

Title:	Document Version:
D5.2 WiseGRID Big Data Cloud-based infrastructure Testing and Refinement	1.0

Project Number:	Project Acronym:	Project Title:
H2020-731205	WiseGRID	Wide scale demonstration of Integrated Solutions for European SmartGRID

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M21 (July 2018)	M21 (July 2018)	RD-PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Paul Lacatus	CRE	WP5

Authors (organisation):
Paul Lacatus (CRE), Catalin Chimirel (CRE), Mihai Sanduleac(CRE), Álvaro Nofuentes (ETRA), Alberto Zambrano(ETRA), Giuseppe Caruso (ENG), Leandro Lombardo(ENG), Stefan Meir(VS)

Abstract:
<p>In the context of WiseGRID WP5 this document will define the architecture of a cloud based Big Data platform and small scale demonstrator.</p> <p>Specifications of the WiseGRID Big Data Cloud-based infrastructure towards enabling: (i) Data Integration from a variety of heterogeneous data sources, (ii) Storage and Processing of huge data volumes in a highly efficient and effective manner, (iii) adaptation of Service-Oriented-Architectures (SOA) towards enhancing flexibility, scalability, re-usability, loose coupling, integration of a variety of functionalities, and (iv) Interoperability and Interconnection with a wide range of applications. Configuration and integration of the WiseGRID Big Data infrastructure with the WG IOP and the individual WiseGRID components.</p>

Keywords:
Big Data, Big Data platforms, data mining, data analytics, JSON, NoSQL, data storing, data processing,

Revision History

Revision	Date	Description	Author (Organisation)
V0.1	1.06.2018	New document	Paul Lacatus (CRE)
V0.2	27.06.2018	ToC refinement	Paul Lacatus (CRE)
V0.3	14.07.2018	Content added	Paul Lacatus (CRE)
V0.4	15.07.2018	Peer Review version without contributions	Paul Lacatus (CRE)
V0.5	25.07.2018	Review by ENG Leandro Lombardo and Catalin Chimirel	Paul Lacatus (CRE)
V0.6	27.07.2018	Release candidate RC1	Paul Lacatus (CRE)
V0.7	27.07.2018	Technical details on sections 9.1.3, 9.1.4, 9.1.5, 9.1.7	Alberto Zambrano (ETRA)
V1.0	30.07.2018	Version for submission	Paul Lacatus (CRE)

INDEX

1 INTRODUCTION	11
1.1 Purpose of the document	11
1.2 Scope of the document.....	11
1.3 Structure of the document	11
2 BIG DATA PLATFORM FOR WISEGRID PROJECT	12
2.1 BIG DATA DEFINITIONS.....	12
2.2 BIG DATA PLATFORMS FOR WISEGRID PROJECT DEFINED BY D5.1	15
3 BIG DATA ONLINE PLATFORM	16
3.1 BIG DATA MONGODB CLUSTER STRUCTURE	16
3.2 BIG DATA MONGODB DEMONSTRATOR STRUCTURE	17
4 BIG DATA OFFLINE PLATFORM	18
4.1 BIG DATA HADOOP CLUSTER STRUCTURE.....	19
4.2 BIG DATA HADOOP ARCHITECTURE ADAPTED FOR WISEGRID PROJECT DEMONSTRATOR	20
5 SINGLE BOARD COMPUTERS FOR WISEGRID DEMONSTRATOR	22
5.1 WHAT ARE SINGLE BOARD COMPUTERS	22
5.2 SINGLE BOARD COMPUTERS ANALYZED FOR WISEGRID DEMONSTRATOR	22
5.2.1 Raspberry Pi 3B+.....	23
5.2.2 Odroid XU4	24
5.2.3 Odroid C2.....	27
6 INSTALLATION OF THE MONGODB CLUSTER	31
6.1 PHYSICAL INSTALLATION OF THE Odroid C2 SBC FOR RACK MOUNTING.....	31
6.1.1 3D PRINTED SUPORTS FOR ODROID C2.....	32
6.1.2 POWER SUPPLY OF THE ODROID C2 SBCs	35
6.1.3 VENTILATION	36
6.2 INSTALLATION OF UBUNTU LINUX OPERATING SYSTEM	36
6.2.1 Defining the IP address system	36

6.2.2	Preparation of the nodes for cluster configuration.....	38
6.3	INSTALLATION OF MANAGEMENT SOFTWARE: WEBMIN	40
6.4	INSTALLATION OF THE MONGODB ENTERPRISE.....	42
6.5	INSTALLATION OF MONGODB CLUSTER	44
6.5.1	Setting of configuration servers	45
6.5.2	Activation of config servers	47
6.5.3	Setting of replicated shards.....	49
6.5.4	Activation of replica shards servers.....	54
6.5.5	Configuration and activation of arbiter instances	59
6.5.6	Configuration and activation of application routers	60
6.5.7	Final settings and online connection	65
7	INSTALLATION OF THE APACHE HADOOP CLUSTER	66
7.1	PHYSICAL INSTALLATION OF THE Odroid XU4 SBC FOR RACK MOUNTING	66
7.1.1	3D PRINTED SUPORTS FOR ODROID XU4	66
7.1.2	POWER SUPPLY OF THE ODROID XU4 SBCs.....	68
7.1.3	VENTILATION	69
7.2	INSTALLATION OF UBUNTU LINUX OPERATING SYSTEM	69
7.2.1	Defining the IP address system	69
7.2.2	Preparation of the nodes for cluster configuration.....	70
7.3	INSTALLATION OF MANAGEMENT SOFTWARE, DISTRIBUTED cli SOFTWARE	73
7.4	INSTALLATION OF THE HADOOP SOFTWARE	74
7.4.1	Installing Java.....	74
7.4.2	Preparing the user, groups and rights for installing Hadoop	74
7.4.3	Installing and deploying Hadoop software	78
7.5	ACTIVATION OF HADOOP CLUSTER	85
7.6	INSTALLATION OF APACHE SPARK SOFTWARE.....	86
7.6.1	Prerequisites to SPARK installation	86
7.6.2	Installation of Apache SPARK software	87
7.6.3	Deployment of SPARK software and settings on all slaves	88
7.6.4	Activation of SPARK software	89
8	APACHE HADOOP CONNECTION WITH THE MONGODB CONECTOR	91
8.1	Installation of Hadoop MongoDB connector.....	94
9	CONNECTION OF APPLICATIONS TO MONGODB CLUSTER.....	95
9.1	Application INTERACTIONS from WiseGRID applications to the Big Data Platform.....	95

9.1.1 Databases created in lab testing phase.....	95
9.1.2 Big data interactions from WG IOP application.....	96
9.1.3 Big data interactions from WG Cockpit application	97
9.1.4 Big data interactions from WiseCORP application	103
9.1.5 Big data interactions from WiseCOOP application.....	107
9.1.6 Big data interactions from WiseHOME application.....	111
9.1.7 Big data interactions from WG EVP application.....	113
9.1.8 Big data interactions from WG FastV2G application.....	117
9.1.9 Big data interactions from WG Staas/VPP application.....	117
9.1.10 Big data interactions from WG RESCO application.....	120

10 CONCLUSIONS AND NEXT STEPS..... 122

10.1 Conclusions.....	122
-----------------------	-----

10.2 Next steps to be implemented.....	123
--	-----

11 REFERENCES AND ACRONYMS..... 124

11.1 References	124
-----------------------	-----

11.2 Acronyms.....	126
--------------------	-----

LIST OF FIGURES

Figure 1 – Big Data 5V [2]	12
Figure 2– BigData Architecture overview.....	14
Figure 3 – Sharding diagram [1]	16
Figure 4 – Replication diagram.....	17
Figure 5 – Network cluster structure.....	17
Figure 6 – Hadoop cluster structure [3]	20
Figure 7 – Hadoop -Spark minimal structure	21
Figure 8 – Raspberry Pi Model 3B+	23
Figure 9 – Odroid XU4 [9]	25
Figure 10 – Odroid XU4 Block Diagram [9]	27
Figure 11 – Odroid C2 [10].....	28
Figure 12 – Odroid C2 block diagram [10]	30
Figure 13 – DIN rail	32
Figure 14 – 19" rack mounts for DIN Rail	32
Figure 15 – Odroid C2 DIN mount view 1	33
Figure 16 –Odroid C2 DIN mount view 2.....	33
Figure 17 – Odroid C2 DIN Mount view 3.....	34

Figure 18 – Minimal MongoDB cluster in lab testing	34
Figure 19 – Wago 221 power rail for connecting Odroid C2	35
Figure 20 – Webmin dashboard on a cluster node	41
Figure 21 – Webmin screen for automatic updates.....	42
Figure 22 – Connection to the MongoDB application router trough MongoDB Compass.....	64
Figure 23 – Odroid XU4 Din Rail Support view 1	66
Figure 24 – Odroid XU4 support view 2	67
Figure 25 – Odroid XU4 DIN rail support view 3.....	67
Figure 26 – Minimal Odroid XU4 Hadoop cluster.....	68
Figure 27 – Hadoop MongoDB batch aggregation [25].....	91
Figure 28 – MongoDB Hadoop Data Warehouse [25].....	92
Figure 29 – Hadoop ETL from MongoDB [25].....	93
Figure 30 – Hadoop ETL to MongoDB [25]	93
Figure 31– Databases created in lab testing phase.....	95
Figure 32 – WG IOP Architecture	97
Figure 33 - WiseGRID Cockpit.....	98
Figure 34 – WG Cockpit interface with Big Data platform	99
Figure 35 – WG Cockpit data mining	99
Figure 36 – Collections inside the WG_Cockpit database	100
Figure 37 – Documents stored in the wgcockpit_config collection	101
Figure 38 – WiseCORP	103
Figure 39 – WiseCORP interface with Big Data platform	104
Figure 40 – WiseCORP data mining.....	105
Figure 41 – Details of the WG_Corp database	106
Figure 42 – WiseCOOP.....	108
Figure 43 – WiseCOOP Big Data Platform interface	108
Figure 44 – WiseCOOP Big Data mining	109
Figure 45 – Details of WiseCoop Database.....	110
Figure 46 – WiseHOME Interaction Diagram	112
Figure 47 – Wise EVP	113
Figure 48 – WiseEVP interface with Big Data platform	114
Figure 49 – WiseEVP Big Data mining.....	115
Figure 50 – WiseEVP database	116
Figure 51 – Sketch of data flow between WG StaaS/VPP components and neighbouring WG tools.....	118
Figure 52 – Object Mapping between Node and MongoDB managed via Mongoose.....	121
Figure 53 – WG RESCO Tool interface with Big Data Platform.....	122

LIST OF TABLES

Table 1 – Odroid XU4 specification	26
Table 2 – Odroid C2 specifications	30
Table 3 – IPs for sharded replica set.....	37
Table 4 – IP for management and application routers nodes	37
Table 5 – Config servers IP and names	45
Table 6 – Replica shards servers.....	49
Table 7 – Names and IP for Hadoop Spark cluster	70
Table 8 –Collections in WGcockpit data base.....	99
Table 9 – Wisecorp database collections	105
Table 10 – Wisecoop Database collections	110
Table 11 – Collections in Wise EVP database	115
Table 12 – List of WG StaaS/VPP monitoring data	119
Table 13 – Deliverable objectives.....	123
Table 14 – List of Acronyms.....	126

EXECUTIVE SUMMARY

The executive summary shortly summarizes the contents of the main document chapter by chapter.

Chapter 1 INTRODUCTION

Introductory chapter presenting the scope and the structure of the document. The main part of the document is focused on installation of demonstrator for lab testing in order to become an extensive manual for deploying similar platforms on the pilot sites.

Chapter 2 BIG DATA PLATFORM FOR WISEGRID PROJECT

The chapter is summarizing the basic notions of Big Data, detailed in D5.1 [1] in the context of WiseGRID project. The Big data platform for WiseGRID project has two main components.

- An online platform based on a computer cluster that is providing online Big Data services as long term data storage and retrieving in separate databases for each application
- An offline platform based on a separate computer cluster that is providing the offline services of data processing. This cluster is based by Apache Hadoop framework and will provide the remote processing of applications based on Apache Spark operators.

Chapter 3 BIG DATA ONLINE PLATFORM

The chapter define general notions of MongoDB computer cluster detailed in D5.1 [1] focusing on sharding and replications mechanisms and defines the minimal structure used by the demonstrator for WiseGRID project.

Chapter 4 BIG DATA OFFLINE PLATFORM

The chapter define general notions of Hadoop Spark computer cluster detailed in D5.1 [1] focusing on defining the minimal structure used by the demonstrator for WiseGRID project.

Chapter 5 SINGLE BOARD COMPUTERS FOR WISEGRID DEMONSTRATOR

The chapter is analysing the technical specifications of three types of single board computers in order to decide which one is appropriate to be used in the two computer cluster that compose the demonstrator for lab testing in WiseGRID project.

The three types are:

- Raspberry pi model 3B+, the latest and the most powerful single board computer from Raspberry pi foundation
- Odroid XU4 the most powerful single board computer on ARM processor produced by the

Korean company Hardkernel.

- Odroid C2 the single board computer produced by Hardkernel that has full official support for Linux 64-bit. Using Linux 64-bit distribution is needed for full performance MongoDB installation.

The demonstrator is using Odroid C2 for MongoDB computer cluster and Odroid XU4 for Hadoop-Spark cluster.

Chapter 6 INSTALLATION OF THE MONGODB CLUSTER

The chapter is a complete and extensive installation manual of a MongoDB cluster that is using sharding and replication having as an example the eleven nodes cluster installed for lab testing purposed as WiseGRID WP5 demonstrator. The chapter targets to be used for installation and deploying the Big Data online platform in the pilot sites of the WiseGRID project. The described phases of installation are:

- PHYSICAL INSTALLATION OF THE Odroid C2 SBC FOR RACK MOUNTING
 - 3D PRINTED SUPORTS FOR ODROID C2
 - POWER SUPPLY OF THE ODROID C2 SBCs
 - VENTILATION
- INSTALLATION OF UBUNTU LINUX OPERATING SYSTEM
 - Defining the IP address system
 - Preparation of the nodes for cluster configuration
- INSTALLATION OF MANAGEMENT SOFTWARE: WEBMIN
- INSTALLATION OF THE MONGODB ENTERPRISE
- INSTALLATION OF MONGODB CLUSTER
 - Setting of configuration servers
 - Activation of config servers
 - Setting of replicated shards
 - Activation of replica shards servers
 - Configuration and activation of arbiter instances
 - Configuration and activation of application routers
 - Final settings and online connection

Chapter 7 INSTALLATION OF THE APACHE HADOOP CLUSTER

The chapter is a complete and extensive installation manual of a Hadoop Spark cluster having as an example the five nodes cluster installed for lab testing purposed as WiseGRID WP5 demonstrator. The chapter targets to be used for installation and deploying the Big Data offline platform in the pilot sites of the WiseGRID project. There are described phases of installation as:

- PHYSICAL INSTALLATION OF THE Odroid XU4 SBC FOR RACK MOUNTING
 - 3D PRINTED SUPORTS FOR ODROID XU4
 - POWER SUPPLY OF THE ODROID XU4 SBCs
 - VENTILATION
- INSTALLATION OF UBUNTU LINUX OPERATING SYSTEM
 - Defining the IP address system
 - Preparation of the nodes for cluster configuration
- INSTALLATION OF MANAGEMENT SOFTWARE, DISTRIBUTED cli SOFTWARE
- INSTALLATION OF THE HADOOP SOFTWARE

- Installing Java
- Preparing the user, groups and rights for installing Hadoop
- Installing and deploying Hadoop software
- ACTIVATION OF HADOOP CLUSTER
- INSTALLATION OF APACHE SPARK SOFTWARE
 - Prerequisites to SPARK installation
 - Installation of Apache SPARK software
 - Deployment of SPARK software and settings on all slaves
 - Activation of SPARK software

Chapter 8 APACHE HADOOP CONNECTION WITH THE MONGODB CONECTOR

The chapter describes the installation of Hadoop java plugin used for connecting to MongoDB:

” MongoDB connector for Hadoop”. This plugin provides Hadoop to use MongoDB as input source for Hadoop and also an output destination.

Chapter 9 CONNECTION OF APPLICATIONS TO MONGODB CLUSTER

The chapter represent the contribution of the developers for WiseGRID applications that are interacting with the Big Data platform and describes each such interaction.

Chapter 10 CONCLUSIONS AND NEXT STEPS

This chapter draws some insight from the development of the WiseGRID WP5 demonstrator and establishes the guiding lines for deployment of the Big Data platform in the pilot sites.

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

In the context of WiseGRID's WP5, this document will define the architecture of a cloud based Big Data demonstrator that will be used by WiseGRID applications.

Specifications of the WiseGRID Big Data Cloud-based infrastructure towards enabling: (i) Data Integration from a variety of heterogeneous data sources, (ii) Storage and Processing of huge data volumes in a highly efficient and effective manner, (iii) adaptation of Service-Oriented-Architectures (SOA) towards enhancing flexibility, scalability, reusability, loose coupling, integration of a variety of functionalities, and (iv) Interoperability and Interconnection with a wide range of applications. The purpose is also to define the configuration and integration of the WiseGRID Big Data infrastructure with the WG IOP and the individual WiseGRID components.

1.2 SCOPE OF THE DOCUMENT

This document will focus on the testing and refinement of the architecture for a Big Data cloud-based platform to suit the requirements of the WiseGRID applications.

1.3 STRUCTURE OF THE DOCUMENT

The document will use defined general concepts and notions about Big Data platforms from D5.1 [1] and describe a demonstrator for this purpose and the lab testing and refinement phase.

The MongoDB a leading open source NoSQL database that has been proposed to be used for implementing the Big Data platform cluster for WiseGRID project in D5.1 is used in a computer cluster demonstrator described in chapter 3 BIG DATA ONLINE PLATFORM and detailed in chapter 6 INSTALLATION OF THE MONGODB CLUSTER.

The interface between the applications of WiseGRID project and the Big Data platform will be direct from the applications routers dedicated to each application in chapter 9 CONNECTION OF APPLICATIONS TO MONGODB CLUSTER.

The Data mining and Data analytics cluster implementation in order to be used in WiseGRID project in appropriate applications is described in chapter 4 BIG DATA OFFLINE PLATFORM and detailed in chapter 7 INSTALLATION OF THE APACHE HADOOP CLUSTER

2 BIG DATA PLATFORM FOR WISEGRID PROJECT

2.1 BIG DATA DEFINITIONS

Big Data is a concept that emerges early in 1990 due to the continuously increases in the amount of data produced in all human activities that has to be stored, managed and processed. Big Data is in fact your data, every data. It is information owned by you, by your company, obtained and processed through new techniques in order to produce value in the best way possible.



Figure 1 – Big Data 5V [2]

- **Volume** refers to the vast amounts of data generated every second. Just think of all the emails, twitter messages, photos, video clips, sensor data etc. we produce and share every second. We are not talking Terabytes but Zettabytes or Brontobytes. On Facebook alone, we send 10 billion messages per day, click the "like" button 4.5 billion times and upload 350 million new pictures each and every day. If we take all the data generated in the world between the beginning of time and 2008, the same amount of data will soon be generated every minute! This increasingly makes data sets too large to store and analyse using traditional database technology. With big data technology we can now store and use these data sets with the help of distributed systems, where parts of the data are stored in different locations and brought together by software. [2]
- **Velocity** refers to the speed at which new data is generated and the speed at which data moves around. Just think of social media messages going viral in seconds, the speed at which credit card transactions are checked for fraudulent activities, or the milliseconds it takes trading systems to analyse social media networks to pick up signals that trigger decisions to buy or sell shares. Big data technology allows us now to analyse the data while it is being generated, without ever putting it into databases. [2]
- **Variety** refers to the different types of data we can now use. In the past we focused on structured data that neatly fits into tables or relational databases, such as financial data (e.g.

sales by product or region). In fact, 80% of the world's data is now unstructured, and therefore can't easily be put into tables (think of photos, video sequences or social media updates). With big data technology we can now harness differed types of data (structured and unstructured) including messages, social media conversations, photos, sensor data, video or voice recordings and bring them together with more traditional, structured data. [2]

- **Veracity** refers to the messiness or trustworthiness of the data. With many forms of big data, quality and accuracy are less controllable (just think of Twitter posts with hash tags, abbreviations, typos and colloquial speech as well as the reliability and accuracy of content) but big data and analytics technology now allows us to work with these types of data. The volumes often make up for the lack of quality or accuracy. [2]
- **Value** Then there is another V to take into account when looking at Big Data: Value! It is all well and good having access to big data but unless we can turn it into value it is useless. So, you can safely argue that 'value' is the most important V of Big Data. It is important that businesses make a business case for any attempt to collect and leverage big data. It is so easy to fall into the buzz trap and embark on big data initiatives without a clear understanding of costs and benefits. [2]

In WiseGRID project a potentially huge amount of data will be produced by different data sources:

- **Various sensors,**
- **Weather data**
- **Energy smart meters,**
- **EVs charging stations,**
- **Energy production units,**
- **Dispatchable consumers**
- **Energy storage units,**
- **DSO data,**
- **WiseGRID applications**
- **Energy marked data**
- **Prosumers, other**

All this data is produced, processed and consumed by different WiseGRID applications. It is needed for the project to have an online platform ready to store the data in an easily scalable system. From the data stored insights, trends, key performance indicators have to be extracted by a processing platform able to increase the value of the data.

The interaction of the WiseGRID applications and big data platform were defined in the beginning and refined by the evolution of applications development

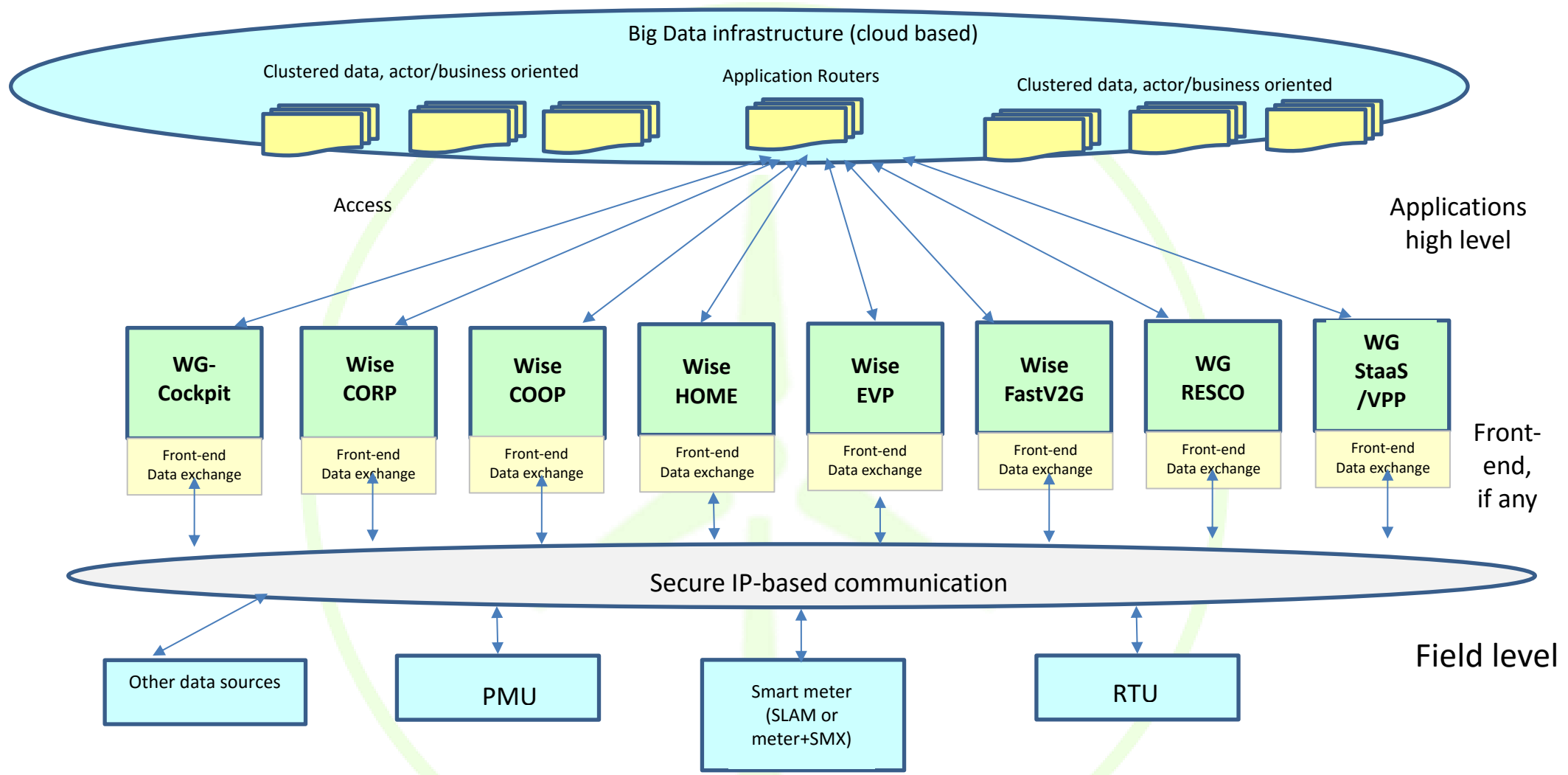


Figure 2– BigData Architecture overview

The WiseGRID BigData platform is an online secured platform providing database as service. The platform is based on two sections:

- for online services, a component for storing huge amount of data in a very flexible format coming from heterogenous sources.
- for offline services, a component for mining and analysing the stored data to get new information, trends, insights from stored data. This information obtained from stored data is getting value from data

2.2 BIG DATA PLATFORMS FOR WISEGRID PROJECT DEFINED BY D5.1

The WiseGRID deliverable D5.1 [1] defines an infrastructure for implementing the Big Data services both online and offline.

1. The first cluster is providing online Big Data services as long term data storage and retrieving in separate databases for each application.
 - A limited access database for each application containing data that are not under the restrictions of GDPR. The access will be granted by a user/password pair contained in each data base.
 - A limited access database for each application that is storing personal data, under the restrictions of GDPR. The access will be granted by user/password pair and also will be limited by the network identity of the client by the firewall installed before the application router
 - Each application will access the database through a dedicated application router installed on a separate cluster node.

The first cluster is running MongoDB NoSQL database management system [1].

2. The second cluster hardware separated from the database cluster is providing the offline services. This cluster is based by Apache Hadoop framework and will provide the remote processing of applications based on Apache Spark operators.
The second cluster is connected to the MongoDB database cluster through the Hadoop MongoDB connector. Each application will have a separate account for each WiseGRID application [1].

3 BIG DATA ONLINE PLATFORM

In time many types of databases were used for dealing with the data amount considered like Big Data and the most appropriate database management systems were the NoSQL database management systems.

3.1 BIG DATA MONGODB CLUSTER STRUCTURE

Deliverable D5.1 [1] explains the selection of the MongoDB database management system for WiseGRID. The WiseGRID online Big data platform is based on a computer cluster that is easily horizontally scalable upon the requirements that for high availability is using the MongoDB mechanism of sharding and replication.

The Sharding mechanism allows the mongoDB databases to be distributed on more than one cluster nodes allowing scalable cluster upon dimension of database requirements and scalability on the fly.

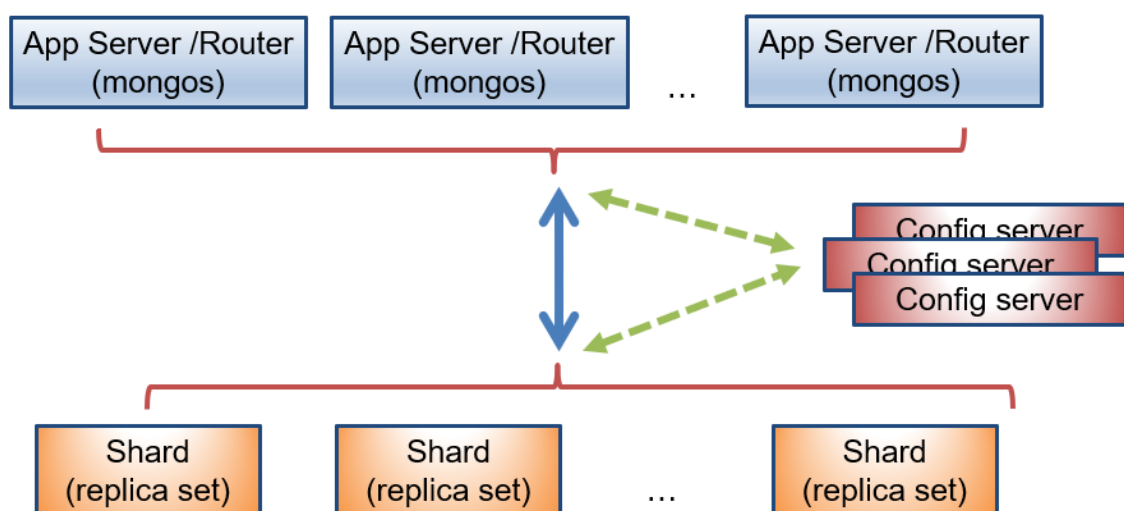


Figure 3 – Sharding diagram [1]

The sharding mechanism allows database extending based on requirements and also distributed access from multiple Application routers /servers providing the required computing power from WiseGRID applications.

The cluster reliability is provided by the replication mechanism. The replication allows that the same data is located on separate nodes and in case of one node is faulty or offline due maintenance procedures the data is still available

The replication mechanism allows that the data from on shard is replicated on two or more nodes of the cluster. The switching from one database instance to another in case of unavailability on one node is based by an “arbiter” running on other nodes than the primary or secondary instances.

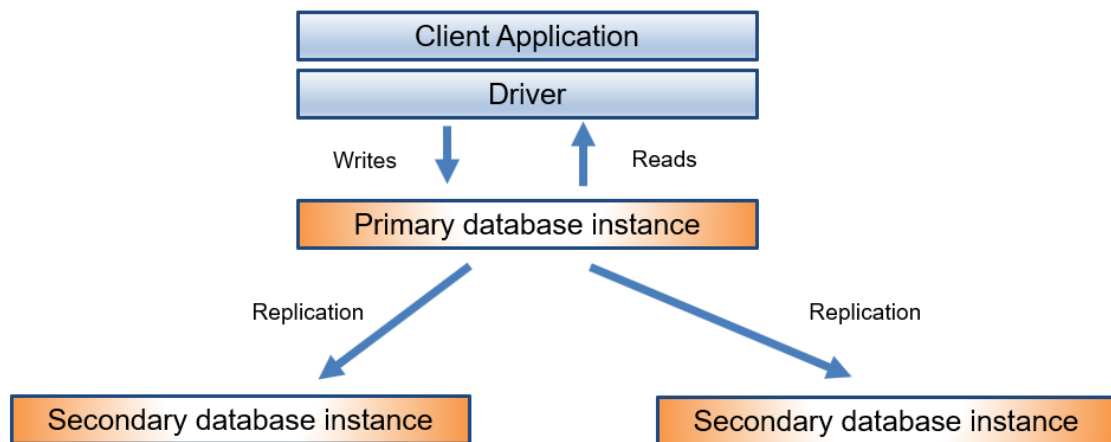


Figure 4 – Replication diagram

3.2 BIG DATA MONGODB DEMONSTRATOR STRUCTURE

The Big Data demonstrator platform infrastructure has to be able to run a NoSQL database, accessible from all the WiseGRID applications heterogeneous ecosystem, that will provide cloud based services to the applications that will need to store long term data, retrieve the stored data, process and filter the stored data.

The MongoDB cluster will be built over multiple nodes that are separated computers connected in a network on the clustering principles.

To have a functional and cost-effective demonstrator that allows scalability the cluster use single board computers as nodes. This document defines a minimal cluster to startup the development that can be scaled on the fly based on further requirements.

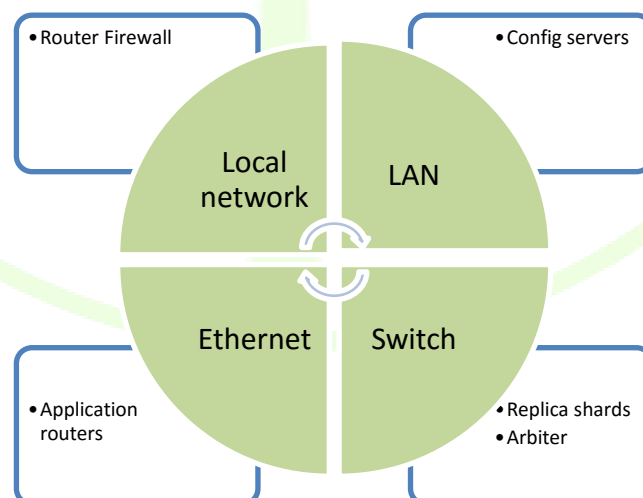


Figure 5 – Network cluster structure

The minimal structure for the lab testing demonstrator is composed by:

- **Router Firewall** provide access from the applications running on external networks than the cluster local network. Functions and services implemented on the router:
 - Firewall to allow only legitimate access from the exterior and only to the nodes that need to be accessed from outside of the local network.
 - IP masquerading to allow nodes connected to the local network to access software repositories, update servers
 - DHCP for simple IP allocation for nodes in configuration process
- **LAN switch** providing ethernet access to all the cluster nodes for data interchange, process communications.
- **Config servers:** cluster nodes that contain data about cluster structure, data sharding. A minimal three configuration nodes are used in production clusters.
 - Config 0 server
 - Config 1 server
 - Config 2 server
- **Replica shards:** cluster nodes that hold database data distributed and replicated on different nodes. The minimal starting structure is of 6 nodes
 - **Replica shard 0 rs0**
 - Replica 0 node for shard 0
 - Replica 1 node for shard 1
 - **Replica shard 1 rs1**
 - Replica 0 node for shard 0
 - Replica 1 node for shard 1
 - **Replica shard 2 rs2**
 - Replica 0 node for shard 0
 - Replica 1 node for shard 1
- **Arbiter:** separate cluster node running a database instance for each replica shard. Does not hold database data but decides the replica role as “primary” or “secondary”
- **Application router:** cluster node running a special instance of database. Does not hold data but process the request from external application and directs to the cluster nodes able to fulfil the request. The number of application routers can be extended based on the load generated by the applications. For the beginning only one application router can be used.

As indicated above a minimum of eleven cluster nodes has to be used for minimal structure.

4 BIG DATA OFFLINE PLATFORM

Big data is not only storing huge amount of data and retrieving it, but also processing stored data in order to obtain new information, trends, and insights.

There are two types of processes:

- **Data mining**
- **Data analytics**

Data mining is a process to structure the raw data and formulate or recognise the various patterns in the data through the mathematical and computational algorithms, data mining helps to generate new information and unlock the various insights. The data is first placed into a data warehouse to do the required extraction of data to produce meaningful relationships and patterns [1].

Data analytics is the art of exploring the facts from the data with specific to answer specific questions, i.e. there is a test hypothesis framework for data analytics. The techniques used in analytics also are same as used in business analytics & business intelligence [1].

Both processes are called as “offline processes” since there are no synchronism between producing and storing data to processing data.

The Big Data offline platform provides scalable computing power and distributed resources in order to allow processing of huge amount of data already stored in the database system.

4.1 BIG DATA HADOOP CLUSTER STRUCTURE

The Big Data platform that will be specific designed for the WiseGRID project was defined like having some specific requirements:

- It is based on **Apache™ Hadoop®** framework that provides distributed processing of large data sets. Large datasets need for processing to be stored in distributed file systems that is located across on multiple nodes of the cluster, processing has to be done by a system that provides scheduling the jobs in distributed applications running on different nodes of the cluster.
- It uses Apache **Spark™** that is a fast and general engine for large-scale data processing. Apache Spark is providing a set of tools for applications written in Java, Scala, Python and R for building parallel processing apps.

Hadoop Cluster

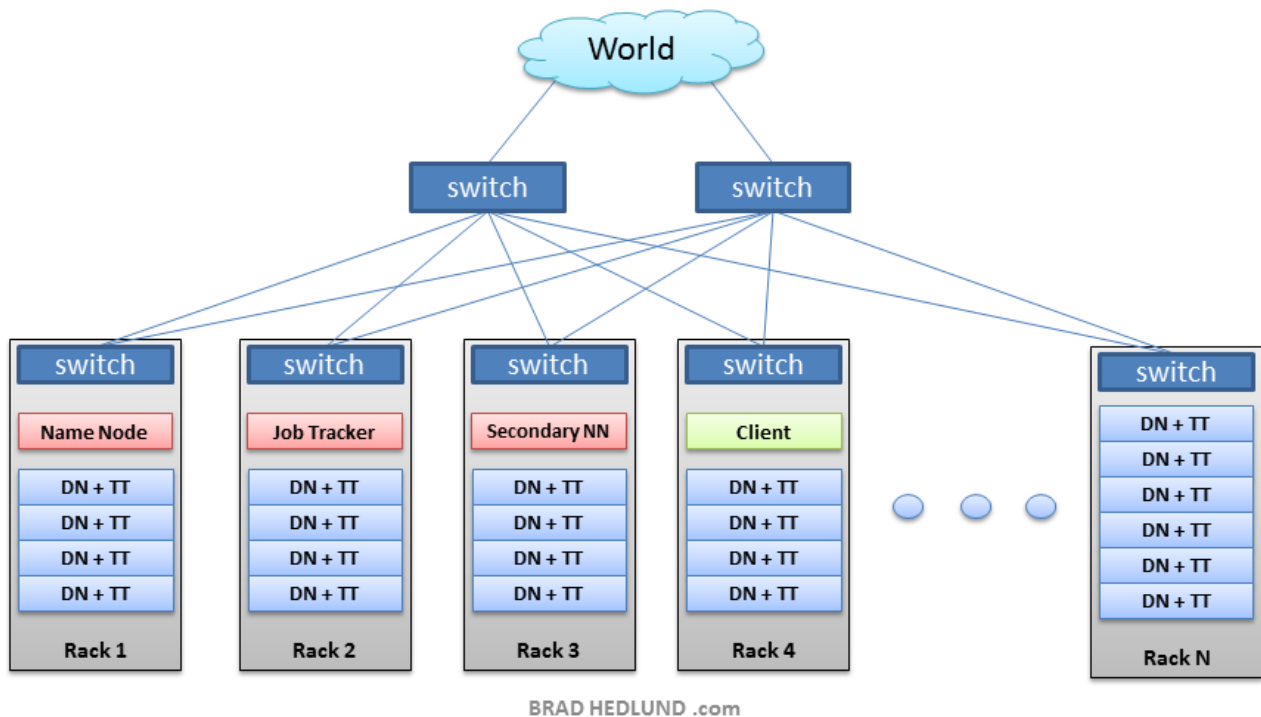


Figure 6 – Hadoop cluster structure [3]

4.2 BIG DATA HADOOP ARCHITECTURE ADAPTED FOR WISEGRID PROJECT DEMONSTARTOR

The offline processing of data stored in the Big Data platform is not expected to be extensive. Up to this development phase there are not many applications that need offline Big Data processing. The Hadoop-Spark cluster is designed to be minimal in this phase of the project but can be scaled to the eventually extent based on the evolution of development in the pilot sites.

The Hadoop-Spark cluster is connected on the same LAN as the MongoDB cluster behind the same router.

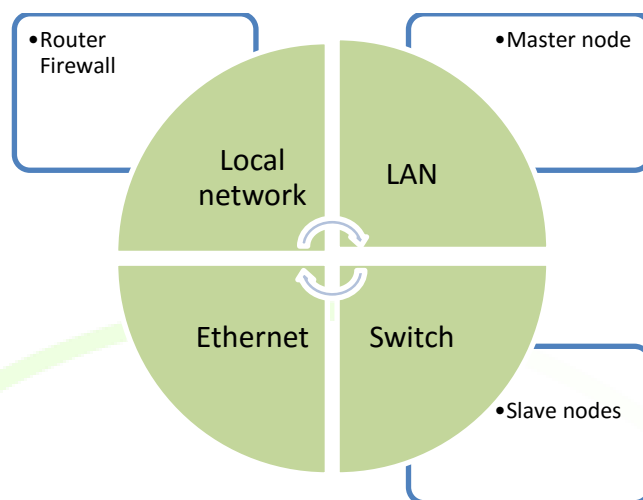


Figure 7 – Hadoop -Spark minimal structure

The cluster is composed of:

- One master node running the HDFS, Hadoop distributed file system, YARN distributed processing framework and Map reduce. The same master is running also Apache Spark framework.
- Four slaves that are participating with their resources to the cluster.

5 SINGLE BOARD COMPUTERS FOR WISEGRID DEMONSTRATOR

5.1 WHAT ARE SINGLE BOARD COMPUTERS

A single board computer is a complete computer build in on a single printed circuit board [4]. This only one printed circuit board contains the processor, RAM memory, In/Out circuits, storage memory, mostly as solid state non-volatile memory in the last period of time.

In the last years an explosive evolution of single board computer that will be named as SBC further was started by launching the Raspberry Pi single board computer mainly for educational purposes. The first version of Raspberry Pi SBC was launched in United Kingdom by Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries [5]. The first version reaches the market in February 2012

One main development that helped the evolution of SBC computers was the proliferation of ARM processors in many small mobile and handheld appliances.

ARM, previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. Arm Holdings develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures—including systems-on-chips (SoC) and systems-on-modules (SoM) that incorporate memory, interfaces, radios, etc. It also designs cores that implement this instruction set and licenses these designs to a number of companies that incorporate those core designs into their own products. [6].

Since the first success of Raspberry Pi in 2012 many types of SBC computers were developed in the family of Raspberry Pi and by many other developer companies that bring on the market SBC computers with continuously increasing of computing power and performances.

With over 100 billion ARM processors produced as of 2017, ARM is the most widely used instruction set architecture in terms of quantity produced [6].

The Linux versions for ARM processors are mature and many distributions are developed and maintained continuously.

The Raspberry Pi Foundation that pushed the race of ARM single board computers published their strategy for 2016-2019 as **“Putting the power of digital making into the hands of people all over the world”** [7]

5.2 SINGLE BOARD COMPUTERS ANALYZED FOR WISEGRID DEMONSTRATOR

The use of single board computers for WiseGRID Bigdata architecture emerged from the beginning of project development due to the possibility to build cost effective computer clusters with remarkable computing power. The possibility of horizontal scaling of computer clusters with a good price performance ratio and by using opensource software was an important criterion of implementing the cluster architecture defined in deliverable D5.1

The first step in implementing the demonstrators was to define the main specifications to find suitable single board computers for the two computer clusters: the MongoDB computer cluster and the Apache Hadoop-Spark cluster.

The MongoDB computer cluster has a strict requirement to use a 64-bit version of Linux so the architecture of the ARM processors that has to be used in MongoDB computer cluster nodes is an ARM V8 architecture.

5.2.1 Raspberry Pi 3B+

The first idea was to use last and the most powerful version of Raspberry Pi that emerged on the market in 2018 as “Raspberry pi model 3B+.

The story of Raspberry Pi evolution is described in Strategy 2016-2018 document [7]

Raspberry Pi Foundation was established in 2008 as a UK-based charity with the purpose “to further the advancement of education of adults and children, particularly in the field of computers, computer science and related subjects”.

Through our trading subsidiary, (Raspberry Pi Trading Limited), we invent and sell low-cost, high-performance computers that people use to learn, to solve problems, and have fun. Between launching our first product in February 2012 and our fourth birthday in February 2016, we sold over eight million Raspberry Pi computers and helped to establish a global community of digital makers and educators.

In October 2015 the Foundation merged with Code Club, a network of volunteer-led after-school coding clubs for 9-11-year olds.

In November 2015 we launched the world’s first \$5 computer, the Raspberry Pi Zero.

We use profits generated from our commercial activities to pursue our educational goals; we also receive funding and in-kind support from generous partners and donors that share our mission [7].

The strongest point of the Raspberry Pi family is the large community of developers that are using this family of SBCs in various applications.

The youngest member of the Raspberry pi family Raspberry Pi Model 3B+ is a single board computer with a credit card form factor as in the following picture.



Figure 8 – Raspberry Pi Model 3B+

5.2.1.1 The specifications of Raspberry Pi Model 3B+

The technical specifications are:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

The technical specifications of the Model 3B+ evolved from the previous versions but there are some drawbacks for using Raspberry Pi Model 3B+ in our cluster.

From the specifications above some insights can be underlined:

Even the line interface of Ethernet has been modified to a Gigabit one, the connection to the processor of the Ethernet interface is done through the USB 2.0 and limited to the maximum throughput of USB 3.0 to 300 Mbps.

The processor is based on an ARM v8 architecture and is able to be used in a 64-bit OS. The Raspberry Pi Foundations official Raspbian, Linux distribution based on Debian is still a 32-bit OS. Asking Eben Upton for: "Would we release a version of our operating environment that was built on top of 64-bit ARM Debian?" the answer was: "Not yet." [8]

The official reasons of not releasing a 64Bit OS:

"That deep backwards compatibility is really important for us, in large part because we don't want to orphan our customers. If someone spent \$35 on an older-model Raspberry Pi five or six years ago, they still spent \$35, so it would be wrong for us to throw them under the bus." [8]

The lack of official mature distribution for a 64-bit Linux prevented the use of Raspberry Pi Model 3B+ in a MongoDB cluster that is strictly requiring a 64-Bit OS.

5.2.2 Odroid XU4

Most powerful alternatives for Raspberry Pi must be used in the WiseGRID computer clusters for the BigData Platform. An alternative SBC was found developed by the Korean company HardKernel. The family of SBC developed by them is Odroid family. This family has many versions developed from 2012 and the most powerful member yet is ODROID XU4 released in the summer of 2015.

ODROID-XU4 is a new generation of computing device with more powerful, energy-efficient hardware and a smaller form factor. Offering open source support, the board can run various flavours of Linux, including the latest Ubuntu 16.04 and Android 4.4 KitKat and 7.1 Nougat.

By implementing the eMMC 5.0, USB 3.0 and Gigabit Ethernet interfaces, the ODROID-XU4 boasts amazing data transfer speeds, a feature that is increasingly required to support advanced processing power on ARM devices.

This allows users to truly experience an upgrade in computing, especially with faster booting, web browsing, networking and 3D games [9].



Figure 9 – Odroid XU4 [9]

5.2.2.1 Technical specifications of ODROID XU4

Processor	Samsung Exynos5 Octa ARM Cortex™-A15 Quad 2Ghz and Cortex™-A7 Quad 1.3GHz CPUs
Memory	2Gbyte LPDDR3 RAM at 933MHz (14.9GB/s memory bandwidth) <u>PoP</u> stacked
3D Accelerator	Mali-T628 MP6(OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile)
Video	supports 1080p via HDMI cable (H.264+AAC based MP4 container format)
Video Out	Standard Type A HDMI connector
Audio	Instead of On-board Audio codec, there is I2S Expansion Port (CON11)
USB3.0 Host	SuperSpeed USB standard A type connector x 2 port, Max Load: total 2Amp for two USB 3.0 host ports
USB2.0 Host	High Speed standard A type connector x 1 ports, Max Load: 500mA/port
Display	HDMI monitor

Storage (Option)	MicroSD Card Slot, eMMC module socket: eMMC 5.0 HS4000 Flash Storage
Gigabit Ethernet LAN	10/100/1000Mbps Ethernet with RJ-45 Jack (Auto-MDIX support)
Serial console port	Connecting to a PC gives access to the Linux console. You can see the log of the boot, or to log in to the C1 to change the video or network settings. Note that this serial UART uses a 1.8-volt interface. We recommend the USB-UART module kit from Hardkernel. Molex 5268-04a (2.5mm pitch) is mounted on the PCB. Its mate is Molex 50-37-5043 Wire-to-Board Crimp Housing.
RTC (Real Time Clock) back-up battery connector	If you want to add an RTC functions for logging or keeping time when of-fline, just connect a Lithium coin backup battery (CR2032 or equivalent). All the RTC circuits are included on the ODROID-XU4 by default. Molex 53398-0271 1.25mm pitch Header, Surface Mount, Vertical type (Mate with Molex 51021-0200)
WiFi (Option)	USB IEEE 802.11b/g/n 1T1R WLAN with Antenna (USB module)
HDD/SSD SATA interface (Optional)	SuperSpeed USB (USB 3.0) to Serial ATA3 adapter for 2.5"/3.5" HDD and SSD storage
Power (included)	5V 4A Power
Case (Option)	Mechanical case & cooler (90 x 59 x 28 mm approx.)
PCB Size	83 x 58 x 20 mm approx.

Table 1 – Odroid XU4 specification

The technical specifications show a much more powerful ARM computer than Raspberry Pi 3b+ but using an octacore ARM Cortex™-A15 Quad 2Ghz and Cortex™-A7 Quad 1.3GHz CPUs that is not a 64-bit processor.

5.2.2.2 ODROID XU4 Block Diagram

The block diagram of the Odroid XU4 published by Hardkernel [9] is shown in the image below

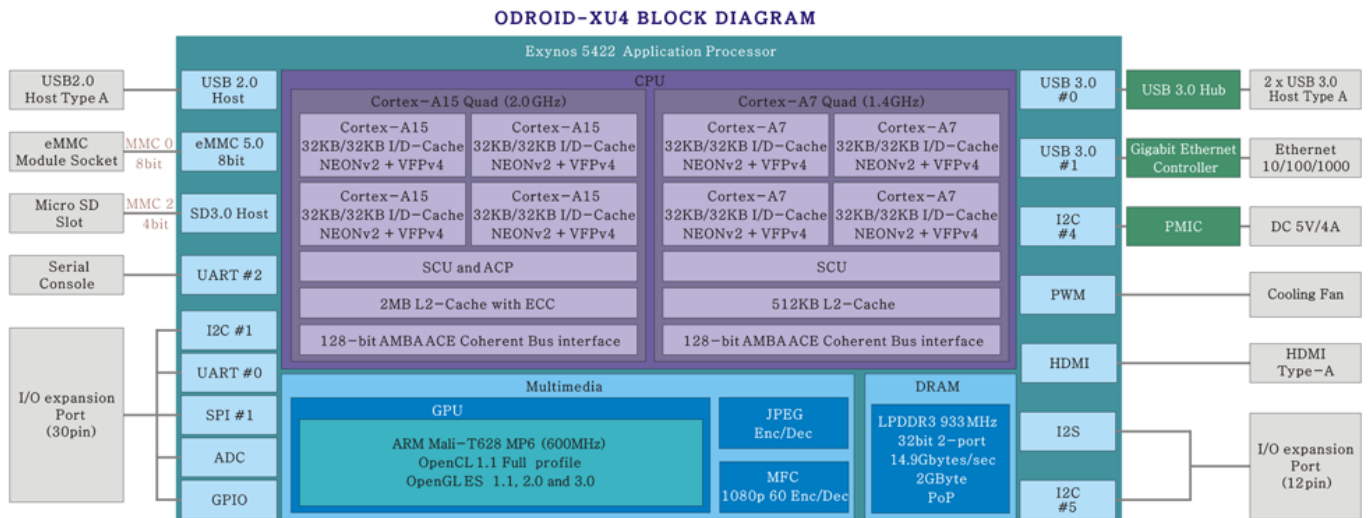


Figure 10 – Odroid XU4 Block Diagram [9]

Some particularities have to be underlined:

- The 10/100/100 Gigabit Ethernet interface is connected to the processor through a dedicated USB3.0 interface allowing full throughput of 1000 Mbit
- The eMMC 5.0 interface allow a higher data transfer than a SD card
- 2Gbyte LPDDR3 RAM at 933MHz (14.9GB/s memory bandwidth) allows a better overall performance

All this indicate that Odroid XU4 is the best candidate for applications where 64-bit OS is not requested as in the Apache Hadoop-Spark cluster where the use of java allows using 32-bit OS

5.2.3 Odroid C2

For the MongoDB cluster is needed an SBC that officially support 64-bit OS. A candidate for this application is Odroid C2

Some of the modern operating systems that run on the ODROID-C2 are Ubuntu, Android, Fedora, ARCH-Linux, Debian, and OpenELEC, with thousands of free open-source software packages available. The ODROID-C2 is an ARM device – the most widely used architecture for mobile devices and embedded 64-bit computing. The ARM processor's small size, reduced complexity and low power consumption makes it very suitable for miniaturized devices such as wearables and embedded controllers. [10]

Odroid C2 is officially running Ubuntu 16.04 LTS for 64-bit

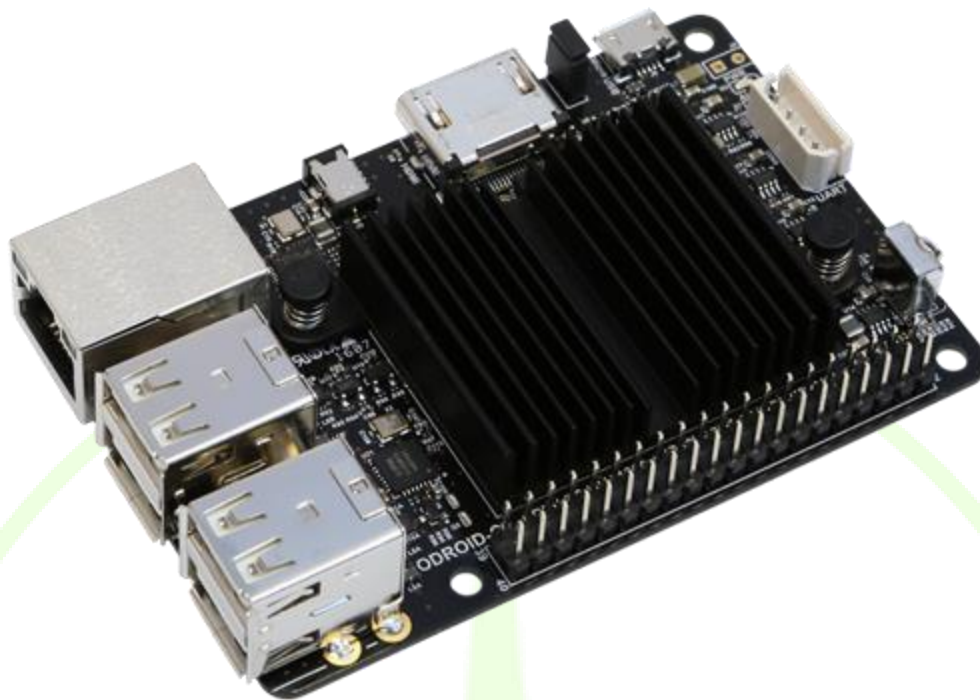


Figure 11 – Odroid C2 [10]

5.2.3.1 Odroid C2 technical specifications

The technical specifications published by Hardkernel are in the table below.

Processor	Amlogic S905: Quad Core Cortex™-A53 (ARMv8 64bit) processor with Triple Core Mali-450 GPU
RAM	2GByte DDR3 (32bit / 912Mhz)
eMMC module socket	The eMMC storage access time is 2-3 times faster than the SD card. You can purchase 4 size options: 8GB, 16GB, 32GB and 64GB. Using an eMMC module will increase speed and responsiveness, similar to the way in which upgrading to a Solid-State Drive (SSD) in a typical PC also improves performance over a mechanical hard drive (HDD).
Micro Secure Digital (MicroSD) Card slot	There are two different methods of storage for the operating system. One is by using a MicroSD Card and another is using an eMMC module, which is normally used for external storage for smartphones and digital cameras. The ODROID-C2 can utilize the newer UHS-1 SD model, which is about 2 times faster than a normal class 10 card. Note that there are some cards which needs additional booting delay time around 30 seconds. According to our test, most Sandisk Micro-SD cards don't cause the booting delay. We will make a compatibility list soon.
5V 2A DC input	This is for 5V power input, with an inner diameter of 0.8mm, and an outer diameter of 2.5mm. The ODROID-C2 consumes less than 0.8A in most cases, but

	it can climb to 2A if many passive USB peripherals are attached directly to the main board.
<i>USB host ports</i>	<p>There are four USB 2.0 host ports.</p> <p>You can plug a keyboard, mouse, WiFi adapter, storage or many other devices into these ports.</p> <p>You can also charge your smartphone with it! If you need more than 4 ports, you can use a powered external USB hub to reduce the power load on the main device.</p>
<i>HDMI port</i>	The Type-A standard-HDMI connector is populated on the board. If you use a UHD/4K 60Hz display, your HDMI cable must be compatible with HDMI 2.0 speed.
<i>Ethernet RJ-45 jack</i>	<p>The standard RJ45 Ethernet port for LAN connection supports 10/100/1000Mbps speed.</p> <p>Green LED Flashes when there is 100Mbps connectivity</p> <p>Yellow (Orange/Amber) LED Flashes when there is 1000Mbps connectivity</p>
<i>Status / Power LEDs</i>	<p>The ODROID-C2 has four indicator LEDs that provide visual feedback.</p> <p>Red LED: Power Hooked up to 5V power</p> <p>Blue LED</p> <p>Alive Solid light: u-boot is running</p> <p>Flashing: Kernel is running (heart beat)</p>
<i>Infrared (IR) receiver</i>	This is a remote-control receiver module that can accept standard 37.9Khz carrier frequency based wireless data.
<i>Micro USB OTG port</i>	<p>You can use the standard micro-USB connector with Linux Gadget drivers on your host PC, which means that the resources in the ODROID-C2 can be shared with typical PCs.</p> <p>You can also add a micro-USB to HOST connector if you need an additional USB host port.</p> <p>Note that this port can be used for power input if you install a jumper near the HDMI connector.</p>
<i>General Purpose Input and Output (GPIO) ports</i>	<p>These 40pin GPIO port can be used as GPIO/I2C/UART/ADC for electronics and robotics.</p> <p>The 40 GPIO pins on an ODROID-C2 are a great way to interface with physical devices like buttons and LEDs using a lightweight Linux controller.</p> <p>If you're a C/C++ or Python developer, there's a useful library called WiringPi that handles interfacing with the pins. We've already ported the WiringPi v2 library to ODROID-C2.</p> <p>Note that all the GPIO ports are 3.3Volt. The ADC inputs are limited to 1.8Volt.</p>
<i>Serial console port</i>	<p>Connecting to a PC gives access to the Linux console.</p> <p>You can see the log of the boot, or to log in to the C2 to change the video or network settings.</p> <p>Note that this serial UART uses a 3.3-volt interface. We recommend the USB-UART module kit from Hardkernel.</p> <p>Molex 5268-04a (2.5mm pitch) is mounted on the PCB. Its mate is Molex 50-37-5043 Wire-to-Board Crimp Housing.</p>
<i>Gigabit Ethernet PHY</i>	Realtek RTL8211F is a highly integrated Ethernet transceiver that complies with 10Base-T, 100Base-TX, and 1000Base-T IEEE 802.3 standards.
<i>USB MTT hub controller</i>	GENESYS LOGIC GL852G is used to implement the 4-port Hub function which fully complies with Universal Serial Bus Specification Revision 2.0.
<i>USB VBUS controller</i>	NCP380 Protection IC for USB power supply from OnSemi.
<i>Power switch port</i>	You can add a slide switch or rocker switch on this port if you want to imple-

	ment a hardware on/off switch. If this port is closed, the power is off. If this port is opened, the power is on.
Power supply circuit	Discrete DC-DC converters LDOs are used for CPU/DRAM/IO power supply.
Power protector IC	NCP372 Over-voltage, Over-current, Reverse-voltage protection IC from On-Semi.

Table 2 – Odroid C2 specifications

Odroid C2 has an opensource hardware with the block diagram bellow.

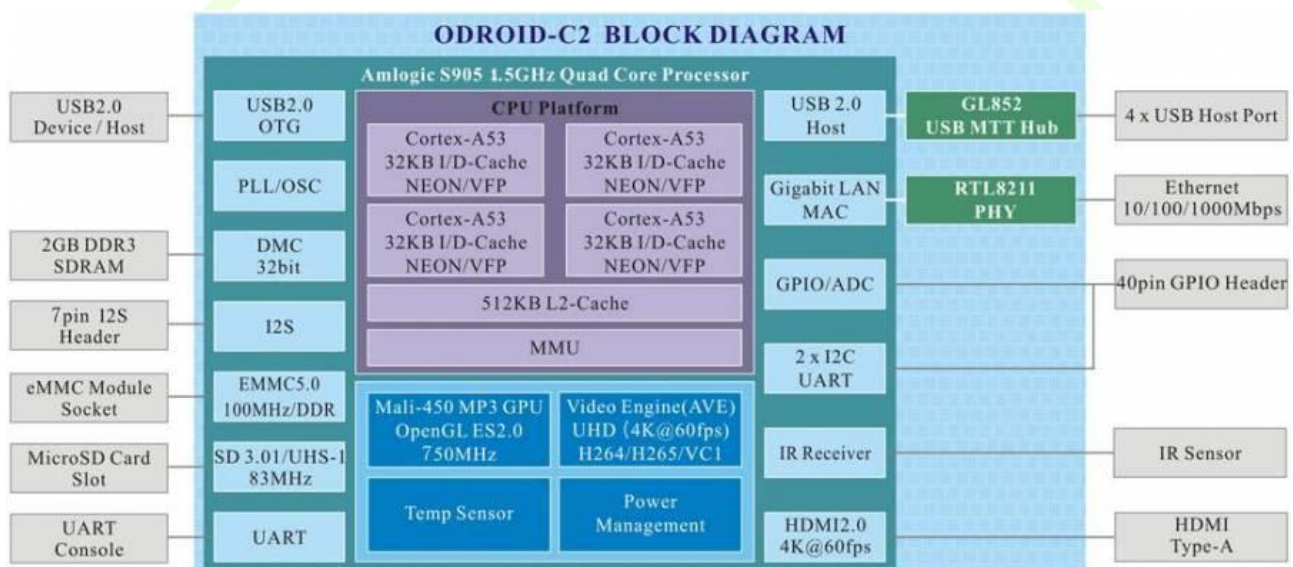


Figure 12 – Odroid C2 block diagram [10]

Some particularities must be underlined in the case of Odroid C2:

- The 10/100/100 Gigabit Ethernet interface is connected to the processor through a dedicated Gigabit LAN MAC interface allowing full throughput of 1000 Mbit
- The eMMC 5.0 interface allow a higher data transfer than a SD card
- 2Gbyte LPDDR3 RAM allows a better overall performance

All this indicate that Odroid C2 is the best candidate for applications where 64-bit OS is requested as in the MongoDB cluster.

6 INSTALLATION OF THE MONGODB CLUSTER

This chapter will detail the hardware and software installation of the MongoDB cluster. This chapter is intended to be a handbook for installing the MongoDB cluster for WiseGRID BigData platform in the pilot sites.

To help the understanding of the software procedure for installation and configuration some format conventions will be used:

- Comments and explanations will be in standard text style
This is how the comments and explanations will be shown
- Linux commands or mongo shell commands will be shown in blue text and italics
This is how a command to be issued will look like
- The output generated by the Linux OS or mongo shell will be shown only when are needed to be checked and will be in blue text and italics with a grey text highlight colour.
This is how an output from OS or mongo shell will look like
- The content of configuration files that has to be modified will be shown in blue text and italics
This is how a configuration file content will look like

Even the software procedures are extensively described, a basic skill for Linux user and administrator is needed to fulfil correctly the procedures and reach the goal to have a running MongoDB cluster. Consider in any moment to check the bibliography indicated in text and mainly the MongoDB online manuals [11]

6.1 PHYSICAL INSTALLATION OF THE ODROID C2 SBC FOR RACK MOUNTING

The SBC computers are very cheap since they do not have any enclosure. For using them in a computer cluster it is needed a technical solution to mount them providing access to the connectors used in the cluster application, to connect the power supply and provide cooling by free or forced airflow.

The solution in this project is using a DIN rail used in electrotechnical and automation industrial panels and some 3D printed adaptors for fixing the SBC computers over the DIN rail. The DIN rail is cut to a length to accommodate a standard 19" computer rack and is fixed to the rack with 3d printed supports.



Figure 13 – DIN rail

The 3d printed supports were designed for this project in Autodesk Fusion 360

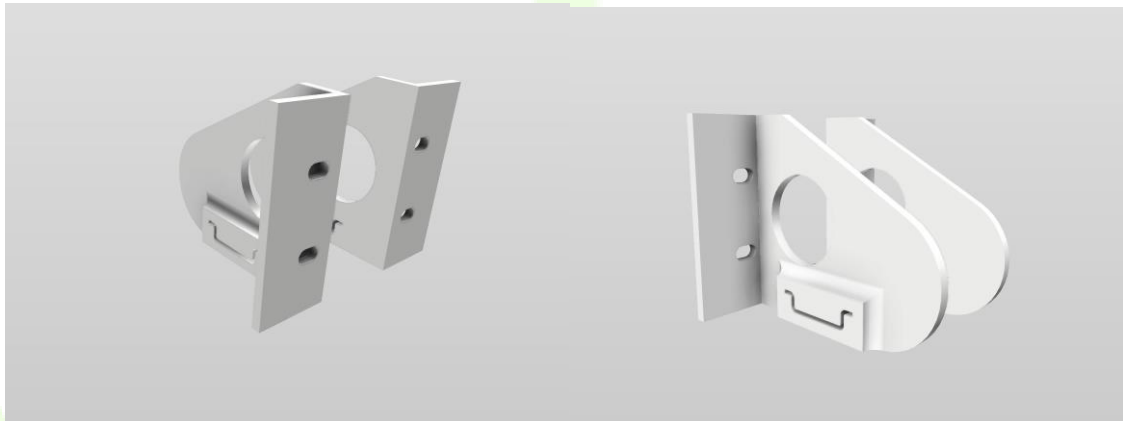


Figure 14 – 19" rack mounts for DIN Rail

6.1.1 3D PRINTED SUPORTS FOR ODROID C2

The supports for fixing the Odroid C2 used by the MongoDB computer cluster must allow the placement of the SBC computers in a stack using optimally the space over the DIN rail, easy access to the ethernet port and easy air flow for cooling. The idea of supports came from “instrng” that published a set of DIN rail supports on “DIN Mounts: Pi, Arduino and disks” [12] on Thingiverse. On the Thingiverse was published only the STL files for 3D printing so for adapting the DIN mounts for other SBC as Odroid C2 the complete design in Autodesk Fusion 360 was remade.

The Odroid C2 support was adapted for Odroid C2 SBC dimensions and some access holes for the SD card was designed. Some views from different angles from the Autodesk Fusion 360 are shown below



Figure 15 – Odroid C2 DIN mount view 1



Figure 16 –Odroid C2 DIN mount view 2

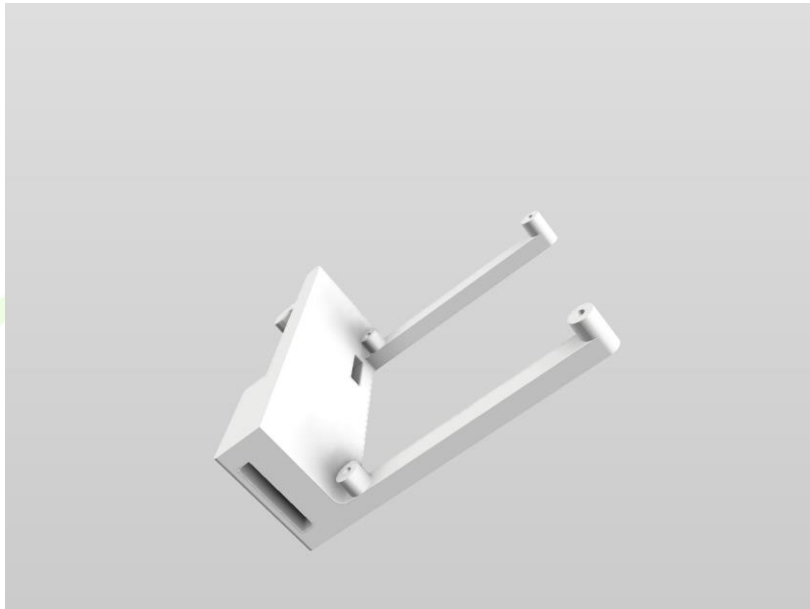


Figure 17 – Odroid C2 DIN Mount view 3

The 11 Odroid C2 for a minimal MongoDB cluster installed on DIN mounts on a rack mountable DIN rail are shown in the below picture

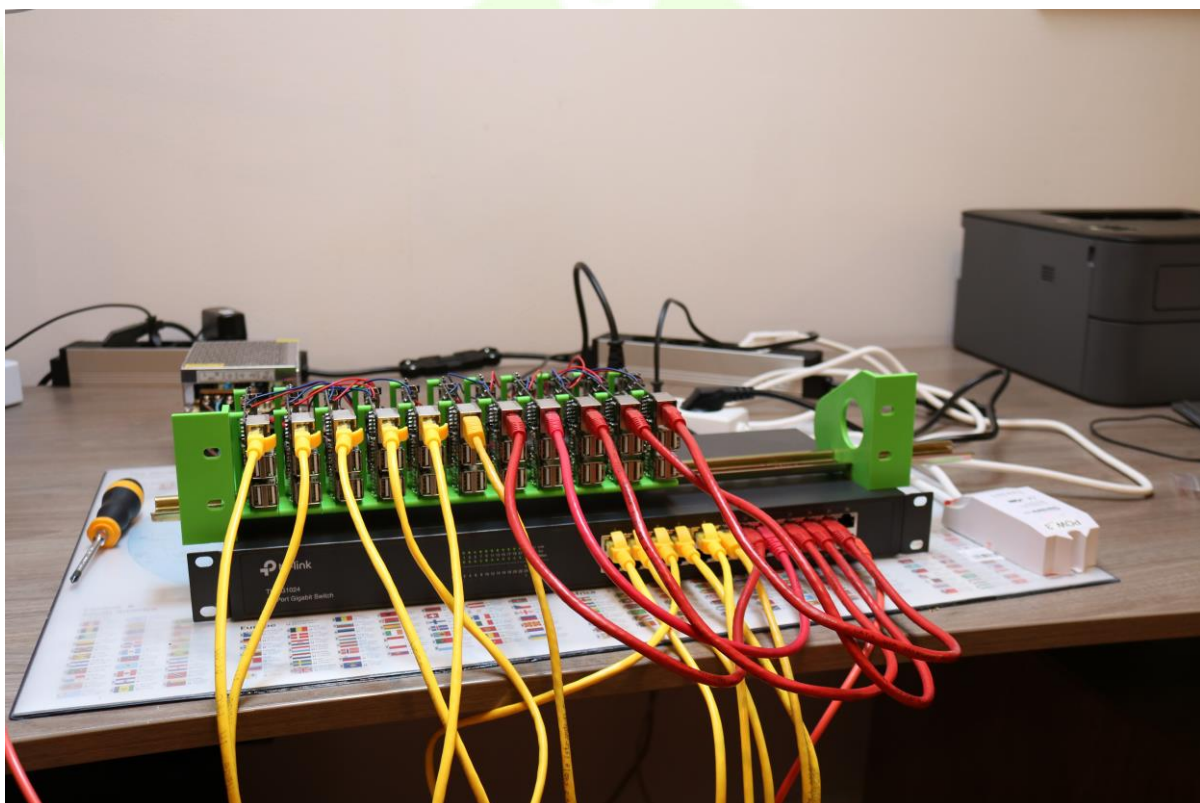


Figure 18 – Minimal MongoDB cluster in lab testing

6.1.2 POWER SUPPLY OF THE ODROID C2 SBCs

From the specifications published by Hardkernel is indicated that Odroid c2 is drawing a maximum current of 2A at 5VDC per SBC.

5V 2A DC input This is for 5V power input, with an inner diameter of 0.8mm, and an outer diameter of 2.5mm. The ODROID-C2 consumes less than 0.8A in most cases, but it can climb to 2A if many passive USB peripherals are attached directly to the main board.

The minimal Odroid C2 cluster is containing 11 SBC so considering a $0.8 \times 11 = 8.8$ A with a maximum of $2 \times 11 = 22$ A two power supplies for 5Vdc at 10 A will fulfil the power requirements. The power supplies do not accept a parallel connection, so the eleven SBC are divided in two separate groups for the point of view of power supply. For each group a small DC power rail is created by Wago 221 connectors with 5 connection each allowing a maximum of 20 A. The Wago connectors are placed in 3d printed sockets that allows installation on the same DIN rail. The sockets and the DIN rail supports are designed by “ebraud” and published as: “Support rail DIN Wago 221” [13]

Each Odroid C2 will be connected separately to the positive and negative power rails by two wires. The Wago connectors allows each SBC to be separated powered for maintenance purposes. The wires can be soldered in the place of a trough hole connector near the power plug or by using a standard connector as in Odroid C2 power outlet. The connector is available by Hardkernel as declared in Odroid C2 Manual [14] in page 5.

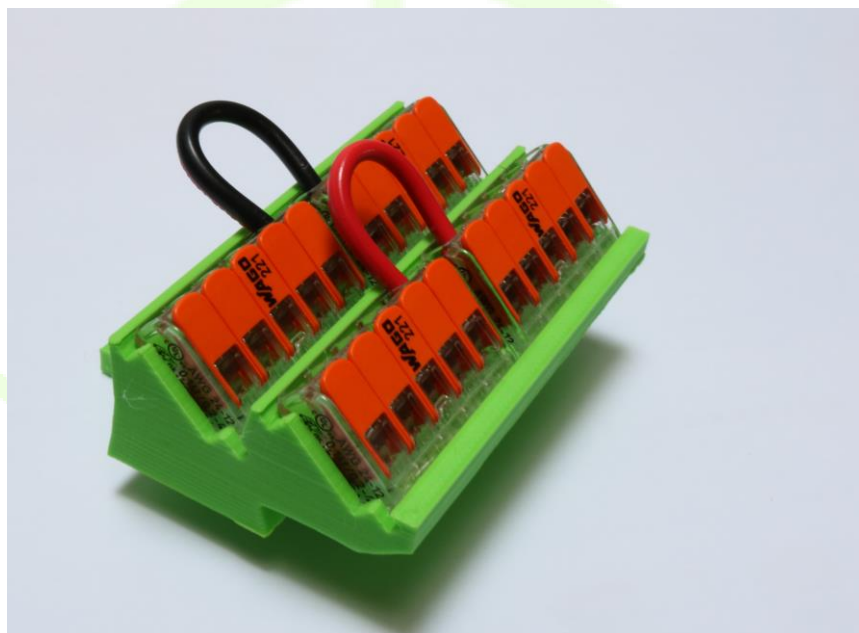


Figure 19 – Wago 221 power rail for connecting Odroid C2

6.1.3 VENTILATION

The Odroid C2 is provided with a heatsink able to dissipate the heat generated by the electronics on the board by a natural convection air flow. The distance from the Odroid C2 in the stack installed in supports described in chapter 6.1.1 allows the natural convection air flow. However, if the DIN rail is installed in a closed 19" rack, a suitable forced ventilation has to be provided to the rack to evacuate heat equivalent of 10 W per each Odroid C2 in the cluster.

Measuring the operating temperature of Odroid C2 is described in the Odroid C2 Manual [14] in page 14.

6.2 INSTALLATION OF UBUNTU LINUX OPERATING SYSTEM

The Odroid C2 eMMC or SD card ordered with the Linux version has preinstalled an image of Ubuntu Mate 64 bits. For using a non-preinstalled SD card or eMMC, card instructions about how to flash an image of ubuntu linux are described in page 28 of Odroid C2 Manual [14].

6.2.1 Defining the IP address system

The MongoDB cluster used for lab testing is a minimal structure. The structure was already explained and is composed of:

- Three sharded replica set each of two replica nodes so a total of 6 nodes
- Three config servers having the same information of the cluster structure
- One arbiter
- One Application router.

The IP address system must be logic, mnemonic and consistent when the cluster is horizontally scaled. The cluster can be scaled in two ways:

- by adding sharded replica set in order to increase the capacity of the database
- increase the number of application routers to allow more applications to access the database in the same time

This are the reasons for the decision to allocate different IP subclass for sharded replica sets and separate IP subclass "management" nodes as config servers and application routers.

The cluster is behind a router that can assign IP by a suitable protocol as DHCP but in this case static IP allocation configured in each node is used.

The network behind the router was defined as an 10.x.x.x private class A network. The router has a static address of 10.1.1.2 and is the gateway for the cluster

The WiseGRID clusters have IPs in the 10.50.x.x sub class using different sub class as 10.50.1.x, 10.50.2.x for each node category. In each sub class the IP used are starting with 10.50.x.11 keeping 10.50.x.0 to 10.50.x.10 as reserved IPs for different purposes.

For the Mongo DB cluster, the 10.50.1. xx subclass is allocated for sharded replica sets and 10.50.2.xx for management and application routers.

The IPs and host names used in mongoDB cluster are in the following tables.

Node	IP	First name	Second name
Replica 0 in shard 0	10.50.1.11	mgdb1_m	sh0r0
Replica 1 in shard 0	10.50.1.12	mgdb2_m	sh0r1
Replica 0 in shard 1	10.50.1.13	mgdb3_m	sh1r0
Replica 1 in shard 1	10.50.1.14	mgdb4_m	sh1r1
Replica 0 in shard 2	10.50.1.15	mgdb5_m	sh2r0
Replica 1 in shard 2	10.50.1.16	mgdb6_m	sh2r1

Table 3 – IPs for sharded replica set

Each node has two names:

- First name for hardware management use and the structure is mgdb#_m where mgdb come from MongoDB , the number of node and the termination _m indicate that is using a 64GB eMMC local storage
- Second name is the function in the cluster and will be used in configuration of MongoDB cluster

The “management” nodes IP from the 10.50.2.x IPs are in the following table

Node	IP	First name	Second name
Config server 0	10.50.2.11	mgdb7_s	config0
Config server 1	10.50.2.12	mgdb8_s	config1
Config server 2	10.50.2.13	mgdb9_s	config 2
Arbiter	10.50.2.14	mgdb10_s	arbiter
Application router 0	10.50.2.15	mgdb11_s	router0

Table 4 – IP for management and application routers nodes

The hardware management names have the same structure while the termination _s indicate that the SBC use 16GB SD card for local storage. These nodes do not store database data but only cluster configuration metadata and uses only processing power so a much cheaper SD card can be used.

6.2.2 Preparation of the nodes for cluster configuration

Linux administrator skills will be required for next chapters.

In order to assign for each node, the role into the cluster and start the cluster some prerequisites configurations has to be done.

All configuration for the nodes will be done by using ssh access from preferably a Linux or MacOSx station in the network or by a Windows station using Putty ssh client.

First, we have to change the default passwords. in the beginning for each node a DHCP address will be automatically allocated so after finding the address in router list of dhcp leases the node can be accessed by SSH

```
ssh odroid@temporary_node_IP
```

The default password is odroid

```
sudo passwd
```

Change password to a suitable password that will be used on all nodes. For lab testing cluster was used "Wisegrid18"

First the nodes will be configured for static IP. Ubuntu mate that is preinstalled on the local storage for Odroid C2 is using DNSmasq application that is no use for the cluster nodes so it has to be deactivated by:

```
sudo vi /etc/NetworkManager/NetworkManager.conf
```

comment the line :

```
dns= dnsmasq
```

by changing to

```
#dns = dnsmask
```

Reboot the node by

```
sudo reboot
```

After reboot log in the node as root by

```
ssh root@temporary_node_ip
```

Install Midnight Commander as configuration tool:

```
apt update
```

```
apt install mc
```

edit /etc/network/interfaces for static IP allocation:

```
vi /etc/network/interfaces
```

add following lines

```
auto eth0
iface eth0 inet static
    address 10.50.1.11
    netmask 255.0.0.0
    gateway 10.1.1.2
    dns-nameservers 10.1.1.1 10.1.1.2 8.8.8.8
```

The yellow high lightened address must be changed for each node with suitable address. The gateway and dns-nameservers lines will be adapted to the host network and routers.

Reboot the node. Login as root. After checking the internet access from the nodes from the new IP make an update of the operating system :

```
apt update
```

```
apt upgrade
```

Set hostname by editing /etc/hostname:

```
vi /etc/hostname
```

add **mgdb1_m** and adapted for all nodes conforming the table 1 and 2

All the nodes must know the names for all other nodes so the /etc/hosts file on all nodes must have the following content:

```
127.0.0.1    localhost
127.0.0.1    odroid64
::1         localhost ip6-localhost ip6-loopback
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
```

```
10.50.1.11  mgdb1_m sh0r0
10.50.1.12  mgdb2_m sh0r1
10.50.1.13  mgdb3_m sh1r0
10.50.1.14  mgdb4_m sh1r1
10.50.1.15  mgdb5_m sh2r0
10.50.1.16  mgdb6_m sh2r1
10.50.2.11  mgdb7_s config0
10.50.2.12  mgdb8_s config1
10.50.2.13  mgdb9_s config2
10.50.2.14  mgdb10_s arbiter
10.50.2.15  mgdb11_s router0
10.50.3.11  xu4_1 master0
10.50.3.12  xu4_2 slave1
10.50.3.13  xu4_3 slave2
10.50.3.14  xu4_4 slave3
10.50.3.15  xu4_5 slave4
```

After this the prerequisite configuration prior of MongoDB installation can be considered done

6.3 INSTALLATION OF MANAGEMENT SOFTWARE: WEBMIN

A management software that can simplify the cluster management is a tool that should be installed. As indicated in D5.1 Webmin is a good candidate for a management software.

Webmin is a modern, web control panel for any Linux machine. It allows to administer a server through a simple interface. With Webmin, changes for settings for common packages on the fly are easy to be done [15].

The installation can be done through “apt” package manager since Webmin has a dedicated repository. The procedure of installing Webmin on Ubuntu 16.04 LTS computers is described in DigitalOcean tutorial: How to Install Webmin on Ubuntu 16.04 [15].

The following procedure must be repeated on all nodes.

Connect to each node using ssh as root.

```
nano /etc/apt/sources.list
```

Then add this line to the bottom of the file to add the new repository:

```
deb http://download.webmin.com/download/repository sarge contrib
```

Save the file and exit the editor.

Next, add the Webmin PGP key so the node will trust the new repository:

```
wget http://www.webmin.com/jcameron-key.asc
apt-key add jcameron-key.asc
```

Install Webmin by:

```
apt-get update
apt-get install webmin
```

Once the installation finishes, the following output will be presented:

Output

```
Webmin install complete. You can now login to
https://your_server_ip:10000 as root with your
root password, or as any user who can use `sudo`.
```

Connecting to one node by https://node_ip:10000 and logging in with root password the following picture should appear:

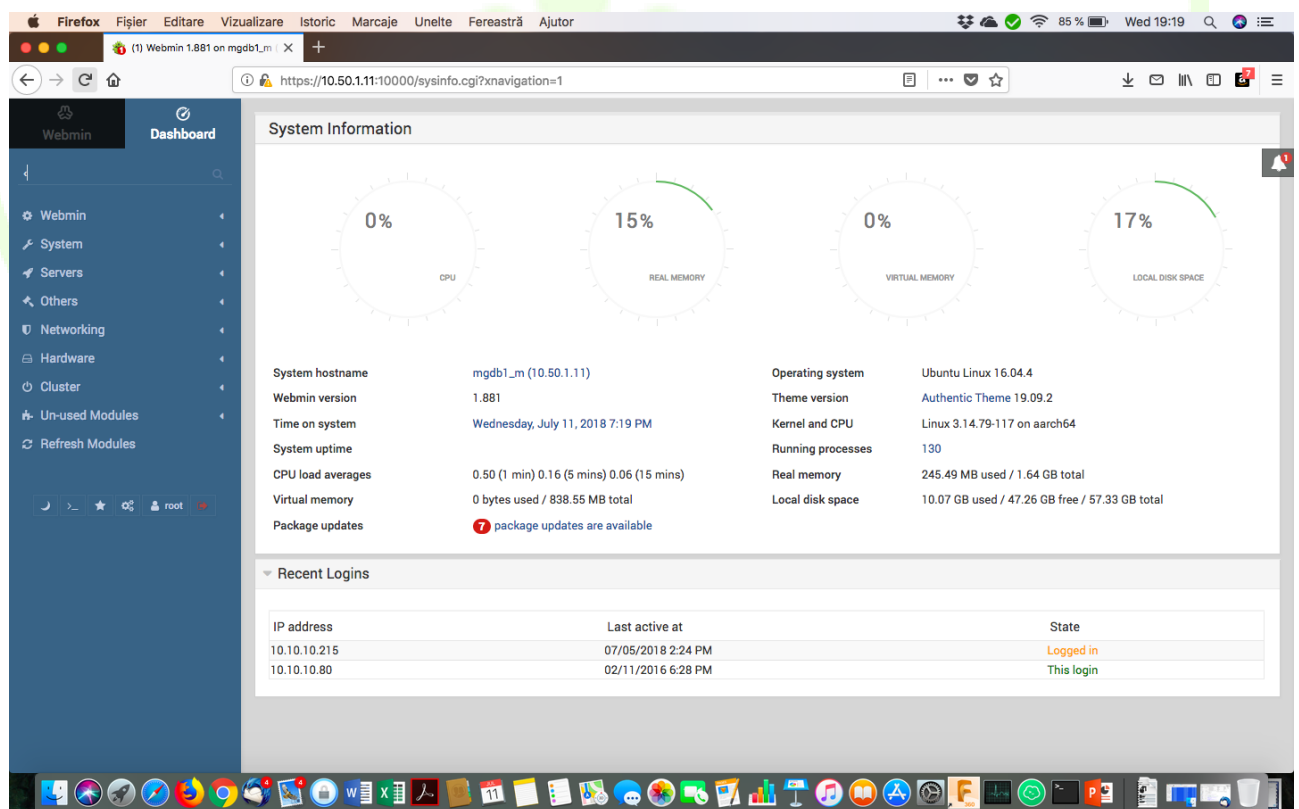


Figure 20 – Webmin dashboard on a cluster node

A good setting for beginning using Webmin is to set the automatic update for packages as in the following screen

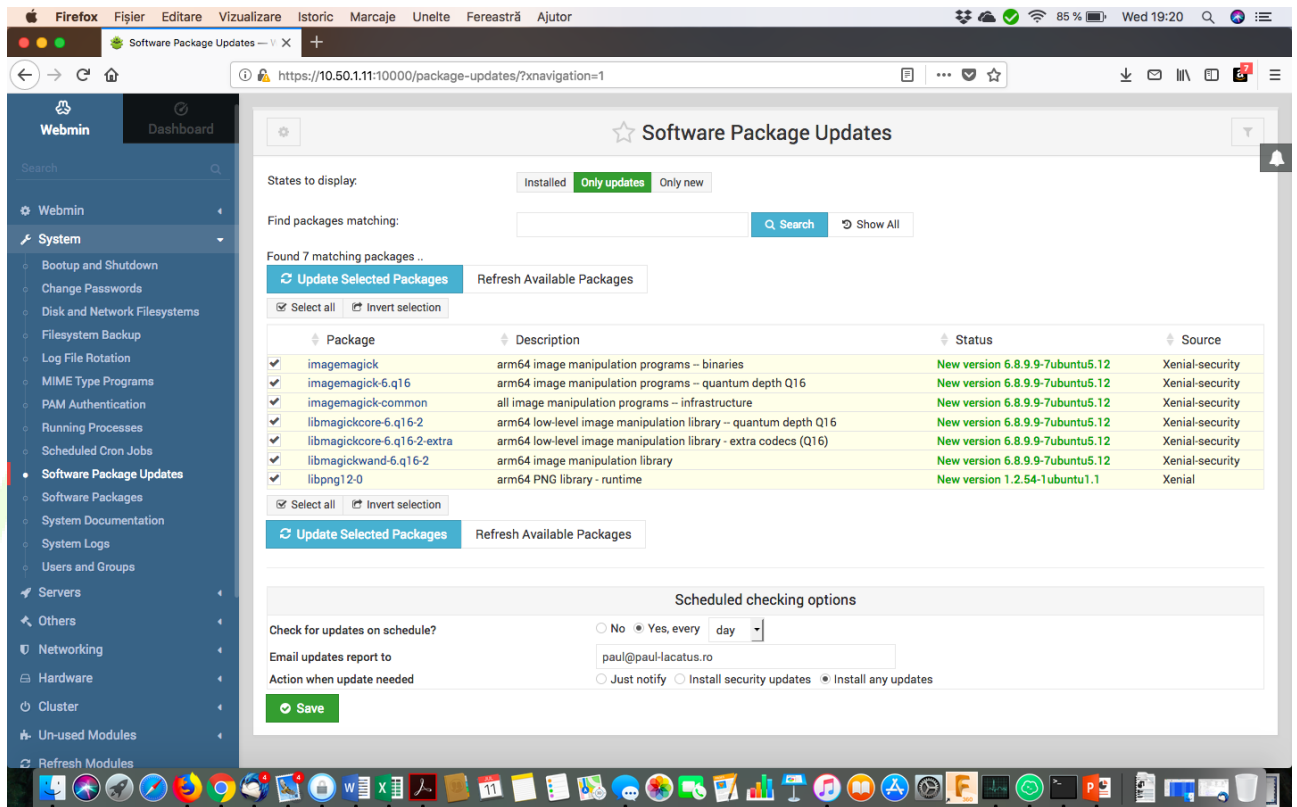


Figure 21 – Webmin screen for automatic updates

6.4 INSTALLATION OF THE MONGODB ENTERPRISE

To assign cluster roles to the nodes a prerequisite is to have installed MongoDB on each cluster nodes. For the MongoDB installation the OS should be up to date . To check use following commands on each cluster node:

```
root@mgdb1_m:~# uname -a
```

```
Linux mgdb1_m 3.14.79-117 #1 SMP PREEMPT Tue Jan 2 23:46:30 BRST 2018 aarch64 aarch64
aarch64 GNU/Linux
```

```
root@mgdb1_m:~# cat /etc/os-release
```

```
NAME="Ubuntu"
```

```
VERSION="16.04.4 LTS (Xenial Xerus)"
```

```
ID=ubuntu
```

```
ID_LIKE=debian
```

```
PRETTY_NAME="Ubuntu 16.04.4 LTS"
```

```
VERSION_ID="16.04"
```

```
HOME_URL=http://www.ubuntu.com/
```

```
SUPPORT_URL="http://help.ubuntu.com/"
```

```
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
```

```
VERSION_CODENAME=xenial
```

```
UBUNTU_CODENAME=xenial
```

An installation of a MongoDB cluster on Odroid C2 computers is described in AndyFelong.com tutorial UPDATE: MongoDB 3.6 on ODROID C2 with Ubuntu 16.04.3 – ARM64 [16]

Add the MongoDB repository and update the packages list. Install any updates and then install MongoDB:

```
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
```

```
echo "deb [ arch=amd64,arm64,ppc64el,s390x ] http://repo.mongodb.com/apt/ubuntu xenial/mongodb-
enterprise/3.6 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-enterprise.list
```

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install mongodb-enterprise
```

After the package installation concludes the version installed can be checked by :

```
root@mgdb1_m:~# mongod --version
```

```
db version v3.6.6
```

```
git version: 6405d65b1d6432e138b44c13085d0c2fe235d6bd
```

```
OpenSSL version: OpenSSL 1.0.2g 1 Mar 2016
```

```
allocator: tcmalloc
```

```
modules: enterprise
```

```
build environment:
```

```
distmod: ubuntu1604
```

```
distarch: aarch64
```

```
target_arch: aarch64
```

```
root@mgdb1_m:~# mongo --version
```

```
MongoDB shell version v3.6.6
```

```
git version: 6405d65b1d6432e138b44c13085d0c2fe235d6bd
```

```
OpenSSL version: OpenSSL 1.0.2g 1 Mar 2016
```

```
allocator: tcmalloc
```

```
modules: enterprise
```

```
build environment:
```

```
distmod: ubuntu1604
```

```
distarch: aarch64
```

```
target_arch: aarch64
```

The mongod service can be started by:

```
root@mgdb1_m:~# service mongod start
```

```
root@mgdb1_m:~# service mongod status
```

```
mongod.service - High-performance, schema-free document-oriented database
```

```
Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor preset:
```

```
Active: active (running) since Wed 2018-07-11 12:59:32 EDT; 8s ago
```

```
Docs: https://docs.mongodb.org/manual
```

```
Main PID: 4386 (mongod)
```

```
CGroup: /system.slice/mongod.service
```

```
â4386 /usr/bin/mongod --config /etc/mongod.conf
```

To start the mongod service upon boot of the node the following command can be used :

```
root@mgdb1_m:~# systemctl enable mongod
```

```
Created symlink from /etc/systemd/system/multi-user.target.wants/mongod.service to /lib/systemd/system/mongod.service.
```

The above steps for MongoDB installation must be applied on all MongoDB cluster nodes.

6.5 INSTALLATION OF MONGODB CLUSTER

The installation of a MongoDB cluster is not a trivial procedure and should be done with caution after reading carefully the above instructions and used bibliography. The general procedure is to log on the nodes as “root” but with caution since using “root” user provide all access rights and any error command will have all the rights. An alternative is to use a sudo group user preceding the commands that request root access with “sudo”.

After the installation of MongoDB on all nodes a user mongodb and a group mongodb are created by the installation script. This can be checked by:

```
root@mgdb1_m:~# cat /etc/passwd | grep mongodb
```

```
mongodb:x:108:65534::/home/mongodb:/bin/false
```

and for groups:

```
root@mgdb1_m:~# cat /etc/group | grep mongodb
mongodb:x:114:mongodb
```

The next step is to create a data folder on each node to be used by the MongoDB processes

```
root@mgdb2_m:~# mkdir /data
root@mgdb2_m:~# mkdir /data/db
root@mgdb2_m:~# mkdir /data/mongo-metadata
```

The `/data` folders must to be owned by the user running mongoDB processes `mongodb`.

```
root@mgdb2_m:~# chown -R mongodb:mongodb /data
```

There are many tutorials online showing procedures of MongoDB sharded cluster deployment that are mentioned in the bibliography but the procedure that will be described bellow is not following exactly none of the tutorials.

This are the tutorials:

[CodingMiles.com](#) Step by step Mongo DB sharded cluster deployment [17]

[www.guru99.com](#) MongoDB Sharded Cluster - Step by Step Implementation [18]

[www.hatcher.com](#) How to deploy a MongoDB cluster (version 3.4) [19]

[www.alibabacloud.com](#) High-availability MongoDB Cluster Configuration Solutions [20]

6.5.1 Setting of configuration servers

The configuration servers have a very important role in the cluster since they hold information about the cluster structure and roles of each node in the cluster. In production cluster is recommended to have three config server nodes. In the lab testing minimal cluster the config servers are :

Node	IP	First name	Second name
Config server 0	10.50.2.11	mgdb7_s	config0
Config server 1	10.50.2.12	mgdb8_s	config1
Config server 2	10.50.2.13	mgdb9_s	config 2

Table 5 – Config servers IP and names

For the config server the mongoDB process is `mongod`. The cluster role for the config servers is “configsvr” and replication setName is “cs”

The mongod process is using as configuration file `/etc/mongod.conf`

The mongod.conf file should be adapted in order to:

- **Allow access from any IP to mongod process on port 27017**

The section net: will have the content

net:

port: 27017

bindIp: 0.0.0.0

- **Set the data folder to /data/mongo-metadata**

The storage section will have the content:

storage:

dbPath: /data/mongo-metadata

journal:

enabled: true

- **Define replication name as “cs”, configuration server**

The section replication will contain:

replication:

replSetName: cs

- Define the cluster role as configsvr

The section sharding will contain:

sharding:

clusterRole: configsvr

The configuration file for a config server contents is:

```
root@mgdb7_s:~# cat /etc/mongod.conf
```

```
# mongod.conf
```

```
# for documentation of all options, see:
```

```
# http://docs.mongodb.org/manual/reference/configuration-options/
```

```
# Where and how to store data.
```

```
storage:
```

```
dbPath: /data/mongo-metadata
```

```
journal:
```

```
enabled: true
```

```
# engine:
```

```
# mmapv1:
```

```
# wiredTiger:
```

```
# where to write logging data.
```

```
systemLog:
```

```
destination: file
```

```
logAppend: true
```

```
path: /var/log/mongodb/mongod.log
```

```
# network interfaces
```

```
net:
```

```
port: 27017
```

```
bindIp: 0.0.0.0
```

```
# how the process runs
```

```
processManagement:
```

```
timeZoneInfo: /usr/share/zoneinfo
```

```
#security:
```

```
#operationProfiling:
```

```
replication:
```

```
replSetName: cs
```

```
sharding:
```

```
clusterRole: configsvr
```

```
## Enterprise-Only Options:
```

```
#auditLog:
```

```
#snmp:
```

The configuration file above should be replicated on all configuration servers in lab testing cluster case config0, config1, config 2

6.5.2 Activation of config servers

Now the three configuration servers are configured, the mongod daemon should be restarted to consider the new settings.

```
systemctl restart mongod
```

```
systemctl status mongod
```


mongod.service - High-performance, schema-free document-oriented database

Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)

Active: active (running) since Thu 2018-07-12 04:25:03 EDT; 36s ago

Docs: <https://docs.mongodb.org/manual>

Main PID: 17004 (mongod)

CGroup: /system.slice/mongod.service

â17004 /usr/bin/mongod --config /etc/mongod.conf

Jul 12 04:25:03 mgdb1_m systemd[1]: Started High-performance, schema-free document-oriented database.

The initiate of the config servers is done by connection on one of the config servers through mongo shell

root@mgdb7_s:~# mongo

MongoDB shell version v3.6.6

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.6.6

Server has startup warnings:

2018-07-12T03:54:00.317-0400 I STORAGE [initandlisten]

*2018-07-12T03:54:00.317-0400 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine*

*2018-07-12T03:54:00.317-0400 I STORAGE [initandlisten] ** See <http://dochub.mongodb.org/core/prodnotes-filesystem>*

2018-07-12T03:54:03.411-0400 I CONTROL [initandlisten]

*2018-07-12T03:54:03.411-0400 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.*

*2018-07-12T03:54:03.411-0400 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.*

2018-07-12T03:54:03.411-0400 I CONTROL [initandlisten]

MongoDB Enterprise cs:SECONDARY>

And sending the commands to the MongoDB

rs.initiate(

```
{
  _id: "cs",
  configsvr: true,
  members: [
    { _id: 0, host: "config0:27017" },
```

```
{_id : 1, host : "config1:27017" },
{_id : 2, host : "config2:27017" }
]
}
)
```

The replica set of config servers should be running

6.5.3 Setting of replicated shards

The replica shards server nodes contain the data of the databases processed by the mongoDB cluster. In the lab testing minimal cluster there are three shards, each one composed by a two replica .

Node	IP	First name	Second name
Replica 0 in shard 0	10.50.1.11	mgdb1_m	sh0r0
Replica 1 in shard 0	10.50.1.12	mgdb2_m	sh0r1
Replica 0 in shard 1	10.50.1.13	mgdb3_m	sh1r0
Replica 1 in shard 1	10.50.1.14	mgdb4_m	sh1r1
Replica 0 in shard 2	10.50.1.15	mgdb5_m	sh2r0
Replica 1 in shard 2	10.50.1.16	mgdb6_m	sh2r1

Table 6 – Replica shards servers

For the sharded servers the mongoDB process is mongod. The cluster role for the sharded servers is “shardsvr”

and replication setName are “rs0 “, “rs1”, “rs2” for each replicated shard

The mongod process is using as configuration file /etc/mongod.conf

The mongod.conf file should be adapted in order to:

- Allow access from any IP to mongod process on port 27017

The section net: will have the content

net:

port: 27017

bindIp: 0.0.0.0

- Set the data folder to /data/mongo-metadata

The storage section will have the content:

storage:

dbPath: /data/mongo-metadata

journal:

enabled: true

- Define replication name as rs0, rs1, rs2 for each replica shard server

The section replication will contain:

replication:

replSetName: rs0

for replica shards in rs0

replication:

replSetName: rs1

for replica shards in rs1

replication:

replSetName: rs2

for replica shards in rs2

- Define the cluster role as shardsvr

The section sharding will contain:

sharding:

clusterRole: shardsvr

The configuration file for a shard server is rs0 contents is:

```
root@mgdb1_m:~# cat /etc/mongod.conf
```

```
# mongod.conf
```

```
# for documentation of all options, see:
```

```
# http://docs.mongodb.org/manual/reference/configuration-options/
```

```
# Where and how to store data.
```

```
storage:
```

```
dbPath: /data/mongo-metadata
```

```
journal:
```

```
enabled: true
```

```
# engine:
```

```
# mmapv1:
```

```
# wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:
#operationProfiling:

replication:
  replSetName: rs0

sharding:
  clusterRole: shardsvr

## Enterprise-Only Options:
#auditLog:
#snmp:
```

The same configuration file should be replicated in sh0r1 server

The configuration file for a shard server is rs1 contents is:

```
root@mgdb3_m:~# cat /etc/mongod.conf
# mongod.conf
```

for documentation of all options, see:

<http://docs.mongodb.org/manual/reference/configuration-options/>

Where and how to store data.

storage:

dbPath: /data/mongo-metadata

journal:

enabled: true

engine:

mmapv1:

wiredTiger:

where to write logging data.

systemLog:

destination: file

logAppend: true

path: /var/log/mongodb/mongod.log

network interfaces

net:

port: 27017

bindIp: 0.0.0.0

how the process runs

processManagement:

timezoneInfo: /usr/share/zoneinfo

#security:

#operationProfiling:

replication:

replSetName: rs1

sharding:

clusterRole: shardsvr

```
## Enterprise-Only Options:
```

```
#auditLog:
```

```
#snmp:
```

The same configuration file should be replicated in sh1r1 server

The configuration file for a shard server is rs2 contents is:

```
root@mgdb5_m:~# cat /etc/mongod.conf
```

```
# mongod.conf
```

```
# for documentation of all options, see:
```

```
# http://docs.mongodb.org/manual/reference/configuration-options/
```

```
# Where and how to store data.
```

```
storage:
```

```
  dbPath: /data/mongo-metadata
```

```
  journal:
```

```
    enabled: true
```

```
# engine:
```

```
# mmapv1:
```

```
# wiredTiger:
```

```
# where to write logging data.
```

```
systemLog:
```

```
  destination: file
```

```
  logAppend: true
```

```
  path: /var/log/mongodb/mongod.log
```

```
# network interfaces
```

```
net:
```

```
  port: 27017
```

```
  bindIp: 0.0.0.0
```

```
# how the process runs
```

```
processManagement:
```

```
  timeZoneInfo: /usr/share/zoneinfo
```

```
#security:
```

```
#operationProfiling:
```

```
replication:
```

```
replSetName: rs2
```

```
sharding:
```

```
clusterRole: shardsvr
```

```
## Enterprise-Only Options:
```

```
#auditLog:
```

```
#snmp:
```

The same configuration file should be replicated in sh2r1 server.

6.5.4 Activation of replica shards servers

Now the six replica shards servers are configured, the mongod daemon should be restarted to consider the new settings.

```
systemctl restart mongod
```

```
systemctl status mongod
```

```
mongod.service - High-performance, schema-free document-oriented database
```

```
Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
```

```
Active: active (running) since Thu 2018-07-12 04:25:03 EDT; 36s ago
```

```
Docs: https://docs.mongodb.org/manual
```

```
Main PID: 17004 (mongod)
```

```
CGroup: /system.slice/mongod.service
```

```
â17004 /usr/bin/mongod --config /etc/mongod.conf
```

```
Jul 12 04:25:03 mgdb1_m systemd[1]: Started High-performance, schema-free document-oriented database.
```

The initiate of the each replica shard is done by connection on the primary server of each replica shard trough mongo shell

```
root@mgdb1_s:~# mongo
```

```
MongoDB shell version v3.6.6
```

```
connecting to: mongodb://127.0.0.1:27017
```

```
MongoDB server version: 3.6.6
```


Server has startup warnings:

2018-07-12T03:54:00.317-0400 | STORAGE [initandlisten]

2018-07-12T03:54:00.317-0400 | STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine

2018-07-12T03:54:00.317-0400 | STORAGE [initandlisten] ** See <http://dochub.mongodb.org/core/prodnotes-filesystem>

2018-07-12T03:54:03.411-0400 | CONTROL [initandlisten]

2018-07-12T03:54:03.411-0400 | CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.

2018-07-12T03:54:03.411-0400 | CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.

2018-07-12T03:54:03.411-0400 | CONTROL [initandlisten]

MongoDB Enterprise cs:SECONDARY>

And sending the commands to the MongoDB for rs0

```
rs.initiate( {
  _id : "rs0",
  members: [ { _id : 0, host : "sh0r0:27017" } ]
})
rs.add("sh0r1:27017")
rs.addArb("arbiter:30000")
rs.status()
{ "set" : "rs0",
  "date" : ISODate("2018-07-12T09:19:02.405Z"),
  "myState" : 2,
  "term" : NumberLong(4),
  "syncingTo" : "sh0r1:27017",
  "syncSourceHost" : "sh0r1:27017",
  "syncSourceId" : 1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1531387136, 2),
      "t" : NumberLong(4)
    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1531387136, 2),
      "t" : NumberLong(4)
    }
  }
}
```

```

    },
    "appliedOpTime" : {
      "ts" : Timestamp(1531387136, 2),
      "t" : NumberLong(4)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1531387136, 2),
      "t" : NumberLong(4)
    }
  },
  "members" : [
    {
      "_id" : 0,
      "name" : "sh0r0:27017",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 3239,
      "optime" : {
        "ts" : Timestamp(1531387136, 2),
        "t" : NumberLong(4)
      },
      "optimeDate" : ISODate("2018-07-12T09:18:56Z"),
      "syncingTo" : "sh0r1:27017",
      "syncSourceHost" : "sh0r1:27017",
      "syncSourceId" : 1,
      "infoMessage" : "",
      "configVersion" : 3,
      "self" : true,
      "lastHeartbeatMessage" : ""
    },
    {
      "_id" : 1,
      "name" : "sh0r1:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",

```

```

    "uptime" : 3234,
    "optime" : {
      "ts" : Timestamp(1531387136, 2),
      "t" : NumberLong(4)
    },
    "optimeDurable" : {
      "ts" : Timestamp(1531387136, 2),
      "t" : NumberLong(4)
    },
    "optimeDate" : ISODate("2018-07-12T09:18:56Z"),
    "optimeDurableDate" : ISODate("2018-07-12T09:18:56Z"),
    "lastHeartbeat" : ISODate("2018-07-12T09:19:01.744Z"),
    "lastHeartbeatRecv" : ISODate("2018-07-12T09:19:01.151Z"),
    "pingMs" : NumberLong(1),
    "lastHeartbeatMessage" : "",
    "syncingTo" : "",
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "infoMessage" : "",
    "electionTime" : Timestamp(1531383912, 1),
    "electionDate" : ISODate("2018-07-12T08:25:12Z"),
    "configVersion" : 3
  },
  {
    "_id" : 2,
    "name" : "arbiter:30000",
    "health" : 0,
    "state" : 8,
    "stateStr" : "(not reachable/healthy)",
    "uptime" : 0,
    "lastHeartbeat" : ISODate("2018-07-12T09:19:01.240Z"),
    "lastHeartbeatRecv" : ISODate("1970-01-01T00:00:00Z"),
    "pingMs" : NumberLong(0),
    "lastHeartbeatMessage" : "Connection refused",
    "syncingTo" : "",
    "syncSourceHost" : "",
    "syncSourceId" : -1,

```

```

    "infoMessage" : "",
    "configVersion" : -1
  },
  ],
  "ok" : 1,
  "operationTime" : Timestamp(1531387136, 2),
  "$gleStats" : {
    "lastOpTime" : Timestamp(0, 0),
    "electionId" : ObjectId("000000000000000000000000")
  },
  "$clusterTime" : {
    "clusterTime" : Timestamp(1531387138, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "$configServerState" : {
    "opTime" : {
      "ts" : Timestamp(1531387115, 6),
      "t" : NumberLong(4)
    }
  }
}

```

The answer might differ since this example is from a running cluster, but the two members of each shard has to be indicated

The commands to initiate rs1 will be sent to sh1r0 are:

```

rs.initiate( {
  _id : "rs1",
  members: [ { _id : 0, host : "sh1r0:27017" } ]
})
rs.add("sh1r1:27017")
rs.addArb("arbiter:30001")
rs.status()

```

The reply from the second shard should indicate the status of sh1r0 and sh1r1

The commands to initiate rs2 will be sent to sh2r0 and are:

```
rs.initiate( {  
  _id : "rs2",  
  members: [ { _id : 0, host : "sh2r0:27017" } ]  
})  
rs.add("sh2r1:27017")  
rs.addArb("arbiter:30002")  
rs.status()
```

The reply from the third shard should indicate the status of sh1r0 and sh1r1

6.5.5 Configuration and activation of arbiter instances

Arbiters are mongod instances that are part of a replica set but do not hold data. Arbiters participate in elections in order to break ties. If a replica set has an even number of members, add an arbiter.

Arbiters have minimal resource requirements and do not require dedicated hardware. You can deploy an arbiter on an application server or a monitoring host.

In the lab testing cluster, one node is used to run three instances of mongod , one for each replica set . For each instance of mongod a separate data folder is used.

First step is creating the data folders on the arbiter node and allocating them to the mongoddb user:

```
root@mgdb10_s:~# mkdir /data  
root@mgdb10_s:~# mkdir /data/arb0  
root@mgdb10_s:~# mkdir /data/arb1  
root@mgdb10_s:~# mkdir /data/arb2  
root@mgdb10_s:~# chown -R mongoddb:mongoddb /data
```

The next step is launching an instance of mongod for each replica set rs0,rs1,rs2 using each a separate data folder /data/arb0, /data /arb1, /data/arb2 .

```
sudo -u mongoddb mongod --port 30000 --dbpath /data/arb0 --replSet rs0 --bind_ip 0.0.0.0 &  
sudo -u mongoddb mongod --port 30001 --dbpath /data/arb1 --replSet rs1 --bind_ip 0.0.0.0 &  
sudo -u mongoddb mongod --port 30002 --dbpath /data/arb2 --replSet rs2 --bind_ip 0.0.0.0 &
```

6.5.6 Configuration and activation of application routers

The application routers are mongoDB cluster nodes that does not hold data but are processing the applications using the database requests and sending them to the cluster. the application routers use mon-gos process instead of mongod used by configuration servers and replica shard servers.

In order to connect to the cluster, the mongos daemon for the application router should know the nodes of the cluster that are running the role of the config servers

Mongos instance can use a similar config file as the /etc/mongod.conf that are indicating the nodes that are running the config server's role.

The sharding field contain the names and port for the config servers:

sharding:

configDB: cs/config0:27017,config1:27017,config2:27017

The complete configuration file is :

```
root@mgdb11_s:~# cat /etc/mongod.conf
# mongod.conf

# for documentation of all options, see:
# http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
#storage:
# dbPath: /var/lib/mongodb
# journal:
# enabled: true
# engine:
# mmapv1:
# wiredTiger:

# where to write logging data.
systemLog:
destination: file
logAppend: true
```

```
path: /var/log/mongodb/mongod.log
```

```
# network interfaces
```

```
net:
```

```
port: 27017
```

```
bindIp: 0.0.0.0
```

```
# how the process runs
```

```
processManagement:
```

```
timeZoneInfo: /usr/share/zoneinfo
```

```
#security:
```

```
#operationProfiling:
```

```
#replication:
```

```
sharding:
```

```
configDB: cs/config0:27017,config1:27017,config2:27017
```

```
## Enterprise-Only Options:
```

```
#auditLog:
```

```
#snmp:
```

For running the application router from the node startup is needed to have a service file mongos.service . The easiest way is to copy the existing file from mongod.service and adapt it:

```
cd /lib/systemd/system
```

```
cp mongod.service mongos.service
```

```
nano mongod.service
```

The mongos.service file content is

```
[Unit]
```

```
Description=High-performance, schema-free document-oriented database application server
```

```
After=network.target
```

```
Documentation=https://docs.mongodb.org/manual
```

```
[Service]
```



```
User=mongod
Group=mongod
ExecStart=/usr/bin/mongod --config /etc/mongod.conf
PIDFile=/var/run/mongodb/mongod.pid
# file size
LimitFSIZE=infinity
# cpu time
LimitCPU=infinity
# virtual memory size
LimitAS=infinity
# open files
LimitNOFILE=64000
# processes/threads
LimitNPROC=64000
# locked memory
LimitMEMLOCK=infinity
# total threads (user+kernel)
TasksMax=infinity
TasksAccounting=false

# Recommended limits for mongod as specified in
# http://docs.mongodb.org/manual/reference/ulimit/#recommended-settings

[Install]
WantedBy=multi-user.target
```

The mongod service should be disabled from systemctl and enabled the mongos service

```
systemctl disable mongos.service
systemctl enable mongos.service
systemctl start mongos.service
```

Test the status of mongos service

```
root@mgdb11_s:/lib/systemd/system# systemctl status mongos
mongos.service - High-performance, schema-free document-oriented database application server
Loaded: loaded (/lib/systemd/system/mongos.service; enabled; vendor preset: enabled)
Active: active (running) since Thu 2016-02-11 11:28:05 EST; 2 years 4 months ago
Docs: https://docs.mongodb.org/manual
```

Main PID: 865 (mongos)

CGroup: /system.slice/mongos.service

/usr/bin/mongos --config /etc/mongod.conf

mgdb11_s systemd[1]: Started High-performance, schema-free document-oriented database application server.

Connect to the mongos router through mongo shell on application router

```
root@mgdb11_s:~# mongo
```

```
MongoDB shell version v3.6.6
```

```
connecting to: mongod://127.0.0.1:27017
```

```
MongoDB server version: 3.6.6
```

```
Server has startup warnings:
```

```
2016-02-11T11:28:09.174-0500 I CONTROL [main]
```

```
2016-02-11T11:28:09.174-0500 I CONTROL [main] ** WARNING: Access control is not enabled for the database.
```

```
2016-02-11T11:28:09.174-0500 I CONTROL [main] ** Read and write access to data and configuration is unrestricted.
```

```
2016-02-11T11:28:09.174-0500 I CONTROL [main]
```

```
MongoDB Enterprise mongos>
```

The mongo shell confirms the connection to the mongos instance . the next steps is to add shards to the application router :

```
sh.addShard( "rs0/sh0r0:27017")
```

```
sh.addShard( "rs0/sh0r1:27017")
```

```
sh.addShard( "rs1/sh1r0:27017")
```

```
sh.addShard( "rs1/sh1r1:27017")
```

```
sh.addShard( "rs2/sh2r0:27017")
```

```
sh.addShard( "rs2/sh2r1:27017")
```

Create a “admin user”

```
use admin
```

```
db.createUser(
```

```
{
  user: "admin",
  pwd: "Wisegrid18",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
}
```

```
}  
)
```

Create a database for each Wisegrid application by creating a collection inside each database. After that the sharding on each database should be activated:

```
sh.enableSharding("WG_Cockpit")  
sh.enableSharding("WG_Coop")  
sh.enableSharding("WG_Corp")  
sh.enableSharding("WG_EVP")  
sh.enableSharding("WG_Resco")
```

The application router is running and we can test the connection to the cluster by MongoDB Compass connecting to the IP of the application router:

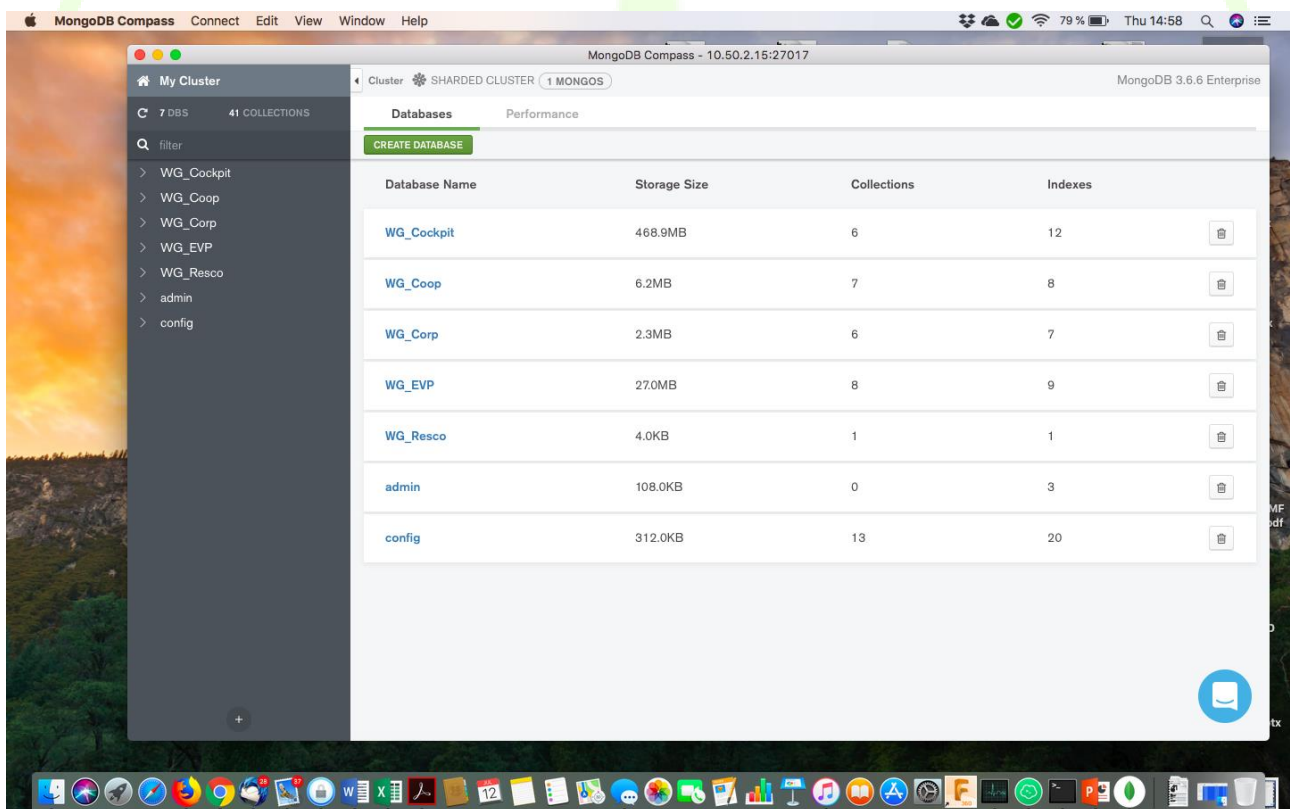


Figure 22 – Connection to the MongoDB application router trough MongoDB Compass

6.5.7 Final settings and online connection

For access to the MongoDB cluster from outside of private network the router is configured for port forwarding so the traffic to port 27027 to the external interface of the router is directed to port 27017 of the router0 node. Further adding of new application routers, the traffic to ports 27028... will be forwarded to port 27017 of router 1 and so on.



7 INSTALLATION OF THE APACHE HADOOP CLUSTER

7.1 PHYSICAL INSTALLATION OF THE ODROID XU4 SBC FOR RACK MOUNTING

The Apache Hadoop-Spark cluster has a minimal structure of five pieces of Odroid XU4 SBCs.

The Odroid XU4 SBC are installed on the same DIN rail with supports for 19" rack mount described in chapter 6.1

7.1.1 3D PRINTED SUPORTS FOR ODROID XU4

The Odroid XU4 used in lab testing are with passive cooling airflow so they only have a heatsink with no fans. This implies that the height of one Odroid XU4 SBC is higher than an Odroid XU4 with heatsink and fan. Also, the connector orientation is completely different from Odroid C2, so the DIN rail stands has to be redesigned. The DIN rail supports for Odroid XU4 were designed using the same Autodesk Fusion 360 with the bigger dimension on vertical axis to have the ethernet socket in front and a width of the support of 38 mm instead of 25 mm as in the odroid C2 SBC support. The support need an opening to allow the access to the boot media selector switch.

Images of the Odroid XU4 support are bellow.



Figure 23 – Odroid XU4 Din Rail Support view 1



Figure 24 – Odroid XU4 support view 2



Figure 25 – Odroid XU4 DIN rail support view 3

7.1.2 POWER SUPPLY OF THE ODROID XU4 SBCs

From the specifications published by Hardkernel is indicated that Odroid XU4 is drawing a maximum current of 4A at 5VDC per SBC. This information is from Odroid XU4 manual [21] page 4.

The minimal Odroid XU4 cluster is containing 5 SBC so considering a maximum of $5 \times 4 = 20A$ one SMPS power supply for 5Vdc at 20 A will fulfil the power requirements. For the cluster a DC power rail is created by Wago 221 connectors with 5 connection each allowing a maximum of 20 A. The Wago connectors are placed in 3d printed sockets that allows installation on the same DIN rail. The sockets and the DIN rail supports are designed by “ebraud” and published as: “Support rail DIN Wago 221” [13]

Each Odroid C4 will be connected separately to the positive and negative power rails by two wires. The Wago connectors allows each SBC to be separated powered for maintenance purposes. The wires can be soldered on the back of the power plug or by using a standard connector as in Odroid XU4 power outlet. The connector is available by Hardkernel as declared in Odroid XU4 manual [21] in page 4.

A picture of a minimal cluster of the five Odroid XU4 with the Wago 221 power rail and connections is in the following picture.

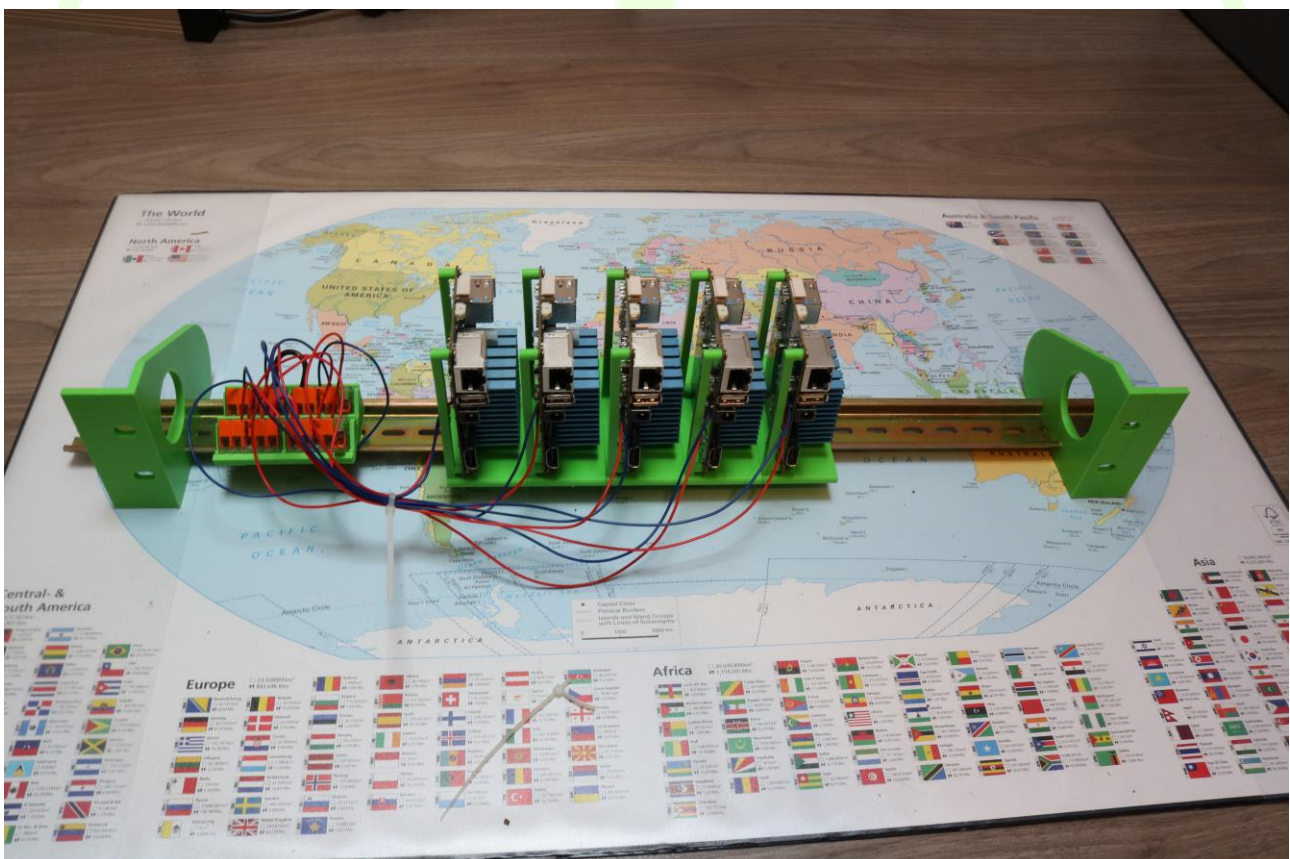


Figure 26 – Minimal Odroid XU4 Hadoop cluster

7.1.3 VENTILATION

The Odroid XU4 used for lab testing are provided with a heatsink able to dissipate the heat generated by the electronics on the board by a natural convection air flow. The distance from the Odroid Xu4 in the stack installed in supports described in chapter 7.1.1 is over 38 mm and allows the natural convection air flow. However, if the DIN rail is installed in a closed 19" rack a suitable forced ventilation has to be provided to the rack to evacuate heat equivalent of 20 W per each Odroid XU4 in the cluster.

7.2 INSTALLATION OF UBUNTU LINUX OPERATING SYSTEM

The Odroid XU4 eMMC or SD card ordered with the Linux version have preinstalled an image of Ubuntu 16.04 LTS. For using a non-preinstalled SD card or eMMC card instructions about how to flash an image of Ubuntu linux are described starting from in page 18 of Odroid XU4 Manual [21].

7.2.1 Defining the IP address system

The Hadoop-Spark cluster used for lab testing is a minimal structure. The structure was already explained and is composed of:

- One master node
- Four slave nodes

The IP address system must be logic, mnemonic and consistent when the cluster is horizontally scaled. The cluster can be scaled in two ways:

- by adding slave nodes in order to increase the capacity of processing of Hadoop Spark cluster
- by adding more master and slave nodes

This are the reasons for the decision to allocate different IP subclass from the mongoDB cluster

The cluster is behind a router that can assign IP by a suitable protocol as DHCP but in this case static IP allocation configured in each node is used.

The network behind the router was defined as a 10.x.x.x private class A network. The router has a static address of 10.1.1.2 and is the gateway for the cluster

The WiseGRID clusters have IPs in the 10.50.x.x sub class so for the Hadoop Spark cluster the class as 10.50.3.x is used in the sub class the IP used are starting with 10.50.3.11 keeping 10.50.3.0 to 10.50.3.10 as reserved IPs for different purposes.

The IPs and host names used in Hadoop Spark cluster are in the following table.

Node	IP	First name	Second name
Master0 node	10.50.3.11	Xu4_1	master0
Slave1	10.50.3.12	Xu4_2	slave1
Slave2	10.50.3.13	Xu4_3	Slave2
Slave3	10.50.3.14	Xu4_4	Slave3
Slave4	10.50.3.15	Xu4_5	Slave4

Table 7 – Names and IP for Hadoop Spark cluster

Each node has two names:

- First name for hardware management use and the structure is Xu4_#
- Second name is the function in the cluster and will be used in configuration of Hadoop Spark cluster

7.2.2 Preparation of the nodes for cluster configuration

Linux administrator skills will be required for next chapters.

In order to assign for each node, the role into the cluster and start the cluster some prerequisites configurations have to be done.

All configuration for the nodes will be done by using ssh access from preferably a Linux or MacOSx station in the network or by a Windows station using Putty ssh client.

It is needed to change the default passwords. In the beginning for each node a DHCP address will be automatically allocated, so after finding the address in router list of dhcp leases the node can be accessed by SSH

`ssh odroid@temporary_node_IP`

The default password is odroid

`sudo passwd`

Change password to a suitable password that will be used on all nodes. For lab testing cluster was used "Wisegrid18"

First the nodes will be configured for static IP. Ubuntu 1604 LTS that is preinstalled on the local storage for Odroid XU4 is using DNSmasq application that is no use for the cluster nodes so it has to be deactivated by:

```
sudo vi /etc/NetworkManager/NetworkManager.conf
```

comment the line :

```
dns= dnsmasq
```

by changing to

```
#dns = dnsmask
```

Reboot the node by

```
sudo reboot
```

After reboot log in the node as root by

```
ssh root@temporary_node_ip
```

Install Midnight Commander as configuration tool :

```
apt update
```

```
apt install mc
```

edit /etc/network/interfaces for static IP allocation :

```
vi /etc/network/interfaces
```

```
add following lines
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 10.50.3.11
```

```
netmask 255.0.0.0
```

```
gateway 10.1.1.2
```

```
dns-nameservers 10.1.1.1 10.1.1.2 8.8.8.8
```

The yellow high lightened address must be changed for each node with suitable address. The gateway and dns-nameservers lines will be adapted to the host network and routers.

Reboot the node. Login as root. After checking the internet access from the nodes from the new IP make an update of the operating system:

```
apt update
```

```
apt upgrade
```

Set hostname by editing /etc/hostname:

```
vi /etc/hostname
```

add **XU4_1** and adapted for all nodes conforming the Table 7 – Names and IP for Hadoop Spark cluster

All the nodes must know the names for all other nodes so the /etc/hosts file on all nodes must have the following content:

```
127.0.0.1    localhost
127.0.0.1    odroid64
::1         localhost ip6-localhost ip6-loopback
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters

10.50.1.11   mgdb1_m sh0r0
10.50.1.12   mgdb2_m sh0r1
10.50.1.13   mgdb3_m sh1r0
10.50.1.14   mgdb4_m sh1r1
10.50.1.15   mgdb5_m sh2r0
10.50.1.16   mgdb6_m sh2r1
10.50.2.11   mgdb7_s config0
10.50.2.12   mgdb8_s config1
10.50.2.13   mgdb9_s config2
10.50.2.14   mgdb10_s arbiter
10.50.2.15   mgdb11_s router0
10.50.3.11   xu4_1 master0 master
10.50.3.12   xu4_2 slave1
10.50.3.13   xu4_3 slave2
10.50.3.14   xu4_4 slave3
10.50.3.15   xu4_5 slave4
```

After this the prerequisite configuration prior of Hadoop installation can be considered done

7.3 INSTALLATION OF MANAGEMENT SOFTWARE, DISTRIBUTED CLI SOFTWARE

The installation of Webmin software is similar to the installation on the nodes of MongoDB cluster and described in the chapter 6.3. Another useful tool for distributed installation on cluster nodes is “parallel-ssh” that allows to run a similar command on more than one computer. Parallel-ssh is written in Python so a simple installation of Python pip will allow the use of the tool.

```
apt-get install python-pip
```

Parallel-ssh is using list files that contains the nodes for executing the same command. The list files are in the root home folder on the master0 node:

```
root@xu4_1:~# cat hadoop_cluster.txt
```

```
10.50.3.11
```

```
10.50.3.12
```

```
10.50.3.13
```

```
10.50.3.14
```

```
10.50.3.15
```

```
root@xu4_1:~# cat hadoop_slaves.txt
```

```
10.50.3.12
```

```
10.50.3.13
```

```
10.50.3.14
```

```
10.50.3.15
```

```
root@xu4_1:~# cat mongo_cluster.txt
```

```
10.50.1.12
```

```
10.50.1.13
```

```
10.50.1.14
```

```
10.50.1.15
```

```
10.50.1.16
```

```
10.50.2.11
```

```
10.50.2.12
```

```
10.50.2.13
```

```
10.50.2.14
```

```
10.50.2.15
```

Another tool that is used to deploy the software on several nodes of the cluster is Rsync . To install run:

```
apt-get install rsync
```

on all the cluster nodes

7.4 INSTALLATION OF THE HADOOP SOFTWARE

Hadoop software is written in Java so some prerequisites must be fulfilled prior of installation of cluster software.

Some tutorials can be used online to guide an installation of a Hadoop cluster on a small cluster of SBCs. This installation was directed by DIY Big Data: Installing Hadoop onto an ODROID XU4 Cluster [22]

Before installing Hadoop, let's discuss what we are trying to accomplish by installing it. Hadoop has three components: the Hadoop File System (HDFS), Yarn, and Map-Reduce. For our purposes, we are most interested in HDFS, but we will play around with the other two. What HDFS will do for us is turn the MicroSD cards installed onto every node into a single "virtual drive" that files can be written to for use by the cluster's analytics applications. HDFS will (if appropriate) break a large file up into parts and then distribute those parts around the cluster. The benefit of this is that when doing distributed computing operations on the file, work will be split up with each node being responsible for processing a set of parts [22].

7.4.1 Installing Java

Oracle's latest version of Java8 is needed on all the nodes of the cluster, on master0 and also in the slaves1 to slave4.

```
add-apt-repository ppa:webupd8team/java
```

```
apt-get update
```

```
apt-get install oracle-java8-installer
```

7.4.2 Preparing the user, groups and rights for installing Hadoop

A user account with the name of "hduser" member of group "Hadoop" is needed on all cluster nodes and this user need password less access on all the slave nodes. The use of parallel-ssh is useful for non-interactive commands. Prior to use parallel-ssh a ssh connection to all member on the cluster is needed to store the key on the master0 ssh files.

```
ssh root@10.50.3.11
```

```
ssh root@10.50.3.12
```

```
ssh root@10.50.3.13
```

```
ssh root@10.50.3.14
```

```
ssh root@10.50.3.15
```

```
parallel-ssh -i -h hadoop_cluster.txt -l root -A "addgroup hadoop"
```

To create the `hduser` in each node a direct connection by `ssh` in each node is required due to interactive mode of “`adduser`” command.

```
root@xu4_1:~# adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:Wisegrid18
Retype new UNIX password:Wisegrid18
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
Full Name []: Hadoop User
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] y
```

The “`hduser`” should be in `sudo` group

```
root@xu4_1:~# adduser hduser sudo
Adding user `hduser' to group `sudo' ...
Adding user hduser to group sudo
Done.
```

Giving “`hduser`” appropriate rights

```
root@xu4_1:~# parallel-ssh -i -h hadoop_cluster.txt -l root -A "usermod -aG sudo hduser"
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
Password:
[1] 11:53:35 [SUCCESS] 10.50.3.11
[2] 11:53:35 [SUCCESS] 10.50.3.12
[3] 11:53:35 [SUCCESS] 10.50.3.13
[4] 11:53:35 [SUCCESS] 10.50.3.14
```

```
[5] 11:53:35 [SUCCESS] 10.50.3.15
```

Allowing “hduser “ to access all the cluster nodes without password request:

```
root@xu4_1:~# su hduser
hduser@xu4_1:/root$ cd
hduser@xu4_1:~$ ssh-keygen -t rsa -P ""
hduser@xu4_1:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
hduser@xu4_1:~$ ssh hduser@localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:oKt67r3/uarmzOGNpXBAHq/0yzlfK7c0BUx8UZdtC50.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 3.10.105-141 armv7l)

* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:      https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
hduser@xu4_1:~$ exit
logout
Connection to localhost closed.

hduser@xu4_1:~$ ssh hduser@master0
```



```
The authenticity of host 'master0 (10.50.3.11)' can't be established.
ECDSA key fingerprint is SHA256:oKt67r3/uarmzOGNpXBAHq/0yzIfK7c0BUx8UZdtC50.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'master0,10.50.3.11' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 3.10.105-141 armv7l)
```

```
* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:      https://ubuntu.com/advantage
```

```
0 packages can be updated.
0 updates are security updates.
```

```
Last login: Thu Jul 5 12:09:22 2018 from ::1
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

```
hduser@xu4_1:~$
```

```
hduser@xu4_1:~$ ssh-copy-id hduser@slave1
```

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/hduser/.ssh/id_rsa.pub"
```

```
The authenticity of host 'slave1 (10.50.3.12)' can't be established.
ECDSA key fingerprint is SHA256:AhZViMThWYyS7/IkJEGZT9+EhAj7+IPxfzd+h/UBDo.
Are you sure you want to continue connecting (yes/no)? yes
```

```
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already
installed
```

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new
keys
```

```
hduser@slave1's password:
```

```
Number of key(s) added: 1
```

```
Now try logging into the machine, with: "ssh 'hduser@slave1'"
and check to make sure that only the key(s) you wanted were added.
```

Testing ssh acces without password on slave1

```
ssh hduser@slave1
```

repeat the *ssh-copy-id* on all the slaves and test the access.

Create the data folder by parallel-ssh

```
root@xu4_1:~# parallel-ssh -i -h hadoop_cluster.txt -l root -A "mkdir -p /data/hdfs/tmp"
```

Warning: do not enter your password if anyone else has superuser privileges or access to your account.

Password:

```
[1] 12:19:02 [SUCCESS] 10.50.3.11
```

```
[2] 12:19:03 [SUCCESS] 10.50.3.12
```

```
[3] 12:19:03 [SUCCESS] 10.50.3.13
```

```
[4] 12:19:03 [SUCCESS] 10.50.3.15
```

```
[5] 12:19:03 [SUCCESS] 10.50.3.14
```

Changing the owner of data folder

```
root@xu4_1:~# parallel-ssh -i -h hadoop_cluster.txt -l root -A "chown -R hduser:hadoop /data/hdfs"
```

Warning: do not enter your password if anyone else has superuser privileges or access to your account.

Password:

```
[1] 12:20:29 [SUCCESS] 10.50.3.11
```

```
[2] 12:20:29 [SUCCESS] 10.50.3.12
```

```
[3] 12:20:29 [SUCCESS] 10.50.3.13
```

```
[4] 12:20:29 [SUCCESS] 10.50.3.14
```

```
[5] 12:20:29 [SUCCESS] 10.50.3.15
```

7.4.3 Installing and deploying Hadoop software

Download the Hadoop software from diybigdata.net repository

```
cd /opt
```

```
wget http://diybigdata.net/downloads/hadoop/hadoop-2.7.2.armhf.tar.gz
```

Unpack the Hadoop package in the /opt directory. I also like creating a symlink to the install from the /usr/local directory. On the master node:

```
root@xu4_1:~# cd /opt
```

```
root@xu4_1:/opt# tar xzf hadoop-2.7.2.armhf.tar.gz
```

```
root@xu4_1:/opt# chown -R hduser:hadoop hadoop-2.7.2
```

```
root@xu4_1:/opt# cd /usr/local
```

```
root@xu4_1:/usr/local# ln -s /opt/hadoop-2.7.2 hadoop
```

The next step is to configure the Hadoop installation on the master node. This requires editing several configuration files found in the `/usr/local/hadoop/etc/hadoop` directory. The contents of the files is posted to the GitHub repository for the ODROID CU4 cluster project [22].

These changes tell Hadoop where it can find the Java libraries and sets the default heap size. Since our devices have 2 GB of RAM and we want to save as much RAM as possible for an Apache Spark install later, we are limiting Hadoop's heap to 384 MB [22].

```
vi etc/hadoop/hadoop-env.sh
```

Adjust the following two lines to match what is shown:

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
```

```
export HADOOP_HEAPSIZE=384
```

```
vi etc/hadoop/core-site.xml
```

Insert following lines between `<configuration></configuration>` tags

```
<property>
    <name>hadoop.tmp.dir</name>
    <value>/data/hdfs/tmp</value>
    <description>Where Hadoop will place all of its working files</description>
</property>
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
    <description>Where HDFS NameNode can be found on the network</description>
</property>
<property>
    <name>hadoop.proxyuser.hduser.groups</name>
    <value>*</value>
    <description>
        What user groups are allow to connect to the HDFS proxy.
        * for all.
    </description>
</property>

<property>
```

```
<name>hadoop.proxyuser.hduser.hosts</name>
<value>*</value>
<description>
    What user hosts are allow to connect to the HDFS proxy.
    * for all.
</description>
```

```
</property>
```

To configure HDFS component edit `hdfs-site.xml`:

```
vi etc/hadoop/hdfs-site.xml
```

Add the following values between the `<configuration>``</configuration>` tags.

```
<property>
    <name>dfs.replication</name>
    <value>2</value>
    <description>The default replication factor of files on HDFS</description>
</property>
<property>
    <name>dfs.block.size</name>
    <value>16777216</value>
    <description>The default block size in bytes of data saved to HDFS</description>
</property>
<property>
    <name>dfs.namenode.rpc-bind-host</name>
    <value>0.0.0.0</value>
    <description>
        controls what IP address the NameNode binds to.
        0.0.0.0 means all available.
    </description>
</property>
<property>
    <name>dfs.namenode.servicerpc-bind-host</name>
    <value>0.0.0.0</value>
    <description>
        controls what IP address the NameNode binds to.
        0.0.0.0 means all available.
```

```

    </description>
  </property>
  <property>
    <name>dfs.namenode.http-bind-host</name>
    <value>0.0.0.0</value>
    <description>
      controls what IP address the NameNode binds to.
      0.0.0.0 means all available.
    </description>
  </property>
  <property>
    <name>dfs.namenode.https-bind-host</name>
    <value>0.0.0.0</value>
    <description>
      controls what IP address the NameNode binds to.
      0.0.0.0 means all available.
    </description>
  </property>
  <property>
    <name>nfs.dump.dir</name>
    <value>/tmp/.hdfs-nfs</value>
    <description>A temporary working directory for files coming into the HDFS
proxy.</description>
  </property>
  <property>
    <name>nfs.metrics.percentiles.intervals</name>
    <value>100</value>
    <description>
      Enable the latency histograms for read, write and commit requests.
      The time unit is 100 seconds in this example.
    </description>
  </property>
  <property>
    <name>nfs.exports.allowed.hosts</name>
    <value>* rw</value>
    <description>Host permissions for connecting to the proxy.</description>
  </property>

```

```
<property>
  <name>dfs.permissions</name>
  <value>true</value>
  <description>Enforce permissions</description>
</property>
<property>
  <name>dfs.permissions.supergroup</name>
  <value>hadoop</value>
  <description>The name of the group of Hadoop super-users.</description>
</property>
```

To configure the Map-Reduce component, edit `mapred-site.xml`:

```
cp etc/hadoop/mapred-site.xml.template etc/hadoop/mapred-site.xml
vi etc/hadoop/mapred-site.xml
```

Add the following values between the `<configuration>``</configuration>` tags. Note that many of these configurations items control the memory allocated during a map-reduce job.

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
<property>
  <name>mapreduce.map.memory.mb</name>
  <value>256</value>
</property>
<property>
  <name>mapreduce.reduce.memory.mb</name>
  <value>384</value>
</property>
<property>
  <name>mapreduce.map.java.opts</name>
  <value>-Xmx192m</value>
</property>
<property>
  <name>mapreduce.reduce.java.opts</name>
  <value>-Xmx256m</value>
```

```

</property>
<property>
  <name>mapreduce.job.tracker</name>
  <value>master:5431</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.resource.mb</name>
  <value>512</value>
</property>

```

To configure the resource manager YARN, edit yarn-site.xml:

```
vi etc/hadoop/yarn-site.xml
```

Add the following values between the <configuration></configuration> tags. Note that many of these configurations items control the memory allocated during a map-reduce job.

```

<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.resource.memory-mb</name>
  <value>512</value>
</property>
<property>
  <name>yarn.scheduler.minimum-allocation-mb</name>
  <value>128</value>
</property>
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>>false</value>
  <description>Whether virtual memory limits will be enforced for containers</description>
</property>
<property>
  <name>yarn.nodemanager.vmem-pmem-ratio</name>
  <value>4</value>
  <description>

```

Ratio between virtual memory to physical memory when setting memory limits for containers

```
</description>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>master:8025</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>master:8035</value>
</property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>master:8050</value>
</property>
```

Create and edit the masters and slaves files:

vi etc/hadoop/masters

Add line

master0

vi etc/hadoop/slaves

Add lines

master

slave1

slave2

slave3

slave4

Deploy software and settings on all slaves

root@xu4_1:~# parallel-ssh -i -h hadoop_slaves.txt -l root -A "mkdir -p /opt/hadoop-2.7.2/"

Warning: do not enter your password if anyone else has superuser

privileges or access to your account.

Password:

[1] 12:53:06 [SUCCESS] 10.50.3.12

[2] 12:53:06 [SUCCESS] 10.50.3.13

[3] 12:53:06 [SUCCESS] 10.50.3.14

[4] 12:53:06 [SUCCESS] 10.50.3.15

```
rsync -avxP /opt/hadoop-2.7.2/ root@slave1:/opt/hadoop-2.7.2/
```

```
rsync -avxP /opt/hadoop-2.7.2/ root@slave2:/opt/hadoop-2.7.2/
```

```
rsync -avxP /opt/hadoop-2.7.2/ root@slave3:/opt/hadoop-2.7.2/
```

```
rsync -avxP /opt/hadoop-2.7.2/ root@slave4:/opt/hadoop-2.7.2/
```

```
root@xu4_1:~# parallel-ssh -i -h hadoop_slaves.txt -l root -A "chown -R hduser:hadoop /opt/hadoop-2.7.2/"
```

Warning: do not enter your password if anyone else has superuser

privileges or access to your account.

Password:

[1] 13:02:46 [SUCCESS] 10.50.3.14

[2] 13:02:46 [SUCCESS] 10.50.3.12

[3] 13:02:46 [SUCCESS] 10.50.3.13

[4] 13:02:46 [SUCCESS] 10.50.3.15

On the master0 node the .bashrc file for “hduser” need to be adapted with Hadoop path:

```
su hduser
```

```
vi ~/.bashrc
```

Add this line at the end:

```
export PATH=$PATH:/usr/local/hadoop/sbin:/usr/local/hadoop/bin
```

7.5 ACTIVATION OF HADOOP CLUSTER

The settings and software deployment should be done now. Logging on the master0 node as “hduser” the hdfs will be formatted:

```
hdfs namenode -format
```

To start HDFS the command is

```
hduser@xu4_1:/root$ /usr/local/hadoop/sbin/start-dfs.sh
```

Starting namenodes on [master]

master: starting namenode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-namenode-xu4_1.out

master0: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_1.out

slave1: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_2.out

slave3: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_4.out

slave2: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_3.out

slave4: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_5.out

hdfs dfs -mkdir -p /user/wisegrid

To stop HDFS

hduser@xu4_1:/root\$ stop-dfs.sh

Stopping namenodes on [master]

master: stopping namenode

master0: stopping datanode

slave1: stopping datanode

slave2: stopping datanode

slave4: stopping datanode

slave3: stopping datanode

7.6 INSTALLATION OF APACHE SPARK SOFTWARE

The Apache SPARK application is a modern tool used for data analytics.

7.6.1 Prerequisites to SPARK installation

To be ready for Apache SPARK installation some folders should be created on all nodes of the cluster

root@xu4_1:~# parallel-ssh -i -h hadoop_cluster.txt -l root -A "mkdir -p /data/spark"

Warning: do not enter your password if anyone else has superuser privileges or access to your account.

Password:

[1] 13:25:29 [SUCCESS] 10.50.3.11

[2] 13:25:29 [SUCCESS] 10.50.3.12

[3] 13:25:29 [SUCCESS] 10.50.3.13

[4] 13:25:29 [SUCCESS] 10.50.3.14

[5] 13:25:29 [SUCCESS] 10.50.3.15

```
root@xu4_1:~# parallel-ssh -i -h hadoop_cluster.txt -l root -A "chown hduser:hadoop /data/spark"
```

Warning: do not enter your password if anyone else has superuser privileges or access to your account.

Password:

```
[1] 13:26:49 [SUCCESS] 10.50.3.11
```

```
[2] 13:26:49 [SUCCESS] 10.50.3.12
```

```
[3] 13:26:49 [SUCCESS] 10.50.3.13
```

```
[4] 13:26:49 [SUCCESS] 10.50.3.14
```

```
[5] 13:26:49 [SUCCESS] 10.50.3.15
```

On all nodes python3 should be installed. If is not installed by default, run this command as root on each node:

```
apt-get install python3
```

7.6.2 Installation of Apache SPARK software

The Apache Spark software can be download from a DIY Big Data repository. The download will be done on master0 node.

```
cd /opt
```

```
wget http://diybigdata.net/downloads/spark/spark-2.1.0-bin-hadoop2.7-double-alignment.tgz
```

```
tar xvzf spark-2.1.0-bin-hadoop2.7-double-alignment.tgz
```

```
chown -R hduser:hadoop spark-2.1.0-bin-v2.1.0-double-alignment
```

```
ln -s /opt/spark-2.1.0-bin-v2.1.0-double-alignment /usr/local/spark
```

Settings for SPARK software to be done on master0 node:

```
cd /usr/local/spark/conf
```

```
cp spark-env.sh.template spark-env.sh
```

```
vi spark-env.sh
```

Add following lines in the file

```
PYSPARK_PYTHON=python3
```

```
PYTHONHASHSEED=12121969
```

```
SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

```
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

```
SPARK_LOCAL_DIRS=/data/spark
```

```
SPARK_WORKER_MEMORY=1300M
```

```
SPARK_WORKER_CORES=3
```

```
SPARK_DAEMON_JAVA_OPTS="-Dspark.worker.timeout=600          -Dspark.akka.timeout=200  -  
Dspark.shuffle consolidateFiles=true"
```

```
SPARK_JAVA_OPTS="-Dspark.worker.timeout=600          -Dspark.akka.timeout=200  -  
Dspark.shuffle consolidateFiles=true"
```

Next, we need to set up the defaults configuration values that Spark uses.

```
cp spark-defaults.conf.template spark-defaults.conf  
vi spark-defaults.conf
```

Add the following lines to the file:

```
spark.master          spark://master:7077  
spark.serializer      org.apache.spark.serializer.KryoSerializer  
spark.executor.memory 1000M  
spark.driver.memory   1000M  
spark.io.compression.codec lz4  
spark.driver.cores    2  
spark.executor.cores  2
```

Finally, is needed to tell spark where the slave nodes are.

```
cp slaves.template slaves  
vi slaves
```

Then add the name of each slave to the end of the file:

```
Slave1  
Slave2  
Slave3  
Slave4
```

7.6.3 Deployment of SPARK software and settings on all slaves

For deploying the software and settings parallel-ssh and rsync will be used

```
root@xu4_1:~# parallel-ssh -i -h hadoop_slaves.txt -l root -A "mkdir -p /opt/spark-2.1.0-bin-v2.1.0-double-  
alignment"
```

*Warning: do not enter your password if anyone else has superuser
privileges or access to your account.*

Password:

```
[1] 13:40:22 [SUCCESS] 10.50.3.12
```

```
[2] 13:40:22 [SUCCESS] 10.50.3.13
```

```
[3] 13:40:22 [SUCCESS] 10.50.3.14
```

```
[4] 13:40:22 [SUCCESS] 10.50.3.15
```

```
rsync -avxP /opt/spark-2.1.0-bin-v2.1.0-double-alignment root@slave1:/opt/
```

```
rsync -avxP /opt/spark-2.1.0-bin-v2.1.0-double-alignment root@slave2:/opt/
```

```
rsync -avxP /opt/spark-2.1.0-bin-v2.1.0-double-alignment root@slave3:/opt/
```

```
rsync -avxP /opt/spark-2.1.0-bin-v2.1.0-double-alignment root@slave4:/opt/
```

```
root@xu4_1:~# parallel-ssh -i -h hadoop_slaves.txt -l root -A "chown -R hduser:hadoop /opt/spark-2.1.0-  
bin-v2.1.0-double-alignment"
```

Warning: do not enter your password if anyone else has superuser

privileges or access to your account.

Password:

```
[1] 13:46:01 [SUCCESS] 10.50.3.12
```

```
[2] 13:46:01 [SUCCESS] 10.50.3.13
```

```
[3] 13:46:01 [SUCCESS] 10.50.3.14
```

```
[4] 13:46:01 [SUCCESS] 10.50.3.15
```

```
root@xu4_1:~# parallel-ssh -i -h hadoop_slaves.txt -l root -A "ln -s /opt/spark-2.1.0-bin-v2.1.0-double-  
alignment /usr/local/spark"
```

Warning: do not enter your password if anyone else has superuser

privileges or access to your account.

Password:

```
[1] 13:48:02 [SUCCESS] 10.50.3.12
```

```
[2] 13:48:02 [SUCCESS] 10.50.3.13
```

```
[3] 13:48:02 [SUCCESS] 10.50.3.14
```

```
[4] 13:48:02 [SUCCESS] 10.50.3.15
```

7.6.4 Activation of SPARK software

For activating the spark software login on mastero on “hduser” user and use the command:

```
hduser@xu4_1:/root$ /usr/local/hadoop/sbin/start-dfs.sh
```

```
Starting namenodes on [master]
```

```
master: starting namenode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-namenode-xu4_1.out
```

```
master0: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_1.out
```

slave4: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_5.out

slave3: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_4.out

slave2: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_3.out

slave1: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-hduser-datanode-xu4_2.out

Starting secondary namenodes [0.0.0.0]

0.0.0.0: secondarynamenode running as process 8140. Stop it first.

hduser@xu4_1:/root\$ /usr/local/spark/sbin/start-all.sh

starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-hduser-
org.apache.spark.deploy.master.Master-1-xu4_1.out

slave4: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-
org.apache.spark.deploy.worker.Worker-1-xu4_5.out

slave3: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-
org.apache.spark.deploy.worker.Worker-1-xu4_4.out

slave2: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-
org.apache.spark.deploy.worker.Worker-1-xu4_3.out

slave1: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-
org.apache.spark.deploy.worker.Worker-1-xu4_2.out

localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-
org.apache.spark.deploy.worker.Worker-1-xu4_1.out

8 APACHE HADOOP CONNECTION WITH THE MONGODB CONECTOR

The connection between the MongoDB database and Hadoop + Spark framework is provided by a MongoDB Connector for Hadoop. The installation of the connector is based by the MongoDB manual pages dedicated to Apache Hadoop connections [23] and Wiki pages on the same subject [24].

The MongoDB also defines some very interesting use cases about the integration between MongoDB, Hadoop and user application. The use cases are:

- Batch aggregation

In several scenarios the built-in aggregation functionality provided by MongoDB is sufficient for analysing your data. However, in certain cases, significantly more complex data aggregation may be necessary. This is where Hadoop can provide a powerful framework for complex analytics [25].

In this scenario data is pulled from MongoDB and processed within Hadoop via one or more MapReduce jobs. Data may also be brought in from additional sources within these MapReduce jobs to develop a multi-data source solution. Output from these MapReduce jobs can then be written back to MongoDB for later querying and ad-hoc analysis. Applications built on top of MongoDB can now use the information from the batch analytics to present to the end user or to drive other downstream features [25].



Figure 27 – Hadoop MongoDB batch aggregation [25]

- Data Warehouse

In a typical production scenario, your application's data may live in multiple datastores, each with their own query language and functionality. To reduce complexity in these scenarios, Hadoop can be used as a data warehouse and act as a centralized repository for data from the various sources.

In this situation, you could have periodic MapReduce jobs that load data from MongoDB into Hadoop. This could be in the form of "daily" or "weekly" data loads pulled from MongoDB via MapReduce. Once the data from MongoDB is available from within Hadoop, and data from other sources are also available, the larger dataset data can be queried against. Data analysts now have the option of using either MapReduce or Pig to create jobs that query the larger datasets that incorporate data from MongoDB [25].

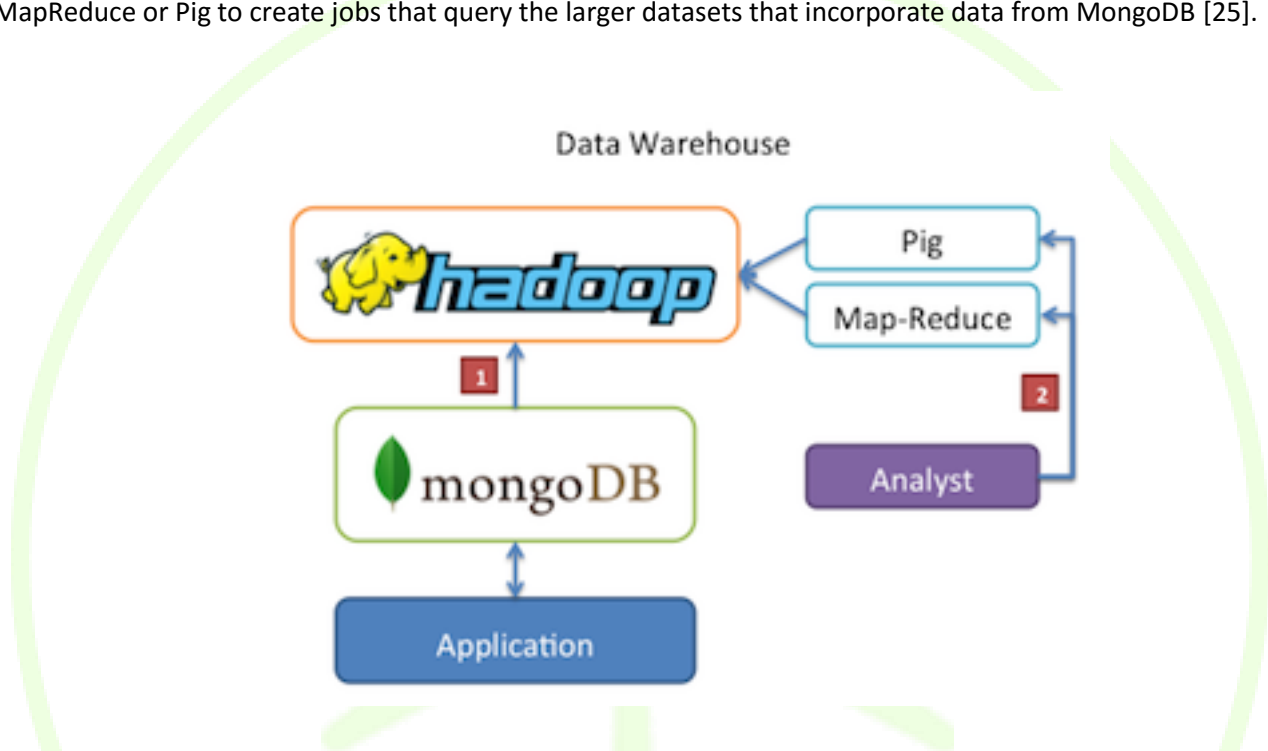


Figure 28 – MongoDB Hadoop Data Warehouse [25]

- ETL Data

MongoDB may be the operational datastore for your application but there may also be other datastores that are holding your organization's data. In this scenario it is useful to be able to move data from one datastore to another, either from your application's data to another database or vice versa. Moving the data is much more complex than simply piping it from one mechanism to another, which is where Hadoop can be used.

In this scenario, Map-Reduce jobs are used to extract, transform and load data from one store to another. Hadoop can act as a complex ETL mechanism to migrate data in various forms via one or more MapReduce jobs that pull the data from one store, apply multiple transformations (applying new data layouts or other aggregation) and loading the data to another store. This approach can be used to move data from or to MongoDB, depending on the desired result [25].

ETL from MongoDB

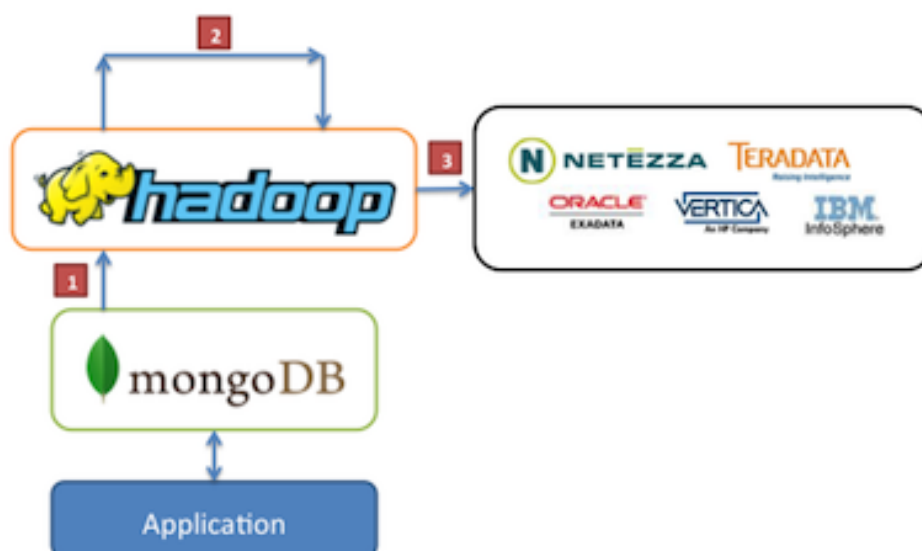


Figure 29 – Hadoop ETL from MongoDB [25]

ETL to MongoDB

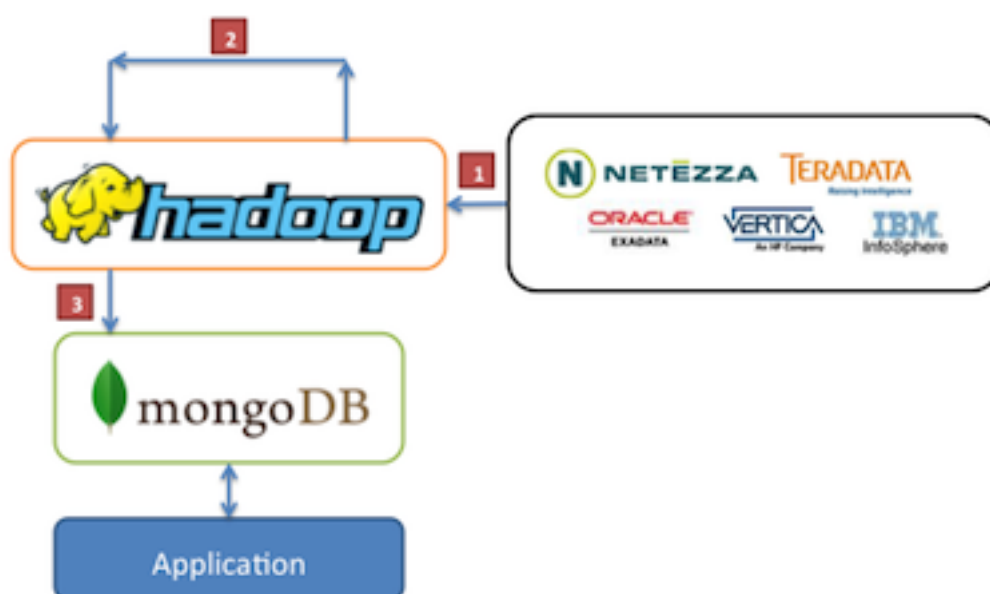


Figure 30 – Hadoop ETL to MongoDB [25]

8.1 INSTALLATION OF HADOOP MONGODB CONNECTOR

MongoDB connector for Hadoop is a java written plugin for Hadoop that provides the ability for Hadoop to use MongoDB as an input source and also an output destination.

The steps for installing:

- Obtain the Hadoop connector. The JARs can be downloaded from from the Maven Central Repository <http://repo1.maven.org/maven2/org/mongodb/mongo-hadoop/>. The JARs are universal and will work with any version of Hadoop.
- Obtain a JAR for the MongoDB Java Driver http://mongodb.github.io/mongo-java-driver/?jmp=docs&_ga=2.76560725.1338393439.1531580407-1387601032.1514913385&_gac=1.220448044.1531315453.EAlalQobChMIstzwx5KX3AIVSKt3Ch08sQyYEAHEYASAAEgKhofD_BwE.
- Move these JARs onto each node of the Hadoop cluster. On each node the jars must be placed somewhere on Hadoop's CLASSPATH (e.g. \$HADOOP_PREFIX/share/hadoop/common), or you can use the Hadoop Distributed Cache to move the JARs onto pre-existing nodes.

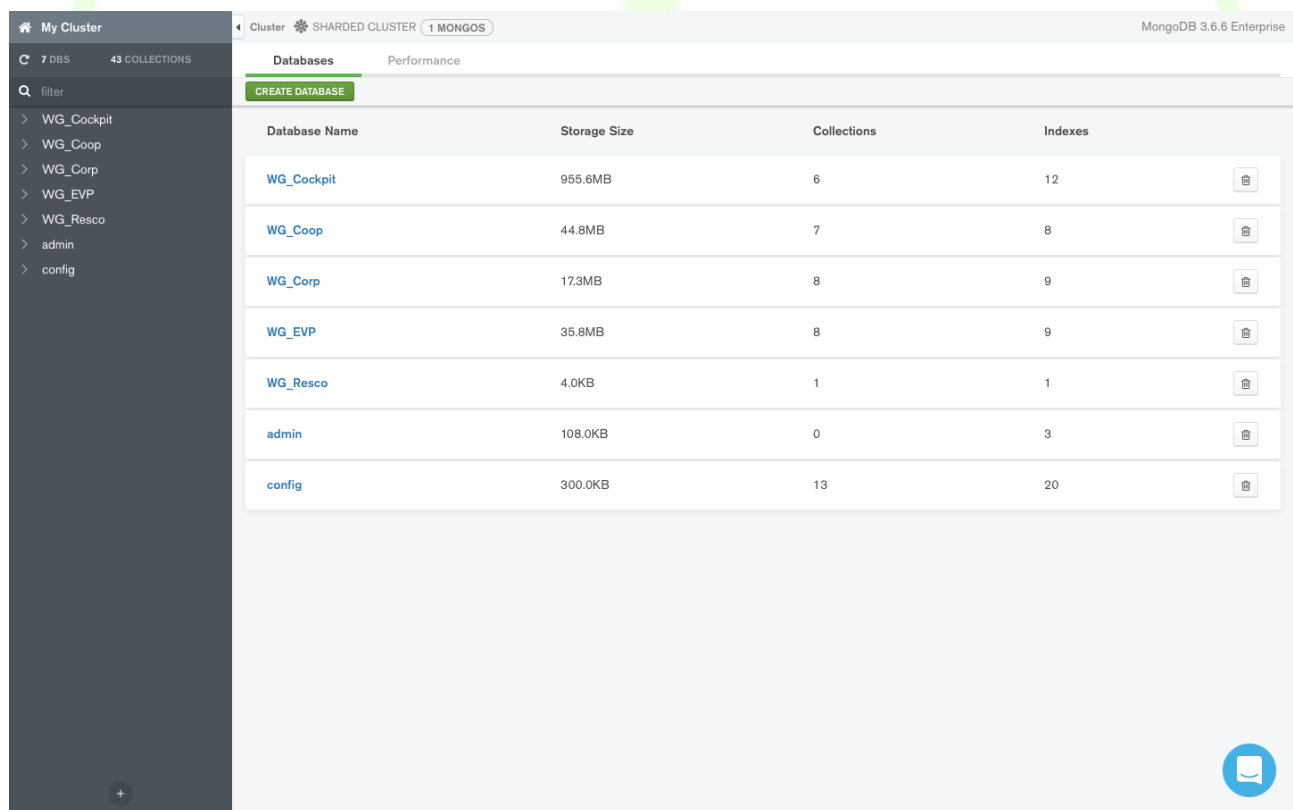
9 CONNECTION OF APPLICATIONS TO MONGODB CLUSTER

9.1 APPLICATION INTERACTIONS FROM WISEGRID APPLICATIONS TO THE BIG DATA PLATFORM

In this chapter the interactions between different applications to the Big Data platform are described for each applications. Some parts in this chapter are also available in D5.1 [1] and are here also for an easy understanding of this chapters.

9.1.1 Databases created in lab testing phase

In lab testing phase for the MongoDB cluster the WiseGRID application created databases and stored data. The databases created can be seen in the following picture:



The screenshot shows the MongoDB Enterprise interface. On the left, a sidebar lists databases: WG_Cockpit, WG_Coop, WG_Corp, WG_EVP, WG_Resco, admin, and config. The main area displays a table of databases with columns: Database Name, Storage Size, Collections, and Indexes. A 'CREATE DATABASE' button is visible at the top left of the table area.

Database Name	Storage Size	Collections	Indexes
WG_Cockpit	955.6MB	6	12
WG_Coop	44.8MB	7	8
WG_Corp	17.3MB	8	9
WG_EVP	35.8MB	8	9
WG_Resco	4.0KB	1	1
admin	108.0KB	0	3
config	300.0KB	13	20

Figure 31 – Databases created in lab testing phase

9.1.2 Big data interactions from WG IOP application

9.1.2.1 Short application description

The WiseGRID Interoperable Platform (WG IOP) is a scalable, secure and open ICT platform, with interoperable interfaces, for real-time monitoring and decentralized control to support effective operation of the energy network. The objective of the platform is to manage and process the heterogeneous and massive data streams coming from the distributed energy infrastructure deployed. This platform enable new services and reduce ICT costs for prosumers and smaller players, whilst it will facilitate cross-network and cross-entity interoperability. In order to increase adoption and speed up deployment, this platform have open interfaces to the relevant energy, IoT and Smart City standards. WG IOP enable the cooperation and synergies among the different actors targeted by the different WiseGRID technological solutions. The core of the architecture is represented by a Message Broker (RabbitMQ message broker has been adopted) by which all message are flowing, the broker is agnostic about the content of the message and its structure, by several micro-services that mainly facilitate the integration of different kind of assets with the the WiseGrid tools and services and by a security module to ensure that all the flow exchanged via the WG IOP are secured and all the operation authorized.

9.1.2.2 Interface with Big Data platform

As detailed in the D4.2 “WiseGRID interoperable Platform Process (WG IOP)”, the WG IOP does not directly access the big data platform, it like a bridge permit the exchange of data that are retrieved or stored by external services (DB interaction services) and WiseGRID tools like the WG RESCO or WG Cockpit and so on from an to the Big Data Platform DB instances. The next image show the WG IOP architecture in which it is possible to identify the interaction with the Big Data platform directly by each tools or external services.

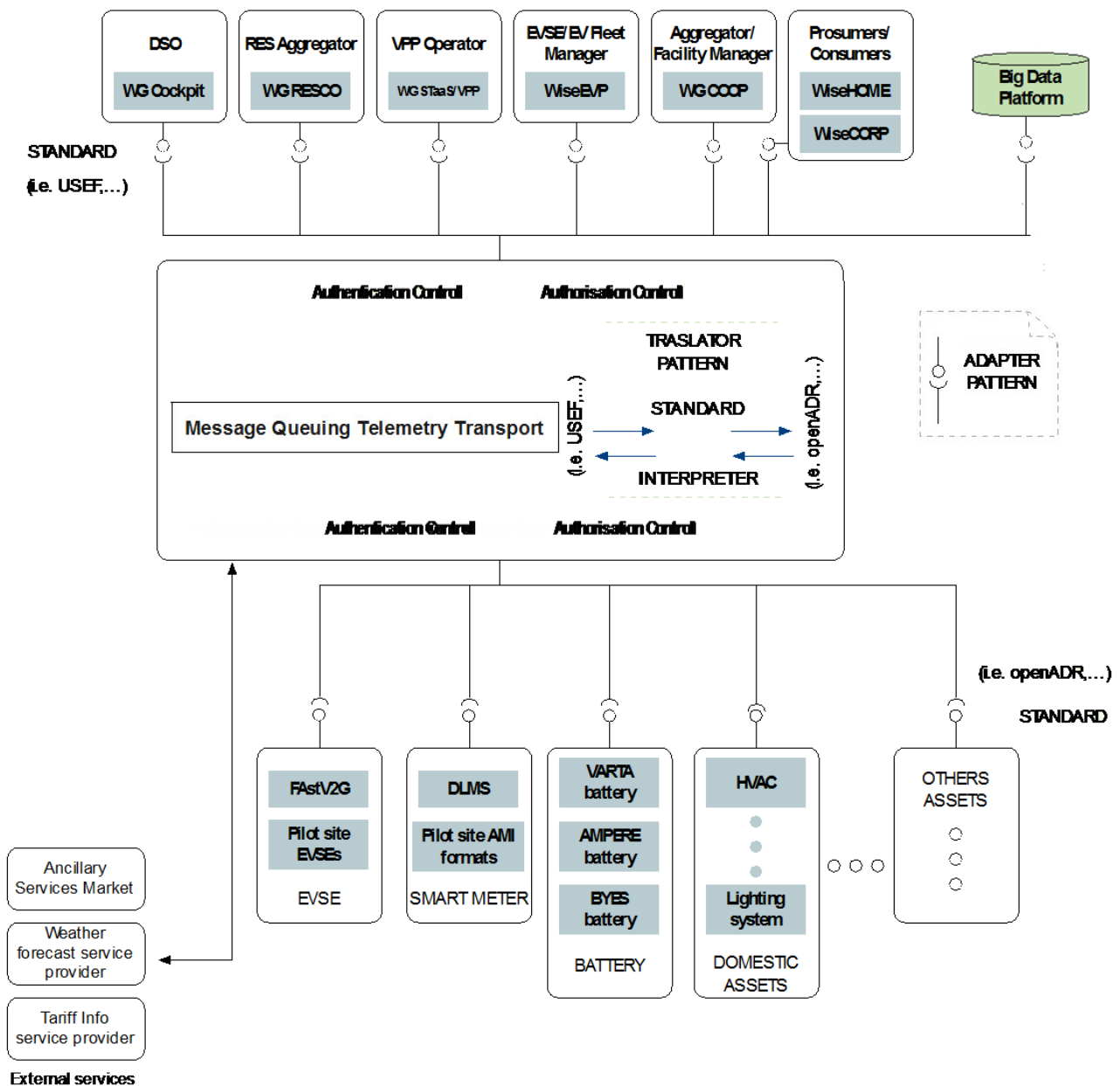


Figure 32 – WG IOP Architecture

9.1.3 Big data interactions from WG Cockpit application

9.1.3.1 Short application description

WiseGRID Cockpit is the WiseGRID technological solution targeting DSOs and microgrid operators, allowing them to control, manage and monitor their own grid, improving flexibility, stability and security of their net-work. Taking into account the goals of the project, the features to be implemented within WiseGRID cockpit consider a scenario of increasing share of distributed renewable resources and services provided by communities of prosumers (aggregated in the form of VPPs or cooperatives in order to achieve higher participation and environmental, social and economic benefits).

The main purpose of the WiseGRID Cockpit is to enable DSOs to manage the fundamental changes that

distribution grids are facing nowadays, some remarkable ones of those being the transition towards a grid with high penetration of distributed renewable energy resources and the presence of additional significant loads coming from electric vehicles among others. In addition, this particular outcome of the WiseGRID project aims at approaching the benefits that new technologies (such as big data or unbundled smart meters) and algorithms (such as state estimation or fault detection) bring to the operation of the grid. Finally, since one of the objectives of the project is the empowerment of the citizens in the energy field, the WiseGRID Cockpit will also demonstrate how that empowerment can be beneficial for several actors - including DSOs -, and how the whole ecosystem of actors can contribute to reach an environmentally and economically sustainable energy system.



Figure 33 - WiseGRID Cockpit

9.1.3.2 Interface with Big Data platform

The big data platform will be used by the WiseGRID Cockpit in order to hold the history of data provided by the sensors of interest of the application. As depicted in the architecture of the WG Cockpit, the main data sources for this information are:

- Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds)
- Advanced Metering Infrastructure systems, retrieving data from already deployed Smart Meters – usually hourly curves retrieved once a day
- SCADA systems, retrieving real-time electric measurements from monitored infrastructure under control of the DSO (buses of the distribution grid) and status of safety elements

This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker into the WiseGRID Cockpit application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

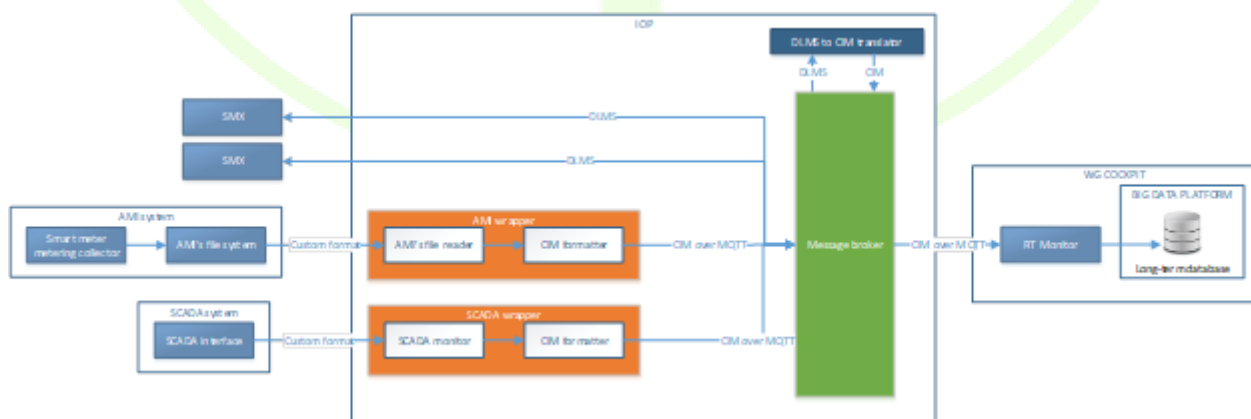


Figure 34 – WG Cockpit interface with Big Data platform

The RT monitor module is, therefore, the module which, inside the WG Cockpit application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseGRID Cockpit will use the Data mining and analytics module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, which will process the information to obtain:

- Hourly/Daily/Monthly aggregated data to produce summaries of interest to the DSO operator (energy imported from HV grid, energy produced)
- Statistical data (average, normal deviation) on electric parameters affecting quality of the service (frequency, voltage)

These jobs appear in the architecture of the WiseGRID Cockpit as KPI engine module. Results of this calculation will be stored back to different collections of the long-term database but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data) and published to the internal Enterprise Service Bus if required by other modules.

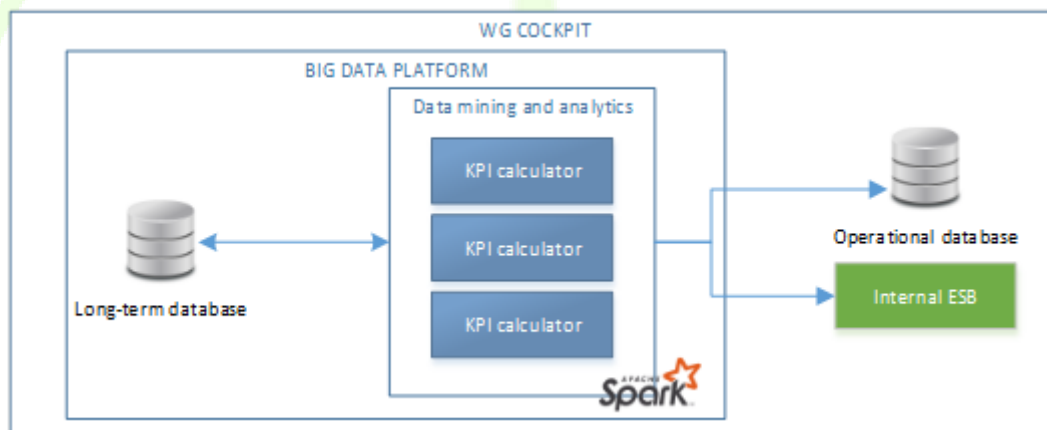


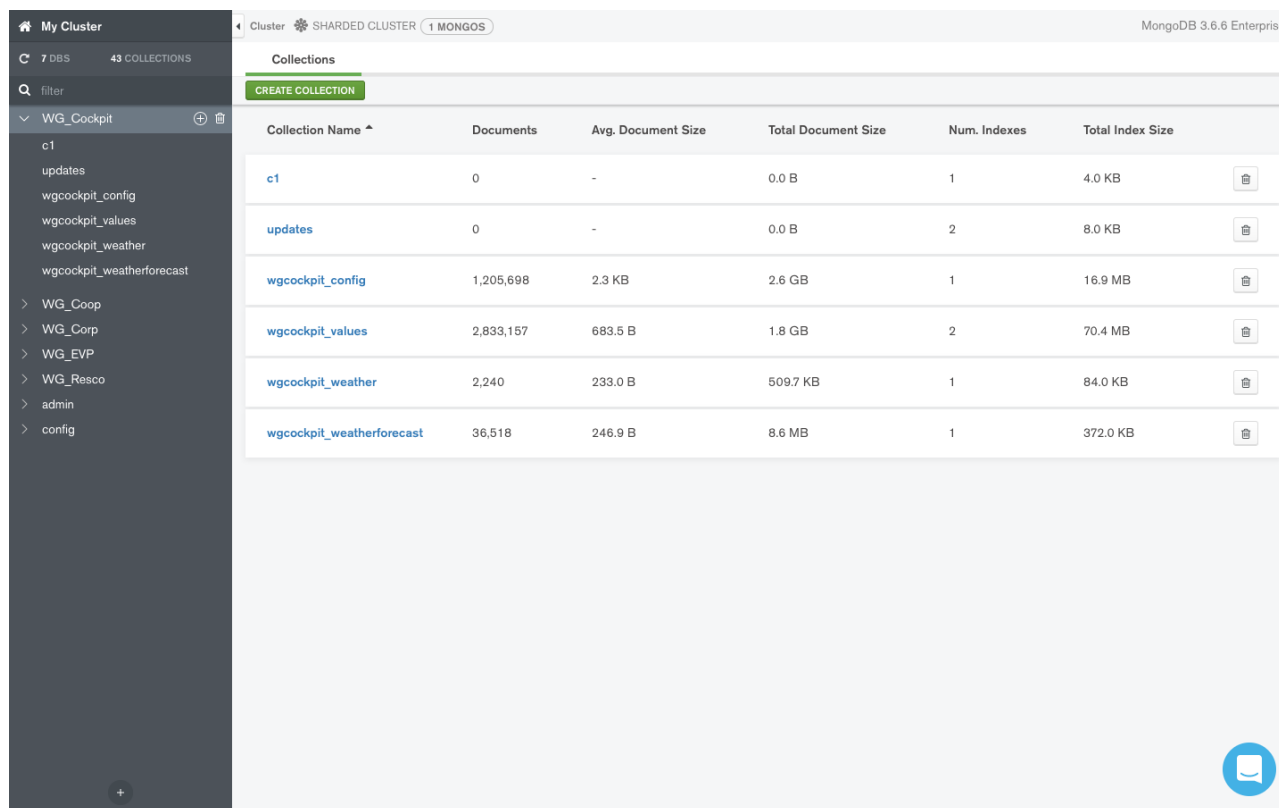
Figure 35 – WG Cockpit data mining

As part of the implementation and lab-testing task, the real-time monitor module of the WiseGRID Cockpit application has been configured to store all data produced within the lab environment in the MongoDB cluster developed in this work package. This implied the creation of one database hosting the collections specified below.

Table 8 –Collections in WG Cockpit database

Collection	Description
wgcockpit_config	Registry of changes in the configuration of the SMXs (smart meters)(Registration upon changes)
wgcockpit_values	History of readouts provided by the smart meters (1 readout every 10 seconds)
wgcockpit_weather	Actual weather history at the location of the corresponding pilot site (1 register every hour)
wgcockpit_weatherforecast	History of weather forecasts at the location of the corresponding pilot site (12 registers every hour)

Details about the database created by the WiseGRID Cockpit in the lab testing phase can be seen in the following picture.



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
c1	0	-	0.0 B	1	4.0 KB
updates	0	-	0.0 B	2	8.0 KB
wgcockpit_config	1,205,698	2.3 KB	2.6 GB	1	16.9 MB
wgcockpit_values	2,833,157	683.5 B	1.8 GB	2	70.4 MB
wgcockpit_weather	2,240	233.0 B	509.7 KB	1	84.0 KB
wgcockpit_weatherforecast	36,518	246.9 B	8.6 MB	1	372.0 KB

Figure 36 – Collections inside the WG Cockpit database

Documents from wgcockpit_config collection stored during lab testing phase are in the following picture:



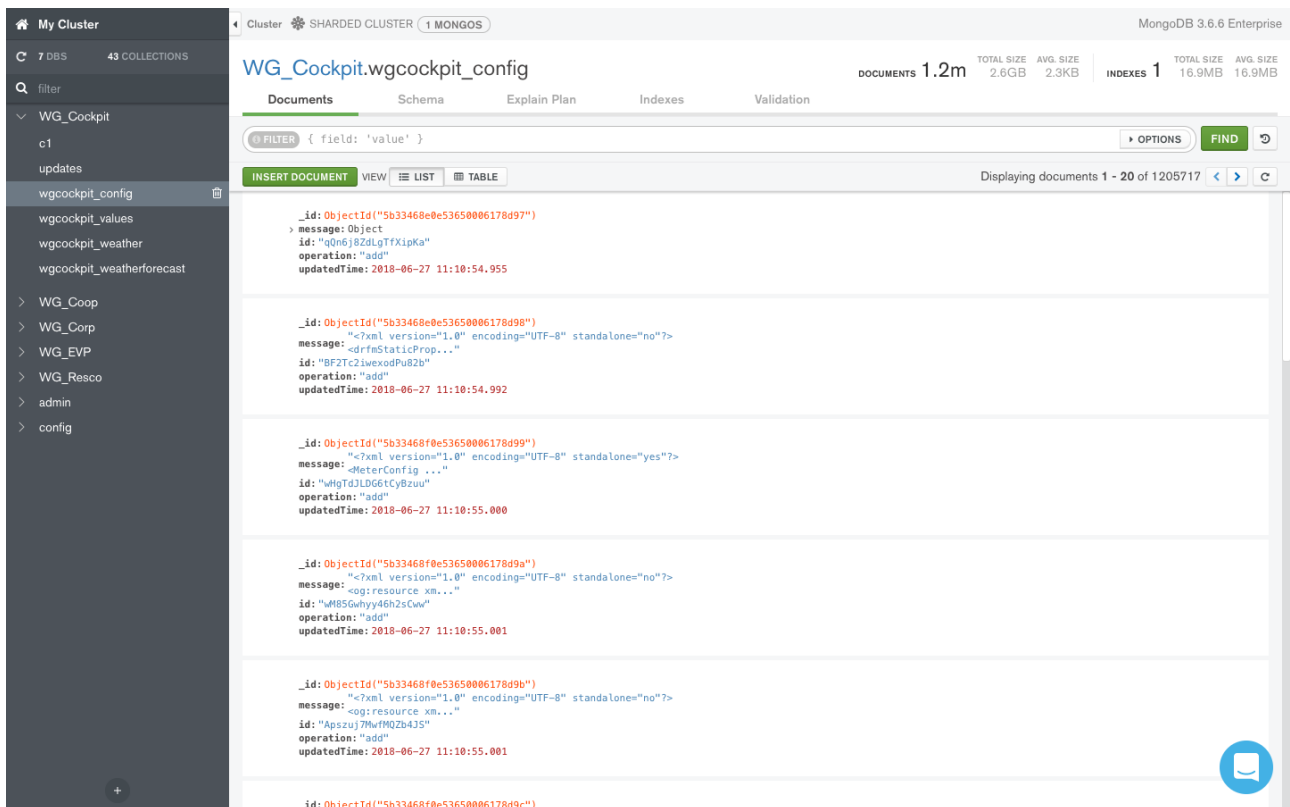


Figure 37 – Documents stored in the `wgcockpit_config` collection

In order to take advantage of the capabilities of the clustered database, the collections need to be *sharded*, so data is actually distributed among the different nodes of the cluster. The advantages of this configuration are described in detail in D5.1 [1]. The most important point when applying this configuration is defining the proper *shard* key for the collection, which is used to automatically create partitions among data that are stored at different nodes. As explained in MongoDB official documentation, *the shard key determines the distribution of the collection's documents among the cluster's shards. The shard key is either an indexed field or indexed compound fields that exists in every document in the collection* [26].

In the case of WiseGRID Cockpit, tests have been carried out with the `wgcockpit_values` collection, the one holding one reading every 10 seconds per devices. The selected shard key was `{mRID:1, timestamp:1}`. By including the ID of the devices, it is assured that measures of different devices are candidates to be separated in different clusters. Since the collection contains historical readout data, timestamp was also chosen to be included in the shard key in order to promote the separation of the data accordingly to the periods in time they refer to. Results of this configuration are shown below:

```
db.getCollection('wgcockpit_values').getShardDistribution()

Shard rs0 at rs0/sh0r0:27017,sh0r1:27017
  data : 601.8MiB docs : 795251 chunks : 20
  estimated data per chunk : 30.08MiB
  estimated docs per chunk : 39762

Shard rs1 at rs1/sh1r0:27017,sh1r1:27017
  data : 567.13MiB docs : 550909 chunks : 22
  estimated data per chunk : 25.77MiB
```

```
estimated docs per chunk : 25041
```

```
Shard rs2 at rs2/sh2r0:27017,sh2r1:27017
```

```
data : 681.75MiB docs : 1495388 chunks : 22
```

```
estimated data per chunk : 30.98MiB
```

```
estimated docs per chunk : 67972
```

Totals

```
data : 1.8GiB docs : 2841548 chunks : 64
```

```
Shard rs0 contains 32.51% data, 27.98% docs in cluster, avg obj size on  
shard : 793B
```

```
Shard rs1 contains 30.64% data, 19.38% docs in cluster, avg obj size on  
shard : 1KiB
```

```
Shard rs2 contains 36.83% data, 52.62% docs in cluster, avg obj size on  
shard : 478B
```

It can be observed that the *shard key* works as expected, distributing 1.8GB of data approximately equally among the three nodes. In addition, the logs show how the shards are replicated (*sh0r0/sh0r1*, *sh1r0/sh1r1*, *sh2r0/sh2r1*), thus increasing the availability of the data in the cluster in case of failure.

9.1.4 Big data interactions from WiseCORP application

9.1.4.1 Short application description

WiseCORP is the WiseGRID technological solution targeting businesses, industries, ESCOs and public facility consumers and prosumers, with the objective of providing them the necessary mechanisms to become smarter energy players. By means of energy usage monitoring and analysis, proper information can be given to facility managers helping them to reduce energy costs and environmental impact.

A key factor towards achieving these objectives is a proper retrieval and analysis of energy usage data, and visualization of meaningful information extracted from it. This information may include:

- Detailed visualization of energy demand at different areas of the building, helping facility managers to identify opportunities for enhancing energy efficiency.
- Energy tariff comparison, enabling a direct economic cost reduction by shifting to a more adequate tariff.
- Energy demand forecast, enabling medium to long term cost estimations and supporting operative decisions about the usage of the facilities.
- Demand flexibility estimation, allowing the execution of optimization algorithms that will - either automatically or by providing advices - shift demand in order to minimize economic costs - by maximizing self-consumption or moving demand to off-peak periods - or minimize environmental impact - by shifting demand to periods where green energy is available.



Figure 38 – WiseCORP

9.1.4.2 Interface with Big Data platform

The big data platform will be used by the WiseCORP in order to hold the history of data provided by the different elements of interest of the application. As depicted in the architecture of the WiseCORP, the main data sources for this information are:

- Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds).
- Advanced Metering Infrastructure systems from the DSO, retrieving data from already deployed

Smart Meters – usually hourly curves retrieved once a day.

- Storage systems.
- CHP devices, publishing their operation setpoint.
- HVAC devices, publishing their operation setpoint.
- Sensors for measuring indoor conditions, mainly temperature.

This data will flow from the source devices/systems, through the WG IOP Message Broker into the WiseCORP application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

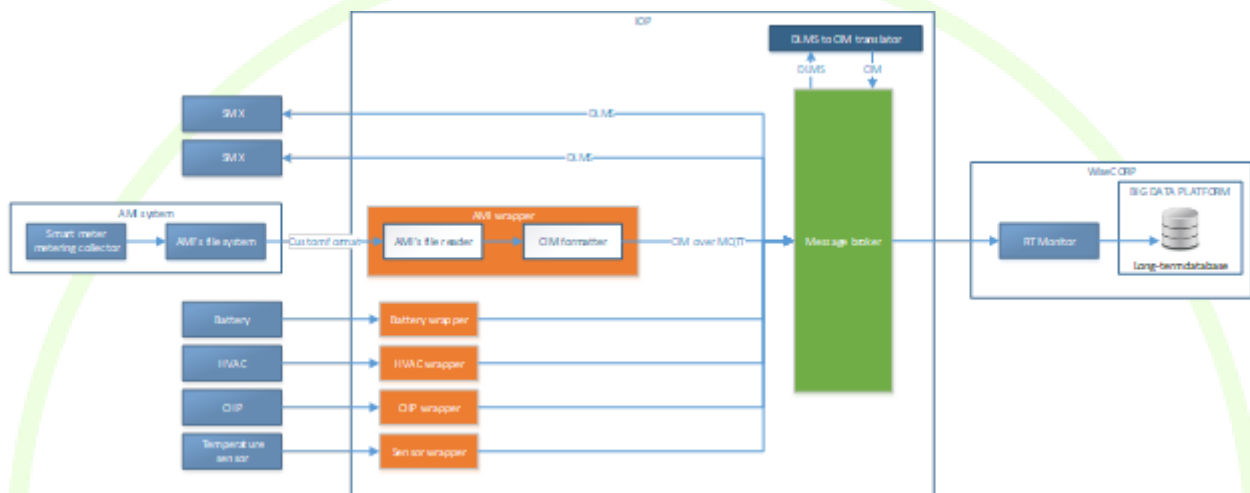


Figure 39 – WiseCORP interface with Big Data platform

The RT monitor module is, therefore, the module which, inside the WiseCORP application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseCORP will use the *Data mining and analytics* module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, which will process the information to obtain:

- Hourly/Daily/Monthly aggregated data to produce summaries of interest to the ESCO or facility manager (demand, production).
- Analysis of the demand (according to source, tariff period, self-consumption capabilities)
- Analysis of the used capacity (histograms of active power).

These jobs appear in the architecture of the WiseCORP as *KPI engine* module. Results of this calculation will be stored back to different collections of the long-term database but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data), or republished to the internal ESB with the objective of making certain KPIs accessible to other modules of the application.

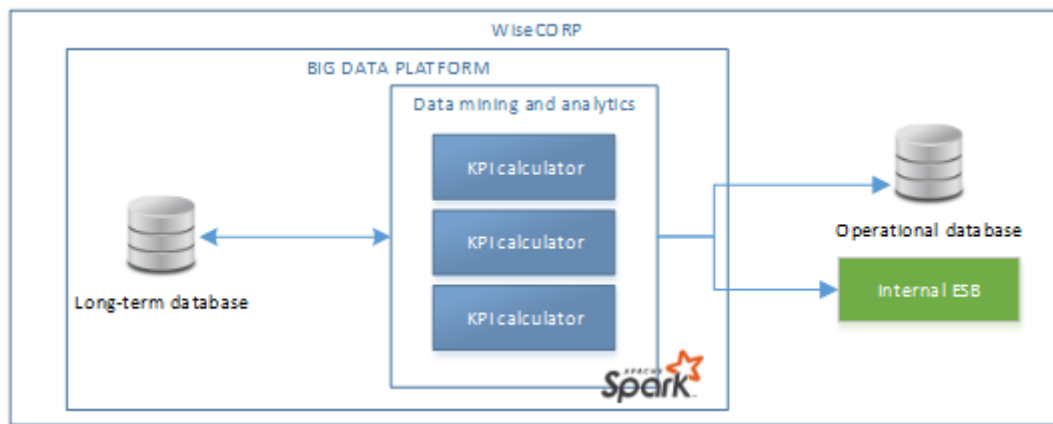


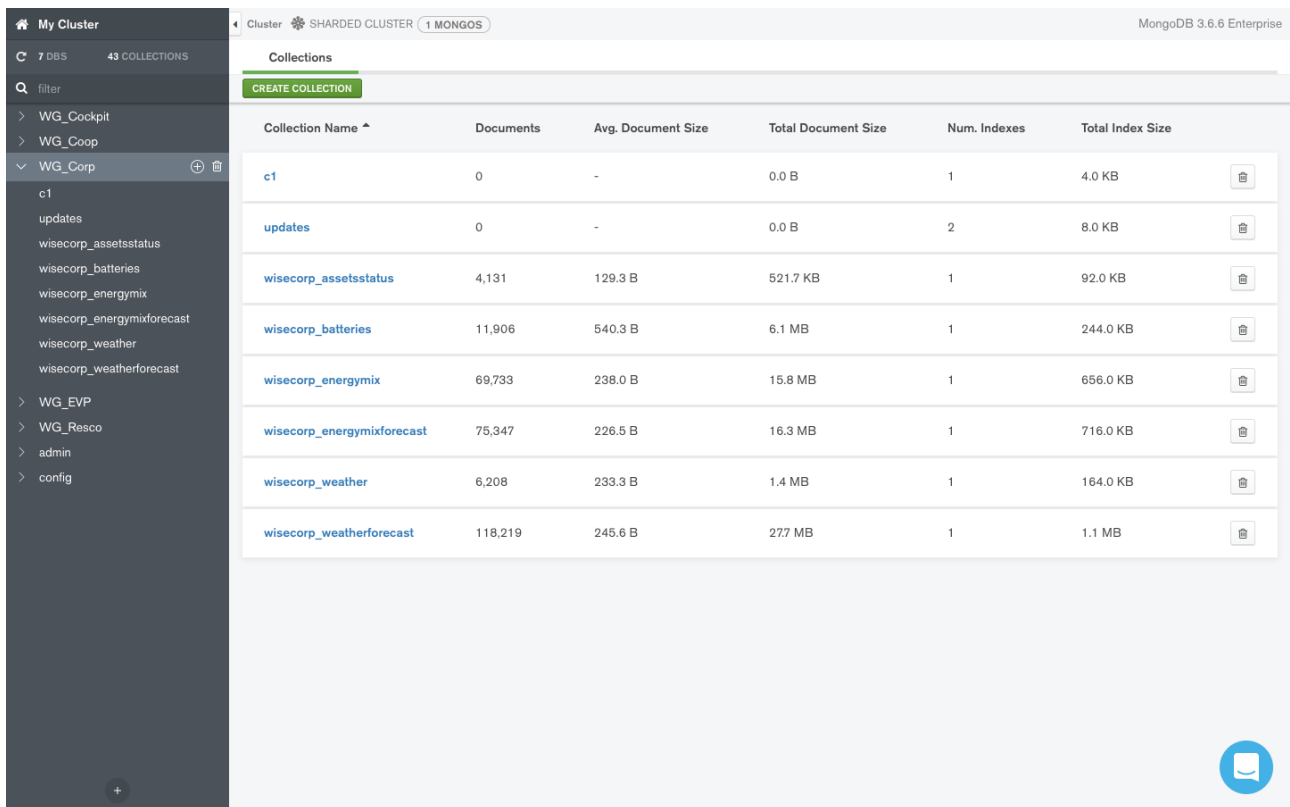
Figure 40 – WiseCORP data mining

As part of the implementation and lab-testing task, the real-time monitor module of the WiseCORP application has been configured to store all data produced within the lab environment in the MongoDB cluster developed in this work package. This implied the creation of one database hosting the collections specified below.

Table 9 – Wisecorp database collections

Collection	Description
Wisecorp_assetsstatus	History of status sent by the different sensors of the buildings (temperature, lighting, smart plug demand, HVAC setpoints)
Wisecorp_batteries	History of status/setpoints sent by the batteries of the buildings (SoC, setpoint, state of health...) (1 register every 10 seconds)
Wisecorp_energymix	Actual energy mix history at the location of the corresponding pilot site (1 register every hour)
Wisecorp_energymixforecast	History of energy mix forecasts at the location of the corresponding pilot site (46 registers every hour)
Wisecorp_weather	Actual weather history at the location of the corresponding pilot site (1 register every hour)
Wisecorp_weatherforecast	History of weather forecasts at the location of the corresponding pilot site (12 registers every hour)

Collections from the WG_CORP database used during the lab testing phase are in the following picture:



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
c1	0	-	0.0 B	1	4.0 KB
updates	0	-	0.0 B	2	8.0 KB
wisecorp_assetsstatus	4,131	129.3 B	521.7 KB	1	92.0 KB
wisecorp_batteries	11,906	540.3 B	6.1 MB	1	244.0 KB
wisecorp_energymix	69,733	238.0 B	15.8 MB	1	656.0 KB
wisecorp_energymixforecast	75,347	226.5 B	16.3 MB	1	716.0 KB
wisecorp_weather	6,208	233.3 B	1.4 MB	1	164.0 KB
wisecorp_weatherforecast	118,219	245.6 B	27.7 MB	1	1.1 MB

Figure 41 – Details of the WG_Corp database

In order to take advantage of the capabilities of the clustered database, the collections need to be *sharded*, so data is actually distributed among the different nodes of the cluster. The advantages of this configuration are described in detail in D5.1 [1]. The most important point when applying this configuration is defining the proper *shard key* for the collection, which is used to automatically create partitions among data that are stored at different nodes. As explained in MongoDB official documentation, *the shard key determines the distribution of the collection's documents among the cluster's shards. The shard key is either an indexed field or indexed compound fields that exists in every document in the collection* [26].

In the case of WiseGRID Cockpit, tests have been carried out with the *wisecorp_assetsstatus* and *wisecorp_batteries* collections, the ones holding the history of status and setpoints of the assets in the building. The selected shard key for both is similar *{id:1, [capture|updated]time:1}*. By including the ID of the devices, it is assured that measures of different devices are candidates to be separated in different clusters. Since the collection contains historical readout data, timestamps were also chosen to be included in the shard key in order to promote the separation of the data accordingly to the periods in time they refer to. Results of this configuration are shown below:

```
db.getCollection('wisecorp_batteries').getShardDistribution()

Shard rs1 at rs1/sh1r0:27017,sh1r1:27017
  data : 19.39MiB docs : 37574 chunks : 1
  estimated data per chunk : 19.39MiB
  estimated docs per chunk : 37574

Totals
  data : 19.39MiB docs : 37574 chunks : 1
  Shard rs1 contains 100% data, 100% docs in cluster, avg obj size on
```

shard : 541B

In this case, it can be observed that the volume of data in the lab-testing environment at the time of writing is not enough to really activate the redistribution of the data among the shards of the cluster. In this case, the logs still show how the shards are replicated (sh1r0/sh1r1), thus increasing the availability of the data in the cluster in case of failure.

9.1.5 Big data interactions from WiseCOOP application

9.1.5.1 Short application description

WiseCOOP is the WiseGRID technological solution targeting aggregators of consumers and prosumers - particularly focused on domestic and small businesses -, supporting them in their roles of energy retailers, local communities and cooperatives - which may have different objectives.

The main goal of the solution is helping consumers and prosumers to work together in order to achieve better energy deals while relieving them from administrative procedures and cumbersome research. In the particular scenario of increasing share of distributed renewable resources, this goal can be achieved by pursuing several objectives:

- Net-metering: supporting the operation of communities of prosumers that invest in renewable energy sources aiming at reducing their environmental impact
- Member profiling: clusters of consumers and prosumers with common energy usage patterns may be identified, allowing the aggregator to negotiate special terms (as for instance energy tariffs) particularly beneficial for those groups
- Demand forecasting: by allowing aggregator (in its retailer role) to forecast the demand of its customers, optimized purchase of energy at the wholesale market is enabled
- Tariff comparison: by offering members a tool for comparing their particular consumption with different available tariffs, those will have access to very valuable information to reduce their energy bills
- Implicit price-based demand response towards modulating the overall demand of the group to achieve a common objective (as, for instance, maximize usage of renewable energy sources produced within the group or minimize deviations between actual demand and energy purchased at the wholesale market)
- Providing clear information to members to raise awareness on efficient energy usage and environmental awareness



The big data platform will be used by the WiseCOOP mainly with the objective of storing the history of energy demand and production of the members of the portfolio aggregated by the organization using the application. As depicted in the architecture of the WiseCOOP, the main data sources for this information are:

- This data will flow from the source devices/systems, through the WG IOP Message Broker into the WiseCOOP application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.



On the other hand, the WiseCOOP will use the *Data mining and analytics* module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, and will take advantage of the machine learning capabilities of this framework, in order to process the following required information:

- Aggregated data of interest to the manager of the aggregation of prosumers and to individual prosumers
 - Cumulative daily consumption in real-time
 - Cumulative weekly consumption in real-time
 - Cumulative monthly consumption in real-time
 - Cumulative yearly consumption in real-time
 - Cumulative energy consumption for the same calendar day of previous year
 - Cumulative weekly energy consumption for the same week of previous year until corresponding day
 - Cumulative energy consumption for the same month of previous year until corresponding day
 - Cumulative energy consumption during the same previous year until corresponding day
 - Cumulative energy consumption for the same day (24h) of previous year
 - Cumulative energy consumption for the whole week (7 full days) of previous year
 - Cumulative energy consumption for the full month of previous year
 - Cumulative energy consumption during the entire previous year
- Profiling (classification) of prosumer portfolio according to characteristics of the demand (time distribution, amount of demand, associated economic cost, associated environmental impact)
- Breakdown of portfolio demand/production according to contract type (domestic, tertiary)

These jobs appear in the architecture of the WiseCOOP as *KPI engine* module. Results of this calculation will be stored back to different collections of the long-term database, but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data), or republished to the internal ESB with the objective of making certain KPIs accessible to other modules of the application.

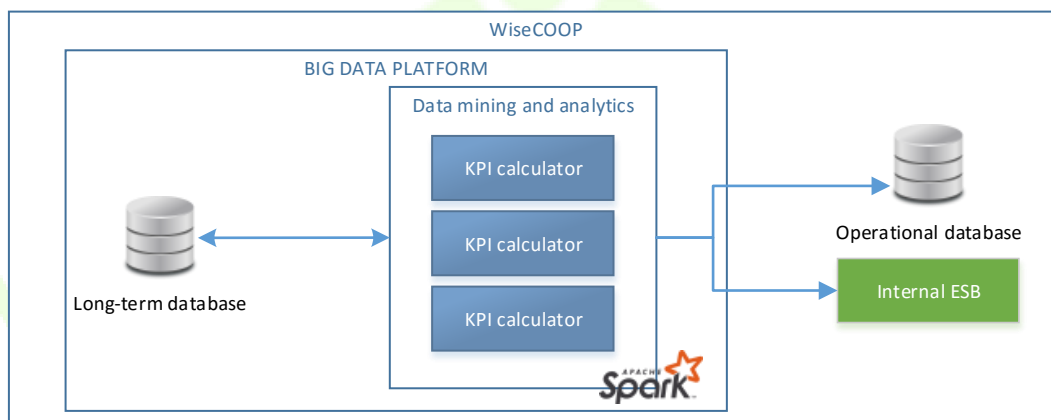
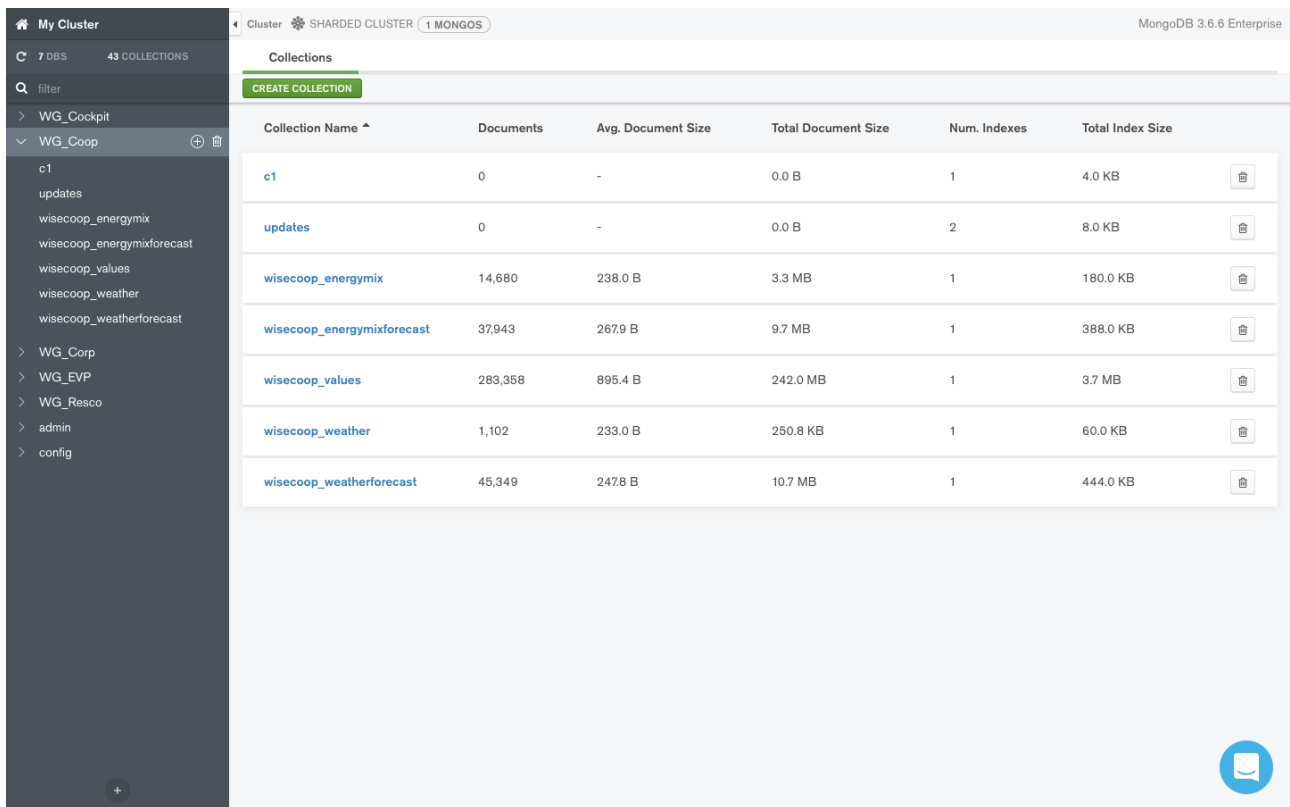


Figure 44 – WiseCOOP Big Data mining

As part of the implementation and lab-testing task, the real-time monitor module of the WiseCOOP application has been configured to store all data produced within the lab environment in the MongoDB cluster developed in this work package. This implied the creation of one database hosting the collections specified below.



The screenshot shows the MongoDB interface for a 'SHARDED CLUSTER' with '1 MONGOS'. The left sidebar lists the database 'WG_Coop' and its collections. The main panel displays a table of collections with the following data:

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
c1	0	-	0.0 B	1	4.0 KB
updates	0	-	0.0 B	2	8.0 KB
wisecoop_energymix	14,680	238.0 B	3.3 MB	1	180.0 KB
wisecoop_energymixforecast	37,943	267.9 B	9.7 MB	1	388.0 KB
wisecoop_values	283,358	895.4 B	242.0 MB	1	3.7 MB
wisecoop_weather	1,102	233.0 B	250.8 KB	1	60.0 KB
wisecoop_weatherforecast	45,349	247.8 B	10.7 MB	1	444.0 KB

Figure 45 – Details of WiseCOOP Database

Table 10 – WiseCOOP Database collections

Collection	Description
Wisecoop_values	History of readings reported by the smart meters monitored used by WiseCOOP
Wisecoop_energymix	Actual energy mix history at the location of the corresponding pilot site (1 register every hour)
Wisecoop_energymixforecast	History of energy mix forecasts at the location of the corresponding pilot site (46 registers every hour)
Wisecoop_weather	Actual weather history at the location of the corresponding pilot site (1 register every hour)
Wisecoop_weatherforecast	History of weather forecasts at the location of the corresponding pilot site (12 registers every hour)

In order to take advantage of the capabilities of the clustered database, the collections need to be *sharded*, so data is actually distributed among the different nodes of the cluster. The advantages of this configuration are described in detail in D5.1 [1]. The most important point when applying this configuration is defining the proper *shard* key for the collection, which is used to automatically create partitions among data that are stored at different nodes. As explained in MongoDB official documentation, *the shard key determines the distribution of the collection's documents among the cluster's shards. The shard key is either*

an indexed field or indexed compound fields that exists in every document in the collection [26].

In the case of WiseGRID Cockpit, tests have been carried out with the *wisecoop_values* collection, the one holding the history of reading of the smart meters. The selected shard key was *{id:1, timestamp:1}*. By including the ID of the devices, it is assured that measures of different devices are candidates to be separated in different clusters. Since the collection contains historical readout data, timestamps were also chosen to be included in the shard key in order to promote the separation of the data accordingly to the periods in time they refer to. Results of this configuration are shown below:

```
db.getCollection('wisecoop_values').getShardDistribution()

Shard rs1 at rs1/sh1r0:27017,sh1r1:27017
  data : 240.22MiB docs : 282602 chunks : 2
  estimated data per chunk : 120.11MiB
  estimated docs per chunk : 141301

Shard rs2 at rs2/sh2r0:27017,sh2r1:27017
  data : 1.73MiB docs : 756 chunks : 1
  estimated data per chunk : 1.73MiB
  estimated docs per chunk : 756

Totals
  data : 241.95MiB docs : 283358 chunks : 3
  Shard rs1 contains 99.28% data, 99.73% docs in cluster, avg obj size on
shard : 891B
  Shard rs2 contains 0.71% data, 0.26% docs in cluster, avg obj size on
shard : 2KiB
```

In this case, it can be observed that the volume of data in the lab-testing environment at the time of writing has just reached the minimum to activate the redistribution of the data among the shards of the cluster. In this case, the logs show how 2 of the shards are used (rs1 and rs2), even if the distribution of data among them is not homogeneous. The logs still show how shards are replicated (sh1r0/sh1r1, sh2r0/sh2r1), thus increasing the availability of the data in the cluster in case of failure.

9.1.6 Big data interactions from WiseHOME application

9.1.6.1 Short application description

WiseHome is an application used by the home users that are in the WiseCOOP framework structure in order to get information of the status of their participation in WiseCOOP. WiseHOME interacts with other application such as WiseCOOP and WG Staas/VPP through WG IOP sending data requests and generation simple and comprehensive graphic interface for the user accessible by a large type of devices as computers and mobile devices.

9.1.6.2 Interface with Big Data platform

WG Home does not interact directly to the Big Data platform and is not storing any data in the long term database from the Big Data Platform. WiseHome is using a small local SQL database in order to process the user authentication data and data got from WG COOP and WG Staas/VPP. The interaction

diagram of WiseHOME to the other WiseGRID application is defined in the bellow diagram:

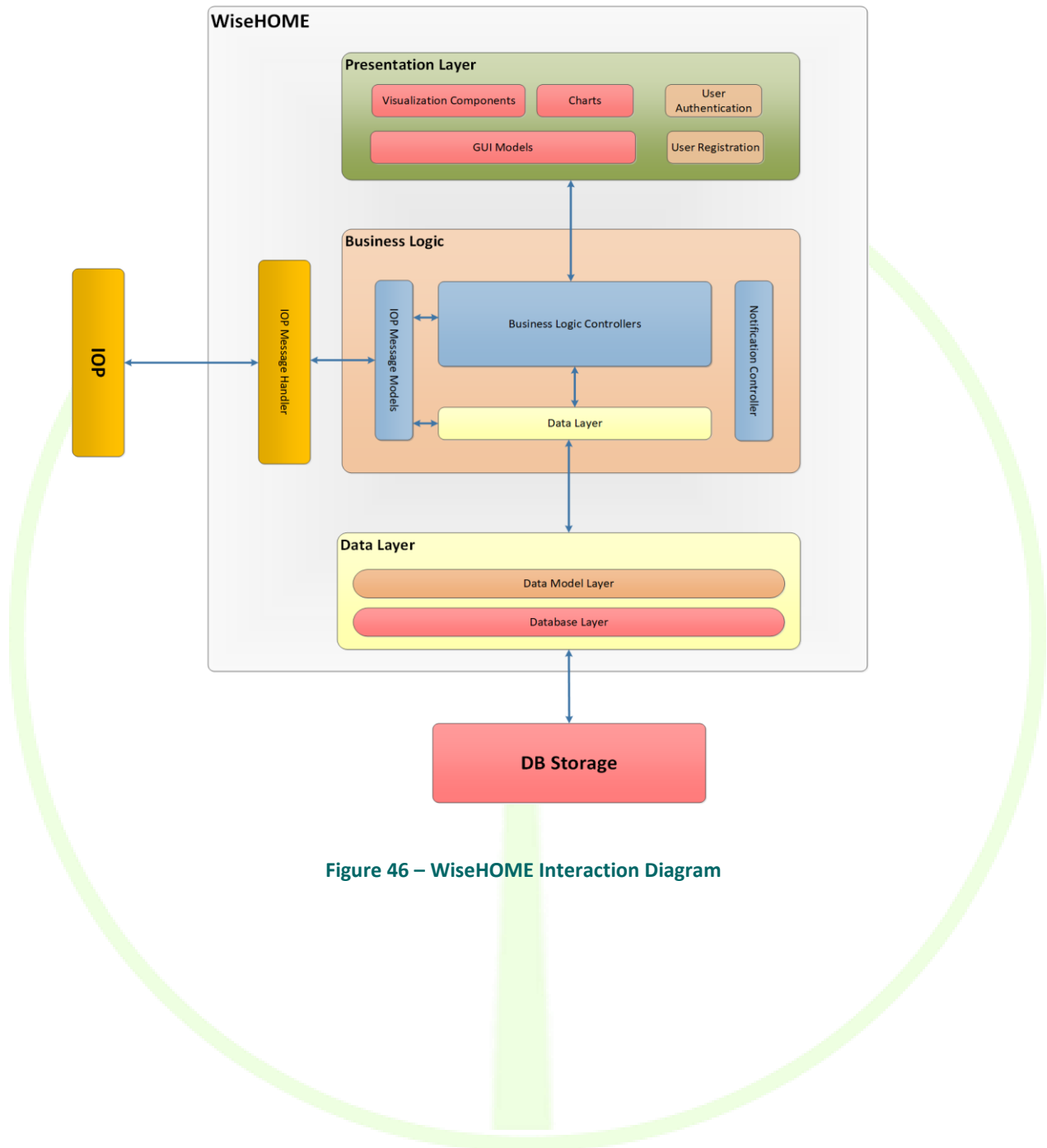


Figure 46 – WiseHOME Interaction Diagram

9.1.7 Big data interactions from WiseEVP application

9.1.7.1 Short application description

WiseEVP is the WiseGRID technological solution for

- Vehicle-sharing companies or electric vehicle fleet managers and
- Electric vehicle infrastructure (EVSE) operators

In order to optimize the activities related with smart charging and discharging of the EVs including V2G (vehicle to grid, energy injection in the distribution network) and V2B (vehicle to building).

The management of the EVSEs charging and discharging processes will meet the following objectives:

- Reduce the EV charging energy bill.
- Follow flexibility requests from DSO to help the electric distribution network operation in exchange for an economic compensation.
- Follow flexibility requests to increase injection of RES production reducing curtailment in exchange for an economic consideration compensation.
- Contribute in the building energy management system with the main objectives of reducing the energy bill and maximize local RES production.

All the aforementioned objectives will be subordinated to the EV user preferences: desired state of charge (SOC) at the time of unplugging the EV.



Figure 47 – WiseEVP

9.1.7.2 Interface with Big Data platform

The big data platform will be used by the WiseEVP for storing the history of energy supplied or injected by the different EVSEs, and hold the history of the parameters read out from the vehicles of the fleet, which will be used for analyzing the usage of those and optimize the charging schemes. As depicted in the architecture of the WiseEVP, the main data sources for this information are:

- Electric Vehicle Supply Equipment (EVSEs), providing readings of energy supplied (or injected under V2G schemes) and the identification of the vehicle associated to these energy flows.
- Electric Vehicles (EVs), which will be monitored to regularly extract information to model their

usage (State of Charge, travelled distances, location, energy charged).

This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker into the WiseEVP application. Similarly to other applications, the Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

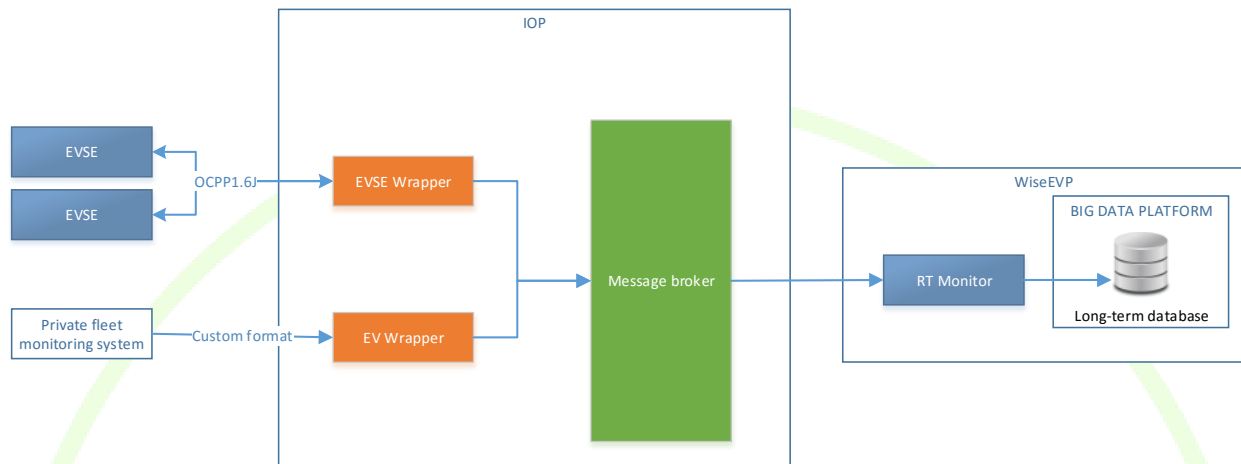


Figure 48 – WiseEVP interface with Big Data platform

The RT monitor module is, therefore, the module which, inside the WiseEVP application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseEVP will use the *Data mining and analytics* module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, and will process the following required information:

- Analysis of energy supplied by EVSEs accordingly to their source
- Hourly/Daily/Monthly aggregations of energy demanded by the fleet
- Hourly/Daily/Monthly aggregations of energy supplied or injected back to the grid by the EVSEs
- Estimation of costs associated to energy demand (both economic and environmental impact)
- Comparison of costs with previous periods
- Analysis of battery health of the EVs of the fleet (trends in energy/distance component)

These jobs appear in the architecture of the WiseEVP as *KPI engine* module. Results of this calculation will be stored back to different collections of the long-term database, but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data), or republished to the internal ESB with the objective of making certain KPIs accessible to other modules of the application.

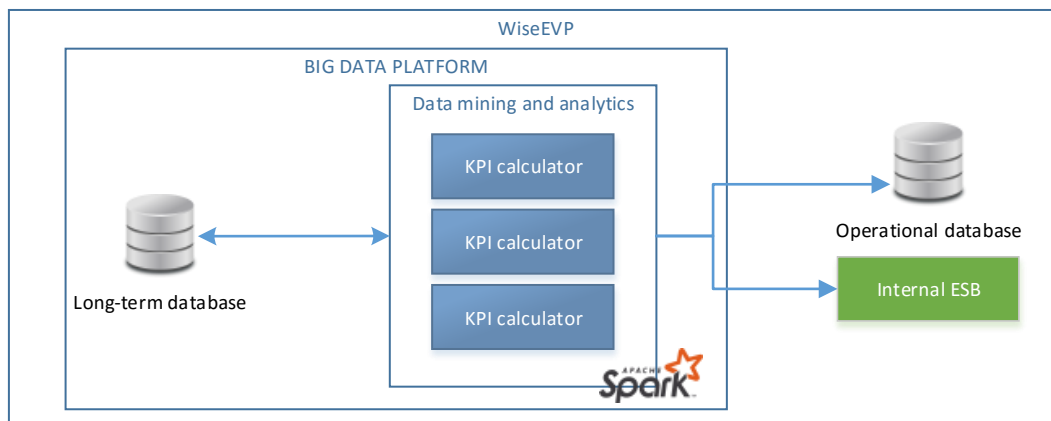


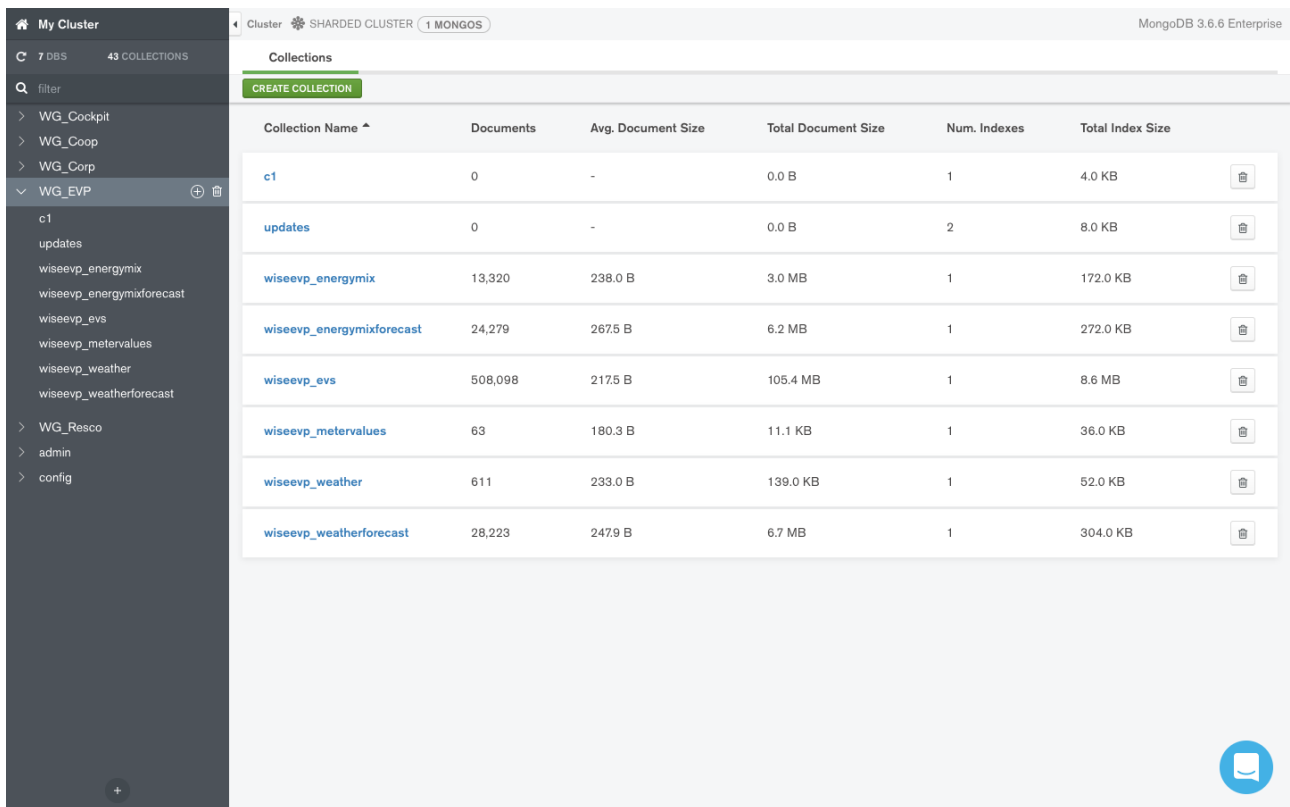
Figure 49 – WiseEVP Big Data mining

As part of the implementation and lab-testing task, the real-time monitor module of the WiseEVP application has been configured to store all data produced within the lab environment in the MongoDB cluster developed in this work package. This implied the creation of one database hosting the collections specified below.

Table 11 – Collections in Wise EVP database

Collection	Description
Wiseevp_evs	History of status reported by electric vehicles (about 1 value every 10 seconds, depending on pilot site-specific implementation)
Wiseevp_metervalues	History of energy readouts notified by charging stations (1 value every 15 minutes during charging sessions)
Wiseevp_energymix	Actual energy mix history at the location of the corresponding pilot site (1 register every hour)
Wiseevp_energymixforecast	History of energy mix forecasts at the location of the corresponding pilot site (46 registers every hour)
Wiseevp_weather	Actual weather history at the location of the corresponding pilot site (1 register every hour)
Wiseevp_weatherforecast	History of weather forecasts at the location of the corresponding pilot site (12 registers every hour)

Details of the database used during the lab testing phase by the WiseEVP:



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
c1	0	-	0.0 B	1	4.0 KB
updates	0	-	0.0 B	2	8.0 KB
wiseevp_energymix	13,320	238.0 B	3.0 MB	1	172.0 KB
wiseevp_energymixforecast	24,279	267.5 B	6.2 MB	1	272.0 KB
wiseevp_ems	508,098	217.5 B	105.4 MB	1	8.6 MB
wiseevp_metervalues	63	180.3 B	11.1 KB	1	36.0 KB
wiseevp_weather	611	233.0 B	139.0 KB	1	52.0 KB
wiseevp_weatherforecast	28,223	247.9 B	6.7 MB	1	304.0 KB

Figure 50 – WiseEVP database

In order to take advantage of the capabilities of the clustered database, the collections need to be *sharded*, so data is actually distributed among the different nodes of the cluster. The advantages of this configuration are described in detail in D5.1 [1]. The most important point when applying this configuration is defining the proper *shard key* for the collection, which is used to automatically create partitions among data that are stored at different nodes. As explained in MongoDB official documentation, *the shard key determines the distribution of the collection's documents among the cluster's shards. The shard key is either an indexed field or indexed compound fields that exists in every document in the collection* [26].

In the case of WiseGRID Cockpit, tests have been carried out with the *wiseevp_ems* and *wiseevp_metervalues* collections, the ones holding the history of status and setpoints of the assets in the building. The selected shard key for both is similar *{id:1, timestamp:1}*. By including the ID of the devices, it is assured that measures of different devices are candidates to be separated in different clusters. Since the collection contains historical readout data, timestamps were also chosen to be included in the shard key in order to promote the separation of the data accordingly to the periods in time they refer to. Results of this configuration are shown below:

```
db.getCollection('wiseevp_ems').getShardDistribution()

Shard rs0 at rs0/sh0r0:27017,sh0r1:27017
  data : 32.14MiB docs : 154628 chunks : 1
  estimated data per chunk : 32.14MiB
  estimated docs per chunk : 154628

Shard rs1 at rs1/sh1r0:27017,sh1r1:27017
  data : 32.14MiB docs : 154629 chunks : 1
  estimated data per chunk : 32.14MiB
```



```
estimated docs per chunk : 154629
```

```
Shard rs2 at rs2/sh2r0:27017,sh2r1:27017
data : 41.11MiB docs : 198872 chunks : 2
estimated data per chunk : 20.55MiB
estimated docs per chunk : 99436
```

Totals

```
data : 105.41MiB docs : 508129 chunks : 4
Shard rs0 contains 30.49% data, 30.43% docs in cluster, avg obj size on
shard : 217B
Shard rs1 contains 30.49% data, 30.43% docs in cluster, avg obj size on
shard : 217B
Shard rs2 contains 39% data, 39.13% docs in cluster, avg obj size on
shard : 216B
```

It can be observed that the *shard* key works as expected, distributing data approximately equally among the three nodes. In addition, the logs show how the shards are replicated (sh0r0/sh0r1, sh1r0/sh1r1, sh2r0/sh2r1), thus increasing the availability of the data in the cluster in case of failure.

9.1.8 Big data interactions from WG FastV2G application

9.1.8.1 Interface with Big Data platform

FastV2G is the Charging Station for Electric Vehicles developed within the WiseGRID project with the capability of transferring energy stored in EVs battery to grid. This asset will be managed by the WiseEVP application, and will not make use of the big data platform.

9.1.9 Big data interactions from WG StaaS/VPP application

9.1.9.1 Short application description

WG StaaS/VPP (“Storage as a Service / Virtual Power Plant”) is WiseGrid’s solution to clustering distributed storage systems and making them usable for the grid in general and for DSOs in particular. This will be achieved by:

- enabling forecasting of the flexibility of the connected distributed batteries,
- enabling collective contribution to various markets (wholesale, ancillary services) with this flexibility, and
- enabling collective scheduling of distributed storage systems in order to fulfil market requests in an optimized way.

This is done by separating WG StaaS/VPP into several sub-components as sketched in Figure 51 – Sketch of data flow between WG StaaS/VPP components and neighbouring WG tools.

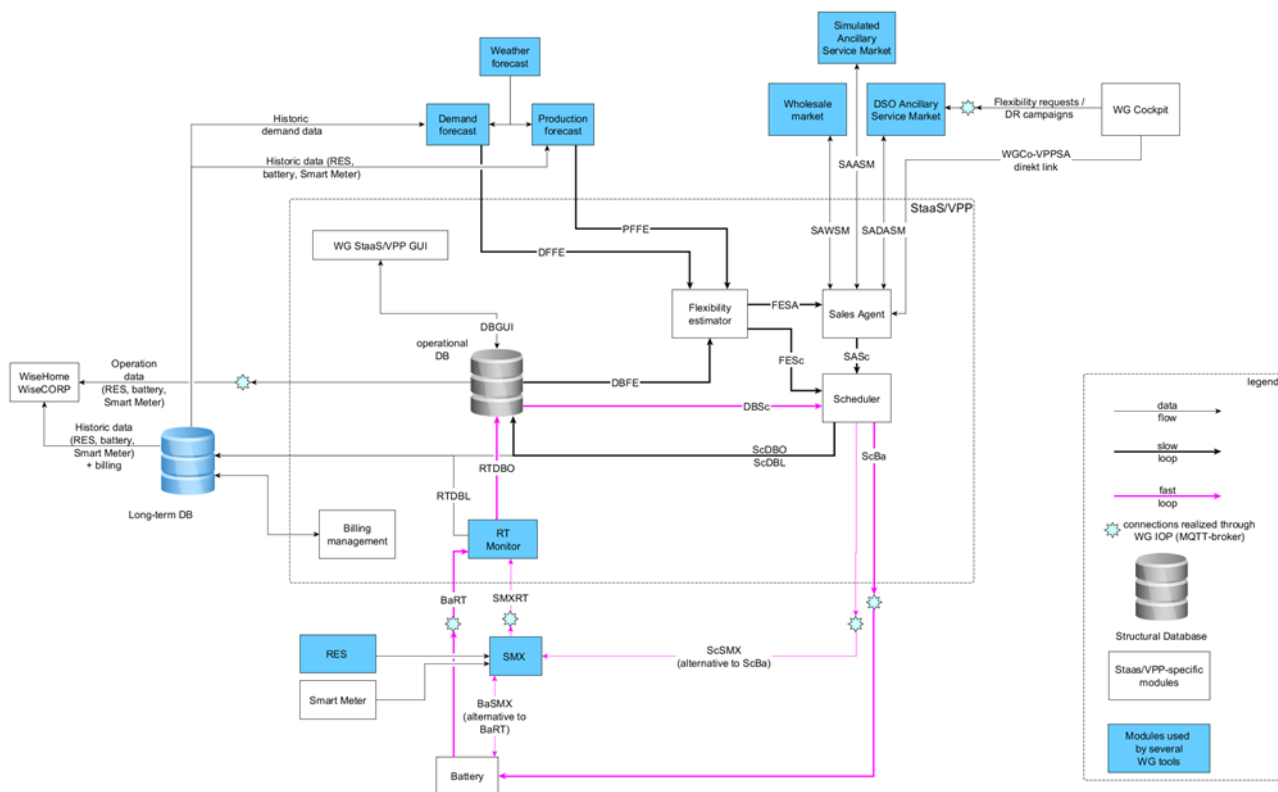


Figure 51 – Sketch of data flow between WG StaaS/VPP components and neighbouring WG tools

There are two main loops of data flow within WG StaaS/VPP:

Slow loop (executed every 15 minutes):

1. Flexibility Estimator determines remaining flexibility within the next 24 hours.
2. Sales Agent tries to sell this flexibility to the markets.
3. If some flexibility could be sold to the markets: Scheduler schedules fulfilling of required services by the batteries and communicates this schedule back to the Flexibility Estimator.
4. back to step 1.

Fast loop (executed every couple of seconds):

1. If services need to be served to the grid right now: Scheduler determines if, e.g., required active power is currently fulfilled. If not: Scheduler re-calculates power to be delivered by each individual battery.
2. Batteries try to follow Scheduler's requests.
3. back to step 1.

9.1.9.2 Interface with Big Data platform

WG StaaS/VPP uses two databases fulfilling different purposes:

- Operational DB: WG-internal DB that holds short-term data, contractual data and data intended to be passed from one sub-component, e.g., the "Scheduler" to another one, e.g., the "Flexibility Estimator".

- Long-term DB: Big data platform as described in this document to store data required for longer periods of time.

The following data are intended to be stored in the long-term DB:

- Monitoring data from the batteries as specified in **¡Error! No se encuentra el origen de la referencia.**
- Battery schedules as determined by the Scheduler sub-component: required metering point active power, required battery system reactive power, required battery system frequency control set-points, ...

Table 12 – List of WG StaaS/VPP monitoring data

Type	Variable	Explanation	Unit	Range
Double	Meter_Active_Power	at the grid connection point of the household as measured by the battery controller	W	[-Pmax, Pmax]
Double	Meter_Reactive_Power	at the grid connection point of the household as measured by the battery controller	VAR	[-Qmax, Qmax]
Double	Inverter_Active_Power	(AC) active power of the battery inverter	W	[-Pmax, Pmax]
Double	Inverter_Reactive_Power	of the battery inverter	VAR	[-Qmax, Qmax]
Double	Inverter_PV_Power	of the PV inverter included in a battery system, if it is such a “hybrid” system	W	[-Pmax, Pmax]
Double	External_PV	of a stand-alone PV inverter, which is somehow measured by or in communication with the battery controller	W	[-Pmax, Pmax]
Double	Inverter_Battery_Power	DC power of the battery inverter	W	[-Pmax, Pmax]
Double	Battery_SOC	usable energy presently in the battery divided by actually usable energy capacity	%	[0, 100]
Double	Battery_SOH	actually usable energy capacity divided by rated capacity	%	[0, 100]
Double	Battery_Voltage		V	[0, 500]
Double	Meter_Grid_Voltage	at the grid connection point of the household as measured by the battery controller	V	[0, 500]
Double	Meter_Grid_Frequency	at the grid connection point of the household as measured by the battery controller	Hz	[0, 500]
Array	Temperatures: Batt_Cell_Max_T, Batt_Cell_Min_T, Inverter_T	maximum temperature of the battery cells, minimum temperature of the battery cells, inverter IC temperature	°C	[-500, 500]
Double	Charge_Available	actual maximum available active charge power (AC) of the battery	W	[0, Pmax]

Double	Discharge_Available	actual maximum available active discharge power (AC) of the battery	W	[0, Pmax]
Int	Status	0: disconnected, 1: connected, 2: charge, 3: discharge, 4: standby, 5: error, 6: busy, 7: islanding	---	---
Array	Alarms	Error numbers	---	---
Double	Inverter_PV_Voltage		V	[0, 1000]
Int	Working mode	as defined in sheet 'Operation'	---	---
Double	Demand	power consumed at the house	W	[-Pmax, Pmax]

Based on these datasets, the “Billing management” sub-component ensures proper remuneration of contribution battery systems by retrieving back historical data from the long-term DB, e.g. active powers actually delivered by individual batteries and other individual contributions to the overall VPP-services. Additionally, the forecast modules can generate production and demand forecasts based on these historical data, which are required to estimate the battery system flexibilities within the Flexibility Estimator module.

9.1.10 Big data interactions from WG RESCO application

9.1.10.1 Short application description

The WG RESCO is a tool conceived for RESCOs - Renewable Energy Service Companies and ESCOs that want to provide RES services to end-users (households or businesses) that do not own nor wish to maintain the necessary equipment. In this perspective, three potential scenarios are envisaged:

- RESCO pays a fee to end-users using their premises (e.g. for installing PVs on their roof), installs and maintains the RES assets and markets all produced energy;
- RESCO provides to customers the supply of energy coming from RES owned by the RESCO (i.e. allowing self-consumption) and markets the production surplus;
- RESCO provides to customers the installation of RES equipment (e.g. PV panels) which are owned and maintained by the RESCO but fully exploited by the end customers (renting business model).

According to that, the WG RESCO tool will support RESCOs in managing the relationship with their customers and the provision of energy to the consumers from renewable energy sources, usually PV, wind power or micro hydro. Since the generation equipment will be owned, serviced and operated by the RESCO itself, the WG RESCO will have a central feature in supporting the maintenance management of those assets.

WiseGRID RESCO Tool was designed and developed based on the previously described perspective and understanding for the business model of a RESCO. Fundamentally, this tool needs to provide all the necessary functionalities that support and facilitate the activities of this business actor. The main functionalities can be summarized as follow:

1. Performs all the necessary functionalities required for the business operations like:
 - Manage contracts with customers
 - Manage portfolio of installed assets
 - Manage assets maintenance
 - Manage energy flows
 - Manage economic flows
 - Manage energy metering and forecasting

- Manage the Market bid
 - Provide data for future Investment Decision
 -
2. Supports the Interoperability with external tools
 3. Incorporates Functional User Interface

9.1.10.2 Interface with Big Data platform

The WG RESCO tool has been developed using the MEAN.JS full-stack JavaScript solution, this choice has been made to build fast, robust, and maintainable production web applications using MongoDB which is the one adopted for the Big Data platform, AngularJS and adopting Node.js as engine.

Currently the RESCO tool store its data in two different data base provided by the Big Data Platform:

- “Local DB” to store Customer data, Contract data, Asset data, Configuration data and real time meter observation coming from RES source and stored by the RT Monitor tool.
- “Long term DB” to store historical data coming from RES source via the RT Monitor module. Similarly, to other applications, the Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform. RESCO tool can also have direct access to the long-term DB to store data needed for future analysis (like billing data, wholesale market offers) and use specific DB interaction services to divide the computational load.

WG Resco tool access directly to Big Data platform via java script. Mongoose ,that is an Object Data Modeling (ODM) library for MongoDB and Node.js is adopted to helps the developer to manages relationships between data, provides schema validation, and it is also used to translate between objects in code and the representation of those objects in MongoDB.

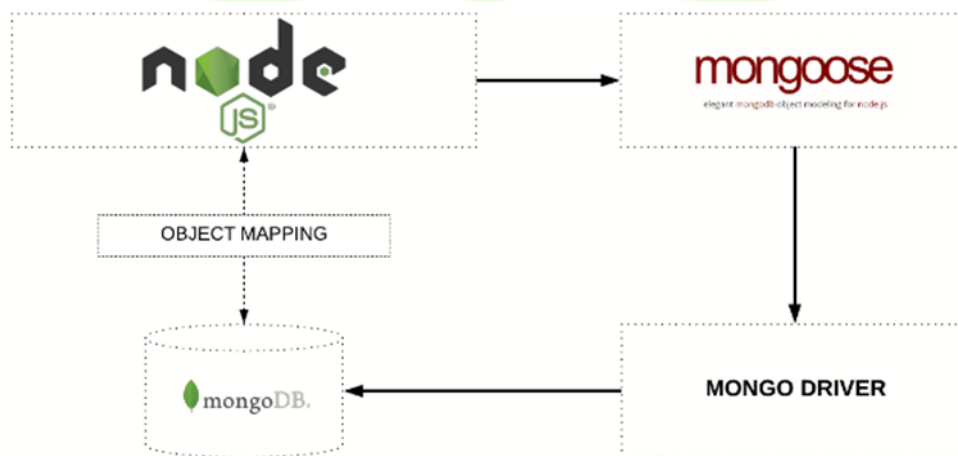


Figure 52 – Object Mapping between Node and MongoDB managed via Mongoose

The next image shows a high-level architecture about the interface between the WG RESCO tool and the Big Data Platform. As shown, the WG RESCO tool can read and write data directly by the Local DB and Long-term DB. It can also send requests to the WG Services and Tools in order to receive data stored into the Big Data Platform that are mainly provided by these services/tools. To do that, the RESCO tool send a specific request queue via the WG IOP, the WG services and tools read the request, perform an query into

the Long-Term DB and send the response queue with the data retrieved via WG IOP to the WG RESCO tool that read the response and consume it.

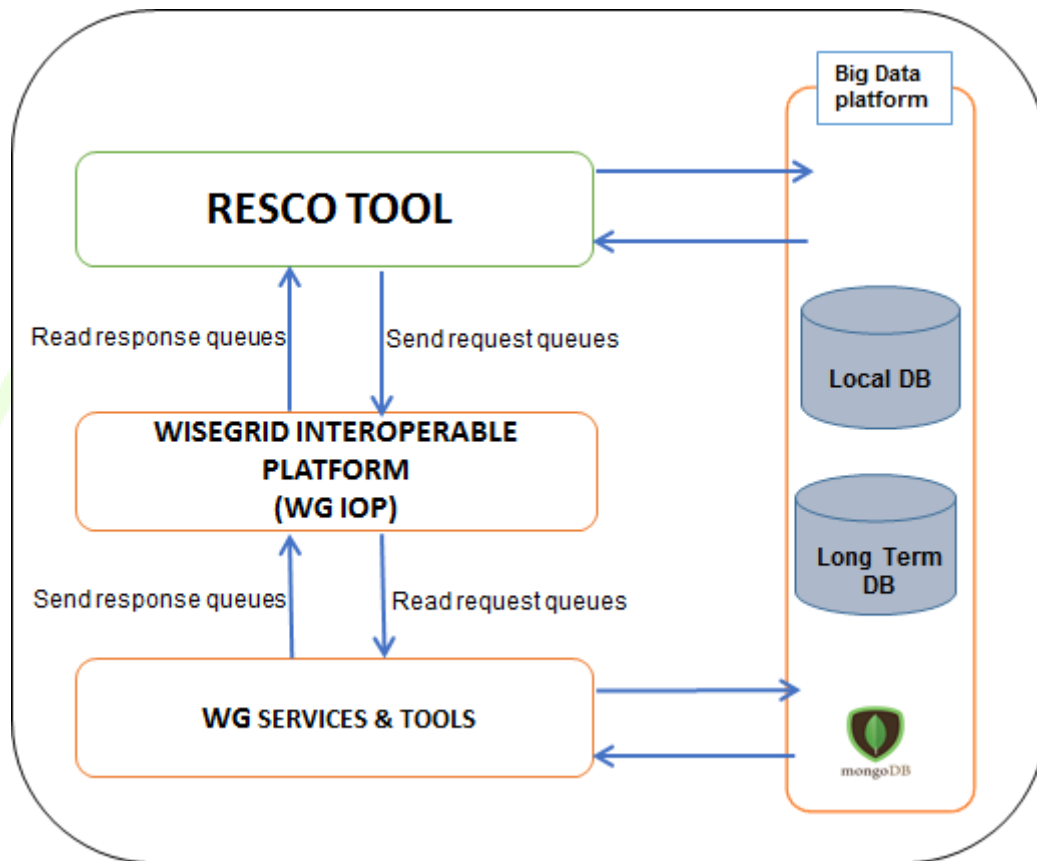


Figure 53 – WG RESCO Tool interface with Big Data Platform

10 CONCLUSIONS AND NEXT STEPS

10.1 CONCLUSIONS

This deliverable defined an infrastructure for implementing the Big Data services both online and offline in a lab testing demonstrator. For providing the Big Data services needed in the WiseGRID project are needed two horizontally scalable computer clusters.

- The first cluster is providing online Big Data services as long term data storage and retrieving in separate databases for each application.
 - A limited access database for each application containing data that are not under the restrictions of GDPR. The access will be granted by a user/password pair contained in each data base.

- Each application will access the database through a dedicated application router installed on a separate cluster node.

The first cluster is running MongoDB NoSQL database management system on a computer cluster of eleven nodes.

- The second cluster is hardware separated from the database cluster is providing the offline services. This cluster is based by Apache Hadoop framework and will provide the remote processing of applications based on Apache Spark operators.

The second cluster is connected to the MongoDB database cluster through the Hadoop MongoDB connector. Each application will have a separate account for each WiseGRID application. The cluster is minimally composed of five nodes.

Table 13 – Deliverable objectives

Objective	Achieved within WP
Defining the hardware infrastructure of a Big Data computer cluster demonstrator providing cloud services to WiseGRID applications for online services like long term data storage and retrieving	WP5.1 WP5.2
Defining the hardware infrastructure of a Big Data computer cluster demonstrator providing cloud services to WiseGRID applications for offline services like data mining and analytics.	WP5.1 WP 5.2
Providing a clear and extensive manual for installation and deploying of the MongoDB cluster to be used in Big Data platform for the pilot sites	WP5.1 WP5.2
Providing a clear and extensive manual for installation and deploying of the Hadoop Spark cluster to be used in Big Data platform for the pilot sites	WP5.1 WP5.2

10.2 NEXT STEPS TO BE IMPLEMENTED

The next step is to test the small-scale demonstrator of both computer cluster, MongoDB computer cluster and Apache Hadoop computer cluster for the interaction with the WiseGRID applications. The testing of the demonstrator will show the extent of the needs of Pilot sites in order to scale the Big Data platform for each pilot site to the applications requirements. This small scale demonstrator will be duplicated for each Demo Site providing the services for demo applications in each site. Based on the data collected from each demo site the computer clusters will be defined from the point of view of required resources.

11 REFERENCES AND ACRONYMS

11.1 REFERENCES

- [1] Lacatus, Paul (CRE), “D5.1 WiseGRID cloud-based big data infrastructure 1.0,” 2018.
- [2] B. Marr, “Big Data the 5 Vs everyone must know,” [Online]. Available: <https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know/>.
- [3] B. Hedlund, “Understanding Hadoop Clusters and the Network,” [Online]. Available: <http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/>.
- [4] Wikipedia, “Wikipedia Single_board computers,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Single-board_computer.
- [5] Wikipedia, “Raspberry Pi,” [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi.
- [6] Wikipedia, “ARM architecture,” [Online]. Available: https://en.wikipedia.org/wiki/ARM_architecture.
- [7] Raspberry Pi Foundation, “Putting the power of digital making into the hands of people all over the world. Strategy 2016-2018,” [Online]. Available: <https://static.raspberrypi.org/files/about/RaspberryPiFoundationStrategy2016-18.pdf>.
- [8] Raspberry pi Foundations, “Any plans for 64-bit Raspbian,” [Online]. Available: <https://www.raspberrypi.org/blog/eben-q-a-1/>.
- [9] Hardkernel, “Odroid XU4,” [Online]. Available: <https://wiki.odroid.com/odroid-xu4/odroid-xu4>.
- [10] Hardkernel, “Odroid C2,” [Online]. Available: <https://wiki.odroid.com/odroid-c2/odroid-c2>.
- [11] MongoDB, “MongoDB Manual,” [Online]. Available: <https://docs.mongodb.com/manual/>.
- [12] i. o. Thingiverse, “DIN Mounts : Pi, Arduino and disks,” [Online]. Available: <https://www.thingiverse.com/thing:2610621>.
- [13] ". o. Thingiverse, “Support rail DIN Wago 221,” [Online]. Available: <https://www.thingiverse.com/thing:2764407>.
- [14] Hardkernel, “Odroid C2 Manual,” [Online]. Available: <https://magazine.odroid.com/wp-content/uploads/odroid-c2-user-manual.pdf>.
- [15] DigitalOcean, “How To Install Webmin on Ubuntu 16.04,” [Online]. Available: <https://www.digitalocean.com/community/tutorials/how-to-install-webmin-on-ubuntu-16-04>.
- [16] A. Felong, “UPDATE: MongoDB 3.6 on ODROID C2 with Ubuntu 16.04.3 – ARM64,” [Online]. Available: <https://andyfelong.com/2018/02/update-mongodb-3-6-on-odroid-c2-with-ubuntu-16-04-3-arm64/>.
- [17] CodingMiles.com, “Step by step MongoDB sharded cluster deployment,” [Online]. Available: <http://codingmiles.com/mongodb-sharded-cluster-deployment/>.
- [18] www.guru99.com, “MongoDB Sharded Cluster - Step by Step Implementation,” [Online]. Available: <https://www.guru99.com/mongodb-sharding-implementation.html>.
- [19] www.hatcher.com, “How to deploy a MongoDB cluster (version 3.4),” [Online]. Available:

<https://www.hachther.com/en/blog/deploy-mongodb-cluster-version-3-4/>.

- [20] www.alibabacloud.com, “High-availability MongoDB Cluster Configuration Solutions,” [Online]. Available: https://www.alibabacloud.com/blog/high-availability-mongodb-cluster-configuration-solutions_490866.
- [21] Hardkernel, “Odroid XU4 manual,” [Online]. Available: <https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf>.
- [22] DIY Big Data, “Installing Hadoop onto an ODROID XU4 Cluster,” [Online]. Available: <http://diybigdata.net/2016/06/installing-hadoop-onto-an-odroid-xu4-cluster/>.
- [23] MongoDB manual, “MongoDB Connector for Hadoop,” [Online]. Available: <https://docs.mongodb.com/ecosystem/tools/hadoop/>.
- [24] Mongo Hadoop Connector Wiki, “Mongo Hadoop Connector Wiki,” [Online]. Available: <https://github.com/mongodb/mongo-hadoop/wiki>.
- [25] MongoDB manual, “Hadoop and MongoDB Use Cases,” [Online]. Available: <https://docs.mongodb.com/ecosystem/use-cases/hadoop/>.
- [26] “MongoDB Shard Keys,” [Online]. Available: <https://docs.mongodb.com/manual/core/sharding-shard-key/>.
- [27] MongoDB, “MongoDB forGiant Ideas,” [Online]. Available: <https://www.mongodb.com/>.
- [28] Wikipedia, “Wikipedia BigData Definitions,” [Online]. Available: https://en.wikipedia.org/wiki/Big_data.
- [29] Webmin site, “Webmin,” [Online]. Available: <http://www.webmin.com/index.html>.
- [30] MongoDB Compass, “MongoDB Compass,” [Online]. Available: <https://www.mongodb.com/products/compass>.
- [31] Apache Software Foundation, “Apache Software foundation,” Apache, [Online]. Available: <https://www.apache.org/>.
- [32] Apache, “Apache Hadoop,” [Online]. Available: <http://hadoop.apache.org/>.
- [33] MongoDB, “Wiki of MongoDB Hadoop Connector,” [Online]. Available: <https://github.com/mongodb/mongo-hadoop/wiki>.
- [34] Apache Spark, “Apache Spark™,” [Online]. Available: <https://spark.apache.org/>.
- [35] Apache Hadoop, “Apache Hadoop Docs, Cluster setup,” [Online]. Available: <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html>.
- [36] Tutorialspoint, “MongoDB Quick Guide,” [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm.
- [37] [Online]. Available: <https://www.mongodb.com/products/compass>.

11.2 ACRONYMS

Table 14 – List of Acronyms

Acronyms List	
BSON	Binary JavaScript Object Notation
CHP	Combined Heat and Power
DB	Data base
DSO	Distribution service operator
ESB	Enterprise Service Bus
ESCO	Energy service company
EV	Electrical vehicle
EVSE	Electrical vehicle service equipment
HV	High Voltage
HVAC	Heat, Ventilation and Air Conditioning
IOP	Interoperability platform
JSON	JavaScript Object Notation
KPI	Key performance indicators
RDBMS	Relational Data Base Management system
RESCO	Renewable energy service company
RT	Real Time
SCADA	Supervisory Control And Data Acquisition
SoC	State of Charge
StaaS	Storage as a service
TM	Technological Manager
VPP	Virtual power plant
SBC	Single board computers
OS	Operating system
SMPS	Switched mode power supply
CLI	Command line interface