Big Data to Enable Global Disruption of the Grapevine-powered Industries

D3.2 - Data Ingestion & Integration Components

DELIVERABLE NUMBER	D3.2
DELIVERABLE TITLE	Data Ingestion & Integration Components
RESPONSIBLE AUTHOR	Pythagoras Karampiperis (Agroknow)

Co-funded by the Horizon 2020
Framework Programme of the European Union

GRANT AGREEMENT N.	780751
PROJECT ACRONYM	BigDataGrapes
PROJECT FULL NAME	Big Data to Enable Global Disruption of the Grapevine-powered industries
STARTING DATE (DUR.)	01/01/2018 (36 months)
ENDING DATE	31/12/2020
PROJECT WEBSITE	http://www.bigdatagrapes.eu/
COORDINATOR	Pythagoras Karampiperis
ADDRESS	110 Pentelis Str., Marousi, GR15126, Greece
REPLY TO	pythk@agroknow.com
PHONE	+30 210 6897 905
EU PROJECT OFFICER	Mr. Riku Leppanen
WORKPACKAGE N. | TITLE	WP3 | Data & Semantics Layer
WORKPACKAGE LEADER	Agroknow
DELIVERABLE N. | TITLE	D3.2 | Data Ingestion & Integration Components
RESPONSIBLE AUTHOR	Pythagoras Karampiperis (Agroknow)
REPLY TO	pythk@agroknow.com
DOCUMENT URL	http://www.bigdatagrapes.eu/
DATE OF DELIVERY (CONTRACTUAL)	30 September 2018 (M9)
DATE OF DELIVERY (SUBMITTED)	25 September 2018 (M9)
VERSION | STATUS	1.0 | Final
NATURE	Report (R)
DISSEMINATION LEVEL	Public (PU)
AUTHORS (PARTNER)	Pythagoras Karampiperis (Agroknow)
CONTRIBUTORS	Panagiotis Zervas (Agroknow), Sotiris Konstantinidis (Agroknow), Antonis Koukourikos (Agroknow)
REVIEWER	Florian Schlenz (GEOCLEDIAN)

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	Initial Table of Contents	20/08/2018	Pythagoras Karampiperis (Agroknow), Panagiotis Zervas (Agroknow), Sotiris Konstantinidis (Agroknow), Antonis Koukourikos (Agroknow)
0.4	Sections 2 and 3	24/08/2018	Pythagoras Karampiperis (Agroknow), Panagiotis Zervas (Agroknow), Sotiris Konstantinidis (Agroknow), Antonis Koukourikos (Agroknow)
0.6	Section 4	02/09/2018	Pythagoras Karampiperis (Agroknow), Panagiotis Zervas (Agroknow), Sotiris Konstantinidis (Agroknow), Antonis Koukourikos (Agroknow)
0.8	Section 1 and 5	18/9/2018	Pythagoras Karampiperis (Agroknow), Panagiotis Zervas (Agroknow), Sotiris Konstantinidis (Agroknow), Antonis Koukourikos (Agroknow)
0.9	Internal Review	21/09/2018	Florian Schlenz (GEOCLEDIAN)
1.0	Final edits after internal review	25/09/2018	Pythagoras Karampiperis (Agroknow), Panagiotis Zervas (Agroknow), Sotiris Konstantinidis (Agroknow), Antonis Koukourikos (Agroknow)

PARTICIPANTS		CONTACT
Agroknow IKE (Agroknow, Greece)		Pythagoras Karampiperis Email: pythk@agroknow.com
Ontotext AD (ONTOTEXT, Bulgaria)		Todor Primov Email: todor.primov@ontotext.com
Consiglio Nazionale DelleRicherche (CNR, Italy)		Raffaele Perego Email: raffaele.perego@isti.cnr.it
Katholieke Universiteit Leuven (KULeuven, Belgium)		Katrien Verbert Email: katrien.verbert@cs.kuleuven.be
Geocledian GmbH (GEOCLEDIAN, Germany)		Stefan Scherer Email: stefan.scherer@geocledian.com
Institut National de la Recherché Agronomique (INRA, France)		Pascal Neveu Email: pascal.neveu@inra.fr
Agricultural University of Athens (AUA, Greece)		Katerina Biniari Email: kbiniari@aua.gr
Abaco SpA (ABACO, Italy)		Simone Parisi Email: s.parisi@abacogroup.eu
APIGAIA (APIGEA, Greece)		Eleni Foufa Email: Foufa-e @apigea.com

ACRONYMS LIST

BDG	BigDataGrapes
BDE	BigDataEurope
HDFS	Hadoop Distributed File System
RDBM	Relational Database Management System
OLTP	Online Transaction Processing

EXECUTIVE SUMMARY

This accompanying document for deliverable D3.2 Data Ingestion & Integration Components describes the mechanisms and tools that will be used in the BigDataGrapes platform to ingest data of different nature from multiple sources. Also, the document describes the tools that will be used for data integration across the different BigDataGrapes platform layers, as well as for long-term storage and preservation of data.

The document first introduces the big picture of the architecture of the BDG platform and where the ingestion components are positioned. Afterwards, the document describes the different nature of data, and which technologies can be used to facilitate the ingestion process.

Then, the data fusion aspect is described, focusing on how the data will be made available across the BigDataGrapes platform, and how the different BigDataGrapes components will communicate with each other, in an effective and fault-tolerant way.

Moreover, the document provides documentation links for all the described technologies along with links to tools that facilitate their setup & maintenance. Finally, the document includes links that point to the dockerized versions of the respective tools, as provided by the BigDataEurope (BDE, https://www.big-data-europe.eu/) Project, which is starting point regarding the technical solutions that BigDataGrapes will built upon (as it has ben described in details in the BigDataGrapes DoA).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The BigDataGrapes platform aspires to provide components that go beyond the state-of-the-art on various stages of the management, processing, and usage of grapevine-related big data assets thus making it easier for grapevine-powered industries to take important business decisions. The platform employs the necessary components for carrying out rigorous analytics processes on complex and heterogeneous data helping companies and organizations in the sector to evolve methods, standards and processes based on insights extracted from their data.

For this purpose, the BigDataGrapes software stack has been designed and built using core technologies and frameworks for efficient processing of large datasets, such as Apache Spark and Apache Hadoop, making sure that the execution environment and methodology retain scalability and efficient use of the computational resources available. The distributed execution paradigm serves as the basis for efficiently solving the equally urgent challenge of Heterogeneity and Scalability in the context of Big Data processing.

In this first version of the deliverable we focus on the ingestion layer and the integration components. As is depicted in the next figure, the ingestion layer allows the BigDataGrapes software stack to communicate with the outside world, and more specifically it imports heterogenous data to be processed and analyzed by the next layers of the software stack. The integration components are crucial for ensuring the smooth interaction among the different layers of the software stack and ensure the sustainability of the software stack over time.

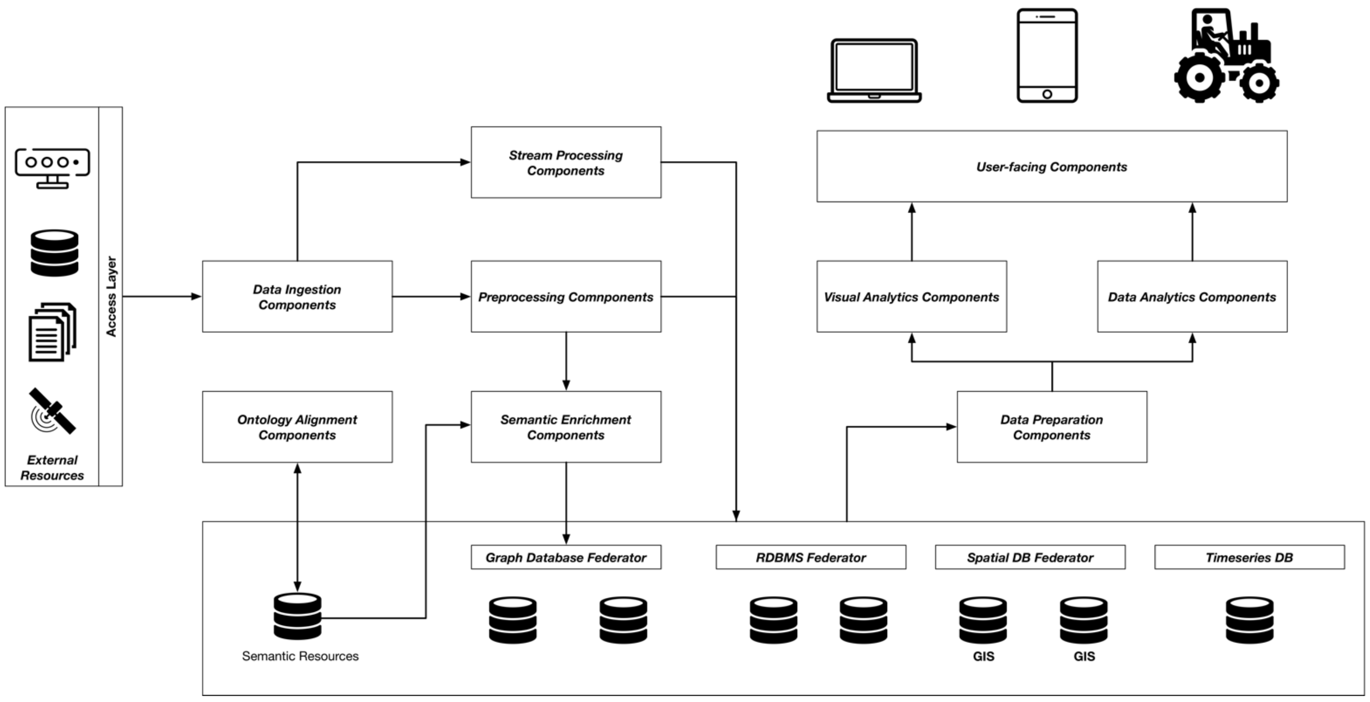
External Resources
Access Layer
Data Ingestion Components
Stream Processing Components
Preprocessing Comnponents
Semantic Enrichment Components
Ontology Alignment Components
User-facing Components
Visual Analytics Components
Data Analytics Components
Data Preparation Components
Graph Database Federator
RDBMS Federator
Spatial DB Federator
Timeseries DB
Semantic Resources
GIS
GIS

Figure 1. BigDataGrapes Architecture

The rest of this document is organized as follows: Section 2 highlights the proposed ingestion technologies used in the BDG software stack. Section 3 describes the integration components that will accommodate the interaction among the different components of the BDG. It also describes the long-term storage available options. Section 4 provides references to documentation and tools for each described technology, along with their dockerized version.

Section 3 describes how to set up and run the current version of the platform. Section 4 exhibits two demonstrators that process raster data using the BDG platform. Finally, Section 5 concludes this document.

2. DATA INGESTION

Data ingestion is the first step for building data pipelines and also one of the toughest tasks in Big Data processing. Big Data Ingestion involves connecting to several data sources, extracting the data, and detecting the changed data. It's about moving data from where it is originated, into a system where it can be stored, processed and analysed. Furthermore, these several sources exist in different formats such as: Images, OLTP data from RDBMS, CSV and JSON files, etc. Therefore, a common challenge faced at this first phase is to ingest data at a reasonable speed and further process it efficiently so that data can be properly analysed to improve business decisions.

In the following figure, which depicts the logical structure of the BigDataGrapes system, the pivotal role of the Data Ingestion components and the need to select mature and/ or develop custom components so as to smoothly import data from sources of different nature is apparent.

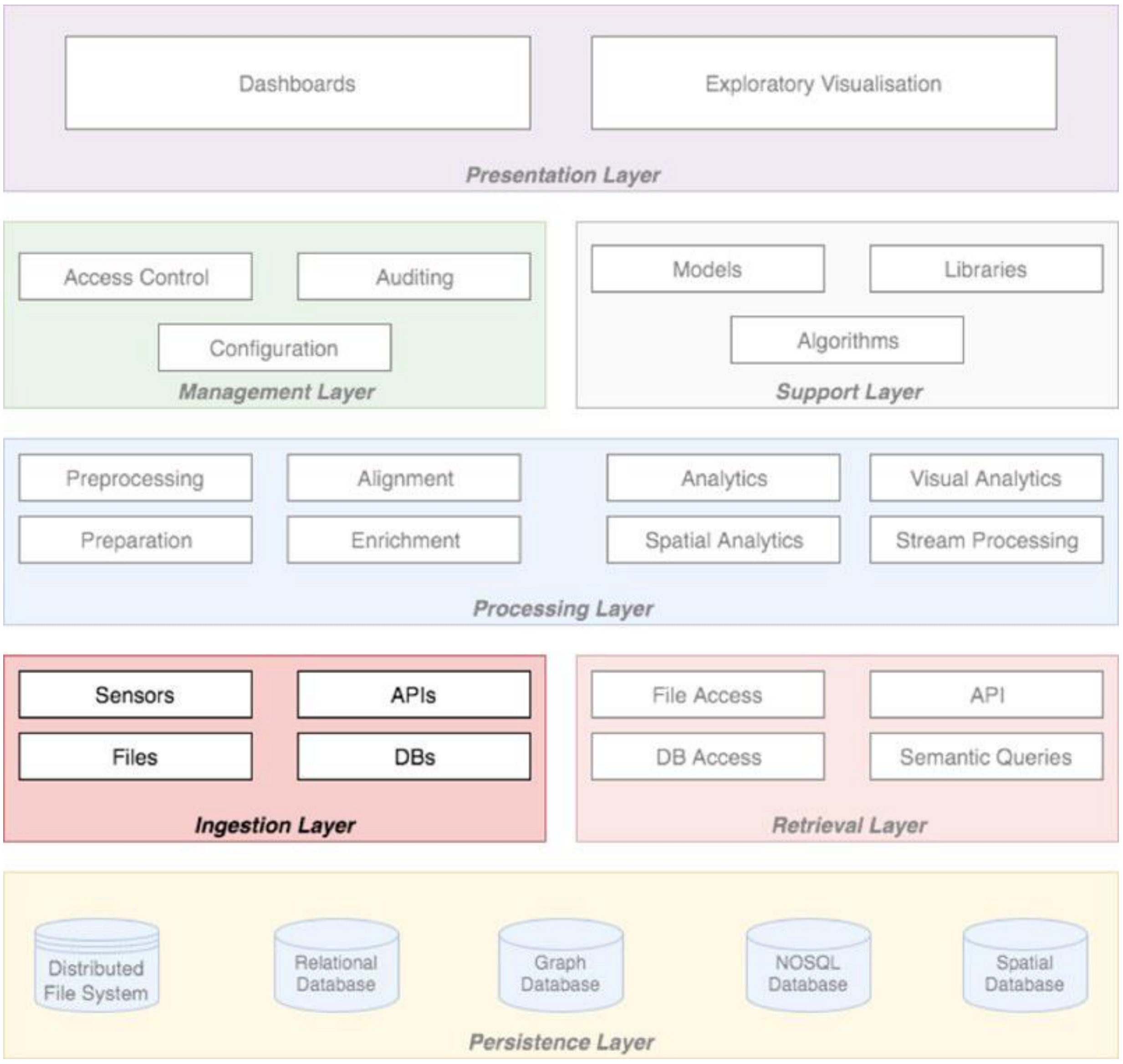

Figure 2. BigDataGrapes Architecture Layers

Data can be streamed in real time or ingested in batches. When data is ingested in real time then as soon as data arrives it is ingested immediately. When data is ingested in batches, data items are ingested in some chunks at a periodic interval of time. Effective Data Ingestion processes begins by prioritizing data sources, validating individual files and routing data items to the correct destination.

2.1 BATCH DATA

Batch data ingestion means the pulling of data in discrete chunks at periodic intervals of time. Usually, batch data sources are RDBM's or filesystems that store the data in various format like CSV, JSON or even images. The sources can be diverse in terms of the nature of the data as well as the totally different file formats.

Therefore, the ingestion component should be able to connect to multiple sources (in meaningful intervals) and pull the data regardless of their size. Moreover, the ingestion component should perform basic sanitization tasks, like deduplication and model validation, to ensure the integrity of the data that are ingested to the overall system.

The diverse nature of batch data sources makes the development of custom ingestion components necessary, that will be customizable to satisfy the different requirements that different sources pose.

2.2 STREAM DATA

The main difference between batch and stream data is the fact that stream data are continuously generated by multiple sources and are emitted as records in small sizes. Stream data are processed sequentially and incrementally.

Therefore, the ingestion component should guarantee data handling, in the sense that an efficient strategy for data loss or late arrived data should be applied. Also, contrary to the batch data case, in stream data the sources push data to the system, therefore, the ingestion component should be able to handle efficiently simultaneous requests from different sources.

Apache Flume[1] is a tool which has been designed specifically for ingesting stream data. Flume is distributed in nature, and its flexible architecture makes it a robust solution. Also, Flume provides a tunable fault-tolerant mechanism that can be customized to satisfy the different requirements of different sources. Its distributed nature encapsulates a variety of failover and recovery mechanisms.

The following concepts summarize the core characteristics of Flume:

- Event: The unit of data that is transported through Flume starting from the data source to its final destination.
- Agent: A process that implements the different flume components. An agent can receive, store and send events to the next destination.
- Client: An interface located between the data source and Flume. The data source pushes the data through that interface, in order to be delivered to a Flume agent.
- Channel: In the context of Flume, a channel is a transient storage component for events. An event stays in the channel until a sink removes it. Such a channel could be a messaging system, a schema-less database or even an RDBMS.
- Sink: Sinks remove events from channels and drains them to other channels, agents or to its final destination.

[1] https://flume.apache.org/

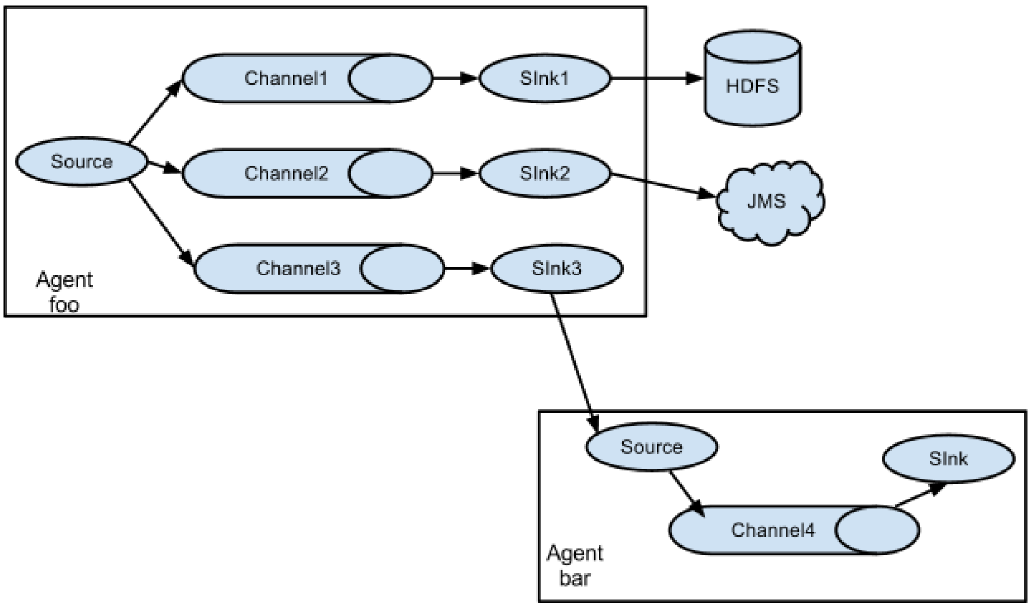

Figure 3. Exemplary Flume data flow
(source: https://flume.apache.org/FlumeUserGuide.html, Sep 2018)

Flume implements a channel-based transaction protocol to ensure event delivery. Whenever an event is about to move from one agent to another, both agents initiate a transaction. The sending agent initiates first its transaction and sends the event to the receiving agent.

Afterwards, the receiving agent initiates its transaction, receives the event, applies its logic to the event, puts it to the next channel and commits the transaction.

Upon committing its transaction, the receiving agent sends a success indication to the sending agent, which in turn commits its transaction. The next figure depicts the communication flow between the two agents.

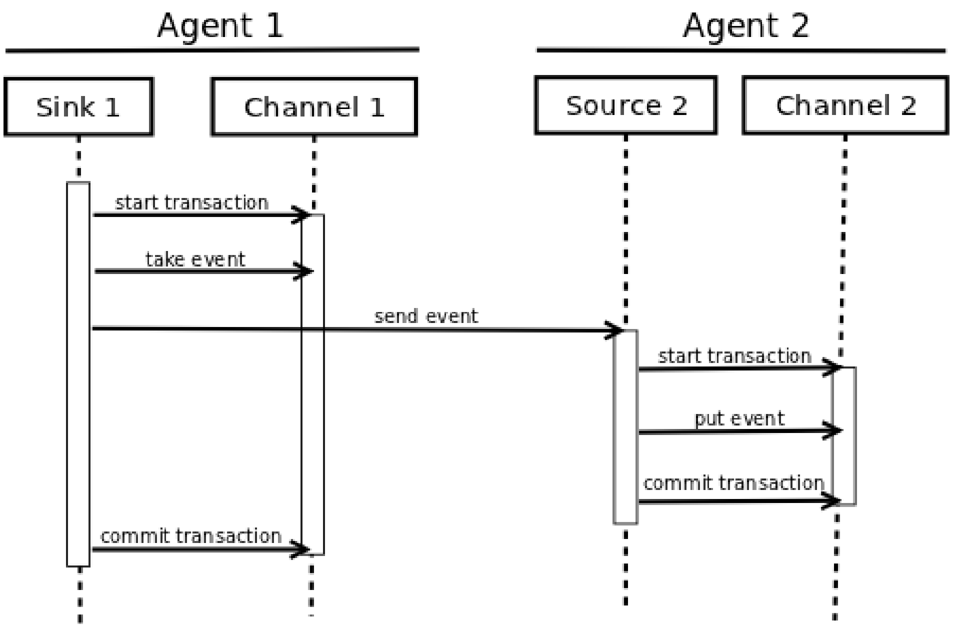

Figure 4. Sequence Diagram between two Agents
(source: https://blogs.apache.org/flume/entry/flume_ng_architecture, Sep. 2018)

When an event hops through many agents, and a communication error is encountered, the next events start getting buffered to the last unaffected agent. This gives enough time for the communication failure to be resolved. In the case that the error is not resolved, the first agent will report to the client, which will allow it to take appropriate action. If the error is resolved in time, the buffered event will start to flow again through the agents.

The following table summarizes the non-functional requirements defined in the deliverable D2.3 BigDataGrapes Software Stack Design and whether Flume satisfies them:

Table 1: Non-Functional Requirements satisfied by Flume

Requirement	Satisfied by Flume	How
STREAMLINED, EFFICIENT DEPLOYMENT STRATEGY	Yes	Dockerized
DEVELOPMENT	Yes	Highly modular
FAULT TOLERANCE	Yes	Channel-based transaction protocol
SCALABILITY	Yes	Distributed by nature
OPEN SOURCE	Yes	-

3. DATA FUSION

3.1 MESSAGING SYSTEM

The ingested data, either batch or stream, should be distributed all over the system by a simple, effective and reliable messaging system. The most popular systems available are RabbitMQ[2] and Apache Kafka[3].

The main difference between the two systems is that Kafka has a dumb broker and relies to the message consumers to implement the desired logic, while RabbitMQ uses a smart broker that communicates the messages to the consumers. This different design choice makes Kafka an interesting candidate in a system that ingest batch and stream data.

Kafka is a messaging framework, that is distributed in nature and runs as a cluster in multiple servers across multiple datacenters. Moreover, Kafka allows the real-time subscription and data publishing of large numbers of systems or applications.

This allows streamlined development and continuous integration facilitating the development of applications that handle either batch or stream data.

An important factor in data ingestion technology, especially when handling data streams, is the fault tolerance capability of the chosen technology. Kafka ensures the minimization of data loss through the implementation of the Leader/Follower concurrency architectural pattern[4].

This approach allows a Kafka cluster to provide advanced fault tolerant capability, which is a mandatory requirement for streaming data applications.

The following figure depicts the general architecture of Kafka:

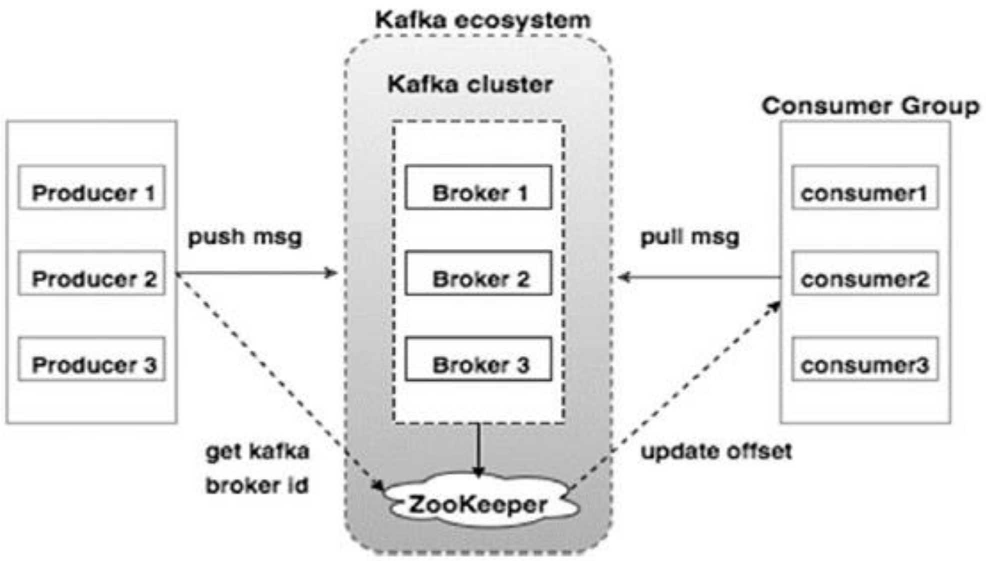

Figure 5. Kafka Architecture
(https://www.tutorialspoint.com/apache_kafka/images/cluster_architecture.jpg, Sep 2018)

The core of Apache Kafka consists of the following four major components:

[2] https://www.rabbitmq.com/

[3] https://kafka.apache.org/

[4] C. D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects, Wiley, 2000.u

- Producer API: Allows applications to communicate with the Kafka cluster and send data in the form of messages
- Consumer API: Allows application to read data in the form of messages from the Kafka cluster
- Connect API: This component has an integral role for streaming systems, because it allows the creation of pipelines that continuously send or receive data from the Kafka cluster
- Streams API: This component offers high-level operators like filters, maps, grouping, windowing, aggregation, and joins. It also allows the development of low-level custom operators.

These core components not only ensure that Kafka is a high-performance messaging framework, in terms of read/write cost[5], but also ensures the scalability of the framework, allowing near-to-real-time processing, regardless of the volume of the data[6] [7].

Kafka is an open source project, that has vast support from the community by providing extensions that facilitate the connection of Kafka with other BigData technologies like the Hadoop Distributed File System (HDFS), the Apache Flink and the Elasticsearch.

Also, rich documentation is available for both batch and stream data facilitating the development of an ecosystem of applications. Also, a Kafka cluster can be dockerized making easier its distribution and deployment.

The following table summarizes the non-functional requirements defined in the deliverable D2.3 BigDataGrapes Software Stack Design and whether Kafka satisfies them:

Table 2: Non-Functional Requirements satisfied by Kafka

Requirement	Satisfied by Kafka	How
STREAMLINED, EFFICIENT DEPLOYMENT STRATEGY	Yes	Can be dockerized
DEVELOPMENT	Yes	Highly modular
FAULT TOLERANCE	Yes	Implements the Leader/Follower pattern
SCALABILITY	Yes	Distributed by nature
OPEN SOURCE	Yes	-

[5] https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

[6] https://insidebigdata.com/2018/04/19/developing-deeper-understanding-apache-kafka-architecture-part-2-write-read-scalability/

[7] https://blog.cloudera.com/blog/2018/05/scalability-of-kafka-messaging-using-consumer-groups/

3.2 LONG TERM STORAGE

It is critical to store the ingested data, either batch or stream, to be available for further processing and analysis. It is also important to select a solution that does not pose processing overheads either when storing the data or retrieving them. Therefore, to minimize the storing and retrieving complexity the use of schema-less database technology helps.

Such a database does not conform to a schema; thus, it does not require any modelling of the data, which in case of multiple sources would be a large burden. Also, a schema-less database can store data with different structure without the need to design a grant common schema or migrating between schemas. Popular schema-less databases are Apache HBase[8] and MongoDB[9].

HBase is a distributed database which follows the Bigtable technology of Google. HBase implements a fault-tolerant method for storing large quantities of sparse data, in the sense that the useful information is a small amount within a large collection of data. Also, HBase has a natural connection with Hadoop, hence can connect with MapReduce processes.

MongoDB is also a distributed database which treats and stores data as JSON documents. Thus, data can have different fields and the data structure is essentially alive since it can be changed over time. Also, MongoDB provides ad-hoc queries, supporting field query, range query and regular expression searches. Moreover, MongoDB has fault-tolerant and load balancing capabilities by providing replication and sharing of the main database.

Table 3: Non-Functional Requirements satisfied by Hbase & MongoDB

Requirement	Satisfied by Hbase & MongoDB	How
STREAMLINED, EFFICIENT DEPLOYMENT STRATEGY	Yes	Can be dockerized
DEVELOPMENT	Yes	Highly modular
FAULT TOLERANCE	Yes	Replication/ Sharding
SCALABILITY	Yes	Distributed by nature
OPEN SOURCE	Yes	-

[8] https://hbase.apache.org/

[9] https://www.mongodb.com/

4. DOCUMENTATION & TOOLS

The following tables contains links that point to the official documentation of the aforementioned tools and also links that point to open source tools that facilitate their configuration and monitoring.

Table 4: Links documenting Flume tool

Flume		
Documentation/ Tool	**Description**	**Link**
Official Documentation	-	https://flume.apache.org/documentation.html
Monitoring Documentation	-	http://www.bigdatareflections.net/blog/?p=83
Docker	-	https://github.com/big-data-europe/docker-flume

Table 5: Links documenting Kafka tool

Kafka		
Documentation/ Tool	**Description**	**Link**
Official Documentation	-	https://kafka.apache.org/documentation/
ZooKeeper	Distributed Coordination	https://zookeeper.apache.org/doc/r3.4.13/
Prometheus	Monitoring/ Alerting	https://prometheus.io/
Landoop	Management	https://www.landoop.com/kafka/kafka-tools/
kafka-benchmark	Benchmarking Tools	https://github.com/fede1024/kafka-benchmark
Docker	-	https://github.com/big-data-europe/docker-kafka

Table 6: Links documenting HBase tool

HBase		
Documentation/ Tool	**Description**	**Link**
Official Documentation	-	https://hbase.apache.org/
HBase Explore	Management/ Administration	https://sourceforge.net/projects/hbaseexplorer/
Docker	-	https://github.com/big-data-europe/docker-hbase

Table 7: Links documenting MongoDB tool

MongoDB		
Documentation/ Tool	**Description**	**Link**
Official Documentation	-	https://docs.mongodb.com/

Studio 3T	Management/ Administration	https://robomongo.org/
MongoDB Tools	Suite of MongoDB utilities	https://github.com/mongodb/mongo-tools
Docker	-	https://hub.docker.com/_/mongo/

5. CONCLUSIONS

This accompanying document for deliverable D3.2 Data Ingestion & Integration Components describes the mechanisms and tools that will be used in the BigDataGrapes platform to ingest data of different nature from multiple sources. The document describes the tools that will be used for data integration across the different BigDataGrapes platform layers, as well as for long-term storage and preservation of data.

The ingestion layer as well as the integration components are key factors of the smooth development and the sustainability of the platform over time. The presented technologies ensure that both batch & stream data will be imported effectively to the platform, and that the components from different layers of the platform will be able to smoothly intercommunicate.