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

D6.1 - Integrated Software Stack and APIs

DELIVERABLE NUMBER	D6.1
DELIVERABLE TITLE	Integrated Software Stack and APIs
RESPONSIBLE AUTHOR	Panagiotis Zervas (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	Panagiotis Zervas
ADDRESS	110 Pentelis Str., Marousi, GR15126, Greece
REPLY TO	pythk@agroknow.com
PHONE	+30 210 6897 905
EU PROJECT OFFICER	Ms. Annamária Nagy
WORKPACKAGE N. | TITLE	WP6 | Integration & Deployment
WORKPACKAGE LEADER	Agroknow
DELIVERABLE N. | TITLE	D6.1 | Integrated Software Stack and APIs
RESPONSIBLE AUTHOR	Panagiotis Zervas (Agroknow)
REPLY TO	pzervas@agroknow.com
DOCUMENT URL	http://www.bigdatagrapes.eu/
DATE OF DELIVERY (CONTRACTUAL)	31 December 2018 (M12), 30 June 2019 (M18, Updated Version)
DATE OF DELIVERY (SUBMITTED)	4 January 2019 (M13), 28 June 2019 (M18, Updated Version)
VERSION | STATUS	2.0 | Final
NATURE	Demonstrator (DEM)
DISSEMINATION LEVEL	Public (PU)
AUTHORS (PARTNER)	Pythagoras Karampiperis (Agroknow), Sotiris Konstantinidis (Agroknow), Panagiotis Zervas (Agroknow), Mihalis Papakonstadinou (Agroknow)
CONTRIBUTORS	-
REVIEWER	Nicola Tonellotto (CNR)

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	Table of Contents	03/12/2018	Pythagoras Karampiperis (Agroknow), Sotiris Konstantinidis (Agroknow), Panagiotis Zervas (Agroknow)
0.4	Section 1, 2, 3	10/12/2018	Pythagoras Karampiperis (Agroknow), Sotiris Konstantinidis (Agroknow), Panagiotis Zervas (Agroknow)
0.6	Section 4,5,6	14/12/2018	Pythagoras Karampiperis (Agroknow), Sotiris Konstantinidis (Agroknow), Panagiotis Zervas (Agroknow)
0.8	Initial version	28/12/2018	Pythagoras Karampiperis (Agroknow), Sotiris Konstantinidis (Agroknow), Panagiotis Zervas (Agroknow)
1.0	Final Version after internal review	4/1/2019	Sotiris Konstantinidis (Agroknow)
1.5	Pre-final draft for 2nd version	6/6/2019	Mihalis Papakonstadinou (Agroknow)
2.0	Updated version	28/6/2019	Panagiotis Zervas (Agroknow), Mihalis Papakonstadinou (Agroknow)

PARTICIPANTS		CONTACT
Agroknow IKE (Agroknow, Greece)	Agroknow	Panagiotis Zervas Email: pzervas@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)	KATHOLIEKE UNIVERSITEIT LEUVEN	Katrien Verbert Email: katrien.verbert@cs.kuleuven.be
Geocledian GmbH (GEOCLEDIAN Germany)	geo|cledian	Stefan Scherer Email: stefan.scherer@geocledian.com
Institut National de la Recherché Agronomique (INRA, France)	INRA SCIENCE & IMPACT	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
SYMBEEOSIS LONG LIVE LIFE S.A. (Symbeeosis, Greece)	Symbeeosis	Kostas Gardikis Email: c.gardikis@gmail.com

ACRONYMS LIST

API	Application Programming Interface
RDBMS	Relational database management system
CSV	Comma-separated values
JSON	JavaScript Object Notation
OLTP	Online transaction processing
IP	Internet Protocol address
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
OWL	Web Ontology Language
SQL	Structured Query Language

EXECUTIVE SUMMARY

This accompanying document for deliverable D6.1 Integrates Software Stack & APIs describes the chosen container platform as well as the software components that consist the BigDataGrapes Software Stack. The document first introduces the chosen deployment platform in which all the current and the future software components will be hosted. The next chapters are organized according to the BigDataGrapes Software Stack Architecture and provide all the necessary steps for deploying the software components of the BigDataGrapes Software Stack.

TABLE OF CONTENTS

LIST OF TABLES

1 INTRODUCTION

The second version of the BigDataGrapes integrated software stack includes all the basic components as have been described and provided by WP's 3 and 4. The platform is based on container technology in order to ensure that the individual components and the platform can be easily configured and smoothly deployed to any infrastructure. Container technology is a way to package an application with its dependencies and minimizes the effort of the integration of different software components, and the deployment of a group or a stack of software in any infrastructure regardless of the underlying hardware. The following figure depicts the different BigDataGrapes Architecture Layers. The first version of the integrated stack contains the necessary technology regarding the Ingestion, the Persistence and the Processing Layers. Moreover, contains the technology that will be used as a basis for the integration of the different layers.

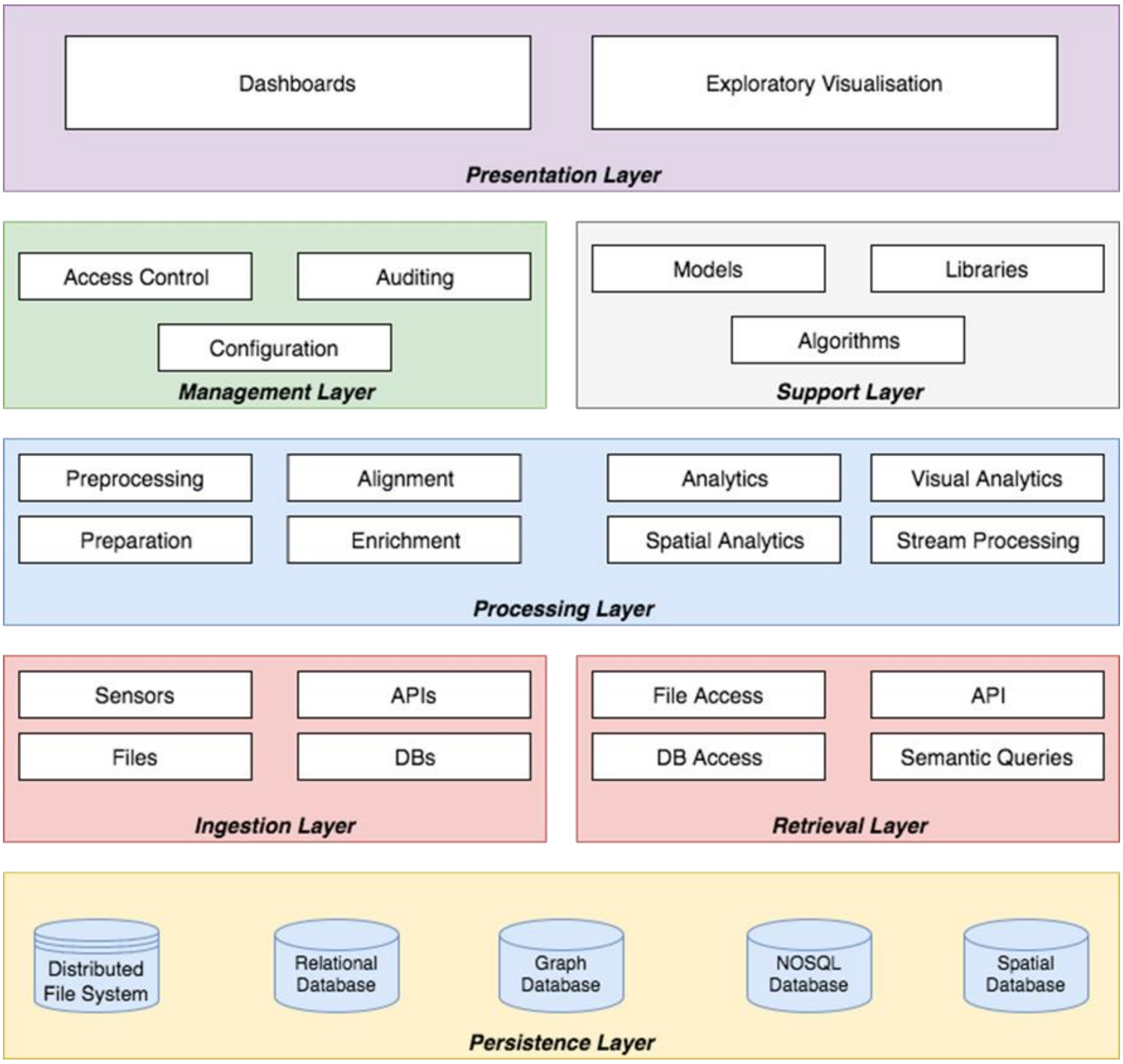

Figure 1: BigDataGrapes Architecture Layers

The second chapter describes the chosen container technology. The next chapters are organized in respect to the BigDataGrapes Software Stack Architecture and provide the necessary details for the containerized version of each component More specifically, Section 3: Provides details about the Ingestion Components, Section 4: Provides details about the Integration Components, Section 5: Provides details about the Persistence Components and Section 6: Provides details about the Processing Components.

2 DOCKER PLATFORM

The Docker platform[1] is a suite of tools that offers containerization functionalities that ensure smooth building, delivery and deployment of complex software systems. The basic functionality of the Docker platform offers a template-way of packaging software components, to be easily deployed, delivered and extended. Moreover, the docker platform includes the Docker Composer, which is an orchestration engine, that is used to simplify the deployment of large and complex systems, independent of the underlying infrastructure.

2.1 DOCKER IMAGE

An image is package that is executable and includes everything needed, from source code to environment variables to run an application. A container is an image with state, thus whenever an image is executed it becomes a container.

2.2 DOCKER COMPOSE

Compose is a tool used to define and run application that use many different containers. It allows the deployment of every component and service with a single command. Compose takes as input a simple yml configuration file, that describes all the different docker images. Moreover, compose provides commands that allow the managing of the whole lifecycle of an application, i.e.

- Start, stop and rebuild services
- Monitor the current status of the services
- Monitor the log output of the running services

The documentation of deploying the BigDataGrapes software stack through Docker Compose and Docker Image can be found in https://github.com/BigDataGrapes-EU/deliverable-D6.1, also the dockerized version of the components can be accessed through https://hub.docker.com/u/bigdatagrapes

[1] https://www.docker.com/

3 DATA INGESTION COMPONENTS

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

3.1 FLUME

Apache Flume[2] 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.

Table 1: Flume deployment

Deploy Flume		
Step	**Description**	**Command**
1	Download Flume Docker	docker pull bigdatagrapes/flume
2	Deploy Flume	docker run --name flume -d -i -t bigdatagrapes/flume /bin/bash
3	Check deployment	docker container ls

Table 2: Flume parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-t	Allocate a pseudo-TTY
-i	Keep STDIN open even if not attached

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

4 INTEGRATION COMPONENTS

4.1 KAFKA

Apache Kafka[3] is a messaging framework, that is distributed in nature and runs as a cluster in multiple servers across multiple datacentres. 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. This approach allows a Kafka cluster to provide advanced fault tolerant capability, which is a mandatory requirement for streaming data applications.

Table 3: Kafka deployment

Deploy Kafka		
Step	**Description**	**Command**
1	Download Kafka Docker	docker pull bigdatagrapes/kafka
2	Deploy Kafka	docker run --name kafka -d -p 2181:2181 -p 9092:9092 --env 192.168.1.77 bigdatagrapes/kafka
3	Check deployment	docker container ls

Table 4: Kafka parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address

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

5 PERSISTENCE COMPONENTS

The layer deals with the long-term storage and management of data handled by the software stack. Its purpose is to consistently and reliably make the data available to the processing layer. The layer incorporates schema-less persistence technologies, that do not pose processing overheads either when storing the data or retrieving them. Therefore, the storing and retrieving complexity is minimized.

5.1 MONGODB

MongoDB[4] is 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 5: MongoDB deployment

Deploy MongoDB		
Step	**Description**	**Command**
1	Download Mongo Docker	docker pull bigdatagrapes/mongo
2	Deploy Mongo	docker run --name mongo -d -p 27017:27017 --env 192.168.1.77 bigdatagrapes/mongo
3	Check deployment	docker container ls

Table 6: MongoDB parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address

5.2 HADOOP DISTRIBUTED FILE SYSTEM

The Apache Hadoop[5] software library is a framework that contains also a file system distributed in nature, called Hadoop Distributed File System (HDFS). HDFS transfers data between nodes and is closely coupled with MapReduce. HDFS breaks the information in separate blocks and orchestrates the distribution to different nodes, thus enabling parallel processing and fault-tolerance.

Table 7: HDFS deployment

Deploy HDFS		
Step	**Description**	**Command**
1	Download HDFS Docker	docker pull bigdatagrapes/hadoop
2	Deploy HDFS	docker run --name hadoop -d bigdatagrapes/hadoop
3	Check deployment	docker container ls

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

[5] https://hadoop.apache.org/

Table 8: HDFS parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID

5.3 HBASE

Apache HBase[6] 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 be connected with MapReduce processes.

Table 9: HBase deployment

Deploy HBase		
Step	**Description**	**Command**
1	Download HBase Docker	docker pull bigdatagrapes/hbase
2	Deploy HBase	docker run --name hbase -d -p 16000:16000 -p 16010:16010 --env 192.168.1.77 bigdatagrapes/hbase
3	Check deployment	docker container ls

Table 10: HBase parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container's port(s) to the host
--env	Set IP address

5.4 ELASTICSEARCH

Elasticsearch[7] is a distributed database, providing a full-text search engine based on Lucene. The distributed nature of Elasticsearch, allows near real-time search in all kinds of documents. The indices of Elasticsearch can be divided into shards, hence supporting automatic rebalancing and routing. Moreover, the indices can be replicated to support efficient fault-tolerance.

Furthermore, Elasticsearch encapsulates out-of-the-box methods for establishing connections with messaging systems like Kafka, which makes integration easier and allows the faster development of real-time applications.

Table 11: Elasticsearch deployment

Deploy Elasticsearch		
Step	**Description**	**Command**

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

[7] https://www.elastic.co/

1	Download Elasticsearch Docker	docker pull bigdatagrapes/elasticsearch
2	Deploy Elasticsearch	docker run --name elasticsearch -d -p 9200:9200 --env 192.168.1.77 bigdatagrapes/elasticsearch
3	Check deployment	docker container ls

BigDataGrapes integrated software stack also provides the Kibana tool, which can be used to monitor & query Elasticsearch through a web user interface.

Table 12: Kibana deployment

Deploy Kibana		
Step	**Description**	**Command**
1	Download Kibana Docker	docker pull bigdatagrapes/kibana
2	Deploy Kibana	docker run --name kibana -d --env ELASTICSEARCH_URL=http://192.168.1.77:9200 --env 192.168.1.77 -p 5601:5601 bigdatagrapes/kibana
3	Check deployment	docker container ls

Table 13: Kibana parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address
ELASTICSEARCH_URL	Set the ip of elasticsearch, that is needed from kibana

BigDataGrapes integrated software stack also Logstash, which works on top of Elasticsearch and is used mainly for log collection

Table 14: Logstash deployment

Deploy Logstash		
Step	**Description**	**Command**
1	Download Logstash Docker	docker pull bigdatagrapes/logstash
2	Deploy Logstash	docker run --name logstash -d bigdatagrapes/logstash
3	Check deployment	docker container ls

Table 15: Logstash parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID

5.5 MYSQL

MySQL[8] is a relational database management system (RDBMS), which provides a robust implementation of the SQL standard.

Table 16: MySQL deployment

Deploy MySQL		
Step	**Description**	**Command**
1	Download MySQL Docker	docker pull bigdatagrapes/mysql
2	Deploy MySQL	docker run --name mysql -e MYSQL_ROOT_PASSWORD=test -p 3306:3306 -p 33060:33060 -d --env 192.168.1.77 bigdatagrapes/mysql
3	Check deployment	docker container ls

Table 17: MySQL parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container's port(s) to the host
--env	Set IP address
MYSQL_ROOT_PASSWORD	Root User password

The BigDataGrapes integrated software stack also provides the PhPMyAdmin[9] user interface to monitor and query the MySQL RDBMS through a web user interface.

Table 18: PhpMyAdmin deployment

Deploy PhpMyAdmin		
Step	**Description**	**Command**
1	Download PhpMyAdmin Docker	docker pull bigdatagrapes/phpmyadmin
2	Deploy PhpMyAdmin	docker run --name phpmyadmin -d --env PMA_HOST=192.168.1.77 -p 8080:80 bigdatagrapes/phpmyadmin
3	Check deployment	docker container ls

Table 19: PhpMyAdmin parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container's port(s) to the host
--env	Set IP address
PMA_HOST	Set the ip of MySQL, that is needed from PhpMyAdmin

[8] https://www.mysql.com/

[9] https://www.phpmyadmin.net/

5.6 GRAPHDB

GraphDB[10] is an RDF triplestore compliant with the core semantic web W3C specifications (RDF, RDFS, OWL). It acts as a SAIL over the RDF4J framework, thus providing functionalities for all critical semantic graph operations (storing, indexing, reasoning, querying, etc.). The query language used is the implementation of the SPARQL 1.1 specifications, while connectors with Elasticsearch and Lucence are incorporated in the system.

Table 20: GraphDB deployment

Deploy GraphDB		
Step	**Description**	**Command**
1	Download GraphDB Docker	docker pull bigdatagrapes/graphdb
2	Deploy GraphDB	docker run --name graphdb -d -p 7200:7200 --env 192.168.1.77 bigdatagrapes/graphdb
3	Check deployment	docker container ls

Table 21: GraphDB parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container's port(s) to the host
--env	Set IP address

5.7 VIRTUOSO

Virtuoso[11] is an engine that acts as a single-point server and middleware for multiple data management paradigms (relational, object-relational, XML, RDF, file-based). The underlying storage mechanism is a traditional relational database with abstraction and serialisation components built into the framework for exposing data of the aforementioned different representations. RDF data are accessed and queried using an extension of the SPARQL specification

Table 22: Virtuoso deployment

Deploy Virtuoso		
Step	**Description**	**Command**
1	Download Virtuoso Docker	docker pull bigdatagrapes/virtuoso
2	Deploy Virtuoso	docker run --name virtuoso -d -p 7200:7200 --env 192.168.1.77 bigdatagrapes/virtuoso
3	Check deployment	docker container ls

Table 23: Virtuoso parameter description

Parameter Description

[10] http://graphdb.ontotext.com/

[11] https://virtuoso.openlinksw.com/

Parameter	Description
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address

5.8 NEO4J

Neo4j[12] is the most popular graph database system at the time of writing. It is a native graph storage framework, following the property graph model for representing and storing data, i.e., the representation model conceptualises information as nodes, edges or properties. Accessing and querying the underlying data is achieved via the usage of the open-sourced Cypher query language, originally developed exclusively for Neo4j.

Table 24: Neo4j deployment

Deploy Neo4j		
Step	**Description**	**Command**
1	Download Neo4j Docker	docker pull bigdatagrapes/neo4j
2	Deploy Neo4j	docker run --name neo4j -d -p 7200:7200 --env 192.168.1.77 bigdatagrapes/neo4j
3	Check deployment	docker container ls

Table 25: Neo4j parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address

5.9 APACHE CASSANDRA

Apache Cassandra[REFERENCE: http://cassandra.apache.org/] is a NoSQL storage engine designed to handle large amounts of write requests. Being a NoSQL engine it can easily handle model updates. It is designed to be easily configurable and deployed in a multi-node, distributed manner.

Table XX: Apache Cassandra deployment

Deploy Apache Cassandra		
Step	**Description**	**Command**
1	Download Mongo Docker	docker pull bigdatagrapes/cassandra
2	Deploy Mongo	docker run --name cassandra -d -p 9042:9042 --env 192.168.1.77 bigdatagrapes/ cassandra
3	Check deployment	docker container ls

Table 6: Apache Cassandra parameter description

Parameter Description

[12] https://neo4j.com

Parameter	Description
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address

6 PROCESSING COMPONENTS

6.1 HADOOP

The Apache Hadoop[13] software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale out from single servers to thousands of machines, each offering local computation and storage. Internally, the framework does not rely on hardware to deliver high availability. Instead, the framework is designed to detect and handle failures at the application layer. In this way, the Hadoop offers highly available service on top of a cluster of computers, each of which may be prone to failures.

Table 26: Hadoop deployment

Deploy Hadoop		
Step	Description	Command
1	Download Hadoop Docker	docker pull bigdatagrapes/hadoop
2	Deploy Hadoop	docker run --name hadoop -d bigdatagrapes/hadoop
3	Check deployment	docker container ls

Table 27: Hadoop parameter description

Parameter Description	
Parameter	Description
--name	Assign a name to the container
-d	Run container in background and print container ID

6.2 SPARK

Apache Spark[14] is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for stream processing. While MapReduce continues to be a popular batch-processing tool, Apache Spark's flexibility and in-memory performance make it a much more powerful batch execution engine. The basic components of a Spark ecosystem are the Master and the Worker node. The master is responsible for negotiating resource requests while the Worker nodes execute tasks over time.

Table 28: Spark Master deployment

Deploy Spark Master		
Step	Description	Command
1	Download Spark Master Docker	docker pull bigdatagrapes/spark-master
2	Deploy Master Spark	docker run --name sparkmaster -d -p 8080:8080 -p 6066:6066 -p 7077:7077 --env 192.168.1.77 bigdatagrapes/spark-master
3	Check deployment	docker container ls

[13] https://hadoop.apache.org/

[14] https://spark.apache.org/

Table 29: Spark Master parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container's port(s) to the host
--env	Set IP address

Table 30: Spark Worker deployment

Deploy Spark Worker		
Step	**Description**	**Command**
1	Download Spark Worker Docker	docker pull bigdatagrapes/spark-worker
2	Deploy Worker Spark	docker run --name sparkworker -d -p 8081:8081 --env SPARK_MASTER=192.168.1.77:7077 bigdatagrapes/spark-worker
3	Check deployment	docker container ls

Table 31: Spark Worker parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container's port(s) to the host
--env	Set IP address
SPARK_MASTER	Set the ip of the Spark Master

6.3 FLINK

For the purposes of stream processing in BigDataGrapes, a powerful, well-supported combination of technologies is the usage of Apache Flink[15]. The Apache Flink is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications. Flink is based on the DataFlow model i.e. processing the elements as and when they come rather than processing them in micro-batches. Dataflow allows Flink to process millions of records per minutes at milliseconds of latencies on a single machine. Micro-batches can contain huge number of elements and the resources needed to process those elements at once can be substantial. In the case of a sparse data stream in which you get only a burst of data at irregular intervals, it can become a major pain point. Furthermore, Flink provides robust fault-tolerance using checkpointing (periodically saving internal state to external sources such as HDFS).

Table 32: Flink deployment

Deploy Flink		
Step	**Description**	**Command**
1	Download Flink Docker	docker pull bigdatagrapes/flink

[15] https://flink.apache.org/

2	Deploy Flink	docker run --name flink -d -p 8081:8081 --env 192.168.1.77 bigdatagrapes/flink jobmanager
3	Check deployment	docker container ls

Table 33: Flink parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address
jobmanager	Assign Flink component to Job Manager

6.4 SPARKLING WATER

Sparking Water[16] stems from H2O, and is a in-memory platform for machine learning, allowing effective and efficient machine learning on big data. Sparkling Water allows users to combine the its machine learning algorithms in conjunction Apache Spark ecosystem. Sparkling Water and Apache Spark allows for a seamless experience for users who want to interact with distributed databases/filesystems, build a model and make predictions. It is used for exploring and analysing datasets held in cloud computing systems and in the Apache HDFS as well as in the conventional operating-systems Linux, macOS, and Microsoft Windows.

Table 34: Sparkling Water deployment

Deploy Sparkling Water		
Step	**Description**	**Command**
1	Download Sparkling Water Docker	docker pull bigdatagrapes/sparklingwater
2	Deploy Sparkling Water	docker run --name sparklingwater -d -p 7200:7200 --env 192.168.1.77 bigdatagrapes/ sparklingwater
3	Check deployment	docker container ls

Table 35: Sparkling Water parameter description

Parameter Description	
Parameter	**Description**
--name	Assign a name to the container
-d	Run container in background and print container ID
-p	Publish a container’s port(s) to the host
--env	Set IP address

[16] https://www.h2o.ai/products/h2o-sparkling-water/

7 CONCLUSIONS

This accompanying document for deliverable D6.1 Integrated Software Stack and APIs describes the container technology that will be used in the BigDataGrapes stack. Moreover, it includes the dockerized versions of all the tools & frameworks described in the various work packages. Every dockerized component is available through the BigDataGrapes docker hub and also detailed documentation regarding the deployment of the entire stack is available through the BigDataGrapes github.

The containerization of the different tools & frameworks ensures the intercommunication of the components that reside in different layers, thus allowing not only the smooth and effective deployment of the BigDataGrapes Platform stack in any infrastructure, but also in decreasing the effort for integrating new components & technologies.