Real-Time Information Processing and Visualization

Subjects: Computer Science, Software Engineering

Contributor: Jose Machado

The processing of information in real-time (through the processing of complex events) has become an essential task for the optimal functioning of manufacturing plants. Only in this way can artificial intelligence, data extraction, and even business intelligence techniques be applied, and the data produced daily be used in a beneficent way, enhancing automation processes and improving service delivery.

Keywords: real time information processing ; Complex Event Processing (CEP) ; data visualization ; Power BI ; Kafka ; Spark ; Tableau

---

1. Software of Real-Time Information Processing and Visualization

This section presents a list of software to be analyzed in this paper. These tools were selected using the methods described in the previous section. For each tool, a general description of the software and its main components is presented, and the advantages and disadvantages of the tool are identified. These tools were divided into two main categories: the first one is related to event brokers, while the second category is related to Business Intelligence (BI) and the data analytics process.

2. Event Broker

2.1 Apache Flume

Apache Flume is a streaming data-based framework that allows the collection and aggregation of data, and to move large amounts of data in a distributed environment [1]. This tool was originally developed by Cloudera, with its ownership moving in 2012 to the Apache Software Foundation [2].

The main purpose of Flume is to ingest event data into the HDFS (Hadoop Distributed File System) in a simple and automated way. However, it can be used for various tasks, including data transport [3].

Flume supports several data reading mechanisms, including Avro, Thrift, and Syslog [4].

2.2 Components

Apache Flume offers multiple components: Source, Channel, Sink, Interceptors, Channel Selectors, and Sink Processors [4].

The first of the three base components to be defined is Source, responsible for data input. It is necessary to expose that it has a major limitation, only extracting and supporting unstructured data (events) [2][4].

The Channel component is the intermediate component that handles data storage in the transition from the Source to the Sink. The operation performed resembles a queue [2][4].

To conclude the base components, the Sink component is the component responsible for sending data to the destination. Usually, the destination is the HDFS, but other options, like HBase, Hive, and Elastic Search, are supported, as well [2][4].

Regarding additional components, the Interceptors are used to modify or review the events transmitted between the source and the channel [5].

The Channel Selectors are useful for identifying which channel to select for data transmission when there are multiple channels. This component is typified according to the type of channel. The Default Channel Selectors, often known as Replicating Channel Selectors, are responsible for replicating all events on each of Flume’s channels. Multiplexing

Channel Selectors use the address provided in the event header to decide the channel to which the event should be sent [4].

Finally, the Sink Processors are responsible for calling a specific selector from a selected group of sinks. To do this, fail-over paths are created for the sinks. These components are commonly used for load balancing events on multiple collectors [4][5].

Figure 1 illustrates the architecture that is obtained when all of these components are in use.

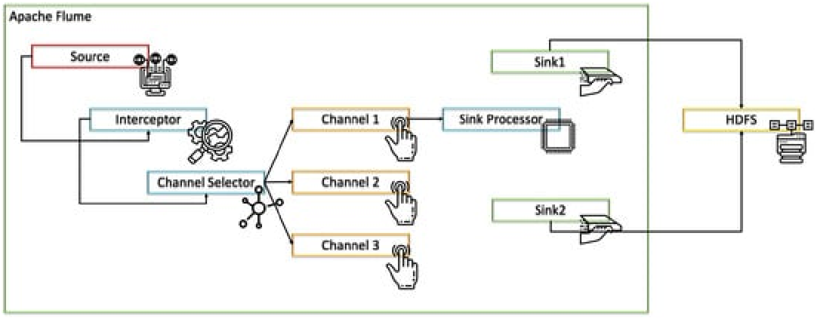

Figure 1. Flume architecture. Adapted from Reference [4].

3. Apache Sqoop

Sqoop is a framework that allows data to be moved and removed from any relational database management system in Hadoop. It is a data management tool built by Apache Software Foundation on Hadoop [6]. In short, the main purpose is to import data from relational databases into Hadoop (HDFS) and export data from the Hadoop file system to relational databases [7].

3.1. Components:

Sqoop is a tool that is designed for the data transfers between (Relational Database Management Systems) RDBMSs and the Hadoop ecosystem. Therefore, it is based on two major components: the connectors that allow Sqoop to overcome the differences in the SQL dialects supported by the different relational databases along with providing the optimized data transfer; plus, the Drivers meaning the JDBC Drive that is the standard Java API for accessing RDBMS and some data warehouses [8].

Thus, the basic architecture for Apache Sqoop is presented in Figure 2.

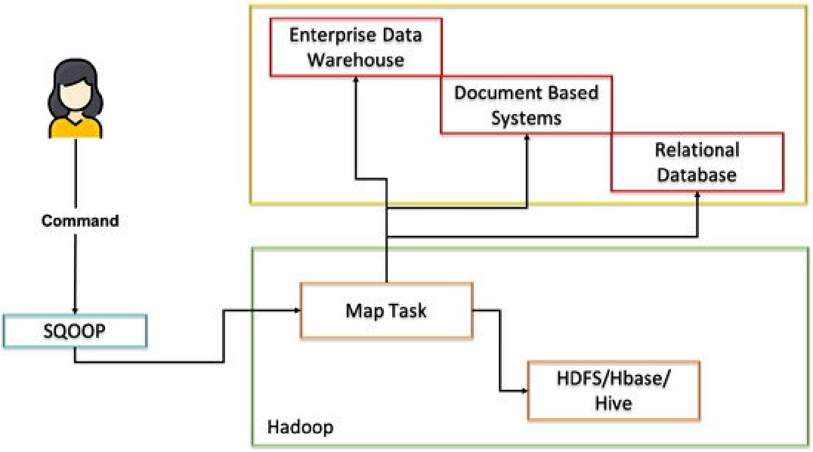

Figure 2. Sqoop architecture.

As can be observed, this tool only makes sense to be used when framed in Hadoop ecosystems.

4. Solace PubSub+

Solace, initially called Solace Systems, is a middleware company based in Canada. This company is dedicated to the manufacture and sale of message-oriented devices and software for routing information. In 2001, Craig Betts founded Solace with the purpose of incorporating the messaging system into hardware [9].

Their flagship product, PubSub+, is an unified message broker that supports publishing/subscribing, queuing, requesting/responding, and streaming using open APIs and protocols through hybrid cloud and Internet of Things (IoT) environments [10].

This software is divided into 5 layers that will be described below:

- **PubSub+ Event Broker**: This layer incorporates three other sub-layers that are described below. Even so, the PubSub+ event brokers have the ability of loading an event mesh.
    - **PubSub+ Event Broker- Software**: The main function of the Solace software sub-layer is to efficiently transport information in the form of events. This transport can be between applications, IoT devices, and user interfaces, all of which can be hosted locally or in a cloud. This software allows the use of various communication protocols, such as open protocols, like Advanced Message Queuing Protocol (AMQP), Java Message Service (JMS), Message Queuing Telemetry Transport (MQTT), Representational State Transfer (REST), and WebSocket. There are two versions of this software, one free (Standard) with support of up to 1000 client connections, and another that offers high performance (Enterprise), with a scale of up to 200,000 client connections.
    - **PubSub+ Event Broker- Appliance**: PubSub+ Appliances have three characteristics that define them exclusively. They are specially designed with high-speed FPGAs and network processors that support extremely low and predictable latency. They offer built-in redundancy and can even continuously replicate all messages to waiting locations.
    - **PubSub+ Event Broker- Cloud**: Solace's cloud service makes software event brokers available as a service. Only in this way can the needs of the software be met in a short period of time, and scale on-demand to any level.
- **PubSub+ Event Mesh**: An event mesh is a layer that dynamically routes events from one application to any other.
- **PubSub+ Streaming APIs and Integrations**: They provide a variety of on and off-ramps, such as the protocols already listed and proprietary APIs for messaging, in order to link old and modern applications and connectors to technologies, like Kafka.
- **PubSub+ Event Portal**: The PubSub+ Event Portal is an event managing tool-presented through the User Interface (UI) available on the Web—which allows for the discovery, constructing, visualizing, sharing, and managing of several aspects of the Event-Driven Architecture (EDA). Here, major elements of the Event portal are described, as well as a general view of its tools. Furthermore, some of the characteristics are discussed, such as the possibility of execution-time EDA, support for Kafka-native objects, event sharing, version control, REST API, AsyncAPI, and other essential characteristics. In addition, tools are provided for building, describing, and discovering events within the system, but also the establishing of connections between applications and events, making it easier to develop event-oriented applications and microservices [11].
- **PubSub+ Platform Security**: The security platform allows for message architectures that obtain a consistent multi-protocol authentication of a client, plus security clearance management in a company environment, all of it integrated with company authentication services while using a minimum amount of components.

These layers are organized as illustrated in Figure 3.

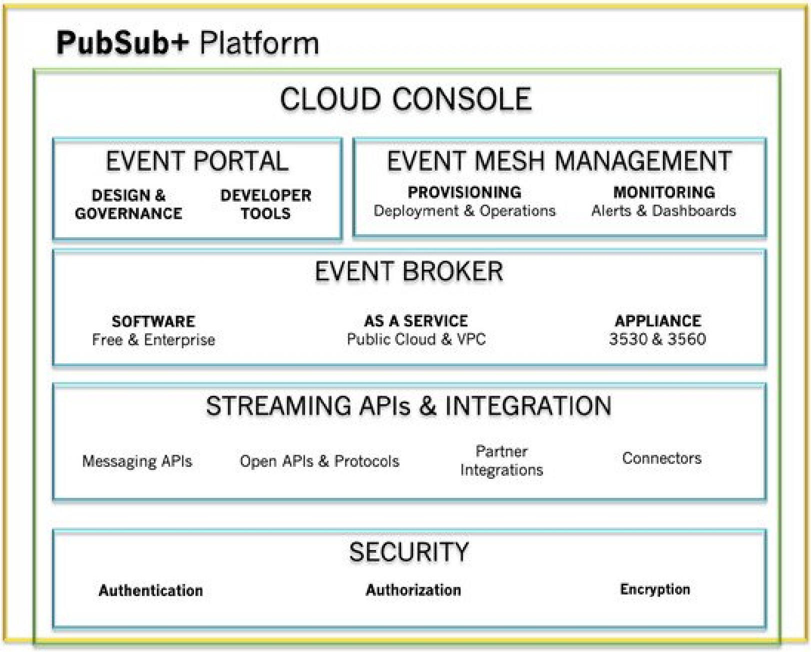

Figure 3. Layout of the internal layers of PubSub+. Adapted from Reference [10].

4.1. Components

Sending messages is analogous to worldwide known postal services. The term *message*, in the technological context, refers to the technology that allows computer systems to share information without the need for direct links or knowledge of each other's location. In the most basic operations, the sending of messages requires 5 components:

- **Publisher**: the entity that sends or publishes the message (also called a producer);
- **Message**: what the publisher wants to say to the subscriber. Messages often contain events, but can also carry queries, commands, and other information;
- **Subscriber**: the ultimate receiver of the message (also called a consumer);
- **Topic**: used when the message is intended to be consumed by more than one subscriber;
- **Queue**: used when the message is intended to be consumed by at most one subscriber.

These components are architecturally organized as illustrated in Figure 4.

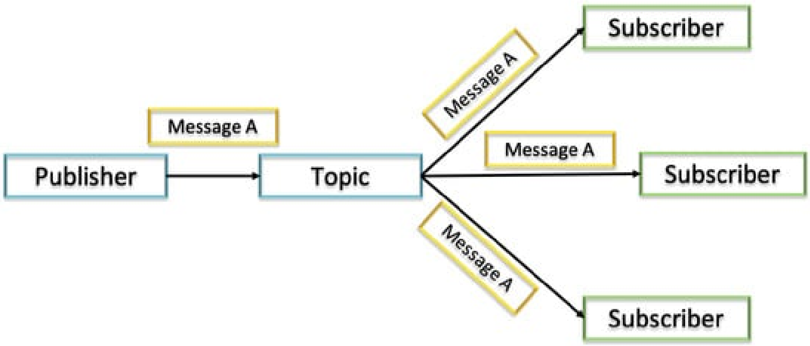

Figure 4. Internal architecture of solace messaging. Adapted from Reference [9].

5. Apache Kafka

Apache Kafka is an open-source platform, developed in Java and Scala, for distributed streaming processing. It was originally developed by LinkedIn in order to solve the processing of large amounts of data with low latency, as well as deal with the processing of data in real-time [12]. It was later donated to Apache Software Foundation, its current owner [13].

Apache Kafka is an event streaming platform, aiming to tackle the challenges in implementing publish/subscribing approaches. With its durable and distributed file system, it is able to support millions of events per second, handling data producers and consumers. Kafka's flexibility and performance enables multiple use cases, from distributing streaming and metrics logging, to the development of event-driven applications, thus being a key component for a data-driven ecosystem [14][15].

This platform runs as a cluster and can contain multiple data centers, allowing efficient communication between data producers and consumers using message-based topics. Each message is made up of its key and value, and a date/time stamp [12][16].

Kafka constitute of four main APIs, the Producer API, the Consumer API, the Streams API, and the Connector API. While the Producer API enables applications to publish messages to Kafka topics, the Consumer API enables the subscription to one or more topics. The Streams API allows the processing of data in Kafka using stream processing paradigms. These two APIs make subscribers able to process the received messages with stream procedures, then sending to other topics. Finally, the Connector API connects applications or data systems to Kafka topics, providing flexibility in building and managing producers and consumers, as well as providing reusable connections between them [17][18].

In Figure 5, the basic architecture of Kafka can be observed.

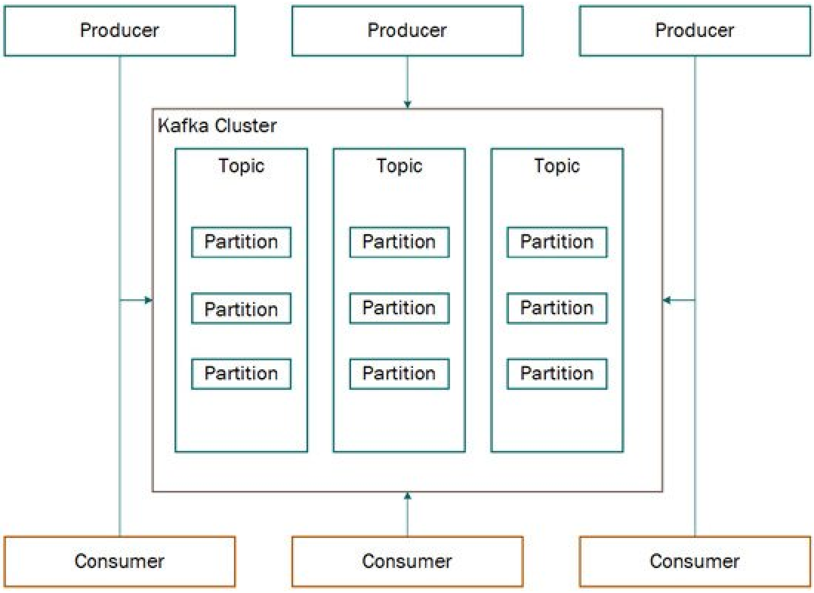

Figure 5. Apache Kafka basic architecture.

5.1. Components

Apache Kafka offers eight different components: Topics, Producers, Consumers, Brokers, Partitions, Replicas, Leaders, and Followers. Hence, the concepts of each of them are presented [15][16]:

- **Topics**: A topic is a category where records are published. Topics can be compared to database tables, where multiple subscribers can subscribe to the same topic [15][16].
- **Producers**: Producers create new messages, and publish them to a specific topic. They are able to assign which partition within the topic the message will go to, by using a message key [15][16].
- **Consumers**: Consumers label themselves with a consumer group name, and each record published to a topic is delivered to one consumer instance within each subscribing consumer group [15][16].
- **Brokers**: A Kafka server is called a Kafka broker, receiving messages from producers, assigning offsets to them, and committing the messages to storage. Kafka can work as a cluster, employing multiple brokers to distribute and replicate the messages [15][16].
- **Partitions**: A partition can be described as an "ordered, immutable sequence of records that is continually appended to a structured commit log" [14]. The Kafka cluster will divide topics into partitions and replicate them to brokers [15][16].
- **Replicas**: Topic replication is essential to design resilient and highly available Kafka deployments [15][16].
- **Leaders**: Each partition is owned by a single broker, making sure that the follower partitions keep their records synchronized [15][16].

- **Followers**: Followers serve as replicas for a partition, for, in the event that the leader disconnects, a follower is then promoted to leader [15][16].

Figure 6 showcases a perspective of the relationships between these components [17].

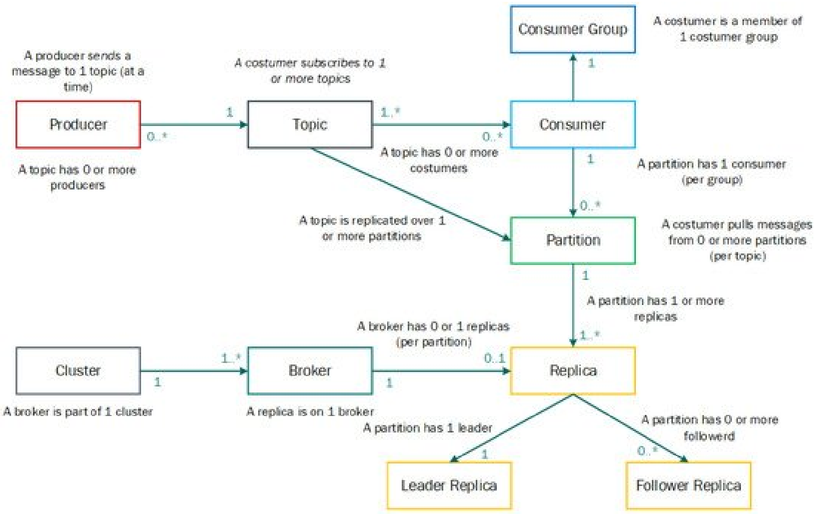
Consumer Group
A costumer is a member of 1 costumer group
A producer sends a message to 1 topic (at a time)
A costumer subscribes to 1 or more topics
Producer
Topic
Consumer
A topic has 0 or more producers
A topic has 0 or more costumers
A partition has 1 consumer (per group)
A topic is replicated over 1 or more partitions
Partition
A costumer pulls messages from 0 or more partitions (per topic)
A partition has 1 or more replicas
A broker has 0 or 1 replicas (per partition)
Cluster
Broker
Replica
A broker is part of 1 cluster
A replica is on 1 broker
A partition has 1 leader
A partition has 0 or more followerd
Leader Replica
Follower Replica
1
0..*
1..*
0..1

Figure 6. Relation between Kafka components. Numbers on figure represent cardinality, with 0.* representing the discrete interval from zero to many, and 1.* representing the discrete interval from one to many. Adapted from [17].

6. Spark

Apache Spark is a Big Data tool that aims to process large datasets in a parallel and distributed way. It extends the MapReduce programming model popularized by Apache Hadoop, enabling the development of large data processing applications. In addition to the extended programming model, Spark also performs much better than Hadoop, reaching in some cases almost 100x higher performance [16].

Another great advantage of Spark is that all components work integrated into the tool itself, such as Spark Streaming, Spark SQL, and GraphX, unlike Hadoop, where it is necessary to use tools that integrate with it but are distributed separately, like Apache Hive. Besides, another important aspect is that it allows programming in three languages: Java, Scala, and Python [19].

6.1. Components:

Spark has several components for different types of processing, all built on Spark Core, which is the component that provides basic processing functions, such as a map, reduce, filter, and collect. Among these, we highlight those present in Figure 7:

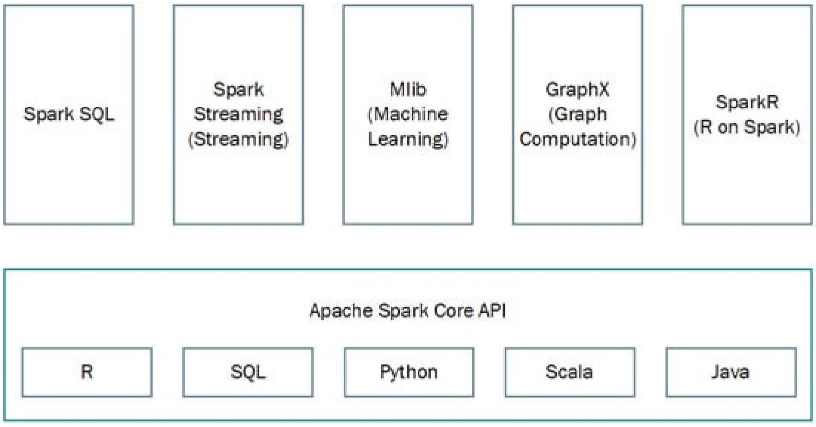

Figure 7. Apache Kafka main components.

- **Spark Core**: Spark integrates a RDD (resilient distributed dataset), handling the partitioning of data across all nodes in a cluster. Two operations can be performed on RDDs, namely Transformations and Actions [20];

- **Spark SQL**: Allows applications to access Spark’s data through a Java Database Connectivity (JDBC) API. That way, SQL queries can be performed on the data, also allowing the usage of BI and data visualization tools [21];

- **Spark Streaming**: It is used to process streaming data in real-time, based on micro-batch computing;

- **MLlib** (Machine Learning Library): It is Apache Spark’s scalable machine learning library. MLib contains a suite of algorithms and utilities that interoperate with programming languages and most scientific computing environments [22];

- **GraphX**: GraphX is a Spark components for graphs and graph-parallel computation. GraphX extends the Spark RDD by introducing graph abstractions, as well as including a “collection of graph algorithms and builders to simplify graph analytics tasks” [23];

- **SparkR**: SparkR is an R package that provides the ability to use Apache Spark from R, as well as a data frame implementation that supports selections, filtering, and aggregation. Furthermore, SparkR enables distributed machine learning on R through MLlib [24].

References

1. The Apache Software Foundation. Welcome to Apache Flume. 2020. Available online: (accessed on 20 October 2020).
2. Hoffman, S. Apache Flume: Distributed Log Collection for Hadoop; Packt Publishing Ltd.: Birmingham, UK, 2013.
3. Vohra, D. Apache flume. In Practical Hadoop Ecosystem; Springer: Berlin/Heidelberg, Germany, 2016; pp. 287–300.
4. The Apache Software Foundation. Flume 1.9.0 User Guide. Available online: (accessed on 20 October 2020).
5. Srinivasa, K.; Siddesh, G.; Srinidhi, H. Apache Flume. In Network Data Analytics; Springer: Berlin/Heidelberg, Germany, 2018; pp. 95–107.
6. The Apache Software Foundation. Sqoop. 2019. Available online: (accessed on 20 October 2020).
7. Vohra, D. Using apache sqoop. In Pro Docker; Springer: Berlin/Heidelberg, Germany, 2016; pp. 151–183.
8. Arvind. Apache Sqoop Graduates from Incubator. 2012. Available online: (accessed on 20 October 2020).
9. What is Solace PubSub+ Platform? Available online: (accessed on 20 October 2020).
10. PubSub+ Platform. Available online: (accessed on 20 October 2020).
11. PubSub+ Event Portal. Available online: (accessed on 20 October 2020).
12. Apache Kafka-Introduction. Available online: (accessed on 20 October 2020).
13. Garg, N. Apache Kafka; Packt Publishing: Birmingham, UK, 2013.
14. Confluent. What is Complex Event Processing? Guide to CEP. Available online: (accessed on 20 October 2020).
15. Shapira, G.; Palino, T.; Sivaram, R.; Narkhede, N. Kafka: The Definitive Guide; O’Reilly Media, Incorporated: Sebastopol, CA, USA, 2017.
16. Confluent Inc. Introduction to Kafka. Available online: (accessed on 20 October 2020).
17. Carter, M. Apache Kafka Architecture: A Complete Guide. Available online: (accessed on 20 October 2020).
18. Chellappan, S.; Ganesan, D. Practical Apache Spark: Using the Scala API; Apress: Berkeley, CA, USA, 2018.
19. Frampton, M. Mastering Apache Spark; Packt Publishing: Birmingham, UK, 2015.
20. Gour, R. Apache Spark Ecosystem—Complete Spark Components Guide. 2018. Available online: (accessed on 20 October 2020).
21. Penchikala, P. Big Data Processing with Apache Spark—Part 1: Introduction. 2015. Available online: (accessed on 20 October 2020).
22. The Apache Software Foundation. MLlib|Apache Spark. Available online: (accessed on 20 October 2020).
23. The Apache Software Foundation. GraphX—Spark 3.0.2 Documentation. Available online: (accessed on 20 October 2020).
24. The Apache Software Foundation. SparkR (R on Spark). Available online: (accessed on 26 February 2021).

Retrieved from https://encyclopedia.pub/entry/history/show/25423