Big Data Management
Building a Big Data Architecture

Project introduction, general structure and guidelines

1. Introduction

In the early 2000s, as relational, expensive and monolithic systems buckled under the needs of modern data, updated approaches were needed to handle data growth. The new generation of systems had to be cost-effective, scalable, available, and reliable. Commodity hardware became cheap and ubiquitous, and several innovations allowed distributed computation and storage on massive computing clusters at a vast scale. Additionally, cloud computing and NoSQL systems enabled new paradigms to process data. These innovations started decentralizing and breaking apart traditionally monolithic services. The Big Data era had begun. As companies saw their data grow into many terabytes and even petabytes, the profile of **big data engineers** emerged.

The explosion of data tools in the late 2000s and 2010s meant that, in order to effectively use these tools and techniques, big data engineers had to be proficient in software development and low-level infrastructure hacking. However, despite the power and sophistication of open source big data tools, managing them was a lot of work and required constant attention. Big data engineers often spent excessive time maintaining complicated tooling and arguably not as much time delivering the business's insights and value. Open source developers, clouds, and third parties started looking for ways to abstract, simplify, and make big data available without the high administrative overhead and cost of managing their clusters, and installing, configuring, and upgrading their open source code.

Whereas data engineers historically tended to the low-level details of monolithic frameworks such as Hadoop, Spark, or Informatica, the trend is moving toward **decentralized, modularized, managed, and highly abstracted tools**. Popular trends in the early 2020s include the modern data stack, representing a vast collection of off-the-shelf open source and third-party products assembled to make analysts' lives easier. At the same time, data sources and data formats are growing both in variety and size. As such, data engineers increasingly find their role focused on things higher in the value chain: data management, data architecture, orchestration, and general data lifecycle management. Data engineering is increasingly a discipline of **design and interoperation:** connecting various technologies in streamlined processing workflows to serve ultimate business goals.

This is the **goal of this project**: to build a Big Data management architecture. This will entail designing the pipeline(s) to ingest, store, process and serve the data, requiring the adequate selection and orchestration of the technologies, tools and/or frameworks that best fit the needs of your system.

2. Project structure

DataOps

DataOps maps the best practices of Agile methodology, DevOps, and statistical process control (SPC) to data. Whereas DevOps aims to improve the release and quality of software products, DataOps does the same thing for data products, which differ from software products because of the way data is used. DataOps has three core technical elements: automation, monitoring and observability, and incident response. Of these, automation is the crucial facet that differentiates slow, error-prone and rigid data management workflows employed traditionally, from fast, reliable and extensible architectures implemented by modern companies.

To ensure the adequate automation of data-handling flows, it is necessary to define a framework to operationalize the different stages and processes required to properly treat the incoming data. Note that there is no gold standard on how to approach the definition of such architecture besides some high-level patterns, as the design highly depends on the type of data handled and the business goal. Hence, in this course we have selected a high-level framework to guide and structure the implementation of the project, but do not assume it is the only or best way to approach such a task.

The framework

The selected framework is composed of **5 stages**, with three of them being storage layers. This pattern has been present in data management architectures for decades, and modern iterations of this same structure encompass the *medallion pattern* or, more generally, the *modern data platform*. It is based on the separation of data in three states: raw, processed and curated. First, we store the raw data into a large repository (i.e. the ***landing zone***, also known as the *bronze* layer), which facilitates further processing as it prevents having to deal directly with the sources. This raw data undergoes transformations, cleansing tasks, and enrichment processes to prepare for effective usage; the resulting product being stored into smaller and specialized databases (the ***trusted zone***, also known as the *silver layer*). Finally, data is further structured, refined and optimized for business intelligence, analytics, and machine learning applications. This final, curated data is stored in these same dedicated databases (the ***exploitation zone***, also known as the gold layer), but now it is prepared for efficient querying and exploitation.

More precisely, the stages are:

1. **Data Ingestion**: This initial stage involves collecting data from various sources, including databases, APIs, streaming services, and external files. The data is extracted in its raw form and loaded into the landing zone without significant transformations.
2. **Raw Data Storage** (Landing zone): Once ingested, the raw data is stored in a large and scalable repository such as a data lake or a distributed storage system. This stage ensures data availability for further processing while preserving historical records.

3. **Processed Data Storage** (Trusted zone): raw data undergoes cleaning and transformation processes, such as quality checks, deduplication, schema standardization, and enrichment activities, all applied to enhance usability before the data is employed in more specific tasks. This step is often implemented via tools designed to handle massive data volumes distributedly, and the processed data is typically stored in structured or semi-structured formats within optimized databases or data warehouses.
4. **Curated Data Storage** (Exploitation zone): The processed data is further refined, the goal being to provide data in the best way possible to be consumed. Complex computations, data integration, schema restructuring, and predefined aggregations are carried out in this stage. Generally, distributed technologies are not needed, and the processing can be conducted with smaller, more dedicated tools (e.g. Python code). The final data is stored in highly optimized databases, data marts, or specialized analytical platforms.
5. **Data Consumption**: The last stage involves serving data to end users, applications, and analytics tools. Data from the exploitation zone is accessed through APIs, dashboards, reports, and direct queries.

Additional considerations

It is crucial to acknowledge the previous step (i.e. "step 0") necessary to the design of any structure: its adequate **contextualization**. Before planning and implementing the architecture, you will have to select the domain you will be addressing, the business value that your system provides (i.e. what problem is it trying to solve) and the datasets that you are going to use to populate your pipeline.

In this project we ask that you follow this framework to structure your pipeline. This will help you in separating the different parts of the architecture and attributing different processes to each. Each zone performs specific tasks, so responsibilities are separated and the characteristics of the data that gets in and out of each zone are known beforehand. This separation of functions facilitates desirable properties of the software infrastructure, such as reusability and openness.

Note that part of the pipelines you implement might not require some of these zones or just a very superficial implementation. On the other hand, you might want to create "new" zones for more specific tasks. This is fine, as this framework should be understood as a general guideline of the steps that data processing requires, rather than a rigid set of rules that must be followed in every scenario. You are allowed to implement the tasks that you see fit in each part of the pipeline as long as they are sensible and fit your data and the tasks to solve.

As you implement each part of the pipeline, be sure to **justify** your design choices. Why did you select this particular technology for ingestion, storage, or processing? What are the trade-offs involved? Any potential alternatives? This reasoning is key to developing well-thought-out solutions and will be critical for the success of your project. We do not ask that you build the "perfect" architecture, but rather that you are able to justify your decisions and weigh the pros and cons of each step.

Finally, in the data engineering sphere there is a plethora of tools to choose from to implement the defined stages. The sheer volume of alternatives can be overwhelming, and you might be unsure of which to select. In the specific project statements we suggest some tools for each step, which tend to be both popular and accessible. As mentioned in the introduction, the goal of modern data architects is to know which is the type of technology that needs to be employed, rather than the specific software. Hence, do not worry about mastering specific tools and rather about understanding the high-level data flows. This is specially the case in the modern environment, where by the time you have fully understood a tool, several others have appeared that do the same thing, but better.

3. Project deliverables and guidelines

Organization and main deliverables

The project will be developed in teams of **three people**, which will be the same for the entire project. If there are remaining students, groups of four members are allowed, but note that the expected quality of the project will be higher.

The evaluation of the project will be separated into **two main deliverables**: one mid-way through the course (**P1**) and one at the end of the semester (**P2**). The content of each deliverable will be detailed in the two statements you will find at the beginning of each part. On a general note, these deliverables include:

- P1: project contextualization, data ingestion and landing zone.
- P2: trusted zone, exploitation zone and data consumption.

For both deliverables we expect two artifacts:

1. An **explanatory document** of your project. This should contain a description of your implementation, justifications on the decisions taken, an overview of the processes implemented and, more generally, any piece of information that helps the supervisor understand what you have done and why.
2. A **project repository** (e.g. Github) where you implement your architecture. The tools we propose to implement the architecture should be easily accessible via Python code, so you should have no major issues when creating this repository. Orchestration and/or technology deployment considerations are further laid out in the separated statements. Instructions on how to access these platforms must be in a mandatory section to be included in the document. Ideally, the supervisor should be able to download and run the entire pipeline.

Logically, both the document and the repository are meant to be incremental. That is, the first deliverable should include the activities regarding the first part, which will be complemented with the second set of tasks for the second delivery.

The grade of the project (P) will be the average of the grades of part 1 (P1) and part 2 (P2):

$$P = 0.5P1 + 0.5P2$$

It is possible to retroactively improve the grade of P1 If noticeable improvements or extension of the P1 tasks are implemented in the P2 deliverable.

Follow-up deliverables

In order to better track your progress in the project, there will be a deliverable **before** each follow-up session. In each, you will have to include parts of the developed architecture (the specific content will be stated in the project statements). The supervisor will use them to provide general guidelines about the project and the implementations (i.e. what is being missed by most groups, suggestions to improve, etc.). Hence, these follow-up deliverables will not directly count towards the grade of the project, as the final grade will only take into consideration the two main deliverables presented in the previous section. Nonetheless, we highly encourage you to work throughout the course and provide your current work in these deliverables, as it will help you to structure the development more naturally along the weeks.

If significant effort is shown, it will be positively noted towards the final grade.

Evaluation guidelines

As stated before, the goal of this project is not to build the perfect pipeline, but rather to reason about your decisions and come up with an adequate solution. Hence, **the document is a fundamental part of the project**, as there you should showcase both rigorous thinking (e.g. detailed discussion about pros and cons, and room for improvements) and soundness (i.e. adequate selection of the tools and solutions for each stage). We expect the document to be well-written and professional.

As for the implementation, it has to present the desirable properties of any software system, to the extent that it is possible to do so. Namely:

- Dynamicity. How easy is it to add a new variable from an existing source? And add a new source? How easy is it to change the transformations executed? Up to which extent these tasks are automatically supported without extra code?
- Reusability. Code and data are not duplicated and they are well organized. A third person should be able to understand your code and run it easily.
- Openness. Your system could be easily extended and evolved with new aspects.
- Reproducibility. The system can be easily executed and is properly documented to do so.

Important note. We want you to be ambitious both when defining the scope of the project as well as implementing it. Basic pipelines, even though they might be correct and adequately justified, **will not be enough to pass the project**. For each section of the pipeline we describe in the project statements, we include both mandatory tasks as well as additional criteria that will be taken into account for the evaluation. We encourage you to consider the inclusion of these complementary aspects, as the more of them you develop, the better the evaluation will be. Logically, including any extensions or extra work not contemplated in the statement will also be taken into account.