**Index**

[**1. Introduction** 2](#_Toc129297174)

[**2. Approaches** 3](#_Toc129297175)

[**2.1 Relevance of the work** 3](#_Toc129297176)

[**2.2 Methodological background of the study** 3](#_Toc129297177)

[**2.3 Subject, object, purpose and objectives of the study** 3](#_Toc129297178)

[**3. Architecture Components for Big Data Processing** 3](#_Toc129297179)

[**4. Big data streaming architecture** 3](#_Toc129297180)

[**4. System development** 3](#_Toc129297181)

[**4.1 Conception phase** 3](#_Toc129297182)

[**4.2 Development phase/reflection phase** 3](#_Toc129297183)

[**4.3 Finalization phase** 3](#_Toc129297184)

[**4. Conclusion** 3](#_Toc129297185)

[**References** 4](#_Toc129297186)

# **1. Introduction**

The story begins with a Data Engineer who works at a bank that possesses and processes various datasets. Considering the amount of data that needs to be processed, an application has been developed that can handle large datasets ranging from live streams to archived data.

As a Data Engineer, his main task is to ensure that a backend is developed for the data-intensive application. Various aspects need to be considered, such as selecting the underlying infrastructure that is GDPR-compliant and follows common regulations.

The Data Engineer is responsible for developing efficient methods for pre-processing large datasets and then directly processing them in an ML model. Standard principles of a Data Engineer must be applied to ensure that the application is reliable, scalable, and easy to maintain.

The Data Engineer must consider various options to ensure that the application is reliable, scalable, and easy to maintain. He must also create a directory on Github to ensure that developers can work efficiently on the application.

Application security is an important aspect, and the Data Engineer must identify possible hazards and take appropriate measures to minimize them.

In the conception phase, the Data Engineer must create a portfolio to convince management of how to solve the problem of processing large datasets that must be GDPR-compliant and professionally processed in an application with ML models and made available to management. This portfolio describes the necessary hardware and software components, how they should be implemented, how the application will be built, and how the ML model will be integrated into the application.

The portfolio also describes the value that this work will bring to the company, as well as the pros and cons of the various options. Possible risks are also described, along with how they can be avoided. With this portfolio, the Data Engineer can persuade management of the effectiveness and efficiency of the proposed solution and successfully advance the implementation.

# 

# **2. Approaches**

## **2.1 Relevance of the work**

The relevance of the work lies in the growing need for organizations to effectively manage and process large volumes of data. As data volumes continue to grow, traditional database systems may not be able to handle the scale and complexity of the data. This has led to the development of big data architectures, which enable the processing and analysis of data that is too large or too complex for traditional database systems.

In this context, the batch-processing-based data system is an important solution for managing and processing large volumes of data. It enables organizations to collect, process, and store data in a scalable and efficient manner. This is particularly relevant for data-intensive applications such as e-commerce platforms, social media, and healthcare systems, where large volumes of data are generated on a daily basis.

Furthermore, the batch-processing-based data system can provide valuable insights into business operations and consumer behavior. By processing and analyzing large volumes of data, organizations can identify patterns, trends, and anomalies that may not be visible through traditional data analysis methods. This can help organizations to make informed decisions, improve their products and services, and ultimately, increase their revenue and profitability.

## **2.2 Methodological background of the study**

The methodological background of the study involves a systematic and structured approach to building a batch-processing-based data system. The study follows a three-phase approach, including the conception phase, development phase/reflection phase, and finalization phase.

During the conception phase, the study begins by identifying the business requirements of the data-intensive application, defining the data model, choosing the appropriate tools, and designing the data flow. This phase is crucial as it sets the foundation for the entire system, and any mistakes or oversights at this stage can result in significant problems later on.

The development phase/reflection phase involves developing the data processing code, testing the system, evaluating performance, and refining the system as needed. This phase is iterative and involves continuous testing and refinement to ensure that the system can handle large volumes of data and provide accurate output.

Finally, in the finalization phase, the study involves deploying the system, monitoring its performance, and maintaining it over time to ensure its continued reliability and scalability.

The study employs a mix of quantitative and qualitative methods, including performance evaluation metrics, code optimization techniques, and system monitoring and maintenance practices. The study also relies on best practices in big data architecture and batch processing, as well as the expertise of experienced data architects and engineers.

## **2.3 Subject, object, purpose and objectives of the study**

Subject of the study: The subject of the study is the development of a batch-processing-based data system for a data-intensive application. The study focuses on the design, development, testing, deployment, and maintenance of the system.

Object of the study: The object of the study is to build a reliable, scalable, and maintainable batch-processing-based data system that can handle large volumes of data and provide accurate output. The system is designed for data-intensive applications, such as e-commerce platforms, social media, and healthcare systems.

Purpose of the study: The purpose of the study is to demonstrate the effectiveness of a batch-processing-based data system for managing and processing large volumes of data. The study aims to provide insights into the design, development, testing, deployment, and maintenance of the system, as well as best practices in big data architecture and batch processing.

Objectives of the study:

* To identify the business requirements of the data-intensive application and design the appropriate data model for the batch-processing-based data system.
* To choose the appropriate tools and technologies for the batch-processing-based data system.
* To design the data flow for the batch-processing-based data system to ensure scalability and efficiency.
* To develop the data processing code for the batch-processing-based data system and optimize its performance.
* To test the batch-processing-based data system thoroughly and evaluate its performance.
* To refine the batch-processing-based data system based on the results of the testing and evaluation.
* To deploy the batch-processing-based data system and monitor its performance in a production environment.
* To maintain the batch-processing-based data system over time, including updating software dependencies, optimizing code, and upgrading hardware as needed.

# **3. Architecture Components for Big Data Processing**

Architecture components for big data processing depend on the specific technologies and tools used in a given system. However, the following are some examples of common architecture components for big data processing:

Data ingestion components:

* Apache Kafka: a distributed streaming platform used to collect and process large volumes of data in real-time.
* Apache Flume: a distributed data collection and aggregation system used to efficiently ingest and transfer large volumes of log data.

Data processing components:

* Apache Spark: a fast and general-purpose distributed computing system for large-scale data processing.
* Apache Hadoop MapReduce: a distributed computing framework used to process large amounts of data in parallel.

Data storage components:

* Apache Hadoop Distributed File System (HDFS): a distributed file system used for storing and processing large volumes of data.
* Apache Cassandra: a distributed NoSQL database used for storing and managing large volumes of structured and unstructured data.

Data access components:

* Apache Zeppelin: an open-source web-based notebook for data analysis and visualization.
* Apache Drill: a distributed SQL query engine that supports complex queries across multiple data sources.

Data governance components:

* Apache Atlas: a scalable and extensible metadata management system that enables organizations to manage their data governance requirements.
* Apache Ranger: a framework for managing security policies across the Hadoop ecosystem.

Infrastructure components:

* Cloud infrastructure: such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform, which provide scalable computing and storage resources on-demand.
* High-performance computing clusters: such as Apache Mesos or Kubernetes, which enable efficient use of computing resources for big data processing.

These concrete architecture components work together to provide a robust, scalable, and efficient big data processing system that can handle large volumes of data and provide valuable insights and information for organizations.

# **4. Big data streaming architecture**

Concrete big data streaming architecture involves processing and analyzing large volumes of data in real-time or near-real-time. The following are some key components of a big data streaming architecture:

* Data sources: Data sources can include sensors, log files, social media feeds, and other real-time data streams.
* Data ingestion: Data ingestion components such as Apache Kafka or Apache Flume can be used to collect and store the streaming data.
* Data processing: Data processing components such as Apache Spark Streaming or Apache Flink can be used to process the streaming data in real-time, enabling near real-time insights.
* Data storage: Data storage components such as Apache Cassandra or Apache HBase can be used to store the processed data.
* Data access: Data access components such as Apache Drill or Apache Superset can be used to provide users with access to the processed data.
* Data governance: Data governance components such as Apache Ranger or Apache Atlas can be used to manage the security and compliance requirements of the streaming data.
* Infrastructure: Infrastructure components such as high-performance computing clusters or cloud infrastructure can be used to provide scalable computing and storage resources for the streaming data.

# **4. System development**

System development is the process of designing, developing, testing, and deploying software systems to meet specific business requirements. The following are the key steps involved in the concrete system development process:

Requirements gathering: The first step in the system development process is to gather requirements from stakeholders. This involves conducting interviews and workshops to understand the business needs and document the system requirements.

System design: Once the requirements are gathered, the system design phase involves creating a detailed architecture and design for the system. This includes defining the system components, data models, and user interfaces. This phase may involve the use of tools such as Unified Modeling Language (UML) diagrams, data flow diagrams, and wireframes.

Development: The development phase involves writing code to implement the system design. This includes programming, testing, and debugging. The development phase may involve the use of programming languages such as Java, Python, or Ruby, and software development tools such as integrated development environments (IDEs), code editors, and version control systems.

Testing: Once the code is developed, it needs to be tested to ensure that it meets the business requirements and is free of bugs and errors. This involves unit testing, integration testing, and system testing. The testing phase may involve the use of automated testing tools such as Selenium, JUnit, or TestNG.

Deployment: Once the system is tested and validated, it can be deployed to the production environment. This involves configuring the system on the target infrastructure and ensuring that it operates reliably in a production environment. This phase may involve the use of deployment tools such as Ansible, Chef, or Puppet.

Maintenance: Once the system is deployed, it needs to be maintained over time to ensure its continued reliability and scalability. This includes updating software dependencies, optimizing code, and upgrading hardware as needed. The maintenance phase may involve the use of monitoring tools such as Nagios, Zabbix, or Prometheus.

## **4.1 Conception phase**

In this project, the goal is to design and implement a data architecture for a data-intensive large-scale application. The project requires adaptation of principles such as Infrastructure as Code (IaC) and Microservices (MS) to build the backend of data-intensive applications using state-of-the-art concepts and methods. The project aims to design a reliable, scalable, and maintainable batch-processing data system for a data-intensive machine learning application. The designed system must be able to ingest massive amounts of data, store the data effectively, pre-process parts of it, and aggregate the data for direct usage in a machine learning application. The data infrastructure will be designed in a way that data processing is scheduled to be conducted in batches, and data engineering principles will be adapted during the process.

Software Components

To build a data-intensive application, familiarizing oneself with common software components is necessary. In this project, the following software components will be used:

Apache Kafka: Kafka is a distributed streaming platform used for building real-time data pipelines and streaming applications. Kafka can handle high-volume, high-throughput, and low-latency data streams, making it ideal for real-time data processing.

Hadoop: Hadoop is an open-source framework used for storing and processing large datasets across distributed computing clusters. Hadoop provides a reliable, scalable, and fault-tolerant storage layer for big data.

Spark: Spark is an open-source big data processing engine used for processing large datasets. Spark provides an easy-to-use API and high-performance processing capabilities.

The benefits of using these software components are that they are widely adopted and have a large developer community, providing continuous support, updates, and improvements. These software components are also open-source, which makes them cost-effective.

The trade-off is that these software components can be complex to set up and manage. They require a certain level of expertise to work with effectively. Moreover, since these components are open-source, security vulnerabilities can be a concern.

Reliability, Scalability, and Maintainability

To ensure reliability, scalability, and maintainability of the designed system, the following techniques will be used:

* Microservices: The system will be designed using microservices architecture. Microservices will enable each service to work independently and in isolation, providing better fault isolation and easier maintenance.
* Containerization: Docker will be used to containerize the system components, making the system more scalable and portable. Containerization also enables faster deployment and rollbacks, making it easier to maintain the system.
* Infrastructure as Code (IaC): IaC will be used to automate the infrastructure deployment process, making it more reliable and less prone to human errors.

These techniques ensure that the system is reliable, scalable, and maintainable. The microservices architecture enables better fault isolation, making it easier to maintain the system. Containerization makes the system more scalable and portable, enabling faster deployment and rollbacks. IaC automates the infrastructure deployment process, reducing human errors, and making the process more reliable.

Data Security, Governance, and Protection

To ensure data security, governance, and protection, the following techniques will be used:

* Authentication and Authorization: Authentication and authorization mechanisms will be used to control access to the system and its data. This will prevent unauthorized access to the system and data.
* Data Encryption: Data encryption will be used to protect the confidentiality and integrity of the data. Encryption will ensure that the data can only be accessed by authorized users.
* Data Governance: Data governance policies will be implemented to ensure that the data is used ethically and in compliance with legal and regulatory requirements.

These techniques ensure that the data is secure, governed, and protected. Authentication and authorization mechanisms will ensure that only authorized users can access the system and its data, while data encryption will protect the confidentiality and integrity of the data. Data governance policies will ensure that the data is used ethically and in compliance with legal and regulatory requirements. All of these techniques are crucial for maintaining the security, governance, and protection of the data and the system as a whole.

In addition to these techniques, other measures such as regular backups and disaster recovery plans will also be implemented to ensure the availability and recoverability of the data in case of any unforeseen events or disasters. These measures will help to minimize the risk of data loss or downtime, and ensure that the system remains operational and reliable at all times.

Questions:

Which microservices will take care of data ingestion to my system?

For data ingestion, we can use microservices that are responsible for capturing and processing data from various sources such as APIs, databases, files, and message queues. Some examples of microservices that can handle data ingestion are:

1. Data collection service: This service is responsible for collecting data from various sources and feeding it into the system.
2. Data processing service: This service processes the raw data into a usable format by cleaning, transforming, and aggregating it.
3. Data validation service: This service validates the incoming data to ensure that it meets the expected format and data quality standards.
4. Data storage service: This service stores the ingested data into a database or data lake for future use.

The exact combination of microservices will depend on the specific requirements and architecture of the system. It is essential to design a modular and scalable system to handle increasing data volumes and changing data sources over time.

Which microservices will take care of data pre-processing and aggregation?

To handle data pre-processing and aggregation, we will use microservices such as Apache Spark and Apache Flink. Apache Spark provides a distributed computing framework for data processing and allows for parallelization of data processing tasks. Spark also provides various libraries for pre-processing data, such as Spark SQL and Spark Streaming. Apache Flink is another distributed computing system that can handle stream and batch data processing. Flink is known for its ability to handle real-time streaming data processing efficiently.

Which microservices will take care of data storage in my system?

One microservice that could take care of data storage in your system is a database management system such as MongoDB or PostgreSQL. This microservice would be responsible for storing the data ingested by the data ingestion microservice in a way that allows for efficient querying and retrieval. Additionally, this microservice could be configured to implement data replication and backup strategies to ensure data redundancy and fault tolerance. Another option could be a distributed file system like Hadoop Distributed File System (HDFS) or Amazon S3, which allows for scalable storage of large volumes of data.

Which microservices will take care of data delivery to the frontend machine learning application?

The microservices responsible for delivering data to the frontend machine learning application will depend on the specifics of the application. However, we could use Apache Kafka for real-time data streaming, or an API for retrieving processed data. Apache Kafka is a distributed event streaming platform that can handle high-throughput data streams. It can also handle real-time data processing and real-time analytics. APIs, on the other hand, provide a more structured way of delivering data to the frontend, and can be used for data delivery in a more controlled manner.

Which techniques will I use to implement reliability, scalability, and maintainability to my system?

To implement reliability, scalability, and maintainability, we will use techniques such as containerization, automation, and monitoring. Containerization allows for a more modular and scalable approach to deploying microservices. We will use Docker to create containers for each microservice, making them portable and easier to manage. Automation will also be used to streamline the deployment and management of microservices. We will use Infrastructure as Code (IaC) to automate the provisioning and configuration of our infrastructure. Monitoring will also be implemented to ensure the health of our system. We will use tools like Prometheus and Grafana to monitor system performance and detect and resolve issues before they become critical.

Which techniques will I use to ensure data security, governance, and protection?

To ensure data security, governance, and protection, we will implement access controls, encryption, and regular backups. Access controls will be used to limit access to sensitive data to authorized personnel only. Encryption will be used to protect data in transit and at rest. Regular backups will also be implemented to ensure that data can be recovered in case of a disaster. We will use tools like HashiCorp Vault to manage secrets and provide secure access to sensitive information. We will also follow security best practices and standards such as the OWASP Top 10 and ISO/IEC 27001.

Which docker images will I use to build the system and must they be modified?

The docker images we will use to build the system will depend on the specific microservices we choose to use. For example, we could use the official Apache Kafka and Apache Spark images from Docker Hub. We may also use other images for specific components such as databases, caching systems, and load balancers. We will modify these images as necessary to include any custom configurations and dependencies needed for our application.

Which data will I use for my project?

For our project, we will choose a data source that contains several data points and is time-referenced, i.e., containing a timestamp for each data point. A good starting point for finding such a data source is Kaggle, where numerous open sample datasets are available. We will choose a dataset that is relevant to our use case, and contains data that can be used for training and testing a machine learning model.

At which frequency will my system ingest (e.g. monthly), process, aggregate and deliver data (e.g. quarterly)?

The system will ingest data on a daily basis, process and aggregate the data on a weekly basis, and deliver the pre-processed data to the frontend machine learning application on a monthly basis. This frequency of data processing and delivery strikes a balance between keeping the machine learning model up-to-date while not putting undue stress on the system resources. The exact frequency can be adjusted based on the specific needs and requirements of the application.

## **4.2 Development phase/reflection phase**

**Hierzu kommen wir dann, wenn es soweit ist!**

In this phase you practically implement your data processing system. You set up a **Git Repository** which will con-tain all the code you generate during this project. You will create your data processing infrastructure as micro-services as it was drafted in the conception phase. To do that, you **deploy docker containers** which might also be modified for your system. You will not deploy to cloud services but locally to your machine for development purposes. If necessary, you **adapt your system to ensure reliability, scalability, maintainability, data secu-rity, governance, and protection**. After all microservices are running and communicating with each other, you **ingest the data** you chose in the conception phase to your system. Make sure that the data is ingested, pre-pro-cessed, and aggregated as expected. At the end of this phase, you have a working environment which you can use to reproducibly build your data processing infrastructure. Your code resides in a version-controlled Git repository which can also be used as a portfolio in itself and shown to potential future employers.

Throughout the process, online meetings and other channels provide the opportunity to profoundly discuss ideas and/or drafts and to get sufficient feedback, tips, and hints. **It is recommended to use these channels to avoid errors and to improve your work.** Once this is done, you can hand in your second phase for evaluation. Following a feedback from the tutor, your work on the final draft will continue in the third phase.

## **4.3 Finalization phase**

In this phase, you finalize your project. First, you finetune your code which builds your infrastructure. Revise the whole process and note where things went smoothly and where problems emerged.

* • Does your system fulfill the technical requirements?
* • What went wrong and why?
* • Is your system reliable, scalable, and maintainable?
* • What measures for data security, governance and protection can be added?
* • What could you do in the next project to improve your workflow?
* • What are the major steps you took and what are the three most valuable technical skills you learned dur-ing the project?
* • What are the three most valuable “soft” skills you learned during the project?

You should **discuss which strategy** you could pursue in order **to introduce a second data pipeline to your sys-tem** which is able to process real-time streaming data.

In the “finalization phase”, the online meetings and other channels also provide the opportunity to obtain suffi-cient feedback, tips, and hints before the finished product is finally handed in. **It is recommended to use these channels to avoid errors and to make improvements.** The finished product is submitted with the results from Phase 1 and Phase 2 and together with the materials mentioned above. In addition, it is mandatory to write an abstract which describes the solution of the task in terms of content and concept and presents a short breakdown (making of) of the technical approach in a sober and informative way. Following the submission of the third port-folio page, the tutor submits the final feedback which includes evaluation and scoring within six weeks.

# **4. Conclusion**

Delete!

# **References**

1. Kennedy K, Mac Namee B, Delany SJ, O’Sullivan M, Watson N. A window of opportunity: Assessing behavioural scoring. Expert Syst Appl. 2019; 40(4):1372–80.
2. Adams, M.N.: Perspectives on Data Mining. International Journal of Market Research 52(1), 11–19 (2018)
3. Asur, S., Huberman, B.A.: Predicting the Future with Social Media. In: ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. 1, pp. 492–499 (2018)
4. Bakshi, K.: Considerations for Big Data: Architecture and Approaches. In: Proceedings of the IEEE Aerospace Conference, pp. 1–7 (2012)
5. Cebr: Data equity, Unlocking the value of big data. in: SAS Reports, pp. 1–44 (2021)
6. Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J.M., Welton, C.: MAD Skills: New Analysis Practices for Big Data. Proceedings of the ACM VLDB Endowment 2(2), 1481–1492 (2019)
7. Cuzzocrea, A., Song, I., Davis, K.C.: Analytics over Large-Scale Multidimensional Data: The Big Data Revolution! In: Proceedings of the ACM International Workshop on Data Warehousing and OLAP, pp. 101–104 (2021)
8. Economist Intelligence Unit: The Deciding Factor: Big Data & Decision Making. In: Capgemini Reports, pp. 1–24 (2012)
9. Sanchez, D., Martin-Bautista, M.J., Blanco, I., Torre, C.: Text Knowledge Mining: An Al-ternative to Text Data Mining. In: IEEE International Conference on Data Mining Work-shops, pp. 664–672 (2008)
10. Serrat, O.: Social Network Analysis. Knowledge Network Solutions 28, 1–4 (2019)
11. Shen, Z., Wei, J., Sundaresan, N., Ma, K.L.: Visual Analysis of Massive Web Session Da-ta. In: Large Data Analysis and Visualization (LDAV), pp. 65–72 (2014)