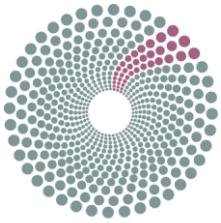

BIG DATA EUROPE

Support Action

Big Data Europe – Empowering Communities with Data Technologies

Project Number: 644564 Start Date of Project: 01/01/2015 Duration: 36 months

Deliverable 3.4: Assessment on Application of Generic Data Management Technologies II

Dissemination Level	Public
Due Date of Deliverable	M10, 31/10/2015 (officially shifted to M12)
Actual Submission Date	M12, 24/12/2015
Work Package	WP3, Big Data Generic Enabling Technologies and Architecture
Task	T3.1
Type	Report
Approval Status	Final
Version	v1.0
Number of Pages	33
Filename	D3.4_Assessment_on_application_of_ generic_data_management_technologies_II.pdf

Abstract: This document assesses a set of data management technologies, ranging from data storage, over computation frameworks to monitoring tools. The assessment is based on the outcomes of the requirements elicitation and the base platform architecture.

Project funded by the European Union's Horizon 2020 Research and Innovation Programme (2014 – 2020)

History

Version	Date	Reason	Revised by
0.0	09/11/2015	Initial version	Erika Pauwels (TenForce)
0.1	26/11/2015	Chapter on HDFS	Hajira Jabeen (InfAI)
0.2	27/11/2015	Chapter on base platform	Erika Pauwels (TenForce) Aad Versteden (TenForce)
0.3	08/12/2015	Chapter on monitoring and deployment interfaces	Erika Pauwels (TenForce) Aad Versteden (TenForce)
0.4	09/12/2015	Chapter on computational frameworks	Hajira Jabeen (InfAI)
0.5	10/12/2015	Chapter on message passing	Jürgen Jakobitsch (SWC)
0.6	11/12/2015	Finalizing for review	Erika Pauwels (TenForce) Aad Versteden (TenForce)
0.7	15/12/2015	Peer review	George Papadakis (UoA)
0.8	22/12/2015	Address peer review comments	Erika Pauwels (TenForce)
1.0	24/12/2015	Final version	Erika Pauwels (TenForce)

Author List

Organisation	Name	Contact Information
TenForce	Erika Pauwels	erika.pauwels@tenforce.com
TenForce	Aad Versteden	aad.versteden@tenforce.com
InfAI	Hajira Jabeen	jabeen@informatik.uni-leipzig.de
SWC	Jürgen Jakobitsch	jakobitschj@semantic-web.at
UoA	George Papadakis	gpapadis@di.uoa.gr

Executive Summary

The Big Data ecosystem is under continuous flux. This reflects itself in the components selected for the Big Data Europe (BDE) platform. We have discussed and evaluated the main components for the BDE platform, but expect changes in the field.

The BDE platform as a whole comprises a base platform and a set of components that can run on it. This is described in "WP3 Deliverable 3.3: Big Data Integrator Deployment and Component Interface Specification" which outlined the pipelines in a more generic sense. In contrast, this deliverable details the way they work. Generic images for a particular technology, like Apache Flink, can be extended in order to implement a component of a pipeline. The pipeline can be constructed by building a Docker Compose configuration. This configuration can then be shared with pilot case partners who may deploy the pipelines. In the near future, running a pipeline should be simple.

We move together with the community to a platform that is efficient and easy to use. We also embrace evolution where it would benefit the platform. We have described the limitations of the current state of the technologies in the platform and foresee these limitations to be resolved by selecting the right technologies for the base platform. This holds especially for the deployment interfaces. It is our goal to support each of the users in a sufficient way. Deployment interfaces are under active development, but don't seem to meet our expectations yet.

The construction of a broader pipeline will likely use a map-reduce based technology. This is an assumption we make based on the input we have received from the pilot case partners. However, our focus might change in the future. We have described the benefits and downsides of Spark and Flink, and how it would fit within the platform. Here too, we make the conscious decision to embrace the future. We are actively looking at Flink, knowing that its support will improve over time.

The last step which has received attention in this deliverable is monitoring. Both the monitoring of the cluster as a whole and of individual pipelines will be considered. What we intend to monitor is based on the way we expect users to use the platform. Monitoring is considered to be a way to build trust in the system. These topics are under active development and will be studied further. The monitoring of the cluster's resources is comparatively well supported, but we should research it further. Monitoring of individual pipelines is a younger topic. Further research is needed to select a solid strategy.

A set of components was selected for the BDE platform. Both for the platform itself, as for the pipelines that can be ran by it. The components are expected to evolve over time. We expect evolution of our logging strategy as we gain more hands-on experience with the platform.

Abbreviations and Acronyms

AMQP	Advanced Message Queueing Protocol
API	Application Programming Interface
AWS	Amazon Web Services
BDE	Big Data Europe
CLI	Command-Line Interface
CPU	Central Processing Unit
DBMS	Database Management System
DNS	Domain Name System
DSL	Domain Specific Languages
ELK	Elasticsearch, Logstash, Kibana
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
OS	Operating System
RDD	Resilient Distributed Dataset
REPL	Read-Evaluate-Print-Loop
REST	Representational State Transfer
SQL	Structured Query Language
STOMP	Streaming Text Oriented Messaging Protocol
UI	User Interface

Table of Contents

List of Figures

List of Tables

1. Introduction

This deliverable is a follow-up of [1] which introduced the 4 Vs of Big Data: Volume, Velocity, Variety and Veracity. The requirements elicitation described in [2] reveals that there is not one V that overshadows the others. Not even within a specific societal challenge. The BDE platform therefore needs to be flexible in supporting Big Data pipelines characterized by any of these Vs. The base platform described in [3] offers a solid foundation to this end. This document focuses on the technologies used on top of the base platform to implement a Big Data pipeline and to monitor and manage the platform.

In more detail, this deliverable narrows down the selection of technologies compared to the previous version. It describes the technologies that are expected to be main technologies in the BDE platform pipelines and stands in contrast to the extensive catalog of technologies described in [1]. Yet, this selection of technologies is by no means fixed. In the fast moving space of Big Data, it may be that new technologies rise, whilst others fall out of grace.

Section 2 assesses the technologies used in the base platform and describes how these technologies enhance the flexibility and ease of use of the BDE platform. Section 3 focuses on the technologies used in the Big Data pipelines. The computational frameworks like Flink [4] and Spark [5] are the most important here. The following sections 4 and 5 handle topics that were not touched in the previous version [1], giving a first insight on monitoring, logging and deployment interfaces. Section 6 concludes this assessment, focusing on the future work that needs to be done in the upcoming months.

2. Base Platform

Figure 1 shows the BDE base platform architecture consisting of Mesos [6] and Docker [7] as introduced and explained in detail in [3]. This section assesses the selected technologies in the base platform together with their advantages, issues and potential alternatives.

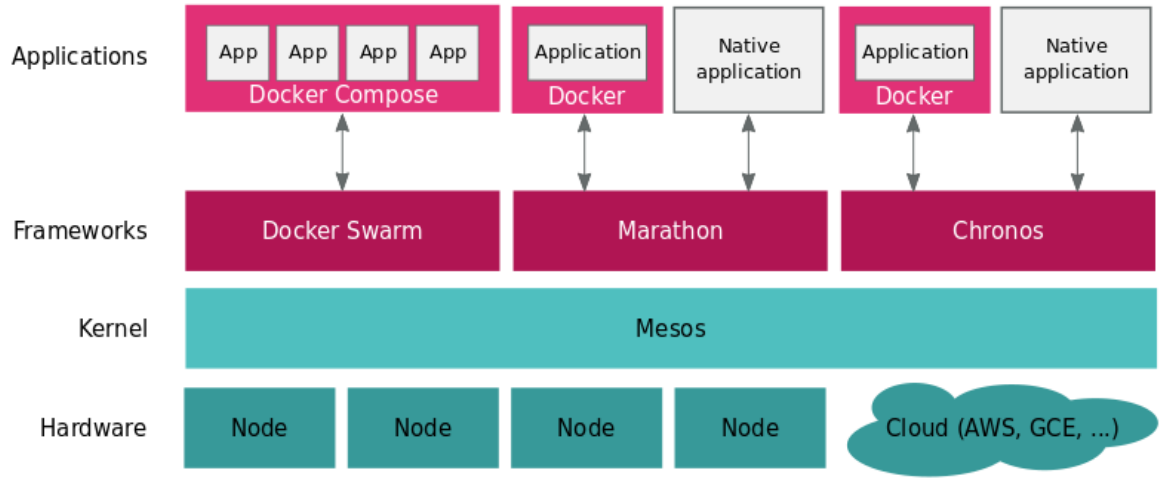

Figure 1: BDE base platform architecture

2.1 Developing a Pipeline Component

An application running on the BDE platform can be seen as a pipeline consisting of multiple components, which are wired together in order to solve a specific Big Data problem. It should be easy for a programmer to develop, build and deploy such a pipeline component. But on the other hand he should not be limited in his choice of technology and implementation by the platform.

Docker offers a good solution in this regard. The Docker container abstraction allows the components to incorporate virtually any technology. One container could run Spark, another could run Storm [8], and yet another container can use a completely custom implementation. Moreover, the containers' templates, named Docker images, can be easily built and shipped to any platform.

At first sight, Docker containers seem very similar to virtual machines as they both run applications in a sandbox. However, as Docker containers reuse their host's kernel and share libraries as much as possible, they are more lightweight than traditional virtual machines and, thus, more suitable for a Big Data platform.

One can argue that Docker containers imply a performance overhead compared to a native installation, in which every component runs on its own set of physical machines. This is true to a certain degree, but as explained above, this overhead is very limited as Docker containers are very lightweight virtual machines. We find that the benefits of Docker containers outweigh this overhead for the BDE platform. Maintaining a set of machines with native component installations would be costly, time-consuming and result in a custom setup per component.

In the fast moving space of Big Data, it is seemingly impossible to know what the optimal technologies will be in the future. As stated earlier, by using Docker, the BDE platform does

not focus on a limited set of technologies, but keeps options open for the future. If a new Big Data technology appears, the BDE platform is able to embrace this technology without modifications to the base platform.

However, this freedom of choice might result in chaos. For example, when each pipeline component would use a unique technology. In order to encourage technology reuse and to facilitate the development of a component, the BDE platform will provide base Docker images.

A base Docker image offers a template implementation for a specific - in our case Big Data - technology. As illustrated in Figure 2, this template can be easily extended by a programmer with his own custom implementation to solve a particular problem. The provisioning of base images makes it a lot easier to implement a new pipeline component without limiting the options of the implementer. Should some base technology not be supported yet, a new base image can be built. The selection of technologies that will be provided as base images mainly depends on the pilot cases that need to be implemented.

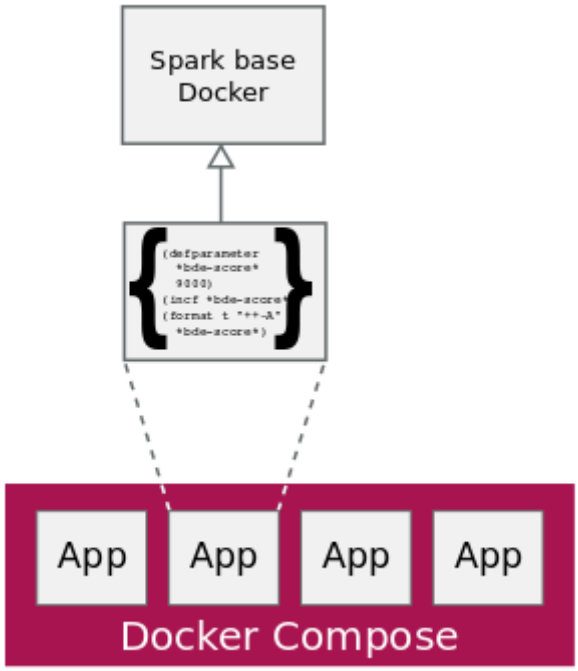

Figure 2: Developing a pipeline component by extending a base Docker image

2.2 Building a Pipeline

A Big Data pipeline consists of multiple components that are wired together into one smooth-running system. Given that the pipeline components are provided as Docker containers, Docker Compose [9] offers a good solution to describe such a pipeline. A Docker Compose file contains the identifiers of all Docker containers which should be ran, and how they need to be wired together. For example, a Docker Compose file could contain a Docker for the Cassandra database [10] and another Docker for the Spark application to perform computation.

Since version 1.9, released on November 3, 2015, Docker supports multihost networks. Containers deployed on different hosts can communicate with each other using the container

names on an overlay network [11]. Docker Compose version 1.5, which is released together with Docker v1.9, has experimental support for the new Docker networking system [12]. Production-ready support is expected in the next release of Docker Compose. A limitation of the current integration between the Docker networking and Docker Compose is the lack of aliases [13]. Containers can only connect to each other by their container name, thus resulting in a one-to-one connection. This is not scalable.

More specifically, in the case of Docker Compose, a container's name is constructed based on the project name, container identifier and an instance counter, unless the user explicitly provides a container name. For example, a container identified by 'database' in the Docker Compose file of the 'bde' project, will be named 'bde_database_1'. When scaling up this container, the following instances will be named 'bde_database_2', 'bde_database_3', etc. Other containers can connect to only one of these instances, for example 'bde_database_1' or 'bde_database_2', as illustrated in Figure 3. They cannot use a more abstract alias like 'database' which will transparently return one of the database containers, be it either 'bde_database_1', 'bde_database_2' or 'bde_database_3', as illustrated in Figure 4. This issue has been acknowledged by the Docker community and is under active development at the time of writing.

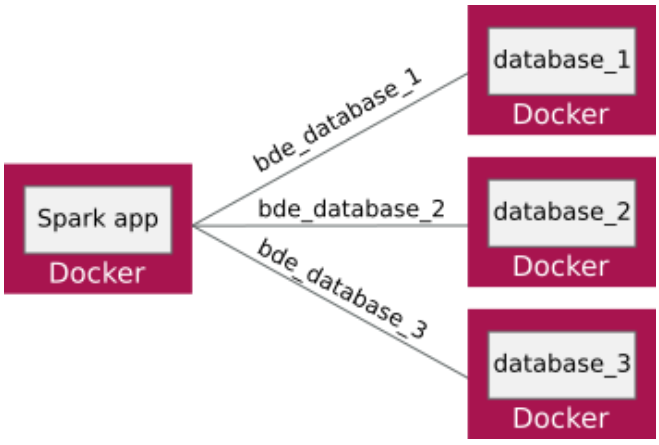

Figure 3: Docker container networking without aliases

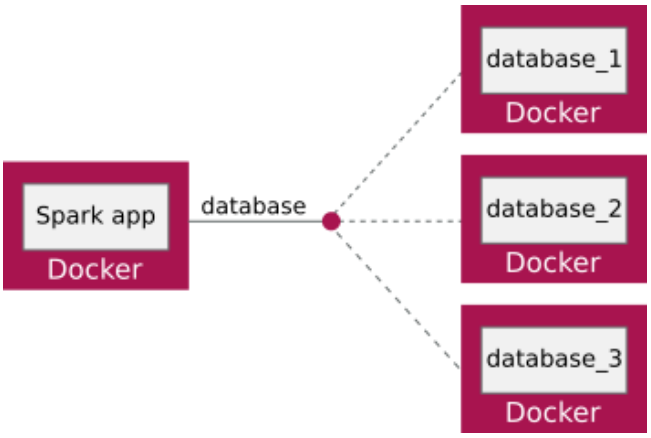

Figure 4: Docker container networking using aliases

2.3 Deployment

Manually maintaining machines is a chore. For big systems, it becomes a big problem. Servers can run on many locations: they can be available in-house, rented from a cloud provider, etc. A flexible architecture should abstract from the hardware, as the platform user should not bother about the underlying machines being used. Mesos offers this flexibility by putting a layer on top of the hardware, as illustrated in Figure 1. It can be thought of as a kernel for a distributed operating system that makes the several hardware nodes in the cluster behave as one big machine to the user.

When talking about resource management in a cluster, Mesos is often compared to YARN [14]. However, the two technologies significantly differ in their main design priority. Mesos was built to be a scalable global resource manager, while YARN was created out of the necessity to scale Hadoop. That is the reason why the latter is also known under the name of MapReduce v2. As a consequence, YARN is optimized to schedule Hadoop jobs which are typically batch jobs with long run times. It is possible to schedule other kinds of workloads on YARN, but this is not an ideal solution.

The BDE platform doesn't want to restrict the user to the technologies of the Hadoop stack. As explained in section 2.1, by using Docker, the BDE platform offers the flexibility to use virtually any software technology. Unlike Mesos, YARN is unable to deploy and schedule Docker containers and, thus, it is not a suitable resource manager for the BDE platform.

On top of the kernel layer, several frameworks are available that offer services to deploy applications on the platform. The three most important frameworks considered in the BDE platform are:

- *Docker Swarm* [15]*:* a clustering tool turning a pool of Docker hosts into a single virtual host
- *Marathon* [16]*:* a startup or init daemon for a distributed OS
- *Chronos* [17]*:* a cron daemon for a distributed OS

Next to these three frameworks, a couple of other frameworks are publicly available on top of Mesos [18]. Moreover, Mesos provides an API in several languages (Java, Python, C++) to develop your own framework.

Docker Swarm as well as Marathon and Chronos are able to deploy dockerized applications which makes them all suitable frameworks for running our Big Data pipelines. Marathon and Chronos can also deploy native applications, unlike Docker Swarm. We don't expect this to happen in the BDE platform, but we keep supporting Marathon and Chronos in the short term in case the need for native applications arise.

Docker Swarm serves the standard Docker API. As a consequence, Docker Compose can seamlessly use Docker Swarm instead of Docker Engine to scale a pipeline to multiple hosts. All Docker Compose commands that can be executed using Docker Engine (e.g. docker-compose up) can also be executed using Docker Swarm without modifications.

This has very interesting ramifications, as we can configure a pipeline by a short description of the pipeline's components and their connections. Running a new pipeline therefore becomes as simple as:

1. Download pipeline description
2. docker-compose up

Terminating a pipeline, if it doesn't terminate automatically, can also be done using one single command (docker-compose stop). Hence, the time to reconfigure a cluster for a new pipeline is minimal when using Docker Swarm and Docker Compose. This is a major benefit for the BDE platform where multiple pilot cases need to run on a limited set of servers.

On November 3, 2015 Docker released a production-ready Docker Swarm version 1.0 which supports the new networking features of Docker Engine v1.9, including multihost networking [19]. Hence, Docker Swarm is able to run and connect Docker containers on multiple hosts. However, the integration of Docker Swarm and Mesos is still in an experimental phase [20]. One can replace the built-in Docker Swarm scheduler with Mesos, so the Mesos master handles the scheduling of Docker containers on request of the Swarm manager and runs Docker containers on top of this infrastructure. The multihost networking, however, does not work yet when using Mesos as scheduling backend.

Contrary to Docker Swarm, Marathon does not support Docker Compose. To deploy a Big Data pipeline on Mesos using Marathon, one needs to start each container in the pipeline manually via the Marathon GUI or REST API. Also, the containers need to be connected at another level than the Docker layer, for example using service discovery or by DNS names, as Marathon does not support Docker container linking or networking. This approach is more cumbersome and less maintainable than using Docker Compose for pipeline deployment.

3. Big Data pipeline components

This section describes the technologies that will be used in the application pipelines on top of the BDE platform. It is nearly impossible to describe and assess each Big Data technology that is available. We have made a selection of components based on the traction they have in the community and an informal evaluation of their technical qualities.

The selection is, however, by no means fixed. We can imagine components with interesting properties to arise and we hope to continuously improve the selection as we gain more experience.

3.1 Sharing data

In a Big Data application, multiple components need to work on a shared dataset. The components may run on different nodes, hence the data would need to be accessible on various nodes. This can be achieved by virtue of a distributed file system or a database. The most common way of sharing data is by use of a distributed file system, namely HDFS [21].

3.1.1 Distributed file systems

Distributed file systems are the distributed variant of a local file system. They aim to be transparent by providing the same interface and semantics as local files systems to access data. HDFS is a mature open-source distributed file system that has proven to cope with Big Data. It clusters and replicates the data across various nodes for optimal access, but adds a set of limitations on the operations which you can perform on the files. It implements a write-once read-many access model, i.e. data can only be added, not modified.

Hadoop utilizes a traditional master-slave architecture that makes use of commodity hardware configured as a cluster, where each server possesses inexpensive internal disk drives. The data in the Hadoop file system is broken down into blocks and spread throughout a cluster. Once that happens, tasks can be carried out on the smaller subsets of data that may make up a very large dataset overall, thus accomplishing the type of scalability needed for Big Data processing.

The open-source nature of HDFS overcomes the need for high-priced specialized storage solutions. This in combination with the ability to carry out some degree of automatic redundancy and failover, makes HDFS popular for modern businesses that look for data warehouse batch analytics solutions.

In more detail, HDFS offers the following benefits over other distributed file systems:

- *Low cost:* HDFS is free under the Apache license. Being open-source and able to work on commodity hardware makes Hadoop an almost no-cost solution. This cost advantage lets organizations store and process orders of magnitude more data per dollar. In Big Data deployments, the cost of storage often determines the viability of the system.
- *Data reliability:* Hadoop provides built-in redundancy and gracious failover. Hadoop requires almost no manual intervention for its automated redundancy and failover tactics. This overcomes the major bottleneck for businesses that cannot afford any downtime, as it guarantees to keep the data store up and operational.
- *Big Data capability:* The slogan for HDFS is its ability to deal with Big 'Google scale' Data with characteristics like velocity, variety, volume etc. HDFS can supply data

at a rate that results in faster batch processes and quicker answers to complex analytical questions.

- *Performance:* Commodity hardware suffices to support most workloads.
- *Portability:* The portability of HDFS overcomes the major bottleneck of data migration, which is a nightmare for data professionals. It also makes data available for multiple frameworks outside the Hadoop stack like Spark, Storm, Flink and many others.
- *Proven and tested technology:* Smart design is the easy part; the difficult part is hardening a system in real use cases. The only way you can prove a system is reliable is to run it for years against a variety of production applications at full scale. HDFS has been proven in thousands of different use cases and cluster sizes, from startups to Internet giants and governments. As a result thousands of issues have already been identified and fixed.

3.1.2 Databases

Roughly speaking, databases can be divided in two kinds: relational databases and NoSQL databases. As explained in [1], NoSQL databases are designed with Big Data needs in mind. They support dynamic schema design and offer increased flexibility, scalability and customization compared to relational databases.

NoSQL databases on their turn can be grouped into four categories: key-value stores, document stores, column-oriented stores and graph databases. [1] describes the characteristics and strengths of each of these categories in detail. But even within a single category, each technology might have a different focus. The selection of a suitable database therefore heavily depends on the data and the problem that needs to be solved. As a consequence, the BDE platform will support a range of database technologies from which one or more will be selected per pilot case.

3.2 Computational frameworks

3.2.1 Hadoop MapReduce

MapReduce is the programming paradigm from Hadoop that allows scalability across hundreds or thousands of nodes in a Hadoop cluster. Hadoop enables resilient, distributed processing of structured and unstructured data sets across commodity computer clusters, in which every node of the cluster possesses its own storage.

MapReduce serves two essential functions:

1. Map: it parcels out work to various nodes within the cluster
2. Reduce: it organizes and reduces the results from each node into a cohesive answer to a query

As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

MapReduce is mostly suited for tasks which can be decomposed into independent smaller subtasks and where the results can be combined to form an overall solution. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once an application is in the MapReduce form, scaling the application to run over hundreds, thousands,

or even tens of thousands of machines in a cluster is merely a configuration change. As a consequence, MapReduce can solve problems that used to take days, in hours or minutes with the traditional parallel processing techniques. This simple scalability has attracted the use of MapReduce model.

MapReduce moves computation processes to the data on HDFS and not the other way around. Hence processing tasks occur on the physical node where the data resides. This approach significantly reduces the network I/O patterns and contributes to Hadoop's processing speed.

However, it is worth noting that Hadoop works well for massive data sizes, but it is not perfectly suited for medium sized data, due to its architecture. Hadoop stores and reads data from disk assuming that the amount of data is too big to fit into memory. Although this is perfect for enormous sized data, it results in slower performance for data which could easily fit in memory. Performing iterative processing on large datasets means reading and writing from disk in each iteration, which can be done faster by storing the data in memory, if possible.

3.2.2 Apache Spark

Compared to Hadoop MapReduce, Spark provides a faster and more general data processing platform. Spark lets you run programs up to 100x faster in memory, or 10x faster on disk, than Hadoop. Recently, Spark took over Hadoop by completing the 100 TB Daytona Gray Sort contest three times faster on one tenth of the number of machines and it also became the fastest open-source engine for sorting a petabyte [22].

Spark makes it possible to write code more quickly by providing over 80 high-level operators. Another important aspect when learning how to use Apache Spark, is the interactive shell (REPL) which is provided out-of-the-box. Using REPL, one can test the outcome of each line of code without being required to first write all code and execute the entire job. The path to working code is thus much shorter and ad-hoc data analysis is made possible.

Additional key features of Spark include:

1. APIs in Scala, Java, and Python, with support for other languages (such as R) on the way.
2. Integration with the Hadoop ecosystem and data sources (HDFS, Hive, HBase, Cassandra, etc.).
3. Standalone installation or on a cluster managed by YARN or Mesos.
4. Compatible with Docker.

Spark Core is the base engine for large-scale parallel and distributed data processing. It is responsible for:

1. memory management and fault recovery
2. scheduling, distributing and monitoring jobs on a cluster
3. interaction with storage systems

Spark introduces the concept of an RDD, an immutable fault-tolerant, distributed collection of objects that can be operated on in parallel. This (mostly) in-memory data structure gives the power to Spark's functional programming paradigm. It is capable to perform large batch calculations by pinning memory. An RDD can contain any type of object and is created by loading an external dataset or by distributing a collection from the driver program.

RDDs support two types of operations:

1. *Transformations*: operations that are performed on an RDD and which yield a new RDD containing the result.
2. *Actions*: operations that return a value after running a computation on an RDD.

By default, each transformed RDD may be recomputed when an action is run on it. However, RDDs can be persisted in memory using the persist or cache method, in which case Spark will keep the elements around on the cluster for much faster access the next time it is queried.

Spark supports batch, stream, iterative and interactive processing. Spark streaming wraps data streams into mini-batches, i.e. it collects all data that arrives within a certain period of time and runs a regular batch program on the collected data. While the batch program is running, the data for the next mini-batch is collected. The user has control over data persistence, which implies that he must perform memory management manually for efficient resource handling.

Spark comes with Spark SQL, which internally has an optimizer (Catalyst) that produces logical and physical plans. This is the same approach most DBMS take. It receives a high-level declarative query specified in Spark SQL DSL or in SQL, optimizes it, and then executes it. This is not at the foundation level as in Flink. Spark also presents an efficient and tested graph processing library named GraphX and offers multiple mature libraries for machine learning.

3.2.3 Apache Flink

Apache Flink is a quickly growing top-level Apache project being developed in Berlin. It is an innovative Big Data framework optimized for cyclic or iterative processing on Big Data by using iterative transformations on collections. This is achieved by an optimization of join algorithms, operator chaining and reuse of partitioning and sorting. At the same time, Flink is also a strong tool for batch processing.

Regarding streaming, Flink processes data streams as true streams, i.e. data elements are immediately pipelined through a streaming program as soon as they arrive. This allows to perform flexible window operations on streams and distinguishes Flink from Spark, which performs microstreaming.

Similar to Spark, Flink also supports both Java and Scala functional APIs. However, a wonderful and novel approach offered exclusively by Flink is to allow users to program at various levels of abstractions. This ranges from Meteor (highest), over Pact (middle), down to Neophele (lowest).

Another impressive feature of Flink is the ability to optimize user-created code. The Flink optimizer analyzes the code submitted to the cluster and performs code optimization by analyzing the user's map and reduce functions. The optimizer produces what it thinks is the best pipeline for running on that particular setup. This may be different if the cluster is larger or smaller.

Flink is built to be a good YARN citizen (which Spark has not quite achieved yet), and it can run existing MapReduce jobs directly on its execution engine. Flink also works on Hortonworks' Tez runtime, where it sacrifices some performance for the scalability that Tez can provide. Flink takes the approach that a cluster should manage itself rather than require a heavy dose of manual tuning. By managing memory automatically, Flink almost eliminates the memory spikes you often see on Spark clusters. When compared to Spark, Flink does not yet support SQL access, as Spark SQL does.

With Flink, you get all you need (batch, iterative and true stream processing) in one framework, without the overhead of programming and maintaining a separate cluster with a different API. Overall, Flink can do what Spark does in the streaming in-memory sense and what MapReduce does in the batch sense, but it can do them all with some memory management, built-in optimizer, and transformation kicks that might be just enough to upend the open-source data processing hierarchy. Combined with the high scalability of Flink, these features promise some powerful potential for optimization such that the data scientist can focus on what to compute without spending too much time on development.

3.2.4 Comparison

We have evaluated three of the most popular Big Data computational frameworks in this section. Table 1 presents the evaluation results based on the criteria that seem important for the upcoming pilot cases of the societal challenges.

Table 1: Comparison of Big Data computational frameworks

	Hadoop MapReduce	**Apache Spark**	**Apache Flink**
Processing type	Batch	Batch Micro Iterations Streaming (micro batch)	Batch True Iterations True Streaming
Data location	On Disk	In Memory / Spills	In Memory / Spills
Memory management	N/A	No	Efficient
Tried and tested	Yes	Yes	New, In progress
Mesos framework available	Yes	Yes	No
Compatible with Docker	To be investigated	Yes	Yes
General purpose	No	Yes	Yes

3.3 Message Passing

Message passing is a central technique in the design of any complex system of volatile software components, where the nature and number of recipients of a data message and the result of an arbitrary calculation, f(x) in general, is not known. The said software components do not adhere to the simplest form of message passing that is a direct method call of another software component; instead, they put their results into a centralized, dedicated result bucket, a Message Broker, that follows the Publish-Subscribe or the Point-to-Point model [23]. By nature of the Lambda Architecture and its conceptual distinction between different layers, the Publish-Subscribe model is considered more appropriate. In general the calculation model shifts from f(x) to f(consume(message)).

From the many Message Brokers that are available [24], RabbitMQ [25], Apache ActiveMQ [26] and Apache Kafka [27] will be considered here.

RabbitMQ is written in the Erlang Programming Language by Pivotal. Using the Erlang Programming Language means that concurrent and distributed computing is supported at its core. Client libraries in different programming languages are available. The STOMP messaging protocol [28] is supported by use of a plugin. AMPQ 1.0 [29] support is in planning [30], but previous versions of the protocol are already implemented [31]. A web interface is available. RabbitMQ uses its own implementation of a message store as a persistence layer [32].

ActiveMQ, an Apache Software Foundation project, implements the Java Message Service specification. From version 5.6 on, the STOMP protocol is supported, and from version 5.8 on, the AMPQ protocol is supported. From version 5.9 on, a replicated LevelDB [33] serves as the backend. It should be mentioned that HornetQ's [34] code base has been donated to Apache and is co-developed as Apache Artemis [35], an ActiveMQ subproject as of June 2015.

Apache Kafka, originating from a LinkedIn in-house project, is now a top level Apache project. Written in the Java Programming Language, it stores messages in flat files. One of the advantages, besides the very good performance with respect to message throughput, is that consumers can read messages from a given offset. That means that failure recovery can be started from the last known message and doesn't have to start from scratch. Messages can be kept for only as long as they are needed. Messages can be consumed via Java 8 Streams, which keeps memory consumption also low on the client side. It is important to note that Apache Kafka doesn't implement any of the existing message passing protocols as of writing [36].

Because Apache Kafka is widely used in Big Data applications nowadays, we will initially focus on this technology for message passing in the BDE platform. However, the elaboration of the pilot cases may reveal that another messaging passing technology is more suitable.

4. Monitoring the Platform

Monitoring is a generic term that covers a broad range. In our assessment we distinguish between resource monitoring and status monitoring. Resource monitoring allows to follow up the health of a server or a component in the platform while status monitoring offers insight in the status of a specific application.

Most components that will be used in the BDE platform already provide an interface by means of a CLI or a web UI where one can monitor the state of that component. The aim of the BDE platform, however, is to provide a uniform monitoring interface to the user where one can follow up the entire platform. This section describes several open-source tools that are available to this end. Commercial solutions such as Datadog [37] or New Relic [38] are not considered in this assessment.

4.1 Resource Monitoring

As said, resource monitoring allows to follow up the health of a server or a component in the platform. Four key metrics are typically used to this end: CPU usage, memory usage, network I/O and disk utilization. Monitoring tools may also offer the feature to send notifications and alerts to an administrator if a component fails or exceeds a predetermined threshold.

4.1.1 Docker Built-In Resource Monitoring

Docker provides the command docker stats <container_name> which will render a live display of key metrics for a given container. A sample output for a Redis container named my-redis is:

```
CONTAINER      CPU %      MEM USAGE/LIMIT     MEM %       NET I/O
my-redis       0.14%      8.684 MB/8.407 GB   0.10%       0 B/0 B
```

A more extensive set of metrics can be retrieved through the Docker REST API [39] by sending a GET request to /containers/<container_name>/stats. A sample response for a Redis container is:

```
{
  "read" : "2015-01-08T22:57:31.547920715Z",
  "networks": {
     "eth0": {
       "rx_bytes": 5338,
       "rx_dropped": 0,
       "rx_errors": 0,
       "rx_packets": 36,
       "tx_bytes": 648,
       "tx_dropped": 0,
       "tx_errors": 0,
       "tx_packets": 8
     }
  },
  "memory_stats" : {
   "stats" : {
     "total_pgmajfault" : 0,
     "cache" : 0,
     "mapped_file" : 0,
     "total_inactive_file" : 0,
     ...
```

```
   },
   "max_usage" : 6651904,
   "usage" : 6537216,
   "failcnt" : 0,
   "limit" : 67108864
  },
  "blkio_stats" : {},
  "cpu_stats" : {
   "cpu_usage" : {
    "percpu_usage" : [
     16970827,
     1839451,
     7107380,
     10571290
    ],
    "usage_in_usermode" : 10000000,
    "total_usage" : 36488948,
    "usage_in_kernelmode" : 20000000
   },
   "system_cpu_usage" : 20091722000000000,
   "throttling_data" : {}
  }
}
```


More information on the available metrics can be found in [40].

The Docker built-in resource monitoring is a great tool for troubleshooting and debugging. Yet, it is limited as a monitoring solution, because it only reports real-time metrics and lacks a GUI. That is the reason why the Docker stats API is often used to feed container resource information in existing monitoring solutions as explained in the following sections.

4.1.2 cAdvisor

cAdvisor [41], abbreviation for Container Advisor, is a monitoring tool developed by Google. It is a running daemon that collects, aggregates and processes information about running containers. cAdvisor has native support for Docker containers. cAdvisor exposes a web UI which displays live information on running processes, CPU usage, memory usage, network I/O and disk utilization (see Figure 5). The metrics can be displayed machine-wide or per container.

Figure 5: cAdvisor web UI visualizing CPU usage metrics

cAdvisor only displays real-time information. By default, the metrics are not stored so cAdvisor cannot be used to display metrics over a time range other than real-time. However, cAdvisor supports exporting statistics to InfluxDB [42] or as Prometheus [43] metrics. These tools are described in the following sections.

4.1.3 InfluxDB and Grafana

InfluxDB is an open-source distributed time series database, i.e. the database is optimized to handle arrays of numbers indexed by time-like metrics. cAdvisor supports exporting its metrics to InfluxDB. Grafana [44], on the other hand, is an open-source application for visualizing large-scale measurement data. It can run queries against the InfluxDB and chart the query results in a nice layout. Next to InfluxDB, Grafana also supports Graphite [45] and OpenTSDB [46].

Setting up InfluxDB and Grafana on top of cAdvisor allows to visualize the metrics gathered by cAdvisor in nice charts over any time range. [47] describes step-by-step how to setup this monitoring stack including cAdvisor, InfluxDB and Grafana using Docker Compose. Figure 6 shows some of the possibilities Grafana offers for creating graphs.

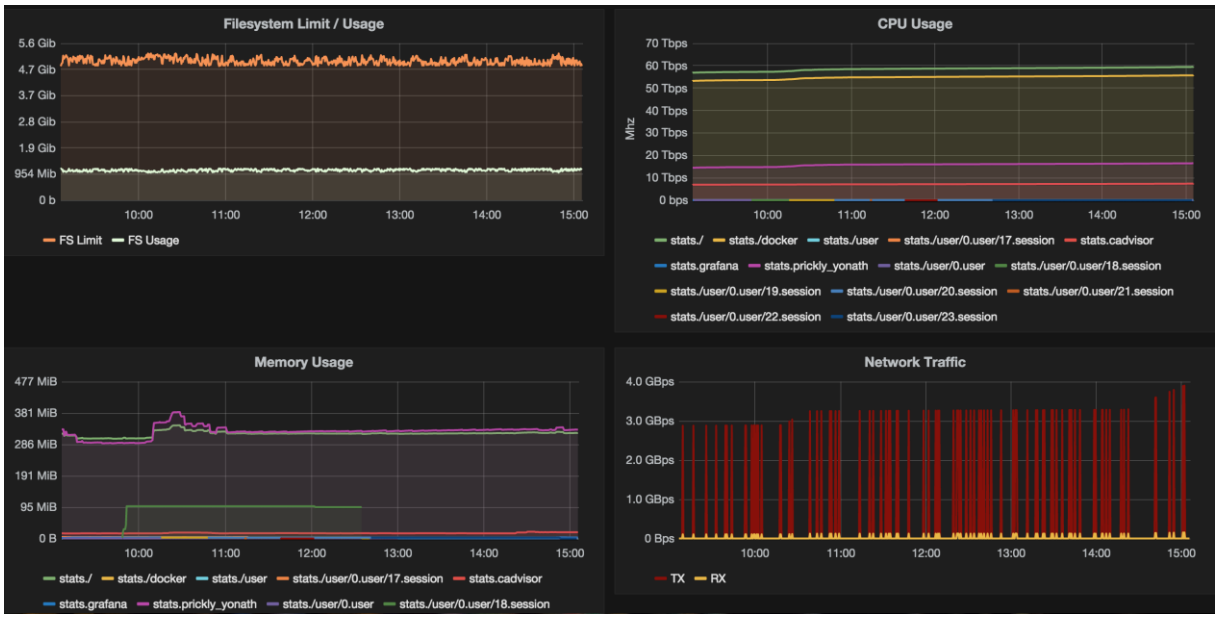

Figure 6: Grafana dashboard visualizing disk, CPU and memory usage and network I/O[1]

4.1.4 Prometheus

Prometheus is an open-source service monitoring system and a time series database. The database implements a highly dimensional data model, i.e. time series are identified by a metric name and a set of key-value pairs, also known as labels. Example labels are a container's name, image, environment etc. For instance:

```
container_memory_usage_bytes{env="prod",
id="23f731ee29ae12fef1ef6726e2fce60e5e37342ee9e35cb47e3c7a24422f9e88",
name="redis-database",image="redis:latest"}
```

Any combination of labels for the same metric name results in a separate time series.

[1] Source: https://www.brianchristner.io/how-to-setup-docker-monitoring/

The Prometheus' query language allows filtering and aggregating based on these dimensions. For example, if you are interested in all containers with a name starting with redis, you can use an expression like:

```
container_memory_usage_bytes{name=~"^redis"}
```

Or, to show the memory usage per name and zone, you can use an expression like:

```
avg(container_memory_usage_bytes{name=~"^consul"}) by (name,zone)
```

The result can be shown as a graph, viewed as tabular data or consumed by external systems via the HTTP API.

Prometheus offers an expression browser to investigate and visualize metrics ad-hoc. This might be useful during debugging. Prometheus also offers a dashboard, called PromDash, with an SQL backend to create persistent graphs. Figure 7 shows an example dashboard.

2

Figure 7: PromDash visualizing container statuses, CPU usage and disk usage

In contrast to Grafana, Prometheus also offers a notification system. Notifications and alerts can be configured using the query language. Alerts can be sent via methods, such as email and HipChat. However, Prometheus' notification system is still considered experimental.

[2] Source: https://github.com/vegasbrianc/prometheus

4.1.5 Comparison

Each of the tools described above has its advantages and disadvantages. Table 2 gives an overview of the main monitoring features each tool offers. The comparison is mainly based on [48].

Table 2: Feature overview of Docker resource monitoring solutions

	Docker Stats	cAdvisor	InfluxDB + Grafana	Prometheus
Ease of use	+	+	+/-	+/-
Metrics persistence			+	+
Scaling			+	
Client libraries			+	+
Alerting				+

As yet, we have not decided which tool will be used in the BDE platform. Further hands-on experience might help in this selection. It might even be the case that the optimal solution is a combination of several of these tools.

4.2 Status Monitoring

Status monitoring tools offer insight in the status of a specific application by means of log files. In this section we describe two generic solutions to consult these log files. However, it might arise from the pilot cases that custom application logging is required (e.g. to follow up the amount of items already processed by an application). The generic solutions might not suffice here but this will be assessed per pilot case.

4.2.1 Docker Built-In Logging

Docker provides the command docker logs <container_name> which will fetch the logs of a given container. Options are provided to follow the log (--follow=true), i.e. new output from the container's STDOUT and STDERR is continuously streamed, or to only show the logs since a specific timestamp (--since=<timestamp>). The logs command is also available on Docker Compose allowing to follow up the log files of multiple containers at once. Each line is then prefixed by the container's name.

Container logs can also be retrieved through the Docker REST API [39] by sending a GET request to /containers/<container_name>/logs.

4.2.2 ELK Stack

As explained before, the target of the BDE platform is to provide a uniform monitoring interface to the user. Connecting to every cluster node individually and executing the Docker logs command per container is cumbersome and not very user-friendly. The ELK stack [49], combining Elasticsearch, Logstash and Kibana, offers a solution by collecting the log files and presenting them in a unified interface to the user, as illustrated in Figure 8. Moreover, the integration of Elasticsearch, which is built on top of Lucene [50], provides the user with powerful full-text search capabilities and data analysis features.

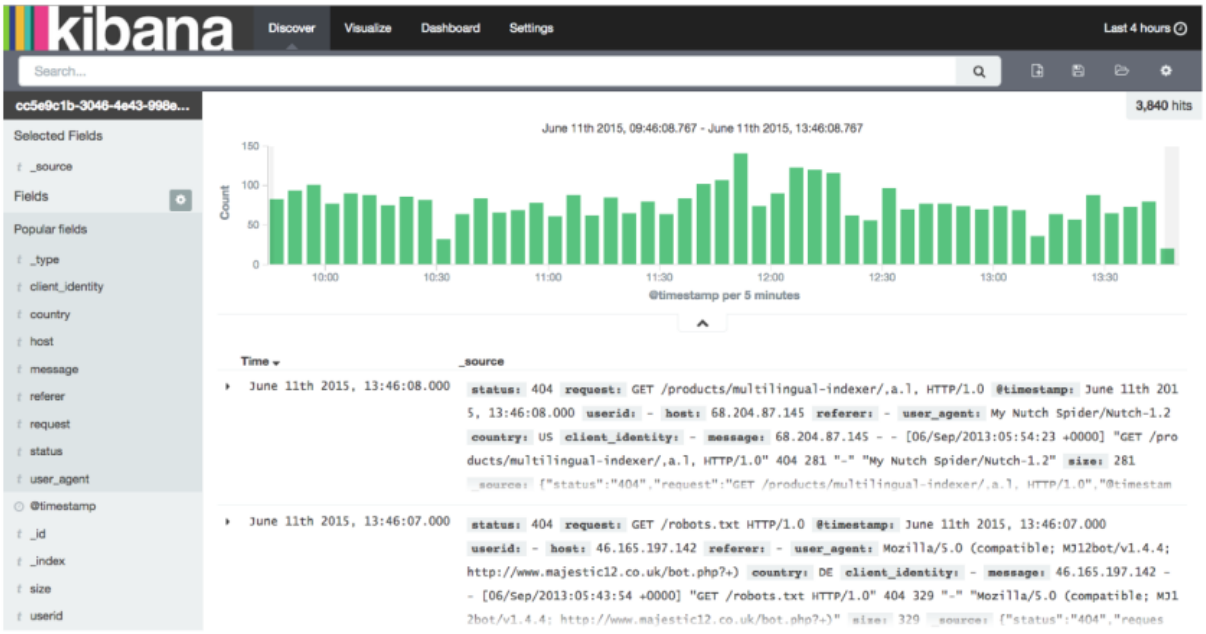

Figure 8: Visualizing log files using the ELK stack

The ELK stack is available through Docker Compose [51] so it is easy to deploy this stack on top of the BDE platform. Further investigation should reveal to what extent the ELK stack meets the need of the pilot cases and whether custom development will be required.

5. Deployment Interfaces

Ease of use is one of the main focus points of the BDE platform. It should be easy for an administrator to deploy the BDE platform and consequently to run Big Data pipelines. Hereby, a GUI might be of great help. When investigating the existing GUIs that are available, we can make a distinction between tools built on top of Docker and tools built on top of Mesos.

Docker Engine and Docker Swarm only offer a CLI to the user, but Docker is currently working hard on two GUI tools: Docker Tutum [52] and Docker Universal Control Plane [53]. Docker Tutum is focused on container deployment in the cloud so it only supports deployment on cloud providers like DigitalOcean, AWS, Microsoft Azure etc. Docker Universal Control Plane, on the other hand, can be used to deploy and manage Docker containers on premise. This tool, however, will only be available as a commercial product. Neither of these tools match the open-source requirement of the BDE platform.

Since the start of the Docker project, the Docker community has developed several GUI applications on top of the Docker API. Examples are DockerUI [54] and Shipyard [55]. These applications are a rather straightforward translation of CLI commands to a graphical interface. For example, a user can list images and containers, inspect a container, start/stop a container, view a container's logs, stats etc. Figure 9 shows a screenshot of a container inspection through DockerUI. DockerUI and Shipyard are very similar. Shipyard differs from DockerUI in its focus on Docker Swarm. As such, it can handle multiple nodes. One can, for example, list the available nodes in Docker Swarm through the Shipyard interface.

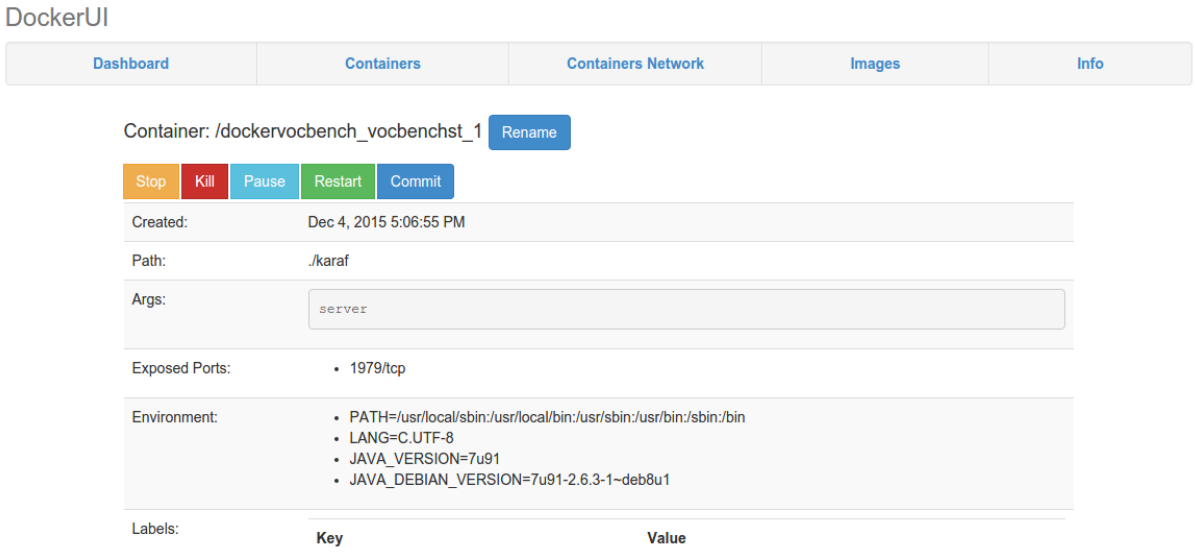

Figure 9: Docker container inspection through DockerUI

Marathon and Chronos, the frameworks available to deploy services on top of Mesos, both provide their own GUI. A user can list, inspect, start, stop, etc. applications through the GUI. However, as with the tools built on top of the Docker API, the screens are a rather straightforward translation of the underlying REST API that the frameworks offer. Figure 10 shows a screenshot to create a new Docker container through the Marathon GUI.

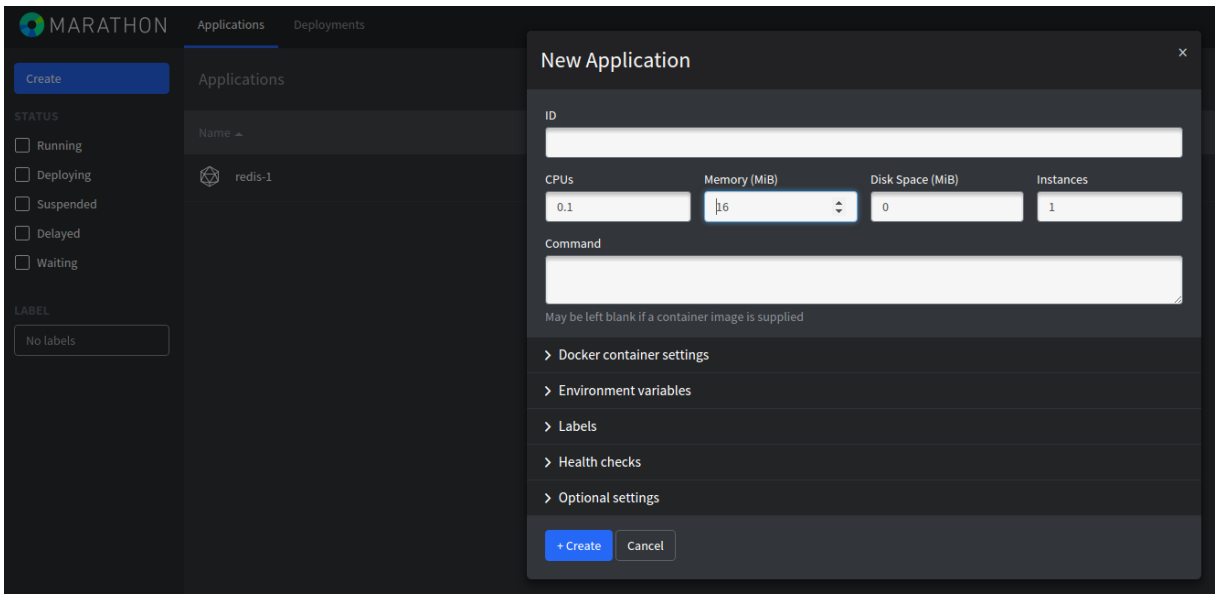

Figure 10: Docker container deployment through Marathon

Some people might prefer a graphical interface over a command line, but all in all, the GUI tools described above don't seem to have a notable added value for the BDE platform. Deploying a Big Data pipeline on top of the BDE platform using these tools remains complex. An ideal GUI for the BDE platform would provide the user with a list of available pipelines that can be executed. Running a pipeline should then be as simple as clicking one button in the interface. At the time of writing, no such tool has surfaced in the Docker ecosystem. We expect the community to build such tools, but we might need to alter them or develop them ourselves.

6. Conclusion

The BDE platform consists of a selected set of components, which are described in this deliverable. In the fast moving space of Big Data, and data management in general, this selection is by no means fixed. Most of the selected components are expected to be comparatively stable, but some of them may change in the future. As explained in [3] and further elaborated in this deliverable, the BDE base platform offers the flexibility to cope with this.

Although there is no follow-up scheduled on this deliverable in the course of the BDE project, we will continuously follow up the evolution of the technologies used in the BDE platform. This covers technologies used in the pipelines as well as in the base platform. The outcome may be new components entering the platform or diminishing support for existing components. Last but not least, the elaboration of the pilot cases will surely have a big impact on the technologies that will be integrated into the BDE platform.

In the upcoming months, we will gather experience with the main technologies and develop support in the form of base Docker images. Some topics that are not yet elaborated like monitoring, logging and GUIs will also be investigated in more depth. A first analysis revealed that tools are already available for monitoring and logging. Together with the various interfaces offered by the components themselves, monitoring and logging seems to boil down to an integration effort to form a uniform solution. Deployment and development GUIs, on the other hand, don't seem to be available off the shelf. The value of these tools heavily depends on the level of user experience that the BDE platform wants to reach. As ease of use is one of the cornerstones of the platform, it may be that only a custom developed GUI offers a satisfying user experience.

The current state of the BDE platform, including its components, is geared towards the future. Our current understanding of the components gives us a basis to work as summarized in Table 3. The Big Data space moves quickly. As we gain more hands-on experience and the community evolves, better options may arrive. We expect the platform to follow these trends, while ensuring that we keep it as productive as it can be.

Table 3: BDE platform components summary

	Category	Technology	DR[3]
Base platform	Cluster manager	Mesos	
	Scheduler	Marathon Docker Swarm	
Pipeline components	Computational frameworks	Apache Flink Apache Spark Apache Hadoop Apache Storm	v v v v
	Data storage	HDFS MongoDB HBase Cassandra OpenLink Virtuoso	(v) v v v v
	Message passing	Apache Kafka RabbitMQ	v v
	Data acquisition	Apache Flume	v
Monitoring	Resource monitoring	cAdvisor Prometheus InfluxDB/Grafana	v v v
	Status monitoring	ELK stack	v

[3] Dockerized component. I.e. not natively installed, but deployed in a Docker container.

7. References

[1] Big Data Europe, "WP3 Deliverable 3.1: Assessment on Application of Generic Data Management Technologies I," 2015.

[2] Big Data Europe, "WP3 Deliverable 3.2: Technical Requirements Specification and Big Data Integrator Architectural Design I," Big Data Europe, 2015.

[3] Big Data Europe, "WP3 Deliverable 3.3: Big Data Integrator Deployment and Component Interface Specification," Big Data Europe, 2015.

[4] "Apache Flink - Scalable Batch and Stream Data Processing," [Online]. Available: https://flink.apache.org/. [Accessed 23 12 2015].

[5] "Apache Spark - Lightning-Fast Cluster Computing," [Online]. Available: http://spark.apache.org/. [Accessed 23 12 2015].

[6] "Apache Mesos," [Online]. Available: http://mesos.apache.org/. [Accessed 23 12 2015].

[7] "Docker - Build, Ship, and Run Any App, Anywhere," [Online]. Available: https://www.docker.com. [Accessed 23 12 2015].

[8] "Apache Storm," [Online]. Available: http://storm.apache.org/. [Accessed 23 12 2015].

[9] "Docker Compose," [Online]. Available: https://www.docker.com/docker-compose. [Accessed 23 12 2015].

[10] "The Apache Cassandra Project," [Online]. Available: http://cassandra.apache.org/. [Accessed 23 12 2015].

[11] "Releases - docker/docker," [Online]. Available: https://github.com/docker/docker/releases. [Accessed 23 12 2015].

[12] "Releases - docker/compose," [Online]. Available: https://github.com/docker/compose/releases. [Accessed 23 12 2015].

[13] "Support for container aliases when joining a network - Issue #737 - docker/libnetwork - GitHub," [Online]. Available: https://github.com/docker/libnetwork/issues/737. [Accessed 23 12 2015].

[14] "Apache Hadoop 2.7.1 - Apache Hadoop NextGen MapReduce (YARN)," [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html. [Accessed 23 12 2015].

[15] "Docker Swarm," [Online]. Available: https://www.docker.com/docker-swarm. [Accessed 23 12 2015].

[16] "Marathon: A cluster-wide init and control system for services in cgroups or Docker containers," [Online]. Available: https://mesosphere.github.io/marathon. [Accessed 23 12 2015].

[17] "Chronos: Fault tolerant job scheduler for Mesos," [Online]. Available: https://mesos.github.io/chronos/. [Accessed 23 12 2015].

[18] "Software projects built on Mesos," [Online]. Available: http://mesos.apache.org/documentation/latest/frameworks/. [Accessed 23 12 2015].

[19] "Releases - docker/swarm," [Online]. Available: https://github.com/docker/swarm/releases. [Accessed 23 12 2015].

[20] "swarm/cluster/mesos at master · docker/swarm · GitHub," [Online]. Available: https://github.com/docker/swarm/tree/master/cluster/mesos. [Accessed 23 12 2015].

[21] "HDFS Architecture Guide," [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Accessed 23 12 2015].

[22] R. Xin, "Spark officially sets a new record in large-scale sorting | Databricks Blog," 05 11 2015. [Online]. Available: https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html. [Accessed 23 12 2015].

[23] "Difference between Point-To-Point and Publish/Subscribe JMS Messaging Models - Difference between Queue and Topic | Vertical Horizons," [Online]. Available: http://verticalhorizons.in/difference-between-point-to-point-and-publishsubscribe-jms-messaging-models/. [Accessed 23 12 2015].

[24] "Message broker," [Online]. Available: https://en.wikipedia.org/wiki/Message_broker. [Accessed 23 12 2015].

[25] "RabbitMQ - Messaging that just works," [Online]. Available: https://www.rabbitmq.com. [Accessed 23 12 2015].

[26] "Apache ActiveMQ -- Index," [Online]. Available: http://activemq.apache.org/. [Accessed 23 12 2015].

[27] "Apache Kafka," [Online]. Available: http://kafka.apache.org/. [Accessed 23 12 2015].

[28] "STOMP," [Online]. Available: https://stomp.github.io. [Accessed 23 12 2015].

[29] "Home | AMQP," [Online]. Available: https://www.amqp.org/. [Accessed 23 12 2015].

[30] "ActiveMQ, Qpid, HornetQ and RabbitMQ in Comparison," [Online]. Available: http://www.predic8.com/activemq-hornetq-rabbitmq-apollo-qpid-comparison.htm. [Accessed 23 12 2015].

[31] "RabbitMQ - AMQP 0-9-1 Complete Reference Guide," [Online]. Available: https://www.rabbitmq.com/amqp-0-9-1-reference.html. [Accessed 23 12 2015].

[32] "RabbitMQ » Blog Archive » RabbitMQ, backing stores, databases and disks - Messaging that just works," [Online]. Available: https://www.rabbitmq.com/blog/2011/01/20/rabbitmq-backing-stores-databases-and-disks/. [Accessed 23 12 2015].

[33] "Apache ActiveMQ -- Replicated LevelDB Store," [Online]. Available: http://activemq.apache.org/replicated-leveldb-store.html. [Accessed 23 12 2015].

[34] "HornetQ - putting the buzz in messaging - JBoss Community," [Online]. Available: http://hornetq.jboss.org/. [Accessed 23 12 2015].

[35] "ActiveMQ Artemis," [Online]. Available: https://activemq.apache.org/artemis/. [Accessed 23 12 2015].

[36] "A Guide To The Kafka Protocol," [Online]. Available: https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol. [Accessed 23 12 2015].

[37] "Cloud Monitoring as a Service | Datadog," [Online]. Available: https://www.datadoghq.com. [Accessed 24 12 2015].

[38] "Docker Container Monitoring | New Relic Integrations," [Online]. Available: http://newrelic.com/docker. [Accessed 24 12 2015].

[39] "Docker - Remote API v1.21," [Online]. Available: https://docs.docker.com/engine/reference/api/docker_remote_api_v1.21/. [Accessed 24 12 2015].

[40] "Docker - Runtime metrics," [Online]. Available: https://docs.docker.com/engine/articles/runmetrics/. [Accessed 24 12 2015].

[41] "google/cadvisor - GitHub," [Online]. Available: https://github.com/google/cadvisor. [Accessed 24 12 2015].

[42] "The Platform for Time-Series Data | InfluxData," [Online]. Available: https://influxdb.com/. [Accessed 24 12 2015].

[43] "Prometheus," [Online]. Available: http://prometheus.io/. [Accessed 24 12 2015].

[44] "Grafana - Graphite and InfluxDB Dashboard and graph composer," [Online]. Available: http://grafana.org/. [Accessed 24 12 2015].

[45] "Graphite - Scalable Realtime Graphing," [Online]. Available: http://graphite.wikidot.com/. [Accessed 24 12 2015].

[46] "OpenTSDB - A Distributed, Scalable Monitoring System," [Online]. Available: http://opentsdb.net/. [Accessed 24 12 2015].

[47] B. Christner, "How to setup Docker Monitoring," 20 05 2015. [Online]. Available: https://www.brianchristner.io/how-to-setup-docker-monitoring/. [Accessed 24 12 2015].

[48] B. Christner, "Docker Monitoring - DockerCon EU 2015," 17 11 2015. [Online]. Available: http://www.slideshare.net/Docker/docker-monitoring-55436594. [Accessed 24 12 2015].

[49] "Powering Data Search, Log Analysis, Analytics | Elastic," [Online]. Available: https://www.elastic.co/products. [Accessed 24 12 2015].

[50] "Apache Lucene - Welcome to Apache Lucene," [Online]. Available: https://lucene.apache.org/. [Accessed 24 12 2015].

[51] "deviantony/docker-elk - GitHub," [Online]. Available: https://github.com/deviantony/docker-elk. [Accessed 24 12 2015].

[52] "Docker Hosting – Run Docker Containers in any Cloud - Tutum," [Online]. Available: https://www.tutum.co. [Accessed 24 12 2015].

[53] "Container Management with Docker Universal Control Plane," [Online]. Available: https://www.docker.com/universal-control-plane. [Accessed 24 12 2015].

[54] "crosbymichael/dockerui," [Online]. Available: https://github.com/crosbymichael/dockerui. [Accessed 24 12 2015].

[55] "Shipyard," [Online]. Available: https://shipyard-project.com/. [Accessed 24 12 2015].