Big Data Analytics: Streaming, Time Series, and Data Lakes

December 4, 2025

Streaming Big Data

- **Streaming data** refers to continuous, real-time data generated by sources such as sensors, logs, and user activity.
- Key challenges:
  - High throughput and low latency processing
  - Fault tolerance and scalability
  - Windowing and event time management
- Common frameworks:
  - Apache Kafka
  - Apache Spark Streaming

Apache Kafka

- **Kafka** is a distributed event streaming platform for high-throughput, fault-tolerant data pipelines.
- Core concepts:
  - **Producer**: Sends data to Kafka topics
  - **Consumer**: Reads data from topics
  - **Broker**: Kafka server that stores and serves data
  - **Topic**: Logical channel for data streams
- Features:
  - Horizontal scalability
  - Durable message storage
  - Real-time analytics and integration

Kafka Usage Examples

Example 1: Simple Producer

```
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers='localhost:9092')
producer.send('my-topic', b'Hello, Kafka!')
producer.flush()
```

Example 2: Simple Consumer

```
from kafka import KafkaConsumer

consumer = KafkaConsumer('my-topic', bootstrap_servers='localhost:9092')
for message in consumer:
    print(message.value)
```

Example 3: Real-Time Data Pipeline

- **Producers** send sensor data to Kafka topics.
- **Consumers** process and analyze data streams for monitoring or alerting.
- **Connectors** integrate Kafka with databases, data lakes, or analytics platforms.

Example 4: Stream Processing with Kafka + Spark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("KafkaSparkStreaming").getOrCreate()
df = spark.readStream.format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "my-topic") \
    .load()
df.selectExpr("CAST(value AS STRING)").writeStream.format("console").start().awaitTermination()
```

Time Series Databases

- **Time series databases (TSDBs)** are optimized for storing and querying time-stamped data.
- Use cases:
  - IoT sensor data
  - Financial market data
  - Server metrics and logs
- Key features:
  - Efficient storage for sequential data
  - Fast aggregation and downsampling
  - Native support for time-based queries

TimescaleDB

- **TimescaleDB** is a PostgreSQL extension for scalable time series data.
- Features:
  - Hypertables for partitioning data by time
  - Native SQL support
  - Compression and retention policies
  - Advanced analytics (window functions, gap filling)
- Example query:

```
SELECT time_bucket('1 hour', timestamp) AS hour, avg(value)
FROM metrics
GROUP BY hour;
```

What is a Data Warehouse?

- **Data warehouse** is a centralized system for storing, integrating, and analyzing structured data from multiple sources.
- Key characteristics:
  - Stores cleaned, processed, and organized data
  - Uses schema-on-write for consistency
  - Optimized for complex queries and reporting
  - Supports business intelligence and decision-making
- Common platforms:
  - Amazon Redshift
  - Google BigQuery
  - Snowflake

Schema-on-Write Explained

- **Schema-on-write** means data is structured and validated before being stored in the database.
- Data must conform to a predefined schema (table structure, data types) at ingestion time.
- Benefits:
  - Ensures data consistency and quality
  - Optimized for fast, reliable querying
  - Supports complex analytics and reporting
- Common in traditional data warehouses (e.g., Redshift, BigQuery, Snowflake)

- Example workflow:
    i. Define table schema
    ii. Clean and transform data
    iii. Load data into warehouse
- Drawback: Less flexibility for storing raw or semi-structured data

Schema-on-Write vs. Schema-on-Read

Aspect	Schema-on-Write	Schema-on-Read
When schema applied	At data ingestion	At query time
Data flexibility	Low (must match schema)	High (can store any format)
Data quality	High (validated before storage)	Variable (validated when queried)
Query performance	Fast, reliable	May require extra parsing/processing
Use cases	Data warehouses, BI	Data lakes, exploratory analytics

- **Schema-on-write**: Data is structured and validated before storage, ensuring consistency and fast queries.
- **Schema-on-read**: Data is stored in raw form; structure is applied when data is accessed, allowing flexibility for diverse data types.

Hive & BigQuery

Apache Hive

- **Hive** is a data warehouse system for querying and managing large datasets stored in Hadoop.
- Uses SQL-like language (HiveQL)
- Supports schema-on-read and batch processing
- Integrates with Hadoop ecosystem (HDFS, Spark)

Google BigQuery

- **BigQuery** is a serverless, highly scalable cloud data warehouse.
- Features:
  - Real-time analytics on massive datasets
  - Standard SQL support
  - Automatic scaling and high availability
  - Pay-as-you-go pricing

Data Lakes

- **Data lake** is a centralized repository for storing raw, unstructured, and structured data at scale.
- Key characteristics:
  - Stores data in native format (e.g., Parquet, ORC, CSV)
  - Supports batch and streaming ingestion
  - Enables advanced analytics and machine learning
- Common platforms:
  - Amazon S3
  - Azure Data Lake Storage
  - Google Cloud Storage

Data Lake vs. Data Warehouse

Feature	Data Lake	Data Warehouse
Data type	Raw, unstructured, any	Structured, processed
Schema	Schema-on-read	Schema-on-write
Cost	Lower (storage)	Higher (compute)
Use cases	ML, analytics, archiving	BI, reporting