Big Data Management
Building a Big Data Architecture

Project guidelines (P2): trusted zone, exploitation zone and downstream tasks

The structure for the P2 document will be slightly different. First, we will introduce, conceptually, the remaining stages of our architecture (section 1). Then, we will describe how these layers can be implemented (section 2). This includes the overall design, database selection and the transformations between zones. As a complementary part of the pipeline, we present data governance tasks (section 3), which ensure an effective and efficient use of data. Finally, we detail the deliverables (section 4).

1. Stages (P2)

1.1 Trusted zone

Raw data comes with all sorts of mistakes or inconsistencies that need to be addressed before it can be used in downstream tasks. This *pre*processing is an essential step of any pipeline that handles data in some capacity. Nonetheless, in the context of our project we are working with a lot of data, and it is very likely that some of this data is going to be used by different exploitation tasks. Hence, executing *generalized* transformations that prepare data for a **specific** application (e.g. training a given ML model) is not a recommended practice, and should be left for further down the pipeline.

For example, a very common preprocessing task prior to training a model is to remove outliers. From the perspective of achieving good predictive performance, removing outliers makes sense, as highly abnormal values can significantly bias the predictions and reduce the efficacy of the average inference. However, if in our pipeline we use the same data to train such a model and also to do, for instance, anomaly detection, the latter task is being significantly conditioned by the processing we did for the former. That is, we are removing the very same thing we aim to detect. Hence, specific data processing techniques should be left for each of the exploitation tasks and their requirements.

As a result, in the trusted zone the goal is to conduct ***generic* data quality tasks** that do not interfere with posterior analytical needs. Another way to think about this is as "correcting the mistakes" that the data contains as a result of its unprocessed ingestion in the landing.

Cleaning the data

Note that the specific processing tasks to be conducted depend on the data that is being handled. For example, structured data such as CSV files can undergo a lot of transformations to ensure its quality, given the constraints associated with the tabular format and a fixed schema. On the other hand, semi-structured data, such as JSON files, might be trickier to process given their variable schema.

We now list some examples of tasks that can be conducted in this zone. Note that not all tasks need to be implemented (nor it might make sense to do so), as these depend on the data that you have:

- Deduplication: removing repeated elements (e.g. based on primary key)
- Schema correction: ensuring data is casted to its proper type.

- Standardization (e.g. define a common format for dates, perform currency conversion or translate text to a specific language).
- Constraint validation, based on business rules (e.g. non-negative ages or correctly formatted emails).
- Schema flattening (e.g., explode arrays, extract nested fields).
- Text normalization (e.g. lowercasing, removing punctuation).
- Spelling correction.
- Compress data for easier handling (mainly for unstructured data).
- Check file integrity and remove/fix corrupted files (mainly for unstructured data).

Tasks

To implement the trusted zone **you need to**:

1. Select the required data storage tools (i.e. databases, see section 2) to implement the trusted zone and set them up.
2. Define where each of your data assets will be stored, and which transformations will be applied to each.
3. Implement the pipelines that move the data from the landing to the exploitation zone. Do so with appropriate technology (see section 2).

Factors that will **positively contribute** towards the grade:

1. Thoroughness and correctness of the cleaning tasks

1.2 Exploitation zone

In the trusted zone we have cleaned the data, which means that it is ready to be exploited to generate insights. However, this data is likely to be poorly organized, as the bucket/folder distribution assigned in the landing zone is the only structuring performed so far. Hence, our goal in the exploitation zone is to provide the data in the best way possible for analysts to conduct their work.

In doing so, we face again the problem of having to provide data for a potentially large series of analytical tasks. Hence, the data organization plan we define has to obey, not to the needs of a specific downstream system, but to a broader principle of flexibility and reusability. This means that, rather than tailoring datasets for individual reports or models, we aim to create **subject/domain-oriented structures** that can serve multiple purposes. The goal is to empower analysts and data scientists to focus on extracting value from data rather than spending time wrangling or interpreting it.

To achieve this, data is (i) modeled in a way that reflects business entities and their relationships. Additionally, (ii) we might compute new data or structures (described below) that provide additional information to the analyst.

Organizing

The organization of the exploitation zone highly depends on the data that you have, and there is no clear pattern to do so. This lack of a one-size-fits-all approach emphasizes the importance of adapting the structure based on the specific nature of your data. It is useful to think about it as the **ground truth** for subsequent tasks, that is, the data that will be given to the analysts for them to perform whatever processes they need.

The first differentiation is regarding the type of data used. Unstructured data has no schema, so all we can do to facilitate the analyst's work when selecting the data to work with is to store it in buckets that clearly indicate the data that is contained. The same structure employed in the landing zone might suffice, or some, more specific, buckets might need to be created.

Structured and semi-structured data, on the other hand, have a schema, so the organization can be both at the file level as well as at the schema level. This can be more clearly seen with structured data. As we know the schemas beforehand we can perform joins (i.e. **integrate data**) between tables that provide information about the same set of entities (e.g. patients; we can join a table containing patient visit records with a table containing diagnostic results). On the other hand, we might want to **vertically partition** a table to reduce redundancy. For example, you might have a table where each row identifies an actor appearing in a movie, alongside properties of the actor (age, place of birth, etc.). Due to an actor being able to participate in a lot of movies, you are going to have a lot of repeated values, as the actor information will be replicated for each new entry. As such, you can create a table *Movies*, a table *Actors* and an intermediary table with foreign keys that references both of them.

Similarly, different versions of the same dataset belonging to different timeframes and ingested at different times (e.g. country economic data; one dataset for each month, each with the same set of attributes) can be combined (i.e. **table union**) into a single table that contains all the historical records of that dataset. On the other hand, a table can be **partitioned horizontally** (e.g. based on a criterion such as geographic region) to provide smaller datasets tailored for more specific analyses, which can help optimize storage and performance by grouping similar records together.

Structured data's consistent schema makes these transformations and integrations relatively straightforward. With semi-structured data, such as JSON or XML files, the process is somewhat more complex due to the flexible and potentially nested structure of the data. However, since there is still an implicit schema present, some level of transformation and organization is possible. For instance, fields that appear consistently across entries can be extracted and normalized into structured tables, enabling similar operations like joins or partitions to be performed once the semi-structured data has been partially structured.

Computing new data

Besides organizing the data, we can use the exploitation zone to calculate additional assets that provide extra benefits for the analysts. Once again, the idea is to generate information that can potentially be used by a wide variety of tasks, preventing the need to re-compute these values everytime the analytical process needs to be conducted, thereby improving efficiency and consistency across the subsequent tasks.

When working with unstructured data we can enhance its analytical value by extracting structured representations that semantically describe its content. For example, textual data can be summarized, classified into categories (e.g., "review", "complaint", "inquiry"), or converted into embeddings that capture semantic meaning. Similarly, image data can be categorized (e.g., "cat", "dog", "vehicle") or transformed into vector representations for machine learning applications. By storing these semantic characterizations in the

exploitation zone, we eliminate the need to repeat these computationally expensive operations during each analysis cycle. This is particularly critical for embedding generation, which is a prerequisite for many machine learning models to interpret and process unstructured inputs.

For semi-structured and structured data, the exploitation zone can also serve as a staging ground for calculating higher-level metrics and aggregates. Leveraging the existing schema, we can compute and store key performance indicators (KPIs) and other summary statistics that are relevant to the organization's strategic goals. These might include metrics such as average sales per quarter, churn rates, or user engagement trends. By maintaining a historical record of these aggregates, analysts gain immediate access to essential insights that reflect the organization's performance over time, without needing to recompute them with each analysis.

Tasks

To implement the exploitation zone you need to:

1. Define the structure of the exploitation zone, that is, determine how the data will be organized and which transformations you will need to apply.
2. Decide which additional data assets you are going to generate (e.g. KPIs, embedding, labels, etc.) and how.
3. Select the required data storage tools (i.e. databases, see section 2) and set them up
4. Implement the pipelines that move the data from the trusted to the exploitation zone, applying the corresponding transformations. Do so with appropriate technology (see section 2).

Factors that will **positively contribute** towards the grade:

1. Effort in providing a coherent and nuanced organization to the data.
2. Appropriate development of new data/artifacts.

We acknowledge that the exploitation zone can be a bit confusing, and the data that you have might severely limit what you can do. Hence, we are not asking you to completely change the structure of the data or to compute a lot of new assets. The goal of implementing this zone is, simply, that you play around with the problem of organizing the data in interesting ways to maximize its utility. As long as you provide some coherent explanation for the choices you make, it will be fine.

1.3 Data consumption and exploitation

At this point most of the data management aspects have been fulfilled, and data is ready to be employed in subsequent tasks or fed to external systems. As mentioned in previous sections, **the focus on this course lies in the management aspects rather than on the analytical processes** and, hence, the quality or thoroughness of the steps implemented here are not the main focus on the project.

The priorities for this stage are that you think about how to exploit your data (you should have a clear idea of this at the beginning of the project), that the systems to do so are set up (i.e. you have a place to move the data to), and that the adequate connections between the exploitation zone and the data consumption mechanisms are in place (i.e.

something gets in and, ideally, something gets out). For example, if you decide to train an ML model with a subset of the resulting data, we will check whether the data is properly shipped to the appropriate platform, some training process is conducted and the resulting model is stored somewhere. The thoroughness of the training process or the quality of the final model is not a priority.

If time becomes a limitation, it is also acceptable to have "placeholders" rather than exploitation systems. That is, set up a system in which some data gets in and, without any processing, something gets out, which should mimic the result of the process that should be conducted. This is fine, as our main concern lies in the adequate orchestration of the process, not the analytical task itself. Once again, you can use **synthetic data** to fill any gap you need, provided that it helps you to devote more time to the data management aspects.

Some examples of tasks you can implement in this step are:

- Training a machine learning model: Use the processed data to train a predictive model. This could be for classification, regression, clustering, or any other ML task.
- Building a dashboard: Create a visualization tool, such as a Power BI, Tableau, or web-based dashboard, to interactively display key insights from the data.
- Setting up an alert system: Implement a real-time monitoring system that triggers alerts based on specific conditions in the data.
- Developing a recommendation system: Use the data to create personalized recommendations for users.
- Powering a search engine or query system: Make the data searchable through an optimized query system, enabling users to find relevant information quickly.
- Natural language processing (NLP) Tasks: process and analyze textual data using NLP techniques like sentiment analysis or named entity recognition.
- Building a chatbot: create a conversational agent that can understand user queries and provide responses based on the data it is trained on.
- Building an image recognition system: create a system that can recognize objects, faces, or text within images using deep learning techniques.

Tasks

To implement data consumption you need to:

1. Define which mechanisms you are going to use to exploit the data.
2. Select the tools you will use to implement such mechanisms.
3. Ship the data from the exploitation zone onto selected systems.
4. Implement the designed tasks.

Factors that will **positively contribute** towards the grade:

1. Quality and complexity of the implemented processes.

2. Implementing the zones

2.1 Design

The most important part is that each zone **has to store the data after the respective transformations have been applied**. That is, you have to define separate databases for

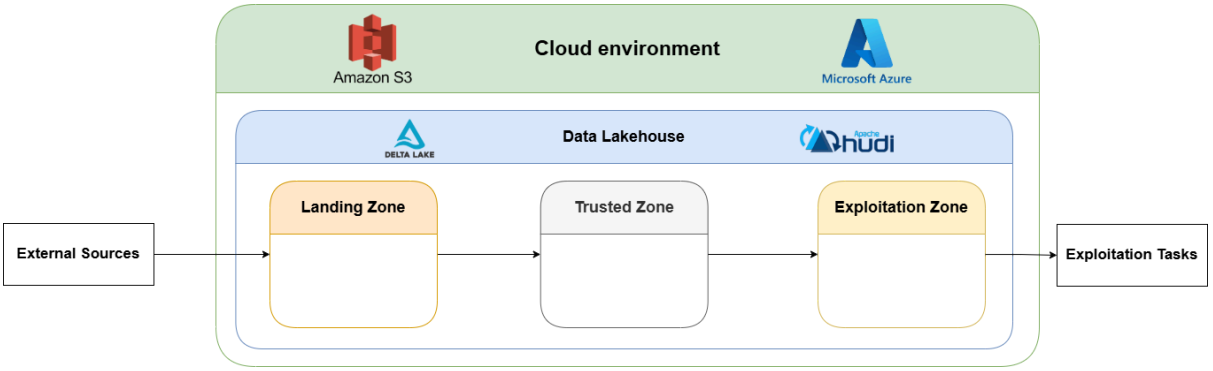
each zone. For example, in the trusted zone you get the data from the landing zone, apply the necessary cleaning tasks, and store the curated data in a separate database. Similarly for the exploitation zone. In this way we create checkpoints that we can go back to access the data in different stages of maturity, either because of specific analytical needs or due to problems in the execution of subsequent tasks (i.e. we do not have to start all the way from the beginning).

In this project we present two main ways of proceeding with the implementation.

1st option: extending your current architecture

To implement the landing zone, distributed technologies are a must due to the variety and volume of the data that will be handled. Companies often opt for, rather than deploying their own distributed cluster, employing cloud services that handle the infrastructure aspects in a transparent manner. For example, the services of AWS S3 or Microsoft Azure can be acquired, and the company only has to worry about sending the data to their servers and retrieving it when needed. Additionally, table formats (e.g. Delta Lake, Apache Hudi) can be employed "on top" of these distributed systems to provide structured data management and ACID transactions, defining a data lakehouse framework.

In your own project, the landing zone should ideally follow a similar pattern. Depending on your setup, you might have implemented a lakehouse structure on a local file system, or you may be using a cloud environment, or even a combination of both. Regardless of the configuration, **this existing foundation can be extended to implement the remaining zones in your pipeline**. Take the following diagram as an example, in which we combine cloud storage with a data lakehouse architecture:



The idea is to take your current system and include more folders/buckets to represent the next zones. The advantage here is that you continue working within an ecosystem you are already familiar with, which reduces the learning curve and simplifies integration. It's important to note that a full cloud-based lakehouse setup is not a strict requirement. If your project already uses a lakehouse on your local file system, you can simply create additional directories to represent the other zones. Similarly, if you're using a cloud provider, adding more buckets or containers to define new zones will suffice.

In the illustrative example above, we highlighted AWS S3 and Microsoft Azure, alongside Delta Lake and Apache Hudi, as possible technologies to set up the entire pipeline.

However, these are not the only options. We recommend continuing with the tools you have already used, but if you want to try other technologies, you are welcome to do so.

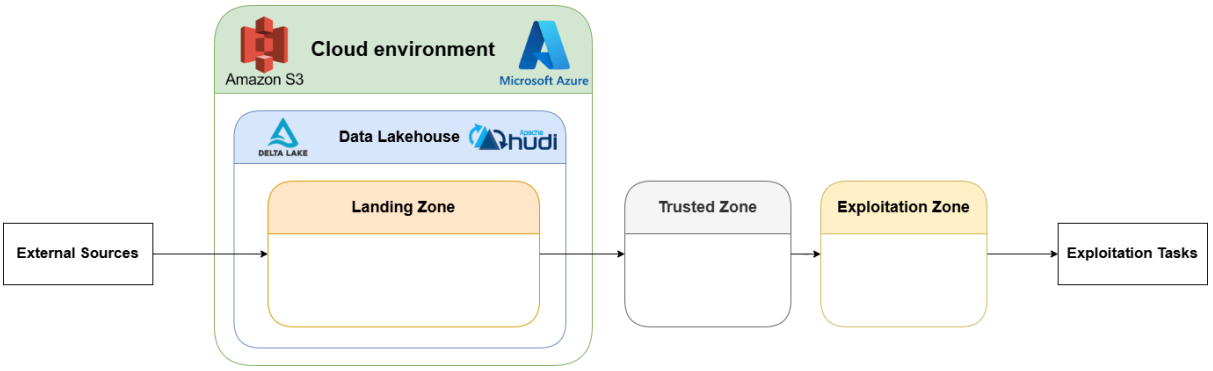
Important note if you opted for a cloud provider: you might not have enough available space in your cloud environment to store all the data of the remaining zones. If that is the case, our recommendation is that you employ a centralized **object storage** (see the *Databases* section below) that "acts" as this cloud platform. For instance, MinIO has the same API as Amazon S3.

It is important to note that this option is the **most common approach in *real* data management systems**, as it does not make sense to move, and process, the data outside of a distributed environment due to the characteristics of modern data, and especially once you have already moved them to such an ecosystem. However, as we are in an academic environment, we present a second approach that will allow you to further play with the design of the system and allow you to utilize more tools.

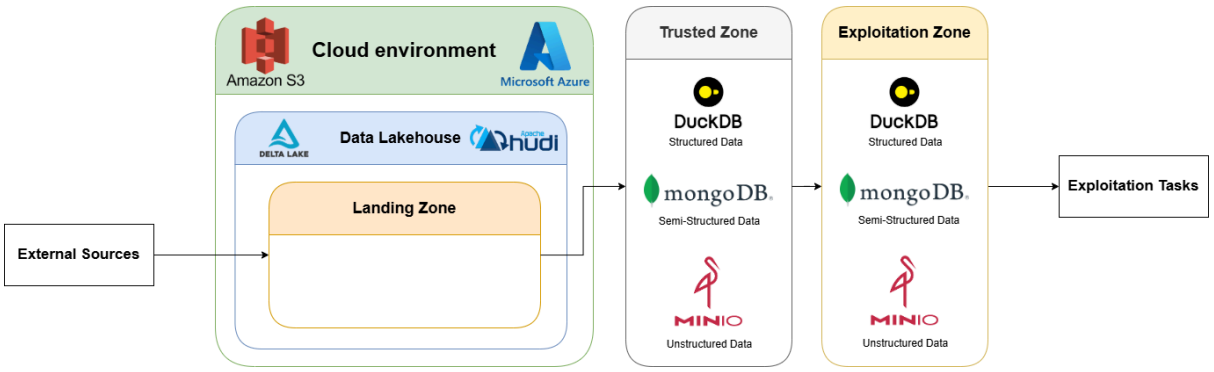
2nd option: moving to smaller, more specialized databases

In traditional data management, before even distributed processing, the pipelines were centered around **data warehouses**. However, these stores could end up containing a lot of data, which could slow the query response time (even in these dedicated systems) and complicate data governance. Moreover, the data ingested by the system might belong to widely different domains, so having all mixed up in a single environment was not ideal.

As a response to that, designers came up with the concept of ***data marts***, subsets of the data warehouse that focuses on a single business function or domain. These smaller repositories are streamlined for quicker querying and a more straightforward setup, catering to the specialized needs of a particular team or function. Their physical implementation was not necessarily different from that of data warehouses, but we can employ this idea of smaller, separated and more specific databases to design a different structure:

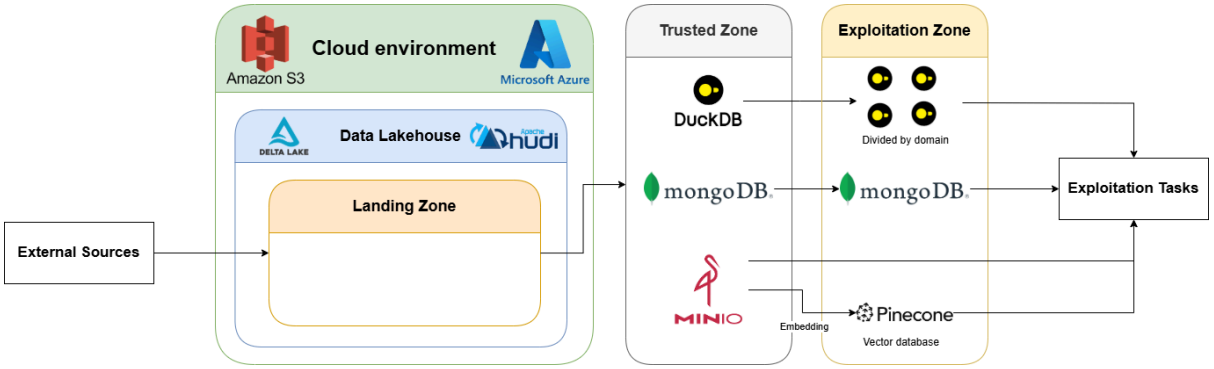


Our new framework is, hence, based on implementing the trusted and exploitation zones in **separate systems**, outside of your chosen technology for the landing zone. This division can be, for example, based on the format of the data:



In the above image we use DuckDB, MongoDB and MinIO to handle each type of data accordingly. These are just examples of tools that you can use, and in the following pages you can find further explanations of each, along with more technologies to implement these zones. As of now, it suffices to know that these are databases, that is, that they can hold data.

Following this idea, you can design the structure that you want based on the criteria that you deem fit. This model will allow you to make more complicated structures, which implies more work on the design aspects, and force you to use different technologies, which is great if there are tools that you want to explore to expand your skill set. A more complex example of this approach can be found next:



In this case, the relational data is separated into several physical stores in the exploitation zone depending on the domain they belong to, which tries to approximate the aforementioned concept of data marts. Additionally, part of the unstructured is transformed into embeddings and stored in a vector database (Pinecone) for efficient handling. The rest of the unstructured data is accessed directly by the exploitation tasks.

Tasks

You need to:

1. Continuing with the work you did with part 1, you have to complete the architecture **diagram** with the new zones, tools, operations, etc.
2. Be sure to adequately reason about the general structure you propose and justify the inclusion of each element.
3. In the following sections we propose some technologies that you might want to use. Be sure to pick the ones that better suit the needs of your pipeline.

Regarding the presented design alternatives, option 1 is more straightforward, whereas option 2 can allow you to make a more interesting pipeline. From an evaluation standpoint, both are perfectly acceptable. Nonetheless, a grade boost will be given to implementations that combine several technologies or define more interesting data management mechanisms.

Also note that you can find an **example of the full project diagram** at the end of this document.

2.2 Databases

The choice of database depends on the specific type of data and the use case at hand. Traditionally, relational databases dominated the landscape, but with the growing complexity and volume of data, newer database models (i.e. NoSQL) have emerged to better handle diverse requirements. Next, you can find a brief overview of the main typologies and some recommendations. Before implementing the pipeline you should decide what type of databases better suit your needs.

Relational databases: you should be familiar with this model at this point. Tools like Oracle, MySQL and PostgreSQL still lead the charts for the most popular systems to store data. Nonetheless, these databases store data row-wise, which makes them great for dealing with transactions, but not so much to deal with analytical workloads (mainly based on aggregations). To fix this, new and "modern" relational databases have been created, which have columnar storage, seamless integration with other data science tools and optimized for analytical queries. Our recommendation is that you try some of these new tools, particularly **DuckDB**, which has the advantages of being an embedded database (i.e. the *database* is a file that can be interacted with from anywhere) and is very easy to use.

Key-Value stores: a non-relational database where records are stored and retrieved based on a unique key, similar to a hash map or dictionary. This structure allows for quick lookups and is well-suited for caching and other fast-access requirements. The most popular key-value store is **Redis**.

Document stores: a specialized key-value store, where a document is a nested object (we can think of each document as a JSON). Documents are stored in collections and retrieved by key. Document stores are great for semi-structured data like JSON, XML, or BSON, and allow for flexible schema design. **MongoDB** (which you have seen during the course) is the most popular document store, and ideal if you need fast read/write queries. Another interesting option inside the AWS ecosystem is **Amazon DynamoDB**.

Wide-column: optimized for storing large amounts of data with high write throughput and low latency, wide-column stores are designed for scalability. These systems use column families instead of rows, allowing them to efficiently store vast amounts of unstructured or semi-structured data. During the course you have seen **HBase**, which can be set up outside of the Hadoop environment and in an embedded mode. Another alternative is **Apache Cassandra**, which is the most popular wide-column store.

Graph databases: explicitly store data with a mathematical graph structure (as a set of nodes and edges). Roughly speaking, graph databases are a good fit when you want to analyze the connectivity between elements. **Neo4J** is a good option, as you have seen/will see it in other courses and it is the most popular graph database.

Time series: a time series is a series of values organized by time. For example, stock prices might move as trades are executed throughout the day, or a weather sensor will take atmospheric temperatures every minute. Any events that are recorded over time are time-series data. A time-series database is optimized for retrieving and statistical processing of time-series data. The most popular tool is **InfluxDB**.

Vector databases: stores and queries embeddings or vector representations of data. These vectors are typically generated using machine learning models like BERT that map data into high-dimensional space. Vector databases are becoming increasingly popular, especially in fields like NLP and search systems that require high-performance similarity search for multi-dimensional vector data. The most popular tool is **Pinecone**.

Object stores: manages data as objects, rather than files or blocks. This is commonly used in cloud computing, and services like Amazon S3 and Azure Blob Storage are popular examples of object stores. They are a good option for unstructured data, which does not fit neatly into the other database types. Besides cloud options, we recommend **MinIO**, which follows the API of Amazon S3, but can be used in a centralized manner.

2.3 Transformations

Once we have defined how the zones will be implemented, it's time to talk about how to generate the transformations that data will undergo while travelling from one zone to the next.

Distributed, large-scale processing with Spark

We are working with data sourced from large, model-agnostic repositories. Because of their size and complexity, processing this data efficiently requires tools that can operate at scale and, ideally, in distributed computing environments. To address this challenge, you will be introduced to **Apache Spark** during the course, a leading technology in the big data ecosystem that is widely used for scalable data processing. Spark is built to handle large datasets across multiple nodes, which makes it also suited to handle real-time or near-real-time pipelines.

You may notice that defining data transformations in Spark differs significantly from working with familiar Python libraries like Pandas. While Pandas offers simplicity and ease of use for in-memory data processing, Spark emphasizes performance, fault tolerance, and distributed execution. As a result, Spark code can feel more verbose and complex, but this added complexity is intentional, and essential for ensuring that your data workflows remain efficient and reliable. You will learn how Spark ensures these properties in the theory part of the course.

Local, smaller-scale transformations with traditional Python

Should you use Spark in *any* process that moves data from one place to another? The answer is, clearly, no. Spark is great when you have to move a lot of data, but, as stated, above, when your scale is smaller, employing Spark is an unnecessary cost. A good rule of thumb involves checking your RAM memory. If the data you work with fits comfortably in memory, Spark is likely not needed.

Depending on the pipeline that you are implementing, you might have some parts in which you know that the amount of data you have is relatively reduced. This will likely be the case for most exploitation tasks at the end of the pipeline, where you work with a subset of the data. Hence, in those scenarios you can develop **dedicated scripts employing traditional, lower-scale Python libraries** such as Pandas.

Streaming

If on P1 you defined streaming ingestion, in P2 you will have to link that streaming with the corresponding downstream task where it is going to be used. A question that might arise here is whether the streamed data should be processed or not.

Conceptually, if you define a hot path (i.e. real-time), data should be sent without delay, so any processing that we did, even if minimal, would not be desired, as the data shipping would be slowed down. On the other hand, a warm path, in which the latency is higher, can more comfortably accommodate some processing before moving the data to its final destination. In a *practical* setting, and as with most things in big data, whether to process the data or not depends on a lot of factors, and there is not a clear, general answer.

For your project, you can decide to process or not the data you stream. You can justify your decision based on the above criteria or any other you deem appropriate. If you decide to apply some processing, we recommend checking the following tools:

- **Spark**: the Spark core installation is equipped with functionalities to ingest streamed data (from Kafka, for example), process it, and send it to another streaming pipeline. You will be using Spark for the batch-ingested data regardless, so it is a good option to also employ it here.
- **Kafka Streams**: a complimentary tool inside the Kafka ecosystem to natively process the Kafka streams. It is more lightweight and specialized than Spark, but it is implemented in Java. If you are not familiar with the programming language, it is best to avoid it.
- **Apache Flink**: a separated tool that can be employed for any type of processing (not only streaming). It is more lightweight than Spark, which allows for lower latencies, and can be utilized via a Python library.

Processing technologies like Spark or Flink consume data from Kafka streams, perform their transformations, and can either write the results directly to a database or publish the processed data to another Kafka topic. This allows you to continue moving data downstream in a fully streaming manner.

3. Governance (Optional)

Note that the governance tasks we describe below are **not** mandatory. As with many aspects highlighted before, including governance processes will simply **positively contribute** towards the grade. Please check the guidelines at the end of the section.

As data pipelines expand to include more zones, tasks, and technologies, their increasing complexity must be addressed through effective data governance. Data governance is a **high-level structured framework that formalizes the management of data** through policies, standards, processes, and ongoing monitoring.

Data governance is a multifaceted concept where multiple scopes need to be considered simultaneously. These scopes include determining whether governance is needed within a single organization (intra-organizational) or across multiple entities (inter-organizational), the involved decision domains (e.g., data quality, security, access control), and understanding the nature of the data involved (e.g., social media, streaming data, biometric data).

When designing data governance, it is important to understand how these scopes tie together to maximize data value while minimizing risks and costs. This can quickly become complex and often involves navigating trade-offs between control, agility, and costs.

A good starting point is to reflect on your business domain and define what you aim to achieve. In many cases, a centralized governance model may be appropriate, where a single authority oversees data ingestion, storage, and delivery across the organization. However, your governance strategy should also reflect your priorities. For example, is securing sensitive customer data your top concern? Or is it more important to ensure trust in machine learning outputs through robust data quality controls?

Within this context, several key governance domains can be explored. In the following sections, we describe a set of data governance concepts and decision domains that are relevant at this stage of your architecture. For each, we also recommend tools and technologies to help you operationalize (i.e., put in practice) data governance.

Data domains and data products

At this stage, your data should be cleaned and available in the exploitation zone. You may have already started organizing data according to business entities or anticipated downstream data science pipelines. Building on this, a common governance strategy in modern big data architectures (e.g., Data Mesh) is to **view data not simply as individual datasets, but as data products grouped within data domains**.

A data product is a well-defined, reusable dataset or API that is treated with the same care and discipline as a software product. Each data product should have clear ownership, documentation, and service-level objectives. It is designed to be consumed by other teams or systems, and is typically aligned with a specific domain or business process. Data products play an important role in governance. Rather than governing every table or pipeline independently, organizations can attach policies, ownership,

quality checks, and access control rules at the level of the data product. This creates a scalable, modular approach to governance.

Organizing data products within the exploitation zone in data domains (e.g., Sales domain) helps in adding another layer of organization since each data product is aligned with a specific area of business activity and thus, may have common standards, policies, and metadata.

Metadata

You have already seen the importance of metadata during P1. Metadata helps describe and organize the actual data and is a fundamental building block for operationalizing governance, enabling data discovery, monitoring, and traceability.

Graph-based structures can be used to describe metadata elements and their relationships, and several vocabularies and ontologies are available to standardize these descriptions. A good starting point is to design a data catalog that provides an overview of the available data products in the exploitation zone and their relationships. The **DCAT** ontology can be used to model and standardize this catalog, and tools such as **Apache Atlas** can be used to support its implementation.

Another interesting usage of metadata is to implement lineage tracking or traceability, which tackles on how data moves and transforms across the different zones, enabling to trace back the origin of a dataset, the steps it has undergone, and what were the tasks and interactions. This can be especially useful to debug data issues or meet audit requirements.

Data Quality

Data integration and enrichment tasks can become complex processes that often involve numerous operations across various data sources. As such, errors or inconsistencies in these processes can lead to unreliable or inaccurate data being served to the exploitation zone. To mitigate these risks, monitoring tools can be used to ensure data quality throughout the pipeline. Tools such as **Great Expectations** can be integrated to define data quality requirements, validate data, and trigger alerts or corrective actions when quality issues are detected.

Data Security

An essential aspect of governance is managing who has access to what data. This includes defining roles, assigning permissions, and implementing policies that restrict access based on user roles, data sensitivity, or zone location.

For example, access to raw data in the Landing Zone may be restricted to data engineers, while curated datasets in the exploitation zone may be more broadly available to analysts and data scientists. Policies should reflect business needs and compliance requirements, such as GDPR.

Although data security and privacy are complex topics with rich research backgrounds, for the purpose of this project, you can imagine your organization and simulate governance roles and access levels across zones. Tools such as **Apache Ranger** or cloud-native **IAM systems** can help enforce and monitor access control.

Tasks

Over your resulting data architecture, **begin to design your data governance strategy**. As previously discussed, data governance requires a shift from purely technical considerations to also include organizational, strategic, and business-aligned scopes. In this phase, we are particularly interested in how well you align your governance proposals with your business context.

1. Take an organizational and strategic point of view. Begin by reflecting on the **data domains** that are relevant to your business. Then, based on these domains, propose a **set of data products** that should exist in your exploitation zone. Are these data products similar or different from what you initially had in your exploitation zone? What insights led you to adjust or reaffirm your original design?
2. Provide at least **one detailed example of a data product** and the metadata you would associate with it to support governance (e.g., description, owner, quality metrics, policies, data lineage, etc.). You may use this template to support your specification.
3. Choose at least **one governance mechanism** (e.g., data quality validation, access control, lineage tracking, cataloging) **and implement it** over your architecture. What are the benefits and potential trade-offs or costs associated with this mechanism?

We will pay special attention to your reasoning, justification of choices, and how clearly you link governance elements to your business case.

4. Deliverables

Main deliverable (P2)

The tasks described in this document correspond to the **second deliverable**, which include:

- **Architecture design** (updated version with the new zones, tools, processes, etc.)
- Stage 3: trusted zone
- Stage 4: exploitation zone
- Stage 5: consumption tasks
- Governance tasks (if applicable)

To implement each stage follow the appropriate descriptions, specifically regarding the *tasks* subsections.

Note that complementary aspects of P1 that are kept for P2 will continue to contribute towards the grade. These are, mainly, **containerization** and **orchestration**.

The structure of the deliverable will be the same as for P1. You have to present a document with all the necessary information to understand the pipeline you implemented, including the appropriate justifications. Additionally, you have to provide access to a Github repository with the code you implemented, adequately documented to be downloaded and executed.

Follow-up deliverables

There will be two follow-up deliverables in this second part, each before their corresponding follow-up sessions. This means that you will have roughly two weeks to deliver them.

First deliverable:

- Architecture design (first version).

Again, the first deliverable does not need to include any code. Note that we still encourage you to start the implementation as early as possible, as if you leave most of it for the last weeks it might conflict with other project deliveries and final exams.

Ideally, the *design* should include, besides the diagram, a clear view of what you want to do. That is, how are you going to set up each zone, which transformations are you going to apply to the data, how far are you going to go in each consumption task, etc. In summary, during the first two weeks you should go over the entire document and decide not only what you want to do, but how; with as much detail as possible.

Second deliverable:

- Architecture design (consolidated).
- Basic implementation of trusted zone, exploitation zone and consumption tasks.

During these two weeks you should focus on setting up all the necessary infrastructure for the entire pipeline. That includes the databases, the downstream tasks, streaming pipelines and all the communications between the zones, with the necessary orchestration and containerization. The quality of the processing at each step is not the priority, but rather that you can move and store data where you need to.

The final two weeks should be devoted to implementing the actual transformations to ensure that the data undergoes all the desired changes, besides any additional aspects you might want to include (especially regarding data governance).

5. Pipeline design example