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 to consider:

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.