**(DLMDSEDE02)**

# CONCEPTION PHASE

This Conception Phase serves as a base of the project as it defines the architecture for a real-time data backend that is capable of processing large data streams. Data consistency is ensured by the system through ingestion, aggregation, and preprocessing, supporting real-time consumption via a streaming report application. Key design considerations are highlighted to lay the groundwork for scalability, security, and maintainability.

## Software Components Familiarisation

Apache Kafka is chosen for data ingestion due to its high throughput, fault tolerance, and scalability. Kafka handles continuous data streams effectively and supports features like log retention and repayable streams, which make it superior to alternatives like Kinesis. Kafka integrates well with other big data tools, making it more flexible for large-scale systems. For real-time processing, Apache Spark is used due to its in-memory processing and advanced aggregation capabilities. Unlike Apache Flink, Spark supports both batch and stream processing, simplifying integration and system management.

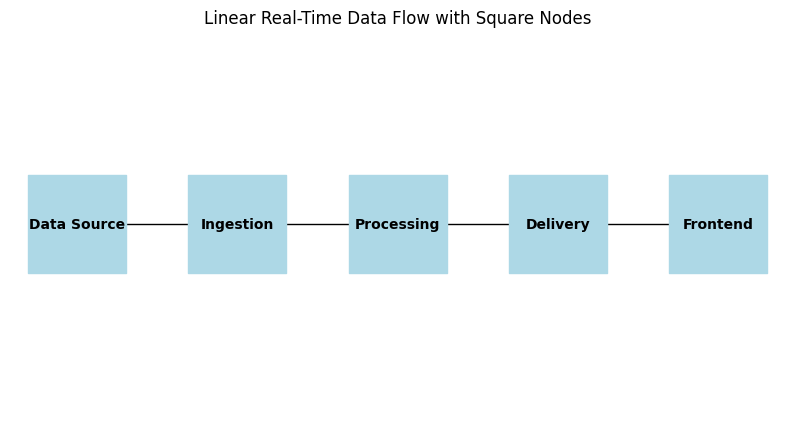


Figure 1: Real-time data flow

## Reliability, Scalability, and Maintainability

Kafka’s data replication and Spark’s checkpointing ensure system reliability. Kafka’s horizontal scaling through partitions and Spark’s scalability through executors make the architecture suitable for distributed environments. Infrastructure as Code tools like Terraform and Ansible automate deployment, ensuring consistency and ease of maintenance. Kafka’s scalability and ease of integration surpass Flink, which, although effective in stream processing, requires more complex configurations and does not handle data ingestion as seamlessly.

## Data Security and Governance

Encryption, RBAC, and GDPR compliance ensure data security. Kafka uses SSL/TLS for secure communication, while Apache Atlas provides governance through metadata and lineage tracking. Kafka’s open-source nature offers better control over data flow and easier integration with third-party tools than Kinesis.

## Containerisation and Microservices

Docker ensures isolated, scalable microservices. Pre-configured Docker images for Kafka and Spark, optimized with multi-stage builds, allow for efficient, lightweight deployment and fault isolation. Kafka’s horizontal scaling fits well with microservices, ensuring independent scaling and fault tolerance.

## Best-Practice Architecture Consultation

Incorporating industry best practices from LinkedIn (Kafka) and Netflix (Spark) validates the chosen tools’ scalability and reliability, ensuring they meet the demands of modern, high-throughput systems.

## Data Source Selection

Large, time-stamped datasets from Kaggle are used to simulate real-time data streams, ensuring the system can handle over 1,000,000 data points and test performance in real-time environments.

## Key Questions Addressed

* **Data Ingestion**: Kafka-based microservices ingest raw data streams efficiently.
* **Data Pre-Processing and Aggregation**: Spark jobs perform transformations and computations, enabling structured outputs for analytics.
* **Data Delivery**: Processed data is delivered to frontend applications via RESTful APIs or WebSocket services.
* **Reliability, Scalability, and Maintainability**: Achieved through data replication, horizontal scaling, and automated deployment.
* **Security and Governance**: Enforced via encryption, RBAC, and compliance with data protection standards.