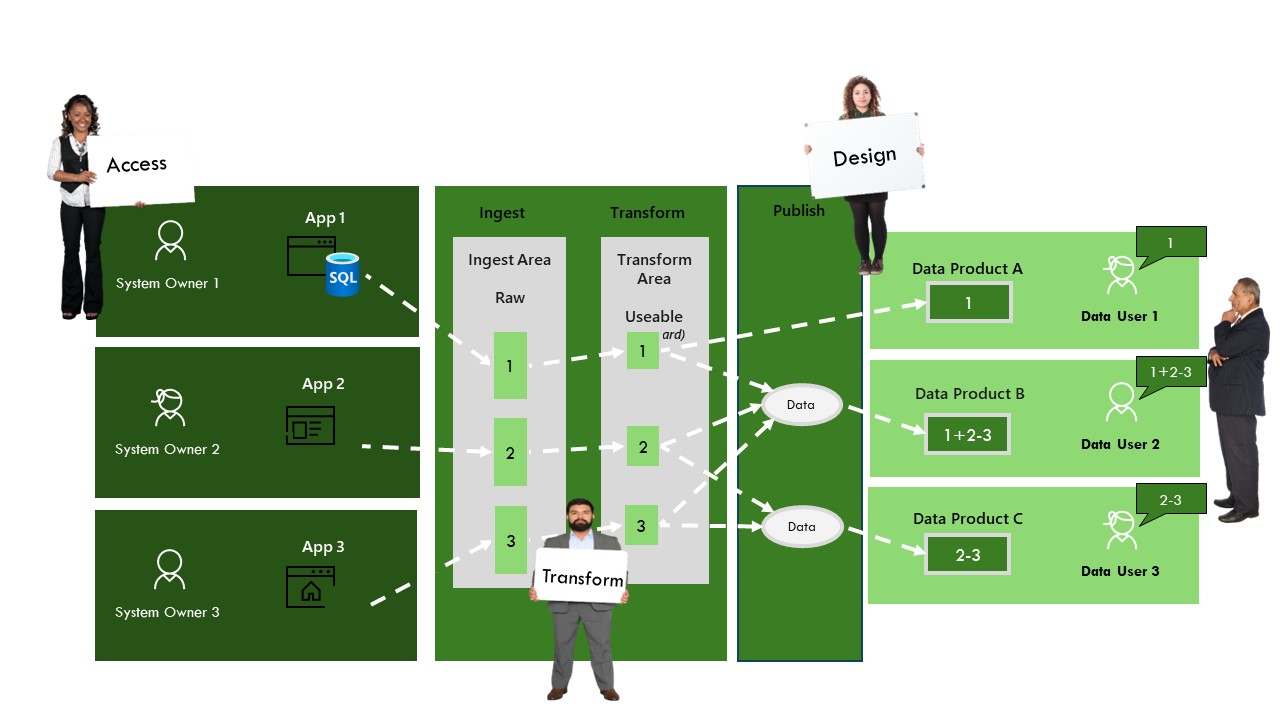


figure 7

*Figure 7*

This also represents how the data platform should be able to exactly support the business needs quickly and smoothly. So, the overall term could be – **if it is not available today, it will be tomorrow**.

## Examples of implementations

The following are some examples of ways to implement a data platform using different services. Please remember that the overall paradigm is technology independence hence you should “mix and match” what suites best in regards of your business opportunities and challenges.

[!NOTE] These folders “only” contains descriptive information for the time being but will eventually have Infrastructure-as-Code examples included.

[Azure Based](Microsoft/Azure/Azure.md) - using PaaS services from Azure

[Synapse based](Microsoft/Synapse/Synapse.md) - using Synapse PaaS service

[Fabric based](Microsoft/Fabric/Fabric.md) - using the SaaS solution Microsoft Fabric

[DataBricks based](Partners/Databricks/Databricks.md) - using the 1st party PaaS service Azure DataBricks

[Snowflake based](Partners/Snowflake/Snowflake.md) - using the SaaS service Snowflake together with Azure services

[Microsoft Purview and the data platform](Microsoft/Purview/Purview.md) - governance and compliance of the data platform using Purview