MOdel based coNtrol framework for Site-wide OptimizatiON of data-intensive processes

D4.4 – Updated Big Data Storage and Analytics Platform

Deliverable ID	**D4.4**
Deliverable Title	**Updated Big Data Storage and Analytics Platform**
Work Package	**WP4 – Cross-sectorial Data Lab**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**2017-11-28**
Status	**Final**
Lead Editor	**PROB**
Main Contributors	**Vincent Bonnivard (PROB), Peter Bednar (TUK), Hassan Rasheed (FIT), Dimitris Kalatzis (CERTH), Jean Gaschler (CAP)**

Published by the MONSOON Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723650.

Document History

Version	Date	Author(s)	Description
0.1	10-10-2017	Vincent Bonnivard (PROB)	First Draft with TOC
0.2	24-10-2017	Peter Bednar (TUK)	Update of TOC and added notes
0.3	08-11-2017	Vincent Bonnivard (PROB)	Filled Python libraries sections
0.4	11-11-2017	Peter Bednar (TUK)	Chapter 4 about the deployment, formatting
0.41	14-11-2017	Jean Gaschler (CAP)	Contribution concerning MONSOON repositories
0.5	17-11-2017	Hassan Rasheed (FIT)	added Chapter on platform modularization
0.6	21-11-2017	Dimitris Kalatzis (CERTH)	Extended Python libraries sections
0.7	22-11-2017	Vincent Bonnivard (PROB)	Version ready for reviewing
1.0	28-11-2017	Vincent Bonnivard (PROB)	Final version

Internal Review History

Version	Review Date	Reviewed by	Summary of comments
0.7	23-11-2017	Marco DIAS (GLN)	Fullyaccepted
0.7	23-11-2017	Massimo De Pieri, Gian Luca Baldo (LCEN)	Approved with minor comments and formal amendments

Table of Contents

1 Executive Summary

This document describes the distributed platform for Big Data Storage and Analytics that provides resources and functionalities for storage batch, and real-time processing of the big data. The platform combines and orchestrates existing technologies from Big Data and analytic landscape and sets a distributed and scalable run-time infrastructure for the data analytics methods developed in the project. The high-level architecture with its provided interfaces for cross-sectorial collaboration is presented. The solutions and technology options available for each logical component of the architecture are briefly explained.

The development approach in MONSOON is iterative and incremental, including three prototyping cycles (ramp-up phase, period 1, and period 2). The physical architecture of the Big Data Storage and Analytics Platform as realized from high-level architecture and the chosen technology stack for a ramp-up phase deployment is thoroughly explained. The platform and its components have been deployed in TUK environment to provide simplified storage infrastructure for the collected monitoring data from both aluminium and plastic domains. The deployment setup and configuration of the physical and virtual infrastructures are also presented.

2 Introduction

2.1 Purpose, context and scope of this deliverable

In the context of MONSOON work package structure, Task 4.2 (Big Data Storage and Analytics Platform) deals with the setup and deployment of distributed and scalable run-time infrastructure for the data analytics methods developed in project. It provides main integration interfaces for cross-sectorial collaboration between the site operational platform and cloud Data lab platform and programming interfaces for implementation of the data mining processes.

Ramp-up phase is the first iteration of the MONSOON development cycle. We shall discuss initial version of the Big Data Storage and Analytics Platform that documents the high-level architecture, setup and deployment of the platform. The detailed description of the overall platform architecture and its components is thoroughly described in Deliverable D2.5 – Initial Requirements and Architecture Specification. The Big Data Storage and Analytics Platform is subject to change and evolve in next iterations as a result of architecture specification updates.

The document is divided as follows. Section 3 covers the architecture of Big Data Storage and Analytics platform. It also describes the high-level architecture components, exposed interfaces and the candidate technologies which can be used to realize the platform. Chapter 4 presents the platform architecture as adapted for the ramp-up phase. Chapter 5 talks about the future deployment plans beyond ramp-up phase which includes discussion about the big data platform providers.

3 Platform Architecture and Components

The platform architecture was specified in deliverable D2.7 and D4.3. Deliverable D2.7 described how the architecture is divided into components and how components interact in order to implement functionalities specified for the Data Lab platform. Deliverable D4.3 then specified which technology can be used for the implementation of the components and which components were deployed for ramp-up phase.

In this update, we have not introduced any new component. The main changes and extensions are related to the implementation of the particular existing components.Changes in this update cover the following main objectives:

- Improve platform modularity in order to achieve gradual scalability and allow to deploy simplified architecture.
- Extend data processing with more convenient non-distributed environment, which will allow data scientists to start with the common technologies and move to more complex distributed environment for the scalability.

Details are described in the following sub-chapters for each component. The components which are not included in the following description were not changed.

3.1 Distributed File System

Distributed File System of the Data Lab platform is based on the HDFS (Hadoop Distributed File System), which provides native integration with the distributed data processing frameworks such as Apache Spark. The main advantage of this integration is that HDFS splits data into large blocks and distributes them across nodes in a cluster. Distributed frameworks then transfer packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently, since it is not necessary to relocate data from one node to another one via network. Distributed data processing frameworks also provide operations for manipulation of the data over HDFS as the integrated part of their application programming interface, so the data scientists don't have to use any other software components to access the data.

In the case of non-distributed data processing in local environment (e.g. Python or R), local task has to access data stored on the HDFS using the network protocols and fetch the copy of the data to the local memory of the processing node. This can be implemented using the following options:

- Map HDFS as the local file system using the standard network storage protocols such as Network File System (NFS)
- Use client library, which provides API for the selected environment.

3.1.1 Map to local file system

Apache Hadoop project provides implementation of the NFS Gateway for HDFS, which allows clients to mount HDFS and interact with it through NFS (Network File System), as if it were part of their local file system. The gateway supports NFSv3. After mounting HDFS, a data scientist can browse the HDFS file systems through their local file systems on NFSv3 client-compatible operating systems, upload and download files between the HDFS file systems and their local file systems and stream data directly to HDFS through the mount point. (File append is supported, but random write is not supported, since HDFS is not the POSIX file system). The main advantage of this approach is that mapping is completely transparent to the data analytics code, and it is possible to change implementation of the underlying storage (for example to local file system or network area storage - NAS) without the modification of the code. This is particularly useful for scoring code of the predictive functions, since they are deployed in the Runtime Container environment, which doesn't provide HDFS storage.

The following schema (Figure 1) shows example of NFS Gateway deployment. NFS Gateway is service running on the same host server as HDFS Name node which manage metadata about the distributed file system. Data are stored on data nodes hosts (node1, node2, node3). Client code can run on the remote server (worker host) and read or write data from/to local file system. Worker host operating system maps remote NFS storage as the part of the local system, so worker code has access to local files or to the mounted NFS storage. Worker host is communicating with the master host through the standard NFS protocol (currently supported version is NFSv3). NFS Gateway then delegate reading or writing of data to Hadoop infrastructure and use native Hadoop RPC for exchanging data from data nodes. The NFS mapping is completely transparent for worker process.

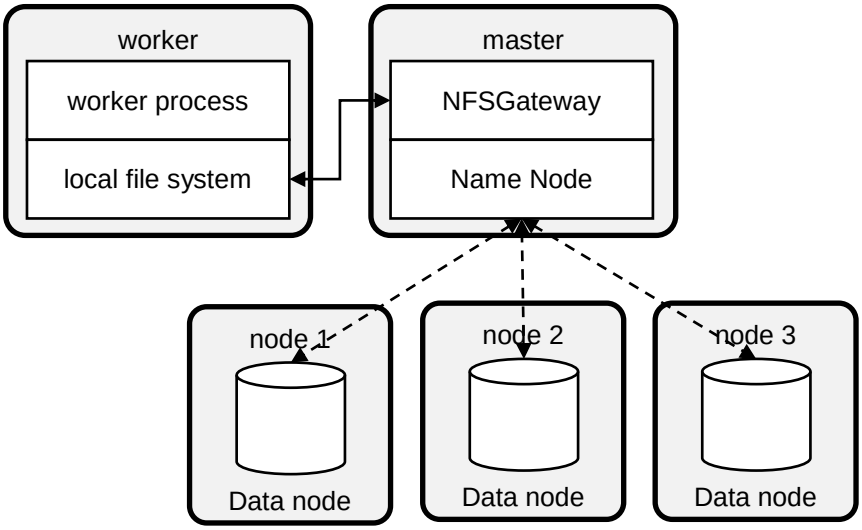

Figure 1 – Example of the deployment of HDFS to NFS mapping.

3.1.2 Use client library

Apache HDFS clients are connected to the distributed file system using the native RPC (Remote Procedure Call) protocol layered over the TCP/IP. Apache Hadoop project provides native client API in Java language. Besides the native RPC protocol, remote clients are usually communicating with the HDFS via standard REST interface – WebHDFS. WebHDFS is also the main communication protocol for exchanging of batch files between the site environment and Data Lab MONSOON platform. According to existing benchmarks, the performance of the RPC and REST interfaces are comparable, but REST interface provides better interoperability. REST interface can also be used outside of the Data Lab platform over the Apache Knox security proxy. The main disadvantage is that the data analytics code will be directly dependent on the API provided by the client library and underlying HDFS storage. The following table summarizes candidate client libraries for Python language.

Library	Protocol	Description
pywebhdfs	WebHDFS	WebHDFS client library implemented in Python language. Provides API for basic file and directory operations. Current version does not support kerberized cluster.
hdfscli	WebHDFS	Python bindings for the WebHDFS API,

		supporting both secure and insecure clusters. Additional functionalities supported by extensions include reading/writing Avro files directly from JSON and enabling fast loading/saving of pandas dataframes from/to HDFS.
libhdfs	Native RPC	This library is the internal part of the Hadoop distribution and provides reference implementation of the data protocol. It is implemented as the C language wrapper around the Java client, which can be further integrated in Python environment. Disadvantage is that Java environment has to be configured on client host machine.
libhdfs3	Native RPC	This library is the native C implementation of the protocol which is not dependent on the Java environment. It provides nearly the same C interface as libhdfs, so many Python wrappers allow to switch between libhdfs and libhdfs3 drivers.
snakebite	Native RPC	This library is the pure Python implementation of the Hadoop RPC protocol. Additionally, it provides wrapper around the Hadoop mini cluster, which can be used to setup environment for testing and development of components interacting with the Hadoop services. Snakebite requires python2 (python3 is not supported yet).

Table 1 – Overview of the Python HDFS client libraries.

3.2 Distributed Database

In the initial specification of the architecture in deliverable D4.3, we have described various options for the implementation of the Distributed Database component. These options can be divided to a) distributed databases based on the Hadoop environment and b) database engines. Distributed databases internally implement the storage mechanism which is usually layered over the distributed file system such as HDFS. Such implementation is directly dependent on the distributed storage.

On the other side, database engines implement query federation strategy where multiple files in different formats are directly processed by dedicated database "drivers" and engine combines the final results. In this way, it is possible to evaluate queries without the need to transform data to common format/schema, which can be problematic with very large dataset and require more computational resources. Apache Drill, Hive and Impala are three open-source database engines currently used in Big Data projects.
Such federated implementation is also more flexible regarding to the underlying storage, because drivers can implement interface to various storage implementation varying from the local file systems to distributed file system or structured/NoSQL databases. From this perspective, Apache Drill is the most flexible solution, since Hive and Impala are dependent on the HDFS.

Additionally to distributed implementations, it is also possible to provide SQL query functionality in the local data processing environment. For Python environment it is possible to leverage libraries and frameworks in Python Data Science Stack. The main library for the manipulation with the structured records in the tabular files or databases is the Pandas library. Pandas provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language. In particular, it offers data structures and operations for manipulating numerical tables and time series. SQL query functionality can be added by PandaSQL extensions, which allows data scientist to query pandas DataFrames using SQL-like syntax (supported SQL dialect is based on the SQL lite database).

3.3 Data Replication Service

In the current version, Data Replication Service is implemented using the WebHDFS interface directly provided by the HDFS system. Site client components are implemented as the Apache NifiWebHDFS connectors streaming files to the Data Lab platform through the security proxy implemented using the Apache Knox. One of the disadvantage of this implementation is that the site components are directly dependent on the WebHDFS protocol, which is specific for the HDFS system and it is not supported by other storage mechanisms. As aconsequence, it is not so straightforward to change internal Data Lab storage mechanism without influencing the site components.

In order to extend Data Replication Service with the new communication protocol or to add a new storage mechanism to the Data Lab platform, it is necessary to implement one of the following integration scenarios:

Implement WebHDFS adapter for the new storage – WebHDFS is the communication protocol based on the REST HTTP web service interface, so it is quite straightforward to implement interface adapter for the new storage. In this case it is not necessary to change site client configuration and adding of the new storage is completely transparent for the site components. It is also possible to leverage Apache Knox security proxy, since it directly supports proxying of the WebHDFS protocol.

Use standard network protocols for file exchange such as SCP or (S)FTP – Most of the standard protocols are already supported by the Apache Nifi, so for the site components, it is necessary just to stream data over the new connector for the new protocol. On the Data Lab site, communication protocol can be implemented by standard service of the hosting server (e.g. (S)FTP, SSH daemon etc.). One disadvantage is that Apache Knox security gateway doesn't support proxying of these protocols, so it will be difficult for example to integrate Hadoop authentication, and communication has to be secured/authenticated by the standard protocol extensions (e.g. SSL/TLS).

Use Apache Nifi web service interface – Since Data Lab platform additionally provides Apache Nifi infrastructure, it is also possible to directly exchange data between two Apache Nifi instances, one used as the data integration platform on the production site and one used as the data distribution platform on the Data Lab. Interface is based on the REST HTTP protocol, for which is also possible to implement security proxying plugin for Apache Knox.

3.4 Data Processing Framework

Initial implementation of the Data Processing Framework specified in the deliverable D4.3 was based on the distributed computation framework such as Apache Spark or Apache Flink. On one hand side, distributed frameworks allow to achieve high scalability for the processing of very large datasets. On the other hand, data scientists have to implement their data processing methods using restricted API of the distributed frameworks based on the functional programming paradigm. In order to simplify common data analytics tasks, we have extended Data Processing Framework with the conventional non-distributed frameworks commonly used for the data analytics in non-distributed environment. Such non-distributed environments

are usually based on the scripting language such as Python or R. We have identified that Python environment will be more convenient for Data Lab since it is compatible with most of the distributed frameworks.By combination of the non-distributed and distributed computation, we will allow data scientists to start processing medium to large datasets in non-distributed environment and then gradually scale-up to very large databases in distributed environment using the same scripting language (Python). The following sub-chapters describe non-distributed data processing environment in more details.

3.4.1 Python environment

The Python language is a powerful object-oriented programming language, widely used by the data science community. Some of itsmainfeatures are listedbelow:

- Cross-platform,
- Can be run on Max OS X, Windows, Linux and Unix,
- Comes with a large standard library,
- Thousands of external libraries, includingmany libraries dedicated to data analysis,
- Can be easily extended by adding new modules implemented in compiled languages,
- Free.

Its main programming-language features are:

- Large variety of basic data types: numbers, strings, lists, dictionaries,
- Object-oriented programming, with classes and inheritance,
- Grouping of code into modules and packages,
- Clean error handling, with exceptions raising and catching,
- Advanced programming features, such as generators and list comprehensions,
- Automatic memory management

The latest Python version is version 3.6. However, version 2.7 is still widely used by the community. The Data Processing Framework of the MONSOON Data Lab is compatible with both 2.7 and 3.6 versions.

Available Python libraries

The following libraries, widely used by data scientists, are available in the Development Environment of the Data Lab:

- **NumPy:** fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.NumPy can also be used as an efficient multi-dimensional container of generic data.
- **SciPy:** an open source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, Fast Fourier Transform, signal and image processing, Ordinary Differential Equation solvers and other tasks common in science and engineering.
- **Pandas:** library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. In particular, it offers data structures and operations for manipulating numerical tables and time series. Its main features are:
    - DataFrame object for data manipulation with integrated indexing.
    - Tools for reading and writing data between in-memory data structures and different file formats.
    - Data alignment and integrated handling of missing data.
    - Reshaping and pivoting of data sets.

- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generationand frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

The library is highly optimized for performance, with critical code paths written in Cython or C.

- **Scikit-learn:** machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- **XGBoost:** optimized distributed gradient boosting library providing the gradient boosting framework for different programming languages, including C++, Python and R. It isdesigned to be highly efficient, flexible and portable.It can be run on single machines, or on distributed processing frameworks such as Apache Hadoop and Apache Spark.
- **Matplotlib:** Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. It can be used to draw line plots, scatter plots, histograms, 3D plots... The plots can be exported to several formats, such as PNG, JPEG, PDF, SVG...
- **Seaborn:** a Python visualization library based on Matplotlib. It provides a high-level interface for drawing attractive statistical graphics. It is built on top of Matplotlib and includes support for NumPy and pandas data structures and statistical routines from SciPy. Some of the features that Seaborn offers are:

  - Several built-in themes that improve on the default matplotlib aesthetics
  - Tools for choosing color palettes to make beautiful plots that reveal patterns in your data
  - Functions for visualizing univariate and bivariate distributions or for comparing them between subsets of data
  - Tools that fit and visualize linear regression models for different kinds of independent and dependent variables
  - Functions that visualize matrices of data and use clustering algorithms to discover structure in those matrices
  - A function to plot statistical timeseries data with flexible estimation and representation of uncertainty around the estimate
  - High-level abstractions for structuring grids of plots that let you easily build complex visualizations

- **TSFresh:** *"Time Series Feature extraction based on scalable hypothesis tests"*. This package provides automatic time series features extraction and selection. Those features describe basic characteristics of the time series such as the number of peaks, the average or maximal value or more complex features such as the time reversal symmetry statistic.
- **NetworkX:** package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Its main features are:
  - Data structures for graphs, digraphs, and multigraphs
  - Many standard graph algorithms
  - Network structure and analysis measures
  - Generators for classic graphs, random graphs, and synthetic networks

  - Nodes can be "anything" (e.g., text, images, XML records)
  - Edges can hold arbitrary data (e.g., weights, time-series)

- **TensorFlow:** symbolic math library, used for a wide array of machine learning and other applications. Platform agnostic, supports both CPUs and (multiple) GPUs. Includes bindings for both Python and C++.
- **Keras:** machine learning front-end framework for Python, built on top of abstractions of other back-end libraries. Platform agnostic. Supports both CPUs and GPUs. Supported back-ends include *TensorFlow, Theano, Deeplearning4j, CNTK, MXNET*.
- **Pyrenn:** a recurrent neural network design toolkit for Python and Matlab. Platform agnostic. Dependencies include NumPy and Pandas.

The table below summarizes the available libraries and associated licenses:

Table 2 - List of Python packages and associated roles and licenses

Library name	Role	License
NumPy	Scientific computing	3-clause BSD license
SciPy	Scientific computing	3-clause BSD license
Pandas	Data structuring	3-clause BSD license
Scikit-learn	Machine learning	3-clause BSD license
XGBoost	Machine learning	Apache License, Version 2.0
Matplotlib	Plotting	Python Software Foundation license
Seaborn	Plotting	3-clause BSD license
TSFresh	Feature extraction	MIT license
NetworkX	Network manipulation	3-clause BSD license
Pyrenn	Recurrent neural network modeling	Creative Commons Attribution 4.0 International license
TensorFlow	Machine learning/scientific computing	Apache License, Version 2.0
Keras	Machine learning	MIT license

3.5 Functions Repositories/Algorithms and Models

The Figure 2below shows the different repositories used in the MONSOON project.

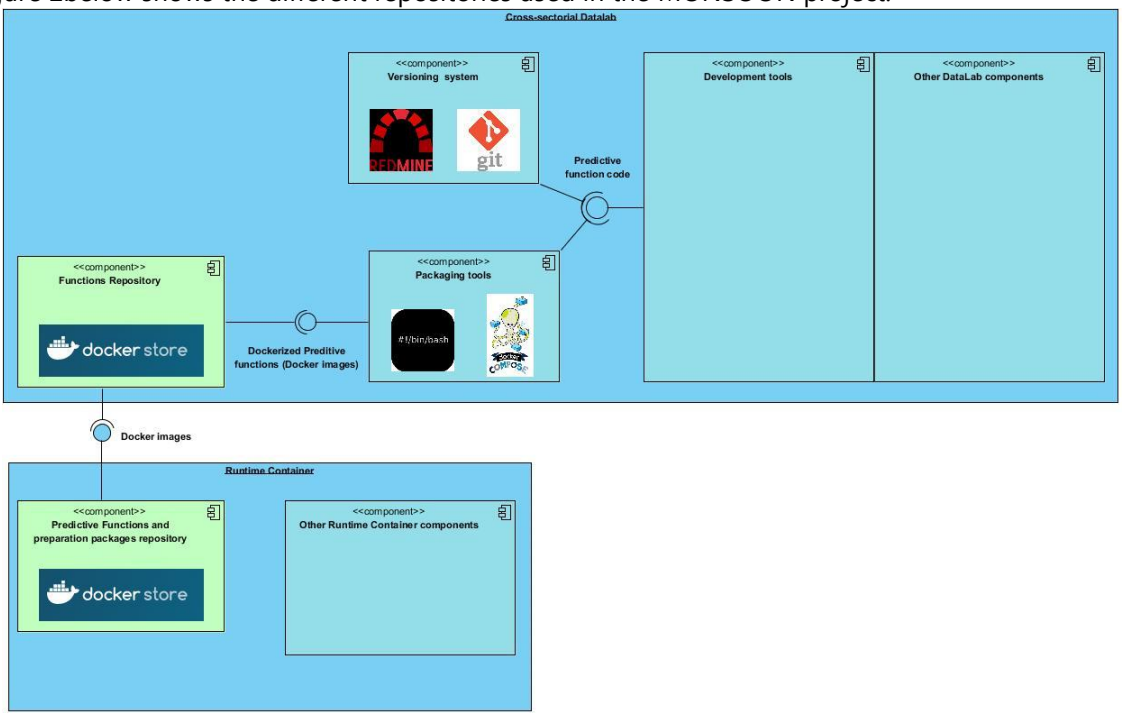

Figure 2 - Functions repositories

In the Datalab, data scientists build and test the Predictive Functions using the Development tools. The generated code is stored in a versioning system based on Redmine/Git. When a final version of a predictive function is ready, the code is packaged as a Docker image using tools like bash scripts or Docker specific scripts. All images are stored in what is called a "Functions repository".
Technically this is a special Docker container called Registry. The Registry is a stateless, highly scalable server side application that stores and permits an easy distributionof Docker images. The Registry is open-source, under the permissive Apache license. The main benefit is a tightly control where your images are being stored.

On the plant, in the site Runtime Container, another Registry will store needed Predictive Function Docker Images (transferred from the Functions Repository). Running Docker containers will be generated from these images.

4 Update of architectureDeployment

Deployment of the architecture is running on the physical servers and virtualized environment based on Microsoft Hyper-V. The physical server configuration and virtualization configuration was not changed and it is the same like to the ramp-up phase. It consists of 3 physical hosts and 3 network area storages interconnected in the cluster and placed in one rack. Virtualization divides the deployment to gateway server, master server, three worker/data nodes (node1, node2, node3) and security Kerberos infrastructure installed on kdc server. All servers are connected using the private virtualized network isolated from the physical network and Internet. Only gateway server has Internet connection through the TUK network and proxy Data Lab services to site clients.

4.1 Deployment infrastructure

Each virtualized server is running multiple services implementing Data Lab components, which were extended in this update. The services can be divided according to the related platform components. The diagram on Figure 3 shows how the services are currently deployed in the virtualized environment. The following components were already available in the Ramp-up phase:

- Hadoop distributed file system (HDFS)
- YARN resource manager
- Hive database
- Apache Spark distributed computation framework
- Apache Zeppelin development environment

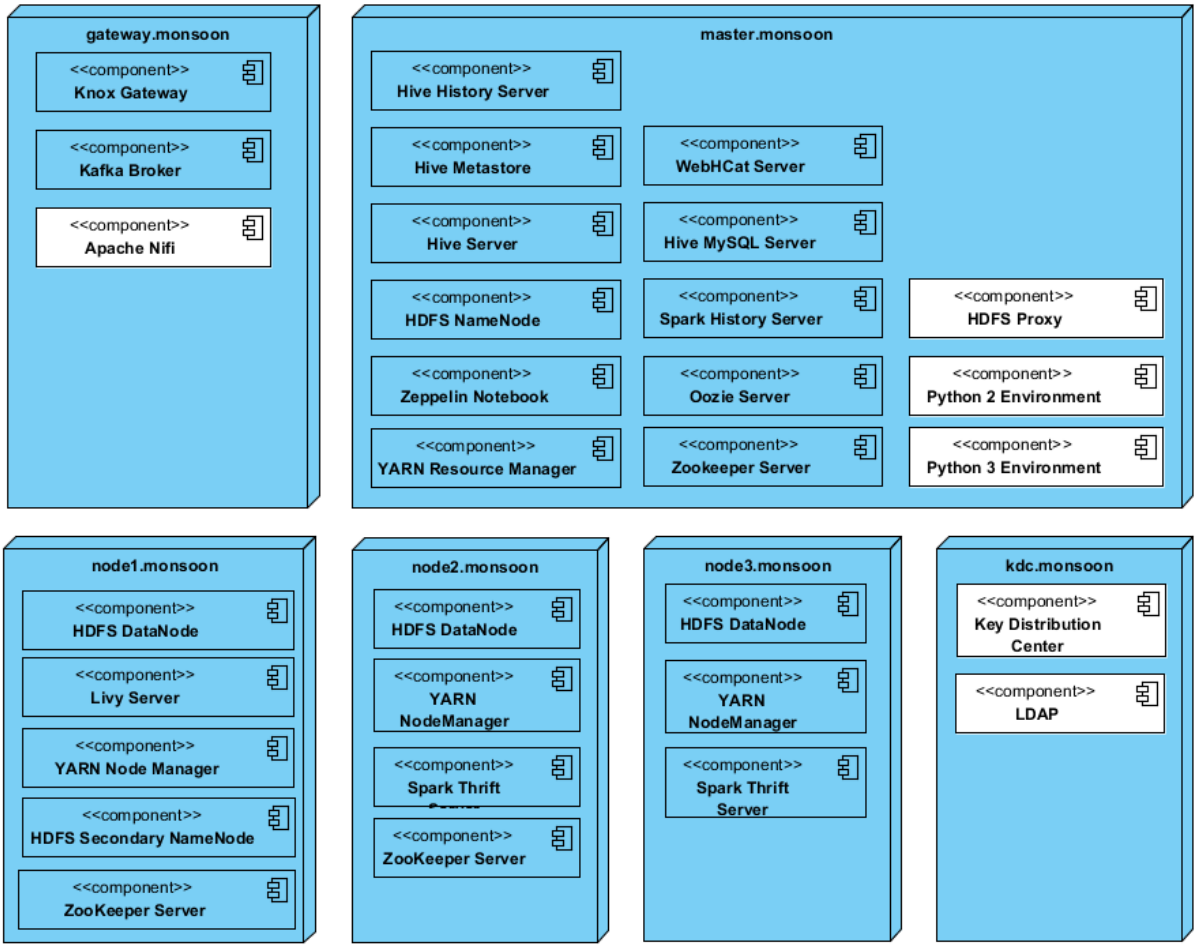

Figure 3 – Updated deployment diagram of the MONSOON Data Lab platform.

The following changes and extensions were implemented in this update:

- HDFS configuration was extended with the HDFS proxy, which provides HDFS to NFS mapping and allow to connect HDFS as the local system for all servers in the cluster.
- Security configuration was extended with the common LDAP server for management of the user identities. LDAP server is deployed on the secured security server which is hosting also the Kerberos Key Distribution Center service. LDAP is currently integrated with the Apache Zeppelin environment.
- Configuration of the Data replication service and Messaging service was extended with the Apache Nifi installation to prepare platform for better modularization (see the description in chapter 3.3).

Besides this, master server was extended with the local data analytics stack for Python language (64-bit environment with Python version 2 and 3 is supported). The stack consists of the following frameworks and libraries.

Library	Version	Notes
NumPy	1.13.3	
SciPy	0.19.1	
Pandas	0.20.3	
Scikit-learn	0.19.0	
XGBoost		
Matplotlib	2.1.0	Depends on graphwiz library for dot command. Current version is 2.26 integrated with pydot 1.2.3 Python package)
Seaborn	0.8.1	
TSFresh	0.11.0	
NetworkX	2.0	
pyrenn	0.1	Updated implementation with bug fixes and extensions from CERTH
tensorflow	1.3.0	CPU only, GPU computation is not supported by physical hosts of the cluster.
keras	2.0	with tensorflow backend

Table 3 – List of deployed libraries and frameworks for Python environment.

All libraries were installed and configured using the PyPi manager.

4.2 Modularization of the platform

As explained in previous chapter, that platform is established based on existing open-source Big Data technologies and frameworks.The current deployment infrastructure of the platform is also briefly explained where different services of the platform are installed and configured on six virtual hosts. Based on hands-on investigation and given experience, it is possible to select and orchestrate existing open source solutions to fully instantiate the platform. However, it turned out that deployment and configuration management is the most critical aspect in realizing such a platform where each of the deployed serviceand system will have a specific configuration (number of machines, IP addresses, disk types and space available, etc.) and that will have to be managed as well as maintained when updating to a new release.Moreover, it covers a set of

interrelated activities to operationalize the platform making sure that all relevant components or services operate correctly, that testing is possible, and that stability and reliability of the platform is maintained when additional features are added or updates are addressed.

On the other hand, it would be optimal to devise a uniform deployment strategy taking into account different deployment options for the platform such as on-premises, cloud/external provider or hybrid (combination of own & hosted). It is also learned that different demonstrative and use-case scenarios in both aluminium and plastic domain pose different infrastructure and data requirements. Hence, it is useful to define different deployment pipelines or modes for the platform where right setof platform services are deployed and orchestrated accordingly instead of full stack deployment.

Towards this goal, it is appropriate to modularize the platform in a form of basic buildingblocks with the objective of easingthe usage of common platform technologies and make integration with other services or applications easy. It will help in saving valuable time that is otherwise spent on exploring deployment methodologies. Nowadays application deployment is often done using virtual machines instead of native installing the application on a physical host. Use of a virtual machine isolates the application in a sandbox, a known and controlled environment. Virtual machines significantly accelerate the software development lifecycle, and enables substantial improvements in infrastructure cost and efficiency.

Our current deployment infrastructure uses Apache Ambari framework for the management, monitoring and provisioning of platform services. It provides the simplified deployment of the Hadoop services on cluster nodes, supports various environments (cloud, virtual, physical) and offer web-based, wizard-driven interface. The rationale behind using Ambari framework is that it reduces the tedious and time-consuming task of selecting from all open-source solutions a consistent set so that they can interplay. It greatly helps in maintaining overall central coordination for interplay and consistency of interfaces and protocols among platform services.Ambari also provides a set of monitoring capabilities with support of health status monitoring, metrics visualization and alerts. Ambari is highly extensible and customizable with Ambari Stacks for bringing custom services under management, and with Views for customizing the Web UI. Views offer a way to plug-in UI capabilities for custom visualization, management and monitoring features in Ambari Web.

We want to adapt a common deployment ground so that exploitation of the platform is as easy as possible, especially when it comes to run the integration testing of predictive functionsin the run-time container.Docker has been chosen for this purpose. Docker is an open source project that provides a simple packaging tool, which can be used for development as well as testing and production deployment. Dockerutilizes a client-server architecture and a remote API to manage and create Docker containers build upon Linux containers. It provides the capability to package an application with its runtimedependencies into a container. Containers run applications natively on the host machine's kernel. They have better performance characteristics than virtual machines that only get virtual access to host resources through a hypervisor. Containers can get native access, each one running in a discrete process, taking no more memory than any other executable.Furthermore, because Docker images consist of small layers, updating and rolling back components is an easy and rather lightweight operation.Docker brings in an API for container management, an image format and a possibility to use aregistry for sharing containers.A basic description of different Docker artefacts (image, container and registry) is given below.

An **image** is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files.

A **container** is a runtime instance of an imagethat actually executed. It runs completely isolated from the host environment, by default only accessing host files and ports if configured to do so.

A **dockerfile** is the main set of instructions a user can execute to form a new Docker image, via the "docker build" command. The instructions are executed in the order they are written.

A **base image**is an image that has no parent. Base images define the runtimeenvironment, packages and utilities necessary for containerized application to run. A base image is read-only, so any changes are reflected in the copied images stackedon top of it.

A **registry**is a repository of images. Registries are public or private repositoriesthat contain images available for download. Some registries allow users to uploadimages to make them available to others.

4.3 Base Docker Image for the Platform

Apache Ambari from a management point of view provides a single point for handling of the cluster operations, configuration and host controls. AmbariServer is usually running on a master node and expose numerous REST APIs. Ambari Agent is supposed to be installed on each of the nodes in a cluster and responsible to register itself with the Ambari Server through an agent interface. Once the Ambari Server and Agents have been installed and bootstrap on cluster nodes, a user can create cluster definition with the help of graphical cluster install wizard. This wizard helps in selecting the distribution repositories, packaging version, available services and the cluster node on which chosen services and components will be installed and deployed.Some configuration management is required to be done in all cluster nodes before Ambari installation.The required configuration, setup of Ambari Server and Agents,and bootstrapping of Ambari Agents on all cluster nodes can be automated with the help of Docker images and wrapper scripts.

A base Docker image for platform is created with specific host configuration as given below:

- switch-off ip-table firewall
- disable SELinux security module
- limit the number of user processes
- disable transparent huge pages
    - THPs is a Linux kernel abstraction layer that automates creation, managing, and using of huge memory pages (i.e. pages larger than 4096 bytes). It was detected that swappable THPs can potentially lead to performance degradation of database applications
- configure virtual memory swappiness
    - The swappiness parameter controls how often the Linux kernel moves processes out of physical memory onto the swap disk. This can lead to slower response times for system and applications if processes are too aggressively moved out of memory
- set noatime parameter for mounted disks
    - POSIX specification requires, that the system maintains time attribute for the last file access. It means that even read access to the file requires subsequent write operation to update file attributes. 'noatime' parameter means that access time will be updated only after the write operation and not for the read operations, which improves file I/O performance
- edit host names in a cluster
    - For each host edit /etc/hosts file and add names and IP addresses for your environment. The following example is for the private network configuration with the gateway, master, node1, node2, node3 and kdc hosts
- install and configure NTP protocol
    - Gateway server is synchronized with one of the NTP servers from http://www.pool.ntp.org/zone/europe pool. Nodes are synchronized with local gateway

This base image is based on CentOS version 7 and is configured to use "systemd" utility for doing the most of above host configuration.

Docker Image for Ambari Server
This image extends the base image and installs the Ambari Server with all required dependencies and libraries that it needed. Notably it performs the following steps:

- configureAmbari repo to download the available distributions from Hortonworks
- install postgresql database, add mysql and psql connectors to Ambari Server so it can be downloaded by other services
- add Ambari Shell to the image so users don't need the additional java image
- execute wrapper script that lookup the consul service and register its hostname with this local DNS service

Docker Image for Ambari Agent
This image is also based on the platform image and installs the AmbariAgent on a given container. The following tasks are being performed:

- configureAmbari repo to download the available distributions from Hortonworks
- execute init script that lookup the consul service and register its hostname with this local DNS service
- execute wrapper script to lookup Ambari Server hostname from Consul Service

A wrapper script is supplied to assist the user in starting the platform with given number of cluster nodes. All containers started are using bridge networking. At first, Consul2 key-value store is started for bookkeepingthe domain names of the cluster nodes.Next, a single container with Ambari Server is started.Afterwards, multiple containers are started as per user provided cluster size. Theyeach run Ambari Agent and can communicate via overlay networks. Once the server start-up and bootstrapping of the agent nodes is done, user can use the intuitive Cluster Install Wizard web interface (as provided by Ambari Server) to create and provision the Hadoop cluster. This process usually include confirming the list of hosts, assigning master, slave, and client components to configuring services, andinstalling, starting and testing the cluster.

Cluster Deployment via Ambari Blueprint
The initial inspiration for modularization of the platform was to be able to do the integration testing with other components and services of the MONSOON platform. A Docker based distribution of the big data and analytics platform simplifying the development and testing of the predictive functions. This distribution should facilitatethe instantiation of Ambaricluster quickly and without requiring user interactions.

Ambari Blueprint automates the process of cluster creation, configuration and provisioning.This allows to quickly create development and test clusters that are (possibly with the exception of size) identical to the production environment.Ambari Blueprints uses "JSON" format for the definition of a cluster.A blueprint document defines the service components to install on a given host group, the configurations to use, and which service stack to use.Acluster creation template (also JSON format)defines environment-specific host information and their mapping to a given host group. Ambari Shell utility (as part of AmbariServer container) is used to registercluster definition and creation JSONswith the Ambari Server using to the Ambari REST API.

Open Issues& Future Actions
As per Docker design and micro-service philosophy, only one process is supposed to be running in one Docker container. However, in our scenario, multiple processes are running in one container such as Ambari Agent (besides application process) that manages the configuration and life-cycle of the running component. Ambari Server interacts with this AmbariAgent for provisioning of the running component. We are unable to

deploy individual application packages as Docker containers as it is internally managed by Ambari Server.However, current approach is still very useful for testing where whole Big Data Storage & Analytics Platform is available as a standalone distribution.

We can deploy the platform services as Docker containers independent of the Apache Ambari but then we lose the benefits of central management, monitoring and provisioning of the whole platform being promised by Ambari framework. We need to investigate how different services of the platform can be deployed as standalone Docker containers while still using Ambari as a provisioning tool. It is also important for our vision of having different deployment pipelines according to use-case scenarios and applications.

At the moment, platform can be deployed on a single Docker Host with multiple Docker containers.As a next step, platform will be deployed onto multiple physical hosts. This can be achieved by Docker Swarm that is a cluster management tool turning a pool of Docker hosts into a single virtual host. The Swarm manager serves the same API as the Docker Engine. As such, using the same commands, platform can be deployed on a single machine or on a cluster.

5 Conclusions

This deliverable presented the updated version of the Big Data Storage and Analytics Platform of the MONSOON project. The platform combines and orchestrates existing technologies from Big Data and analytic landscape and sets a distributed and scalable run-time infrastructure for the data analytics methods developed in the projectThe physical architecture of the platform and the chosen technology stack have been precisely described; solutions and technology options available for each logical component have been presented.

The platform is now almost ready to be used and tested by the multiple developers of the MONSOON project. The choice of the distributed database and its installation, as well as the Dockerization of the platform (mainly for integration testing purposes), are the next steps that will finalize the deployment of the platform.

Acronyms

Acronym	Explanation
CDH	Cloudera Distribution for Hadoop
DFS	Distributed File System
FTP	File Transfer Protocol
HDFS	Hadoop Distributed File System
HDP	Hortonworks Data Platform
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
NIC	Network Interface Card
NFS	Network File System
NAS	Network Attached Storage
RAIDS	Redundant Array of Independent Disks
RDD	Resilient Distributed Dataset
RDBMS	Relational database management systems
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VLAN	Virtual Local Area Network

List of Figures

List of Tables